(* begin module version *)
version = 5.05 of delman1   1993 January 27

```
ddddddd   eeeeeeee  ll        m      m     aa     n      nn
dd    dd  ee        ll        mm    mm    aaaa    nn     nn
dd    dd  ee        ll        mmm  mmm   aa  aa   nnn    nn
dd    dd  eeeeeee   ll        mmmmmmmm  aa    aa  nnnn   nn
dd    dd  ee        ll        mm mm mm  aa    aa  nn nn  nn
dd    dd  ee        ll        mm    mm  aaaaaaaa  nn  nnnn
dd    dd  ee        ll        mm    mm  aa    aa  nn   nnn
dd    dd  ee        ll        mm    mm  aa    aa  nn     nn
ddddddd   eeeeeeee  llllllll  mm    mm  aa    aa  nn     nn

                       11
                      111
                       11
                       11
                       11
                       11
                       11
                       11
                    11111111
```

THE DELILA SYSTEM MANUAL

THOMAS D. SCHNEIDER

1. Don't Panic! You don't have to absorb this all at once!
2. There is an index at the end of any printed copy of Delman!
3. To create Delman2, see file aa.p

(* end module version *)

```
 IIIIIIII  N      NN  TTTTTTTT  RRRRRRR    OOOOO
    II     NN     NN     TT     RR     RR  OO     OO
    II     NNN    NN     TT     RR     RR  OO     OO
    II     NNNN   NN     TT     RR     RR  OO     OO
    II     NN NN NN      TT     RR     RR  OO     OO
    II     NN  NNNN      TT     RRRRRRR    OO     OO
    II     NN   NNN      TT     RR  RR     OO     OO
    II     NN     NN     TT     RR   RR    OO     OO
 IIIIIIII  NN     NN     TT     RR     RR    OOOOO
```

```
DELILA SYSTEM MANUAL OUTLINE

INTRO: Introduction To The Delila System
     OUTLINE: Outline For The Delila Manual
     DESCRIPTION: What Is The Delila System?
     ORGANIZATION: Organization Of The Manual
     POLICY: Our Policies, A Disclaimer, Obtaining The Delila System,
             Our Address And Acknowledgements

TRANSPORT: Transportation Of The Delila System
     REQUIREMENTS: What You Will Need To Get The Delila System Running
     TAPE.FORMATS: Tape Data Formats

ASSEMBLY: Assembly Of The Delila System Programs
     INTRO: What We Mean By Assembly
     CHACHA: Changing Characters And Getting The First Program Running
     REMBLA: Removing Excess Blanks From Files
     WORCHA: The Reserved Word Problem
     MODULE: Module Libraries - What They Are And How To Use Them
     EXAMPLE: An Example Of Constructing A Delila System Program
     PROBLEMS: Problems That May Arise During Assembly

GUIDE: Hello, Computer - A Guide To The New User
     INTRO: Introduction To The Guide And Your Computer
     ADVICE: Advice And Tips To The New User
     DELILA: How To Use The Delila System On Your Computer

PROGRAM: System Independent Notes On Programming
     ESSAY: Suggestions On How To Learn And Do Programming
     FABLE: A Fairy Tale For Programmers
```

(* end module delman.intro.outline.1 *)

```
USE: Uses Of The Delila System
     INTRO: Introduction
     STRUCTURE: Library Structure: Trees, Nested And Named Objects
     LANGUAGE: Delila - The Language
     AUXILIARY.PROGRAMS:  Lister And Search
     DATA.FLOW: Data Flow And Data Loops
     COORDINATES: The Coordinate System Of A PIECE
     CONTROL: How To Control The Responses Of Delila
     COMPARISON: Ways To Compare Sequences
     ALIGNED.BOOKS: How To Make And Use Aligned Books
     PERCEPTRON: Use Of The Pattern Programs
     ENCODE: Use Of The Fabulous And Powerful Encode Program
     DBPULL: Using The Data Base Extraction Programs
     SEARCH: Using The Search Program

CONSTRUCTION: Constructing Your Own Libraries
     INTRO: Introduction
     STRUCTURE: More  On Library Structure - Logical Vs Physical Structure
     CATAL: Making New Libraries - The Catalogue Program
     EXAMPLE: An Example Of Constructing Delila Libraries
     DATA.ENTRY: Using Your Own Data
     LIBRARY.DESIGN: Making A Delila Data Base
     [FORM...]: The Forms For Library Module Entry

DESCRIBE: Program And Data Descriptions
     CONVENTIONS: Notation For Naming, Writing And Running Programs
     SHORT.CLUSTER: Short Clustered Descriptions Of Delila System Files
     DOCUMENTATION: How Programs Are Documented
        The format for documentation in the Delila System is in
        file aa.p at the start of the Delman2 manual.

INDEX
     An Alphabetical Listing Of The Pages In The Manual.
     (See The Page Named DELMAN.INTRO.ORGANIZATION
      For How To Generate The Index.)
```

(* end module delman.intro.outline.2 *)

WHAT IS THE DELILA SYSTEM?

The Delila System is a collection of Pascal programs and data originally written at the University of Colorado, Boulder that allows one to manipulate and study sets of nucleic-acid sequences. A set of sequences is called a library. There is a librarian, and "her" name is Delila. One gives Delila a list of instructions that name desired fragments. Delila then searches the library, collects all the sequences together and produces a "book". The book may then be searched for patterns, listed with translation to amino acids, or studied in various ways using programs other than Delila ("auxiliary" programs). Since books may be small, these analyses can be efficient.

Books have the same form as libraries. In other words, libraries have a particular structure so that Delila can work with them. Books have that same structure. For example, given a Master DNA sequence library one can use Delila to make a subset such as a transcript library, containing sequences of mRNA. From the transcript library subsets for gene initiation regions can be made and these are guaranteed to be sequences from mRNA. During all these manipulations the numbering of the sequences remains consistent so that one can refer back to the original library or the literature. (The technical differences between libraries and books will be discussed later.)

Any auxiliary program that searches a library will know about the structure of the library. Using this structure and the search results, the program can write Delila instructions that specify the locations of the found objects. Once again, using Delila, one can loop back and create a book of these objects. Also, the instructions (instead of the sequences) can be manipulated by various programs.

A NOTE FOR PROGRAMMERS

Each auxiliary program that reads a book or library knows about the library structure. To make programming easy, a set of routines was written as an interface between the actual database (kept in a file) and the program calls and variables. These "book reading routines" are kept together in what we call a Module Library, containing many chunks of Pascal code. Each module performs certain kinds of tasks. The modules are transferred from the module library into the source code of each auxiliary program by using the Module program. In this way all changes to the interface packages can be made once in the Module Library, followed by a series of transfers. We may send the Delila System with modules removed because there is no reason to send duplicate code. After transportation you would assemble the programs.

We hope that this section gave you a rough overview of what the Delila System can do. Many more details and examples can be found in the sections that follow.

(* end module delman.intro.description *)

libdef - the definition of the Delila Library System (a file)
moddef - the definition of the Module Transfer System (a file)
doodle.info - describes Pascal graphics portable under UNIX

Some of the Delila programs and the method of moving modules around are described in these papers:

Schneider, T.D., G.D. Stormo, J.S. Haemer and L. Gold. (1982)
A design for computer nucleic-acid sequence storage, retrieval and manipulation.
Nucleic Acids Research, 10: 3013-3024.

Schneider, T.D., G.D. Stormo, M.A. Yarus, and L. Gold (1984)
Delila system tools.
Nucleic Acids Research, 12: 129-140.

Some related papers are:

Stormo, G.D., T.D. Schneider and L.M. Gold (1982)
Characterization of translational initiation sites in E. coli.
Nucleic Acids Research, 10: 2971-2996.

Stormo, G.D., T.D. Schneider, L. Gold and A. Ehrenfeucht (1982)
Use of the 'Perceptron' algorithm to distinguish translational initiation sites in E. coli.
Nucleic Acids Research, 10: 2997-3011.

Clift, B., D. Haussler, R. McConnell, T. D. Schneider and G. D. Stormo (1986)
Sequence Landscapes.
Nucleic Acids Research, 14: 141-158.

Schneider, T.D., G.D. Stormo, L. Gold and A. Ehrenfeucht (1986)
The information content of binding sites on nucleotide sequences.
J. Mol. Biol. 188: 415-431.

Stormo, G.D., T.D. Schneider and L. Gold (1986)
Quantitative analysis of the relationship between nucleotide sequence and functional activity
Nucleic Acids Research, 14: 6661-6679.

T. D. Schneider (1988)
Information and entropy of patterns in genetic switches.
In G. J. Erickson and C. R. Smith, editors, Maximum-Entropy and Bayesian Methods in Science and Engineering, volume 2, pages 147--154, Dordrecht, The Netherlands, Kluwer Academic Publishers.

T. D. Schneider and G. D. Stormo (1989)
Excess information at bacteriophage T7 genomic promoters detected by a random cloning technique.
Nucleic Acids Research, 17:659--674.

```
Reference for Dotmat, Helix, Matrix and Keymat:
    J. V. Maizel, Jr. and R. P. Lenk
    PNAS 78: 7665-7609 (1981)

A reference for Index:
    L. J. Korn, C. L. Queen and M. N. Wegman
    PNAS 74: 4401-4405 (1977)


(* end module delman.intro.references *)
```

ORGANIZATION OF THE MANUAL

The Delila Manual is broken into several somewhat independent sections. When Delman is paged by program PBREAK (see Technical notes below) you will find an index at the end. We anticipate at least two kinds of reader:

1) The builder who wants to get a Delila System running on a local computer. The section on transportation will help you get the data into your computer. The section on assembly will guide you through the difficult task of getting the programs running. At that point the Delila Libraries will still not be ready to use: you must construct catalogues as described in the section on CONSTRUCTING YOUR OWN LIBRARIES (DELMAN.CONSTRUCTION). Finally you will be able to use the Delila System. We suggest that you first look over the entire manual and associated documents. Then begin the transport. Good luck!

2) The user who wants to use a Delila System that is already running on a local computer. You may be interested in looking over the sections on transportation and assembly of the system, but this is not necessary. If you don't know anything about using the computer you should start at DELMAN.GUIDE. In any case, read the section on USE OF THE DELILA SYSTEM (DELMAN.USE).

Each program is described in a separate manual, Delman 2.

TECHNICAL NOTES (These are not be useful to people just starting.)

1. The section DELMAN.GUIDE must be rewritten after transportation to a new computer system.

2. DELMAN is physically broken into a set of modules. Each module is a page of the manual. The individual pages can be extracted (or transferred and rearranged) by using the program MODULE, as described in the document MODDEF and DESCRIBE.MODULE. The pages may be looked at on-line with the SHOW program (DESCRIBE.SHOW). The manual or extracted modules may be broken into pages for output to a lineprinter by using the PBREAK program with a parameter file containing:

```
   (* begin module
   1
```

There is no closing "*)" in the trigger because many different module names may follow the trigger, so the trigger is for the common part of the module beginnings.

You can generate another index of the contents of this manual in the List file of program Module if you use Delman as the Modlib and a copy of Delman as Sin. (See MODDEF for the definitions of these files.)

(* end module delman.intro.organization *)

OBTAINING THE DELILA SYSTEM BY FTP

The Delila system is available by anonymous ftp in the archive at ncifcrf.gov in the directory pub/delila.

OBTAINING THE DELILA SYSTEM BY TAPE

We prefer not to have to write tapes or disks, but we will send the Delila System by tape as a single package if you do not have have ftp access. Under most circumstances we cannot send parts of the system or subsets of the data. Please send us a tape as described in delman.transport.tape.formats, and we will write out the entire current version and send it back to you. There is no fee. You may redistribute the system. If you receive a a copy of the system from someone else, you may want to check back with us to see if there have been any major changes or corrections. Referring to the version number of the program or documentation will help us know if there were any changes.

DISCLAIMER

No claim or guarantee is made that Delila System programs and data are free of error. Although we send source code, we cannot guarantee that this code will compile and run on all computers. We believe that our code is reasonably efficient, but we cannot be responsible for any costs due to using the Delila System. We do not offer programming support, though we are willing to answer questions about the Delila System.

We would appreciate a detailed description of any program errors (bugs) or data errors that you encounter.

OUR ADDRESS

Tom Schneider
NCI/FCRDC Bldg 469. Room 144
P.O. Box B
Frederick, MD 21702-1201
(301) 846-5581 (-5532 for messages)
(301) 846-5598 fax
network address: toms@alum.mit.edu

National Cancer Institute
Laboratory of Mathematical Biology

ACKNOWLEDGEMENTS

Jeff Haemer, Mike Aden and Gary Stormo were instrumental in the original design of the Delila system.

Many people have helped us by reading and commenting on this manual. We would like to thank: Ginny Fonte, Larry Gold, Jeff Haemer, John Hoffhines, Jane Hessler (VA), Brent Hughes, Billie Lemmon, Melissa Mockensturm, Sandy Parkinson (UT), Pat Roche, Herb Schneider, Susan Scolman, Sidney Shinedling, Britta Singer, Rosemary Sweeney, and Mike Yarus.

Computer time and resources were generously provided by the University of Colorado at Boulder, and the Frederick Biomedical Supercomputing Center.

Funds for this project were provided through grants NIH 1 R01 GM28755, NIH 5 R01 GM19963 and ACS NP-178D.

(* end module delman.intro.policy *)

Please use this page to write comments you have about the manual
and the Delila system. Our address is on page delman.intro.policy. Thankyou.

Name: Date:

(* end module delman.intro.comments *)

```
tttttttt  rrrrrrr      aa     n     nn   ssssss
   tt     rr    rr    aaaa    nn    nn  ss    ss
   tt     rr    rr   aa  aa   nnn   nn  ss
   tt     rr    rr  aa    aa  nnnn  nn   ssssss
   tt     rr    rr  aa    aa  nn nn nn        ss
   tt     rrrrrrr   aaaaaaaa  nn  nnnn        ss  --------
   tt     rr  rr    aa    aa  nn   nnn        ss
   tt     rr   rr   aa    aa  nn    nn  ss    ss
   tt     rr    rr  aa    aa  nn    nn   ssssss


ppppppp    oooooo   rrrrrrr   tttttttt
pp    pp  oo    oo  rr    rr     tt
pp    pp  oo    oo  rr    rr     tt
pp    pp  oo    oo  rr    rr     tt
pp    pp  oo    oo  rr    rr     tt
ppppppp   oo    oo  rrrrrrr      tt
pp        oo    oo  rr  rr       tt
pp        oo    oo  rr   rr      tt
pp         oooooo   rr    rr     tt
```

(* end module delman.transport *)

TRANSPORTATION - WHAT YOU WILL NEED

If you have obtained the Delila System by computer tape, you will need some way of moving the data on the tape into your computer. We suggest that you find someone who has already dealt with tapes.

All Delila System programs are written in the language Pascal. There are many books available on this language, but the definition of the language is in:

K. Jensen and N. Wirth
Pascal User Manual and Report
Springer-Verlag, New York 1978

Some of the Delila programs have been automatically translated to C. See the README file for further details.

To run Pascal programs you will need a Pascal compiler on your computer, and enough memory to use it. It is impossible to make an accurate estimate of the memory requirements, because this depends on the computer system. However, we once set up an older version of the entire system on two computers:

```
CDC Cyber/KRONOS 5000 pru x 640 char/pru = 3,200,000 characters
DIGITAL VAX/VMS  7000 blocks x 512 char/block = 3,584,000 characters
```

Since then more programs have been added, and we find roughly:

```
4,300,000 characters of source code and files
5,300,000 bytes of compiled code on a Pyramid 90x computer running UNIX.
```

Since these estimates include object code, it is possible that the amount you require will be more or less. The estimates do not include memory required for running the system.

Since transportation of programs from one computer to another is still a tricky business, we recommend that either you learn about tapes, your computer, and Pascal, or that you find local people who know about these things and are willing to give you help.

The first Delila system file on the tape is called AAA (the name guarantees that it will be first). It lists the name of all the Delila files on the tape, in the order that they were taped. Following AAA the other files are in alphabetical order. Files are described in the manual section DELMAN.DESCRIBE.

If you keep notes on difficulties that you encounter and how each was solved, transportation of future versions of the Delila System will be easier.

(* end module delman.transport.requirements *)

```
TAPE DATA FORMATS

We send the Delila System (programs and data) out on tape.
Send us a standard 2400 foot tape.  We will send you back the tape with
the format:

   9 track
   1600 bits per inch
   Unlabeled
   Standard ASCII character set
   80 characters per record
   10 records per block

We can also send UNIX tar tapes.
The first file on the tape lists the names of all the files on the tape.
```

(* end module delman.transport.tape.formats *)

```
   AA      SSSSSS    SSSSSS   EEEEEEEE  M      M  BBBBBBB   LL        YY    YY
  AAAA    SS    SS  SS    SS  EE        MM    MM  BB    BB  LL         YY  YY
 AA  AA   SS        SS        EE        MMM  MMM  BB    BB  LL          YYYY
AA    AA   SSSSSS    SSSSSS   EEEE      MMMMMMMM  BBBBBBB   LL           YY
AA    AA        SS        SS  EE        MM MM MM  BB    BB  LL           YY
AAAAAAAA        SS        SS  EE        MM    MM  BB    BB  LL           YY
AA    AA        SS        SS  EE        MM    MM  BB    BB  LL           YY
AA    AA  SS    SS  SS    SS  EE        MM    MM  BB    BB  LL           YY
AA    AA   SSSSSS    SSSSSS   EEEEEEEE  MM    MM  BBBBBBB   LLLLLLLL     YY
```

ASSEMBLY OF THE DELILA SYSTEM PROGRAMS

At this point we will assume that all the programs and data are in files on your computer. Be sure to read the sections in PROGRAMS AND DATA DESCRIPTIONS (DELMAN.DESCRIBE.CONVENTIONS) that discusses our file naming and running conventions.

This section will guide you in the construction of the Delila System programs. There are several stages to this process:

changing characters - making sure that all the characters are correct
removing blanks - blank characters at the end of lines can be removed
   to speed processing and save memory.
changing words - changing the words that your compiler thinks
   are reserved words in Pascal (but aren't in standard Pascal...)
module corrections - making sure that modular chunks of code function
   correctly on your computer.
module transfers - inserting chunks of code into programs
compilation and debugging - making the programs and finding out why
   things don't work ("If something can go wrong, it will." - Murphy)

We have written some tools to aid you in this process - but to use the tools you must first get some of them running - so the first steps must be done by hand.

Remember to take dated notes about your problems and how they were solved.

USE OF COMMAND FILES

Most computer systems allow one to put commands in a file and execute them. If you can do this, it will speed up assembly enormously. One such "command" file could contain instructions to remove blanks, change characters, change words, transfer modules and perhaps even try to compile. However, it would be better to have several command files, each of which did a small part, giving you more flexibility.

(* end module delman.assembly.intro *)

CHANGING CHARACTERS

When characters are written to tape they are encoded as binary strings. When your computer reads the tape, the characters are decoded for storage on your computer. If the decoding does not exactly reverse the encoding, then the characters you receive will not be the same as the ones that we send. For example, you many have a pound sign for each exclamation mark that we sent. Your first task is to find out what changes occurred (if any). To aid you, we provided a list of characters with English descriptions in the file 'chars'. Look at this file and write down the changes required.

Use the editor on your computer to correct the characters in the file CHACHAS. Now try to compile CHACHAS. Determine the reasons for any errors. (For example, you may have to switch double and single quotes to satisfy the compiler or you may have to remove the non-standard linelimit call.)

The CHACHA program will now assist you in converting characters in the files from the tape. You should try it out on chars, remembering not to destroy the original file. NOTE: Some Pascal compilers may not allow programs that read "nonstandard" characters. (Example: small characters.) You may be able to get around this by setting compiler defaults.

(* end module delman.assembly.chacha *)

REMOVING EXCESS BLANKS FROM FILES

The files that you get off the tape may have extra blanks (spaces) at the ends of lines. This may be due to transportation itself, or the source computer may add extra blanks to lines. Although these blanks will not affect the function of most programs, they will slow down program execution and use up extra memory.

Transportation can also add blank lines to the end of the file. Some programs will object to this. Catal is one example.

The program Rembla (remove blanks) will remove all blanks from the ends of lines in a file, and any extra blank lines at the end. We recommend that you include this as a step during assembly of programs. It should also be done for data files, especially the libraries.

(* end module delman.assembly.rembla *)

THE RESERVED WORD PROBLEM

The language Pascal defines certain words (such as PROGRAM, VAR, BEGIN and END) to be reserved words. These words cannot be used as variable names. This in itself presents no difficulties for portability. However, your Pascal compiler (like ours) may reserve more words than just the standard set. If one of the Delila System programs uses a non-standard reserved word of your compiler, then the program will not compile. You will not have to change all these names by hand because we have sent a program to do it automatically.

Non-standard reserved words should be listed somewhere in the manual for your Pascal compiler. Use this list and the program WORCHA to remove all the reserved names. We suggest using new names that are not likely to appear in a program. Example: MODULE could be converted to ZMODULE without loss of meaning. ZMODULE is not likely to be already used in a program.

Worcha will not alter literals or comments, so the program's operation will not be affected by this change. If one makes the changes with a standard editor, then the program may not act as described in this manual.

(We hope that those people who design compilers will consider this problem in the future.)

(* end module delman.assembly.worcha *)

ASSEMBLY USING MODULES

First, familiarize yourself with DELMAN.DESCRIBE.CONVENTIONS.

You are now ready to assemble a Delila auxiliary program. The raw source LISTERR cannot be compiled as it now stands because it is missing a set of replaceable chunks of code (called modules) to read books (the book reading interface modules). These are to be found in DELMODS, as stated in the first few lines of LISTERR. Notice that DELMODS is a program - compile and run it. This will almost certainly fail. Correct those modules that cause problems. See the section on assembly problems.

Modules can be moved around using the MODULE program. The details of this process are described in MODDEF, which you should study now.

--------------------------- READ MODDEF NOW --------------------------------

(* end module delman.assembly.module.1 *)

```
     Prepare to do the module transfers by compiling MODULES.
     All programs should be tested on small inputs at first.
Test the Module program with the example module source and library:
     MODULE(EXSIN,EXMODLI,EXSOUT,EXCT,LIST,OUTPUT)
Exsout should be identical to the sout example in ModDef.
Examine list and exsout.

Now try:
     MODULE(LISTERR, DELMODS, LISTERS, DELCAT, OUTPUT)
The OUTPUT file will tell you the progress MODULE makes during the
transfer.  Modules in DELMODS will be copied into the right places of LISTERR
and the result will be LISTERS (LISTER with inserts - source code).
It will be useful to save DELCAT for further transfers from DELMODS.

Compile LISTERS.  Run the LISTER (using the default parameters):
     LISTER(EXOBK, EXOLIT)
     The file EXOLIT is a listing of the example book EXOBK.  It should be
identical to EXOLI.  The possible exception is the begin-page character:
some computers use a 1 to indicate jump to the next page, while others
use control-L.

     We would now like to know that LISTER works correctly.  To do
this requires a comparison program.  MERGE will do.  However, to
construct MERGE requires modules from PRGMODS.  Compiling PRGMODS and
running it will test interactive i/o.  The procedures in PRGMODS
that may need modification are PROMPT, READCHAR and READLINE, in
decreasing order of system dependence.  You should modify LINELIMIT
and HALT by transferring the corrected modules from DELMODS into
PRGMODS.  Prepare PRGMODS and run it.

     Prepare MERGE and use it to prove that EXOLIT = EXOLI.

     You may now construct the rest of the programs.  Note that some
of them use several module libraries.  For the next stage of setting
up the Delila System compile CATALS, LOOCATS and DELILAS.  You must
now construct the libraries: skip to CONSTRUCTING YOUR OWN LIBRARIES,
(DELMAN.CONSTRUCTION).

     NOTE FOR A SECOND TRANSPORTATION
     If you obtain a later version of the Delila System, then Delmods and
other module libraries are likely to be altered.  You will want to replace
modules in the new DELMODS and PRGMODS with your own (system dependent)
versions.  If you did this directly, you would also replace corrections
and changes to DELMODS.  To avoid this problem, simply construct a small
module library (containing for example LINELIMIT, DATETIME modules and
the interaction modules).  Then use this to change DELMODS and PRGMODS.
```

(* end module delman.assembly.module.2 *)

AN EXAMPLE OF CONSTRUCTING A DELILA SYSTEM PROGRAM

In this example we show the series of steps used to set up a Delila system program, given that the module libraries are ready (that is, they compile and run). The example is for Patser, which requires both Delmods and Auxmods. We assume that the tools needed to do this are already set up, as discussed on the previous pages. As noted in DELMAN.ASSEMBLY.INTRO, it is frequently possible to automate these steps.

1. Change Characters

```
chacha(patserr,patser1,chachap)
```

Chachap must contain the changes you determined earlier.

2. Remove Blanks

```
rembla(patser1,patser2)
```

3. Change Words

```
worcha(patser2,patser3,worchap)
```

Worchap must contain a list of special reserved words and what they are to become.

4. Insert Modules

```
module(patser3,auxmods,patser4,auxcat)
module(patser4,delmods,patsers,delcat)
```

Auxcat and delcat will be generated by Module if they were empty. You can reuse them later with their respective module libraries. The module libraries needed are listed in the first few lines of each program. It is not necessary to pickup the DESCRIBE module to compile the program.

5. Compile

Patsers is now a source code.

(* end module delman.assembly.example *)

ASSEMBLY PROBLEMS

Transportation and assembly problems occur most often because of unavoidable system dependent features of particular Pascal compilers.

INTERACTIVE INPUT

For interactive input we wrote several modules that work on our computer (INTERACT in PRGMODS). These procedures may or may not be transportable, so you may have to modify them. For example, interactive input on a cyber Pascal compiler requires the file name "input/" - you would have to remove the "/" for your compiler. (This is no longer necessary, as the source code is now under UNIX which does not require this.)

DATE AND TIME PROCEDURES

The module for date and time calls (module PACKAGE.DATETIME in DELMODS) must be rewritten. We strongly recommend that you keep the same form for the dates in libraries so that these routines remain interfaces. Changing the form of the date would make transportation of libraries difficult because they would not have the same structure in different locations.

Modules that will work on a VAX computer are in VAXMODS. You may find it easier to adapt these to your computer rather than the ones that are in Delmods.

If your computer does not have a clock, the simplest way to get this module running is to add DATE and TIME procedures in the form called by READDATETIME. These dummy procedures could return either a fixed time or a random time made by a true random number generator. The date and time is used to uniquely identify books and some data files.

QUOTES

CDC Cyber Pascal compilers require double quotes(") where the standard is the single quote (').

SOLUTION: use CHACHA to convert:

" to ' and ' to "

In some cases you will have to use two single quotes so that Pascal prints a single quote. Some programs that print 5' and 3' are Lister, Helix, Matrix and Dotmat. To convert, simply alter the constant called 'prime'.

(* end module delman.assembly.problems.1 *)

```
LINELIMIT
      In CDC Cyber Pascal compilers, output to files is limited to 1000
lines unless the LINELIMIT procedure
is called.  Your compiler may not require or recognize this silliness.
SOLUTION: The calls to linelimit are isolated to the procedure
UNLIMITLN in the module by the same name in DELMODS and PRGMODS.  Simply
surround the call (inside the modules!!!) with comments.

INTERNAL FILES (thanks to Sandy Parkinson)
      An "internal file", for the discussion here, is a file used
by a Pascal program as a scratch pad.  It is not connected to the
outside world.  Some computer systems and their Pascal compiler
require that all files be connected to the outside, as they are not
capable of creating temporary files.  At least two Delila programs
use internal files: Module and Split.  Correction of this problem
requires some programming.  It may not be possible to do it for Split.

COMPARISONS OF PACKED ARRAYS
      May cause you some problems.  One solution is to use arrays
that are not packed and to write your own comparison procedure.

THINGS THAT WE HAVE NOT THOUGHT OF...
     Please tell us!  Our address is in DELMAN.INTRO.POLICY.

     For notes on the writing of transportable programs see DELMAN.PROGRAM
     and DELMAN.DESCRIBE.CONVENTIONS.WRITING.
```

(* end module delman.assembly.problems.2 *)

```
 GGGGGG   UU    UU  IIIIIIII  DDDDDDD   EEEEEEEE
GG    GG  UU    UU     II     DD    DD  EE
GG        UU    UU     II     DD    DD  EE
GG        UU    UU     II     DD    DD  EEEE
GG        UU    UU     II     DD    DD  EE
GG  GGGG  UU    UU     II     DD    DD  EE
GG    GG  UU    UU     II     DD    DD  EE
GG    GG  UU    UU     II     DD    DD  EE
 GGGGGG    UUUUUU   IIIIIIII  DDDDDDD   EEEEEEEE
```

HELLO COMPUTER - A GUIDE TO THE NEW USER

ABOUT THIS SECTION: This section is a guide to using the computer. Whenever you have questions about the computer, this is the place to look, because the rest of the manual is about the Delila System ONLY. That is to say, we have split this manual into several parts - and it will not help for you to look for the right thing in the wrong part. The reason for this is that the information about the Delila System can be moved from one computer to another (just like the Delila System) but information about computers usually cannot be moved. DELMAN.GUIDE must be REWRITTEN for other computers and operating systems.

ABOUT THIS COMPUTER: This manual section is written specifically for UNIX operating systems. (UNIX is a trademark of Bell Laboratories.)

OTHER DOCUMENTS AND RESOURCES:
In general, ask around.

Type
    help
to get pointers.

Learn how to use the UNIX manual program (man).

The apropos program is useful for finding things.

There are hundreds of books on UNIX. Find one you like. Many people seem to like:
   UNIX for People by P. Birns, P. Brown and J. C. C. Muster
   Prentice-Hall, Inc, 1985

The easiest way to learn to use a computer is to use the computer! Obtain a login identification and plunge in.

DO NOT REVEAL YOUR PASSWORD TO ANYONE!!!

(* end module delman.guide.intro *)

SOME ADVICE TO A NEW COMPUTER USER:

1) YOU CAN'T HURT THE COMPUTER. Don't hesitate to try things and to play around!

2) After you learn how to get on and off the computer your best bet is to get a firm grip on what files are, how you can make them and how to manipulate them. The easiest way to understand what is happening is to watch it happen. You should use the commands that display your files after each file manipulation - until you have a good feeling about what is happening. If you do this you will quickly become confident about what you are doing.

3) A lot of the general principles that you pick up will be similar on other computers.

4) Be wary of the characters you type. Notice that a zero (0) is NOT the same as the capital letter O - the computer can tell them apart. This is also true for a one (1) and the small l.

5) Do not do any serious work while you learn to use the computer. You are likely to destroy some of your files. That will hurt you and not the computer. Loss of good data can be terribly frustrating.

6) If you have a problem TRY A SIMPLER CASE,
TRY TO ISOLATE THE PROBLEM.

7) An experienced advisor is worth a thousand hours of computer time.

UNCRITICAL ACCEPTANCE OF COMPUTER RESULTS

"So useful has the computer become in all branches of statistical analysis that there may be some tendency to forget that even it has its limitations. The computer cannot work magic--not yet anyway. It will do only what it is instructed to do, and the validity of the results is determined by the accuracy and adequacy of the data put in and the wisdom of the people writing the instructions. Granted, the computer can perform a great many calculations much more rapidly than mere mortals can do them. Nevertheless, speed of computational work is not the same thing as infallibility in aiding with the decision-making process. A statistical critic, of all people, should guard against being overawed by the news that certain information was turned out by a computer. The mere fact that computers are being used these days even to cast horoscopes should be ample proof that a computer is no more immune to spewing out nonsense than are real flesh-and-blood people."

-from FLAWS AND FALLACIES IN STATISTICAL THINKING
by Stephen K. Campbell (N.J. Prentice-Hall Inc., 1974), p. 182

(* end module delman.guide.advice *)

```
     HOW TO USE THE DELILA SYSTEM ON THIS COMPUTER

 Computer:  Cutterjohn and Sparky.

 The Delila System programs and documentation are kept in the directory
     ~toms/delila
 The binary forms (which you can run) are in
     ~toms/bin
 If you put this directory in your path, then they will simply be commands.
```

(* end module delman.guide.delila *)

```
PPPPPPP   RRRRRRR    OOOOOO    GGGGGG   RRRRRRR      AA     M      M
PP     PP RR     RR OO     OO GG     GG RR     RR    AAAA    MM     MM
PP     PP RR     RR OO     OO GG        RR     RR   AA  AA   MMM   MMM
PP     PP RR     RR OO     OO GG        RR     RR  AA     AA MMMMMMMM
PP     PP RR     RR OO     OO GG        RR     RR  AA     AA MM MM MM
PPPPPPP   RRRRRRR   OO     OO GG  GGGG  RRRRRRR    AAAAAAAA  MM     MM
PP        RR  RR    OO     OO GG     GG RR  RR     AA     AA MM     MM
PP        RR   RR   OO     OO GG     GG RR   RR    AA     AA MM     MM
PP        RR     RR  OOOOOO    GGGGGG   RR     RR  AA     AA MM     MM
```

(* end module delman.program *)

SUGGESTIONS ON HOW TO LEARN AND DO PROGRAMMING
(An Essay By Tom Schneider)

ABOUT LANGUAGES

A computer language is the meeting ground between the absolutely rigid requirements of a computer (it must be told exactly what to do) and the ambiguous and flexible uses of human languages (such as "go jump in a lake", "pour me a cup" etc).

Recently many academic institutions in the USA have allowed students to substitute computer languages for a knowledge of human languages. Although a knowledge of computers is becoming increasingly important in our society, this change is short sighted: no computer language is anywhere near as powerful or beautiful as those practiced by humans. With dedication one can easily learn twenty computer "languages" in a few years, whereas the polyglot is rare indeed. It is important to learn both kinds of language. For one to substitute FORTRAN for French is preposterous cheating.

HOW DO LANGUAGES WORK? COMPILERS

Every kind of computer has its own internal "machine" language. It is difficult for a person to write or read this because it consists of long stretches of ones and zero's: 01001010101110100000011 1011011110100111001010010100 1010... Every "bit" (a one or a zero) must be exactly right or the machine will not operate correctly. Most people can't deal with such immense amounts of detail. The solution is to force the computer to keep track of the details and let the person think in word-like and sentence-like units:

```
IF SUNNY THEN REJOICE
         ELSE MOPE;
```

Once one has written a set of sentences in a "higher" level language, one must have the computer convert them to its own internal machine language (this is not strictly true, but we will only discuss one method here). The process is called compiling. A self-contained and consistent set of "sentences" and "paragraphs" is called a program. Obviously one also needs a program to do the compiling - that program is called a compiler.

For example, one relatively modern language is called Pascal. A Pascal compiler sits ("resides") in ("on" - so much for jargon) a particular computer. It converts statements made in the Pascal language into machine zero's and one's for that computer (and only that computer). In other words, it converts a SOURCE code into an OBJECT code. The object code can be made to operate ("run") only on one kind of computer. (Note: the word "code" means "program". Also, on some computers one must convert the object code into "executable" code before it can be run.)

(Here is something to puzzle over. It is now common practice to write a compiler in the same language that the compiler compiles. The Pascal compiler was written in Pascal. It's like pulling oneself out of the mud by the bootstraps... how did it start?)

WHY PASCAL?

One of the first languages written was called FORTRAN. In its day (the 1950's) it was a great boon because one no longer needed to write in machine language (or even one step up, assembly). Since that time many new ideas have been incorporated into languages. Some of them (such as recursion and complex data types) fall outside the range that FORTRAN can handle. This evolution is to be expected. Yet people still try to teach an old dog, so there have been a series of "improvements" to FORTRAN. The result is a great mish-mash of dialects. For these reasons (and other things like the dread FORMAT statement) it is difficult (although not impossible) to write good transportable code in FORTRAN. ("Transportable" or "machine independent" means that the program will work on several different computers.)

Pascal is a more modern language, so it includes recently developed concepts. One can write excellent crystal clear code in this language. Unfortunately this property does not prevent one from writing poor and obscure code!

TOPDOWNING: How To Write Clear Code

There are as many ways to write code as there are people. Yet a few simple principles allow one to organize one's thoughts quickly and efficiently.

Writing a program is just like ... writing an outline.
One starts at the "top" by writing the main things to be done:

```
  Tom's Day
     I.   Morning
     II.  Travel To Work
     III. Work
     IV.  Travel Back Home
     V.   Evening
```

Then one writes the first section:

```
     I. Morning
        A. Get Up
        B. Shower
        C. Get Dressed
        D. Eat
        E. Put On Coat
```

This is repeated for the other sections. Eventually we get even deeper:

```
     I. Morning
         A. Get Up
           1. Huh?
           2. Open eyes
           3. Yawn
              ...
```

In Pascal, one dispenses with the numbering of sections. Instead, each section has a name. A section is called a procedure. Since you can read all about procedures, I won't go into more detail here.

The main advantage to this method is that if one is careful, each procedure is isolated from all the others. There is only one thing to think about at a time.

Many computer languages, including Pascal, allow one to jump from one statement to others in the program. These GOTO statements invariably lead to poor programs because one creates nests of GOTO's that jump all over the place. These can be difficult to figure out. I have seen a case where a professional programmer didn't know about an inefficient series of jumps that he had written. Even large companies sell code that is a tangled mess. Modern programmers have found that the solution is amazingly simple:

DON'T USE GOTO'S

The Delila system programs use only one GOTO, in a procedure named HALT which terminates the program by jumping to the end of the program. This is necessary because Pascal does not provide for a program abort procedure. (Pascal HALT is not standard.) There are NO other circumstances when a GOTO is required!!

A METHOD FOR WRITING PROGRAMS

This is what I do when I write a program: I have a stack of old computer paper (or standard size paper, not printer size). I write one procedure on each sheet. An entire procedure is "no longer than" one page. In fact, any procedure longer than a page is usually a warning that I need more procedures. It is not necessary at first to write the details of every procedure, only to define the procedures. Starting from the top I work down a ways, realize that I need a set of primitive procedures (eg. to manipulate text lines) so I define them, but the way they work can be written later. So as the highest levels of the program are formed, the lower levels are defined. Eventually it is time to write details of the lower levels. Sometimes the higher level can be simplified as the lower levels become clearer.

As you can tell from this description, one begins from the top, but the entire structure changes as one goes. Don't be afraid to toss out a procedure that's no good - it's only one page and the paper can be recycled.

The last point is important: be flexible. Don't keep banging your head against a logical dilemma. I have often outlined a whole program - and then tossed it out because there was a better solution. Learn when to drop. Clues: you find yourself trying to do many things at once; the primitive procedures that you have devised are awkward to use; and you find it impossible to document a procedure.

Document a procedure??

DOCUMENTATION: The Key To Immortal Code

Even in a high level language like Pascal, it is possible to have a functioning program that is not easy to understand. To define a procedure I often write down the name of the procedure, the variables (pieces of information to be manipulated) that it uses and then a few English sentences that define exactly how the variables are to be used. This is all one needs for the higher levels of the outline. Those written sentences are called comments. They are part of the documentation required to make the program easy to write and ... easy to read.

It is impossible to overemphasize the importance of documentation because nobody EVER does enough (me included).

If you don't document, within a short time (e.g. a month to half a year) you will have forgotten the details of the program - and it will be painful to figure it out again. Worse than that - nobody else will be able to work with it!

It is not hard to write out what you are trying to do in a particular section of code or procedure, and it has a real advantage: one is forced to think clearly.

There are several places in a program that ought to have comments:

PROGRAM STATEMENT - the program should state its purpose in life, how it should be used, who wrote it and the date of the latest version. Some technical details can be included.

CONSTANTS - Include a constant called VERSION and CHANGE THIS EVERY TIME THAT YOU CHANGE THE SOURCE CODE. Write the version to all output from the program. This will assure that all output can be unambiguously associated with a particular version of the program. This will save you many headaches! (Note: some computers keep track of file versions. FILE VERSIONS WILL NOT SUBSTITUTE FOR AN INTERNAL CONSTANT because the program output is not affected and it is not transportable.)

All CONSTANTS, TYPES and VARIABLES should have a short description of their purpose. DON'T USE ONE VARIABLE FOR TWO PURPOSES - you will be unable to document these cases properly and the code will be confusing.

Each PROCEDURE or FUNCTION should have a short description that tells how to use it and gives the purpose of each passed variable.

```
*****************************************************************************
*    SUMMARY: programming is vastly simplified by using two simple tactics: *
*             topdowning and documentation.                                 *
*****************************************************************************
```

A NOTE ON DATA STRUCTURES

Higher level languages, such as Pascal (but not FORTRAN) allow one to describe data in forms (structures) that resemble the way one thinks about the problem. To take advantage of these facilities, it pays to name each "variable" (a structured box into which data is put) and "type" (the structure of the box) carefully. A good name will make operations on the variable obvious, and errors will stand out because they will "sound" wrong.

LOCATING ERRORS: Debugging

Even with top down programming and documentation, errors are made.

These are called "bugs". There are several kinds:

SYNTAX - the compiler will yell at you for things like spelling mistakes
BOMBING - the program stops abruptly when it should not
LOGIC - the program produces strange results
SUBTLE - the program can't handle certain rare conditions correctly

SYNTAX - It helps to check what you type in. Since I put one procedure per hand written page, this is the easiest unit to check. Many subtle bugs can also be caught this way.

BOMBING - It is often obvious where the program died. Work backwards through the logic to find the error. Clear, top-down code makes this much easier: one can often tell immediately where the problem is. Tracing also can help. See below.

LOGIC and SUBTLE - Some computer systems allow one to trace the path that the computer follows through a program. So far I have not found these useful because they are cumbersome and they put out too much data.
A few well placed write statements will trace the program flow quite well.
(A "write statement" could print the value of a variable out for you and tell you where the computer currently is in the program.)
In Pascal, one method is to make a global constant:

```
     DEBUGGING = TRUE; (* FOR DEBUGGING PURPOSES *)
```

and use it this way:

```
     IF DEBUGGING THEN WRITELN(OUTPUT, "BEGIN PROCEDURE CIRCLE");
```

By changing the value of DEBUGGING one can turn the trace on and off.
To turn off an individual trace point, one can "comment it out":

```
     (* IF DEBUGGING THEN WRITELN(OUTPUT, "BEGIN PROCEDURE CIRCLE"); *)
```

The symbols "(*" and "*)" will make Pascal ignore the contents, because they become comments. The advantage of this over removing the statement is that it allows one to reactivate it easily.

By far, the most time saving method is to write clear, well documented code.

TESTING CODE

It is often worthwhile to test a program on a small set of examples that one has worked out by hand. You should be aware however, that correct answers to tests do not prove that the program is correct. (This may seem obvious, but it is an easy mistake to make.) Sometimes one can prove the correctness of a program. This is a current field of research in computer science.

HOW TO READ MANUALS

Obtain your own copy of the manual and begin to read. Get a general idea of how the language, editor or system works. Don't worry about details yet. As soon as you have an idea about how to do something, try it on the computer. Play. Later on, you can read through the manual seriously if you want. However there is often a lot of detail that you would have to memorize. It is simpler to know that something can be done (by reading it once lightly) and to look it up when you need to do it.

A program written for one computer may not run on another computer because the compilers for the two computers may not understand the same language. Moving a program from one computer to another is called transportation. If you are going to the trouble and effort to write a good program, then you may as well make it easy for other people to use it. Your program would then be transportable.

Obviously to be transportable, a program must be well written and documented. That is not all. You must avoid all the fancy "features" that your compiler advertises, because no one else has these. If you are forced to use some feature, then isolate it to a few replaceable procedures. We have provided you with a transportable(!) mechanism for replacing chunks of code like this - see the document MODDEF and the MODULE program.

PROGRAM MAINTENANCE... SENILITY... AND DEATH.

The most costly aspect of using computer programs is not their initial writing, but maintaining them once they are written. This is well documented in the literature. But why should a program need maintenance? Aren't they fixed text that does not change? In the simplest sense this is true. But over time, bugs in the code are found and fixed, and needs and expectations change. Programs are not static, they evolve. Good programming techniques and documentation make maintenance easier during the life time of a program, but eventually the program becomes so hard to change that one must scrap it altogether and start a fresh design. So programs have a birth, a life of use and maintenance and, finally, a senility before they die.

REFERENCES

"Pascal User Manual and Report", Second Edition, by Kathleen Jensen and Niklaus Wirth. Springer-Verlag, 1978.

"Software Tools in Pascal", Brian W. Kernighan and P. J. Plauger. Addison-Wesley Publishing Co. 1981.

"Algorithms + Data Structures = Programs", Niklaus Wirth. Prentice-Hall, Inc., 1976.

"Structured Programming", O. J. Dahl, E. W. Dijkstra and C.A.R. Hoare, Academic Press. London, 1977.

"Selected Writings on Computing: A Personal Perspective", E. W. Dijkstra, Springer -Verlag, New York, 1982.

(* end module delman.program.essay *)

A Fairy Tale For Programmers

The Three Most Important Concepts
for Writing Good Code

1. Put comments in your code.

2. Don't ever forget that six months from now your program will be useless even to you without comments.

3. Several people who published a rather well known article on using computers to study sequences (and whose names shall remain unsaid to protect the guilty) sent their programs to us two years after they had published their article. It turned out that we could not use their programs directly because we did not have available the language that they used. It was necessary to translate each line of code into our language before we could use their program. Ok, fine, we know how to do that. But despite the fact that these were old programs that they had been working on for a long time, there were almost no comments in their code. That made the translation 100 times more difficult!! One sees an equation in the code - what does it mean? If they do something in a funny way, was it a mistake or is it important to do it that way? What a headache!
We threw out their programs and wrote our own.

MORAL: Code that is not documented in English will
not survive in the long run. Therefore:
Put In Comments.
Comment As You Code, NOT AFTERWARDS - Comments Are Part Of The Code.
Change The Comments When You Change The Code, NEVER PUT THIS OFF.

Epilogue

Years later, out of curiosity, the program called CODE (COmment DEnsity) was written. We were startled to discover that the frequency of characters devoted to comments in our code averages around 30 percent!

(* end module delman.program.fable *)

```
 UU    UU   SSSSSS   EEEEEEEE
 UU    UU  SS    SS  EE
 UU    UU  SS        EE
 UU    UU   SSSSSS   EEEE
 UU    UU        SS  EE
 UU    UU        SS  EE
 UU    UU        SS  EE
 UU    UU  SS    SS  EE
  UUUUUU    SSSSSS   EEEEEEEE
```

(* end module delman.use *)

Use Of The Delila System

INTRODUCTION

This section of the Delila Manual assumes that you have read the introduction to the manual, that a Delila System is running on your computer, and that you know how to get on the computer, to make files, to modify and correct files, and to run programs (See DELMAN.GUIDE.).

There are several sources of information that you can keep in mind:
1) The papers in DELMAN.INTRO.REFERENCES will show you how we have used the Delila System.
2) LIBDEF. This is a technical specification of Delila and the libraries. However, there is a set of detailed examples that can be read profitably without reading all the definitions.
3) The section of DELMAN called Program and Data Descriptions (DELMAN.DESCRIBE) lists everything that is available to you. Whenever you want a tool to do something, that is the place to look.

In this section we will first discuss the structure of a Delila Library and how you can find your pet (pet's?) sequence in it. Next we describe how to tell Delila to go and fetch your sequences. We will then discuss programs that let you study the sequences. The sequence analysis will bring us back to Delila.

(* end module delman.use.intro *)

(* begin module delman.use.structure.1 *)

LIBRARY STRUCTURE

Think about a tree. The trunk spreads into a series of branches, sticks and twigs. A Delila library looks something like that, except that there are several kinds of branch, stick and twig, much as each twig ends in a leaf, bud or a flower.

We have given names to the kinds of branches and leaves in Delila libraries. Near the trunk there are the ORGANISM and the RECOGNITION-CLASS. An ORGANISM is a cluster of data pertaining to a real-world organism. The term "organism" is somewhat ambiguous, so it is a matter of taste as to the classification of some creatures (is a virus a traveling plasmid?). In our library T4, T7 and E. coli information is stored in ORGANISMs.

A RECOGNITION-CLASS is a cluster of data about any process that recognizes specific nucleic-acid sequences. These include chemical modification and restriction enzymes. (At present this portion of the library is not fully implemented, so we will not discuss it further.)

The library structure can be diagrammed in a schema:

```
   A-->>--B  means A has one or more of B.
   C--->--D  means C has one of D.
```

```
                             LIBRARY
                              :   :
                              V   V
                              V   V
                              :   :
                  ............:   :.............
                  :                            :
              ORGANISM                RECOGNITION-CLASS
                  :                            :
                  V                            V
                  V                            V
                  :                            :
              CHROMOSOME                       :
               : : : :                         :
               V V V V                         :
               V V V V                         :
               : : : :                         :
   ............: : : :.........                :
   :       ....:   :....      :                :
   :       :           :      :                :
  MARKER  TRANSCRIPT  GENE   PIECE....       ENZYME
   : :     :           :     : : :   :         :
   V V     V           V     : : :   V         V
   : :     :           :.....: : :   :         :
   : :     :...................: :   :         :
   : :...........................:   :         :
   :                                 :         :
  SEQUENCE                       SEQUENCE  SEQUENCE
```

(* end module delman.use.structure.1 *)

In this schema you can see that ORGANISMs have one or more CHROMOSOME branches. Once again, the term CHROMOSOME is intended to be somewhat flexible. In Delila it means a complete biological unit of nucleic-acid either DNA or RNA. For example, we refer to both the ECOLI (the 5 million base one) and the CHROMOSOME PBR322 (the 4.3kb plasmid).

Notice that real-world chromosomes are "inside" their organism. In the same way, one can think of CHROMOSOMEs to be inside their ORGANISM and ORGANISMs to be inside a library. You may think of a Delila Library either as a tree or a series of objects, one nested inside the other. A little reflection will show that these are equivalent because one can convert from one form to the other.

Every ORGANISM and CHROMOSOME has a name by which it can be identified. For example, T4 is the name of the coliphage of rII fame, while ECOLI is the name for Escherichia coli. There is other information stored at these branch points as well. An ORGANISM tells us the genetic map units used, such as centiMorgan or kilobasepair. The CHROMOSOME goes on to specify the beginning and ending of the corresponding chromosome in the given units.

Now we will delve inside a CHROMOSOME. There are MARKERs, TRANSCRIPTs, GENEs and PIECEs. What is going on? So far we have been leaning toward a description of an ideal situation where all the nucleic-acid sequence information of a chromosome would be stored inside a single data object -- a PIECE. Although this fits small phages such as PHIX174 and FD, it is nowhere near true even for ECOLI. There are many disconnected fragments of E. coli sequence now known. As sequencing progresses, the fragments will connect more and more until the entire sequence is known. So a PIECE may be either the entire sequence information in a CHROMOSOME or only one of many fragments. In this way we can store sequences in their natural arrangement, and still accommodate data that is fragmented due to technical limitations. As more sequence is obtained, the SEQUENCE inside a PIECE is extended or fused to neighboring PIECEs.

Like all the other library objects, a PIECE has a name, usually related to its biological functions. To keep all the fragments straight, each PIECE tells its location on the genetic map. The nucleic-acid sequence is stored inside a SEQUENCE, written 5' to 3'. Besides these data, each PIECE stores a useful set of information: a coordinate system.

For the purposes of identification, every published sequence is given a set of consecutive integers corresponding to basepairs or bases along the DNA or RNA sequence. This numbering scheme is captured in the coordinates of each PIECE. Using Delila, subfragments of a PIECE can be easily obtained. These are also PIECES and every base in the new PIECE has the same number that its parent did. This has WONDERFUL consequences: every printout can refer to the original published literature. It is also easy to compare the results from several analyses.

(* end module delman.use.structure.2 *)

Let's move on to the GENE, one of the other data-objects inside a CHROMOSOME. A GENE defines the endpoints of the genetic information of a protein in the SEQUENCE of a PIECE. For example, in ORGANISM ECOLI; CHROMOSOME ECOLI there is a PIECE LAC. The GENE LACI refers to this PIECE by pointing to the first G of the GTG and the A of the TGA.

A TRANSCRIPT is similar to a GENE, but it defines any region transcribed into mRNA. For consistency, we consider a tRNA to be a TRANSCRIPT and not a GENE. GENE is reserved for the coding sequence of polypeptide products.

Suppose that a mutation is known for your favorite sequence. The MARKER is designed to record the change made by the mutation. MARKERs can also record splice junctions and other interesting sequence features. In the future Delila will allow one to obtain both a sequence and its mutated forms using MARKERs.

Notice that MARKERs, TRANSCRIPTs and GENEs all refer or point to a particular PIECE. Each PIECE therefore has a "family" of related branches. It is here that the tree-like structure of the library begins to break down: some of the branches are connected to one another in a kind of network.

Now it is time to become practical. Obtain a copy of HUMCAT. This is a catalogue of the library, the HUMan's CATalogue. (Delila also has one for herself). Look around HUMCAT. Notice that it is organized by ORGANISM, CHROMOSOME, and so forth. Find a GENE or TRANSCRIPT that you are interested in. In the next section you will learn how to obtain it to play with.

(* end module delman.use.structure.3 *)

DELILA - THE LANGUAGE

WHY WRITTEN INSTRUCTIONS?

One of our major design decisions was the use of written instructions for the librarian. While we realize that this is somewhat foreboding to a new user, it does have several advantages over direct interactive use. One is that it is easier to correct mistakes in the list of sequences that are to go into the book than it is to change sequences by hand. Corrections to instructions are done with a text editor. Also, the amount of information necessary to obtain a fragment of sequence is usually less than the information in the sequence itself, so storing instructions instead of sequences is efficient. Another advantage is that a complete and concise record may be kept. As we will see later, the instructions can also be generated by auxiliary programs, allowing one to automate many complex manipulations.

WHAT IS THE DELILA LANGUAGE?

This section describes the use of the language Delila:

```
   DEoxyribonucleic-acid
     LIbrary
       LAnguage.
```

The language is not as complex or comprehensive as a natural language such as English or French. It was designed for a particular task: telling a nucleic-acid data base manager - the librarian - the set of fragments that one wants to collect for study. (The name Delila is an anachronism that we can't bear to part with...)

Since the library is structured like a tree, the language must allow one to specify individual branches. Eventually a particular PIECE will be identified, and one can request one or more fragments from the PIECE. Let us look at an example:

```
TITLE "EX1: THE LACI GENE";
ORGANISM ECOLI;
   CHROMOSOME ECOLI;
      GENE LACI;
      GET ALL GENE;
```

(Note: this instruction set is kept in the file EX1IN, so you can try it. All EXn examples are sent with the Delila System.)

Statements in Delila end with a semicolon (;) - there are five statements above. The first statement will give a title to the book. The next three specify a particular GENE in the library structure. One thinks of this as a series of steps climbing the library tree. Starting at the "root" of the library, we first named the ORGANISM ECOLI. This moves us out to that ORGANISM. Then the CHROMOSOME was chosen to be ECOLI - the main chromosome (as opposed to a plasmid such as PBR322). Next, the particular gene, lacI, is specified by "GENE LACI;".

As we noted in the section on structure, GENES point to the particular PIECE that they reside on. GENE LACI points to the PIECE LAC. Although we need not know this for the request, Delila knows it automatically. When the GET is performed, Delila will obtain the sequence of lacI from the G of the GTG through the A of the TGA.

After Delila has read each of these statements, the information about the object (ORGANISM, CHROMOSOME or GENE) is put into the book. The GET generates a PIECE that is also placed into the book.

(* end module delman.use.language.1 *)

```
     TRY IT OUT
     Type a file containing Delila instructions that specify the gene
you chose at the end of the section on library structure.  For this
discussion, we will use the name EX1IN, although you may use another
name.  Find the entry on Delila (DESCRIBE.DELILA) in the back of this
manual and run it:
     delila(ex1in,ex1bo,ex1dl)
Look at the ex1dl file. This is the Delila Listing.  The first
line will look like this:
  82/01/21 23:17:51     DELILA 1.20     PASS 1           PAGE 1
Delila performs two passes through the instructions.  Pass 1 checks for
spelling and syntax errors.  If you made a typing mistake, it will be noted
in the listing and Delila will not begin Pass 2.  Should Pass 1 be
successful, then Pass 2 begins.  Notice that there are several lines that look
something like this:
* 81/01/18 22:29:26, 80/11/19 22:17:46, LIBRARY 1: BACTERIOPHAGE
* 81/01/18 22:29:26, 80/11/19 22:17:46, LIBRARY 2: E. COLI AND S. TYPHIMURIUM
These are the full titles of the libraries from which you are pulling
sequences.  Each title has three parts separated by commas:
     1) the instant (date and time in descending order) that the library
        was created.
     2) the instant that the PARENT of this library was created.
     3) the title of the library.
Notice that Delila also prints the current date and time at the top
of the listing (if your system has these functions).  The first line of a
book or library contains its full title.  For this example, this is:
* 82/01/21 23:17:51, 81/01/18 22:29:26, EX1: THE LACI GENE
What is the "genealogy" of the book that you obtained?
     Back to the listing, Pass 1.  The instructions that you typed are
repeated on the listing.  To the left are two columns of numbers -
the leftmost is the line number and the next is the statement number
(there can be several statements on one line or one line may contain
only part of a statement).  This information is sometimes useful.
     Now let's look at the listing, Pass 2.  Notice that the instructions
that you typed are repeated again, but that there are extra lines
inserted.  In Pass 1 Delila checked for typing errors, while in Pass
2 Delila pulls out data items and places them into the book.  As
each item is put into the book, it is given a number:
     2     2        ORGANISM ECOLI;
                             #1
This is useful for some auxiliary programs.  We will discuss control of
the numbering in a later section.
     If your instructions worked then there will be two other numbers just
below the get:
     5     5              GET ALL GENE;
                            #4
                                     ^29^1111
These numbers show you the numbers of the beginning base (29) and
the ending base (1111) for the PIECE put into the book.
```

(* end module delman.use.language.2 *)

```
        RANGE DEFAULTS
     It is quite possible that you got an error message at this point:

     4     4           GENE LACZ;
     5     5              GET ALL GENE;
                            #4
                                     ^1234^100000
---ERROR(S)---------------------------^206^203
203: OUT OF RANGE AND DEFAULT RANGE = HALT
206: WE DO NOT KNOW THIS LIMIT (A WARNING)

This indicates that only part of the gene you are interested in
exists in the library.  Delila detects the fact that one end of
the GENE goes off the end of its PIECE, and says that this limit (the
end of the gene) is unknown.  (This is indicated by the 100000.)  Normally
Delila will HALT when this situation is discovered.  You can change this by
using the instruction:
     DEFAULT OUT-OF-RANGE REDUCE-RANGE;
anywhere before the problem but after the TITLE.  This resets the default
response to an out of range situation.
     In REDUCE-RANGE mode, Delila will attempt to find the closest edge
of the PIECE and use that.  The listing will show a record of what
Delila does:
     6     6              GET ALL GENE;
                            #4
                                     ^1234^100000^1419
---ERROR(S)---------------------------^206^208

206: WE DO NOT KNOW THIS LIMIT (A WARNING)
208: OUT OF RANGE AND DEFAULT RANGE = REDUCE (A WARNING)
In this case the PIECE in the book begins at 1234 and ends at 1419.
     To cause Delila to continue without putting any PIECE down in the book
one would use:
     DEFAULT OUT-OF-RANGE CONTINUE;
You may use several default statements to affect how Delila responds.
To reset the default to halting, use HALT instead of CONTINUE or
REDUCE-RANGE.  (See DELMAN.USE.CONTROL)


     Use the programs COUNT and LISTER to look at your book.
```

(* end module delman.use.language.3 *)

MORE ON INSTRUCTIONS

There are several ways to obtain sequences in a book. For example one could use:

```
TITLE "EX2: AN ABSOLUTE GET";
(* FIRST WE WILL SPECIFY THE LAC PIECE: *)
ORGANISM ECOLI; CHROMOSOME ECOLI; PIECE LAC;
(* NEXT WE WILL REQUEST A PARTICULAR FRAGMENT OF THAT PIECE: *)
GET
   FROM 29  (* THE BEGINNING ABSOLUTE POSITION *)
   TO 1111; (* THE ENDING ABSOLUTE POSITION *)
```

There are several things to note about these instructions. First, there are 5 instructions and four comments. A comment is the text between a (* and a *). You should use comments freely to document what you are doing. This is made easy by the fact that comments can extend over several lines. Delila ignores comments.

Several instructions can be put on one line (the specifications, above) and one instruction can be spread over several lines (the request).

The GET above defines two basepairs in the LAC sequence. The sequence between (and including) these bases is put into the book. Delila always puts sequence in the book 5' to 3'. Thus to get the complement of the instructions above, one simply uses:

```
GET FROM 1111 TO 29;
```

RELATIVE VERSUS ABSOLUTE REQUESTS

In contrast to EX2 we could write:

```
TITLE "EX3: A RELATIVE GET";
ORGANISM ECOLI; CHROMOSOME ECOLI; GENE LACI;
GET FROM GENE BEGINNING
      TO GENE ENDING;
```

In this case we did not state absolute numbers to define our book. Yet in all three examples (EX1, EX2, and EX3) the same PIECE will be generated in the book.

There are two ways to define a base in a sequence. One is to give its exact coordinate as in EX2. That is called an ABSOLUTE reference. The other way is to define the distance from a fixed point, as in EX3: a RELATIVE reference.

Both absolute and relative referencing have advantages and disadvantages. Using absolute coordinates allows us to pinpoint particular bases. However, Delila libraries evolve over time, and when two previously separate PIECEs are fused, only one coordinate system is kept. An absolute reference will not last. On the other hand, a relative reference will last because the GENE BEGINNING will always be the start of the gene no matter what happens to the actual coordinate system.

(* end module delman.use.language.4 *)

FORMS OF REQUESTS

By now you may have noticed that there are two kinds of GET:

```
     GET ALL ... ;
     GET FROM ... TO ... ;
```

The two positions of the FROM-TO form are independent as long as one refers to locations on the same PIECE. In absolute terms one can say

```
     GET FROM -22 TO 56; (* ABSOLUTE *)
```

or one can make it relative to a gene beginning:

```
     GET FROM GENE BEGINNING - 10
           TO GENE BEGINNING +  5;
```

One can even write instructions relative to an absolute location:

```
     GET FROM 56 - 10 TO 56 + 5;
```

This is to be pronounced "get from fifty-six minus ten to fifty-six plus five". We will come back to this form later.

MARKERs, GENEs, TRANSCRIPTs and PIECEs all have a BEGINNING and an ENDING that you can use. For example,

```
     TITLE "EX4: NON-CODING LAC LEADER";
     ORGANISM ECOLI; CHROMOSOME ECOLI;
     GENE LACZ; (* NOW DELILA KNOWS THE PIECE *)
     TRANSCRIPT LACZ;
     GET FROM TRANSCRIPT BEGINNING
           TO GENE BEGINNING -1;
```

Notice that both a GENE and a TRANSCRIPT can be specified at the same time.

AMBIGUOUS DIRECTIONS

Consider the circular genome of ORGANISM G4. The numbering of the PIECE is from 1 to 5577. Suppose that you asked for:

```
     TITLE "G4 COORDINATE PUZZLE";
     ORGANISM G4; CHROMOSOME G4; PIECE G4;
     GET FROM 1 TO 10;
```

This is ambiguous! There are TWO PIECES that run from 1 to 10: one clockwise and the other counterclockwise. In this case Delila will supply you with the clockwise fragment. However to be more specific in one's request, one would write:

```
     GET FROM 1 TO 10 DIRECTION +;
```

or

```
     GET FROM 1 TO 10 DIRECTION -;
```

But there are still two other possibilities!

```
     GET FROM 10 TO 1 DIRECTION +;
     GET FROM 10 TO 1 DIRECTION -;
```

Delila is capable of handling most requests like these. (Certain of the most complex cases remain to be solved.)

(* end module delman.use.language.5 *)

RESPECIFICATION

What if one wanted to specify more than one "leaf" (GENE, TRANSCRIPT, or MARKER) at one time? Then one would use:

```
TITLE "EX5: THE REGION BETWEEN LACI AND LACZ";
ORGANISM ECOLI; CHROMOSOME ECOLI;
PIECE LAC; (* NOW DELILA KNOWS THE PIECE *)
GET FROM (GENE LACI) ENDING + 1 TO (GENE LACZ) BEGINNING - 1;
```

This form is called a "respecification", to distinguish it from a specification.

MULTIPLE REQUESTS

After Delila has completed a GET, as in the last few examples, the specifications are still in effect and one can do more GETs, change the specification, more GETs, etc:

```
TITLE "EX6: MULTIPLE SPECIFICATION AND REQUESTS";
ORGANISM ECOLI;
   CHROMOSOME PBR322;
      GENE AMPR; GET ALL GENE; (* GET GENE OF BETA-LACTAMASE *)
   CHROMOSOME ECOLI; (* CHANGE SPECIFICATION *)
      TRANSCRIPT 16SRRNAB; GET ALL TRANSCRIPT; (* 16S RRNA *)
      TRANSCRIPT 23SRRNAB; GET ALL TRANSCRIPT; (* 23S RRNA *)
ORGANISM PHIX174;
   CHROMOSOME PHIX174;
      (* GET TWO OVERLAPPING GENES *)
      GENE A; GET ALL GENE;
      GENE B; GET ALL GENE;
```

WHEN DOES DELILA ACT?

During Pass 2, Delila places the various items into the book. Thus as ORGANISM, CHROMOSOME, GENE or TRANSCRIPT instructions are read, they are executed immediately. This is not true for the PIECE in the example EX3 because at that point Delila does not know the endpoints of the sequence desired. Delila "knows" which PIECE you are interested in, but not what particular bases. When Delila reads the GET, the bases become apparent. You can see this in the Pass 2 listing: a PIECE is not given a number, rather the number is listed for the GET that generates the PIECE in the book. The numbers are for objects in the book, not for those in the library.

(* end module delman.use.language.6 *)

AUXILIARY PROGRAMS: LISTER AND SEARCH

In the section on language, we discussed how one can use Delila to generate books containing sequences one is interested in. It is difficult to read the sequences in a book because they are in an awkward (from your viewpoint) compressed format. In every day use, we almost never look inside a book because there is a much easier way: generate a fancy listing using the program LISTER.

In the section on the Delila language you used LISTER to look at the books that you generated. (If you have not done this, then you should do it now.) As other programs, LISTER will print sequence 5' to 3'. If you want the complement, it is easy to use Delila to obtain it.

LISTER is an example of an auxiliary program. In contrast, Delila is the center of the Delila System. The purpose of Delila is the manipulation of sequence information. Other "auxiliary" programs perform tasks such as making listings or doing analyses. These programs are explained in DELMAN.DESCRIBE.

The only other auxiliary program that we will discuss here is the SEARCH program. SEARCH will search a book for a simple pattern. As you will recall, books have the same structure as libraries. As SEARCH proceeds to look into an ORGANISM it will know the name of the ORGANISM:

```
     ORGANISM ECOLI;
```

Then it will enter the CHROMOSOME:

```
     CHROMOSOME PBR322;
```

Finally it begins to search a PIECE:

```
     PIECE PBR322;
```

In other words, SEARCH can write Delila instructions that trace the search path. Suppose that we had told SEARCH to search for the pattern 5' AAGCTT 3' (HindIII). We also tell it that the FROM should be -5 and the TO +10. When search finds the site it can then write:

```
     GET FROM 29 -5 TO 29 +10 DIRECTION +;
```

29 is the position of the first A of AAGCTT in PBR322.
These Delila instructions are an answer to the search!

You should try this and the other Auxiliary programs.

(* end module delman.use.auxiliary.programs *)

DATA FLOW AND DATA LOOPS

In the section on Auxiliary programs we discussed the use of the SEARCH program to locate patterns in books. The search results appear in three ways: on the screen, in a file for printing, and as Delila instructions. These instructions can be given to Delila to generate the sequences of found sites. One can view this entire process as a flow of data between one program and the next. Since this manual can not have (nice) line figures, we strongly urge you to look at the flow figures in the published papers listed in DELMAN.INTRO.DESCRIPTION. Connecting parts of the Delila system together is much like playing with tinkertoys.

Data flowing in the Delila system can pass through a program several times. Our first example was the conversion of a book to a library and the subsequent extraction of book subsets. The SEARCH program provides a more complex case where searching of a book generates Delila instructions that can be used to create a new book. The new book is the set of located sequences. This cyclic string of events is called a loop.

Once you are acquainted with these data flow loops you can look at the SEPA program. This program deals entirely with Delila instructions of the form:

```
     GET FROM 56 -40 to 56 +60;
```

along with ORGANISM, CHROMOSOME and PIECE specifications. The SEARCH program produces instructions in this form. SEPA is used to separate instruction sets.

For example, suppose you are interested in all the AluI (5' AGCT 3') sites that are not part of PvuII (5' CAGCTG 3') sites. You have used DELILA and SEARCH to generate two sets of instructions, ALUIMIX and PVUII. You then can use SEPA to get the set that you want:

```
     SEPA(PVUII,ALUIMIX,PVUIIO,ALUI)
```

PVUIIO would be a reorganized non-redundant list of the PvuII instructions, and ALUI would list all AluI sites that are not PvuII sites. Both our second and third papers describe the way that we use SEPA. (Note: to do a search like this one must be sure that the sites are numbered the same way. The search rule for AluI would be #AGCT, while the search for PvuII would be C#AGCTG. The # symbol tells SEARCH to write the number of the following base in the instructions. This forces the SEARCH program to number the same A in the two cases.)

(* end module delman.use.data.flow *)

THE COORDINATE SYSTEM OF A PIECE

In the sections on library structure and the Delila language, we kept touching on the topic of coordinate systems for PIECEs. Delila is required to maintain the numbering of sequence fragments, and a coordinate system is the means to do so. This is not a simple problem, for one must handle both linear and circular genomes. For the new user, it suffices to know that Delila can do that, and you could skip this section.

Let us start with the simpler case, a linear PIECE. The SEQUENCE in the library is numbered consecutively from 1 to 100. So far so good, we need to record three pieces of information:

```
     CONFIGURATION: LINEAR
     BEGINNING:     1
     ENDING:        100
```

Any subset of the PIECE such as:

```
     GET FROM 40 TO 50;
```

will also be linear and can be handled by these three variables. Notice that one could:

```
     GET FROM 50 TO 40;
```

to obtain a complement. In that case the BEGINNING is greater than the ENDING and the numbering decreases.

What if the CONFIGURATION is CIRCULAR? Then based on our discussion about ambiguous directions, we should at least add a

```
     DIRECTION:    +
```

for linear sub-fragments. However the situation can be worse than that!

Let us imagine a circular PIECE in the library. It is numbered 1 to 100 in the direction 5' to 3' of one DNA strand. We then make a request:

```
     GET FROM 10 TO 90 DIRECTION -:
```

The PIECE to be placed in the book is 21 bases long, with descending numbers, EXCEPT for a COMPLETELY UNPREDICTABLE DISCONTINUITY where the numbering jumps from 1 to 100. Some more information about the "parent" coordinates must be stored.

(* end module delman.use.coordinates.1 *)

The problem is to record the necessary coordinate information and to avoid becoming confused. In the Delila System, the numbering of each PIECE has two parts: a COORDINATE part and a PIECE part.

The COORDINATE part defines the location of a sequenced region on the genetic map. Once that is established, the PIECE part tells what fragment is stored in the PIECE. Both parts are transmitted to the book by Delila, but the coordinate part is fixed and unchanging while the PIECE part will vary depending on the fragment. In summary so far:

COORDINATE part = defines the relation of coordinates to the genetic map

PIECE part = defines the relation of SEQUENCE to the COORDINATE part

For the coordinate part:

GENETIC MAP BEGINNING This number locates the beginning nucleotide of the coordinate system on the genetic map. We use these numbers to order the PIECEs in our Master library.

The COORDINATE CONFIGURATION refers to the topological shape of the coordinates. A linear genetic map could only have PIECEs with linear coordinates. For a circular genetic map, circular coordinates may be chosen, but when only a portion of the sequence is known, each PIECE may be more conveniently handled as a linear coordinate system.

A COORDINATE DIRECTION defines the orientation of the numbering system with respect to the genetic map. + means "in the same direction as", - means "in the opposite direction as".

The COORDINATE BEGINNING and COORDINATE ENDING nucleotides are integers that specify the limits of the coordinate system. They are usually the ends of the largest known contiguous sequence. The BEGINNING base corresponds to the genetic map beginning, the bases are consecutively numbered, and the ENDING is always greater than the BEGINNING number.

The coordinate system described above provides a framework for stating the exact numbering of the SEQUENCE in a PIECE. This also requires four items of information: configuration, direction, beginning and ending, all relative to the coordinate system.

The PIECE CONFIGURATION may be circular only if the coordinate configuration is also circular. When the coordinates are linear, the PIECE must also be linear.

The PIECE DIRECTION may be + or - with respect to the coordinates, representing homology or complementarity to the coordinate system.

The PIECE BEGINNING and ENDING are the numbers of the endpoints of the SEQUENCE. Both must lie within the bounds set by the COORDINATE BEGINNING and ENDING. The BEGINNING is always the 5' end of the molecule.

(* end module delman.use.coordinates.2 *)

```
     It turns out that this system handles all the confusing cases noted
earlier.  To write out the nine values of coordinates we will keep
this order:
     (GENETIC MAP BEGINNING,
      COORDINATE CONFIGURATION,
      COORDINATE DIRECTION,
      COORDINATE BEGINNING
      COORDINATE ENDING,
      PIECE CONFIGURATION,
      PIECE DIRECTION,
      PIECE BEGINNING,
      PIECE ENDING)
The linear piece that we began this section with would be:
     (1,LINEAR,+,1,100,LINEAR,+,1,100)
(The GENETIC MAP BEGINNING and COORDINATE DIRECTION are arbitrary.)

The first subset was "GET FROM 40 TO 50;":
     (1,LINEAR,+,1,100,LINEAR,+,40,50)

The complement: "GET FROM 50 TO 40;" is:
     (1,LINEAR,+,1,100,LINEAR,-,50,40)


The circular PIECE is:
     (1,CIRCULAR,+,1,100,CIRCULAR,+,1,100)
The request
     GET FROM 10 TO 90 DIRECTION -;
would make:
     (1,CIRCULAR,+,1,100,LINEAR,-,10,90)

     You should work out the results for the other three possible request on
this circular PIECE:
     GET FROM 10 TO 90 DIRECTION +;
     GET FROM 90 TO 10 DIRECTION +;
     GET FROM 90 TO 10 DIRECTION -;

HINT: It helps to make diagrams.


     The catalogue program, described in DESCRIBE.CATAL, will list
the coordinate systems for pieces of a book or library in tabular format.
```

(* end module delman.use.coordinates.3 *)

HOW TO CONTROL THE RESPONSES OF DELILA

There are several situations in which Delila manipulates the information in a library in a way that may not always be what one wants. That is, there are certain things that Delila does in the absence of any instructions. These default actions can be changed by using a special class of instructions - they are called default resets. There are four basic kinds of default (as defined in LIBDEF) but we will discuss only three of them here.

OUT-OF-RANGE DEFAULT

We discussed this default in the section on the Delila language (DELMAN.USE.LANGUAGE). A request may be outside the limits of a PIECE in a library for two reasons:
1) The place is outside the coordinate system and is therefore unsequenced (Delila calls it "unknown").
2) The place is within the coordinates, but the PIECE does not extend that far in the particular library being used.

In either case, Delila's actions will be based on the RANGE default:

DEFAULT OUT-OF-RANGE REDUCE-RANGE;

Delila will attempt to find the nearest edges of the PIECE and use these. (NOTE: there are known bugs associated with this process, although it works in almost all cases.)

DEFAULT OUT-OF-RANGE CONTINUE;

Delila will not place the requested PIECE in the book, and will continue to process any further instructions.

DEFAULT OUT-OF-RANGE HALT;

Delila will stop processing instructions. The book will not be useable by auxiliary programs.

In all cases, a warning message is put into the listing.

KEY DEFAULT

One can use this default to prevent the information about MARKERs, TRANSCRIPTs and GENEs from going into the book. For example:

DEFAULT KEY GENE OFF;

will turn off printing of the GENE information. The various data items in a library will contain free form notes about the object. (You can use the REFER program to look at these.) This command can also be used to turn off the NOTEs when one wants to reduce the size of the resulting book.

(* end module delman.use.control.1 *)

NUMBERING DEFAULT

In the section on language we discussed the numbering of the items going into a book. This command is used to control the numbering. One can turn it on or off:

```
     DEFAULT NUMBERING OFF; (* NOTHING FROM HERE ON WILL BE NUMBERED *)
```

One can set numbering for particular items:

```
     DEFAULT NUMBERING PIECE; (* ONLY PIECES WILL BE NUMBERED *)
     DEFAULT NUMBERING TRANSCRIPT GENE; (* BOTH TRANSCRIPTS AND GENES
                                           WILL BE NUMBERED *)
```

To make numbering more flexible, one can reset the number that the next item will get:

```
     DEFAULT NUMBERING 27; (* THE NEXT ITEM WILL BE NUMBERED 27 *)
```

This default can be used to make sure that particular items will have the same numbers in different books.

The number will be put into the notes of the item as the first line in the notes. This allows them to be easily found by auxiliary programs.

NOTE INSERTION

One can put one's own notes into the next object placed in the book by using:

```
     NOTE "THIS IS THE REPLICATION ORIGIN FROM PHIX174";
     GET FROM ...
```

Since this is not a default reset, it does not use the word "default". The new notes will follow the notes that were in the library. By turning off notes from the library, and using note insertion, one can replace notes in a library. Notes in PIECEs can be seen with program REFER.

One can put these default or note insertion statements anywhere in a set of Delila instructions. More details on these and other commands can be found in LIBDEF.

All the defaults have initial values:

```
   default type       initial value
   ============       ==============
   KEY
        NOTE           ON
        MARKER         ON
        TRANSCRIPT     ON
        GENE           ON

   OUT-OF-RANGE        HALT

   NUMBERING           ON, 1, ALL
```

(* end module delman.use.control.2 *)

SEQUENCE COMPARISONS AND STRUCTURE ANALYSIS

The purpose of this section is to point out auxiliary programs that can be used to compare two sequences or find structures in a sequence.

Sequence comparisons can be done with DOTMAT, which forms all possible pairs between sequences in two books. For each pair, one sequence is put on the X axis of a coordinate system and the other is on the Y axis. Both 5' ends are at the origin and X runs down the printout page while Y runs across the page. (Simply rotate the page 90 degrees counter-clockwise to get standard Cartesian coordinates.) The sequences are compared for complementarity at each possible (X,Y) pair formed between the two sequences. A "dot" is placed at a coordinate if pairing can occur. Notice that the display will be symmetrical around the line Y = X. Long stretches of pairing will run on diagonals (along segments of lines Y = -X + C). To look for homology using DOTMAT, use DELILA to obtain the complement of one of the pieces.

DOTMAT produces all possible pairings. Sometimes one wants to eliminate the short helixes, to make finding the longer ones easier. The pair of programs HELIX and MATRIX will do this.

One can use these two programs to find overlaps between sequences obtained by shot-gun cloning. Put the complete sequence on the X axis book and 20 bases from each end of the other sequence in the Y axis book. Search for long oligo's, say 15 or longer. If there is a significant overlap, you will get a response from HELIX.

Another program that can be used for comparisons is the INDEX program. With this tool you can make an index of the locations of the oligo-nucleotides in a book. The measure of the similarity between oligonucleotides in the final alphabetized list of oligo's is related to sequence homologies. This method is extremely powerful.

MATRIX/HELIX vs INDEX

MATRIX/HELIX

- advantage: The 2 dimensional plot is easy to look at.
- disadvantage: It is slow. For two sequences M and N bases long, a dot matrix operation takes MxN operations. It is so-called Order N Squared in computation time since the time to compare a sequence with itself is a function of the square of the sequence length.

INDEX

- advantage: It is fast, since the sorting algorithm is order NlogN.
- disadvantage: One can't get a feeling for the results easily. One method is to mark listings made with LISTER.

(* end module delman.use.comparison *)

HOW TO MAKE AND USE ALIGNED BOOKS

WHAT IS AN ALIGNED BOOK?

To perform statistical analysis on sequence sites (eg. ribosome binding sites, promoters, splice junctions, etc.) one needs a way to align a set of PIECEs in a book. For ribosome binding sites, we have used the A of the AUG or various points in the Shine/Dalgarno. A book is aligned by chosing one base from each PIECE to be the alignment point. The alignment bases could be chosen by a list of coordinates, but we have found that there are advantages to using Delila instructions to specify the base:

```
TITLE "EX7: ALIGNED BOOK";
ORGANISM ECOLI; CHROMOSOME ECOLI;
PIECE LAC;
GET FROM 29 -5 TO 29 +10; (* LACI RBS *)
GET FROM 1234 -5 TO 1234 +10; (* LACZ RBS *)
```

Here, the zero point for LACI alignment is base 29 and for LACZ it is base 1234. The "from parameter" is -5 and the "to parameter" is +10. The instructions allow one to align the book that is created from the instructions. WARNING: the instructions must follow a rigid format; this is described in DELMODS in module info.align, along with details on how to write programs using aligned books.

(See also DELMAN.USE.DATA.FLOW and DESCRIBE.ALIST)

AUXILIARY PROGRAMS FOR ALIGNED BOOKS

After generating an aligned book (a book and an aligning instruction set) one can list it using program ALIST or obtain a histogram that tells the composition of the book at each point relative to the aligned base with HIST. A chi-squared analysis of an aligned book is done using HISTAN.

GENERATING A SET OF ALIGNED RIBOSOME BINDING SITES

We have provided the instructions for creating a set of aligned gene starts, in file GAIN. GAIN was originally created from instructions of the form:

```
ORGANISM ...; CHROMOSOME ...;
GENE ...;
GET FROM GENE BEGIN TO GENE BEGIN +2;
...
```

This is file GRIN (genes relative to begin instructions). The resulting book was searched (one would use SEARCH with a rule of (A/G/T)TG) to generate the instructions in aligned form. GAIN was then made by replacing the from-position with the word FIRST and the to-position with LAST. To use GAIN you must first create the transcript library from file TRAIN (TRAnscript library Instructions, use DELILA with LIB1 and LIB2). Then replace FIRST and LAST with the desired range. Notice that there are a few cases, marked "SPECIAL" that you must deal with individually. Notice also, that genes that are oriented in the direction opposite the PIECE had to be set up by hand (this may be automated someday). The instructions could now be named GAIN1, and DELILA can be used to generate the aligned book.

A detailed example of these operations is given in DELMAN.CONSTRUCTION.EXAMPLE.

(* end module delman.use.aligned.books *)

(* begin module delman.use.perceptron.1 *)

USE OF THE PATTERN PROGRAMS

"Perceptron" is the name given to a class of algorithms for pattern recognition with learning capabilities. Minsky and Papert have written an excellent book on the topic ("Perceptrons", MIT Press, 1969) which explores both the limitations and potentials of the method. They also prove the "Perceptron Convergence Theorem" which guarantees that a solution will be found if one exists. We have written an article (Stormo, et. al., 1982, Nucleic Acids Research, 10: 2997-3011) which describes our use of the algorithm to investigate translational initiation sites.

The algorithm takes as input patterns which can be divided into two classes, and finds a "Weighting Function" which serves to distinguish the patterns in the two classes. More rigorously, if we encode a sequence into a string of bits, S, the algorithm attempts to find a W such that W*S >= T (some "threshold") if and only if S belongs to one class of the two classes of sequences. We mean by "*" the dot, or inner product of S and W, which are vectors of the same dimensions. If we start with two sets of sequences, S+ and S-, and an arbitrary W and T, the algorithm can be described by the following three step procedure:

```
      Test: choose a sequence S from S+ or S-,
            if S is in S+ and W*S >= T go to Test,
            if S is in S+ and W*S <  T go to Add,
            if S is in S- and W*S <  T go to Test,
            if S is in S- and W*S >= T go to Subtract;
      Add:  replace W by W + S,
            go to Test;
      Subtract: replace W by W - S,
            go to Test.
```

An example of this process is shown in our NAR paper (reference given above). (Note: this process can be done without goto's...)

The program which implements the perceptron algorithm to work on sequences is called PatLrn. Other programs which use the output of PatLrn are:

- PatLst - a lister program for the output of PatLrn;
- PatAna - does some simple analyses of the output of PatLrn;
- PatVal - evaluates the aligned sequences in a book by the PatLrn output;
- PatSer - searches a book for sites which are evaluated with a given PatLrn W output to be above some user specified value.

(* end module delman.use.perceptron.1 *)

EXAMPLES FOR THE PATTERN PROGRAMS

The files "exspbk" and "exsnbk" are the sets of positive and negative sequences used in the example of Figure 1 of our "Perceptron" paper (NAR 10, 2997-3011). The file "expa1" contains the initial pattern from that same example. Given these files and the program "PatLrn" you can recreate the example thusly:

```
     PatLrn(exspbk,a,exsnbk,b,pat,expa1).
```

The file "pat" should be identical (except for the date/time) to the file "expa2" that we have provided. You can check that with the "Merge" program if you want. It is also identical to the solution pattern from the example and it keeps track of the number of changes needed to get to that solution. The files "a" and "b" are empty in this case, because we are aligning the sequences by their first bases. If we wanted to align them by any other base those files would contain the instructions which generated the sequences (see DELMAN.USE.ALIGNED.BOOK).

Now use the program "PatAna" to do some simple analyses of the pattern.

```
     PatAna(pat,patan).
```

The file "patan" is identical to the file expan2 that we provided. It contains some useful information about the pattern, such as the minimum and maximum sequence values which could be obtained from this pattern, as well as the average value expected for random sequences and a feeling for the distribution of values.

The program "PatVal" will use a pattern to evaluate a book of sites. Try:

```
     PatVal(exspbk,a,pat,valp).
        and
     PatVal(exsnbk,b,pat,valn).
```

"valp" is the evaluation of each sequence of the positive class, and "valn" is the evaluation of each of the negative class sequences. Check with the example in the paper to see that they are correct. Again the "a" and "b" files are empty because we are aligning by the first base of the sequences.

The program "PatSer" will use a pattern to search through a sequence, using each base in turn as the aligned base. Those sites which are evaluated above some minimum, either set by the user or taken to be the minimum functional from the pattern itself, are identified. Furthermore, instructions to get those sites so identified are written to the file "inst". Try this on an example file:

```
     PatSer(exsebk,pat,val,inst).
```

notice that when the pattern extends beyond the sequence the sites are still evaluated, but the user is notified of the over-extension.

The program "PatLst" is used to make nice horizontal printings of the patterns, such as for use as publishable figures. Try this on the W51 matrix which is from the paper and which we provide. Read the page DESCRIBE.PATLST to see how to set the width of the pattern printed to a page to whatever you want.

(* end module delman.use.perceptron.2 *)

A NOTE ABOUT SIGNIFICANCE

While the example we provide in the paper, and that you have just done, is convenient for demonstrating the method, separating two sets of two sequences, each five long, is in fact trivial. Try:

```
PatLrn(exspbk,a,exsnbk,b,newpat).
```

"newpat" is identical to "expa0" that we provided, and as you can see is not interesting. The mathematical problem of when it becomes significant that one can separate two sets of sequences is still an open problem, but we can say some things. As the number of sequences in each class gets larger the probability of separation decreases, as it does when the number of nucleotides in each sequence diminishes. As a good rule of thumb we like to have more sequences in the smallest class (usually the functional class) than there are nucleotides in any one of the sequences. Under these conditions one can be reasonably confident that a solution pattern is likely to identify features of biological significance.

(* end module delman.use.perceptron.3 *)

USE OF THE "ENCODE" PROGRAM

The program Encode was written to allow a user to encode sequences into strings of integers in a flexible way. For instance, one can encode the sequences as mono-, di-, tri-, or higher oligonucleotides. One can assign specific oligos to certain positions or record only that they are within some "window" of positions. Within a window all the oligos may be counted or only some, such as only those "in frame".
The program takes as input the book of sequences and the instruction set which generated it and which specifies the alignment. If the instruction file is empty then all the sequences are aligned by their first bases.
The other input file, which must be non-empty, is the parameter file "EncodeP" which specifies how the sequences are to be encoded. It is the options of the parameter file which give the program its flexibility and power, and so they should be thoroughly understood.
The parameter file may contain any number of individual parameter records, each of which will in turn be applied to each sequence in the book. This allows one to encode different regions of the sequences differently, or to encode one region in more than one way. Each parameter record has five pieces of information, each written on a separate line:

line 1 - the range over which this parameter record is to operate; this line has two integers which are the bases, relative to the aligned base, for which to use this encoding;

line 2 - the size of the window; the window begins at the start of the range and contains this many nucleotides in it; the number of each base, or oligo, which occurs in this window is written to the output; note that positional information within the window is lost, so that if exact position is needed the window size should be 1;

line 3 - the shift to the next window; this specifies how many bases to move the window over to its next position; this is repeated until the window begins beyond the end of the range;

line 4 - this specifies the coding level, and the arrangement of the bases to be coded; the coding level is the number of bases in the oligos which are encoded, i.e., 1 means monos are encoded, 2 means dis are encoded, ...; for coding levels greater than 1 the user may allow for skips between the encoded bases; for instance, one may want to encode as di-nucleotides bases which are separated by a nucleotide; this would be declared on this line by writing "2 : 1"; likewise, one could encode as a tri-nucleotide the first bases of three consecutive codons by the line "3 : 2 2", where the 3 indicates the coding level (tri-nucleotides) and the 2's represent the number of bases skipped between each encoded base; if there is no colon after the coding level declaration, all skips are assumed to be 0;

line 5 - the shift to the next coding site; this allows the user to not count every occurrence of the oligos in the window, but rather to move some number of bases to the next encoded site; if all the oligos are wanted, this number should be 1.

The above line information constitutes a single parameter record. The parameter file may contain any number of these records concatenated together. Each sequence will be encoded by the entire list of parameter

```
records and the resulting string of integers will be written to the
"EncSeq" file.  The encoded string for each sequence ends with a special
"end of sequence" symbol, which is listed in the file header.
For examples of how this program works see "DELMAN.USE.ENCODE.2".

(* end module delman.use.encode.1 *)
```

EXAMPLES OF USING THE "ENCODE" PROGRAM

The files "ExEncIn" and "ExEncBk" contain the sequence around the beginning of the rIIB gene of T4, and the instructions which align this sequence by the ATG of the gene. The aligned sequence looks like:

```
---                 ++
111--------- +++++++++11
21098765432101234567890 1
.......................
ATAAGGAAAATTATGTACAATATT
```

Notice that the 0 base is the A of the ATG (this is what we aligned by) and that our sequence contains the 12 preceding bases and the 11 following. This is through the fourth amino acid of the protein. If we wanted to encode only the mono-nucleotides of the initiation codon we would make our parameter file:

```
   0 2
   1
   1
   1
   1
```

this would give the encoding:

```
 1 0 0 0 0 0 0 1 0 0 1 0 -1
```

Notice the -1 which specifies the end of the encoded sequence. Each 4 integers before that specifies which base occurs at each of the three encoded positions. The A is encoded as 1 0 0 0, the T as 0 0 0 1, and the G as 0 0 1 0.

If we wanted to know the number of each mono-nucleotide in this whole region and we didn't care about their positions, we would encode as:

```
   -12 11
   24
   24
   1
   1
```

This would give the encoding:

```
 12 1 3 8 -1
```

Notice that this is really just the composition of the sequence, since our window covers the entire sequence. We could get the di-nucleotide composition with the parameters:

```
   -12 11
   24
   24
   2
   1
```

and get the encoding:

```
 5 1 1 5 1 0 0 0 1 0 1 1 4 0 1 2 -1
```

Notice that this encoded string is a vector of 16 integers (up to the end of sequence mark, -1). The number in each element of the vector is the number of each di-nucleotide in the sequence, in the order AA,AC,AG...TC,TG,TT.

Examples continued in DELMAN.USE.ENCODE.3.

(* end module delman.use.encode.2 *)

Examples of using the "encode" program, continued from DELMAN.USE.ENCODE.2.

To encode the di-nucleotide composition of the Shine and Dalgarno region and also the mono-nucleotides of the coding sequence, each in its own position, we would make this list of parameters:

```
   -10 -6
   5
   5
   2
   1
   0 11
   1
   1
   1
   1
```

This would give us the encoding:

```
 2 0 1 0 0 0 0 0 1 0 1 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 1 0 0 0 0 1
 1 0 0 0 0 1 0 0 1 0 0 0 1 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 1 0 0 0 1
 -1
```

Here the first 16 integers are the di-nucleotide composition of the Shine and Dalgarno region, and appended to that are the mono-nucleotide encodings for each position of the coding sequence. We could get the di-nucleotides of successive codon first positions by:

```
   0 11
   12
   12
   2 : 2
   3
```

or we could get the codon composition by:

```
   0 11
   12
   12
   3
   3
```

or we could get the di-nucleotide encoding of the first and last position of each codon, including the position of the codon by:

```
   0 11
   3
   3
   2 : 1
   3
```

These are left as exercises to the user, and it is encouraged that the user make up other tests and try them until this program is easy to use.

(* end module delman.use.encode.3 *)

In addition to Delila, there are at least two other generally available large nucleic sequence data bases. The DB program system handles both the European Molecular Biology Laboratory (EMBL) libraries and those of the Genetic Sequence Databank (GenBank(TM)).

If you want to contact someone who helps operate these data bases use the following addresses:

GenBank
c/o Computer Systems Division
Bolt Beranek and Newman Inc.
10 Moulton St.
Cambridge, Ma. 02238
USA

Graham Cameron
European Molecular Biology Laboratory
Postfach 10.2209, 0-6900 Heidelberg, West Germany

The DB program system is a small set of programs. DBcat prepares catalogs for DBpull. DBpull extracts part or all of an entry of either EMBL or GenBank format. DBbk converts database entries into the Delila book form that Delila programs use. All of these programs handle both data base formats even when both occur together in the same library.

At this point, please obtain some sample library entries from both data bases and look them over.

Embl and GenBank libraries are arranged in series of entries, each entry possessing a unique entry id, a nucleic acid sequence, and other miscellaneous information. Most of the lines in the libraries start with a word or abbreviated code that indicates what kind of information the line contains. The following definitions will clarify these points.

Library definitions:

Entry: An entry starts with a line which begins with an "ID" (EMBL) or a "LOCUS" (GenBank). All subsequent lines are part of the entry until the line that contains simply "//". "//" is the entry terminus code for both data bases.

Entry id: On the first line of each entry, after the "LOCUS" or the "ID", comes a few spaces and then a weird looking word or code that may or may not resemble a familiar biological name. This is the entry id, it is the name the entry is known by and it is what DBpull uses to identify which entries it will extract.

Line codes: The phrases "ID" and "LOCUS" are line codes. There are other line codes in each entry such as "REFERENCE" and "ORIGIN" in GenBank and "DE" "SQ" in EMBL. Some lines do not have a code and some have one, but it is indented. Other lines have codes, but there is no other information on the line. these special cases will be discussed below in the definition of line code request instructions.

Now that you are familiar with the data bases you can understand the DBpull instruction set. Each instruction takes up only one line. Each line does one of two things; either it indicates what entry type (GenBank or EMBL) is requested on the following lines or it makes an actual request for part or all of an entry identified by its entry id. Please note that the following definitions will be made clearer by referring to the examples that follow.

(* end module delman.use.dbpull.define *)

Note: Instructions are entirely upper case because that is what the computer system DBpull was designed on required.

Instructions that determine entry request type of succeeding lines:

EMBL: This indicates that requests for entries somewhere in the EMBL libraries will be on the following lines.
GENBANK: Same for requests found in the GenBank libraries.
GENB: Same as "GENBANK".

Instructions that tell which entries are to be pulled:

Entry id: An instruction line beginning with an entry id will pull part or all of that entry. The parts extracted will depend on which of the "instructions that define extraction" (defined below) follows the id on the same line.

Wildcard id: This request looks like an entry id request but somewhere in the entry name are one or two "*" symbols. The "*" represents any number of unspecified characters. It may be inserted at the beginning of the id, at the end, or at both the beginning and the end but not the middle. (Confused? see instructions example 3 below)

EVERY: The word "EVERY" at the start of a request line calls for every entry of a particular entry type. (See instruction example 4)

Instructions that define extraction:

Line codes: Following the instruction that tells which entry or entries are to be pulled, on the same line, come instructions that structure the extraction. One or more line codes occurring in this space will result in the lines of the entry which have matching codes being pulled. Genbank line codes are actually words. The full word or an abbreviation will work, but the abbreviation can not be shorter than 3 letters. "LOC", for instance, will pull the "LOCUS" line while "LO" would not. When there are one or more lines in the entry directly below a pulled line that either do not possess a line code, posses indented codes, or posses the code "xx", these additional lines will be extracted also.

RAW: Instead of line codes one can simply insert the word "RAW". This will pull only the sequence of the entry without origin or coordinate labels. The sequence will end with a "." to separate it from other sequences and to make it suitable for input into Makebk. (see delman.describe.makebk) Also, if the first request of fin is "RAW", fout will have no dateline and therefore it will not make a suitable secondary data base for DBpull.

ALL: Instead of "RAW" or line codes the word "ALL" will result in an entire entry being extracted.

(* end module delman.use.dbpull.instructions *)

Instruction examples (DBpull input file Fin)

Example 1:

```
EMBL
ADCXXX ID DE SQ
GENBANK
M13 LOC REFERENCE
ANABANIFH LOCUS
```

Comments: The first and third lines indicate what types of entries are requested on the following lines. If, for instance, M13 were an EMBL entry this set of instructions would not find it.

Example 2:

```
GENB
T7 RAW
MS2 ALL
```

Comments: The two requested ids are not in alphabetical order and the DBpull output file fout will have the same order as the requests.

Example 3:

```
EMBL
*RNA SQ ID
*RNA* ID SQ
GENB
M* ORI SITES
GOOGOOGAGA ALL
T7 RAW
```

Comments: The character "*" is a wildcard; it represents any number of unspecified characters.
The first request will grab any entry whose id ends in "RNA", the second any one that has "RNA" anywhere in it, and the third any id which starts in an "M". The fourth request is a joke and, like any other non-existent id, will yield a "not found" message and then halt the program. If there were no GenBank entry ids beginning in "m" a "not found" would appear but DBpull would not halt because this id request is a wildcard. The logic behind this distinction is that wildcards are used to search for the possible existence of an entry, but regular ids are used only for entries that are well known by the user. Note that "ORI" (origin) pulls sequence in GenBank and "SITES" tells you where the genes and other features are. "SQ ID" and "ID SQ" are equivalent; lines are pulled in the order that they occur.

Example 4:

```
EMBL
EVERY ID
GENB
```

Comments: This example would make a catalog for users of the entire EMBL and GenBank data bases. The catalog would be alphabetical because the catalog files used by DBpull (produced by DBcat) are presorted. If "catalogs for humans" are provided with your libraries do not try this example; it is very expensive. If you do try it, you might want to request additional line codes to "LOC" and "ID" for a more informative catalog.

(* end module delman.use.dbpull.examples *)

(* begin module delman.use.search.1 *)

Use of the Search Program

i. searching dna sequences for particular strings

The search program works on books of sequences. Any search pattern will be looked for in each sequence of the book. Search patterns consist of strings of nucleotides, such as 'aatggct'. You may also specify ambiguous patterns, such as 'a or g', in either of two ways: '(a/g)' or 'r'. All possible ambiguities can be asked for, by either way. From within the search program type 'l' to see the list of one-letter codes for each ambiguous base combination. One can also include in the search positions for which you don't care what the base is, indicated by 'n'. For instance, 'anc' would search for a and c separated by any base. One can also use 'e' (for extension) to vary the spacing between specified regions. The 'e' is considered to be an 'n' and also as nothing. For example, 'aec' would search for both 'anc' and 'ac'. We used this feature to search for 'shine and dalgarno' sequences before 'atg's by specifying 'gga5n4eatg'. This means 'gga followed by 5 to 9 unspecified bases followed by atg'.

One can search for strings which are close to the specified by allowing mismatches to the specified sequence. This is done by typing 'm' as a search command, and then specifying how many mismatches are allowed. If there are regions within the specified sequence where you want no mismatches, this is stated by enclosing that region between and '<' and '>'. For example, if mismatches were set to 1 and the pattern searched were 'aat<ggc>t', then the 'ggc' must be found exactly, but the rest of the pattern need only be within one of a perfect match.

The search program returns to you the positions of the matches found in the book. Unless otherwise specified, the position corresponds to the first base of the pattern. However, one can ask for the position to be another base by preceding that base by '#'. For example, 'aa#atggct' would return as the position of the match the 'a' of the 'atg'.

It is also possible to make searchs for relations between bases. Six relations are allowed: identity (i); non-identity (ni); complementarity (c); non-complementarity (nc); complementarity including g-t pairs (w); and non-complementarity including g-t pairs (nw).
Relational searchs are specified by first
the symbol '^', followed by the pattern position this base is to be related to, followed by the relation. For example, 'n^1i' would find all sites in which there is a repeated base (aa, cc, gg or tt). Notice that the base to which the relation refers must proceed the point of the relation in the pattern. Searching for the pattern '5n^1c' would find sites of complementary bases separated by 4 unspecified bases.

More information on search patterns and other commands in general can be obtained by typing 'help' while in the program.

(* end module delman.use.search.1 *)

ii. Creating Delila Instruction Files

The search program also allows one to create instruction files so that the located sites may be put into a book for further analysis. This is especially useful when you want to include in the analysis regions around the sites. For instance, you could set the 'from' distance to -60 and the 'to' distance to +40. Then by searching for 'gga5n4e#atg' you would get the instructions necessary to obtain the sequences from -60 to +40 around the atg's which are preceded by Shine and Dalgarno sequences. Help on using this feature of the program can be obtained by typing 'd help' while in the program.

(* end module delman.use.search.2 *)

```
           cccccc    oooooo   n      nn
          cc    cc  oo    oo  nn     nn
          cc        oo    oo  nnn    nn
          cc        oo    oo  nnnn   nn
          cc        oo    oo  nn nn nn
          cc        oo    oo  nn  nnnn  --------
          cc        oo    oo  nn   nnn
          cc    cc  oo    oo  nn     nn
           cccccc    oooooo   nn     nn


 ssssss   tttttttt  rrrrrrr   uu    uu   cccccc   tttttttt
ss    ss     tt     rr    rr  uu    uu  cc    cc     tt
ss           tt     rr    rr  uu    uu  cc           tt
 ssssss      tt     rr    rr  uu    uu  cc           tt
      ss     tt     rr    rr  uu    uu  cc           tt
      ss     tt     rrrrrrr   uu    uu  cc           tt     --------
      ss     tt     rr  rr    uu    uu  cc           tt
ss    ss     tt     rr   rr   uu    uu  cc    cc     tt
 ssssss      tt     rr    rr   uuuuuu    cccccc      tt


          iiiiiiii   oooooo   n      nn
             ii     oo    oo  nn     nn
             ii     oo    oo  nnn    nn
             ii     oo    oo  nnnn   nn
             ii     oo    oo  nn nn nn
             ii     oo    oo  nn  nnnn
             ii     oo    oo  nn   nnn
             ii     oo    oo  nn     nn
          iiiiiiii   oooooo   nn     nn
```

(* end module delman.construction *)

```
     CONSTRUCTION OF DELILA LIBRARIES

     Introduction
     This section assumes that you are familiar with DELMAN.USE.
Construction of a Delila System Library involves several steps:
     - Entry of the raw sequence data (twice)
     - Correction of the sequences
     - Gathering of the information about the sequences
     - Creation of a "module" for insertion into the library
       (not the same module type as the ones used by program Module.)
     - Insertion of the module
     - Construction of a catalogue
     - Checking that the library is correct.

     When you are gathering the data to create part of a library
(the library insertion module) you may find the forms in
DELMAN.CONSTRUCTION.FORM useful.  Use the Module program to make
as many copies as required.


     NOTES FOR TRANSPORTATION
     Since the libraries that we send you have already been checked, you
need only run the CATAL program (as discussed below) to generate the
catalogues for these libraries.  After that, Delila can be used.
```

(* end module delman.construction.intro *)

MORE ON LIBRARY STRUCTURE - LOGICAL VS PHYSICAL STRUCTURE

In DELMAN.USE.STRUCTURE we discussed the structure of a Delila Library. The descriptions were about how the parts are connected, and what is inside each part. This is the logical structure of the data base. We did not discuss the details of how a library is actually constructed, because it is not necessary to know these things when working with the Delila System. The description of these details is the description of the physical structure of the data base.

Since we do not yet have an extensive set of tools for constructing Delila Libraries, it is necessary to describe the physical structure enough so that you can build your own libraries. Because these details are rigorously stated in LIBDEF, most things are automated by program Makebk, and Catal does lots of checking, we will only discuss the general concepts here.

The logical structure of a library follows the schema shown in LIBDEF or DELMAN.USE.STRUCTURE. This structure is a two dimensional net. Libraries are implemented physically in files, and so are linear structures. If we exclude for the moment the references to a PIECE by MARKERs, TRANSCRIPTs and GENEs, then the library structure is a a tree. Any tree can be represented as a nested series of objects in linear order:

```
ORGANISM   (open  parenthesis for an ORGANISM)
CHROMOSOME (open  parenthesis for a  CHROMOSOME)
GENE       (open  parenthesis for a  GENE)
GENE       (close parenthesis for a  GENE)
PIECE      (open  parenthesis for a  PIECE)
PIECE      (close parenthesis for a  PIECE)
CHROMOSOME (close parenthesis for a  CHROMOSOME)
ORGANISM   (close parenthesis for an ORGANISM)
```

If you look at any book (eg. EXOBK) or library (eg. LIB1) you will see this structure. Lines in a library either define the structure or are chunks of data (attributes). Attributes are signaled by an asterisk (*) as the first character on the line.

We must now allow various objects to refer to PIECEs. This is done by a reference to the name of the PIECE. For example, one of the attributes in a GENE is the name of the PIECE that the GENE is on. (In cases where the GENE spans two PIECEs, we use two GENEs.)

To simplify the operation of the CATAL program (to be described later) we have added one more rule. All objects that refer to a particular PIECE are called the "FAMILY" of the PIECE. The rule is that a FAMILY precedes its PIECE in the physical (file) implementation.

(* end module delman.construction.structure *)

MAKING NEW LIBRARIES - THE CATALOGUE PROGRAM

The first technical difference between Libraries and Books in the Delila System is that Libraries have catalogues while Books do not. Catalogues serve several purposes. First, since they are a condensed list of the objects in a Library, they allow objects to be found quickly. There are catalogues for both Delila and for people (the latter is called a HUMCAT - HUMan's CATalogue). These are constructed by the program CATAL.

Since a library may be constructed by hand, it is also convenient to check the Library's physical structure at the time the catalogue is made.

The Problem Of Duplicate Names

Using Delila, a Book may be easily constructed that contains two objects within the same structure (if they are in different structures, it won't matter). For example:

```
ORGANISM ECOLI;
   CHROMOSOME ECOLI;
      GENE LACI; (* THIS IS ON PIECE LAC *)
      GET ALL GENE DIRECTION HOMOLOGUOUS;
      GET ALL GENE DIRECTION COMPLEMENT;
```

If this Book were to become a Library, then a reference to PIECE LAC would be ambiguous since there are two PIECEs with that name within the CHROMOSOME. The CATAL program detects these cases and makes the names differ by adding symbols to the names of second and subsequent duplicately named objects. The second technical difference between Books and Libraries is that Books may have duplicate names, while Libraries may not.

Notes For Transportation

Unknown ends of objects (such as a GENE) are represented in this version by a number that is off the end of the coordinates of the PIECE. For consistency, we have used +100000 or -100000 so that these can be more easily recognized (to our knowledge no continuous sequences are this long ... yet!). If your computer cannot handle integers this large, then you can reduce these numbers, as long as they are outside of the individual coordinates.

(* end module delman.construction.catal *)

AN EXAMPLE OF CONSTRUCTING DELILA LIBRARIES

In this example we show the series of steps used to set up the Delila libraries provided on the tape. The special bracket notation ([...]) is used here to indicate the contents of a file. A slash (/) inside the brackets indicates the beginning of a new line in the file. Other notation is described in DELMAN.DESCRIBE.CONVENTIONS.

1. Generate Library Catalogues

```
catal(humcat,[ADVANCE DATES],lib1,cat1,newlib1,lib2,cat2,newlib2)
copy(newlib1,lib1)
copy(newlib2,lib2)
```

The humcat should be identical to or similar to the one we send. (Note: l3 is empty, and c3 and newlib3 will not be written, but your computer may require that these files exist as empty files in order to run Catal. A similar situation holds for Delila and many other programs.)

2. Build Transcript Book

```
delila(train,trabk,tradl,lib1,cat1,lib2,cat2)
```

There will be warnings that can be ignored at this point.

3. Build Transcript Library

```
catal(trahu,[ADVANCE DATES],trabk,tract,trali)
```

You will see a number of cases where duplicate names are resolved.

4. Test Grin File

```
delila(grin,grbk,grdl,trali,tract)
comp(grbk,cmp,[3])
```

cmp should show 140 ATG, 7 GTG, 2 TTG.

5. Test Gain File

Within the Gain file, the "FIRST", "LAST" and "SPECIAL" cases must be replaced by numbers. The WORCHA program comes in handy here, because it will do this easily:

```
worcha(gain,ga3in,[FIRST/0/LAST/2/SPECIAL/0])
delila(ga3in,ga3bk,ga3dl,trali,tract)
comp(ga3bk,cmp,[3])
```

cmp should be the same as for Grin.

6. Expanding Grin

You can now expand the "FIRST" to "LAST" region of Gain, taking care not to violate the "SPECIAL" cases.

(* end module delman.construction.example *)

RULES OF RAW SEQUENCE INSERTION

(1) A raw sequence is a file containing only the letters A, C, G or T (no U is allowed, use T). You may type these letters or a set of letters on the keyboard that is convenient (eg. 1234); then convert the letters to ACGT using the program CHACHA.

(2) For reasons of transportability and readability, the length of each sequence line should not exceed the width of characters on a typical terminal: Do not type more than 60 bases per line. You can reformat the data with REFORM or MAKEBK.

(3) Sequences can and should be entered in free format with spaces to improve the readability of the sequence during entry. This also helps in the corrections described below. Much later it helps one to find parts of the sequence during fusion of PIECEs.

(4) Before entry, use a pencil to mark off intervals of sequence to type. This makes entry easier since there are rest points. I often check off each (or every other) interval as I go, so I rarely get lost and duplicate or delete intervals. If you can keep the lines like those in the paper, the sequence will be easier to check and correct later (but remember rule 2).

(5) Two people should INDEPENDENTLY enter the sequence. Independence is important: one person will FREQUENTLY make the same mistake twice. Do not be fooled into entry of a sequence and its complement by one person. We have had two cases where the same deletion was entered in the same place by one person, even though he was typing the sequence and its complement. Have two people independently type the sequence and the complement. By doing it this way, you will also catch some typographical errors if you are using a published source. (Another method: if one person is to enter both strands, be sure that they are typed from two copies on which different intervals are used.)

The method of independent entry allows automatic correction. It seems to be faster and more reliable than other methods.

(6) I caught the deletions mentioned above by knowing how long the sequence should be. You should not rely on the computer for the length. Predict it and then check it.

(7) The file names of the two copies should include the initials of the person who typed the file. See the example below.

(8) A complemented or inverted strand may be re-complemented or re-inverted using the program REFORM. Note that the free format of (3) will be lost. You should use the reformatted sequence only for checking, and not for the final Library insertion, since you would lose the formatting if you did.

(9) At this point you have two files of "raw" sequence. The sequences may be merged together and corrected using MERGE.

FOR EXAMPLE: If the sequence was OMPA, TS and MA typed the raw copies, and the copy of MA contains the format desired for the Library, you could use MERGE like this:

```
     MERGE(OMPAMA,OMPATS,OMPA,GARBAGE)
```

(10) Be sure to save all raw files (eg. OMPAMA, OMPATS, OMPA) until the library insertion is completed and taped or backed-up.

(* end module delman.construction.data.entry *)

SEQUENCE INSERTION PROCEDURE

The following procedure assures the accurate and complete insertion of sequences into a Delila Library. Overview of the method:

```
                   REFERENCE OBTAINED
                          :
     .....................*....................
     :                    :                   :
     V                    V                   V
     :                    :                   :
RAW SEQUENCE         RAW SEQUENCE       DESIGN BOOK
   COPY 1               COPY 2                :
     :                    :                   :
     V                    V                   :
     :                    :                   :
  CHACHA               CHACHA                 :
     :                    :                   :
     V                    V                   :
     :                    :                   :
     :.......MERGE........:                   :
               :                              :
               V                              :
               :                              :
          RAW SEQUENCE                        :
          CORRECTED COPY                      :
               :                              :
               V                              V
               :............MAKEBK............:
                              :
                              V
                              :
                   LIBRARY INSERTION MODULE
                              :
                              V
                              :
                       LIBRARY INSERTION
```

I. Obtaining Sequences

   A. Sequences may be obtained from
      1) Publications and preprints
      2) Computer transfer
      3) Your lab

   B. One copy of the source article and the sequence (or two copies of the sequence when no paper is available) are to be made for entry to our reference shelf. The photocopies must be of GOOD quality, with NO loss of information.

II. Raw Sequence Insertion (See DELMAN.CONSTRUCTION.DATA.ENTRY for details)

   A. Double entry is preferred over other methods.
   B. Programs are available to make this easy: REFORM and MERGE.

RAWBK may be used on the checked raw sequence to get results quickly.

C. THE NAME OF THE GAME IS ACCURACY.

III. Book Design

A. First be sure that you understand library structure and coordinate systems. See LIBDEF and DELMAN.USE.

B. Use forms to write out inserted sections. These can be found in the sections that begin with "DELMAN.CONSTRUCTION.FORM".

C. Check the library to see if you can fuse the new sequence to previous sequence.

D. Decide on a coordinate system or fuse to previously defined coordinates. (NOTE: when there is no zero, add 1 to the negative numbers.) Write this information on the source copy for our reference shelf.

E. Record the source of all fragments and special information (eg: no zero, negative numbers incremented) in the PIECE notes. Put a complete reference into the PIECE notes. Include the positions on the coordinate system, such as: (-1288 to -208)

F. Record all MARKERs, TRANSCRIPTs and GENEs in your coordinates. Unknown values are either +100000 or -100000, depending on which end of the coordinates the value is beyond.

G. Create the Library insertion module using MAKEBK. All MARKERs, TRANSCRIPTs and GENEs pointing to a PIECE must be placed immediately prior to the PIECE that they refer to. They are called the "family" of the PIECE. (Note: we call this piece of a Delila library a module, but this is not the same as the ones the Module program works with. The meaning should be clear from the context.)

IV. Insertion - With The Utmost Of Care

A. Always insert whole Library insertion modules. Replace old parts of the library by modifying a module and reinserting it (with an editor).

B. Quickly check the book structure for blatant errors.

V. Checking the new Library

A. The catalogue program (CATAL) is used to check library structure and to generate human and librarian catalogues.

B. Modules that contain only parts of books can be made into whole books by placing a shell around the module. Example: a PIECE and its family can be inserted into a shell of a fake ORGANISM and CHROMOSOME to check the PIECE structure.

C. Correct modules are inserted into the library and CATAL is run on the entire library. Be sure that file CATALP is empty, to ensure that the dates are advanced.

D. End point checking: all coordinate numbers should be checked. To do this, use DELILA to pull out: COORDINATE, PIECE, GENE, TRANSCRIPT and MARKER endpoints. This is painful, but it has caught many errors. Example:

```
      GET FROM GENE BEGINNING TO GENE BEGINNING +2;
```

should give mostly ATG, and a few XTG. (SOMEDAY THIS MAY BE AUTOMATED)

VI. Listings Of The New Library

These are often useful (program to use in parenthesis)

A. LIB (SHIFT)

B. HUMCAT (CATAL)

C. REF (REFER)

```
     D. LIS (LISTER)  may be large.                                                    83

(* end module delman.construction.library.design *)
```

NAME: LIBDEF, 1980 JUNE 9

ORGANISM

* SHORT NAME

* LONG NAME

NOTE

*

*

*

*

NOTE

* GENETIC MAP UNITS (REAL)

(INSERT A SERIES OF
ORGANISMS AT THIS
POINT)

ORGANISM

(* end module delman.construction.form.organism *)

```
                   NAME:                     LIBDEF, 1980 JUNE 9


CHROMOSOME

*                                            SHORT NAME

*                                            LONG NAME

NOTE

*

*

*

*

NOTE

*                                            GENETIC MAP BEGINNING (REAL)

*                                            GENETIC MAP ENDING (REAL)







(INSERT A SERIES OF
MARKERS, GENES, TRANSCRIPTS,
AND PIECES AT THIS POINT)






CHROMOSOME


(* end module delman.construction.form.chromosome *)
```

```
                   NAME:                        LIBDEF, 1980 JUNE 9


MARKER
*                                             SHORT NAME
*                                             LONG NAME
NOTE
*
*
*
*
NOTE
*                                             PIECE REFERENCE
*                                             GENETIC MAP BEGINNING (REAL)
*                                             DIRECTION (+/-)
*                                             BEGINNING NUCLEOTIDE (INTEGER)
*                                             ENDING NUCLEOTIDE (INTEGER)

*                                             STATE (ON/OFF)
*                                             PHENOTYPE
DNA
*
*
DNA

MARKER


(* end module delman.construction.form.marker *)
```

```
                   NAME:                      LIBDEF, 1980 JUNE 9


TRANSCRIPT
*                                             SHORT NAME
*                                             LONG NAME
NOTE
*
*
*
*
NOTE
*                                             PIECE REFERENCE
*                                             GENETIC MAP BEGINNING (REAL)
*                                             DIRECTION (+/-)
*                                             BEGINNING NUCLEOTIDE (INTEGER)
*                                             ENDING NUCLEOTIDE (INTEGER)
TRANSCRIPT


(* end module delman.construction.form.transcript *)
```

```
                   NAME:                     LIBDEF, 1980 JUNE 9


GENE
*                                            SHORT NAME
*                                            LONG NAME
NOTE
*
*
*
*
NOTE
*                                            PIECE REFERENCE
*                                            GENETIC MAP BEGINNING (REAL)
*                                            DIRECTION (+/-)
*                                            BEGINNING NUCLEOTIDE (INTEGER)
*                                            ENDING NUCLEOTIDE (INTEGER)
GENE


(* end module delman.construction.form.gene *)
```

```
                   NAME:                     LIBDEF, 1980 JUNE 9


PIECE
*                                            SHORT NAME
*                                            LONG NAME
NOTE
*                                            (NOTES INCLUDE PRECISE REFERENCE
*                                            FOR EVERY BASE IN THE PIECE)
*
*
NOTE
*                                            GENETIC MAP BEGINNING (REAL)
*                                            COORDINATE CONFIGURATION
                                                (CIRCULAR/LINEAR)
*                                            COORDINATE DIRECTION (+/-)
*                                            COORDINATE BEGINNING (INTEGER)
*                                            COORDINATE ENDING (INTEGER)

*                                            PIECE CONFIGURATION
                                                (CIRCULAR/LINEAR)
*                                            PIECE DIRECTION (+/-)
*                                            PIECE BEGINNING (INTEGER)
*                                            PIECE ENDING (INTEGER)
DNA
* (INSERT SEQUENCE HERE)
DNA
PIECE
```

(* end module delman.construction.form.piece *)

```
DDDDDDD   EEEEEEEE   SSSSSS    CCCCCC   RRRRRRR   IIIIIIII  BBBBBBB   EEEEEEEE
DD    DD  EE        SS    SS  CC    CC  RR    RR     II     BB    BB  EE
DD    DD  EE        SS        CC        RR    RR     II     BB    BB  EE
DD    DD  EEEE       SSSSSS   CC        RR    RR     II     BBBBBBB   EEEE
DD    DD  EE              SS  CC        RR    RR     II     BB    BB  EE
DD    DD  EE              SS  CC        RRRRRRR      II     BB    BB  EE
DD    DD  EE              SS  CC        RR  RR       II     BB    BB  EE
DD    DD  EE        SS    SS  CC    CC  RR   RR      II     BB    BB  EE
DDDDDDD   EEEEEEEE   SSSSSS    CCCCCC   RR    RR  IIIIIIII  BBBBBBB   EEEEEEEE
```

(* end module delman.describe *)

PROGRAM NAMING CONVENTIONS

Every Delila System program exists in several forms:

1) Raw source code - without modules inserted. Example: "lister.r" would be the raw code for the LISTER program. We are not sending code this way.

2) Pascal source code - with all modules inserted. This code is ready to compile. Example: "lister.p". (Our previous convention was to add an s to the end of the file name to indicate this.)

3) Compiled code. Our convention is to remove the suffix: "lister". To simplify the manual, programs are listed under the compiled code name (lister).

PARAMETER FILE NAMES

A file that controls the operation of a program is called a parameter file. For LISTER this file is LISTERP. For SPLIT it is ... SPLITP (get it? HA! HA! sorry.)

RULES FOR PARAMETER FILES

1) If the file is not empty then the file must contain values for all parameters. With few exceptions, this should reduce the number of complex rules that one must deal with.

2) Each parameter is on its own line.

3) Parameters are left justified on the line.

4) A parameter may be followed by one or more spaces and then any comment. This lets the user write reminders of what the allowed values are.

WHY CAN'T DEFAULT PARAMETER VALUES BE STATED IN THIS MANUAL?

1) If default values are changed, then the manual must also be changed. since there is no automatic mechanism to assure that these remain the same, it is likely that it will be forgotten. The manual would then be out of date.

2) The manual entry defines the program but does not enforce details of operation. It is somewhat like the LIBDEF specification.

3) It is easy to find out what the defaults are since almost every program states the values used in its listing. Running a small test takes only two minutes.

(* end module delman.describe.conventions.naming-parameters *)

PROGRAM WRITING CONVENTIONS

Program source code will always follow certain rules:

1) The first line(s) will be the Pascal PROGRAM statement.

2) The module libraries that are sources of the modules will be stated.

3) One of the global constants will be called VERSION. This number or string identifies the particular version of the source code. We change VERSION every time that we modify the source file. The program name and VERSION are written to the OUTPUT file when the program runs.

4) There will be a document module that describes the program. The module is identical to the one in this manual such as
DESCRIBE.LISTER
It follows the format defined in
DELMAN.DESCRIBE.DOCUMENTATION.PROGRAMS

5) All constants, types, variables, procedures, functions and sections of code will have comments that describe their function.

6) Interactive programs always have a HELP command.

FOR TRANSPORTATION:

1) Put non-standard features inside modules.

2) Program lines longer than 80 characters are avoided. (NB: This is ALWAYS possible in PASCAL). The FLAG program will detect any lines that are too long.

3) Reading into packed arrays is forbidden. Read into unpacked arrays and pack or transfer values.

4) The Pascal Users Manual suggests that PASCAL identifiers "must differ over their first 8 characters." There are two problems related to this. Assume that the transport is from a computer that requires N characters to differ, where N > 8 (eg. 10).

a) Transport to a computer that requires M < N may cause names like A23456789 and A2345678X to be considered identical, and compilation will be prevented.

b) Transport to a computer that recognizes M > N will detect cases where one name was written two ways, with the difference in the last characters (between N and M). The "most famous" such case was in CATAL: HUMCATLINE and HUMCATLINES were used on a computer where N = 10 and failed on computers where M > 10.

The solution in both cases is to avoid names that differ beyond 8 characters. Is somebody willing to write a program to detect this?

(* end module delman.describe.conventions.writing *)

```
     PROGRAM RUNNING CONVENTIONS

In this manual we will use a single notation to mean running a program:
     lister(book,list)
means to run the program LISTER using a file named BOOK.  The program
will produce output to file LIST.

The names BOOK and LIST are not necessarily the same as the file names
declared in the source of LISTER (LISTERS), we assume that the names
are mapped one on one.  Also, file names to the right may not be
always mentioned, to simplify the notation.  For example:
     edit(inst1)
        :
        :   (create Delila instructions in file INST1)
        :
     delila(inst1,book1,delist1)
            (run DELILA to create a book named BOOK1 and
             a Delila listing DELIST1 that shows where the errors are.
             the library and catalogue are not mentioned.)
     lister(book1,list1)
            (Run the auxiliary program LISTER.
             OUTPUT and LISTERP are not mentioned.)

The file OUTPUT will always contain messages and diagnostics intended
for the CRT screen or teletype.
The file INPUT is always used for interactive input by the programs.


To fully define the files that a program uses we will write:
     LISTER(BOOK: IN; LIST: OUT; LISTERP: IN; OUTPUT: OUT)
IN and OUT define the direction of information flow into or out of
the program.  INOUT would mean that the source file may be modified
(such as by an editor).  This is a symbolic way to represent the data
flow diagrammed in our papers (see DELMAN.INTRO.DESCRIPTION).


NOTE: The mapping of logical file name (the one the program knows) to
physical file name (the actual one the computer system uses) is
frequently done with an ASSIGN or LINK command in the job control language of
the computer.

(* end module delman.describe.conventions.running *)
```

```
     Short clustered descriptions of some Delila System files

DOCUMENTS
     AAA     Names Of Delila System Files
     chars   Character List
     delman1 Delila System Manual
     delman2 Delila System Manual, for program descriptions
     libdef  Delila Library System Definition
     moddef  Module Transfer System Definition

LIBRARIES
     humcat  Human's Catalogue For The Library
     lib1    Library 1: Bacteriophage
     lib2    Library 2: E. Coli And S. Typhimurium

DELILA INSTRUCTIONS
     train   Transcript Library Instructions
     grin    Gene Starts In Relative Form (Use Transcript Library)
     gain    Gene Starts In Absolute Form (Use Transcript Library)

SEARCH PROGRAM RULES
     genrule Finds Genes And Non-Genes
     enzrule Finds Restriction Enzyme Sites In Books

WEIGHT MATRICES FOR THE PERCEPTRON
     w101    101 Wide, Finds All Genes In Transcript Library
     w71     71 Wide, Finds All Genes In Transcript Library
     w51     51 Wide, Finds All Genes And Some Nongenes

EXAMPLES
     ex0bk   Example Book
     ex0hu   Example Catalogue For Humans
     ex0dl   Example Delila Listing
     ex0in   Example Instructions - To Create EX0BK
     ex0li   Example Listing From LISTER
     ex0lo   Example Loocat On Catalogue from EX0BK

EXAMPLE DELILA INSTRUCTIONS FOR DELMAN
     ex0in   "ex0: example"
     ex1in   "ex1: the laci gene"
     ex2in   "ex2: an absolute get"
     ex3in   "ex3: a relative get"
     ex4in   "ex4: non-coding lac leader"
     ex5in   "ex5: the region between laci and lacz"
     ex6in   "ex6: multiple specification and requests"
     ex7in   "ex7: aligned book"
     ex8in   "ex8: non-coding lac leader- via respecification"

EXAMPLES FOR TESTING THE MODULE PROGRAM
     exsin   example source in
     exmodli example modlue library
```

```
     expepin Delila Instructions For Testing Pemowe

EXAMPLES FOR TESTING THE PERCEPTRON
     exspbk  Example Sequences Positive Book
     exsnbk  Example Sequences Negative Book
     expa0   Example Pattern 0, Learn EXSPBK Vs EXSNBK With Zero Start
     expa1   Example Pattern 1, An Initial Matrix For Learning
     expa2   Example Pattern 2, Learn EXSPBK Vs EXSNBK Using EXPA1 As Start
     expan2  Result Of Patana On EXPA2
     exsebk  A Book For Searching With EXPA2

EXAMPLES FOR TESTING ENCODE PROGRAMS
     exencin Example Encode Instructions
     exencbk The Book For EXENCIN
     exencen Example Encoding Of EXENCBK

FONTS FOR BIGLET
     font    font for the biglet program
     phont   demonstration font for the biglet program

EXAMPLE PARAMETER FILES
     Often a program will have a file associated with it
that controls it and is called a parameter file.  For example, the
pbreak program uses a parameter file called pbreakp.  Many programs
have example files.  They are not listed here, but you may want
to look for them before you run the program.  An example is the xyplo
program, for which there are the files xyplop.demo, xyin.demo,
xyplop.test and xyin.test.
     As programs are modified, this section will not always be up to date.

(* end module delman.describe.short.cluster.files *)
```

```
     Short clustered descriptions of Delila System programs
     Documentation exists as describe.[name]

 MODULE LIBRARIES
   auxmod: modules for auxiliary programs
   delmod: delila module library
   doodle: pascal graphics library and preprocessor for pic under unix
   cybmod: specific module library for the cyber computer
   genmod: genbank access modules
   matmod: mathematics modules
   prgmod: programming modules for the delila system
   unixmod: specific module library for the unix operating system
   vaxmod: specific module library for the vax computer

 MODULE MANIPULATION
   module: module replacement program
   makemod: create a set of empty modules from a list of names
   makman: make manual entries from a source code
   maknam: make manual entry names
   modin: generate modularized delila instructions for absolute sites
   modlen: determine module lengths
   makemod: create a set of empty modules from a list of names
   nulldate:  modules to neutralize the date-time functions
   pbreak: breaks a file into pages at a certain trigger phrase
   show: show modules in a module library
   undel: remove references to delman in modules

 TOOLS
   biglet: text enlargement program
   calc: a calculator that propagates errors
   calico: character and line counts of a file
   cap: put capital letters inside quotes of a program
   censor: removes code from a program
   chacha: changes characters in a file
   code: find the comment density of a pascal program
   column: pull defined column from input
   concat: concatenate files together
   copy: copy one file to another file
   decat: break a file into 10 files
   decom: remove comment starts from within a comment
   difint: differences between integers
   flag: points out excessively long lines
   ll: line lengths
   lig: ligation theory
   lochas: look at characters in a file
   merge: compare two files and merge them
   nocom: remove comments
   number: add line numbers to a file
   rembla: remove blanks from ends of lines in a file
   repro:  make multiple copies of a file
   same: counts the number of lines that are identical in two files
   shell: basic outline for a program
```

```
  shift: copy one file to another file, with a blank in front of each line            97
  short: find locations of short lines in a file
  shortline: make short lines out of long lines
  split: split a wide file into printable pages
  sqz: squeeze the input file to fit into fewer characters per line
  sumfile: sum of file sizes
  test: a simple test program for Pascal
  unshi: remove first column of characters from a file
  ver: look at the version of a program
  verbop: increment the version number of a program
  vernum: print the version number of a program
  versave: save the file under the version number
  unsqz: unsqueeze the input file
  whatch: what characters are in a file?
  worcha: word changing program
  wl: wrap lines in a file
  woco: word counting program
  wordlist: lists words in a file
  ww: word wrap

TOOLS FOR TEX
  notex: remove tex and latex constructs
  ref2bib: refer to bibtex converter
  sortbibtex: sort a bibtex database
  untex: remove tex and latex constructs
  untitle: remove titles from bbl file
  unverb: remove verbatim sections from a latex file

GRAPHICS
  doodle: pascal graphics library and preprocessor for pic under unix
  domod: doodle modules
  dops: pascal graphics library and preprocessor for postscript
  dosun: pascal graphics library and preprocessor for Sun graphics
  shrink: reduce size of postscript graphics

  genhis: general histogram plotter
  genpic: convert genhis output to pic input

  xyplo: plot x, y data
  log: convert columns of data to log

  dnag: graphics of dna

LIBRARIAN
  delila: the librarian for sequence manipulation
  catal: cataloguer of delila libraries, the catalogue program
  loocat: look at a catalogue

GENBANK
  dbbk: database to delila book conversion program
  dbcat: database catalog production and sorting program.
  dbfilter: filter GenBank databases to remove unwanted entries
  dbinst: extract Delila instructions from a GenBank database
  dblo: look at the catalogue of a genbank/embl database
```

```
  dbpull: database extraction program.                                          98

AUXILIARY PROGRAMS FOR DATA BASE CONSTRUCTION
  makebk: make a book from a file of sequences.
  rawbk: make a raw sequence into a book
  reform: raw sequences reformatted

AUXILIARY PROGRAMS FOR SEQUENCE LISTING
  lister: list the sequences of pieces in a book with translation
  parse: breaks a book into its components

AUXILIARY PROGRAMS FOR ALIGNED SEQUENCES
  alist: aligned listing of a book
  gap: gaps in aligned listing of a book
  hist: make a histogram of aligned sequences.
  histan: histogram analysis.
  malign: optimal alignment of a book, based on minimum uncertainty

AUXILIARY PROGRAMS FOR ANALYSIS
  cluster: cluster indana subindexes into groups of duplicate entries
  coda: composition file to data for genhis
  comp: determine the composition of a book.
  compan: composition analysis.
  count: counts the amount of sequence in a book
  frame: evaluator of potential reading frames
  indana: analysis of an index
  index: make an alphabetic list of oligonucleotides in a book
  pemowe: peptide molecular weights
  search: search a book for strings

AUXILIARY PROGRAMS FOR HELIXES
  dotmat: dot matrices of two books
  helix: find helices between sequences in two books
  keymat: keyed-matrices for helices between two books
  matrix: dot matrices for helices between two books
  rep: records repeats between sequences in two books

  sorth: sort helix list
  instal: delila instruction alignment

AUXILIARY PROGRAMS FOR PATTERN LEARNING
  patana: pattern analysis
  patlrn: pattern learning
  patlst: lister of patlrn output.
  patser: pattern searcher
  patval: pattern evaluations of aligned sequences

AUXILIARY PROGRAMS FOR ENCODED SEQUENCES
  encfrq: encoded sequence frequency analysis
  encode: encodes a book of sequences into strings of integers
  encsum: sum of the vectors of encoded sequences

 AUXILIARY PROGRAMS FOR INFORMATION ANALYSIS
  calhnb: calculate e(hnb), var(hnb), ae(hnb), avar(hnb), e(n)
```

```
  frese: frequency table to sequ                                                   99
  palinf: find palindromes, based on information theory
  rf: calculate Rfrequency
  rseq: rsequence calculated from encoded sequences
  rsim: Rsequence simulation
  rsgra: rsequence graph
  dalvec: converts Rseq rsdata file to symvec format
  makelogo: make a graphical ‘sequence logo’ for aligned sequences
  ckhelix: check that the helix location is where one wants
  alpro: frequency and information of aligned protein sequences
  alword: frequency and information of aligned words
  dirty: calculate probabilities for dirty DNA synthesis
  sites: analyse sites from randomized sequence data base
  bkdb: convert a book to database format for the sites program
  siva: site information variance
  diana: diaucleotide analysis of an aligned book
  tri: test environment for triangle array
  digrab: diagonal grabs of diana data
  da3d: diana da file to 3d graphics
  dotsba: dots to database
  Ri: Rindividual is calculated for every site in the aligned book
  scan: scan a book with a wmatrix and generate a vector
  vfilt: vector filter
  tod: to database format for sites program
  winfo: window information curve

AUXILIARY PROGRAMS FOR OTHER USES
  refer: print the references in the pieces of a book
  sepa: separates delila instruction sets
  lenin: convert a list of lengths into Delila instructions

RANDOM NUMBERS AND SEQUENCES
  markov: markov chain generation of a dna sequence from composition.
  tstrnd: test random generator
  gentst: test random generator
  normal: generate normally distributed random numbers
  rndseq: generate random dna sequences
  aran: aligned random sequences

MATHEMATICS
  av: average integers
  binomial: produce the binomial probabilities for a found black to white ratio
  binplo: produce the binomial probabilities for a found black to white ratio
  cerf: complement of the error function
  cisq: circle to square
  chi: estimates chi squared from degrees of freedom
  linreg: linear regression
  mnomial: produce the multinomial distribution for base probabilities
  pcs: partial chi squared
  riden: ring density graph
  ring: z space ring
  sphere: plot density of shannon spheres
  stirling: test of stirling’s formula
  zipf: Monte Carlo simulation for Peter Shenkin’s problem
```

```
 MISCELLANEOUS
   aa: not actually a program, this is the header page for Delila manual
   asciicode: converts ascii table to Pascal code
   binhex: convert binary to hex
   hexbin: convert hex to binary
   mstrip: remove control m's from a file
   epsclean: clean an eps file

   kenin: create Delila instructions from Kenn's all.gen instructions
   kenbk: book from a file of sequences of sequences provided by Kenn Rudd

   tipper: copy a file to the output file with special symbols at end
   todawg: change a book into dawg format

   ev: evolution of binding sites
   evd: evolution display

   makedate: make a date file
   makessbdate: make a date file from a Sample_Sheet.bin file

 PROGRAMS TO CONTROL MACHINERY
   odti: munch od and time plates together for xyplo
   titer: analyse titertek optical density data
   spec: analyse two spectra from the camspec
   ssbread: read a sample sheet from the ABI sequencer
   tkod: read od values from tk data

(* end module delman.describe.short.cluster.programs *)
% makman 1.32
```

```
ddddddd   eeeeeeee  ll        m      m     aa     n     nn
dd    dd  ee        ll        mm    mm    aaaa    nn    nn
dd    dd  ee        ll        mmm  mmm   aa  aa   nnn   nn
dd    dd  eeeeeee   ll        mmmmmmmm  aa    aa  nnnn  nn
dd    dd  ee        ll        mm mm mm  aa    aa  nn nn nn
dd    dd  ee        ll        mm    mm  aaaaaaaa  nn  nnnn
dd    dd  ee        ll        mm    mm  aa    aa  nn   nnn
dd    dd  ee        ll        mm    mm  aa    aa  nn    nn
ddddddd   eeeeeeee  llllllll  mm    mm  aa    aa  nn    nn


                     222222
                    22    22
                          22
                       2222
                      22
                     22
                    22
                    22
                    22222222
```

Note: this page is kept in file aa.p on our UNIX system to make it easy to make a manual of all the program documentation with this as the first page.

This is done by concatenating all the program source codes together and running this through makman and pbreak:

```
cat *.p | makman | pbreak > delman2.print &
```

If your version of pbreak does not add blanks in front of the lines of delman2.print, you can run delman2.print through the program maknam to create a short listing of what each program does.

(* end module describe.delman2 *)
version = 4.07 of aa.p delman2 1993 Jan 27 Schneider-Stormo

```
(* begin module describe.documentation.programs *)                              102
<(*>
<name>
      program name<:> a one-line description of the program.
                      See description (below) for more details.

<synopsis>
      name<(>file1<: >i/o<, >file2<: >i/o<, >file3<: >i/o<, >...<)>
      This is the program statement with each file name followed
      by the input/output (i/o) use of the file:
         in     the file is used strictly for input (read-only)
           out  the file is used strictly for output (write-only)
         inout  the file is used for both input and output (read/write)
         intty  the file is used for interactive input (teletype)

<files>
      file1<: > multiple line detailed description of file 1
      file2<: > multiple line detailed description of file 2
      file3<: > multiple line detailed description of file 3
      ...

<description>
      The purpose and use of the program.
      All programs in the delila system are documented in the form shown on
      this page.
      <...>  indicates a literal, you must include it.
      <...>* these sections are optional, others are obligatory.
      This rigid style model will encourage uniformity and help the reader
      to know where to look.
         Note:  the description should be in flowing language, to introduce the
      program to people and make them interested in using it.
         Warning: do not make any describe module longer than 60 lines
      or it will not fit as a page in delman.

<examples>*
      An example of the use of this form is module describe.lister

<documentation>*
      Other sources of information or documents on the program.

<see also>*
      Other programs and related programs.

<author>
      One should be proud of one's work, and one should be
      responsible for it.

<bugs>
      problems with the program and how to get around them (if known).
      Since in many cases, no bugs are known, this section is intended to
      include bugs in the design of the program.  How might the program be
      written better if one were to start again from scratch?

<technical notes>*
```

```
      Details about the implementation that may be relevant to a user.              103
      These notes are not, repeat not, to contain values of constants,
      since these may change (use the name of the constant).

<*)>
(* end module describe.documentation.programs *)
version = 1.00 of describe.documentation.programs
```

```
(*
name
      alist: aligned listing of a book

synopsis
      alist(inst: in, book: in, alistp: in, colors: in, namebook: in,
            list: out, clist: out, output: out)

files
      inst: delila instructions of the form 'get from 56 -5 to 56 +10;'
         (This file may be empty, in which case the sequences will be
         aligned by their 5' ends.)
      book: the book generated by delila using inst
      alistp: parameters to control the program.  If empty, the range of the
         instructions are used.  Otherwise,
         1. The first line contains one line with two integers
         defining the range to display.  This allows one to have a wide
         alignment, but look only at a portion.
         2. If the first character of the second line is 'p' the piece
         information is given in the list.
         3. If the first character of the third line is 'n' then paging
         is not done to the list.
      namebook: names of genes or transcripts from this book appear in
         the list.  If namebook is empty, then only the items specified in
         alistp are given.
      list: the aligned listing
      clist: the aligned listing, in PostScript color
      colors: colors defining the bases, see makelogo for definition.
      output: messages to the user

description
      Alist is useful for looking at aligned sets of sequences.
      The pieces in the book are aligned according to the instructions in
      file inst, and listed in the list file.  Each piece is identified, and
      a bar of numbers (called a 'numbar') that are read vertically defines
      the locations of bases around the aligning point.

example
      To generate the input set, start with a set of instructions that name
      genes and get them (as 'get from gene beginning -0 to gene beginning
      +2;').  Produce namebook.  Check for genes that are reversed relative
      to the piece (use hist and alist without instructions), and correct
      the delila instructions.  To convert these instructions to absolute
      form, use program search with 'd f -54321 t +12345 q atg gtg ttg' on
      namebook.  Now convert -54321 and +12345 to the range of interest
      (beware of absolute locations with the same numbers).  Finally,
      generate the book using delila.  (Someday this process will be simpler.)

documentation
      delman.use.aligned.books

author
      Thomas D. Schneider
```

```
bugs
      If you use relative instructions, then alist will bomb.
      Ie, do not use instructions of the form:
          get from gene beginning - 5 to gene beginning +5;

      Alist is not very smart about how it finds the instructions.  It uses
      the first letter of the line to find the instruction 'get'.
      Unfortunately, if the word 'gene' is found, alist does not know this
      and will bomb.  Simply add blanks infront of the word 'gene' if you
      want to keep the gene instruction.

      There is also an unsolved bug in alist:
      When the pieces and instructions are not 'just right', alist will
      produce listings that are thousands of characters wide...  The reason
      for this is not completely clear, but it is related to attempting
      to extend the from-to range of an aligned book, and perhaps to incorrect
      responses of delila when attempting to 'reduce' a piece beginning or
      ending that is off the end of a fragment of a circular piece.  The code
      now contains traps that halt the program when wide listings would have
      been generated.

technical notes
      variable nametype defines the kind of name picked up in namebook.

*)
(* end module describe.alist *)
version = 4.64; (* of alist.p 1993 January 26
```

```
(*
name
   alpro: frequency and information of aligned protein sequences

synopsis
   alpro(protseq: in, symvec: out, output: out)

files
   protseq:  Aligned protein sequences.  The first line, intended for
      identification of the entire data set, is skipped.  The header line must
      begin with an asterisk '*'.  The remaining lines are used for the
      sequences.  They are divided into `entries'.  The beginning of an entry
      has any (positive) number of identification lines, each of which begins
      with an asterisk '*'.  The sequence follows.  Gaps are indicated with
      dashes (-).  The end of the sequence is indicated by a period.
   symvec:  table of frequencies and information content.  The information
      measure is corrected for small sample size (Schneider et al, 1986).

   output: messages to the user

description
   Take an aligned set of protein sequences and produce input to the
   makelogo program for producing a logo.

   The program originally only created a vector that contained the characters
   of the alphabet, so the output was called an 'alvec'.  To reflect the use of
   symbols, the name of the output file was changed to symvec, but I like
   'alpro', and 'prosym' is awkward that I decided to keep the name alpro.

examples

* This is an example sequence.
AG-EGCTT.
* This is the second example sequence.
* It is the last one.
YLREBS-A.

documentation
   Jotun Hein, Methods of Enzymology 183:626-645 (1990)
   Schneider et al. JMB 188:415 (1986)

@article{Schneider.Stephens.Logo,
author = "T. D. Schneider
 and R. M. Stephens",
title = "Sequence Logos: A New Way to Display Consensus Sequences",
journal = "Nucl. Acids Res.",
volume = "18",
pages = "6097-6100",
year = "1990"}

see also
   makelogo.p
```

```
author
   Thomas D. Schneider
   National Cancer Institute
   Laboratory of Mathematical Biology
   Frederick, Maryland  21702-1201
   toms@alum.mit.edu

bugs

technical notes
   The feature which adjusts the stack height when there is a small amounts of
   data, (described in the second paragraph of page 6100 of the logo paper),
   has been removed now because the ability to display the variance as a
   standard deviation by makelogo alerts the person that the position has
   little data in it.  Thanks to Peter Shenkin for the suggestion.

   The original feature was described as follows:

      "Positions that contain mostly spacer characters for the alignment are
      also reduced in weight by multiplying the information by the maximum
      number of sequences and dividing it by the actual number at the spacer
      position.  Thus if there are 10,000 sequences, a position with 200 A's
      would would be close to 2 bits of pattern.  However, since the position
      only represents 2% of the sequences, this program would only give it a
      weight of 0.02*2 = 0.04 bits.  A better method is not known.  However,
      this prevents one from being fooled by positions that don't appear in
      most sequences."

*)
(* end module describe.alpro *)
version = 1.52; (* of alpro.p 1992 March 6
```

```
(*
name
   alword: frequency and information of aligned words

synopsis
   alword(words: in, symvec: out, output: out)

files
   words:  Aligned words.  Since the input is usually to be a UNIX dictionary,
      there need not be any header lines.  However, if they exist, they must
      begin with an asterisk '*'.

      The remaining lines are used for the words.

   alwordp: parameters to control the program.  If the file is empty
      defaults are used.
      If the first line begins with the letter `e' then the words are
      aligned by their last character.
      If there is a first line, the second line must have the maximum
      word length to be included in the calculation.  Words longer than
      this will be skipped (and reported to output).

      If the first character of the second line is 'a' then all of the
      words in the file will be read.  Otherwise, only the first
      word on each line will be read.
   symvec:  table of frequencies and information content.  The information
      measure is corrected for small sample size (Schneider et al, 1986).
   output: messages to the user

description
   Take an aligned set of protein sequences and produce input to the
   consensus program for producing a logo.

examples

* This is an example sequence.
AGGEGCTT.
* This is the second example sequence.
* It is the last one.
YLREBS.

documentation
   Jotun Hein, Methods of Enzymology 183 (1990)
   Schneider et al. JMB 188:415 (1986)

see also
   alpro.p, makelogo.p

author
   Thomas Dana Schneider

bugs
```

```
*)
(* end module describe.alword *)
version = 2.07; (* of alword.p 1992 June 4
```

```
(* begin module describe.aran *)
(*
name
   aran: aligned random sequences

synopsis
   aran(book: in, aranp: in, list: out, sequ: out, output: out)

files
   book: the book generated by Delila

   aranp:  Parameters to control the program.
      The FIRST LINE must contain one real number which is the degree of
      conservation.  For example, if this is 0.85, then each base will have 85%
      chance of being the same, while the other bases will be 5% each.

      The SECOND LINE must contain the number of sequences to generate.

   list: details of the run.

   sequ: the aligned sequences, for input to makebk

   output: messages to the user

description
   Aran takes a sequence as a starting point and generates random
   sequences from it.  The program simulates a very simple dirty synthesis of
   the sequence.  The synthesis is to be mostly the bases given in the
   sequence.  The probability of conserving each base (f) is defined in the
   parameter file.  If a particular base is not conserved, then the other
   three bases are assigned probabilities of (1-f)/3.

example
   See alist

documentation
   delman.use.aligned.books

author
   Thomas D. Schneider

bugs
   See alist

technical notes
   The program constant seqmax defines the length of the longest sequence
   that can be created.

*)
(* end module describe.aran *)
version = 1.15; (* of aran.p 1990 Oct 3
```

```
(* begin module describe.asciicode *)
(*
name
   asciicode: converts ascii table to Pascal code

synopsis
   asciicode(ascii: in, code: out, output: out)

files
   ascii:  The ascii file must contain this table:

|  0 NUL|  1 SOH|  2 STX|  3 ETX|  4 EOT|  5 ENQ|  6 ACK|  7 BEL
|  8 BS |  9 HT | 10 NL | 11 VT | 12 NP | 13 CR | 14 SO | 15 SI
| 16 DLE| 17 DC1| 18 DC2| 19 DC3| 20 DC4| 21 NAK| 22 SYN| 23 ETB
| 24 CAN| 25 EM | 26 SUB| 27 ESC| 28 FS | 29 GS | 30 RS | 31 US
| 32 SP | 33  ! | 34  " | 35  # | 36  $ | 37  % | 38  & | 39  '
| 40  ( | 41  ) | 42  * | 43  + | 44  , | 45  - | 46  . | 47  /
| 48  0 | 49  1 | 50  2 | 51  3 | 52  4 | 53  5 | 54  6 | 55  7
| 56  8 | 57  9 | 58  : | 59  ; | 60  < | 61  = | 62  > | 63  ?
| 64  @ | 65  A | 66  B | 67  C | 68  D | 69  E | 70  F | 71  G
| 72  H | 73  I | 74  J | 75  K | 76  L | 77  M | 78  N | 79  O
| 80  P | 81  Q | 82  R | 83  S | 84  T | 85  U | 86  V | 87  W
| 88  X | 89  Y | 90  Z | 91  [ | 92  \ | 93  ] | 94  ^ | 95  _
| 96  ` | 97  a | 98  b | 99  c |100  d |101  e |102  f |103  g
|104  h |105  i |106  j |107  k |108  l |109  m |110  n |111  o
|112  p |113  q |114  r |115  s |116  t |117  u |118  v |119  w
|120  x |121  y |122  z |123  { |124  | |125  } |126  ~ |127 DEL

   code:  Pascal code that converts integers to these names.
   output: messages to the user

description

   This program generates a chunk of Pascal code that is useful
   for detailed investigation of file characters.

examples

documentation

see also
   lochas.p

author
   Thomas Dana Schneider

bugs

technical notes

*)
(* end module describe.asciicode *)
version = 1.01; (* of asciicode.p 1993 January 26
```

```
(* begin module describe.auxmod *)
(*
name
      auxmod: modules for auxiliary programs

synopsis
      auxmod(hst: in, cmp: in, patt: in, output: out)

files
      hst: a histogram from hist for testing, or empty
      cmp: a composition from comp for testing, or empty
      patt: a pattern matrix from patlrn for testing, or empty
      output: the version of auxmod is printed.  test results are printed.
         successful compilation and running of the program indicates that
         the modules are correct.

description
      auxmod is a collection of modules used only rarely in various
      auxiliary programs.  it includes modules for reading compositions
      (comp.), histograms (hist.), helix lists (findcolon and
      gethelix) and pattern matrices (matrix.).

see also
      delmod, module, hist, comp, patlrn

author
      gary d. stormo and thomas d. schneider

bugs
      none known

*)
(* end module describe.auxmod *)
version = 'auxmod 1.39 86 dec 12 gds/tds';
```

```
(* begin module describe.av *)
(*
name
   av: average integers

synopsis
   av(input: in, output: out)

files
   input:  give pairs of integers
   output: rounded average of the integers

description
   Genbank features are given as endpoints; we need to convert
to the central base for delila instructions.  This program lets one
do that.  The program rounds the result.

examples

documentation

see also

author
   Thomas Dana Schneider

bugs

technical notes

*)
(* end module describe.av *)
version = 1.03; (* of av.p 1992 Jun 2
```

```
(* begin module describe.biglet *)
(*
name
   biglet: text enlargement program

synopsis
   biglet( fin: in, font: in, bigletp: in, fout: out, output: out )

files
   fin: contains user's text to be enlarged.
   font: the first line contains the actual height and width of characters
         in the font.  The following lines contain character images.  A
         character image has two parts, a reference character and the
         letter image.  Characters in the image that match the reference
         character are printed, while a mismatch prints a space.
   bigletp: contains parameters to control enlargement.  If the file is
         empty the fonts are not enlarged. otherwise, each line
         contains the height and width enlargement factors.  The line
         may also contain a character inside quote marks (single or
         double) to substitute for the matched characters of the font
         images.  Each line of bigletp corresponds to a fin text line.
         If there are no further lines, previously set values are used.
   fout: each line of fin is expanded by bigletp parameters and printed
         out in the form of the font images.
   output: messages to the user.

description
   Each letter of text (in file fin) is expanded and printed as a larger
   letter which is composed of many smaller letters.  The expansion can be
   set for each text line or for all lines with one parameter setting.
   There is an optional parameter which allows all the large letters of a
   specified line to be composed of a single character.  The larger letters
   are based on a file called font which can contain any sort of images.

examples
   For a font file whose first line is a left justified 5 4:
   f (sixth letter)     (a space)         - (a dash)
   fff-              ----                 xxxx     Note: in the file each
   f---              ----                 xxxx     character image must be
   fff-              ----                 ---x     left justified and be
   f---              ----                 xxxx     directly below the
   ----              ----                 xxxx     previous image.
   Also, each image has mismatches at its right and below used for spacing.
   for bigletp:          example 1)   2 1         example 2)   3 2 'r'
                                                               1 2 'w'
   The first example magnifies the first and all subsequent text lines
   twice in height.  The second example magnifies the first line at 3 by 2
   and composes it out of 'r's.  The next line will be twice as wide as the
   font and composed of 'w's.  All subsequent fout text will be also be
   twice as wide but made up of the usual font characters.

   The phont file is a demonstration font file, while the font file
   is a working font.
```

```
author
   Matthew A. Yarus

bugs
   none known

technical notes
   If your font images are larger than program allows change constants
   letmaxhi and letmaxwi in biglet source code.
*)
(* end module describe.biglet *)
version = 1.65; (* of biglet 1986 dec 15
```

```
(* begin module describe.binhex *)
(*
name
   binhex: convert binary to hex

synopsis
   binhex(input: in, output: out)

files
   input: binary representation of an image, from binhex
   output:  hexadecimal representation of an image, PostScript
   shape: First line contains two characters to skip and then
      two integers, the width and height of the image.

description
   To allow one to work with a PostScript hex image in binary format
   it is converted.

examples

documentation
   PostScript red book p. 170

see also

author
   Thomas Dana Schneider

bugs

technical notes

*)
(* end module describe.binhex *)
version = 1.08; (* of binhex.p 1991 October 17
```

```
(*
name
   binomial: produce the binomial probabilities for a found black to white ratio

synopsis
   binomial(xyin: out, xyplop: out, binomialp: in, output: out)

files
   xyin: a table of probabilities of finding the given black to white
      ratio, versus the true probability.  The form is a series of lines
      that begin with '* ', followed by two columns of numbers.
      The first column is the number of blacks, and the second column is
      the corresponding value of p(black:white|pb) = the probability of
      obtaining black and white given pb, the probability of black.
      This file is direct input to the xyplo program.
   xyplop: the controls for the xyplo program to generate the graph.
      These may be modified by the user before plotting.
   binomialp: parameters to control the program, on three lines:
      blacks and whites: two integers on the first line, representing the
         number of black balls and white balls obtained in an experiment
      probability of black
      plot max: maximum number of blacks to show.

description
   Suppose there exists a large bin containing both black and white
   balls.  The true fraction of black balls in the bin is fraction, and
   the fraction of white balls is (1-fraction).  We obtain a sample of
   black and white balls from the bin, given as the first two parameters
   in binomialp.  The probability of getting this black:white sample is:

                                 (black+white)!        black            white
       p(black:white|fraction) = -------------- fraction      (1-fraction)
                                 black!white!

   The program generates these probabilities for a given fraction.
   The results are in a form that the xyplo program can use to plot.

see also
   xyplo, binplo

author
   Thomas Dana Schneider

bugs
   none known

*)
(* end module describe.binomial *)
version = 1.42; (* of binomial, 1988 feb 24 *)
```

```
(*
name
   binplo: produce the binomial probabilities for a found black to white ratio

synopsis
   binplo(xyin: out, xyplop: out, binplop: in, output: out)

files
   xyin: a table of probabilities of finding the given black to white
      ratio, versus the true probability.  The form is a series of lines
      that begin with '* ', followed by two columns of numbers.
      The first column is the value of fraction, and the second column is
      the corresponding value of p(black:white|fraction) = the probability of
      obtaining black and white given fraction.  This file is direct input
      to the xyplo program.
   xyplop: the controls for the xyplo program to generate the graph.
      These may be modified by the user before plotting.
   binplop: parameters to control the program
      blacks and whites: two integers on the first line, representing the
         number of black balls and white balls obtained in an experiment
      points: one integer on the second line, how many data points should
         be generated in the fout.  If points is zero, then the program
         tests its binomial probability procedure by adding all the
         probabilities that correspond to the binomial distribution.
         For example, with 1 black and 18 white balls, the test is to
         add the probabilities for (0,19), (1,18), ... (19,0).  This
         value should be close to 1.00 if the procedure is correct.

description
   Suppose there exists a large bin containing both black and white
   balls.  The true fraction of black balls in the bin is fraction, and
   the fraction of white balls is (1-fraction).  We obtain a sample of
   black and white balls from the bin, given as the first two parameters
   in binplop.  The probability of getting this black:white sample is:

                               (black+white)!        black             white
      p(black:white|fraction) = -------------- fraction      (1-fraction)
                               black!white!

   the program generates these probabilities for all values of fraction,
   and gives the results in a form that the xyplo program can use to plot.

see also
   xyplo

author
   Thomas Dana Schneider

bugs
   none known

*)
(* end module describe.binplo *)
```

```
(* begin module describe.bkdb *)                                                          120
(*
name
   bkdb: convert a book to database format for the sites program

synopsis
   bkdb(book: in, database: out, output: out)

files
   book: a book containing many sequences of the same size.
   database: the format used by the sites program.
   output: messages to the user

description
   The program converts a book to the database format used by the sites
   program.

examples

documentation

see also
   sites.p

author
   Thomas Dana Schneider

bugs
  It sure would be nice to have on uniform type of format, but the GenBank
  format is not yet defined (and it is 5 years after GenBank was told
  by a national advisor to do this!), so we wait.

technical notes

*)
(* end module describe.bkdb *)
version = 1.01; (* of bkdb.p 1991 January 14
```

```
(*
name
   calc: a calculator that propagates errors

synopsis
   calc(input: in, output: out)

files
   input: reverse polish calculator input
   output: results

description
   The program is based on the idea of the dc program under UNIX.  That
   program takes input as reverse polish and calculates values.  This
   program does the same, but values have estimates so one may calculate and
   propagate errors.  Tokens (commands and numbers) are usually separated by
   spaces or carriage returns.  Tokens that begin with a digit or a dash (-)
   are numbers.  Numbers always come in pairs, the first is the estimate and
   the second is the error.

   Some of the commands are:
     h give current list of all commands and functions
     numbers (as pairs) are entered on the stack
             5 2
          means 5 +/- 2
             5'
          means 5 +/- 0, so you can avoid giving the estimate if you want.
          any other legal command may replace the single quote as "5p".
     + add the top two numbers on the stack together
     _ (UNDERSCORE) subtract the top number from the next number
       on the stack (underscore is used to be distinct from minus sign, -)
     * multiply the top two numbers on the stack together
     / divide the top number on the stack by the next number on the stack
     s print the stack, top down
     p print the top number on the stack

   Note:  When the program is asked to do calculations silently,
   (using the t command) it immediately shuts up and does not say that it
   is doing so.  This makes it easier to write programs without having them
   announce in the output that they are doing silent calculations.

documentation
   An Introduction to Error Analysis,  John R. Taylor
   University Science Books, Mill Valley, CA. 1982.

author
   Thomas Schneider

bugs
   Pascal numeric input is used, so anything that can make Pascal
   bomb will bomb this program.  For example, "- ", will cause
   the program to think there is a number after the dash, and (our)
   Pascal will object.  This should be protected against now,
```

```
   so the program should never bomb (famous last words).

   The u (uncertainty) function error estimate is set to zero when
   the probability is zero.  This is a guess.

*)
(* end module describe.calc *)
version = 2.44; (* of calc.p 1992 September 3
```

```
(* begin module describe.calhnb *)
(*
name
      calhnb: calculate e(hnb), var(hnb), ae(hnb), avar(hnb), e(n)

synopsis
      calhnb(fin: in, fout: out, output: out)

files
      fin: the genomic composition (integers) on one line followed by
         a set of integers, one per line representing values of n
      fout: a table showing n, e(hnb), ae(hnb) and their difference.
         the variances var(hnb) and avar(hnb) are tabulated along with
         the difference between their square roots.  this is the difference
         between the standard deviations.  e(n) is found from the genomic
         entropy minus e(hnb).
      output: messages to the user.

describe
      given a genomic composition and a series of integers (n) that
      represent the number of sample sites, calhnb calculates the sampling
      error as e(hnb) and the variance var(hnb).  it also finds the
      approximations ae(hnb) and avar(hnb).  these values are presented in a
      table along with the differences between the exact and approximate
      calculations.  this table will allow a user to decide when to use the
      approximations.  beware that the exact calculation becomes very expensive
      for large n.

documentation
      "Information content of binding sites on nucleotide sequences"
      T. D. Schneider, G. D. Stormo, L. Gold, and A. Ehrenfeucht
      JMB 188:415-431 (1986)

see also
      rseq

author
      thomas d. schneider

bugs
      none known
*)
(* end module describe.calhnb *)
version = 2.21; (* of calhnb 1988 feb 24
```

```
(* begin module describe.calico *)
(*
name
      calico: character and line counts of a file

synopsis
      calico(input: in, output: out);

files
      input: a file for which one wants to know the number of characters
         and lines
      output: the number of characters and lines in input

description
      there are many circumstances when one would like to know the number of
      characters and the number of lines in a file.

examples
      will a file fit on one page?  can this file be put into the
      memory of a personal computer for transportation to another computer?

author
      susan p. scolman and thomas d. schneider

bugs
      none known

technical notes
      blanks at ends of lines are counted as characters.
      only the end of line mark is counted, not carriage return and line feed.

*)
(* end module describe.calico *)
version = 1.08; (* of calico.p 1993 January 27
```

```
(* begin module describe.cap *)
(*
name
      cap: put capital letters inside quotes of a program

synopsis
      cap(sin: in, sout: out,output: out)

files
      sin: the source program or file
      sout: the source program with capital letters in all
         quote strings.
      output: messages to the user

description
      A pascal program under Unix must be small characters, yet a
      database will often be in capital letters, so the program will
      not recognize the data.  This program makes the sin program
      have capital letters only in the quote strings.

author
      thomas d. schneider

bugs
      none known

*)
(* end module describe.cap *)
version = 1.08; (* of cap.p 1989 July 8 *)
```

```
(*
name
      catal: cataloguer of delila libraries, the catalogue program

synopsis
      catal(humcat: out, catalp: in,
            l1: in, cat1: out, lib1: out,
            l2: in, cat2: out, lib2: out,
            l3: in, cat3: out, lib3: out,
            output: outt)

files
      humcat: the catalogue generated for humans.  it includes the names
         of things in the libraries and their coordinates.  humcat is quite
         wide so you will need a line-printer to print it.  alternatively
         you can use the split program.
      catalp: a parameter to control the program.  the library
         dates are not changed if the first character is 'n' (no date
         modification) or 'b' (book source of library, dates are not to
         be changed).  otherwise the dates are advanced.
      l1: the first input file of the library
      cat1: the first catalogue
      lib1: the first output library
      l2: the second input file of the library
      cat2: the second catalogue
      lib2: the second output library
      l3: the third input file of the library
      cat3: the third catalogue
      lib3: the third output library
      output: progress report and error messages

description
      the catalogue program checks all the input libraries for correct
      structure.  duplicated names are removed and a new set of library
      files is created, along with their catalogues for delila.  a catalogue
      is also generated for people to use.  each new library is associated with
      one catalogue.  under most circumstances this pair can be given to
      delila along with pairs created at different times.

documentation
      libdef (defines catal), delman.use.coordinates, delman.construction

see also
      loocat, delila, split

author
      Michael Aden and Thomas Schneider

bugs
      not all checks on the library structure are made.  some checks from
      libdef are now outdated or not done: p. 3.1 2 d, e, f, g and l.

technical notes
```

```
     the circumstances when a library-catalogue pair must not be used with
     another pair:  it is not possible for delila to check for two
     organisms with the same name that exist in different libraries.  in
     this case, run the two libraries through catal together to eliminate
     the ambiguity.  if this is not done, the results will be anomalous.

*)
(* end module describe.catal *)
version = 9.23; (* of catal.p 1992 September 14 *)
```

```
(*
name
   censor: removes code from a program

synopsis
   censor(input: in, output: out)

files
   input:  input program with private text
   output: output program without private text

description
   The program allows one to maintain a Pascal program for personal use which
   contains features that are not yet to be made public.  The program contains
   special comment marks that delimit the text to be removed.  There are two
   situations.

   The first is the case of sections of text inside comments.  Any text
   surrounded by [[ and ]] will not be copied to the output.  This includes the
   double brackets themselves.

   The second case is sections of normal code.  Letting '@' represent the
   asterisk (so that this description does not run into trouble when it
   is inside a Pascal comment), the text between and including the symbols
   (@[[@) and (@]]@) is not copied to the output.

examples

documentation

see also

author
   Thomas Dana Schneider

bugs

technical notes

*)
(* end module describe.censor *)
version = 1.46; (* of censor.p 1991 February 20
```

```
(*
name
   cerf: complement of the error function

synopsis
   cerf(input: in, list: out, output: out)

files
   input:  Give the z value you want evaluated.  Enter a number less than
      zero to stop the program.
   list: the complement of the error function and the error function
   output: messages to the user

description
   The area under the Gaussian distribution is found, given values of z.
   The error function is:
      erfc(y) = (2/sqrt(pi)) * integral from y to infinity exp(-t*t) dt.

documentation
   This is program ERFD3, figure 11.7, p. 330-333 in
   Pascal Programs for Scientists and Engineers
   Alan R. Miller, Sybex, 1981

author
   Thomas Dana Schneider

bugs
   none known

technical notes
   the tolerance may be adjusted, see the constants.

*)
(* end module describe.cerf *)
version = 1.04; (* of cerf 1988 September 14
```

```
(* begin module describe.chacha *)
(*
name
      chacha: changes characters in a file

synopsis
      chacha(fin: in, fout: out, chachap: in, output: out)

files
      fin:     any file in which one wants to translate one set of
               characters into another set.
      fout:    the file to which the translated copy of fin is written.
      chachap: the chacha parameter file which contains the translation sets.
               chachap must only contain 2 lines.
               the first line contains the characters used in fin, typed one
                  right after the next with no blanks at the beginning.
               the second line contains the characters that the
                  characters in the first line are to be translated
                  into, typed in the same way and in corresponding order.
               if you want to change a character to blanks,
                  or vice versa, then you must have the blank character
                  in between other characters in chachap.
      output:  where error messages will appear.

description
      chacha translates characters in a file to a new set of characters.
      also, more than one character can be translated in one run
      of the program.

examples
      to convert between double and single quotes, use:
         '"
         "'
      to convert blanks to periods, use:
         j j
         j·j
      in the chachap file.
      each character on the first line on chachap will be translated
      into the character directly beneath it on the second line in
      the output file.

documentation
      delman.assembly.intro  and  delman.assembly.chacha

see also
      worcha

author
      patrick r. roche

bugs
      none known

technical notes
```

```
      the maximum number of characters that can be translated is
      constant top.  caution: top is also the maximum line length.

*)
(* end module describe.chacha *)
      version = 3.10; (* of chacha 1985 apr 17 *)
```

```
(*
name
   chi: estimates chi squared from degrees of freedom

synopsis
   chi(input, output);

files
   input:  degrees of freedom
   output: messages to the user

description
   estimates chi squared, given degrees of freedom.

documentation
   @book{Finberg1978,
   author = "S. Finberg",
   title = "Analysis of Cross Classified Catagorical Data",
   publisher = "MIT Press",
   address = "Cambridge, Mass?",
   year = "1978",
   comment = "from Chip Lawrence, S=Steven"}
   appendix iii

author
   Thomas Dana Schneider

bugs
   it's only an estimate

*)
(* end module describe.chi *)
version = 1.03; (* of chi 1988 July 12
```

```
(*
name
   cisq: circle to square

synopsis
   cisq(cisqp: in, xyin: out, output: out)

files
   cisqp: parameters to control the program
      First line: lowest value of m, mlo.
      Second line: highest value of m, mhi.
      Third line: increment in the value of m, mstep.
      Fourth line: desired radius of a circle if m = 2, reffective.
      Fifth line: number of steps to take to move around 360 degrees.
      Sixth line: A factor by which to increase the value of theta, spinfactor.
         1 gives a square, 1.5 gives a hexagon.
   xyin: input to the xyplo program.  Curves that are close to integer
      values of n have the symbol m, others have the symbol r.  This allow
      them to be distinguished by the graphics routines.
   output: messages to the user

description
   Plot the equation
      |x|^m + |y|^m = |reffective|^m
   where reffective is the "effective" radius of the curve, |x| is the absolute
   value of x, and ^ means to raise to the mth power.  This gives a line if m =
   1, a circle if m = 2 and approaches a square as m -> infinity!

   The method for producing the curves is to re-express the equation in polar
   coordinates.  One must be a bit careful to distinguish between the effective
   radius (reffective) and the current polar coordinate (r).  After making this
   distinction we can write:

      x = r cos theta
      y = r sin theta

   and rearrange to solve for r, while keeping reffective fixed as it should
   be.

   Dividing the basic formula by r (>0) and converting to polar coordinates
   gives:

         (reffective/r)^m := / ((|cos(theta)|)^m + (|sin(theta)|)^m);

    To do this in Pascal, we have to use the form, a^m = exp(m*ln(a)).
    This gives:

         exp(m * ln(r/reffective)) := 1 / ( exp(m * ln(abs(cos(theta))))
                                          +
                                          exp(m * ln(abs(sin(theta)))) )

    where we have also introduced the absolute function on the sine and cosine.
    One more rearrangement gives
```

```
        r := reffective * exp( ln(
                                  1 / ( exp(m * ln(abscostheta))
                                        +
                                        exp(m * ln(abssintheta)) )
                                ) / m);
```

which is the form used in the code.

In the cases where the sine or cosine are zero (ie on the axes), we must not calculate at all, to avoid log of zero. We simply set r = reffective in those cases.

The program has a special feature to speed up the angle of the calculation (theta) so that it moves faster than the angle at which the graph is plotted. With a factor of 3/2, the four corners become 3/2 * 4 = 6 corners, and we obtain a hexagon.

examples

To produce a nice square, use the parameters:

```
0.5   First line: lowest value of m.
5.0   Second line: highest value of m.
0.1   Third line: increment in the value of m
1     Fourth line: desired radius of a circle if m = 2.
100   Fifth line: number of steps to take to move around 360 degrees.
1     Sixth line: A factor by which to increase the value of theta.
         1 gives a square, 1.5 gives a hexagon.
```

To produce a hexagon transformed into a circle, use the parameters:

```
1.5   First line: lowest value of m, mlo.
2.0   Second line: highest value of m, mhi.
0.1   Third line: increment in the value of m, mstep.
1.0   Fourth line: desired radius of a circle if m = 2, reffective.
100   Fifth line: number of steps to take to move around 360 degrees.
1.5   Sixth line: A factor by which to increase the value of theta, spinfactor.
         1 gives a square, 1.5 gives a hexagon.
```

It is not clear why one has to use the lowest value of n as the same as the theta factor (6th parameter), but it works! (One would have to prove that with these parameters one gets an exact straight hexagon edge.)

documentation

Inspired by:

```
> Article 7568 in sci.math:
> From: pvmg0487@uxa.cso.uiuc.edu
> Subject: hexagonal cone function sought
> Message-ID: <107700002@uxa.cso.uiuc.edu>
> Date: 22 Nov 89 22:25:00 GMT
>
> I would like to generate a 3-D cone like object, but with a hexagonal
> base.  Any suggestions as to an appropriate equation?
>
```

```
> Thanks -- Vernon

> Article 7578 in sci.math:
> From: toms@alum.mit.edu (Tom Schneider)
> Subject: Re: hexagonal cone function sought
> Message-ID: <1405@fcs280s.ncifcrf.gov>
> Date: 25 Nov 89 01:18:14 GMT
> References: <107700002@uxa.cso.uiuc.edu>
> Reply-To: toms@fcs260c2.UUCP (Tom Schneider)
> Organization: National Cancer Institute, Frederick
> Lines: 30
>
> In article <107700002@uxa.cso.uiuc.edu> pvmg0487@uxa.cso.uiuc.edu writes:
> >
> >I would like to generate a 3-D cone like object, but with a hexagonal
> >base.  Any suggestions as to an appropriate equation?
> >
> >Thanks -- Vernon
>
> Well, that's pretty surprising, since just today I was thinking about a
> function that does almost exactly what you want!  It turns out that the
> equation x^n + y^n = r^n is a line (diamond) if n = 1, a circle if n = 2 and
> approaches a square as n -> infinity!  So all one needs to do is express this
> in polar notation, and then scrunch an extra two corners in to get what you
> want!
>
> First, use the form x^n + y^n = rmax^n (to avoid confusion!) and substitute x
> = r cos(theta), y = r sin(theta).  Divide both sides by r^n, and rearrange to
> get r expressed as a function of theta.  To get the powers, I had to use a^b
> = exp(b*ln(a)).  The thing is symmetrical around the 4 quadrants, so I
> avoided logs of negative numbers by taking the absolute values of the sine
> and cosine functions.  Also, at angles of n*pi/2, one gets division by zero,
> so just substitute the desired radius.
>
> I have done this by writing a Pascal program that will do the job.  Pretty!
> It turns out that to get a hexagon, you have to plot between n=1.5 and n=2
> because of the scrunching.  Email me if you want a copy of the program.
>
>   Tom Schneider
>   National Cancer Institute
>   Laboratory of Mathematical Biology
>   Frederick, Maryland  21701-1013
>   toms@alum.mit.edu

> From daemon Tue Nov 28 09:34:41 1989
> Return-Path: <daemon>
> Date: Tue, 28 Nov 89 08:35:52 -0600
> From: Paul Vernon McDonald <pvmg0487@uxa.cso.uiuc.edu>
> Message-Id: <8911281435.AA01048@uxa.cso.uiuc.edu>
> To: toms@alum.mit.edu
> Subject: Pascal code for hexagon
>
> Tom,
> I'd be most grateful to receive your code, if you are willing to share it.
```

```
> I curently have a working version of the hexagon, done in piecewise
> fashion, but I'd be interested in a generic solution.  In fact I plan
> to use other shapes in the future, so your code may be of great help.
>
> Thanks,
>
> Vernon McDonald
> University of Illinois
> Department of Kinesiology
> Urbana, IL, 61801
> vmcdonald@uiuc.edu
>
> From toms Tue Nov 28 13:17:17 1989
> To: pvmg0487@uxa.cso.uiuc.edu
> Subject: Cisq
>
> Vernon:
>   Sure, I wrote the code mostly because of your posting.  But actually it
> has suddenly become very important to my work (it's a long story...) and so
> it is useful to me to have it.  I have to brush it up a bit and I will
> send it to you.  If you have a PostScript printer, then you may also
> want the xyplo program, which produces PostScript x-y plotting of data.
> This made writing cisq (circle square) easier because I only needed to
> create the right numbers and xyplo did the graphics for me.
> Tom

see also
   xyplo.p, the Pascal program that produces PostScript x-y plotting graphics.

author
   Thomas Dana Schneider

bugs
   One might also want to produce the hexagon for INCREASING values of n,
   rather than being confined into the region n=1.5 to 2.  It seems that to do
   this requires that one do a fancy job of warping the square region into the
   appropriate triangular region.  This should be pretty easy with the right
   afine transformation, but the program doesn't have that feature in it.
   Fortunately, it is not necessary.

technical notes

*)
(* end module describe.cisq *)
version = 1.43; (* of cisq.p 1989 December 19
```

```
(* begin module describe.ckhelix *)
(*
name
   ckhelix: check that the helix location is where one wants

synopsis
   ckhelix(makelogop: in, ckhelixp: in, output: out);

files
   makelogop: the parameter file of the makelogo program
   ckhelixp:
      wave location: the point in bases on THIS logo which is to
         align with the other logos.
         NOTE:  this is NOT necessarily the high or low point
         of the wave as given by the wave parameter file of the
         makelogo program, hence it is not read from that file.
      zero:  location of the desired center in cm on the page
   output: messages to the user

description
   The program is used to determine the position to place a sequence logo so
that a particular point of the cosine wave (in bases of the nucleic acid
coordinate system) is exactly at a given point on the page in cm.  This allows
one to adjust the location of the logos so that they can overlap.

examples

documentation

see also

author
   Thomas Dana Schneider

bugs

technical notes

*)
(* end module describe.ckhelix *)
version = 1.01; (* of ckhelix.p 1992 April 28
```

```
(*
name
   cluster: cluster indana subindexes into groups of duplicate entries

synopsis
   cluster(clusterp: in, subind: in, inst: in, book: in,
           pairs: out, clumps: out, output: out)

files
   clusterp: The cluster parameter file that consists of the following:
             FIRST LINE  'y' turns the flag on, 'n' turns it off
                  (debugging) allows one to look at raw data in the bags.
             The debugging flag controls the printing of the raw data above the
             regular output of the cluster program, which is created solely by
             procedure showRAWbag.  This can then be compared with the data in
             the chart for correctness.  Raw data consists of the series of
             coordinate pairs in the bag and the sides they are matched on.
             printed above the standard output structure.

             example: -  (  630,   69)   R
                      L  (  649,   88)   -  {20}  {20}
                      *************************************
                                          |   630       663
                      HUMUK               |     ----------
                                          |         34
                      HUMUPA              |     ----------
                                          |    69       102
                      *************************************

             It is important to note that the raw data will only appear in the
             pairs output file, and will not be written in clumps at all.  This
             means that parameter 3, writepairs, must also be turned on for
             this flag to be effective.

             SECOND LINE 'y' turns the flag on, 'n' turns it off
                  (showfragments) allows one to see pairs that are fragmented.
             The showfragments toggle controls printing the outputs of pairs
             with "imperfect" matches.  That is, in some cases a repeating
             sequence will match in several frames, causing repeated sequence
             matching and producing a large list of coordinate pairs.  This
             list can be shown if the parameter is turned on, but the statement
             "WARNING:  sequence pairs are overmatched" will appear if it is
             turned off.  The actual sequences will be shown in either case,
             so the comparison can always be done by hand by the user.  The
             output is excessively long, but the sequences will be shown, so
             the comparison can be done by the user.

             example:    1     acggatcgtgtgtgtgtgtgtgtgtacgatcggatcgat
                         2     acggatcgtgtgtgtgtgtgtgtgtacgatcggatcgat

             These sequences will have matches between all of the 'gt' base
             pairs, resulting in an overwhelming number of matches.  The
             maximum number of possible matches is found by taking the length
```

of the sequences and dividing it by the value in the overmatched parameter (FIFTH LINE) times the number of instructions that match between any two pieces in the dbinst. This results in a maximum number of matches between any two pieces. Any pieces above this limit will can have their output completely shown or can generate a warning message (see showfragments, SECOND LINE). In addition to preventing the example case, showfragments will also prevent the display of any other case that may cause an excessive number of matches.

THIRD LINE 'y' turns the flag on, 'n' turns it off
(writepairs) controls the printing of the pairs output file. If writepairs is on, the original clustering pairlist will be printed into the output file pairs. If it is off, this file will not be printed. This parameter must be turned on to effectively use the debugging parameter (see FIRST LINE).

FOURTH LINE 'y' turns the flag on, 'n' turns it off
(writeclumps) controls printing of the clumps output file. If writeclumps is on, the original clustering pairlist will be sent through the clumping procedures. The output file clumps will contain the sequences involved in the matches on the pair in addition to the clumped version of the pairlist. The clumping process takes an excessive amount of time for very large files, since the program must traverse the entire pairlist to find all related pairs, then put the pairs on to the clumplist, then go through the book and find sequences to match every instruction in every pair of every clump. Although it is much easier to determine which pieces are true repeats through use of the clumps file, it is certainly possible to do so by simply using the pairs output file.

FIFTH LINE any integer
(matchparameter) is the number of matches to be allowed between two instructions. This can be determined by dividing the sequence length from the book by the minimum window size from the subindex, or a maximum number of matches between instructions can be set. An integer less than or equal to 0 will calculate maximum matches using the above method. Any number greater than 0 will be used as the new maximum matches.

example: if the instructions call for the sequences

```
piece1: get from 100 -50 to 100 +50;
piece2: get from 200 -50 to 200 +50;
```

The sequence length is 101. If the windowsize read from the subindex = 15, then 6 possible matches can occur between these two instructions (101 div 15 = 6).

The TOTAL number of matches between two pieces is found by multiplying matchparameter by the number of instructions in a given pair. If a piece has more matches than this, it is considered to be overmatched, the bag will not be printed, and the

```
    statment 'WARNING: sequence pairs have too many matches.' will
    appear.  Overmatched pairs can be printed using the showmatches
    parameter (see SECOND LINE).
   subind: a subindex from the indana program matching the inst and the book
   inst: a set of delila instructions that correspond to the book
   book: a delila book that contains the sequences being clumped
   pairs: the output list of paired sequences
   clumps: the output list of clumped sequences
   output: When errors occur, the program halts and produces an error message

description
   Duplicate entries in the subind subindex are clustered into a unified list
   of pairs and copied to output files as sequence numbers, lengths, and
   sequence base pairs.

   Pairs are determined by the indana program, which delegates sequence
   similarities with an '*'.  Cluster takes the subindex and shows the
   coordinate range and length of the similarity by pairs.  The pairs file is
   a list of relationships between two sequences, the clumps file takes this
   list of pairs and groups related ones together. The seqalign modules of the
   program then access the book and get the corresponding sequences to print
   out with the instruction number and piece name.

documentation
   none

see also
   index.p, indana.p

author
   R. Michael Stephens

bugs
   None currently known.

technical notes
   The read for the indana window size is based on the '[' character before
   the number in the subind heading.  Any changes to indana that alter this
   format must be reflected in the getwindowsize procedure.
*)
(* end module describe.cluster *)
version = 5.06; (* of cluster.p 1992 September 18
```

```
(*
name
      coda: composition file to data for genhis

synopsis
      coda(cmp: in, data: out, codap: in, output: out)

files
      cmp: a composition, the output of program comp
      data: identification lines are followed by
         the number of occurences of each oligo and the sequence
         of the oligo, one pair per line.  the form of the
         file can be changed using the parameters in codap.
      codap: parameter file.  four parameters, one per line.
         1. composition depth to be used in the data file (integer)
         2. the least frequent oligo to record in data (integer).
         3. the most frequent oligo to record in data (integer).
         4. if the first character is 'b', the number of each
         oligo is given before the oligo, 'a' means after.
         'n' means do not give the number.  's' means the data file will be
         used as input to the search program.  no numbers are given and
         commands to search are made which will result in a list
         of the locations of the selected oligos.
            if parameters 2 to 4 are missing they default to
            0 100000 b.

      output: messages to the user

description
      coda converts a composition file from the comp program
      into a list of oligos.  unlike the original composition
      file, this list may contain all oligos of the length desired
      (to save space, comp removes an n-long oligo when the
      two n-1 long oligos inside it do not exist).  however,  coda
      can be told to only include frequent or infrequent oligos
      using the parameter file.  two ways to use the data are:
      1. use the data file as input to genhis to determine the
      distribution of the composition.
      2. use the 's' feature to generate instructions for the
      search program.  search converts the list of oligos to
      locations in a sequence.  unshi then is used to remove
      the extra blanks and genhis then gives a map of the
      locations of rare or common oligos.

example
      file: datat7

see also
      comp, genhis, search, unshi

author
      thomas dana schneider
```

```
bugs
     none known
*)
(* end module describe.coda *)
version = 2.04; (* of coda, 1986 dec 15
```

```
(* begin module describe.code *)                                                   143
(*
name
      code: find the comment density of a pascal program

synopsis
      code(fin: in, output: out)

files
      fin: a pascal source code.
      output: a report on the comment density of the pascal program.

description
      with the comment density program, you can find out how much of
      your program is devoted to comments.  in general, the better programs
      will have more comments than those that are poor.  the program gives
      you the percent of characters devoted to comments.  a typical value
      should probably be around 30 percent of the characters devoted to
      description.  suggested places to put comments are in the delman
      manual in the module delman.guide.programming.

author
      thomas d. schneider

bugs
      the program does not keep track of blanks, so one's style with
      blanks could affect the percentage.

*)
(* end module describe.code *)
version = 2.06; (* of code 1986 dec 9
```

```
(* begin module describe.column *)
(*
name
   column: pull defined column from input

synopsis
   column(input: in, columnp: in, output: out)

files
   input:  file with several columns of data separated by spaces
   columnp: parameters:  one line:  which column to extract
      Lines in input that start with '*' are simply copied to the output.
   output: messages to the user

description
   The column program allows one to extract columns from a dataset.

examples

documentation

see also

author
   Thomas Dana Schneider

bugs

technical notes

*)
(* end module describe.column *)
version = 1.03; (* of column.p 1992 September 16
```

```
(*
name
      comp: determine the composition of a book.

synopsis
      comp(book: in, cmp: out, compp: in, output: out)

files
      book: the sequences;
      cmp: the composition, determined for mononucleotides up to
            oligonucleotides of length "compmax", see file compp;
      compp: parameter file used to set the length of the oligonucleotides for
            which the composition is to be determined ("compmax");  that number
            must be the first thing in the file; if the file is empty
            compmax is set by default to the constant "defcompmax";
      output: for messages to the user.

description
      counts the number of each oligonucleotide (from length 1 to compmax) in
      the book and prints that to file "cmp".  the output is printed in order
      of increasing length of oligonucleotide (i.e., first the monos, then the
      dis, ...).  if there are no occurences of an oligonucleotide, but its
      one-shorter parent did occur, it will be given a zero.  none of its
      descendants will be printed in the composition file.

see also
      compan, histan

authors
      gary stormo and tom schneider

bugs
      none known

technical note
      the algorithm is an interesting application of linked lists.  the
      composition is stored as a tree, and a number of "spiders" climb the
      tree during its construction.

*)
(* end module describe.comp *)
version = 5.25; (* of comp, 1988 oct 10 *)
```

```
(* begin module describe.compan *)
(*
name
      compan: composition analysis.

synopsis
      compan(cmp: in, anal: out, companp: in, output: out)

files
      cmp: the input composition, which is the output of program comp;
      anal: the output analysis of this program;
      companp: for parameters; should contain a single integer which specifies
         the maximum level for which the composition is analyzed.  the
         maximum allowed level is 4, or the maximum level for which the
         composition was determined.
      output: for user messages;

description
      calculates chi squared from a composition using:
         1) assumption of equal frequencies to calculate mono, di, tri
            and tet expecteds;
         2) mono frequencies to calculate di, tri  and tet expecteds;
         3) di frequencies to calculate tri and tet expecteds;
         4) tri frequencies to calculate tet expecteds;
      the partial chi squared values are given for each oligo.
      the 'information' content of the composition is also calculated,
      using the standard information theory definition:
         information = -sum(frequency * log(frequency)),
      where the sum is over each oligonucleotide of a given length
      and the log is taken to the base 2.  this gives the information
      in bits.

see also
      comp

author
      gary stormo

bugs
      the program cannot do calculations for compositions larger than 4
*)
(* end module describe.compan *)
version = 3.23;  (* of compan, 1988 oct 10
```

```
(*
name
      concat: concatenate files together

synopsis
      concat(afile: in, bfile: in, abfile: out, output: out)

files
      afile: the first file to be copied to abfile
      bfile: the second file to be copied to abfile
      abfile: the concatenation of afile and bfile
      output: messages to the user

description
      concat joins two files, afile and bfile, into a single file
      named abfile.  afile is first copied to abfile, followed by bfile.
      a warning is given to the user if either afile or bfile is empty, but
      in this case, the program copies the other file to abfile anyway.

examples
      one can use concat to join delila instruction sets in the cyclic
      teaching of the perceptron (see our third nar paper).  note that
      delila will not accept several titles in the instructions, so be sure
      that one of the two sets has no title, or remove it by hand.

author
      billie lemmon and thomas schneider

bugs
      none known
*)

(* end module describe.concat *)
 version = 1.08; (* concat 1986 dec 9
```

```
(* begin module describe.copy *)
(*
name
      copy: copy one file to another file

synopsis
      copy(fin: in, fout: out, output: out)

files
      fin: the file to be copied
      fout: the copy of fin
      output: messages to the user

description
      copy makes one copy of the file fin on the file fout.  you may
      discover that this is a simple task that you often want to do, but
      that your system does not provide an easy way.

see also
      shift

author
      thomas d. schneider

bugs
      none known

*)
(* end module describe.copy *)
version = 1.06; (* of copy.p 1985 march 9 *)
```

```
(* begin module describe.count *)
(*
name
      count: counts the amount of sequence in a book

synopsis
      count(book: in, list: out, output: out);

files
      book: any book from the delila system
      list: the number of bases in each piece and the total number of bases
      output: messages to the user

description
      count is a tiny tool, much like a tooth pick, that is handy to have
      around.  the count is based on the coordinate system of each piece,
      not on the actual number of bases.

author
      thomas d. schneider

bugs
      if the number of bases does not match the coordinate system, then
      no warning is given to the user.

*)
(* end module describe.count *)
version = 3.07; (* of count.p 1991 Aug 6
```

```
(* begin module describe.cybmod *)
(*
name
      cybmod: specific module library for the cyber computer

synopsis
      cybmod(output: out)

files
      output: where the date and time will appear.

description
      cybmod contains modules that will replace corresponding modules in
      the other module libraries which are cyber-system dependent. this
      will allow easy transportation of the delila system to cyber computers
      running under kronos.

documentation
      moddef, delman.describe.module

see also
      delman.describe.delmod, moddef, delman.describe.module

see also
 delmods, prgmods, matmods, vaxmods

author
      thomas d. schneider

bugs
      none known

technical notes
      the datetime package required a const 'namelength' and a type 'alpha'.
      these are part of the book.const and book.type modules of delmod, and
      are identical to those types and consts.  note:  programs which use
      the datetime package must have these types and consts either from
      delmod or manually declared.
*)
(* end module describe.cybmod *)
version = 1.02; (* of cybmod 1986 nov 11'*)
```

```
(* begin module describe.da3d *)
(*
name
   da3d: diana da file to 3d graphics

synopsis
   da3d(da: in, scene: out, output: out)

files
   da: output of the diana program; position to poistion correlations
   da3dp: parameter file to control scene.
           horizontal: shift of graph horizontally (in cm)

           vertical: shift of graph vertically (in cm)

           xlocation: location of viewer in bases

           ylocation: location of viewer in bases

           zlocation: location of viewer in bits

           magnify: magnification factor for whole scene, 1 = no change.

           xmagnify: magnification factor for x axis only.

           ymagnify: magnification factor for y axis only.

           zmagnify: magnification factor for z axis only.

           datacolumn: column of da which to use for the graph.

   scene: 3D scene of the da data according to da3dp.  Result is in
      PostScript
   output: messages to the user

description
   Show the position to position correlation data in three dimensions.

examples

documentation

see also

author
   Thomas Dana Schneider

bugs

technical notes

*)
(* end module describe.da3d *)
version = 1.17; (* of da3d.p 1991 December 11
```

```
(*
name
   dalvec: converts Rseq rsdata file to symvec format

synopsis
   dalvec(rsdata: in, dalvecp: in, symvec: out, output: out)

files
   rsdata: data file from rseq program

   dalvecp: parameters to control dalvec
      If empty, then the normal sequence logo will be produced.
      If the first character of the first line is a 'c', then a chi-logo
      is produced.  The height of this logo is the information.  The
      heights of the individual letters are, however, not the frequencies,
      but rather their partial chi-square values.  The expected value
      is 1/4 of the number of characters.  This is compared to the observed
      value by:
        partial chi-square =(observed - expected)^2/expected
      These partial values are normalized and placed in symvec in place of
      the relative frequencies.  Thus the significance of each letter is
      used.  When the observed is less than expected, the reported value
      is made negative.  Makelogo prints these characters upside down.
   symvec: reformating of the rsdata file for input to the makelogo program.
      A series of header lines begining with asterisk ("*") are produced.
      The next line contains one integer which is the number of symbols
      in the vector (4 for DNA or RNA, 20 for proteins).
      After this, the format of the file is a series of entries.  Each entry
      has two parts.  The first part is on one line and contains
         position total information
         position: the position number
         total:  the sum of the values in the vector
         information: the information content of the vector.
      The remaining parameters on the line are from the rsdata file:
         rs: sum of rsl
         varhnb: variance of rsl
         sumvar: sum of varhnb
         ehnb: 2-e(n)

      The second part consists of a list of 4 integers, representing the
      the numbers of bases or amino acids at the position in an aligned
         set of sequences.

   output: messages to the user

description
   Convert the rsdata file from rseq into a format that the makelogo program
   can use.  The format is a 'symbol vector'.

   ChiLogos: If you leave the parameter file empty, then the standard sequence
   logo will be created.  However, if the first letter of the file is a 'c',
   then a new kind of logo will emerge from makelogo: the chi-logo.  The height
   is as it was before.  The height of the individual letters is different,
```

```
   instead of being proportional to the frequency of the letter, it is                    153
   proportional to the significance of the letter, based on the chi-square
   test.  That is, the expected number of letters is the number of letters at
   that position, n(l) divided by 4 (for simplicity!).  The observed number
   comes from the rsdata file.  The partial-chi square is
   (observed-expected)^2/expected.  Note that the sum of the partials is the
   normal chi-square.  So bases that contribute strongly get big.  Also, bases
   that are under represented are printed UPSIDE DOWN, so you can (usually)
   tell you have a chilogo at a glance.  The chilogo allows one to see the
   importance of the infrequent letters.  The technical mechanism for making a
   letter upside down is to have its number negative in the symvec file.

author
   Thomas D. Schneider

examples

see also
   rseq, makelogo

bugs

   The program originally only created a vector that contained the characters
   of the alphabet, so the output was called an 'alvec'.  To reflect the use of
   symbols, the name of the output file was changed to symvec, but I like
   'dalvec', and 'dsymvec' is so awkward that I decided to keep the name
   dalvec.

*)
(* end module describe.dalvec *)
version = 2.14; (* of dalvec.p 1991 October 1
```

```
(*
name
      dbbk: database to delila book conversion program

synopsis
      dbbk(db: in, l1: out, changes: out, output: out)

files
      db: contains one or more complete entries from either the EMBL
         or GenBank genetic sequence data bases.  These entries may be
         obtained by using the original libraries or by using an entry
         extraction program.  Dbpull is the delila program for data base
         accessing; to get complete entries the instruction 'all' must
         have been used in the dbpull fin file.  (See delman.use.dbpull)
      l1: each db entry is represented in l1 by a delila style
         entry containing information extracted from the db entry.
         All of l1 has the biologically oriented structure of
         a standard delila book.  The first line of l1 is not part
         of an entry, but contains the computer system date and the
         title of the book.
      changes: Delila programs cannot handle sequences that have
         ambiguities because Delila was designed on the assumption
         that people would finish their sequences.  Unfortunately
         this is not true, and the databases contain bases other
         than acgt to indicate ambiguity.  These are converted to
         "a" and the cases are reported in this file.
      output: messages to the user.

description
      This program converts GenBank and EMBL data base entries into a
      book of delila entries.  The organism name is fused together
      with a period and is used for both organsim and chromosome names.
      Organism and chromosome only change if the name changes in db.

see also
      delila, dbpull, libdef, catal

author
      Matthew Yarus

bugs
      databases do not have enough data on genes within each piece to make
      a book with gene sections.

      The changes file is a design bug in Delila.

      Genus names are limited to genuslimit (a constant) to avoid
      names longer than the standard Delila limit.

technical notes
      dbbk is known to convert GenBank entries from July 1989.
      It may not work on later versions.
*)
```

```
(* end module describe.dbbk *)
version = 3.13; (* of dbbk.p 1992 December 10
```

```
(* begin module describe.dbcat *)
(*
name
      dbcat: database catalog production and sorting program.

synopsis
      dbcat (dbl1, dbl2, dbl3, dbl4, dbl5, dbl6, dbl7, dbl8: inout,
             ecat: out, gcat: out, output: out )

files
      dbl1, dbl2, dbl3, dbl4, dbl5, dbl6, dbl7, dbl8:  text libraries that
         contain entries of either embl(european molecular biology labratory)
         or genbank(genetic sequence data bank) types. in both cases the
         general format is a series of entries, each entry beginning with a
         twenty letter identification code name for a particular genetic
         sequence followed by many lines of other relevant information. all
         lines begin with a two or three letter code identifying the purpose
         of the line. however, the two entry types have different line codes
         and contain similar but not identical kinds of information.
      ecat:  catalog of embl type library entries. each catalog entry
         contains the location of the beginning of the library entry, a
         number signifying which library the entry is found in, and the
         special identification code of the entry's genetic sequence.
      gcat:  same as ecat except containing information on genbank entries.
      output:  messages to the user.

description
      this program makes catalogs for use in the program dbpull. in
      addition to sorting catalog entries in the innate alphanumeric
      order of the computer it is run on, dbcat marks both catalogs and
      libraries with the date of the run so that dbpull never uses mis-
      matched sets of information.

documentation
      delman.describe.dbpull, embl and genbank libraries.

see also
      loocat, catal, dbpull.

author
      matthew yarus

bugs
      none known

technical notes
      dbcat functions on genbank(tm) release 9 (june 1, 1983)
*)
(* end module describe.dbcat *)
version = 2.10; (* of dbcat.p 1989 July 11
```

```
(*
name
   dbfilter: filter GenBank databases to remove unwanted entries

synopsis
   dbfilter(input: in: output: out, dbfilterp: in)

files
   input: a database of GenBank entries
   output: database after the filtration.  When errors occur, the program halts
           and produces an error message at the end of the output file.
   dbfilterp: parameters to control the program
        FIRST LINE: the name of the organism to use, consisting of two parts
        (eg, Homo sapiens).

description
   GenBank entries in input that contain the requested organsim are copied
   to output.

   The GenBank ORGANISM contains the two part genus/species name, such as:

  ORGANISM  Homo sapiens

   Entries of an unwanted ORGANISM type are not copied from input to output.
   Those of the desired type are transferred directly.

examples
   If dbfilterp contains:
      Homo sapiens
   then only those entries with the ORGANISM type Homo sapiens will be copied
   into output. All others will be filtered out.

documentation
   none

see also
   dbinst.p dbbk.p

author
  R. Michael Stephens

bugs
   Error messages are buried at the bottom of the output file.

technical notes
   Constant maxlines determines the greatest number of lines that can be handled
   between LOCUS and ORGANISM.

*)
(* end module describe.dbfilter *)
version = 1.08; (* of dbfilter.p 1992 November 1
```

```
(*
name
   dbinst: extract Delila instructions from a GenBank database

synopsis
   dbinst(db: in,
          binst: out, einst: out,
          oinst: out, sinst: out,
          olength: out, slength: out,
          dbinstp: in, locuslist: out, missing: out, output: out)

files
   db: a set of GenBank entries
   binst: instructions for finding the beginning of a feature
   einst: instructions for finding the ending of a feature
   oinst: instructions for finding the whole feature, called the "object".
      They are given in the form "from begin + f to end + t" where f and t are
      the "from" and "to" parameters given in dbinstp.
   sinst: instructions for finding the regions between features, called
      the "space".  They have the same form as those of oinst.
   olength: list of object lengths
   slength: list of space lengths
   dbinstp: parameters to control the program
     First line: the name of the feature to use.
     Second line: two integers, the base "from" and the base "to" relative to
        the alignment point to write the instructions.
        If "from" is larger than "to" then generic names "before" and "after"
        are written.  This allows one to make a generic file of instructions
        to be copied and edited later.
     Third line: 4 characters without spaces that control which instruction
        files are to be written.  To have all 4 on, use 'beos', for begin, end,
        object and space.  Any other character means that the corresponding
        file will not be written.  The file will be rewritten however.
     Fourth line: 2 characters without spaces that control which length
files are to be written.  To both on, use 'os', for object and space.
Any other character means that the corresponding file will not be
written.  The file will be rewritten however.
     Fifth line:  If the first character is 'r' then remove obviously
duplicated instructions and object or space lengths.  When alternative
splicing occurs, GenBank records the endpoint several times, so that
the sequence instructions are identical.  By using this toggle switch,
such cases are eliminated.
     Sixth line:  If the first character is 'f' then the coordinates of the
        instruction are written whether or not the object is off the end
        of the sequence.  This allows one to pick up objects that are
        partially on a piece.

        If the first character is 's' then select against the feature if
        either end is missing.  This makes the length list correspond
        to the instruction set.

     Seventh line:  Alignment shift.  This integer is added to the
        from and too coordinates of the instructions written out.
```

Normally this should be 0. An example helps. Normally, if the zero of splice donor sites is defined the first base on the intron, then if one is writing instructions based on exon coordinates the zero base will be 1 too low. By making the alignment shift 1, the instructions written out will match the expectations of other programs.
Note: object coordinates are shifted accordingly; this may not be quite what you want if you are using them from the olength file! However, the length is not affected.

locuslist: a list of all the loci in the db that have features of interest. This list can be used with dbpull to create reduced databases containing only those entries that contain the features we want.

missing: Features that are listed under the database COMMENT are listed here. These are "EMBL features not translated to GenBank features". We do not consider these to be reliable. They are NOT included in the binst, einst or olength, slength instructions.

output: messages to the user

description

The GenBank entries in db are scanned, and Delila instructions are generated, according to a desired feature table item. Four kinds of instruction are made: beginning, ending, object and space. Beginning appears only if the data for the beginning of the feature is in the db. Ending appears only if the data for the ending of the feature is in the db. Object appears only if both the beginning and ending are there. Space only appears if there was an ending to the previous feature, and the current feature has a beginning. Thus object and space instructions is guaranteed to be a "natural" length.

The names for the instructions are determined as follows. The GenBank ORGANISM contains the two part genus/species name, such as:

```
ORGANISM  Homo sapiens
```

The parts are joined into "Homo.sapiens", and this becomes the name of the organism and chromosome in the instructions. The instructions for organism and chromosome only change when the genus/species name changes in db. The LOCUS name of the entry is picked up and used as the piece name. These naming conventions are the ones generated automatically by the dbbk program, so one need not think about it most of the time.

In each entry, lines of the form:

```
 pept    <     1       46     Ig V-R-H region protein, exon x
```

are located and used to generate Delila "get" statements.

If a "<" appears before the first number, then no instruction is written to binst, since the beginning point is before the GenBank sequence.

If a "<" appears before the second number, then no instruction is written to einst, since the ending point is after the GenBank sequence.

If a "<" or ">" appears in the db, then no object instructions or lengths are written.

If a ">" appears in the previous feature or ">" appears in the current feature, then no space instructions or lengths are written.

So for the above example, only one Delila instruction would be written:

```
get from 46 -10 to 46 +20;
```

if the dbinstp contained -10 20, and

```
get from 46 before to 46 after;
```

if the dbinstp contained 10 -20.

where "before" and "after" are replaced by the integers from dbinstp.

examples

If dbinstp contains:

```
pept
-10 20
```

then instructions to get peptide starts (binst) and ends (einst) from -10 to +20 will be made.

Instructions for the entire peptides, from -10 before the start of the peptide to 20 bases after will also be made.

Instructions for the regions between peptides, from -10 inside each previous peptide to 20 bases into the inside of the next peptide will also be made.

documentation

none

see also

dbbk.p

author

Thomas Dana Schneider

bugs

The program does not produce the instructions for space between the first object and the beginning of the sequence or the space after the last object in the sequence. This is possible (and perhaps should be controlled by a parameter) but it would not produce "natural" lengths because those space lengths depend on the length of the reported sequence.

It is not clear that spaces are done properly anymore. Possible bug at "SPACE PROBLEM".

Genus names are limited to genuslimit (a constant) to avoid names longer than the standard Delila limit.

technical notes

```
   The expected column locations of the complement flag in the database, (the
   'before end of piece' and the 'after end of piece' flags) are given in the
   program constants.

*)
(* end module describe.dbinst *)
version = 3.39; (* of dbinst.p 1992 September 16
```

```
(* begin module describe.dblo *)
(*
name
      dblo: look at the catalogue of a genbank/embl database

synopsis
      dblo(cat: in, list: out, output: out)

files
      cat: a catalogue from program dbcat
      list: a listing in tabular form of the catalogue
      output: messages to the user

description
      the program dbcat creates a machine readable catalogue of the
      locations of entries in a genbank /embl database.  one cannot
      read this directly because it is a compressed internal format
      of the computer.  (that is, it is a file of records.)  to
      read the file, one must convert it into normal characters, which
      is what dblo does.

author
      thomas schneider

see also
      dbcat, dbpull, loocat, delila

bugs
      none known

*)
(* end module describe.dblo *)
version = 1.09; (* of dblo 1989 July 11
```

```
(*
name
      dbpull: database extraction program.

synopsis
      dbpull (fin: in, fout: out,
              dbl1, dbl2, dbl3, dbl4, dbl5, dbl6, dbl7, dbl8: in,
              ecat: in, gcat: in, output: out)

files
      fin:  User requests for extractions from libraries.  Each request
         takes up a single line and consists of a genetic sequence identi-
         fication code followed by either a single special extraction code
         or a series of line code requests.  If an entry request is to be
         found in embl format the request line must have a line containing
         simply 'embl' somewhere above it.  A line containing only 'genb'
         will instruct dbpull to look only for genbank format entries on
         the following request lines.  Important note: the exact form of
         fin instructions is found in delman.use.dbpull.instructions.

         If no request is given, then ALL is assumed.  This means that
         the program will now run using a raw list of entry names.
      fout:  contains fulfilled requests in the same entry order as fin.
         this file may serve, itself, as a database library for dbpull as
         long as 'id ' or 'loc' occur with every request.(one of these two
         line codes identifies the beginning of each entry and holds its id)
      dbl1-dbl8:  same files, in the same order, as dbcat. see delman.
                  describe.dbcat.
      ecat:  same as in dbcat also.
      gcat:  same as in dbcat also.
      output:  messages to the user.

description
      this program uses catalogs generated by the dbcat program to quickly
      extract all or part of embl or genb type entries from data base lib-
      raries. the user may choose one of two special requests('all', which
      pulls out an entire entry or 'raw', which pulls out only the genetic
      sequence) or s-he may simply request a number of line codes. the wild-
      card character '*' represents any number of unspecified characters
      in an id request. this allows one fin line to extract several entries
      whose ids have characters in common. the id 'every' extracts all ids
      it is compared to. dbpull also checks the production dates of all the
      catalogs and libraries to see that they are consistent.

documentation
      dbhelp, delman.describe.dbcat, embl and genbank libaries.

see also
      dbcat.

author
      matthew yarus
```

```
bugs
      none known

technical notes
      1:dbpull functions on genbank(tm) release 9 (june 1, 1983). 2: if the
      value of the constant checknum is increased, dbpull will do a more
      complete check of its catalogs.
*)
(* end module describe.dbpull *)
version = 2.41; (* of dbpull.p 1989 November 14
```

```
(*
name
   decat: break a file into 10 files

synopsis
   decat(input: in, decatp: in,
         f0,f1,f2,f3,f4,f5,f6,f7,f8,f9: out,
         output: out)

files
   input:  multiple line detailed description of file 1, etc
   decatp: parameters.  one integer, the number of bytes to put into
     each file.
   fx: input split into parts f1..fx
   output: messages to the user

description
   Break a file into parts.  Any excess goes into the last file.
   The files are split at the next line after the size given in decatp
   has been exceeded.  This avoids broken lines, but it means that the
   user must leave a safety.

   Purpose: to be able to send files larger than 50000 bytes.
   The mailer at boulder objects to ones larger.
   Test for correctness: cat f0 ... f9 >x; diff of input and x should be empty.

author
   Thomas Dana Schneider

bugs
   fixed number of files.

*)
(* end module describe.decat *)
version = 1.13; (* of decat 1989 September 25
```

```
(* begin module describe.decom *)
(*
name
   decom: remove comment starts from within a comment

synopsis
   decom(input: in; output: out)

files
   input:  a program with comments within comments.
   output: the same program with internal comments neutralized.

description
   At the moment there are many cases in the delila system where the construct:
              ( ( )
   exists (where '(' means the begin of a comment and ')' means the end).
   This is a result of the version mechanism of the module program.
   Until this is changed, these will hang around.  The Sun compiler
   gives a warning about these, and to remove the warnings, the '*'
   after the second '(' can be removed by this program.

see also
   module.p

author
   Thomas Dana Schneider

bugs
   WARNING:  Some programs have comment starts inside quotes.  DECOM
   IS NOT SMART ENOUGH TO AVOID CHANGING THESE.  If they exist, decom
   will mess up your program.  Compare the output of decom with the
   input before you accept the results.

*)
(* end module describe.decom *)
version = 1.03; (* of decom, 1988 Dec 14
```

```
(*
name
      delila: the librarian for sequence manipulation

synopsis
      delila(inst: in, book: out, listing: out,
             lib1: in, cat1: in,
             lib2: in, cat2: in,
             lib3: in, cat3: in,
             output: out, debug: out)

files
      inst: instructions written in the language delila that tell the
         program delila what sequences to pull out of the library.
      book: the set of sequences pulled out of the library.
      listing: the instructions are listed along with errors found or
         actions taken.
      lib1: the first library from which to obtain sequences
      cat1: the first catalogue, corresponding to lib1
      lib2: the second library
      cat2: the second catalogue, corresponding to lib2
      lib3: the third library
      cat3: the third catalogue, corresponding to lib3
      debug: traces through the actions taken, for debugging delila
         (only produced if constant debugging is true.)
      output: messages to the user

description
      delila is a data base manager for nucleic acid sequences.  it takes
      a set of instructions, written in the language delila (deoxyribonucleic
      acid library language) and a large set of sequences called a library.
      the output is a listing of the actions taken (or errors) corresponding
      to the instructions, and a "book" containing the sequences desired.

examples
      see the documentation

documentation
      libdef (defines delila), delman.intro, delman.use, delman.construction

see also
      catal, loocat

author
      thomas d. schneider, gary d. stormo and paul morrisett
      useful suggestions by jeff haemer

bugs
      there are many known bugs in delila.  most are related to extracting
      linear fragments of circular sequences.  we are designing a second
      version of delila which should solve these problems.
         the following features are not available in this program:
      recognition classes and enzymes, markers,
```

```
      automatic printing to the book of structures that intersect a piece,
      get all (for org, chr, rec and enz), get every and if.

*)
(* end module describe.delila *)
version = 1.77; (* of delila.p 1989 November 14 *)
```

```
(* begin module describe.delmod *)
(*
name
      delmod: delila module library

synopsis
      delmod(book: in, output: out)

files
      book: any book from the delila system, or an empty file.
      output: the version of delmod is printed along with test results
         if the book is not empty.  Successful compilation and running of the
         program indicates that the modules are correct.

description
      Delmod is a collection of modules used by delila system programs.
      The easiest way to obtain a list of the modules is to run the module
      program using delmod for both sin and modlib (with dummy files for
      the other input).  There are a number of information modules, indicated
      by names beginning with 'info.'.  There are also a number of packages of
      modules that pickup other modules.  These begin with 'package.'.  You
      should note that some modules are constants, others types, etc.  These
      must remain in their proper location to allow compilation.

examples
      A good book to use to test delmod is ex0bk.

see also
      module

author
      Thomas D. Schneider and Gary D. Stormo

bugs
      none known

*)
(* end module describe.delmod *)
version = 'delmod 6.64 93 Jan 10 tds/gds'
```

```
(* begin module describe.diana *)                                          170
(*
name
   diana: dinucleotide analysis of an aligned book

synopsis
   diana(book: in, inst: in, dianap: in, da: out, output: out);

files
   book: the standard delila book that is to be analyzed
   inst: the instruction set that was used to make the book
   dianap: Parameters to control the program.  If there are two integers on the
      first line, then they determine the from-to range over which to do the
      analysis.  If from > to or the file is empty, the range from book and
      inst is used.
   da: the di-analysis file that contains the output triangular array.
          column 1: Tells what that row of data is.  There are three choices:
                n  : the column is a normal data element in the triangle.
                d  : the column is an element on the diagonal of the triangle,
                   where the two coordinates are equal.
                i  : the column contains the information content of a triangle.
  column 2: Tells what the dinucleotide is.  There are 16 dinucleotides
aa, ac, ag, at, ... , tt as well as 'in' or 'id', which denote
columns that contain information content.  'id' means that the
information is for the diagonal, while 'in' are off-diagonal.
Combining in with id gives the entire information triangle.
          column 3: The position on the sequence that corresponds to the x
                coordinate on a Cartesian graph.
          column 4: The position on the sequence that corresponds to the y
                coordinate on a Cartesian graph.
          column 5: A column of constants usable by xyplo.
          column 6: A column of constants usable by xyplo.
          column 7: The number of data points at a position
          column 8: The frequency of the dinucleotide in column 2 at position
                (column 3, column 4).  If the column is an information column,
                then this is the information at that position.
          column 9: One minus the frequency (or 1 - column 8).  If this is
                an information column, then this is the chi-square value.
  column 10: A column of constants usable by xyplo.  If this is in an
information row, then this column is the number of degrees of
        freedom at that position
          column 11: A column of constants usable by xyplo.
  column 12: A column of constants usable by xyplo.
  column 13: Information from column 8 normalized by dividing by
                the maximum possible information (4 for non-diagonal, 2 for
                diagonal).
  column 14: 1 minus column 13
  column 15: correlation coefficient for x to y on information (in or
                     id) rows
   output: error messages from the program

description
   Diana goes through a book and looks at relationships of dinucleotide pairs
   within the sequences.  The output of the program is in the form of a
```

```
   triangular array.  For every pair of coordinates, the frequency of the
   dinucleotide pair is tabulated.  The program also calculates the chi-square
   for the dinucleotide given the expected mono-nucleotide frequencies.  The
   correlation information is calculated as the information in the dinucleotide
   frequencies less the information in each of the two mononucleotides.  The
   output of the program may be sent into xyplo for graphics.  Note the
   distinction between the 'in' and 'id' information columns.  Information in a
   binding site is usually going to appear on the diagonal, so by making this
   distinction, one can eliminate the diagonal information peaks for statstical
   analysis with genhis.

documentation
   none

see also
   xyplo.p, alist.p, genhis.p

author
   R. Michael Stephens

bugs
  The program has no error correction for small sample size. It assumes that 2
  bits is the maximum uncertainty for single bases and that 4 bits is the
  maximum uncertainty for dinucleotides.

technical notes
*)
(* end module describe.diana *)
version = 1.77; (* of diana.p 1992 June 12
```

```
(* begin module describe.difint *)
(*
name
   difint: differences between integers

synopsis
   difint(input: in, output: out);

files
   input: a set of integers, one per line
   output: the difference between each integer and the previous one.

describe
   lines that begin with an asterisk ('*') are first copied to the output.
   then the difference between each integer in input and the previous one is
   given to the output.  the program acts as if the integer before the first
   one is zero.

author
   thomas dana schneider

bugs
   none known

*)
(* end module describe.difint *)
version = 1.05; (* of difint, 1986 dec 2 *)
```

```
(*
name
   digrab: diagonal grabs of diana data

synopsis
   digrab(input: in, ii: in, xyin: out, xyplop: out, output: out)

files
   input: User defines the value of n.
   ii:  output of the diana program, filtered for information lines only.
   xyin:  input to the xyplo plotting program, extracted lines.
   xyplop:  parameters controlling xyplo
   output: messages to the user

description
   This program extracts lines that have the form (x,x+n) from the da output of
   diana.  The result may be plotted with xyplo.

examples

documentation

see also
   diana.p, xyplop.p

author
   Thomas Dana Schneider

bugs

technical notes

*)
(* end module describe.digrab *)
version = 1.04; (* of digrab.p 1990 August 10
```

```
(*
name
   dirty: calculate probabilities for dirty DNA synthesis

synopsis
   dirty(dirtyp: in, distribition: out, xyin: out, output: out)

files
   dirtyp: parameter file.
      one line giving the number of random bases that will be used (r).
      one line giving the average number of changes desired (n)
   distribution: the distribution of numbers of changes at the
      peak for n
   xyin: Graphics output of the program.  The input to xyplo for plotting.
      The graph gives three curves against the independent variable p,
      which is the probability of getting the correct base and randoms
      is the number of random bases:

      o = probability of only one base changed, as randoms (1-p)p^(randoms-1)
      m = probability of one or more bases changed: 1 - p^randoms
      n = probability of n bases changed

      I have not found this output to be too useful; I concentrate
      on the distribution file.
   output: messages to the user

description
   If one is designing a randomized ("dirty") DNA synthesis, how
   heavily should it be randomized?  To use this program, pick the size
   of the region you want to randomize, r.  Then make a guess at the
   average number of changes you want over the region, n.  Put r and n
   into dirtyp and run the program.  Look at the distribution file.
   the line for n=0 is the frequency that you will get back the
   original sequence.  You must chose whether this is tolerable.  For
   example, when I synthesized the T7 promoters, I knew that I could
   find at least 1 promoter in 100 clones by toothpicking, and I was
   willing to toothpick thousands.  This way I was sure to get some
   positives, even if they were the original sequence.  (As it turned
   out, the frequency of functional promoters was much higher than
   1%.)  If you have a strong selection, you could make this a small
   number, by increasing the number of changes per clone.  With more
   changes per clone you will get more data from the randomization, so
   make it as high as you can tolerate.

   The program calculates the ratio of bases to random bases.  In the
   experiment described in the NAR paper, the technician put 4 drops of
   the appropriate base with 1 drop of the equiprobable mix.  This made
   the dirty bottle.

example

This is the analysis used in the NAR paper.  With the dirtyp file containing:
```

```
27    the number of random bases that will be used.                              175
4     the number of changes desired (n)

the distribution file is:
* dirty 2.38
* distribution of number of changes calculated from binomial
* 27 random positions
*  4 average number of bases changed
* p = probability of correct base = 0.85185185
* fraction of [base] : 0.80246914
* fraction of [random n] : 0.19753086
*
* ratio of [base] to [random N]: 4.06250000
*
* TO DO THE SYNTHESIS,
* add one part of an equimolar mixture of the 4 bases
* to 4.06250000 parts of the "wild type" base
*
* In the following table,
*    n = number of changes
*    f = frequency of n changes
*    s = running sum of frequencies f (should approach 1.0)
* In the first row, where n=0, f is the frequency of wild type sequences
*
n = 0   f = 0.01317741   s = 0.01317741
n = 1   f = 0.06187652   s = 0.07505392
n = 2   f = 0.13989473   s = 0.21494866
n = 3   f = 0.20274599   s = 0.41769465
n = 4   f = 0.21156103   s = 0.62925568
n = 5   f = 0.16924883   s = 0.79850451
n = 6   f = 0.10792679   s = 0.90643130
n = 7   f = 0.05630963   s = 0.96274093
n = 8   f = 0.02448245   s = 0.98722338
n = 9   f = 0.00898872   s = 0.99621210
n =10   f = 0.00281386   s = 0.99902596
n =11   f = 0.00075629   s = 0.99978226
n =12   f = 0.00017537   s = 0.99995763
n =13   f = 0.00003519   s = 0.99999282
n =14   f = 0.00000612   s = 0.99999894
n =15   f = 0.00000092   s = 0.99999986
n =16   f = 0.00000012   s = 0.99999999
n =17   f = 0.00000001   s = 1.00000000
n =18   f = 0.00000000   s = 1.00000000
n =19   f = 0.00000000   s = 1.00000000
n =20   f = 0.00000000   s = 1.00000000
n =21   f = 0.00000000   s = 1.00000000
n =22   f = 0.00000000   s = 1.00000000
n =23   f = 0.00000000   s = 1.00000000
n =24   f = 0.00000000   s = 1.00000000
n =25   f = 0.00000000   s = 1.00000000
n =26   f = 0.00000000   s = 1.00000000
n =27   f = 0.00000000   s = 1.00000000

see also
```

```
documentation

   @article{Schneider1989,
   author = "T. D. Schneider and G. D. Stormo",
   title = "Excess Information at Bacteriophage {T7} Genomic Promoters
            Detected by a Random Cloning Technique",
   year = "1989",
   journal = "Nucleic Acids Research",
   volume = "17",
   pages = "659-674"}

author
  Tom Schneider
  National Cancer Institute
  Laboratory of Mathematical Biology
  Frederick, Maryland
  toms@ncifcrf.gov

bugs
   n must be an integer

*)
(* end module describe.dirty *)
version = 2.38; (* of dirty, 1989 March 28
```

```
(* begin module describe.dnag *)
(*
name
      dnag: graphics of dna

synopsis
      dnag(bdna: in, dooin: out, output: out)

files
      bdna: b- form dna coordinates.  lines beginning with '*' are ignored.
         on each line following is the coordinate of one atom.
         the first character is the kind of group:
            * P = phosphate, D = deoxyribose
            * A = adenine, G = guanine, C = cytosine, T = thymine
         the next character is blank
         the next two characters are the atom and its number
         then the locations are given, separated by spaces:
            radius (angstrom) - angle (degree) - z axis (angstrom)
      dooin: graph of dna in doodle format
      output: messages to the user

description
      dnag generates a graph of DNA.

documentation
      B-DNA Cylindrical Polar Coordinates from
      S. Arnott and D. W. L. Hukins
      Biochem. and Biophys. Res. Comm 47: 1504-1509 (1972)
      "Optimised Parameters for A-DNA and B-DNA"

      M. Karplus and R. N. Porter
      Atoms & Molecules
      Benjamin/Cummings Publishing Co., Menlo Park, Ca, 1970
      p. 204-7, crystal radii

author
      Thomas D. Schneider

bugs
      The location of the strings may not be centered exactly in the circles.
To make this easy to adjust, two fudge factors (fudgex and fudgey) are provided
as constants.

*)
(* end module describe.dnag *)
version = 1.73; (* of dnag.p 1993 January 26
```

```
(*
name
   domod: doodle modules

synopsis
   domod(input: in, output: out)

files
   input: text.  portions surrounded by .PS and .PE are
      searched for function names.  when a function name is found,
      the parameters on the same line are read.
   output: copy of input text except that the functions detected
      during reading are translated into doodle commands.

description
      Domod contains the doodle modules.  Calls to the procedures
   cause the corresponding doodle command to be written to the output
   file.  Since this is the same as the input, the program only reformats
   the input.  That is, in UNIXease,
      domod<a>b
      domod<b>c
      diff b c
   shows no difference between b and c.  The program serves as a module
   library for the procedures that generate doodle commands.

see also
   doodle dosun

author
   Thomas D. Schneider

bugs
   domod does not copy correctly outside of pictures.  Inside
   of pictures it appears to read the entire demo and copy it to
   output correctly, such that domod<demo>a;domod<a>b;diff a b
   gives no differences.

technical note
   The globals picxglobal and picyglobal are updated, so
   a program that does graphics using these calls can use these
   variables to find out where it is.
*)
(* end module describe.domod *)
version = 1.40; (* of domod.p 1989 Aug 9
```

```
(*
name
   doodle: pascal graphics library and preprocessor for pic under unix

synopsis
   doodle(input: in, output: out)

files
   input: text.  portions surrounded by .PS and .PE are
      searched for function names.  when a function name is found,
      the parameters on the same line are read.
   output: copy of input text except that the functions detected
      during reading are translated into pic commands.

description
   Doodle is a preprocessor for the pic program.  (Yes you got it
   right... doodle is a preproprocessor for troff.)
   The pic preprocessor takes a series of commands and converts
   them to troff input under the unix operating system.
   Commands allow one to draw pictures and imbed them into
   text.  Doodle creates pic commands for things like lines
   and axes and spirals and other things.
      Doodle's main purpose is to be a testing shell for a general set of
   pascal graphics routines, available as modules.

see also
   the doodle manual, doodle.info, module

author
   Thomas D. Schneider

bugs
   none known

*)
(* end module describe.doodle *)
version = 1.95; (* of doodle, 1988 jan 6
```

```
(*
name
   dops: pascal graphics library and preprocessor for postscript

synopsis
   dops(demo: in, input: in, output: out)

files
   demo: a file for demonstration of the program.
      Start dops interactively.  Start a picture with
          .PS 81 2 2
      then type
          demo
      Graphics instructions will be read from the file 'demo', and the
      corresponding postscript will appear on the output.
      You can try instructions by hand.  Then type
          .PE
          ^d (control-d)
      to conclude.
   input: Graphics instructions.  Portions surrounded by .PS (with
      the appropriate parameters) and .PE (.PS =picture start and .PE =
      picture end) are searched for function names.  When a function
      name is found, the parameters on the same line are read.
   output: the functions detected within .PS to .PE
      are translated into PostScript graphics

description
   Dops converts the graphical instructions made by modules from
   domod.p and produces graphics in the language PostScript.

examples
   To demonstrate the 3-D graphics, use
     .PS 81 2 2
     test3d
     .PE
     (control-d to leave the program)
   A complete test file is called 'demo', which should be run
   non-interactively.

see also
   doodle.p, domod.p, dosun.p
   PostScript Language Tutorial and Cookbook,
   PostScript Language Language Reference Manual
   both from Addison Wesley, 1985
   demo - file that demonstrates all functions

references

@article{Schneider1982,
author = "T. D. Schneider
 and G. D. Stormo
 and J. S. Haemer
 and L. Gold",
```

```
title = "A design for computer nucleic-acid sequence storage, retrieval and
manipulation",
year = "1982",
journal = "Nucleic Acids Research",
volume = "10",
pages = "3013-3024"}

@article{Schneider1984,
author = "T. D. Schneider
 and G. D. Stormo
 and M. A. Yarus
 and L. Gold",
title = "Delila system tools",
year = "1984",
journal = "Nucleic Acids Research",
volume = "12",
pages = "129-140"}

author
   Thomas D. Schneider
   National Cancer Institute
   Laboratory of Mathematical Biology
   Frederick, Maryland
   toms@alum.mit.edu

bugs
   none known

technical note
   NONSTANDARD is a comment that means that this portion of the code
is dependent on non-standard pascal (or graphics) for its function.
*)
(* end module describe.dops *)
version = 2.63; (* of dops.p 1991 November 2
```

```
(* begin module describe.dosun *)
(*
name
   dosun: pascal graphics library and preprocessor for Sun graphics

synopsis
   dosun(demo: in, input: in, output: out)

files
   demo: a file for demonstration of the program.
      type 'demo' to run it.
   input: text.  portions surrounded by .PS and .PE are
      searched for function names.  when a function name is found,
      the parameters on the same line are read.
   output: copy of input text except that the functions detected
      during reading are translated into Sun graphics.

description
      Dosun is equivalent to doodle (see doodle.p) but produces
   output directly to the screen using Suncore graphics.

see also
   doodle.p, suncore graphics manual, domod.p

author
   Thomas D. Schneider

bugs
   none known

technical note
   NONSTANDARD is a comment that means that this portion of the code
is dependent on non-standard pascal for its function.
*)
(* end module describe.dosun *)
version = 2.17; (* of dosun, 1988 jan 13
```

```
(*
name
      dotmat: dot matrices of two books

synopsis
      dotmat(xbook: in, ybook: in, mlist: out, dotmatp: in, output: out)

files
      xbook: a book from the delila system

      ybook: a book from the delila system

      mlist: a dot matrix for each sequence pair between the two books.
         the "dots" are printed as numbers:
            1 means gt base pair
            2 means at base pair
            3 means gc base pair
         xbook sequences are written down the page and those of
         ybook go across the page.
         if mlist is wider than your printer, use the split program.

      dotmatp: parameters to control the mlist.
         if dotmatp is empty, default values are used.  otherwise
         if the first line begins with a "g" then g-u pairs are printed.

      output: messages to the user

description
      dotmat produces a dot matrix for all complementary base pairs
      between all pairs of sequences in the two books.  because a list of
      helices is not made, the program is much more efficient for short
      minimum helices than is the pair of programs helix and matrix.

documentation
      delman.use.comparison
      J. V. Maizel, Jr. and R. P. Lenk PNAS 78: 7665-7609 (1981)

see also
      helix, matrix, split

author
      thomas d. schneider

bugs
      none known

*)
(* end module describe.dotmat *)
version = 3.06; (* of dotmat 1986 dec 12 *)
```

```
(* begin module describe.dotsba *)
(*
name
   dotsba: dots to database

synopsis
   dotsba(dots: in,  database: out, output: out)

files
   dots: dot input format of sequences.
      First line is the header line for the database.
      Second line is the standard, not to be copied to the database.
      Following lines have a period (dot, '.') replacing bases that
      are the same as the standard or a different base.  There may be
      any number of spaces.  Following this is a bar (|).  Following
      the bar are other data to be copied to the database: clone number,
      primer for sequencing, and the date.
   database:  reformatted data ready for sites program
   output: messages to the user

description
   To convert from dots format to one the sites program can use.
   It should not have been necesary to do this, but Peter Papp didn't
   type the original sequences in unfortunately.

examples

documentation

see also
   sites.p

author
   Thomas Dana Schneider

bugs
   This is a stupid program.

technical notes

*)
(* end module describe.dotsba *)
version = 1.07; (* of dotsba.p 1990 December 9
```

```
(* begin module describe.encfrq *)
(*
name
      encfrq: encoded sequence frequency analysis

synopsis
      encfrq(encseq: in, cmp: in, fout: out, output: out)

files
      encseq: the output of the encode program
      cmp: a composition from the comp program.
      fout: frequency tables for each parameter set.  these are followed
         by z values for each frequency.  if cmp is empty, then equal
         frequencies are assumed.
      output: messages to the user.

description
      the frequency of each n-tide (mono- or di- or etc) is displayed in
      fout.  the actual number of sequences passing through a particular
      n-tide and position (ie, a parameter window) is taken into account.
         a second set of tables of z values are also presented.
      these are calculated from the composition provided in comp (p, the
      probability of obtaining the n-tide), the actual number of
      occurences (b) and the number of sequences at that position (n).
      the distribution of b can be described as a binomial distribution,
      with mean (m) np and standard deviation (s) sqrt(npq).  b is then
      normalized to obtain z: z=(b-m)/s.  if n is large, then z is
      normally distributed, and the probabilities can be found on any
      table for the normal distribution (use a two tailed test).  a rule
      of thumb for when the normal distribution can be used is that
      both np and n(1-p) should be greater than 5.  locations that violate
      this rule are marked with a '*'.
         locations of the z table that contain z values of 3 or greater are
      displayed to the right of the z table.  since these look somewhat
      like a dna footprint, they are called z-footprints.  the output
      for dinucleotide z-footprints is very wide, so one must split
      it up using the split program.  recommended values for splitp are
      p/14/112/4, where the slash means "start a new line".

see also
      encode, comp, split

author
      thomas d. schneider

bugs
      none known

*)
(* end module describe.encfrq *)
version = 1.52; (* of encfrq.p 1993 Jan 27
```

```
(*
name
      encode: encodes a book of sequences into strings of integers

synopsis
      encode(inst: in, book: in, encseq: out, encodep: in, output: out)

files
      inst: the instructions generating the book; for aligning the sequences
      book: the sequences to be encoded
      encseq: the encoded sequences
      encodep: parameter file for describing how the sequences are to be
            encoded.  see description section for format of this file
      output: for messages to the user

description
         this program is used to encode a book of sequences into a string of
      integers. each sequence in the book is encoded into a single string of
      integers (ended by an 'end of sequence' symbol) according to the user
      specified parameters, which are in the file 'encodep'.
         the parameters are stored as a list of parameter records, of which
      there may be any number.  each parameter record has five lines of
      information which it must include (all i's and j's are integers):
      1.  i j specify the nucleotides, relative to the aligned base,
              over which this parameter record is to operate; these may
              be any integers, but i <= j is required;
      2.  i   is the size of the windows to be encoded; within the window
              the number of each oligonucleotide of length 'coding' are
              determined and printed as part of the total sequence vector;
      3.  i   is the shift to the next window to be encoded;
      4.  i : j1 j2 j3 ...  is the 'coding'-level and arrangement; the
              'coding'-level, i, is the number of nucleotides in the oligos we
              are counting, i.e., 1 means monos, 2 means dis, ...;  if i > 1
              then we can also skip bases between the ones we are encoding;
              if the i is followed next by a colon, there must be i-1 integers
              (j1..j(i-1)) which specify the number of bases to be skipped
              between the ones which are encoded; for example, if we have the
              sequence xyz and we are interested in the di-nucleotides we can
              get the xy by the parameter '2 : 0', or we could get the xz by
              parameter '2 : 1'; if there is no colon all the skips are
              assumed to be zero;
      5.  i   is the shift to the next coding site within the window;
              this allows us to encode only some of the oligos within a window,
              such as only those that are in-frame;
      multiple parameter records can be concatenated in the encodep file
      and then each sequence in the book will be encoded according to each
      parameter record into a single vector of integers.

documentation
      delman.use.encode, delman.use.aligned.books

author
      gary stormo
```

```
bugs
     none known
*)
(* end module describe.encode *)
version = 1.28;  (* of encode.p 1991 Jan 11
```

```
(* begin module describe.encsum *)
(*
name
      encsum: sum of the vectors of encoded sequences

synopsis
      encsum(encseq: in, sum: out, output: out)

files
      encseq: the file of encoded sequences; this is the output of
            the program 'encode'
      sum: the output of this program
      output: for messages to the user

description
      this program takes as input a file of encoded sequences, from the
      encode program, and sums the individual sequence vectors into one
      vector of their sums.  this is useful for doing histograms or
      compositions of many sequences.

see also
      encode

author
      gary stormo

bugs
      none known
*)
(* end module describe.encsum *)
version = 1.20; (* of encsum.p 1991 apr 3
```

```
(*
name
   epsclean: clean an eps file

synopsis
   epsclean(input: in, output: out)

files
   input: eps file from microtex scanner
   output: cleaned eps file ready to print

description

   1. On the Mac:  scan an image into the mac using the microtex B&W scanner
   software.  Use:
      EPS format
      600 dpi
      2x2 high contrast
      line art

   2. Use the fetch program to move it to Unix with parameters:
      binary
      rawdata

   3. On Unix, the file comes over with all control-M's instead of newlines (ie
   instead of ascii 012).  However, if all controm-M's were converted to
   newlines, the image would be wrecked because it apparently contains control
   M's!

   4. Run the file through this program to make it useable.  This program works
   by finding the second occurance of the word 'dopic'.  Before this point,
   control-M's are converted to ascii 012, after this point they are left alone.
   The first occurance of 'dopic' is the definition of the "do picture" routine
   that defines the image.  The second one calls the routine, so the raw
   data follow that point.

examples

documentation

see also

author
   Thomas Dana Schneider

bugs

technical notes

*)
(* end module describe.epsclean *)
version = 2.09; (* of epsclean.p 1993 January 26
```

```
(*
name
      ev: evolution of binding sites

synopsis
      ev(list: out, all: inout, evp: in, output: out)

files
      list: a record of the evolutionary events that occurred
         See the description of evp for the kinds of data that can be printed.
         The data may be graphed as desired using the xyplo program.
         The first few lines of the file form an informative header.
         All of these lines begin with an asterisk, '*'.
         The data themselves are organized into individual lines
         broken by spaces into a set of tokens.  These are described in
         the header.
      all: all the variables, genome sequences and genetic structure
         to allow continuation of the evolution
      evp: parameters to control the program, one per line:
         Number of creatures (c, integer).
         Number of potential binding sites per creature (G, integer).
         Number of sites per creature (gamma, integer).
         Width of the recognizer in bases (width, integer).
         Bases per integer of the recognizer weights (bpi, integer).
         Mutation rate in hits per creature per generation (mu, integer).
         Seed: a real number between 0 and 1 used to start the random
            number generator.  The date and time is used if this
            number is outside 0 to 1.
         Cycles: number of additional generations to run (cycles, integer).
         Display interval:  for example, 10 means every 10th generation.
         Display control: the first 7 characters on the line control
            the kind of data printed to the list file:
               a = display average number of mistakes and the standard
                   deviation for the population.
               c = display changes in the number of mistakes.
                   The current Rsequence is given if r (below) is turned on.
                   This allows graphs of Rsequence vs mistakes to be made.
               g = display genomic uncertainty, Hg.  If this deviates much
                   from 2.0, then the simulation is probably bad.
               i = display individuals' mistakes
               o = display orbits: information of individual sites is shown
               r = display information (Rsequence, bits)
               s = current status is printed to the output file.
            These may be in any order.   Any other characters (eg, blanks) are
            ignored.
         Selecting: boolean. If true, then the organisms are sorted
            by their mistakes.   If false, then the organisms are randomly
            sorted.  Normally this should be 'true', but it does allow one
            to switch the selection off suddenly and watch things like no
            evolution and the decay of existing patterns by entropy increase.
            Selecting is true unless the first character on the line is an 'f'.
         StorageFrequency:  The frequency (every so many generations) with which
            to store a copy of everything in the all file.  If the computer
```

```
          crashes part way through a long run, then the run can be continued
          from the last storage.  Of course, there a storage is always made
          at the end of the evolution.
      output: messages to the user, including warnings about conditions,
         If the display control in evp (see above) is includes 'o', then
         the generation number and the range of mistakes are given.
         If the display control includes 'a', then the mean and standard
         deviation of the mistakes are also given.

description
      A population of evolving creatures is simulated.  Each creature
      consists of a genome made of the 4 bases.  All creatures have
      a certain number of binding sites, and the recognizer for the
      sites is encoded by a gene in each genome.  The genomes are completely
      random at first.  The recognizer of each creature is translated
      from the gene form to a perceptron-like weight matrix.  This
      matrix is scanned across the genome.  The number of mistakes
      made is counted.  There are two kinds of mistake:
          how many times the recognizer misses a real site
      and
          how many times a non-site is detected by the recognizer.
      These are weighted equally.  (If they were weighted differently
      it should affect the rate but not the final product of the simulation.)
      All creatures are ranked by their number of mistakes.  The half of the
      population with the most mistakes dies; the other half reproduces
      to take over the empty positions.  Then all creatures are
      mutated and the process repeats.
         The integer weights of the recognizer are stored as base 4
      numbers in twos complement notation.  a=00, c=01, g=10, t=11.
      If 'bases per integer' were 3, then aaa encodes 0, acg is 6, etc.
      txx and gxx (where x is any base) are negative numbers; ttt is -1.
         The threshold for recognition of a site is encoded in the genome
      just after the individual weights.  It is encoded by one integer.

documentation
      for information calculations, see:
      Schneider et al, J. Mol. Biol. 188: 415-431 (1986)

examples

1. A lovely evolution can be had with the following evp:
********************************************************************************
   32 NUMBER OF CREATURES
 1024 NUMBER OF BASES PER CREATURE, G
   64 NUMBER OF SITES PER CREATURE, gamma
    6 WIDTH OF THE RECOGNIZER IN BASES
    5 BASES PER INTEGER OF THE RECOGNIZER
    1 MUTATION RATE IN HITS PER CREATURE PER GENERATION
 0.50 SEED FOR THE RANDOM NUMBER GENERATOR
40000 CYCLES
   10 DISPLAY INTERVAL
cgrs567 a=av, c=change, i=indivls, g=Hg, r=Rs, o=orbit, s=status
true  SELECTING
********************************************************************************
```

```
The list file may be plotted with this parameter file for xyplo, the xyplop:
*****************************************************************************
1 2       zerox zeroy         graph coordinate center
x 0 40000 max       (character, real, real) if zx='x' then set xaxis
y -1 6.00 zy min max          (character, real, real) if zy='y' then set yaxis
10 28     xinterval yinterval number of intervals on axes to plot
7 7       xwidth    ywidth    width of numbers in characters
0 2       xdecimal  ydecimal  number of decimal places
6 6       xsize     ysize     size of axes in inches
generation
Rsequence (bits) | Hg (bits) near 2 | mistakes/gamma are connected circles
n         zc                  if zc='c' then a crosshairs put on zero of x and y
n 2       zxl base            if zxl='l' then make x axis log to the given base
n 2       zyl base            if zyl='l' then make y axis log to the given base
          --------------------------------------------------------------------
1 3       xcolumn   ycolumn   columns of xyin that determine plot location
2         symbol column       the xyin column to read symbols from
0  0      xscolumn  yscolumn  columns of xyin that determine the symbol size
          --------------------------------------------------------------------
          symbol to plot      'c'=circle, 'b','d'=box, 'x', '+', 'I', 'f', 'g'
r         symbol flag         character in xyin that indicates that this symbol
0.05      symbol sizex        side in inches on the x axis of the symbol.
0.05      symbol sizey        as for the x axis, get size from yscolumn
cl 0.05   connection (example for connection is c- 0.05 for dashed 0.05 inch)
n  0.05   linetype  size      linetype l.-in and size of dashes or dots
          --------------------------------------------------------------------
          symbol to plot      'c'=circle, 'b','d'=box, 'x', '+', 'I', 'f', 'g'
g         symbol flag         character in xyin that indicates that this symbol
0.05      symbol sizex        side in inches on the x axis of the symbol.
0.05      symbol sizey        as for the x axis, get size from yscolumn
cl 0.05   connection (example for connection is c- 0.05 for dashed 0.05 inch)
n  0.05   linetype  size      linetype l.-in and size of dashes or dots
          --------------------------------------------------------------------
c         symbol to plot      'c'=circle, 'b','d'=box, 'x', '+', 'I', 'f', 'g'
c         symbol flag         character in xyin that indicates that this symbol
0.05      symbol sizex        side in inches on the x axis of the symbol.
0.05      symbol sizey        as for the x axis, get size from yscolumn
cl 0.05   connection (example for connection is c- 0.05 for dashed 0.05 inch)
n  0.05   linetype  size      linetype l.-in and size of dashes or dots
          --------------------------------------------------------------------
.
          --------------------------------------------------------------------
. 0 6 0.10
. 0 5 0.10
- 0 4 0.10
. 0 3 0.10
. 0 2 0.10
. 0 1 0.10
l 0 0 1
*****************************************************************************
Note that dotted lines are placed across the graph for each bit, but a
dashed line is put for Rfrequence (= 4 bits).  The vertical axis represents
the three kinds of data in the list file.
```

```
see also
      evd, xyplo

author
      Thomas Dana Schneider

bugs
      none known

*)
(* end module describe.ev version = 2.50; (@ of ev 1988 oct 6 *)
version = 3.21; (* of ev.p 1989 December 14
```

```
(* begin module describe.flag *)
(*
name
      flag: points out excessively long lines

synopsis
      flag (fin: in, fout: out, output: out)

files
      fin: a text file; typically pascal source code.
      fout: the first line of fin followed by a list of the lines which are
            too long.  the list gives the line number of each line, the line
            itself and a flag on the last acceptable character.
      output: the number of lines in fin which contain more than 80 characters.
description
      during transportation of files from one computer to another, lines
      longer than 80 characters are often truncated to 80 characters to make
      'card images' on the tape.  this byzantine practice is left over
      from the days when cards were the state-of-the-art in talking to
      computers.  since the tape does not know what a 'card image' is,
      and since cards are going the way of the passenger pigeon,
      this is like equipping a nuclear oil tanker with oars.  maybe
      someday things will be different, but until then, flag exists
      and will detect long lines, allowing one to fix a program or file
      before transportation.  note: trailing blanks on each line are
      ignored.

author
      john hoffhines and tom schneider

bugs
      none known

technical notes
      the constant 'maxline' defines the number of characters accepted
      on each line.  we recommend that maxline be set to 80 because this
      is the standard number of characters on a punched card.

*)
(* end module describe.flag *)
const version = 1.14; (* of flag.p 1991 Feb 20
```

```
(* begin module describe.frame *)
(*
name
      frame: evaluator of potential reading frames

synopsis
      frame(test: in, norm: in, result: out, output: out)

files
      test: encoded vectors of the sequences to be tested for reading frames
      norm: encoded vector of the sequences used as the standard for testing
      result: the results of the tests; each sequence from test is evaluated
         for each of the three possible reading frames
      output: for messages to the user

description
      this calculates correlation coefficients between the standard and each
      of the three possible frames of the test sequences.  the sequences must
      be encoded so that each of the oligos (of whatever length is desired)
      are counted in each of the three frames.

examples
      the files framet and framen are examples of test and norm

documentation
      delman.use.frame

see also
      encode

author
      gary d. stormo

bugs
      none known
*)
(* end module describe.frame *)
version = 1.16;  (* of frames 1986 dec 9
```

```
(* begin module describe.frese *)
(*
name
   frese: frequency table to sequ

synopsis
   frese(fresep: in, sequ: out, output: out)

files
   fresep: input frequency table (parameters to the program)
     a set of integers, 5 per line, representing first the coordinate
     and then the numbers of a,c,g and t to use.
   sequ: sequences which could have produced the fresep frequencies,
     ready for input to makebk.
   output: messages to the user

description
   Frese converts a table of frequencies to a set of raw sequences
   so they may be analyzed.  The raw sequences have the same frequencies,
   but, of course, are not the same as the original sequences.

examples

documentation

see also
   makebk.p

author
   Thomas Dana Schneider

bugs

technical notes

*)
(* end module describe.frese *)
version = 1.01; (* of frese.p 1991 November 30
```

```
(*
name
      gap: gaps in aligned listing of a book

synopsis
      gap(inst: in, book: in, gapp: in, data: out, output: out)

files
      inst: delila instructions of the form 'get from 56 -5 to 56 +10;'
         (This file may be empty, in which case the sequences will be
         aligned by their 5' ends.)
      book: the book generated by delila using inst
      gapp: parameters to control the program.  If empty, the range of the
         instructions are used.  Otherwise,
         1. The first line contains one line with two integers
         defining the range to find gaps in.  This allows one to have a wide
         alignment, but look only at a portion.  (This is equivalent to the
         alist display range.)
 2. If the first character of the second line is 'p' the piece
 information is given in the data list.
         3. minimum number of gaps to report
         4. modulus: integer. Gaps are reported if if (gap mod modulus) = 0.
            For example, if modulus = 5, only gaps of 5 long are reported.
            To get all gaps, use modulus = 1.
      data: the gap listing.  First column is the number of gaps.
 If the sequences are numbered, the second column is the sequence
 number.
         If the display is on (2nd parameter), then the piece name is given,
         followed by the coordinate of the zero base.
      output: messages to the user

description
      Gap is useful for determining the distribution of gaps in an aligned set.
      The pieces in the book are aligned according to the instructions in
      file inst, and listed in the list file.  Each piece is identified.

example

documentation

see also
   alist.p

author
      Thomas D. Schneider

bugs
   as in alist.p

technical notes

*)
(* end module describe.gap *)
```

```
(*
name
      genhis: general histogram plotter

synopsis
      genhis(data: in, histog: out, genhisp: in, output: out)

files
      data:  File of numbers to be histogrammed.  Header lines that begin with
 '*' may be copied from this file to 'histog' or may be skipped.  The
 column from which to read the data may also be specified.  See the
 description of the file 'genhisp' to see how to do this.  Once the
 data region has begun, (that is, there is at least one non '*' line),
 then lines that begin with '*' are also skipped.
      histog: the output histogram.  contains the header lines copied from
         file 'data', plus data about the numbers (min, max, mean, variance
         and st. dev.), and the plot.  may also contain a standard plot.
      genhisp: parameter input file.  this is used to change any of the
         parameters from default values.  any may be changed and they can
         be specified in any order.  the first character on a line tells
         what parameter is to be set, the other information sets it.  the
         parameters that can be changed, and their line codes:
         h - sets header reading;  this is followed by two integers, the
            first specifying the number of lines to copy and the second the
            number of lines to skip;  if the first number is <0 those lines
            beginning with '*' are copied;  default is -1 0.
         c - sets the column of data that is to be analyzed and plotted;
            the default is column 1;  note: a column is any string
            of nonblank characters; columns are separated by blanks;
         p - sets the standard plot;  poisson and gaussian plots are
            available and are specified by following the p by either p or g.
         x - sets the x-axis scale; this is to be followed by either an n
            or an s, and then a number;  if n, then the number of intervals
            on the x-axis is set; if s, then the size of intervals is set;
            default is to set the number of intervals to constant 'defslots'.
         r - sets the range of data to be plotted;  this is followed by two
            numbers which specify the subrange of the data for which the plot
            is desired;  default is to plot all the data.
      output: for messages to the user.

description
      This program takes numerical data from a file and plots a histogram of
      those data.  It also calculates the min, max, mean and variance of the
      data.  If desired, the user may get a standard plot, based on the mean
      and variance, plotted along with the data.  The user may specify the size
      or number of intervals on the x-axis.  The y-axis is automatically scaled
      to fit on a page.  The scaling factor is reported to the user.

example
      try file datat7
      with genhisp of
         x n 20
         p g
```

```
author
      Gary Stormo

bugs
      Try different x axis intervals:  regular spikes can be data artifacts!

technical notes
      The constant 'pageheight' is used to set the scaling factor so that
      the plots do not exceede the size of a page.
*)
(* end module describe.genhis *)
version = 1.67;  (* of genhis.p 1992 November 16
```

```
(* begin module describe.genmod *)
(*
name
      genmod: genbank access modules

synopsis
      genmod(entries: in, output: out)

files
      entries: a set of genbank entries for a given organism
      output: messages to the user and tests of the modules

description
      these are modules containing procedures to access genbank
      entries.

author
      thomas d. schneider

bugs
      none known

*)
(* end module describe.genmod *)
version = 1.33; (* of genmod, 1986 feb 4
```

```
(*
name
      genpic: convert genhis output to pic input

synopsis
      genpic(histog: in, genpicp: in, picin: out, output: out)

files
      histog: the output of the genhis program
      genpicp: parameters to control the histogram are one per line.
            if they are missing, defaults are used. all are in inches.
         boxwidth; width of the histogram boxes.
         boxheight; height of the histogram boxes.
         intervalsize; the space for the interval number.
         histogramvalue; the space for the histogram value.
         boxshift; how much to shift the boxes up relative to the numbers.
         ifield: number of characters devoted to the interval
         idecimal: number of characters devoted to the interval's decimal places
         nfield: number of characters devoted to the number of numbers
      picin: the data in histog are converted to PostScript
      output: messages to the user.

description
      The genhis program generates a histogram in simple
      character format.  The program genpic converts this
      simple histogram into PostScript commands.  Therefore, one can
      imbed output from genhis in text of a paper.

author
      Thomas D. Schneider

bugs
      none known

technical note
       defaults for the parameters are in module genpic.const.

*)
(* end module describe.genpic *)
version = 2.01; (* of genpic.p 1992 November 16
```

```
(* begin module describe.gentst *)
(*
name
   gentst: test random generator

synopsis
   gentst(gentstp:in, data: out, output: out)

files
   gentstp: parameter file controlling the program.
      Three numbers, one per line:
         seed: random seed to start the process
         total: the number of numbers to generate
         components: the number of random numbers between 0 and 1 to add
            together to generate the total
   data: the input file for genhis.  this is a set of numbers which
      should have gaussian distribution if the random number generator
      is a reasonable one.  It will be N(0,1), a normal distribution
      with mean 0 and standard deviation 1.
   genhisp: control file for genhis
   output: messages to the user

description
   test of a random number generator by creating a gaussian
   distribution of numbers for plotting by genhis

example
  seed := 0.5;
  total := 10000;
  components:= 100;

see also
   tstrnd, genhis

author
   thomas d. schneider

bugs
   none known

technical notes
   the constant n in procedure randomtest determines how many times
   the random number generator will be in a series of tests.  if n
   is small, the the test will be poor, if it is large then the test may
   take a long time.

*)
(* end module describe.gentst *)
version = 3.12; (* of gentst.p 1993 Jan 27
```

```
(* begin module describe.helix *)                                              204
(*
name
   helix: find helices between sequences in two books

synopsis
   helix(xbook: in, ybook: in, hlist: out, helixp: in, output: out)

files
   xbook: a book from the delila system
   ybook: a book from the delila system
   hlist: a list of helices between pieces in xbook and ybook.
      the first line is the program identification
      the second two lines are the x and y book titles
      the third line gives the minimum length or the maximum energy
          of helixes recorded
      the fourth line states whether or not g-u pairs are allowed
      the fifth line states whether or not energies are printed
      the following lines are the helices
      breaks between sequences are indicated.
   helixp: parameters that control the helix list.
      if helixp is empty, default values are used.
      otherwise, the file must contain three lines:
      1. if this number is a positive integer, it specifies the minimum length
         in base pairs of the helixes written in hlist.  if it is a negative
         real number, it specifies the maximum energy in kcal of the helixes
         written.
      2. if the first character is a "g" then g-u pairs are allowed,
         otherwise not.
      3. if the first character is an "e" then the energy of each helix
         will be written in hlist.
   output: messages to the user

description
   All sequences in xbook are compared to all sequences in ybook.
   The complementary helices (of some minimum length and longer or of
   some maximum energy or less) are listed in hlist by the 5' ends of the
   helix on both sequences.  This information, along with the length of
   the helix, determines the location of the helix.  One can allow g-u
   pairing if desired.  If the helix lengths desired are very short,
   it is better to use dotmat (see "technical notes" below).
      The new Rules are now used to calculate the helix.

documentation
   delman.use.comparison

   J. V. Maizel, Jr. and R. P. Lenk PNAS 78: 7665-7609 (1981)

   Tinoco et al. Nature New Biology vol 246 pp 40-41, 1973.

   S. M. Freier, R. Kierzek, J. A. Jaeger, N. Sugimoto,
   M. H. Caruthers, T. Nelson, and D. H. Turner,
   "Improved free-energy paramters for predictions of RNA duplex stability"
   PNAS 83: 9373-9377 (1986)
```

```
see also
   matrix, dotmat, keymat

author
   Thomas D. Schneider

bugs
   GU pairs and bulges are not done using the new data.
   An option for pair-wise (rather than multiplicative) comparisons
   of sequences would be nice.

technical notes
   The shortest length helix ever recorded in hlist is determined
   by the constant absminlength.  This overrides the parameters.

*)
(* end module describe.helix *)
      version = 3.23; (* of helix 1990 December 21 *)
```

```
(* begin module describe.hexbin *)
(*
name
   hexbin: convert hex to binary

synopsis
   hexbin(input: in, output: out)

files
   input:  hexadecimal representation of an image, PostScript
   shape: First line contains two characters to skip and then
      two integers, the width and height of the image.
   output: binary representation of an image

description
   To allow one to work with a PostScript hex image in binary format
   it is converted.

examples

documentation
   PostScript red book p. 170

see also

author
   Thomas Dana Schneider

bugs

technical notes

*)
(* end module describe.hexbin *)
version = 1.07; (* of hexbin.p 1991 October 17
```

```
(* begin module describe.hist *)
(*
name
      hist: make a histogram of aligned sequences.

synopsis
      hist(inst: in, book: in, hst: out, histp: in, output: out);

files
      inst: the instructions which generated the book, used to
            align the sequences; the format of the instructions must be
            'get from # -x to # +y;' - the alignment is done on the base #.
            if this file is empty the sequences are aligned by their 5
            prime ends.
      book: the sequences, not longer than 'dnamax' (see technical note);
      hst: the histogram table, giving the position occurence
            of all oligonucleotides from oligomin to oligomax  in length
            (see file histp).  note - if the histogram is wider than can
            be printed on a page, use the program split to print hst.
      histp: to set the length of oligonucleotides searched for; contains two
            integers, one per line, the first specifying oligomin
            (the length of the shortest oligonucleotide which is searched for),
            the second oligomax (the longest oligonucleotide searched for,
            which cannot be greater than 4);
            note: if oligomin is zero, then the bases are counted.
      output: for error messages.

description
      makes a histogram of the occurences of oligonucleotides at positions
      relative to some aligned base.  this is done for all oligonucleotides
      with lengths from  'oligomin' to 'oligomax', set by file histp.

see also
      split, align, achsq

author
      gary stormo

bugs
      none known

technical note
      the constant 'dnamax'  from the module book.const may be too
      large for efficient use of this program.  if you do not expect to do
      histograms on aligned sequences of greater than, say, 120 nucleotides
      you can go into the source program (hists) and change 'dnamax' before
      compiling.
*)
(* end module describe.hist *)
version = 4.24; (* of hist.p 1992 Nov 9
```

```
(* begin module describe.histan *)
(*
name
      histan: histogram analysis.

synopsis
      histan(hst: in, cmp: in, chisq: out, output: out)

files
      hst: the histogram input; is the output of program hist;
      cmp: a composition input; is the output of program comp;
      chisq: the chi-squared analysis output;
      output: for user messages;

description
      histan determines the chi-squared values at each position for a
      histogram.  the observed values come from the histogram.  if a
      composition is provided the expected values come from that, otherwise
      the expected values assume equal frequencies of all bases.  the
      chisquared is calculated for each level of oligonucleotide (i.e.,
      monos, dis, tris) for which the histogram data exists.

see also
      hist, comp

author
      gary stormo

bugs
      none known
*)
(* end module describe.histan *)
version = 4.21;  (* of histan.p 1992 Nov 9
```

```
(*
name
      indana: analysis of an index

synopsis
      indana(ind: in, ana: out, subind: out, indanap: in, output: out)

files
      ind: an index produced by the index program.  it must not be a
         teaching index.
      ana: a histogram of the similarities of the index along with the
         mean, standard deviation and frequency distribution of the
         the similarities.
      subind: portions of the index selected by the parameters in indanap.
         pairs (or adjacent sets) of lines of the index are printed.  the
         similarities of the original index are maintained.  this means that
         the first similarity of a pair (or set) is not a reflection of the
         similarity to the line above it.  the ones that are 'true'
         are marked with an asterisk [*].
      indanap: parameters to control indana, containing 3 lines:
         1. the lowest similarity to put into subind
         2. the highest similarity to put into subind

description
      An index is usually quite large, so it is difficult to look at by hand.
      Indana allows one to select a portion of the index by various criteria.
      The portion is called a "sub-index".  If the original book contained a
      number of highly similar oligo- nucleotides, then the histogram of
      similarities will show a spike of high similarities.

see also
      index

author
      Thomas Schneider

bugs
      none known

*)
(* end module describe.indana *)
version = 5.24; (* of indana.p 1992 September 18
```

```
(*
name
      index: make an alphabetic list of oligonucleotides in a book

synopsis
      index(book: in, ind: out, indexp: in; output: out)

files
      book: the book of sequences to be indexed
      ind: the alphabetized index to the book
      indexp: parameters to control index.  if this file is empty, then
         default values are used.  otherwise there may be 4 or 5 lines:
         first line: the number of bases in the alphabetizing window
         second line: the number of bases to print before the central window
         third line: the number of bases to print in the central window
         fourth line: the number of bases to print after the central window
         fifth line: if the first letter is a 't', then the index
            will run in a teaching mode.  do not use this mode on large books.
         sixth line: if the first letter is 'f' then only the first
            oligo of each sequence is used for alphabitization.  This produces
            a drastic reduction in the number of oligos sorted.  It is
            meant to be used to sort aligned sequences, to see if there are
            identical copies.
      output: messages to the user

description
      The index program generates an index of oligonucleotide fragments in a
      book.  The first base of the alphabetizing window is stepped across all
      bases of the sequence, creating a list of overlapping oligos and their
      positions.  The oligos are then sorted along with their positions.  Three
      printing windows allow one to look at bases before the first base, from
      the first base some distance on (this is not the alphabetizing window)
      and a third set even further 3'.  It is not inefficient to make the
      alphabetizing window large when there are no long repeats in the
      sequences (as when comparing two similar genes).  Following the printing
      windows are: the sequence number of the piece in the book (provided by
      delila); the position of the first base; the orientation of the oligo;
      and the similarity.  This last item is the number of bases that an oligo
      matches the previous oligo in the index, up to the point that they
      differ.  High similarity means a repeat.

examples
      The index can be used to locate restriction enzyme sites, by simply
      'looking them up'.  It has the advantage that when new enzymes become
      available, one does not need the computer to locate their sites.  Direct
      repeats will show up as high similarity oligos, and if one gets the
      complement along with a sequence in a book (using delila) then inverted
      repeats can be found.  The first column of the alphabetizing window
      contains all the mononucleotides; the first two, the di's, etc.

documentation
      L. J. Korn, C. L. Queen and M. N. Wegman, PNAS 74: 4401-4405 (1977)
```

```
see also
      search, helix, delila, delman.use.comparison

author
      Gary Stormo and Thomas Schneider

bugs
      One cannot sort more sequence than can fit into the computer memory.

technical notes
      The constant mapmax determines the maximum number of bases indexed.
*)
(* end module describe.index *)
version = 9.24; (* of index.p 1992 September 18
```

```
(*
name
      instal: delila instruction alignment

synopsis
      instal(xbook: in, ybook: in, shlist: in, inst: in,
             rinst: out, sinst: out, list: out, output: out)

files
      xbook: a delila system book containing one piece used to
         align the ybook pieces.
      ybook: a delila system book containing pieces to be aligned by
         the piece in xbook.
      shlist: the output of the sorth program.  these sorted helixes
         must have been generated using helix(xbook,ybook,hlist,...) and
         then sorth(hlist,shlist,...,[.../1/...]).  that is, sorth must
         have been used to select only the top 1 helix from hlist.
      inst: the instructions used to generate ybook (or a comparable set
         of instructions that correspond to the ones for ybook).
      rinst: reduced instructions: those instructions from inst that
         have a unique helix in shlist.  in other words, inst is copied to
         rinst only for instructions that have a unique alignment.
         the other instructions are also copied, but surrounded by
         comment delimiters to neutralize them.  neutrialized
         instructions are followed by a delila instruction that will
         maintain the original piece numbers.
      sinst: shifted instructions: the instructions written to rinst are
         realigned by the helixes of shlist.  the new alignment
         is the coordinate where the 5 prime end of the xpiece
         would lie on each y piece.
      list: progress of the realignemnt.
      output: messages to the user.

description
      the purpose of instal is to automatically realign a set of instructions.
      for example, if one has a set of instructions that define the
      initiation codon of some procaryotic ribosome binding sites,
      one may want another alignment by the shine and dalgarno.  to
      do this, the following steps are needed:
      1. the instructions (inst) are converted to a book (ybook) using
      delila.  instructions that define the 3 prime end of the 16s rrna are
      written and used to create xbook.
      2.  potential helixes between xbook and ybook are found with the
      helix program, making an hlist.
      3. the strongest helixes of the hlist are selected using the
      sorth program.  thus each piece of ybook has a unique (or no)
      helix associated with it.
      4. instal is used to alter the original instructions.  instructions
      (pieces) with no unique helix are neutralized by putting them in
      comments to in both rinst and sinst.  in addition, the instructions
      of sinst are 'shifted' so they are aligned by the 5 prime end of xpiece.

see also
```

```
      delila, helix, sorth

author
      thomas dana schneider

bugs
      none known

technical notes
      the largest shift that is recorded is specified by the
      constant absshift

*)
(* end module describe.instal *)
version = 1.47; (* of instal 1985 may 5
```

```
(* begin module describe.kenbk *)                                              214
(*
name
      kenbk: make a book from a file of sequences of sequences provided by Kenn
      Rudd

synopsis
      kenbk(sequ: in, book: out, output: out, input: intty)

files
      sequ: file of sequences in Kenn Rudd's format.

 That format consists of lines and sequence.  A line starts with the
 '>' character.  This is followed by the sequence name then one or more
 spaces.  Then the expected size of the sequence is given.   The next
 line begins the sequence, in capital letters.  The next sequence is
 indicated by another '>'.

      book: the output file containing the sequences and the necessary
         information  for it to be a proper book.  the user types in the
         required information after prompts from the program.

      output: for messages and queries to the user;

      input: interactive input.

description
      kenbk takes a file of raw sequences (sequ) in Kenn Rudd's format
      and converts that into a proper delila book format, getting the
      title of the book from the user.

see also
      rawbk

author
      Thomas Schneider

bugs
      Delila cannot handle N's so they are converted to A's.  This
      should not affect searches much.
*)
(* end module describe.kenbk *)
version = 1.13; (* of kenbk.p 1991 May 31
```

```
(* begin module describe.kenin *)
(*
name
   kenin: create Delila instructions from Ken's all.gen instructions

synopsis
   kenin(allgen: in, inst: out; output: out)

files
   allgen:  gene instructions of the form provided by Kenn Rudd:
      piecename l1 l2 geneA l3 l4 geneB
      These are on a single line.  The first location is the start
      of the gene.  If l1>l2, the gene is on the complementary strand
   keninp: parameters to control the program.
      First line: FROM and TO of the output instructions.
      Second line: if the first character is 'b' then both open reading frames
 and identified genes are written to inst.  If it is 'n' then no open
 reading frames (orf) genes are not made into instructions.  If it is
 'o' then ONLY orfs are used.
   inst:  Delila instructions corresponding to allgen:
      piece piecename;
      get from l1 -FROM to l1 + TO;
   output: messages to the user

description
   This program converts Kenn Rudd's list of gene locations in his
   database into Delila instructions.

examples

documentation

see also

author
   Thomas Dana Schneider

bugs

technical notes

*)
(* end module describe.kenin *)
version = 1.24; (* of kenin.p 1991 Mar 23
```

```
(* begin module describe.keymat *)
(*
name
   keymat: keyed-matrices for helices between two books

synopsis
   keymat(xbook: in, ybook: in, hlist: in, kmlist: out,
          keymatp: in, output: out)

files
   xbook: a book from the delila system
   ybook: a book from the delila system.  If you want to look for
      structures in one sequence, then use the program copy to make
      a copy of xbook in ybook.
   hlist: the helix listing for xbook and ybook made by program helix
   kmlist: the matrices listed out.  Sequences from the x book are printed
      vertically, while those from the y book are horizontal.
      Depending on the parameters selected in keymatp, the helices are
      printed as either numbers representing the type of base pair,
      the actual base from the xbook, the actual base from the ybook,
      or a symbol representing the energy of the helix.
      If kmlist is wider than your printer, use the split program.
   keymatp: parameters to control the kmlist
      if keymatp is empty, default values are used.  Otherwise, keymatp
      must contain at least 3 lines.
         line 1: contains a positive integer - the minimum length helix to
            record from hlist into kmlist;
         or a negative real number - the maximum energy of a helix to
         record from hlist to kmlist.
         line 2: contains 2 positive integers greater than or equal to 1.
            These are the x and y scaling factors (respectively) which
            allow you to display large matrices in a small space by
            scaling them down.
         line 3: contains either 'n', 'x', 'y', or 'e',  which define what
             symbols will be printed for the helices in the kmlist.
            'n' - helices will be printed as a set of numbers:
                  1 = g-t bp    2 = a-t bp    3 = g-c bp
            'x' - helices will contain the base from the x-book sequence.
            'y' - helices will contain the base from the y-book sequence.
            'e' - a key symbol for the energy of each helix will be printed.
               The program will produce a table of energies and their
               corresponding key symbols.
               Note: the third parameter request is overridden when either
               scale factor is larger than 1.
         line 4: resolution of energy display.  This defines
                 the resolution of the matrix w/respect to
                 energies.  Used when line 3 is 'e'.
            'n' - numbers ('0' to '9') used for the keys
            'l' - numbers ('a' to 'z') used for the keys
            'a' - numbers ('a' to 'z' and '0' to '9') used (not available)
   output: messages to the user

description
   Keymat produces a keyed-dot matrix for the two books.  The display
```

```
   can use numbers and letters to indicate the energy of various helixes.
   One major feature is the ability to compress large regions onto a page
   using scaling factors.
   Only helices of some length (or longer) or of some maximum energy (or less)
   are printed.  The helices are made using program helix.
   This program was based on the matrix program.

documentation
   J. V. Maizel, Jr. and R. P. Lenk PNAS 78: 7665-7609 (1981)

see also
   helix, dotmat, matrix, split

author
   patrick r. roche
bugs
   If maximum energy is strongest helix, the program may object.
   The alphanumeric range does not work.  It bombs with a 'bus error'
   as it reads in the x piece.
*)
(* end module describe.keymat *)
version = 5.37; (* of keymat 1987 feb 16
```

```
(* begin module describe.lenin *)
(*
name
   lenin: convert a list of lengths into Delila instructions

synopsis
   lenin(lengths: in, leninp: in, finst: out, linst: out, output: out)

files
   lengths: The olength or slength file from dbinst.
      The file is expected to contain comment lines that start with '*'.
      These are followed by columns of LENGTH, FIRST-POSITION,
      LAST-POSITION and PIECE-NAME.
   leninp: parameters to control the program.
       First line: FROM and TO for the finst instructions
       Second line: FROM and TO for the linst instructions
   finst:  Delila instructions constructed from the lengths file
      according to the parameter file, and the FIRST-POSITION of the object
      or space.
   linst:  Delila instructions constructed from the lengths file
      according to the parameter file, and the LAST-POSITION of the object
      or space.
   output: messages to the user

description
   The program allows one to make a set of instructions that correspond
   to the ends of objects that exist in the GenBank entries.
   Dbinst does not do this; and it is easier to do it this way.

   For the finst file, the Delila instructions created are of the form:

   piece PIECE-NAME;
   get from FIRST-POSTION FROM.first to FIRST-POSITION TO.first;

   while for the linst file, the Delila instructions created are of the form:

   piece PIECE-NAME;
   get from LAST-POSTION FROM.last to LAST-POSITION TO.last;

examples

documentation

see also
   dbinst.p

author
   Thomas Dana Schneider

bugs
   Title of finst is not correct.  To correct, don't have dbinst put
the title, and have lenin construct it.

technical notes
```

```
*)
(* end module describe.lenin *)
version = 1.24; (* of lenin.p 1990 August 21
```

```
(* begin module describe.lig *)
(*
name
   lig: ligation theory

synopsis
   lig(input: in, list: out, output: out)

files
   input: user commands
   list: a 'hard copy' of the inputs and the outputs
   output: ligation predictions

description
   This program computes the results of a ligation reaction
   for insertion of a linker onto both ends of a linearized plasmid.
   The user gives
      the size of a plasmid in KB,
      the pico moles or micrograms of plasmid (you get to chose)
      the size of an insert in KB,
      the pico moles of or micrograms of insert
      the volume of the reaction in micro liters
   The program calculates whether circular or linear molecules are favored
   for the plasmid alone and with the insert.  The logic is:
   1.  The plasmid alone should circularize.
   2.  The plasmid with the insert should be linear.
   Thus, as the ligation proceeds, the first thing that happens is that
   the plasmid and insert ligate together (2 above).  Then the concentration of
   ends is lower, and circularization will be favored (1 above).
   Obviously this is all really rule of thumb, but it does seem to work
   in my experience.

documentation
   A. Dugaiczyk, H. W. Boyer and H. M. Goodman
   J. Mol. Biol. 96: 171-184 (1975)
   'Ligation of EcoRI Endonuclease-generated DNA fragements
    into Linear and Circular Structures'

author
   Thomas Dana Schneider

bugs
   none known

*)
(* end module describe.lig *)
version = 1.27; (* of lig 1988 Jan 5
```

```
(* begin module describe.matmod *)
(*
name
      linreg: linear regression

synopsis
      linreg(input: in, output: out)

files
      input:
          first line is which pair of columns to correlate as x then y
          remaining lines are the data in columns, ending with end of file.
      output: regression results

description
      linear regression is performed between the indicated data columns.

author
      thomas schneider

bugs
      none known

*)
(* end module describe.linreg *)
version = 2.00; (* of linreg 1985 dec 19
```

```
(*
name
      lister: list the sequences of pieces in a book with translation

synopsis
      lister(book: in, list: out, listerp: in, output: out)

files
      book: any book generated by the delila system

      list: a carefully numbered listing of the sequences in book,
         with an index to the pieces at the end

      listerp: lister parameters to control the listing.
         If listerp is empty, default values are used.
         Otherwise, the file must contain four integers, one per line:
         1. the number of bases per line in the listing.  Note that besides
            margin characters, there will be one blank with each base.
            This must be a multiple of 3 whenever one is printing amino acids.
         2. the mode for listing amino acids:
            0 = none
            1 = predict peptides starting at aug or gug. show nonsense codons.
            2 = translate all frames
         3. an integer in the range 0 to 7.  The binary representation of
            this number determines which amino acid frames are allowed to
            be printed.  The highest bit is the highest printed frame.
         4. Amino acid code: one character
               1 = 1 letter code
               3 = 3 letter code
         5. an integer that controls the listing of the sequence.
               0 = no sequence (but show amino acids and sequence numbering)
               1 = show sequence
               2 = show sequence and complement underneath
         6. Output format:  one character
               c = computer defined page character (often will be a control-L)
               l = LaTeX document page notation
               n = no page marks
         7. Page length (integer)

      output: messages to the user

description
      Lister is a general purpose program for the listing of nucleic-
      acid sequences.  Every fifth base is carefully marked with an
      asterisk directly above it.  Every tenth base is numbered with the
      number defined by the coordinate system.
         The listing can include translation to amino acids.  The amino acid
      is set directly below the codon.  Dashes mark the frame.

examples
      If listerp contains:

30    basesperline: number of bases per line in the listing
```

```
1     aastate: 0=no aa; 1=predict peptides; 2=translate all frames
7     frameallowed: binary; highest bit is highest frame on, etc.
1     codelength: 1 or 3 letters per amino acid
2     seqlines: 0=no sequence; 1=single strand; 2=double strand
c     pageaction: c=computer; l=LaTeX; n=none
55    pagelength:  page length
      listerp: parameters for the lister program

30    the listing will be 30 bases wide,
1     with predicted peptides for
7     the top frame.
1     The translated sequence is listed in single letter code.
2     Both DNA strands will be given.
c     The computer's default will be used to page the output.
55    Each page will break at 55 lines.

      More examples for frame control (parameter 3):
         7 (111 in binary) will translate all frames
         4 (100 in binary) will translate only the first frame
         3 (011 in binary) will translate the second and third frames

author
      Thomas D. Schneider

bugs
      none known

*)
(* end module describe.lister *)
      version = 5.50; (* of lister.p 1992 December 21 *)
```

```
(* begin module describe.ll *)                                                          224
(*
name
   ll: line lengths

synopsis
   ll(input: in, output: out)

files
   input: the source of the lines
   output: the length of each line

description
   The lengths of lines in the input file are given to the output file.
   A useful way to use the program is to find the longest length line in
   the file using the Unix sort routine:
       ll < myfile | sort | tail -1

see also
   Unix sort routine, flag.p.

author
   Thomas D. Schneider

bugs
   none known

*)
(* end module describe.ll *)
   version = 1.03; (* of ll 1991 Mar 23
```

```
(*
name
   lochas: look at characters in a file

synopsis
   lochas(input: in, output: out)

files
   input:  a file
   output: identification of ascii characters in the file
      each line contains:

         the first three characters: the ordinal of the character
         blank (ie " ")
         dash (ie "-")
         the character or a blank in special cases
         dash (ie "-")
         blank (ie " ")
         6 characters: the number of the character in the file
         blank (ie " ")
         the remainder contains one of:
            NULL - a null character found
            BLANK - a blank character was found
            HIGH ORDER BIT REMOVED - the character had its high order bit
                set.  To print it, this was removed.
      or
         END OF LINE
         which indicates that an end of line condition was found.
         This is counted as a single character.

description
   The program allows one to inspect the characters in a file.

examples

documentation

see also

author
   Thomas Dana Schneider

bugs

technical notes

*)
(* end module describe.lochas *)
version = 1.05; (* of lochas.p 1993 January 6
```

```
(* begin module describe.log *)
(*
name
   log: convert columns of data to log

synopsis
   logp: parameter file.  Base to take the log.
   input: lines starting with '*' are copied to output.
      The log is taken of the first two columns of input and this is
      written to output
   output: copied header lines and log of first two input columns

description
   The program takes the log of the first two input columns, and writes
   to the output their logs to the given base.  This lets one convert data
   for a log-log plot with the xyplo program.

see also
   xyplo

author
   Thomas D. Schneider

bugs
   To generalize this program, it would be nice to specify which columns
   are to be transformed, and other columns would be just copied to the
   output.  This requires that the program be able to take a list of
   columns, sort them, then skip and copy to the columns to be transformed.
   Other columns may contain non-numeric characters, so the copy must be
   of characters.  It would also be nice to have the program do other
   functions, like square root and sine.

*)
(* end module describe.log *)
version = 1.08; (* of log 1986 april 19
```

```
(* begin module describe.loocat *)
(*
name
      loocat: look at a catalogue

synopsis
      loocat(cat: in, list: out, output: out)

files
      cat: a catalogue generated by the catal program
      list: a listing of the contents of cat
      output: messages to the user

description
      loocat allows one to look at a catalogue that the librarian
      delila normally looks at.  these catalogues are files of a special
      type of record (called item) so that delila can read the information
      rapidly.  however this makes it difficult to see what the catalogue
      contains.  loocat is useful for understanding or debugging catalogues.

documentation
      libdef, delman.construction.catal

see also
      catal, delila

author
      gary stormo and thomas schneider

bugs
      none known

*)
(* end module describe.loocat *)
version = 1.10; (* of loocat 1985 apr 19
```

```
(* begin module describe.makebk *)
(*
name
      makebk: make a book from a file of sequences.

synopsis
      makebk(sequ: in, book: out, output: out, input: intty)

files
      sequ: file of raw sequences, each ending in a '.';  no characters are
         allowed in this file except the bases (a,c,g,t,u) and period and blank.
      book: the output file containing the sequences and the necessary
         information  for it to be a proper book.  the user types in the
         required information  after prompts from the program.
      output: for messages and queries to the user;
      input: interactive input.

description
      makebk takes a file of raw sequences (sequ) separated by periods (.)
      and converts that into a proper delila book format, getting the
      required information from the user.  the user may also have
      makebk fill in the piece information automatically, using
      default values.

see also
      rawbk

author
      gary stormo

bugs
      none known
*)
(* end module describe.makebk *)
version = 2.42; (* of makebk 1986 nov 14
```

```
(*
name
   makedate: make a date file

synopsis
   makedate(input: in, thedate: in, output: out)

files
   input: the date from the user
      The date may end with a single letter, to distinguish
      between several dates on one day.

      Acceptable formats of the date are
        1992 Jul  4
        1992 Jul 4
        1992 Jul 14
        1992 Jul 14a
        1992 Jul 1 a
        1992 Jul 1a

   thedate:  the date file created
   output: messages to the user

description
   Create a file containing a date

examples

documentation

see also
   tod.p

author
   Thomas Dana Schneider

bugs

technical notes

*)
(* end module describe.makedate *)
version = 1.27; (* of makedate.p 1992 October 30
```

```
(*
name
   makelogo: make a graphical ‘sequence logo’ for aligned sequences

synopsis
   makelogo(symvec: in, makelogop: in, colors: in, marks: in,
            [[ wave: in,]] logo: out, output: out)

files
   symvec: A "symbol vector" file from the alpro or dalvec program.
     If the file is empty, the alphabet is printed.  This allows one
     to determine the correction factors described below.
     If the error bars have a negative size, they are not displayed.
     This allows the sites program to control the display when it
     would not be appropriate.
     If the number of a symbol is negative in symvec, then the
     symbol will be rotated 180 degrees before being printed.
     The absolute value is used by makelogo to determine the height.
     This allows statistical tests which find rare symbols to be
     significant to show that the symbol is rare by having
     it up side down.  Notice that ACGT are all easy to distinguish
     from their upside down versions, but unfortunately this is not always
     true for protein sequences.
   makelogop: parameters to control the program.
     line 1: contains the lowest to highest range of the binding
             site to do the logo graph. (FROM to TO range)
     line 2: bar: sequence coordinate before which to print a vertical bar
             NOTE: the vertical bar takes up a small amount of horizontal space.
             This will offset the logo from that point on by a tiny amount.
     line 3: xcorner and ycorner.  This is the coordinate of the
             lower left hand corner of the logo (in cm).
             These should be real numbers.
     line 4: rotation: angle in degrees to rotate the logo.  Warning:
             rotations other than by factors of 90 degrees may produce
             incorrect logos because character scaling depends on the
             orientation of the characters.  (Essentially, it’s a design
             fault of PostScript.)
     line 5: charwidth: (real, > 0) the width of the logo characters, in cm
     line 6: barheight barwidth: (real, > 0) height of the vertical bar, in cm,
             and its width, in cm.
     line 7: barbits: (real) The height of the vertical bar, in bits, is given
     by the absolute value of barbits.  If barbits is positive, an
     "I-beam" will appear at the top of the symbol stack.  The I-beam
     indicates one standard deviation of the stack height, based
     entirely on how small the sample of sequences is.  If the value of
     barbits is negative, the I-beam is not displayed.  Not knowing how
     big the sampling effects are can fool one, so one should usually
     have the I-beam, even if it is ugly.
WARNING: it is not known how to calculate the error for data
     derived from a dirty DNA synthesis experiment (see Schneider1989,
     reference given below).  In that case the error could be
     calculated (in program sites) from the number of sequences, so
     that the error bar would be an underestimate of the variation.
```

```
     Unfortunately, when I tried this, people interpreted the error bar
     as the size they saw, so this does not work well visually.
     Therefore when data come from the sites program, the I-beam is
     suppressed.
The combination of barheight and barbits determines the size of
     the logo in bits per centimeter.  Both must be specified even if
     no vertical bar is desired.
     line 8: barends: if the first character on the line is a 'b', then bars
             are put before and after each line, in addition to the other bar.
             The first bar on each line is labeled with tic marks and
             the number of bits.  If you don't want this, you can remove
             the call to maketic in the logo.
     line 9: showingbox: if the first character on the line is an 's',
             then show a dashed box around each character.
    line 10: outline: If the first character is 'o' then the characters show
             up in outline form.  Otherwise, they are solid.
    line 11: caps: if the first letter is 'c' then alphabetic
             characters are converted to capital form.
    line 12: stacksperline: number of character stacks per line output
    line 13: linesperpage: number of lines per page output
    line 14: linemove: line separation relative to the barheight
    line 15: numbering: if the first letter is 'n' then each stack is
     numbered.  Otherwise, the number is suppressed as a PostScript
     comment.  This allows you to modify the logo file by hand to
     reinstate numbering for only the positions you want by removing
     the percent (%) symbol from in front of the calls to makenumber.
    line 16: shrinking: (real)  Factor by which to shrink the characters.  If
     shrinking <= 0 or shrinking >= 1 then the characters exactly fit
     into the dashed box.  If shrinking > 0 and shrinking < 1, the
     characters are shrunk inside the dashed box.  To use this feature,
     the parameter showningbox be on, so that the user does not create
     a logo whose height is misleading.
    line 17: strings: the number of user defined strings to follow.  Each
     string definition takes up two lines.  The first is the (x,y)
     coordinate of the string, the second is the string itself.  The
     coordinates are in centimeters relative to the coordinate
     transforms performed above.  (This way, the title position stays
     the same relative to the logo.)
    line 18: (x,y,s) coordinates of first user defined string (if strings >= 1)
             followed by the factor by which to scale the string.  A factor
             of 1 means no scaling.  In addition, if the x coordinate is
             negative, then the string is centered by using the string
             width, the stacksperline and charwidth.
    line 19: the first user defined string (if strings >= 1)
    line 20: (x,y,s) coordinates of second user defined string (if strings >= 2)
    line 21: the second user defined string (if strings >= 2)
             (etc. for the remaining strings.)
    The remainder of the file is ignored and may contain comments.
   colors: Defines the color of each character printed.
     Any number of lines that begin with an asterisk [*] can be used
     as comments to identify the file or portions of the file.
     Put into the file one line for each character that is to
     have a color other than black.  The line must contain:
                    character red green blue
```

```
    The last three parameters are real values between 0 and 1 (inclusive).
    The values depend on the PostScript interpreter, but 0 means black
    and a value of 1 means the most bright.
    To assign the asterisk a color, proceed it with a backslash [as \*].
    To assign the backslash a color, proceed it with a backslash [as \\].
    If the file is empty, the logo is made in black and white and the lower
    half of the I-beam error bar is made white so that when it is inside the
    letters it is visible.
  marks: an empty file means no marks are made.  Otherwise, a series
    of lines containing four pieces of data that define marks to be placed
    over the output:
       mark: o means open circle, b means filled circle.
       base coordinate: a real number that determines the center of the mark
       bits coordinate: a real number that determines the position of the
          mark in bits.
       scale: a positive real number by which to scale the mark.
    The symbols must be in increasing order of position in the site.
[[
  wave: Define a cosine wave over the graph.  Empty file means no cosine
    wave, otherwise the parameters of the wave are given one per line:
      extreme: char;  h or l, the extreme high or low point on the curve
         defined by the wavelocation and wavebit
      wavelocation: real; the location in bases of the extreme
      wavebit: real; the location in bits of the extreme
      waveamplitude: real; the amplitude of the wave in bits
      wavelength: real; the wave length of the wave in bases
      dash: real; the size of dashes in cm.  Zero or negative means no dashes
      thickness: real; thickness of the wave in cm.  Zero or negative
         means the value defaults to PostScript line thickness.
]]
  logo: the output file, a PostScript program to display the logo.
  output: messages to the user

description
  The makelogo program generates a ‘sequence logo’ for a set of aligned
  sequences.  A full description is in the documentation paper.  The input is
  an ‘symvec’, or symbol-vector that contains the information at each
  position and the numbers of each symbol.  The output is in the graphics
  language PostScript.

  The program now indicates the small sample error in the logo by a small
  'I-beam' overlayed on the top of the logo.  Although the user may
  turn this off to make pretty logos, I strongly recommend use of it
  to avoid being fooled by small amounts of data.

  Making A Logo As Part of Another Figure
  ---------------------------------------

  The normal logo file is designed to stand by itself.  However, it is often
  desirable to incorporate the logo as part of another figure.  The difficulty
  is that the stand-alone logo PostScript program will erase the page (which
  wipes out any previous figure drawing) and show the page (which prints the
  page right after the logo).  To prevent these actions, the lines of
  PostScript code which do this have comments that contain the word REMOVE.
```

All you have to do is remove these lines and your logo will be able to fit
into your figure. In Unix this can be easily done by:

```
grep -v REMOVE logo > logo.ps
```

If you do this, then it is advisable to do the erasepage and the
showpage yourself. A convenient way to do this is to have several
files that contain postscript commands, and to use a shell script
to concatenate them together:

```
cat start.ps logo.ps end.ps > myfigure.ps
```

If you have a large number of logos together in one figure, you can reduce
the size of the final figure by another trick. Logo files begin with a
header which is the same from one figure to the next assuming you don't
change colors/letter combinations. So the first logo in the figure must
contain this header, but later ones don't really need it. You can remove
the header material by using the censor program:

```
censor < logo.ps > logo.no.header.ps
```

author

Thomas D. Schneider
National Cancer Institute
Laboratory of Mathematical Biology
NCI/FCRDC Bldg 469. Room 144
P.O. Box B
Frederick, MD 21702-1201
(301) 846-5581 (-5532 for messages)
network address: toms@alum.mit.edu

examples

makelogop parameters:

```
-15 2      FROM to TO range to make the logo over
1          sequence coordinate before which to put a bar on the logo
15 2       (xcorner, ycorner) lower left hand corner of the logo (in cm)
90         rotation: angle to rotate the graph
1.0        charwidth: (real, > 0) the width of the logo characters, in cm
10 0.1     barheight, barwidth: (real, > 0) height of vertical bar, in cm
2          barbits: (real) height of the vertical bar, in bits; < 0: no I-beam
no bars    barends: if 'b' put bars before and after each line
show       showingbox: if 's' show a dashed box around each character
no outline outline: if 'o' make each character as an outline
100        stacksperline: number of character stacks per line output
1          linesperpage: number of lines per page output
1.1        linemove: line separation relative to the barheight
numbers    numbering: if the first letter is 'n' then each stack is numbered
1          shrinking: factor by which to shrink characters inside dashed box
2          strings: the number of user defined strings to follow
2 14 1     coordinates of the first string (in cm)
First TITLE
3 13 1     coordinates of the second string (in cm)
SECOND TITLE
```

```
   colors:
* Color scheme for logos of DNA (for the makelogo program).
* color order is red-green-blue
*
* green:
A 0 1 0
a 0 1 0
*
* blue:
C 0 0 1
c 0 0 1
*
* red:
T 1 0 0
t 1 0 0
*
* orange:
G 1 0.7 0
g 1 0.7 0

[[
   wave:
l        extreme: char; h or l, the high or low extreme to be defined
2        wavelocation: real; the location in bases of the extreme
1.0      wavebit: real; the location in bits of the extreme
0.5      waveamplitude: real; the amplitude of the wave in bits
10.4     wavelength: real; the wave length of the wave in bases
0        dash: real; the size of dashes in cm.  dash <= 0 means no dashes
0.1      thickness: real; thickness of the wave in cm. <=0: default.

]]
```

A test symvec is provided with the program, file 'symvec.demo', to be run with 'colors.demo' and 'makelogop.demo'.

```
documentation
   Description of Logos:
@article{Schneider.Stephens.Logo,
author = "T. D. Schneider
 and R. M. Stephens",
title = "Sequence Logos: A New Way to Display Consensus Sequences",
journal = "Nucl. Acids Res.",
volume = "18",
pages = "6097-6100",
year = "1990"}

   The Blue Book:
@book{PostScriptTutorial1985,
author = "{Adobe Systems Incorporated}",
title = "PostScript Language Tutorial and Cookbook",
publisher = "Addison-Wesley Publishing Company",
address = "Reading, Massachusetts",
callnumber = "QA76.73.P67P68",
isbn = "0-201-10179-3",
```

```
   The Red Book:
@book{PostScriptManual1985,
author = "{Adobe Systems Incorporated}",
title = "PostScript Language Reference Manual",
publisher = "Addison-Wesley Publishing Company",
address = "Reading, Massachusetts",
callnumber = "QA76.73.P67P67",
isbn = "0-201-10174-2",
year = "1985"}

   Dirty DNA synthesis experiments:
@article{Schneider1989,
author = "T. D. Schneider
 and G. D. Stormo",
title = "Excess Information at Bacteriophage {T7} Genomic Promoters
Detected by a Random Cloning Technique",
year = "1989",
journal = "Nucl. Acids Res.",
volume = "17",
pages = "659-674"}

see also
   rsgra.p, rseq.p, dalvec.p, alpro.p, sites.p, censor.p

bugs
   Some chi-logo (upside down characters) do not display on OpenWindows,
   but do print ok on the Apple LaserWriter IIntx.  The reason is completely
   obscure.

   A bug in NeWS 1.1 is that characters that are scaled too small are
   forced to be big.  This messes up the logo and can be confusing.
   Another bug in NeWS 1.1 prevents one from using the outline, but
   the dashed boxes will show up.
   Sometimes displaying a logo in NeWS 1.1 on a Sun 4 will cause an 'illegal
   instruction', after which one is thrown completely off the computer.  The
   source of this is not known, since it is not repeatable.
   The first two bugs are resolved under OpenWindows 2; the third has not
   been observed.
   These NeWS bugs do not apply to the Apple LaserWriter IIntx,
   which prints everything correctly.

technical notes
   Unfortunately PostScript fonts are not exactly the same height.  Thus if A
   and T are the standard, then C and G hang above and below the line.
   This has been solved in this version of makelogo.  As a consequence,
   the user never need to determine any character sizes empirically, and
   the logos should work on any PostScript printer.

   Special thanks go to the following people for their help in solving this
   problem:

   Kevin Andresen [kevina@apple.com]
```

```
    "The problem facing you is that, while the PostScript language is more or
    less standard, the font shapes depend on the designer, type vendor, or
    language implementation.  The fonts used in NeWS are not exactly the same
    as those from Adobe, which are not the same as those from Bitstream, which
    are not the same as the original lead type, etc.  (This is an
    industry-wide issue.)  One way to compensate for this in PostScript is to
    use the charpath and pathbbox operators and scale appropriately."

    He provided a program, which I then rewrote and generalized.  That version
    almost worked, but not quite.  This was solved by:

    finlay@Eng.Sun.COM (John Finlay) who said:
    "It would appear that the calculation of the pathbbox for characters varies
    with the scale of the characters (I don't know why exactly but would
    speculate that there's probably some weirdness with the font hints and
    scaling).  I modified your postscript to iterate once on the size and
    recalculate the pathbbox at the scaled size.  Seems to printout OK (inside
    the boxes) on a LWI, LWII and in NeWS2.0 (though NeWS still seems to get the
    wide slightly wrong)."

    shiva@well.sf.ca.us (Kenneth Porter) was also involved and actively
    interested.  My apologies if I have forgotten someone else who contributed.

    The letter I and the vertical bar (|) are treated specially since in the
    Helvetica-Bold font they are rectangles and would completely fill the
    character space.  In addition, the letter I is centered by makelogo.

    Thanks go to Joe Mack for suggesting numbering and titles (strings) and to
    Pete Lemkin and Wojciech Kasprzak for pointing out that the shrink option
    would be helpful.  Thanks to Jeff Haemer for pointing out that the
    PostScript program should begin with '%!', and for suggesting that
    the string fonts should be different from the logos themselves.

    MISSING LOGO LETTER PROBLEM

    The OpenWindows PostScript on a Sun workstation will mess up displaying a
    stack of letters if the vertical movement is too small.  The result is that
    the letters above that point are missing.  This occurs if there is a highly
    conserved base and very few other bases.  The result is a huge gap where the
    highly conserved base should be.  Other printers do fine, so this is a
    problem with the Sun implementation of PostScript (will they ever get it
    right???).  If you don't have this window system, set the constant
    gooddisplay to true.  If you do want the logos to show up properly on the
    screen, use false.  Unfortunately, this will mean that the vertical
    translation for the small letters won't be done, so the display will be very
    slightly wrong.

*)
(* end module describe.makelogo *)
version = 7.53; (* of makelogo.p 1992 June 8
```

```
(* begin module describe.makessbdate *)
(*
name
   makessbdate: make a date file from a Sample_Sheet.bin file

synopsis
   makessbdate(input: in, thedate: in, output: out)

files
   input: a Sample_Sheet.bin file
      The date may end with a single letter, to distinguish
      between several dates on one day.

      Acceptable formats of the date are
        1992 Jul  4
        1992 Jul 4
        1992 Jul 14
        1992 Jul 14a
        1992 Jul 1 a
        1992 Jul 1a

   thedate:  the date file created
   output: messages to the user

description
   Create a file containing a date

examples

documentation

see also
   tod.p

author
   Thomas Dana Schneider

bugs

technical notes

*)
(* end module describe.makessbdate *)
version = 1.28; (* of makessbdate.p 1993 January 25
```

```
(* begin module describe.makman *)
(*
name
   makman: make manual entries from a source code

synopsis
   makman(input: in, output: out)

files
   input: a source code containing one or more modules with names
      of the form 'describe.name'.  The module must be proceeded
      by a "version = " identification line.
   output: the modules with names of the form 'describe.name'.
      This is followed by the "version = " line.

description
   Modules with names of the form "describe.name" are copied from the
   input to the output.  By appending a set of such modules together
   from several programs, one can create a manual.  The pages may
   then be broken apart with the break program.

see also
   module.p, break.p, shell.p

author
   Thomas D. Schneider

bugs
   none known
*)
(* end module describe.makman *)
version = 1.32; (* of makman.p 1993 January 27
```

```
(* begin module describe.makmod *)
(*
name
      makemod: create a set of empty modules from a list of names

synopsis
      makmod(fin: in, fout: out, output: out)

files
      fin: a set of names separated by blanks or end-of-lines.
      fout: a file of modules with the names listed in fin.
      output: messages to the user.

description
      makmod creates a set of empty modules that have
      the names given in the fin file.  one may then use
      the module program to extract modules by the same names
      from a module library (for example).

examples
      if the fin file contains:
         first second
         3.rd

      the fout file will contain:                                 *)
         (* begin module first *)
         (* end module first *)

         (* begin module second *)
         (* end module second *)

         (* begin module 3.rd *)
         (* end module 3.rd *)                                    (*

see also
      moddef, module, show

author
      john hoffhines

bugs
      none known

*)
(* end module describe.makmod *)
version = 1.11; (* of makmod 1986 dec 12
```

```
(* begin module describe.maknam *)
(*
name
   maknam: make manual entry names

synopsis
   maknam(input: in, output: out)

files
   input: a source code containing one or more modules with names
      of the form 'describe.name'.  Generally though, the output
      of the makMAN program is used.
   output: The name line of each describe module, which is always assumed to be
      the third line below the '(@ begin module descibe.x' line.  (@ is used
      here instead of * to prevent compilers from complaining.)

description
   For each module with a name of the form "describe.name",
   the third line of that module is copied to the output.
   This generates a file containing the name description of the program.
   (See example line above.)  This program is intended to be used on the
   concatinated output of the makman program, so that
   one can create a manual page that describes each program.

see also
   makman

author
   Thomas D. Schneider

bugs
   none known
*)
(* end module describe.maknam *)
version = 1.06; (* of maknam.p 1993 Jan 27
```

```
(*
name
      malign: optimal alignment of a book, based on minimum uncertainty
synopsis
      malign(inst: in, book: in, malignp: in, uncert: out, newalign: out,
         optalign: out, optinst: out, bestinst: out, output: out)
files
      inst: delila instructions of the form 'get from 56 -5 to 56 +10;'
      book: the book generated by delila using inst
      malignp: parameter file with the following parameters:
      winleft, winright: left and right ends of window for calculating
          uncertainty, relative to aligned base
      shiftmin, shiftmax: minimum and maximum shift of aligned base
      iseed: integer random seed
      nranseq: number of random sequences, or 0 to use sequences in book
      nshuffle: number of times to redo alignment after random shuffle
      ifpaired: 1 to treat each pair of sequences as complementary strands,
         0 not to
      standout: output run #, pass # and H to standard output every pass
         if 1, every run if 0, or not at all if -1
      npassout: output H and alignment every npassout passes, or only at
         end of runs if zero, or not at all if -1
      nshiftout: output L and H(L) every nshiftout sequence shifts, or only
         at end of passes if zero, or not at all if -1
      tolerance: tolerance in change of H
      ntolpass: maximum number of passes with change below tolerance
      uncert: uncertainty as function of position, for the last run, at the
      end of each pass or after selected number of sequence shifts
      newalign: values of H and the relative alignments; starting, final, and
      intermediate if selected
      optalign: user-readable listing of unique optimal relative alignments
      and number of times each was achieved
      optinst: list of unique optimal alignments in absolute coordinates,
      to be used to make inst file for selected alignment
         This file is like optalign, but the coordinates are for the
         original sequence.
      bestinst: a new inst file at the very best alignment
description
      Given a book of aligned sequences, this program searches for the alignment
      of the sequences that has the lowest uncertainty, i.e. the highest value
      of Rsequence.  The user specifies the "window" of bases within which
      uncertainty is calculated, and the maximum number of bases that each
      sequence is allowed to shift from the original alignment.  The program
      considers each sequence in turn, shifting it to an alignment with minimum
      uncertainty while holding the other sequences fixed.  A "pass" is complete
      when all sequences have been considered.  A "run" is complete when no
      alignments have changed in the preceding pass, and the alignment is then
      considered "optimal".  The first run starts with the original alignment;
      every run after that starts with a "shuffled" alignment obtained by
      shifting each sequence independently by a random amount between the
      allowed limits.  The program maintains a list of all of the unique optimal
      alignments achieved from these starting alignments, and it outputs them in
      order of increasing uncertainty.
```

```
author: David Mastronarde
bugs: The realignment algorithm, which shifts all sequences by the same amount
      to attempt to keep the window near its original position, is somewhat
      ad hoc in nature and the effects of different settings for it parameters
      have not been explored.  If the window spans two real sites with competing
      alignments, many optimal but meaningless alignments with similar
      uncertainties may be obtained.  The random sequences can't be examined. *)
(* end module describe.malign *)
version = 2.22; (* of malign.p 1991 February 8
```

```
(*
name
      markov: markov chain generation of a dna sequence from composition.

synopsis
      markov(cmp: in, mkvseqs: out, listing: out, markovp: in, output: out)

files
      cmp: the input composition, which is the output of program comp.
      mkvseqs: the output dna sequences of this program.
      listing: contains the following information about program execution:
         program and version number.
         first three lines of the input composition file used to
            generate the sequences with.
         the four input parameters - number of sequences, length of sequences,
            the seed number, and the depth of sequence generation.
         a listing of any sequence that could not be generated from the
            prior length specified in the markovp file.
            the listing contains the sequence number, the sequence position
            and the depth of restart of sequence generation.
      markovp: for parameters; markovp must contain four numbers
         each on separate lines:
         1. number of sequences desired - an integer
         2. the length of each sequence - an integer
         3. a seed number between zero and one or outside this range
         if a computer date and time seed is desired.  the seed
         is used to start the random number generator. - a real
         4. the number of bases prior to the one about to be inserted
         which are to influence the choice of the base to be inserted.
         zero means equiprobable random sequences are desired.
         example:   20 number of sequences desired
                    100 length of sequences desired
                    2.0 for a computer date and time seed
                    3 composition depth used to generate next base
      output: for user messages.

description
      markov generates a set of random dna sequences which have
      approximately the same composition as the one in the
      composition file supplied to the program.  the user chooses
      the depth of the composition to be used.  for example, if
      trinucleotides (composition depth = 3) are used, the
      previous two bases determine the probability of the next base in
      the sequence.  this is called a markov chain.
         sometimes the program will work itself
      into a corner, when no composition exists for the
      previous few bases.  in these cases, the program restarts
      with the longest possible oligonucleotide that does exist in the
      composition.  these cases are recorded in the listing.

see also
      comp, compan, rndseq
```

```
author
      john eberwein, gary stormo, tom schneider

bugs
      none known
*)
(* end module describe.markov *)
version = 3.73; (* of markov, 1989 march 5
```

```
(* begin module describe.matmod *)
(*
name
      matmod: mathematics modules

synopsis
      matmod(encseq: in, output: out)

files
      encseq: empty or the output of the encode program for testing parameter
         reading routines.
      output: the version of matmod is printed.  successful compilation
         and running of the program indicates that the modules are correct.

description
      self contained modules for mathematical manipulation.
      included is a procedure for linear regression analysis of data pairs,
      a random number generator, newton's method to find roots of functions,
      and routines for reading the parameters from the encode program.

see also
      delmod, module, encode

author
      thomas d. schneider and gary d. stormo

bugs
      none known

technical notes
      the constant n in procedure randomtest determines how many times
      the random number generator will be in a series of tests.  if n
      is small, the the test will be poor, if it is large then the test may
      take a long time.

*)
(* end module describe.matmod *)
version = 'matmod 2.05 88 dec 15 tds/gds';
```

```
(*
name
      matrix: dot matrices for helices between two books

synopsis
      matrix(xbook: in, ybook: in, hlist: in, mlist: out,
             matrixp: in, output: out)

files
      xbook: a book from the delila system

      ybook: a book from the delila system.  If you want to look for
         structures in one sequence, then use the program copy to make
         a copy of xbook in ybook.

      hlist: the helix listing for xbook and ybook made by program helix

      mlist: the matrices listed out.  Sequences from the x book are printed
         vertically, while those from the y book are horizontal.
         helices are printed as a set of numbers:
            1 means gt base pair
            2 means at base pair
            3 means gc base pair
         if mlist is wider than your printer, use the split program.

      matrixp: parameters to control the mlist
         If matrixp is empty, default values are used. otherwise, the first
         line contains one number.  If this number is a positive integer,
         it specifies the minimum length helix in base pairs from hlist to
         record in mlist;
         if this number is a negative real number, it specifies the maximum
         energy in kcal of the helixes written in mlist.

      output: messages to the user

description
      Matrix produces a dot matrix for the two books.  Only helices
      of some length (or longer) or of some maximum energy (or less) are
      printed.  The helices are made using program helix.

documentation
      delman.use.comparison
      J. V. Maizel, Jr. and R. P. Lenk PNAS 78: 7665-7609 (1981)

see also
      helix, dotmat, split, keymat

author
      Thomas D. Schneider

bugs
      none known
```

```
      The constant maxarray defines the maximum area that the program
      can handle.

*)
(* end module describe.matrix *)
      version = 3.28; (* of matrix 1987 feb 13 *)
```

```
(*
name
      merge: compare two files and merge them

synopsis
      merge(afile: in, bfile: in, apfile: out, bpfile: out,
            output: out, input: intty)

files
      afile: the first input file
      bfile: the second input file
      apfile: the afile with corrections from bfile or the user
      bpfile: the bfile with corrections from afile or the user
      output: messages to the user
      input: interactive input from the user.

description
      the merge program was designed to aid in the entry of sequences.
      merge will also compare any two files for differences.
      two typed copies of the data are made (afile and bfile).  merge will
      compare the files, ignoring spaces and end-of-lines.  this allows
      the two data files to be typed independently by two people in two
      formats.  differences between the files are flagged, and the user
      may then indicate which file is correct and merge will fix the other
      file.  the user may also modify the files using a small
      editing facility.  the changes go to the prime files (apfile, bpfile).
      to be sure that apfile and bpfile are identical after the merge,
      you can merge them again.  several commands can be put on one line
      separated by blanks.  if you type an unrecognizable command, or ask for
      help at any time then merge will list the commands available.

examples
      when two sequences were compared, the program gave this output:
i am 91% sure that this has a deletion in b (insertion in a) of 5 characters:
file a: line 1
aatccttatccctcctaatttcgtttttgct
       >iiiii<       x         at 9 insertion, 1 mismatch downstream
file b: line 1
aatccttacctaatttccttttttgct
       ><        x         at 9 deletion, 1 mismatch downstream
      the sequences matched before the points indicated.  file a had
      an insert of "tccct" followed several bases later by a c to g change.
      the mismatch made merge less sure of the deletion.
      one must look at the original sequence to make the correction.

author
      thomas d. schneider

bugs
      1. lines without any characters are not copied from the file to its
      pfile.  see procedure readline.
      2. excessive blank characters may fool the guess procedure, since it
      does not remove blanks before guessing.  removing blanks would make
```

```
      it difficult to autofix, and the spacings would be lost in the pfile.
      3. the program can not compare more than one line from each file at a
      time,  so the guess is limited to what is visible on a line.  the entire
      program must be rewritten to allow multiple line guessing.
*)
(* end module describe.merge *)
version = 9.53; (* of merge 1989 May 1 *)
```

```
(*
name
   mnomial: produce the multinomial distribution for base probabilities

synopsis
   mnomial(mnomialp: in, list: out, output: out)

files
   mnomialp: parameters to control the program, as pairs of lines
      first line: na,nc,ng,nt
      second line: pa,pc,pg,pt
        this may repeat.
      When ng=nt=0 and pg=pt=0, the binomial distribution is calculated.
      The mean (n*pa) and standard deviation (sqrt(n*pa*pc)) are given.
   list: results
   output: messages to the user

description
   This program calculates the multinomial distribution:
                                 (na+nc+ng+nt)!   na   nc   ng   nt
    p(na,nc,ng,nt|pa,pc,pg,pt) = -------------- pa   pc   pg   pt
                                 na!nc!ng!nt!

see also
   binplo

author
   Thomas Dana Schneider

bugs
   none known

*)
(* end module describe.mnomial *)
version = 1.18; (* of mnomial, 1988 Dec 14
```

```
(* begin module describe.modin *)
(*
name
      modin: generate modularized delila instructions for absolute sites

synopsis
      modin(fin: in, inst: out, output: out)

files
      fin: sequence site positions in a special format.
      inst:  modularized delila instructions
      output: messages to the user

description
      The existence of a file containing modularized delila instructions
      allows one to pull, from the file, instructions for generating
      specific sequence sites using the module program.  For instance,
      using modin, one may make a file containing delila instructions for
      all the laci amber mutation sites, one site per module.  Then, using
      the module program, one could pull from the file instructions for
      sites a9, a16, a19, and a21 by using the module program.  This
      would be useful if one had several different sets of amber mutations
      to analyse separately.

see also
      module, delila, describe.modin.use

author
      John Hoffhines

bugs
      The program has not been used much, so its usefulness is not known.

*)
(* end module describe.modin *)
version = 1.43; (* of modin.p 1993 Jan 27
```

```
(*
name
      modin.use: more information on using the modin program

     MODIN FIN FILE FORMAT (BNF)

<BOOK TITLE><SETS>
<SETS>::=<SET>|<SET><SETS>
<SET>::=SET <KEY> <KEY> <KEY> <KEY> <SIGNED NUMBER> <SIGNED NUMBER>
            <MODULE GROUP>
<MODULE GROUP>::=<MODULE PARAMETERS>|<MODULE GROUP> <MODULE PARAMETERS>
<MODULE PARAMETERS>::=M <NUMBER> <IDENTIFIER>|<NUMBER> <IDENTIFIER>

     NOTES ON MODIN FIN FILE FORMAT

1) Any terms undefined here are defined in LIBDEF.

2) Keys designate, respectively, the words "organism", "chromosome",
    and "p" (piece), "g" (gene), or "t" (transcript).

3) The module parameter "m" sets delila instructions direction
    to - for one module only.  Default is +.

4) Within the module parameters section, "number" is DNA base position,
   and "identifier" immediately following it is that which will be
   used in that position's module name.

5) Only one set per line is allowed, without its module group.  Module
   groups follow on subsequent lines; more than one per line is allowed,
   but they may not be truncated by the end of a line.

*)
(* end module describe.modin.use *)
version = 1.43; of describe.modin.use 1993 Jan 27 *)
```

```
(*
name
      modlen: determine module lengths

synopsis
      modlen(fin: in, fout: out, modlenp: in, output: out)

files
      fin: a text file containing modules
      fout: a list of the module names and their lengths
      modlenp: parameters to control modlen.
         if the file is empty, modlen gives a complete list of
         modules and their lengths to fout, and notes those
         longer than 'a certain number of lines' lines to output.
         otherwise the file is expected to contain two integers,
         each at the start of a line.  these are the shortest and
         longest lengths to print to output.
      output: messages to the user and modules with lengths
         determined by modlenp.

description
      the delila manual consists of module pages.  this tool allows
      you to find out if the pages will fit onto your printer.

see also
      module, show, break

author
      thomas d. schneider

bugs
      none known

technical notes
      'a certain number of lines' is set by the constant defshort.
*)
(* end module describe.modlen *)
version = 1.36; (* of modlen 89 July 14 *)
```

```
(*
name
      module: module replacement program

synopsis
      module(sin: in, modlib: in, sout: out,
             modcat: inout, list: out, output: out)

files
      sin: the source program or file
      modlib: a library of modules (if empty, modules of sin are stripped)
      sout: the source program with modules replaced from modlib
      modcat: an alphabetic index to modlib that is recreated
         if it does not match modlib
      list: progress of the transfer.  meaning of the list columns:
         nesting depth:  how deeply the module was nested inside other modules
         action:  what was done with the module.  if a module was not
            transferred, a symbol on the left flags the situation:
              (blank) successful transfer
            * module not found in the source
            v no transfer because version modules can not be transferred
            ? recursive transfers were aborted because the modules may be
              infinitely nested (the depth at which this happens can be
              increased by changing the program - ask your programmer).
              (problem: can you construct this bizarre infinite situation?)
         module name: the name of the module in the source.  in recursive
            cases, these are from the modlib.
      output: messages to the user

description
      the module program allows one to construct libraries of special
      purpose program modules, which one simply 'plugs' into the
      appropriate place in a program.  this speeds up both program design
      and error correction.  module is more general-purpose than the standard
      'include' type processes because it performs a replacement rather than
      a simple insertion.  the operation is recursive, so a module may be
      composed of other modules.  the replacement mechanism also allows one to
      run the program in 'reverse' so that module-libraries are created by
      extracting modules from existing programs.  this makes the building of
      module libraries easy, and helps keep them updated with new modules and
      improvements to old ones.
         for a full description, see the documentation.

documentation
      moddef, delman.assembly.modules,
      delman.intro.organization 'technical notes'

see also
      delmod, prgmod, matmod, break, show (especially...)

author
      thomas d. schneider
```

```
      none known

*)
(* end module describe.module *)
version = 'module 6.07  88 jan 6 tds';
```

```
(* begin module describe.mstrip *)
(*
name
   mstrip: remove control m's from a file

synopsis
   mstrip(input: in, output: out)

files
   input: a file which contains control m's (^M as seen in vi)
      which one desires not to have control m's
   output: the input copied to the output without the ^M's.

description
   the tip program in conjunction with the cyber produces extra
   control m's at the ends of lines in the scripts.  this program
   removes them.

author
   tom schneider

bugs
   none known
*)
(* end module describe.mstrip *)
version = 1.01; (* of mstrip.p 1993 Jan 27
```

```
(* begin module describe.nocom *)
(*
name
   nocom: remove comments

synopsis
   nocom(input: in; output: out)

files
   input:  a program with comments.
   output: the same program with the contents of the comments removed

description
   This program removes comments from a Delila or Pascal source code
   so that one can compare two outputs of dbinst.

see also
   dbinst

author
   Thomas Dana Schneider

bugs
   may not apply to nocom:

   WARNING:  Some programs have comment starts inside quotes.  DECOM
   IS NOT SMART ENOUGH TO AVOID CHANGING THESE.  If they exist, nocom
   will mess up your program.  Compare the output of nocom with the
   input before you accept the results.

*)
(* end module describe.nocom *)
version = 1.03; (* of nocom.p 1990 May 14
```

```
(*
name
   normal: generate normally distributed random numbers

synopsis
   normal(normalp:in, data: out, output: out)

files
   normalp: parameter file controlling the program.
      Two numbers, one per line:
         seed: random seed to start the process
         total: the number of numbers to generate
   data: This is a set of numbers which should have Gaussian distribution
      if the random number generator is a reasonable one.
      It will be N(0,1), a normal distribution
      with mean 0 and standard deviation 1.
   genhisp: control file for the genhis histogram plotting program.
   output: messages to the user

description
   Test of a random number generator by creating a gaussian
   distribution of numbers for plotting by genhis.
   Method: if U is a member of the set [0..1] and Un and Un+1
   are two members, then define
      theta = Un 2 pi
      r     = sqrt(-2 ln(Un+1))
   then when these polar coordinates are converted to Cartesian
   coordinates, one gets two independent Normally distributed numbers,
   with mean 0 and standard deviation 1.  To get other standard deviations
   multiply by a constant, and to get other means, add a constant.

   The proof was from a friend; I only have sketch notes at the moment.
   I'm sure it is available in standard texts.  However, it works,
   as shown by the example.

example
  seed := 0.5;
  total := 10000;
     The mean was 0.00 (to two places) and the standard deviation was 1.01.

see also
   gentst, tstrnd, genhis

author
  Tom Schneider
  National Cancer Institute
  Laboratory of Mathematical Biology
  Frederick, Maryland
  toms@ncifcrf.gov

bugs
   none known
```

```
*)
(* end module describe.normal *)
version = 3.16; (* of normal.p 1993 Jan 27
```

```
(* begin module describe.notex *)
(*
name
   notex: remove tex and latex constructs

synopsis
   notex(input: in, output: out)

files
   input: a tex or latex file
   output: the file with:
      '\xxx' command words converted to spaces,
      '{$}' converted to spaces
      free floating '.' ',' '(' ')' removed
      comments (%) removed

      multiple spaces are comressed to single spaces.
      multiple lines are compressed to 2 lines (to preserve the
      paragraph structure).

      Only characters numbers and blanks are left behind

description
   This reduces the number of words counted by wc to something close to correct.
   It is harsher than untex in that it specifically filters out
   everything except numbers, alphabetic chracters and the blank.

author
   Thomas D. Schneider

bugs
   citations and comments on lines by themselves leave a blank line.

*)
(* end module describe.notex *)
version = 1.32; (* of notex.p 1991 February 1
```

```
(* begin module describe.nulldate *)
(*
name
   nulldate:  modules to neutralize the date-time functions

synopsis
   nulldate(output: out)

files
   output: where the (neutralized) date and time will appear.

description
   If transportation of a program or translation to C is hindered by the
   presence of the date-time modules, then one may want to blank out the
   function of those modules for the time being.  Thus all the dates produced
   will be zero, but one will be able to run the programs Nulldate contains
   modules that will replace corresponding modules in the other module
   libraries which are system dependent.  This will allow easy transportation
   of the Delila system to other computers.

documentation
   moddef, delman.describe.module

see also
   delman.describe.delmod, moddef, delman.describe.module
   delmods, prgmods, matmods, vaxmod

author
   Thomas D. Schneider

bugs
   none known

technical notes
   The datetime package required a const 'namelength' and a type 'alpha'.
   These are part of the book.const and book.type modules of delmod, and
   are identical to those types and consts.  Note:  programs which use
   the datetime package must have these types and consts either from
   delmod or manually declared.
*)
(* end module describe.nulldate *)
version = 1.03; (* of nulldate 1991 Nov 5 *)
```

```
(* begin module describe.number *)
(*
name
   number: add line numbers to a file

synopsis
   number(input: in, output: out)

files
   input:  input file
   output: input file with line numbers at the start

description
   Add line numbers to the input file.

examples

documentation

see also

author
   Thomas Dana Schneider

bugs
   Perhaps should have option to define number of columns for
   the line numbers.

technical notes

*)
(* end module describe.number *)
version = 1.08; (* of number.p 1991 September 16
```

```
(* begin module describe.odti *)
(*
name
   odti: munch od and time plates together for xyplo

synopsis
   odti(od: in, time: in, odtime: out, output: out)

files
   od: a file containing just an od plate from the tk program.
      blank wells are indicated by an '*'.
   time: a file containing times.
      blank wells are indicated by an '*'.
   odtime: the od and time values are spliced together,
      lines beginning with * are copied to output, then
      the time followed by each od are put on lines by themselves.
      this is the form that that the xyplo program can use for
      plotting.

description
   the od and time plates are fused together for plotting with xyplo.

auther
   tom schneider

see also
   xyplo and tk (written in basic)

bugs
   does not take full tk output.

*)
(* end module describe.odti *)
version = 1.02; (* of odti.p 1993 Jan 27
```

```
(* begin module describe.palinf *)                                      264
(*
name
      palinf: find palindromes, based on information theory

synopsis
      palinf(book: in, fout: out, palinfp: in, output: out)

files
      book: a book from the delila system
      fout: locations of palindromes
      palinfp: parameters to control palinf, one per line
         1. the minimum rsequence of the palindrome to detect.
            alternatively, if the number is negative, it is the
            desired significance of the detected peaks, given in
            standard deviations.
         2. (optional) size (integer).  the largest size palindrome allowed;
            base pairs across both halves of the site.  if omitted, the
            entire sequence is used (which may be very expensive).
            if this number is even, the next higher odd number will be used.
         3. (optional) if the first character of this line is an 'm' then
            palinf will plot palindrome size (m) versus information content
            (rsequence).  a sharply rising curve indicates a good palindrome.
            'x' means plot position (x) versus information content (rsequence).
            a different character, such as 'n', means to list
            the detected palindromes.
      output: messages to the user.

description
      Each piece of the book is searched for imperfect palindromes with
      significance determined by the first parameter in palinfp.
      There are two kinds of palindrome: even and odd, refering to
      the size of the palindrome in bases.  An odd palindrome will have a
      central base, while an even one will not have one.  Method of use:
      search without the 'm' option to pick out sites of interest.  Then use
      'm' under 'stringent conditions' or on a smaller fragement to see the
      structure of the palindrome.  The final r value will be the maximum of
      r values for all smaller palindromes.
      Note: equiprobable compositions are assumed for e(hnb).

examples
      the parameters [21/71/m] will locate the E. coli lac operator
      uniquely in the 401 bases surrounding the start of the lacZ transcript.

documentation
      Schneider, T.D., G.D. Stormo, L. Gold and A. Ehrenfeucht (1986)
      The information content of binding sites on nucleotide sequences.
      J. Mol. Biol. 188: 415-431.

author
      thomas schneider

bugs
      If parameter 2 is very large, spurious sites will be found.
```

```
technical notes
      Limiting the size of the palindrome will increase the search speed.
*)
(* end module describe.palinf *)
      version = 2.28; (* of palinf 1987 feb 10
```

```
(*
name
      parse: breaks a book into its components

synopsis
      parse(book: in, list: out, parsep: in, output: out)

files
      book: a book from the delila system
      list: a listing of the parts in the book
      parsep: parse parameters from the user
         if parsep is empty, default values are used.  otherwise, parsep
         must contain four lines corresponding to the variables
         that the user may reset.  they are:
            number of bases printed per line
            symbol to mark the end of sequences
            print header information
            print information about each sequence
            print raw sequences
         the last 3 items are boolean (true/false) values.  if you want to
         have the information, put a t (standing for true) at the beginning of
         the line.  if you do not want it, put an f (standing for false).
      output: messages to the user

description
      to parse is to break into component parts.  this program breaks a book
      into parts.  this allows one to easily look at sequences of a book
      without having to look at the book structure or the fancy listing
      provided by the lister program.

examples
      if parsep contains 60/./f/f/t then the sequences will be listed, with
      the '.' character ending each sequence.  all other information would be
      lost.

author
     thomas schneider

bugs
      only piece information is listed.

*)
(* end module describe.parse *)
      version = 2.20; (* of parse 1988 feb 24
```

```
(* begin module describe.patana *)
(*
name
      patana: pattern analysis

synopsis
      patana(pattern: in, anal: out, output: out)

files
      pattern: a pattern matrix, the output of the pattern learning
         program patlrn;
      anal: the analysis of the pattern matrix;
      output: for messages to the user;

description
      patana does some simple analyses of a pattern matrix.  for each
      position (i.e., row) of the matrix it calculates the:
         sum;
         average;
         variance;
         maximum;
         minimum.
      it also calculates the sum of each of those measures.

      the sum of sums is used in other calculations.  if all training
      sequences are the same length, this is the difference between
      the number of + class sequences and - class sequences added
      together to make the w matrix.

      the sum of the average is an estimate of the mean response to
      random sequences

      the sum of the variance is a variance that estimates the
      spread of responses to random sequences.  take the
      square root to obtain the standard deviation.

      the sum of the maxima is the largest response possible.

      the sum of the minima is the smallest response possible.

see also
      patlrn

author
      gary d. stormo (modified by tom schneider)

bugs
      none known
*)
(* end module describe.patana *)
version = 2.18; (* of patana 1987 jul 2
```

```
(*
name
      patlrn: pattern learning

synopsis
      patlrn(funcbook: in, funcinst: in, nfuncbook: in, nfuncinst: in,
            pattern: out, start: in, minmax: in, ignore: in, patlrnp: in,
            output: out)

files
      funcbook: the book of sequences belonging to the functional class;
      funcinst: the instructions for funcbook, for aligning the sequences;
      nfuncbook: the book of sequences for the nonfunctional class;
      nfuncinst: the instructions for nfuncbook, for aligning the seqs;
      pattern: the resulting wmatrix which separates the classes;
      start: a matrix for initializing wmatrix to. it is initialized to
            all 0's if this file is empty;
      minmax: to set the values of funcmin (the minimum value for a functional
            sequence) and nfuncmax (the maximum value for a nonfunctional
            sequence). if this file is empty they are set to 1 and 0,
            respectively, and vary along with the matrix;
      ignore: a file specifying regions of the sequences which are to be
            ignored in the learning process; the maximum number of regions
            which can be ignored is set by the constant 'maxignore';  the
            file must contain two integers per line, the first specifying
            the 5' end and the second the 3' end of the region to be ignored.
      patlrnp: parameter file for setting maxtimes, the number of times
            through all the sequences before stopping without a solution;
      output: for messages to the user.

description
      patlrn uses the 'perceptron' algorithm to find a weighting function
      (a 'wmatrix') which serves to distinguish the sequences in the two
      classes from one another.  our paper, stormo et.al., nar 10, 2995 (1982),
      describes the algorithm in detail and gives an example of its use.

see also
      patlst, patana, patser, patval

author
      gary d. stormo (modified by tom schneider)

bugs
      the section of code for ignoring regions of the sequences in the
      learning process (i.e., when the file 'ignore' is not empty) has
      been overlayed over the rest of the code, rather than worked into
      it, and consequently, using this feature can be quite inefficient.

technical note
      the program will be more efficient if the constant 'dnamax' in the
      module 'book.const' is made to be the size of the sequences used
      by the program.  for instance, setting it to whatever 'maxmatrix'
      is would be a good idea.
```

```
*)
(* end module describe.patlrn *)
version = 3.24; (* of patlrn 1986 dec 9
```

```
(* begin module describe.patlst *)
(*
name
      patlst:  lister of patlrn output.

synopsis
      patlst(pattern: in, patout: out, patlstp: in, output: out)

files
      pattern: the input pattern matrix to be reformatted;  this is the
               output of the program patlrn.
      patout:  the output reformatted pattern matrix.
      patlstp: a parameter file for specifying the pagewidth of the patout
               file;  must contain an integer as the first thing on the first
               line, which specifies the number of matrix elements to be printed

               across a page;  if this file is empty the pagewidth is set to the
               constant 'defpagewidth'.
      output:  for messages to the user.

description
      patlst takes the output from the patlrn program and reformats the
      pattern matrix to run horizontally across the page. it is broken
      so that it fits neatly on the page.  this is useful for making
      publishable copies of the pattern matrices.

see also
      patlrn, patana

author
      gary d. stormo

bugs
      none known
*)
(* end module describe.patlst *)
version = 1.08;  (* of patlst 1989 July 8
```

```
(* begin module describe.patser *)
(*
name
      patser: pattern searcher

synopsis
      patser(book: in, pattern: in, scale: in, patserp: in,
             values: out, inst: out, output: out)

files
      book: the book of sequences to be searched. only numbered sequences are
         searched.
      pattern: the pattern used to search with. this is the output of the
         pattern learning program, patlrn.
      scale: contains one integer, by which the values should be divided
         to bring them into the correct scale if a matrix from rseq was used.
      patserp: parameter file, to set the value of 'printmin', the minimum
         value of a site in order for it to be identified in the file 'values'.
         if this file is empty, 'printmin' is set to the functional sequence
         minimum of the pattern matrix.
      values: the sites, and their values, which are evaluated above 'printmin'.
      inst: the instructions to get the regions around the sites identified
         in the file 'values'.  the region obtained is identical to the pattern
         used in the search.
      output: for messages to the user.

description
      patser uses a pattern matrix, the output of the pattern learning program
      patlrn, to search a book of sequences.  each base in each sequence is
      used as the 'aligned base', and the sites which are evaluated above
      'printmin' are identified in the file 'values'.  instructions which can
      be used to obtain those sites, and the nucleotides around them over the
      region contained in the pattern matrix, are put into the file 'inst'.

      NOTE: if the pattern is off the end of the sequence it is
      nolonger reported.

see also
      patrln, patval

author
      gary d. stormo (modified by tom schneider)

bugs
      none known
*)
(* end module describe.patser *)
version = 2.31; (* of patser.p 1992 June 16
```

```
(* begin module describe.patval *)
(*
name
      patval: pattern evaluations of aligned sequences

synopsis
      patval(book: in, inst: in, pattern: in, scaleup: in,
            values: out, output: out)

files
      book: the book of sequences to be evaluated;
      inst: the instructions generating the book, for alignment;
      pattern: the wmatrix used to evaluate;
      scale: contains one integer, by which the values should be divided
         to bring them into the correct scale if a matrix from rseq was used.
      values: the value of each sequence in the book;
      output: for messages to the user.

description
      Patval uses a pattern matrix (the output of patlrn) to evaluate a
      book of aligned sequences.

see also
      delman.use.perceptron
      patlrn, patser, patlst, patana

author
      Gary D. Stormo

bugs
      none known
*)

(* end module describe.patval *)
version = 2.23; (* of patval 1989 Mar 29
```

```
(*
name
   pbreak: breaks a file into pages at a certain trigger phrase

synopsis
   pbreak(pbreakp: in, input: in, output: out, list: out)

files
   pbreakp: The parameter file which contains the trigger on one line.
      Only one trigger is allowed in pbreakp.
      The next line may contain one integer which represents the right most
      position (in characters, 1 is the first character on a line)
      where the trigger will be looked for.  Default is an enormous number.
   input: the file to break up
   output: the broken file
   list: where messages will appear.

description
   The program pbreak will go through a file, line by line, looking for a
   "trigger" phrase.  Upon finding the trigger on a line, pbreak will insert a
   "new page" mark at the beginning of the line.  This will cause the printer
   to start a new page at this line when the file is printed.  A page number is
   added and an alphabetical index of the lines containing the trigger strings
   and their page numbers is printed at the end of the output file.  The
   trigger phrase can be any string of characters and is in the file pbreakp.
   The pbreak program is thus useful for breaking up large files into
   workable-size chunks, or to make a large file more readable.

examples
   Pbreak has been used to make pascal source code easier to read and
   work with by using the trigger "procedure" to make a file which
   when printed has one procedure to a page.  Pbreak also has been used
   to make the delila manual, delman.  Delman is one large, continuous
   textfile, and pbreak is used to break delman into its formatted pages
   by using the parameters
        (@ begin module
        1
   which will only recognize modules that begin at the left margin.
   (Note: the @ in the example above must be replaced by a '*' to
   make the example work.  The (@ form fools the compiler, and prevents
   it from thinking I'm doing something funny.)

documentation
   delman.intro.organization: "technical notes"

author
   Patrick R. Roche
   modified by Tom Schneider

bugs
   none known

technical notes
```

```
   Three procedures, firstpage, makepage and lastpage contain instructions for        274
   forming new pages.  These are system dependent.  Constant pagelength
   determines the size of the page.

   If a line is too wide, the page number will not be printed, however the
   number will override the characters on the line in the index at the end.
   The constant "top" defines the maximum length of a buffer, thus the maximum
   length of an input line or a trigger.  The constant "pagewidth" defines the
   page width for numbering of pages and the printing of the index.  Pagewidth
   should be set to the desired page width - 1 to come out right.  The constant
   liston (true or false) indicates whether or not to display the index on the
   file list.

*)
(* end module describe.pbreak *)
version = 4.26; (* of pbreak.p 1993 Jan 27
```

```
(* begin module describe.pcs *)
(*
name
   pcs: partial chi squared

synopsis
   pcs(pcsp: in, list: out, output: out)

files
   pcsp:  A series of lines, each of which contains
     of 4 integers, representing the numbers of a,c,g,t.
   list: partial chi squares calculated for the integers
   output: messages to the user

description
   calculate the partial chi squared values of 4 bases

examples

documentation

see also

author
   Thomas Dana Schneider

bugs

technical notes

*)
(* end module describe.pcs *)
version = 1.02; (* of pcs.p 1991 August 23
```

```
(* begin module describe.pemowe *)
(*
name
      pemowe: peptide molecular weights

synopsis
      pemowe(book: in, list: out, output: out)

files
      book: any book from the delila system.  only one weight is given per
         piece in the book.  the first triplet of the piece is the first
         codon translated.
      list: a list of the piece numbers and names in the book, along with
         the molecular weight of the peptide and
         the number of atoms of the peptide for each piece.
      output: messages to the user.

description
      pemowe is designed to find the molecular weights of polypeptides
      that might be coded by a particular sequence.  it is to be used in
      cases where one knows where a particular peptide is, not when one wants
      the weights of all possible peptides.  one should use delila to
      construct the book.  the calculation of the weights takes into account
      loss of water for each peptide bond formed.  calculation ends at stop
      codons or at the end of the piece.

examples
      sanger et. al., nature 265: 687 (1977) on page 692 gives a list of
      calculated molecular weights from phix174.  these can be used to test
      pemowe, by using the delila instructions expepin.  the largest deviation
      from sanger's numbers is for gene j at 3 percent.

documentation
      data is from the crc handbook of chemistry and physics 60th ed, 1980.

author
      thomas d. schneider

bugs
      only one peptide per piece is calculated.  one could write another
      program that predicted peptides (like lister), and generated
      instructions for pulling out those peptides using delila.

*)
(* end module describe.pemowe *)
version = 2.20; (* of pemowe.p 1993 Jan 19
```

```
(* begin module describe.prgmod *)
(*
name
      prgmod: programming modules for the delila system

synopsis
      prgmod(input: intty, output: out)

files
      input: interactive file used for testing the program
      output: messages to the user.

description
      prgmod is a set of generally useful modules for programming.
      these include procedures for interactive input/output, producing
      bars of numbers for graphs (called 'numbars') and sorting of
      arrays with a very fast algorithm.
         successful compilation and running of the program indicates
      that the modules are correct.  the program is interactive, so
      to test the modules, follow the instructions prgmods provides.

see also
      delmod, module, alist (uses numbar), index (uses quicksort)

author
      thomas d. schneider

bugs
      none known

technical notes
      the interactive routines may have to be changed when the program
      is transported.

*)
(* end module describe.prgmod *)
version = 4.11; (* of prgmod.p 1991 Apr 22
```

```
(* begin module describe.rawbk *)
(*
name
      rawbk: make a raw sequence into a book

synopsis
      rawbk(raw: in, book: out, input: intty, output: out)

files
      raw: a file with a sequence on it, that is only the letters
         a,c,g,t, or u, with any spacing and carriage returns.
      book: a file which contains the sequence in the book form
         such that it will interface with the delila system programs.
      input: the interactive input from the keyboard. rawbk needs to
         get some information from the user to name the sequence.
      output: where error messages will appear.

description
      The purpose of this program is to allow one to rapidly create a book
      from a raw sequence. rawbk will take a 'raw' sequence and put it into
      the standard form of a book so that the delila system programs can be
      used on the sequence.  The user is asked for one name, which will
      become the name of all things in the book (title, organism,
      chromosome and piece).
         The program reads thru 'raw', keeping track of characters and lines.
      It will flag any letters other than 'a','c','g','t', or 'u', that appear
      in the file and note their locations. it will count the bases. if any
      characters were flagged, or any other error occurs, rawbk will
      put 'halt' into the book, in the same form the librarian does, to
      prevent further use of the book.  Otherwise, the book is constructed
      to contain one piece of sequence.  The coordinates begin with base 1.

see also
      makebk

author
      Thomas D. Schneider

bugs
      The program should use book writing routines from delmods, but it
      has not been updated yet.

*)
(* end module describe.rawbk *)
version = 3.12; (* of rawbk 1988 july 9
```

```
(*
name
   ref2bib: refer to bibtex converter

synopsis
   ref2bib(refs: in, bib: out, output: out)

files
   refs: reference list used by refer
   bib: reference list used by bibtex
   output: messages to the user

description
   The program converts from refer to bibtex reference list formats.

documentation
   man refer, LaTeX reference manual

author
   Thomas Dana Schneider
   National Cancer Institute
   Laboratory of Mathematical Biology
   Frederick, Maryland
   toms@alum.mit.edu

bugs
   only a few of the refer types have been converted.
   see comments in the code for the entire list.

*)
(* end module describe.ref2bib *)
version = 1.46; (* of ref2bib 1988 December 14
```

```
(* begin module describe.refer *)
(*
name
      refer: print the references in the pieces of a book

synopsis
      refer(book: in, list: out, output: out)

files
      book: any book from the delila system
      list: references in the pieces of the book, organized by organism
         chromosome and map location.
      output: messages to the user.

description
      refer is a convenient way to obtain the references for pieces.
      since each piece note contains other information, this will also be
      printed.

author
      thomas schneider

bugs
      references have no standard format, so the output can not be
      formatted more than what is in the book.

*)
(* end module describe.refer *)
version = 2.06; (* refer 1986 dec 2
```

```
(* begin module describe.reform *)
(*
name
      reform: raw sequences reformatted

synopsis
      reform(fin: in, fout: out, input: intty, output: out)

files
      fin: the raw sequences to be reformatted.  the file must contain
         only the letters: 'a', 'c', 'g', 't', and/or 'u'.
      fout: the reformatted sequence.
      input: the user defines how to reformat the sequence:
         reformat - the sequence is only reformatted.
         invert - the order of the sequence stays the same, but the bases
            are complemented.
         complement - the order of the sequence is reversed and the bases
            are complemented.
         the user also specifies the number of bases to be printed on
         each line of fout.
      output: messages to the user.

description
      Reform allows one to type a file containing raw sequence data typed in
      whatever form is convenient, and to convert it into a form that
      the merge program can use to compare to a second typed copy.
      For example, when a sequence is to be entered from two strands,
      the sequence and its complement, one can enter both strands and then
      invert the second strand for comparison by merge.  Alternatively,
      the second strand could be entered backwards and the complement taken
      using this program.

see also
      merge

author
      Thomas Schneider

bugs
      The first typed line is ignored because of a problem with
      the standard input procedures on Unix.  Simply type a carriage
      return when the program starts up.

*)
(* end module describe.reform *)
version = 1.20; (* of reform 1992 September 12
```

```
(* begin module describe.rembla *)
(*
name
      rembla: remove blanks from ends of lines in a file

synopsis
      rembla(fin: in, fout: out, output: out)

files
      fin: a text file
      fout: a copy of fin with trailing blanks removed from all lines,
         any blank lines at the end of the file will also be removed.
      output: messages to the user

description
      blanks can creep onto the end of lines in a file without one knowing
      it, either by the computer system, from transportation or an editor.
      this program removes those blanks, so that less storage is needed
      for the file.  some programs require that there be no blank lines in the
      file,  yet transportation can generate blank lines at the end of
      the file.  this program will remove such lines.

author
      thomas d. schneider

bugs
      none known

*)
(* end module describe.rembla *)
version = 2.08; (* of rembla 1986 dec 12
```

```
(*
name
      rep: records repeats between sequences in two books

synopsis
      rep(hlist: in, xbook: in, ybook: in,
             fout: out, pout: out, repp: in, output: out)

files hlist: a list of helices for xbook and ybook generated
             by the program helix.
      xbook: a book from the delila system.
      ybook: a book from the delila system.
      fout:  a file containing the following information about each repeated
         sequence that satisfies the criteria of repp:
         *  the 5 prime ends of the two occurrences of the repeat.
         * rlength, length of the repeated sequence.
         * distance: if direct repeats, the number of bases from five prime end
            to five prime end of each repeat; if inverted repeats, the number
            of bases from three prime end to five prime end (i.e., pseudo-loop
            distance). in every case, the smallest possible distance is given.
      pout: a file containing information about palindromes (only filled when
            inverted repeats are found in related sequences, see below).
      repp: input parameter file, must contain 3 characters, one per line.
               this may be followed by 4 integers, one per line.
         * mode of repeat:
            d = direct repeat (xbook and ybook have opposite directions)
            i = inverted repeat (xbook and ybook are in the same direction)
         * the types of xbook and ybook used in helix program:
            u = unrelated (any two sequences - no distances are calculated)
            r = related (sequences derived from the same piece of dna. the
                coordinate numbering of both books must be the same in order
                to calculate distances.)
         * the energies of hlist reflect the composition of the repeat
            e = "energies" are to be reported
            n = no "energies" are to be reported
         * minimum number of bases in a repeat to be recorded.
         * maximum number of bases in a repeat to be recorded.
         * minimum distance between repeated sequences to be recorded.
         * maximum distance between repeated sequences to be recorded.
      output: messages to the user.

description
      rep uses information generated by the helix program to record the
      occurrences of repeated sequences of dna. helices are interpreted as
      repeats, direct or inverted depending upon the input sequences.
      repeats that meet the criteria of minimum length and minimum and/or
      maximum distance between half repeats are reported in fout.
      palindromes are reported in pout.

see also
      helix, matrix, keymat

author
```

```
bugs  1.  when xbook and ybook have sequence in common, hlist reports
      each "helix" twice.  rep is able to eliminate duplicates only
      when xbook and ybook overlap completely.  thus in cases of
      partial overlap, some repeats may be duplicated.
      2.  rep uses external coordinates to calculate distances and will
      bomb with complicated coordinates. *)
(* end module describe.rep *)
version = 1.73; (* of rep.p 1993 Jan 27
```

```
(*
name
        repro:  make multiple copies of a file

synopsis
        repro(fin: in, fout: out, input: intty, output: out)

files
        fin: any file of which multiple copies are wanted
        fout: the new copies of the desired files
        input: the paramiter file giving (n), the number of copies
           to be made. this is the interactive file.
        output: messages to the user

description
        This tool enables the user to make any number of copies
        of a file.  Each copy begins on a new page.

examples
        A typical use for this program is to make multiple copies
        of the delman manual after breaking it into pages with
        the program break.

see also
      break

author
        Billie Hall Lemmon

bugs
        none known

*)
(* end module describe.repro *)
version = 2.04; (* repro 1986 dec 12
```

```
(* begin module describe.rf *)
(*
name
   rf: calculate Rfrequency

synopsis
   rf(input: in, output: out)

files
   input: interactive input from the user
   output: messages to the user

description
   calculate Rfrequency as

   Rf = - log (number of binding sites / genome size)

   where the log is to the base 2, giving the result in bits.

documentation
   Schneider, T.D., G.D. Stormo, L. Gold and A. Ehrenfeucht (1986)
   The information content of binding sites on nucleotide sequences.
   J. Mol. Biol. 188: 415-431.

author
   thomas d. schneider

bugs
   The program depends on interactive procedures on a particular computer
   and so may need to be modified on transportation.

*)
(* end module describe.rf *)
version = 1.07; (* of rf 1988 October 14
```

```
(*
name
   ri: Rindividual is calculated for every site in the aligned book

synopsis
   Ri(inst: in, book: in, rsdata: in, values: in, rip: in,
      xyin: out, sequ: out, ribl: out, output: out)

files
   inst: delila instructions of the form 'get from 56 -5 to 56 +10;'
      (This file may be empty, in which case the sequences will be
      aligned by their 5' ends.)
   book: the book generated by delila using inst
   rsdata: data file from rseq program
   values: a file containing the values of the objects to which the Ri
      values are to be compared.  The file may be empty.
   rip:  Parameters to control the program.
      On the FIRST LINE are the FROM and TO over which to do the Ri
      calculation.  These must not exceed either of those in the inst/book or
      the rsdata.

      The SECOND LINE defines the column of the values file to use.

      The THIRD LINE: two integers: the lowest and highest evaluation to report
      to xyin and sequ.  If the first character of the line is 'a' then all
      evaluations are reported.  Otherwise two real numbers are expected.
      Sequences within this range are printed to xyin and to sequ depending
      also on the fifth parameter.

      The FOURTH LINE: two integers: the lowest and highest evaluation to
      report to xyin and sequ.  If the first character of the line is 'a' then
      all evaluations are reported.  Otherwise two real numbers are expected.
      Sequences within this range are printed to xyin and to sequ depending
      also on the fifth parameter.

      The FIFTH LINE determines whether or not to produce any raw sequences in
      the sequ file.  If the first character of the line is 'p', sequences
      selected according to the third and fouth parameters are printed to sequ
      file.  (This is a complete on-off switch for the sequ file.)

      The SIXTH LINE determines whether or not to print the sequence of the
      site being analyzed.  If the first character is 'p' then the sequence is
      printed to the xyin file.

      The SEVENTH LINE determines whether or not to print sequences which have
      a partial site.  The problem is that if there is part of a site, then the
      Ri value is questionable, depending on where the deletion was.  The best
      analysis would not use a partial site, as it messes up the statistics.
      If the first character is:

         n  Don't print the line at all.
         i  Keep the line, but force the Ri value to be -infinity.
            This allows the lines of xyin to be correlated to the values
```

```
        still.
      - (any other character): print as it is.

    The EIGHTH LINE determines what to do when f(b,l) = 0. Positions for
 which f(b,l) = 0 will have negative infinity in the Ri(b,l) table.
 The letter 's' means to use Rodger Staden's method of giving 1/(n+t),
 where t is a non-negative integer following the 's'.  When t = 0, it
 is Staden's method.  Using t=1 may be the most logical choice.  If
 there is no 's', the program expects a number which the value for
 negative infinity.  It should be a value sufficiently below zero so
 that sites that are being excluded from the definition according to
 f(b,l) are separated from the true sites.  -1000 is a useful value, as
 it will always displace sites with exceptions far away from zero.

  xyin: input to the xyplo program.
     The Ri(b,l) table is reported in comments in the table, along with
     the value of the consensus (largest possible evaluation) and the
     anti-consensus (smallest possible evaluation).  The rest of the
     file contains these columns of data:
        piece number
        piece name
        length of region analyzed on this piece
        sequence region analyzed
        Rindividual for the piece
        value from the values file (or 0 if values is empty)

  sequ: the raw sequences reported to xyin if any selection is made
     (fourth line of rip file).  These end in periods, so they can be
     given to makebk to create a book.

  ribl: weight matrix Ri(b,l).  The information content for each base b at
     each position l, in bits.  Lines that start with * are notes.  the next
     line contains the matrix FROM-TO coordinates, this is followed by the
     matrix in the order A, C, G, T from FROM to TO.

  output: messages to the user

description
  The program determines the individual informations of the sites in the book
  as aligned by the instructions, according to the frequency table given in
  the rsdata file.  The program calculates the Ri(b,l) table:

      Ri(b,l) := 2 - (- log2( f(b,l)))

  and sums this up for each sequence.  Ri is defined so that the average of
  the Ri's for a set of sequences is Rsequence.  However, if the sequences are
  incomplete, the average will probably be less than Rsequence.  The xyin
  output is ready to read into the xyplo program for plotting and linear
  regression.  The ribl matrix is ready to be used to scan sequences with the
  scan program.

  The program can be used in subtle ways.  For example, one can analyze the
  individual information of the left half of a binding site.  This result can
  then be used in the values file to compare against the analysis of the right
```

```
author
   Thomas D. Schneider

examples

rip:

-10 +10       From-to range to do the evaluation
1             column of the values file to copy to xyin
a 0 1000      lowest to highest Ri to put in xyin and sequ (a = any)
a -1000 +1000 lowest to highest Value to put in xyin and sequ (a = any)
n             p means print sequence to the sequ file
p             p means print sequence to the xyin file
-             -: accept all sites; n: no partials; i: partials -> -infinity
s 1           s: use Staden's Method, f(b,l)=1/(n+t); else negative infinity

documentation

@article{Staden1984,
author = "R. Staden",
title = "Computer methods to locate signals in nucleic acid sequences",
journal = "Nucl. Acids Res.",
volume = "12",
pages = "505-519",
year = "1984"}

and

@unpublished{SchneiderRi,
author = "T. D. Schneider",
title = "Measuring the Information of Individual Binding Sites
on Nucleotide Sequences",
comment = "indiv.tex",
note = "in preparation"}

see also
   rseq.p, xyplo.p, scan.p

bugs

technical notes

*)
(* end module describe.ri *)
version = 1.95; (* of ri.p 1993 January 27
```

```
(* begin module describe.riden *)
(*
name
   riden: ring density graph

synopsis
   riden(color: in, xyin: out, output: out)

files
   color: output of the ring program
   ridenp: parameter file for this program.  Two lines:
           First line: largest radial distance recorded
           Second line: number of bins to store the data in
   xyin:  histogram of the density
   output: messages to the user

description
   This program converts the graph generated by the ring program into
   a form that allows one to see if the results are as expected.

examples

documentation

see also
   ring.p

author
   Thomas Dana Schneider

bugs
   Program only works for D=2.  The curves don't match for D=4, but
   do for higher dimensions.  It is not obvious why.

technical notes

*)
(* end module describe.riden *)
version = 1.28; (* of riden.p 1989 November 25
```

```
(*
name
   ring: z space ring

synopsis
   ring(data: in, ringp: in, color: output, output: out)

files
   data: set of Gaussianly distributed variables from the program gentst.
   ringp: parameters:
     first line: total dimensionality D.
     second line: number of points to do.  If the end of the data is reached,
       the actual number of points generated is reported to output.
     third line: number of steps to generate the fD(r) graph.
       (The range for this is always -2.5 to +2.5 on both x and y axes.)
       If the number of steps is less than 1, then no smooth graph is done.
     fourth line: a real number, "0 <= partial <= 1" by which to multiply
       the actual fD(r) density by to obtain the density reported to the
       color file.  This allows one to tone down the gray scale, or to
       avoid having the highest density of color equal the lowest (as when
       the hue is used and a hue of 1 is the same as a hue of 0).
     fifth line: printing of data on plot (one character):
       d=dimension,p=dimension+point,a=all,n=none

   color: a xyin file for input to the xyplo or riden program.  The columns are:
        1   symbols:    f=from fD(r), s = simulated point');
        2   x:          x coordinate
        3   y:          y coordinate
        4   xwidth:     width of symbol on x axis
        5   ywidth:     width of symbol on y axis
        6   density:    density
        7   inverse:    1 - density (for inverse plotting)
        8   maximum:    MAXimum density
        9   minimum:    MINimum density
       10   maximum:    MAXimum density
       11   minimum:    MINimum density
       12   partial:    partial density for grey tones

      Partial is the largest density allowed.  When plotted in color, hues come
      from a color wheel in which the highest color is almost identical to the
      lowest color.  That is the color of hue=1 is almost identical to the
      color of hue = 0.  To avoid this effect, make partial less than 1.0.  A
      partial less than 1.0 also avoids completely black gray scale plots.

   output: messages to the user, number of points generated.

description
   Simulate mapping from many-dimensional to 2-dimensional Z space.  Sets of D
   Gaussian values are read from the data file, squared, summed and square
   rooted.  The x and y value in Z space is determined from an angle and a
   radius.  The angle is found from the last two Gaussian values, while the
   radius is determined by the noise (rms) for all dimensions.  The statistical
   function fD(r) is to be graphed in color or gray scale using xyplo, while
```

```
   the simulated points are graphed as points on top of the smooth fD(r)
   function.  The program output is ready to read into the xyplo plotting
   program.

examples
   ringp used for generating figures:
16      total dimensionality
100     number of points to do
128     steps for plotting smooth fD(r) graph
0.50    partial
d       d=dimension,p=dimension+point,a=all,n=none

   xyplop used for generating figures:
2 2        zerox zeroy         graph coordinate center
x -2.5 2.5 zx min max          (character, real, real) if zx='x' then set xaxis
y -2.5 2.5 zy min max          (character, real, real) if zy='y' then set yaxis
10 10      xinterval yinterval number of intervals on axes to plot
4 4        xwidth    ywidth    width of numbers in characters
1 1        xdecimal  ydecimal  number of decimal places
5 5        xsize     ysize     size of axes in inches
x
y
c          zc                  if zc='c' then a crosshairs put on zero of x and y
n 2        zxl base            if zxl='l' then make x axis log to the given base
n 2        zyl base            if zyl='l' then make y axis log to the given base
           *************************************************************
2 3        xcolumn   ycolumn   columns of xyin that determine plot location
1          symbol column       the xyin column to read symbols from
4 5        xscolumn  yscolumn  columns of xyin that determine the symbol size
10  8  7   hue saturation brightness   columns for color manipulation
           *************************************************************
r          symbol to plot      c(circle)bd(dotted box)x+Ifgpr(rectangle)
b          symbol flag         character in xyin that indicates that this symbol
-1.0       symbol sizex        side in inches on the x axis of the symbol.
-1.0       symbol sizey        as for the x axis, get size from yscolumn
n          no connection (example for connection is c- 0.05 for dashed 0.05 inch)
n  0.05    linetype  size      linetype l.-in and size of dashes or dots
           *************************************************************
r          symbol to plot      c(circle)bd(dotted box)x+Ifgpr(rectangle)
f          symbol flag         character in xyin that indicates that this symbol
-1.0       symbol sizex        side in inches on the x axis of the symbol.
-1.0       symbol sizey        as for the x axis, get size from yscolumn
n          no connection (example for connection is c- 0.05 for dashed 0.05 inch)
n  0.05    linetype  size      linetype l.-in and size of dashes or dots
           *************************************************************
c          symbol to plot      c(circle)bd(dotted box)x+Ifgpr(rectangle)
s          symbol flag         character in xyin that indicates that this symbol
0.0858     symbol sizex        side in inches on the x axis of the symbol.
0.0858     symbol sizey        as for the x axis, get size from yscolumn
n          no connection (example for connection is c- 0.05 for dashed 0.05 inch)
n  0.05    linetype  size      linetype l.-in and size of dashes or dots
           *************************************************************
g          symbol to plot      c(circle)bd(dotted box)x+Ifgpr(rectangle)
g          symbol flag         character in xyin that indicates that this symbol
```

```
-1.0      symbol sizex        side in inches on the x axis of the symbol.             293
-1.0      symbol sizey        as for the x axis, get size from yscolumn
n         no connection (example for connection is c- 0.05 for dashed 0.05 inch)
n  0.05   linetype  size      linetype l.-in and size of dashes or dots
          **********************************************************************
.
          **********************************************************************


Useful color parameters are:

8 6 10  Light density plot, printable on a black and white device (best).
8 7 10  Dark density plot, printable on a black and white device.
6 8 10  Color plot, red background.
7 8 10  Color plot, purple background (neat).
6 7 10  Color and density varying to make the simulated points easy to see.
        (red background)
7 6 10  Color and density varying to make the simulated points easy to see.
        (white background - lovely!)

Warning: since the program has changed, these may no longer be correct.

documentation
   ccmm

see also
   gentst.p xyplo.p riden.p

author
   Thomas Dana Schneider

bugs
   none known.  Confirm that the density distribution is correct by
   using program riden.

technical notes

*)
(* end module describe.ring *)
version = 3.00; (* of ring.p 1989 Nov 25
```

```
(*
name
      rndseq: generate random dna sequences

synopsis
      rndseq(fout: out, rndseqp: in, output: out)

files
      fout: the random sequence
      rndseqp: parameters to control the generation of the sequence,
         on 4 lines:
         number (integer): the number of sequences to generate;
         length (integer): the length of each sequence;
         a c g t (4 integers): the proportions of bases desired;
         seed (real): a number between 0 and 1 is the starting seed
            for the random number generator.  a number outside
            this range indicates that the date and time should
            be used.  the date and time 83/10/17 20:15:32
            makes a seed of 0.235102710138.  the date-time
            is used backwards to assure that 1) the seed is
            always unique, and 2) it varies rapidly with time.
      output: messages to the user.

description
      rndseq creates randomly generated dna sequences, separated
      by periods.  the number, length and composition of the sequences
      are all specified by the user.  the user can also set the
      start point (seed) of the pseudo-random number generator.  if the
      same seed is given at a later time, then the same series of
      bases will be produced.  alternatively, the user can have the
      program use the current date and time to create a unique seed.

examples
      5  sequence
      100   long
      1 1 1 1  equiprobable sequences
      2  the date and time are used to start the random generator.

author
      thomas dana schneider

bugs
      none known

technical notes
      the number of characters per line is set by constant linelength.
*)
(* end module describe.rndseq *)
version = 1.08; (* of rndseq 1986 nov 14
```

```
(* begin module describe.rseq *)
(*
name
      rseq: rsequence calculated from encoded sequences

synopsis
      rseq(encseq: in, cmp: in, rsdata: out, wmatrix: out, output: out)

files
      encseq: the output of the encode program
      cmp: a composition from the comp program.
          if cmp is empty, then equal frequencies are assumed.
      rsdata: a display of the information content of each position
         of the sequences, with the sampling error variance.
         This output is ready to be used as input to rsgra or as data
         for genhis for plotting.
      wmatrix: a weight matrix for searches.
      scale: contains an integer that is the amount by which the values
         in wmatrix have been multiplied.  By dividing by this scale up
         factor the wmatrix values will be normalized to bits.  This allows
         the wmatrix to contain integers.
      output: messages to the user.

description
      Encoded sequences from encseq are converted to a table of frequencies
      for each base (b) at each aligned position (l).  rsequence(l)
      and the variance var(hnb) are calculated and shown along with
      their running sums.  rsequence and the variance due to sampling
      error are shown for the whole site, but the running sums let one
      find rsequence and the variance for any subrange desired.
         n, the number of example sequences may vary with position, so
      both n and e(hnb) are shown.
         A w matrix, w(b,l) is generated that can be used to search
      for sites.  When applied to the original aligned sequences, the
      average of the individual values will be rsequence.  (this will
      not be exactly true if the number of samples varies with
      position in the site, n(l)).

documentation
      Schneider, T.D., G.D. Stormo, L. Gold and A. Ehrenfeucht (1986)
      The information content of binding sites on nucleotide sequences.
      J. Mol. Biol. 188: 415-431.

see also
      encode, comp, encfrq

author
      Thomas D. Schneider

bugs
      Does not handle di-nucleotides or longer oligos

technical notes
      Constants maxsize (procedure calehnb) and kickover (procedure
```

```
      makehnblist) determine the largest n for which e(hnb) is used.  Above
      this, ae(hnb) is used.  Do not set these below 50 without careful
      analysis.  Other constants are in module rseq.const.

*)
(* end module describe.rseq *)
version = 5.32; (* of rseq.p 1990 Oct 2
```

```
(* begin module describe.rsgra *)
(*
name
   rsgra: rsequence graph

synopsis
   rsgra(rsdata: in, picture: out, rsgrap, [[wave: in, ]]marks: in,
         output: out)

files
   rsdata: data file from rseq program
   picture: graph of rsequence in PostScript
   rsgrap: parameters to control the program.
      first line: two integers that define the from-to range to display
[[
   wave: parameters to control the program.  empty file means no cosine
      wave, otherwise the parameters of the wave are given one per line:
      extreme: char;  h or l, the extreme high or low point on the curve
         defined by the wavelocation and wavebit
      wavelocation: real; the location in bases of the extreme
      wavebit: real; the location in bits of the extreme
      waveamplitude: real; the amplitude of the wave in bits
      wavelength: real; the wave length of the wave in bases
]]
   marks: an empty file or a set of integers, one per line that are the
      locations of bases that should be specially marked on the graph.
      If the first line of the file begins with the letter 'b' and is
      followed by a real number, then this number defines the location of
      a bar to be placed on the graph immediately after the position given.
   output: messages to the user

description
   Rsgra generates a graph of Rsequence versus position l.
[[
   The user may overlay a cosine wave on the graph, to signify
   the orientation of the major grove, for example.
]]

   See the discussion about the REMOVE feature in makelogo.p.

author
   Thomas D. Schneider

bugs
   none known

*)
(* end module describe.rsgra *)
version = 4.99; (* of rsgra.p 1992 July 21
```

```
(*
name
      rsim: Rsequence simulation

synopsis
      rsim(rsimp: in, cmp: in, xyin: out, output: out)

files
      rsimp: paramters to control the program:
         n: number of sequences to use to generate each fbl(simulated)
         rangelow,
         rangehigh: low and high bounds of the range of the matrix
         Rs: estimated value of Rsequence from the rsgra program
 SD: Standard Deviation of Rs based on sample size from the rsgra
    program.
            This defines the range Rslower = Rs - SD; Rsupper = Rs + SD.
         seed: a real number between 0 and 1 used to start the random
            number generator.  The date and time is used if this
            number is outside 0 to 1.  (N.B. if the system random
            number generator has been used in procedure rnd, then
            this parameter will have no effect.)
         simulations: number of fbl(true) to make
         Rtlower: lower limit to Rsequence(true) to work with.  This allows
            one to remove the small ones and get on with the ones of interest.
         Rtupper: upper limit to Rsequence(true) to work with.
         selection: if the first character of the line is 's', then only
    those points which fall in the Rslower to Rsupper range are put
    into xyin.  (Ie, only the 'p' values.)  This allows very large
    crunches to be done which don't create such a large xyin file.
      cmp: composition file from comp program.  If it is empty, the program
         will assume equiprobable bases.
      xyin: output of the program, input to the xyplo program
         column 1: values of R(simulated) that fall within the Rslower
                   and Rsupper range are indicated by a 'p', others by 'n'.
         column 2: Rsequence(true)
         column 3: Rsequence(simulated)
      output: messages to the user.

description
      Rsim stands for Rsequence-simulation.  The program generates a set of
      Rsequence values to determine the variation of Rsequence for small sample
      sizes.

      Method.

      A frequency table is constructed with zero information content, namely it
      contains 0.25 in all positions (l) and bases (b).  This table, fbltrue,
      is 'evolved' by altering the frequencies until it has an information
      content Rsequence(true) (=Rtrue) at least as high as Rtlower.

      A set of n sequences is generated using the fbltrue probabilities, and
      the information content, Rsimulated, is calculated for the set.  We
      select out those Rsimulated values which fall within the range of the
```

Rs+/-SD. This is repeated many times. The distribution of Rtrue values (which correspond to the selected Rsimulated values) represents the range of possible information contents of frequency tables which could have produced the observed results. In this way, we bootstrap ourselves to get the range. Note that SD is only a measure of small sample size.

Use.

Run an information analysis of the sites. This analysis determines n, rangelow and rangehigh for the rsimp. From the output of rseq (rsdata file), determine Rs and SD over the same range. Begin with only a few simulations. It is preferable to determine how long each simulation takes using at timing program like the UNIX /usr/5bin/time, so that the time for the final simulations can be predicted. 10,000 simulations is sufficient for the final analysis. Set Rtlower and Rtupper wide at first to be sure to capture the whole distribution.

Graph the results with the xyplo program, using the rsim.xyplop file for parameters. The output looks like:

```
             Rsimulated
             |                     .  . .
             |                   .  .
             |                  .  .
   Rs + SD   |               o  o
   Rs        |              oo o
   Rs - SD   |            ooo
             |          ..
             |         .
             |     .
             |   .
             | ..
             ----------------------------
                      Rtrue

               ^                        ^
               Rtlower                  Rtupper
```

The program choses a random number between Rtlower and Rtupper, Rtrue. Then it creates the fbltrue matrix with all 0.25 values. This places Rtrue at 0 initially. The matrix is evolved up to the current Rtrue value. Therefore the set of all fbltrue matricies should have a flat information content distribution. YOU MUST CHECK THAT THIS IS TRUE!! Copy the xyin file to the name 'data' and use the genhis program with these parameters:

```
c 2
x n 30
```

to get a histogram of the distribution of Rtrue, coming from column 2 of the file. The distribution should be reasonably flat over the entire region of the small circles (o) above. If it is not, you must determine what is wrong before continuing.

Those small circles represent the range that Rs +/- SD slices horizontally from the distribution of Rtrue versus Rsimulated. Recall that an each Rtrue leads to an fbltrue from which a single simulation of n binding sites is created; the information content of that is Rsimulated.

So we want the distribution of Rtrue within the bounds of the slice. To do this, we select that slice for analysis. In UNIX, we pull out all lines from xyin which have 'p' in them (p means: "plot this"). Use:

```
grep p xyin > data
```

Then run genhis with these parameters:

```
c 2
p g
x n 30
```

Notice how well or poorly the plotted gaussian ("p g") fits your distribution. If it is a good fit you are done. Take the standard deviation which genhis provides. Use the original Rsequence value for the mean. (The mean found on the genhis listing this way will be approximately Rsequence, but it has been created by passage through the simulation, so is not as good as the orginal data.)

documentation


```
@article{Schneider1986,
author = "T. D. Schneider
 and G. D. Stormo
 and L. Gold
 and A. Ehrenfeucht",
title = "Information content of binding sites on nucleotide sequences",
journal = "J. Mol. Biol.",
volume = "188",
pages = "415-431",
year = "1986"}

@article{Stephens.Schneider.Splice,
author = "R. M. Stephens
  and T. D. Schneider",
title = "Features of spliceosome evolution and function
inferred from an analysis of the information at human splice sites",
journal = "J. Mol. Biol.",
volume = "228",
pages = "1124-1136",
year = "1992"}
```


see also

rseq, xyplo, genhis, rsim.xyplop

author

Thomas D. Schneider

```
bugs                                                                   301
      Does not handle di-nucleotides or longer oligos

technical notes
      Constants maxsize (procedure calehnb) and kickover (procedure
      makehnblist) determine the largest n for which e(hnb) is used.  Above
      this, ae(hnb) is used.  Do not set these below 50 without careful
      analysis.  Other constants are in module rsim.const.

      Although it is possible to create more than one Rsimulated from
      each Rtrue, this causes vertical streaks on the graph, and so
      will distort the simulation.  It's better to get a completely
      clean one each time.

      Originally, a psudo random generator was used to create fbltrue from a
      random matrix (rather than 0.25) but this causes problems because such a
      matrix contains information and so low information points are under
      represented and higher ones over represented.  This distorts the
      statistics!

      The program contains a portable random number generator.  Unfortunately
      this can be 10 times slower than the non-portable one available on most
      systems.  The procedure rnd allows one to switch between the two.  When
      the system generator is used, one may find that the random numbers repeat
      exactly from one run to the next.  The seed parameter would not affect
      the results.  To avoid this problem, the random number generator is run
      until the requested seed is produced, within the tolerance given by the
      constant seedtolerance.  The runs are displayed on the output.

*)
(* end module describe.rsim *)
version = 2.17; (* of rsim.p 1993 January 26
```

```
(* begin module describe.same *)
(*
name
      same: counts the number of lines that are identical in two files

synopsis
      same(a: in, b: in, output: out)

files
      a: any file
      b: a file to be compared to file a
      output: messages to the user.

description
      same counts the number of lines that are identical in two files,
      a and b.  if the files are identical up to the end of one file,
      but the other file continues, same counts the identical lines and
      tell the user which file is shorter.   no lines are examined after
      the first line that differs between the two files.
         blanks at the ends of lines are ignored.

see also
      merge

authors
      britta swebilius singer and thomas d. schneider

bugs
      none known

*)
(* end module describe.same *)
version = 1.11; (* of same 1985 apr 20
```

```
(*
name
   scan: scan a book with a wmatrix and generate a vector

synopsis
   scan(book: in, ribl: in, scanp: in, data: out, output: out)

files
   book: a book from the delila system
   ribl: a weight matrix from sites or ri programs.
      Lines that start with * are notes.  the next line contains the matrix
      FROM-TO coordinates, this is followed by the matrix in the order A, C, G,
      T from FROM to TO.
   scanp: parameters to control the program.
      seqs: One integer on the first line is the number of sequences to scan to
      produce the vector.  0 = none, positive = that number; negative = all.

      Ri cutoff: One real on the second line is the information content at or
      above which to report in the data file.

      Probability cutoff: One real on the third line is the lowest probability
      which to report in the data file.  The probability of a site is determined
      from the mean and standard deviation of the Ri distribution.

      range: two integers that define the FROM-TO range of the ribl matrix to
      use.

      ways:  One integer.  2 means scan both the sequence and its complement.
 1 means simply scan the sequence.  0 means to let the program figure
 it out.  The program determines the symmetry of the matrix.  If it is
 symmetrical, it will only scan one way.  If it is asymmetrical, both
 scans are done.

   data: The results.  Comments are lines that begin with '*'.  The columns are
      defined in comments in the file.  The matrix is searched over both the
      sequence and its complement.  Ri is reported, as is the Z and probability
      based on the mean and st.dev.
   output: messages to the user

description
   The Ri(b,l) weight matrix is scanned across the sequences in the book to
   produce a vector.

examples

documentation

see also
   sites.p ri.p genhis.p

author
   Thomas Dana Schneider
```

```
technical notes
   The mean and standard deviation of the Ri distribution are stored just
   after the Ri(b,l) table in the ribl file.  They are produced automatically
   by the ri program.

*)
(* end module describe.scan *)
version = 1.92; (* of scan.p 1993 January 26
```

```
(*
name
      search: search a book for strings

synopsis
      search(book: in, inst: out, result: out, input: intty, output: out)

files
      book: any book from the Delila system
      inst: Delila instructions of the form 'get from 56 -5 to 56 +5;'
         that define the location of found strings.  one must turn on printing
         to the inst file to obtain these (see below).
      result: a transcript of the results seen on the output file.
         Lines not containing numerical data begin with an '*' so that
         they can be ignored by other programs such as genhis and xyplo.
      input: typed input from the user, or a file of rules.
      output: messages, results and prompts to the user.

description
      (note: in the following examples, do not type the quote marks.)
      the search program allows one to look for simple patterns in a book.
      the patterns can be like 'ggag', that is, with particular bases
      (always written 5' to 3') or it can include unknown 'spacing' bases,
      as in 'ggagnnnnnnnnnatg'.  any base will be allowed in the n positions.
      one can shorten the instruction: 'ggag9natg', and one can make some of
      the spacing 'extentable' as in 'ggag5e4natg' which allows a 5 to 9
      spacing between the two elements.  one can obtain Delila instructions
      for the strings found by turning on printing, setting 'from' and 'to'
      values and searching.  for example: 'd p f -5 t +10 q gga6e3n#atg'
      sets up printing, with from=-5, to=+10.  the search will result in
      instructions for strings centered on the a of the atg (by the # symbol).
      the form '(a/g)ct' means to search for both 'act' and 'gct'.
      you may specify numbers of mismatches, and control how much is printed.
      you can type many commands on one line, separated by spaces.
      you can also search for relations between bases.  currently the
      allowed relations are: identity, non-identity, complementarity and
      non-complementarity.  see delman.use.search or type 'help' while inside
      the program to get more information.

      If one is working with an odd binding site (one with an odd number
      of bases) one should use the # symbol to obtain Delila instructions.
      The complement sequence will continue to number the central base.
          gaa#nttc complemented becomes gaa#nttc

      If one is working with an even binding site (one with an even number
      of bases) one should use the % symbol to obtain Delila instructions.
      The complement sequence will continue to number the following base.
          ga%attc complemented becomes ga#attc

documentation
      delman.use.search

author
```

```
bugs
     there is overlap between the letters used as commands to the program
     and letters used as ambiguous bases.  for instance, h can mean (a/c/t) or
     it can mean 'help'.  the best way to avoid confusion is to
     always start search strings with either a,c,g,t,n or (.
     warning: if you use a file for input, be sure that the rules include
     a quit command and have no errors in them.  it is possible that errors
     will lead to an infinite loop.   (this may be a general problem with
     interactive i/o in pascal on your computer.)

*)
(* end module describe.search *)
version = 5.64; (* of search.p 1993 January 9
```

```
(*
name
      sepa: separates delila instruction sets

synopsis
      sepa(presites: in,  mixture: in,
              sites: out, nonsites: out, output: out)

files
      presites: delila instructions for sites of interest.  they are in
         any order and may contain several references to the same place.
         (let us call this a.)
      mixture: delila instructions for both sites and nonsites, as obtained
         from the search program.
         (let us call this b.)
      sites: the presites are reordered and redundant requests are removed.
         (these reordered instructions we will call a".)
      nonsites: the mixture is reordered, redundant requests and requests in
         the presites instructions are removed.
         (using the previous notation, this would be (b-a)".)
      output: messages to the user.

description
      the separate program has two main purposes:
      1) to eliminate redundancy in both the site and the nonsite sets.
      2) to eliminate the sites from the nonsite set.
      the delila instructions must be in the form output by the search
      program (as in delmods book.iw modules).  once the separation is
      completed, you may obtain the aligned book by using delila.

documentation
      delman.use.data.flow, nar 10(9): 2971 and 2997 1982

see also
      search, delila, alist

author
      thomas d. schneider

bugs
      sepa can not tell that these instructions are identical:
         get from 56 -10 to 56 +10 direction -;
         get from 56 -10 to 56 +10;
      because the second one may not be direction -.  this potential problem
      can be avoided by always giving the direction.

      also, it is advisable to make aligned listings with alist to be sure
      that the new aligned book is correct.

*)
(* end module describe.sepa *)
version = 2.08; (* of sepa.p 1990 Aug 15
```

```
(* begin module describe.shell *)
(*
name
   shell: basic outline for a program

synopsis
   shell(afile: in, output: out)

files
   afile:  multiple line detailed description of file 1, etc
   output: messages to the user

description
   The purpose and use of the program.
   This page is to be copied and edited for making new programs.

examples
   An example of the use of this form is module describe.lister

documentation
   Other sources of information or documents on the program.

see also
   aa.p

author
   Thomas Dana Schneider

bugs
   problems with the program and how to get around them (if known).

technical notes
   Details about the implementation that may be relevant to a user.

*)
(* end module describe.shell *)
version = 1.00; (* of shell.p 1993 January 8
```

```
(* begin module describe.shift *)
(*
name
      shift: copy one file to another file, with a blank in front of each line

synopsis
      shift(fin: in, fout: out, output: out)

files
      fin: the file to be copied with shifting
      fout: the shift of fin
      output: messages to the user

description
      shift makes a copy of the file fin on the file fout, with an extra
      blank line as the first character of each line.  this is useful on
      computer systems with a line printer that uses the first character
      for carriage control.  one can then shift files, such as programs,
      before printing them.

see also
      shift

author
      thomas d. schneider

bugs
      none known

*)
(* end module describe.shift *)
version = 1.03; (* of shift 1985 apr 25
```

```
(* begin module describe.short *)                                     310
(*
name
      short: find locations of short lines in a file

synopsis
      short(fin: in, fout: out, shortp: in, output: out)

files
      fin: the file to be analyzed
      fout: a list of lines that are short.
      shortp: a parameter to determine what 'short' means.
         this is one integer.  lines of this length or shorter
         will be reported to fout.
      output: messages to the user

description
      database programs that scan a line and assume that there are
      a certain number of characters on the line will lose track of
      the correct location if the line is shorter than they expect.
      this has happened with delila and dbcat.  the short program
      scans a file for lines shorter than a given length and
      lists them in the fout file.  the purpose of the program is
      to help debug database programs.


author
      thomas schneider

bugs
      none known
*)
(* end module describe.short *)
version = 1.01; (* of short 1985 may 30
```

```
(* begin module describe.shortline *)                                        311
(*
name
   shortline: make short lines out of long lines

synopsis
   shortline(input: in, output: out)

files
   input:  text to be wrapped
   output: wrapped text

description
   This Pascal program takes ASCII text and filters it.  Lines longer than the
constant maxline are forced to be maxline long by inserting carriage returns.

author
   Thomas Dana Schneider

bugs
   the constant maxline is fixed at compile time, of course.

*)
(* end module describe.shortline *)
version = 1.00; (* of shortline.p 1991 Oct 4
```

```
(*
name
   show: show modules in a module library

synopsis
   show(modlib: in, modcat: inout, print: out, input: intty, output: out)

files
   modlib: a module library as used by program module
   modcat: a module catalogue for modlib, generated by program module or
           show.  it is used (if it is not empty) for faster startup.
   print:  modules that the user pulls out from modlib
   input:  typed instructions from the user
   output: messages to the user

description
   Among other uses, the show program lets you look at pages of the delila
   manual by using the computer.  Each page is a unit we call a 'module'.  The
   name of the module that contains the page you are reading is
   'describe.show'.  Notice that the name has two parts separated by periods.
   The show program takes advantage of this naming convention to let you select
   the section(s) of the manual that you want to see.  Show generates a list of
   the module names.  For delman this is

      1 * version
      2   delman.

   With this list of name-parts one has several choices:  you can choose to
   look at the "version" page by typing "version." or "1" (without quotes).
   The * in the list means that the page will print on the terminal.  To look
   at the list of pages that begin with "delman." you would simply type
   "delman." or "2".  The period in the list means that there are sub-parts to
   the name, such as "delman.intro".

   The names form a tree-like structure that the show program knows about.
   You can climb down the tree by either typing the name or the number given.

   One can type more parts to a name than one.  For example, the command
   "delman.describe.module" would print documentation on the module program.

   Commands are separated by blanks.  Show considers any consecutive
   string of characters (with no blanks) that contains a period to be
   a module name.  Anything without a period is a command, such as "top"
   which gets one to the top of the name tree.

   Once you find a section that you want to step through page by page, you can
   use the n command.  You can also simply hit the carriage return repeatedly.

   Type "help" for a list of other commands and details.

documentation
   moddef
```

```
see also
   module

author
   Thomas D. Schneider and Billie H. Lemmon

bugs
   Some combinations of n and l commands may make the parent on the list
   incorrect.  Go to the top to correct this.

   On Unix systems, the program will ignore the first line you type.  Simply
   hit a carriage return when the program starts.

technical notes
   The names in the module library must be separated by periods for the show
   program to recognize the parts of the names.

*)
(* end module describe.show *)
   version = 3.06; (* of show.p 1989 July 8
```

```
(* begin module describe.shrink *)
(*
name
   shrink: reduce size of postscript graphics

synopsis
   shrink(input: in, output: out)

files
   input:  A PostScript program, containing a translate command.
   shrinkp: Parameter file.  The first line contains the scale factor.
   output: A copy of the input with the scale instructions.  A scale
      command is placed immediately after the translate command, so that
      the shrinking occurs toward the zero of the image.

description
   One often wants to run rsgra to look at a large region of aligned sequence,
   but the normal output won't fit on a page.  By passing the PostScript
   file through this program, one can scale the graphics to something
   that fits on a page.

examples
   0.5  would reduce the size of the image by a factor of 2.
   Note: the 0 is necessary for most Pascal compilers.

documentation

see also
   rsgra.p

author
   Thomas Dana Schneider

bugs
   The program is very specific in what it does.

technical notes

*)
(* end module describe.shrink *)
version = 1.01; (* of shrink.p 1989 November 14
```

```
(*
name
   sites: analyse sites from randomized sequence data base

synopsis
   sites(database: in, standard: in,
         caps: out, latex: out, list: out, sorted: out,
         stats: out, tables: out, rsdata: out, output: out)

files
   database: database consisting of DNA sequence data.
      The first line is the name of the database.
      The remaining lines consist of experimental packages.
      The start of a package is a line like:
          @ -27 11 -21 5 0.85
      The '@' must be left justified as the first character on the
      line.  The numbers are defined to be:

          @ FROM.range TO.range FROM.random TO.random fraction.canonical

      FROM.range: the coordinate of the first base reported in the database
      TO.range:   the coordinate of the last base reported in the database
      FROM.random: the coordinate of the first randomized base
      TO.random:   the coordinate of the last randomized base
      fraction.canonical:  the fraction of the canonical base during
                           chemical synthesis.

      The next line defines the canonical sequence which was 'randomized'.  It
      is in the format of the remaining sequences.  The first sequence in the
      package is always the standard, so do not forget to include it!

      The sequences follow the standard.  The format of the standard and the
      randomized sequences consists of:

      DNA sequence, plasmid name, primer, experiment, date (year, month, day)

      separated by one space each instead of commas.
      The sequence may contain any of the characters: "acgtxd.".
      "x" means that the base is not known.  "d" means that that base
      was deleted.  The program will reject these sequences (to make pure
      data), but this allows them to be stored in the database. "." means
      'the same as the standard sequence in this position'.  This allows
      one to enter sequences as a set of changes from the standard.

      The next experimental package begins with another '@'.  The data from
      each experimental package are gathered as frequencies and normalized by
      using the given canonical base frequency.  The normalized frequencies
      from all the packages are averaged to produce the final results.  This
      allows one to combine several experiments together, however all
      experiments are given the same weight.  This is reasonable if the
      experiments have similar canonical frequencies and numbers of sequences,
      but is probably not correct if one experiment carries more "importance"
      than another.  A method to accounting for these different weightings is
```

not known.

standard: Use the rsdata output of the rseq program from the natural sequences as your standard. It is used for statistical comparison of the experiment to wild-type sequences.

caps: listing of the database sorted and with capital letters showing changes from the standard and database errors.

latex: just like list, but in a form that can be run through the typesetting program LaTeX.

list: listing of the database in an easy-to-read format showing only the changes from the standard. Also gives the tables of numbers of bases.

sorted: the list sorted by sequence

stats: frequency statistics of the database differences. summary of information results.

tables: frequency tables for various stages of the normalization.

rsdata: This simulates the output of the rseq program by giving the numbers of bases (b) at each position (i). When the frequency tables are normalized in this program, the effective number of sequences is lost. To make sure that the numbers reported in rsdata are accurate, they are multiplied by constant scaleup. The table can be run through dalvec and makelogo to make a sequence logo. The variance, varhnb, is set to be negative to indicate that no method is known for how to calculate it. An earlier version of the program gave the minimum error based on the number of sequences in the database, but people tended to miss this fact when looking at the final sequence logo, so were unduely impressed by the data.

output: messages to the user

description

The function of the sites program is to gather, collate and analyze data from a randomization experiment. See the reference given below.

It was designed to help enter sequence data. One may enter several copies of a particular sequence, and they will be joined together by merging their data. Sequences of the same clone are identified by their common plasmid names. Inconsistent data are flagged.

First the program sorts the data and checks that multiple entries are consistant with one another. If they are not, the program halts and you should look into the caps file to figure out what is wrong.

The program converts the database into a more readable form in list, and provides statistical analysis. If the standard is:

```
gaattcaaattaatacgactcactatagggagaaagctt pTS37 kc7 ex100 87 nov 2
```

and one of the data base lines is:

```
gaattcaaattaattcgactcactttagggaaaaagctt pTS331 1204 ex394 87 nov 2
```

```
   the program presents the data in file list as:
..............t.........t......a....... pTS331 1204 ex394 87 nov 2
   which is more readable.  This allows entry as a sequence, but display
   in a form that is easy to understand.

   If two primers are used, and data are found for both, then the
   name becomes 'both'.

   The stats file contains tables of the wild type frequencies and
   the experimental frequencies.

documentation

@article{Schneider1989,
author = "T. D. Schneider
 and G. D. Stormo",
title = "Excess Information at Bacteriophage {T7} Genomic Promoters
Detected by a Random Cloning Technique",
year = "1989",
journal = "Nucl. Acids Res.",
volume = "17",
pages = "659-674"}

see also
   siva.p, dalvec.p, makelogo.p

author
   Tom Schneider

bugs
   For sorting all plasmid initials are ignored, sorting is by the plasmid
   number only.

   A correction for small sample size is not known for the normalized
   experimental data.  Certainly the method given in program Calhnb is not
   right.  Therefore, the program does not report the expected variation.

*)
(* end module describe.sites *)
version = 7.91; (* of sites.p 1993 January 25
```

```
(* begin module describe.siva *)
(*
name
   siva: site information variance

synopsis
   siva(sorted: in, sivap: in, incu: out, curves: out, list: out,
        output: out)

files
   sorted: the output of the sites program that contains a sorted
      list of sites for each experiment performed.
   sivap: parameters to control the program.
      first line: two integers, from and to coordinates over which
         to do the calculations.
      second line: repeats, the number of times to take passes through
          the data removing subsets.  This improves the statistics.
   incu: the xyin input to xyplo, output of this program.  Two columns:
      first column is the number of sites used to find the information
      second column is the amount of information in bits
        The curves loop around along the axis, so they remain connected.
      curves: another xyin file, for graphing the wiggling info curves
        first column is the position across the site
        second column is the information
        The curves loop around along the axis, so they remain connected.
   list: statistical picture of the result.  Two columns:
      first column is the number of sites used to find the information
      second column is the average amount of information (corresponds
         to the second column of incu, but is the average)
      third column is the variance of the information (corresponds
         to what your eye picks out as the thickness of the incu curves)
   output: messages to the user

description
   Siva calculates the variance of the information in a set of randomized sites
   by eliminating each site in turn and keeping track of the increase in the
   information content.  The information content must increase, since with
   fewer samples there must be less variation (this is the small sample bias
   effect).  The program allows one to graph the information content versus the
   number of sites removed (incu).  When this is done repeatedly, with
   different orders of removing the sites, a thick band of curves is created.
   The thickest part of this band shows the greatest possible amount of
   variation that could be in the total set of sequences.

   To be even-handed, the program removes the first sequence, then randomly
   removes the others.  This creates the first curve.  Then the program removes
   the second sequence and randomly removes the others for the second curve.
   If there are n sequences, then n removal curves will be generated.  This is
   one complete repeat of the process.  If you want, you can do this a number
   of times to get better statistics, using the repeat parameter in sivap.

   The largest variation in the information content is surely greater than the
   variation of the information content in all the sets of removals of sites.
```

```
   For several experiments, the statistics are joined into one set.  With
   several experiments, surely the variation of the combined experiments would
   be less than the variations found for the individuals.  So if one experiment
   gives a greater variation, that will increase the variation siva reports in
   list, so the highest value in list is an upper limit on the variation.

documentation
   @article{Schneider1989,
   author = "T. D. Schneider
    and G. D. Stormo",
   title = "Excess Information at Bacteriophage {T7} Genomic Promoters
   Detected by a Random Cloning Technique",
   year = "1989",
   journal = "Nucl. Acids Res.",
   volume = "17",
   pages = "659-674"}

see also
   sites.p

author
   Thomas Dana Schneider

bugs
   none known

*)
(* end module describe.siva *)
version = 1.95; (* of siva.p 1993 January 26
```

```
(* begin module describe.sortbibtex *)
(*
name
   sortbibtex: sort a bibtex database

synopsis
   sortbibtex(fin: in, fout: fout, output: out)

files
   fin: a bibtex database
   fout: bibtex database sorted by the key
   output: messages to the user, including errors in the
      structure of the database and duplicate entries.

description
   Sort a BibTeX database by the citation keys.

examples

documentation

see also
   rembla.p

author
   Thomas Dana Schneider

bugs
   Entries are defined by blank lines.  Use rembla to make sure that
   there are no extra spaces on the ends of lines.

technical notes

*)
(* end module describe.sortbibtex *)
version = 2.12; (* of sortbibtex.p 1990 June 20
```

```
(*
name
      sorth: sort helix list

synopsis
      sorth(hlist: in, shlist: out, list: out, sorthp: in, output: out)

files
      hlist: a list of helixes generated from program helix.
      shlist: a list of helixes, where the longest or strongest helix
         has been chosen from each piece to piece comparison ('set').
      list: progress of the program.
      sorthp: parameters to control the program.
         1. characters on the first line of the file determine the priority
         order for sorting the helixes.  all commands must end with 'a' to
         indicate 'ambiguous'.  the commands are:
            ea - sort on energies (see technical notes)
            la - sort on lengths (see technical notes)
            ela - sort first on energies then on lengths.
            lea - sort first on lengths then on energies.
         2. the second line of the file must contain one integer,
         'top'.  up to 'top' of the strongest helixes will be
         written to shlist. if 'top' = 1, then any set of helixes that are
         ambiguous are not copied to the shlist.  this allows
         one to find the strongest unambiguous helix in each set.
         3. the third line is the minimum length or maximum energy
         of helixes to be sorted.
      output: messages to the user.

description
      the strongest helixes in hlist are sorted and copied to shlist.
      the user can sort on energy, length, energy then length, or
      length then energy.  the user may chose more than one helix
      to be output (eg, the top 10).

see also
      helix

author
      thomas dana schneider

bugs
      none known

technical notes
      when only one variable is sorted on, the order of the other
      variable will not be meaningful because it is determined by
      the way the sort algorithm works.
      the constant 'maxhelix' determines the maximum number of
      helixes that can be sorted.
*)
(* end module describe.sorth *)
version = 2.40; (* of sorth 1985 may 5
```

```
(*
name
   spec: analyse two spectra from the camspec

synopsis
   spec(csdata: in, baseline: in, xyin: out, output: out)

files
   csdata:  contains one spectrum from the Camspec to be used
     as the data.
   baseline:  contains one spectrum from the Camspec to be used
     as the baseline.
   xyin: input to xyplo program
   output: messages to the user

description
   Analysis of spectra produced by the camspec.

   Setup:  Establish communications with the Camspec.

   Give the T command.  If it does
   not respond with '2', use the A command and repeat T until it does.
   This sets the mode to transmittance.

   Then set to 450 nm with           450G
   Check that it is set with         U
   Set to 100% transmittance with    B
   Check that it is 100% with        V
   You can use the O command to do both checks at once.

   Use T and A to set the mode to 0, for reporting in absorbance.
   'AT' should do it.

   Obtain the spectrum with
      xl h
   where x is the interval:
      L 5nm
      M 1nm
      N 0.5nm
   and l is the lower wavelength, h is the high.  Eg,
      L400 700
   scans from 400 to 700 nm with steps of 5nm.

examples

documentation

see also
   xyplo.p

author
   Thomas Dana Schneider
```

```
technical notes
   The spectrum is set to -maxint at both ends so that when multiple
   xyin spectra are concatenated, the return lines all run below the graph
   (where you won't see them with xyplo).  Xyplo will object, but ignore it.

   The camspec sends absorbance data multiplied by 100.  This number is
   in constant 'correctionfactor'.

*)
(* end module describe.spec *)
version = 1.10; (* of spec.p 1992 June 15
```

```
(*
name
   sphere: plot density of shannon spheres

synopsis
   sphere(spherep: in, sigma: out, xyin: out, output:out)

files
   spherep: parameters.
     The first line is the step size interval (0.01 works well).
     the second line is the maximum radius to calculate out to (= maxr,
         3.1 works well).
     Each following line is a dimension to plot.
     If the dimension number is negative, it must be followed on the same line
     by the coordinates of the position to place the dimension numeral.
   sigma: lists the estimates for Rmaximum +/- sigma,
      taken as the radius when the curve passes through exp(-1/2).
   xyin: input to xylop, the plot
   output: messages to the user

description
   Create a graph of radius versus density of Shannon spheres
   at various given dimensions.  The output is run through xyplo.

   The function is:

   pd(R) = R^(D-1) * exp( sqr(R)/ (2* sqr(sigma)))

   where '^' means to exponentiate and
   where sqr(sigma) * (D-1) - sqr(Rmaximum)
   so setting Rmaximum = 1 relates sigma and D.

   The graph is in the range (0,0) to (r=maxr,1)).
   The curve is normalized so that its maximum is at (1,1).
   (except when dimension = 1, where it is at (1,0).

   Since xyplo can't plot several separate curves, without being
   told each symbol, this program simply starts at (0,pd(r)), draws
   the curve to (maxr,pd(maxr)), then circles back by drawing lines
   to the x axis (2*maxr,0) and then the origin (0,0).  By setting
   the region that xyplo plots below maxr, one gets nice, fully
   correct curves that do not appear to be connected.

documentation
   [1988 jan 23,5]

see also
   xyplo

author
   Thomas Dana Schneider

bugs
```

```
   none known
```



```
*)
(* end module describe.sphere *)
version = 1.38; (* of sphere 1989 November 23
```

```
(* begin module describe.split *)
(*
name
      split: split a wide file into printable pages

synopsis
      split(sin: in, sout: out, splitp: in, output: out)

files
      sin: the file to be split into pages
      sout: the split result
      splitp: parameters to control split.  if splitp is empty,
         defaults are used.  otherwise splitp must contain 3 to 5 lines:
         1. if the first character is p (for 'page prompting') then the
            pagination is controlled by the sin.  (this is done by
            duplicating the first several columns to all the horizontal
            pages, as determined by the second parameter.)
            otherwise, pages begin as determined by the second parameter.
         2. for page prompting (see parameter 1) this is the number of
            columns to duplicate from the left margin to all pages.
            if not page prompting, then this is the lines per page in sin.
         3. columns per page in sin (not less than 1).
         4. number of header lines to copy to sout before splitting the rest.
         5. if 4. is negative, this is a trigger inside quotes (").
            -(4.) lines beyond this trigger splitting will begin.
         note: columns and lines per page refer to the input file, sin.
         to find the actual width of the output file pages, add 1 to
         parameter three (when not page prompting) or
         add parameter two to parameter three (when page prompting).
         one extra line is added per page for the page coordinate.
      output: messages to the user.

description
      the split program slices up the sin file into an array of pages,
      each located by an (x,y) coordinate.  in this way a file which is too
      large to print can be printed and then reconstructed.  in otherwords,
      if you have a program which produces output that is wider than the
      printer page (or the screen of the crt, for that matter) then you can
      run your output through split to obtain pages that will print ok.
      the upper lefthand corner of each page tells the coordinate of the page
      as (x down, y across).  a header page shows all the page coordinates.

examples
      if splitp contains: n/60/130/10 (on 4 lines) then sin will be
      split into 60 line by 130 column pages, after 10 header lines.
      if splitp contains: p/1/120/-5/"trigger" then each page will be 120
      characters wide and the first column will be copied to each page.
      the header extends 5 lines beyond and including the trigger.
      for p/5/132 the first 5 columns will be copied to each page.

author
      thomas d. schneider

bugs
```

```
      none known

technical notes
      constant pagecharacter is the (system dependent) begin page character.
*)
(* end module describe.split *)
version = 3.52; (* split 1986 nov 14
```

```
(*
name
   sqz: squeeze the input file to fit into fewer characters per line

synopsis
   sqz(fin: in, fout: out, output: out, sqzp: in);

files
   fin: a text file with lines longer than 80 characters
   fout: the squeezed file.  all lines that end with the endofline symbol
      are to be continued on the next line.
      the endofline symbol is written out as the first character of the
      file, so that the unsqz program can use it.
      if the endofline symbol is found anywhere in the fin file,
      then the fout will be emptied, and the program will halt.
   sqzp: if not empty, then the first character redefines
      the endofline symbol.
   output: messages to the user

describe
   for transportation, this program allows a file to be compressed
   to fewer than 80 characters per line.

see also
   unsqz

author
   thomas dana schneider

bugs
   none known

technical note:
   the default endofline character is defined by a global constant
*)
(* end module describe.sqz *)
version = 1.13; (* of sqz.p 1993 Jan 27
```

```
(*
name
   ssbread: read a sample sheet from the ABI sequencer

synopsis
   ssbread(ssb, report: in, output: out)

files
   ssb: A sample sheet from the ABI Sequencer
      Each sample must have plasmid and primer names in the
      sample name section.
   report:  reading of the sample number, plasmid and primer names
   output: messages to the user

description
   The program allows one to test the reading modules for program tod.

examples

documentation

see also
   tod.p

author
   Thomas Dana Schneider

bugs

technical notes

*)
(* end module describe.ssbread *)
version = 1.11; (* of ssbread.p 1993 January 26
```

```
(* begin module describe.stirling *)
(*
name
   stirling: test of Stirling's formula

synopsis
   stirling(output: out)

files
   output: a table of Stirling's approximation

description

examples

documentation
   Stirling's approximation for factorial is compared to the exact factorial
   function.  The results can be plotted with xyplo.

see also
   xyplo

author
   Thomas Dana Schneider

bugs

technical notes

*)
(* end module describe.stirling *)
version = 1.03; (* of stirling.p 1993 January 27
```

```
(*
name
   sumfile: sum of file sizes

synopsis
   sumfile(input: in, output: out)

files
   input:  the input to this program should be from the Unix command:
       du -s ~/_*
      (where the underscore should be removed; it is there to avoid
       a stupid compiler bug!!)
   output: The output is three columns:
      first column is the first column of the input, the size in kb of
         the various files
      second column is the running sum of the first column
      third column is the same as the second column of the input, the
         names of the files.

description
   The program allows one to find out how many files will fit onto a tape.

examples
   An example of the use of this form is module describe.lister

documentation
   see the man page for du.

author
   Thomas Dana Schneider

bugs
   none known

technical notes
   none.

*)
(* end module describe.sumfile *)
version = 1.00; (* of sumfile 1989 March 31
```

```
(*
name
      tipper: copy a file to the output file with special symbols at end

synopsis
      tipper(fin: in, tipperp: in, output: out)

files
      fin: the file to be copied
      tipperp: this file indicates what string should be copied
         to the output file after the fin file.  if tipperp
         is empty, then tipper uses the string normally recognized by
         the unix tip program to mean end of file.
         (the string is determined by typing ~seofread? to tip)
         otherwise the contents of tipperp are copied to
         output after the fin file.
      output: the copy of fin, followed by either string as
         determined by the tipperp file.  tipper will not give its
         version number unless the fin file is empty.

description
      tipper makes one copy of the file fin on the file output,
      and then appends a special string of characters to the end of
      the file.  these characters are intended to be recognized
      by the tip program under unix to indicate the end of the file.
      this makes transportation of files from a remote system to
      a unix operating system quite easy.

see also
      copy, the tip program under the unix operating system, whatch.

author
      thomas d. schneider

bugs
      tip responds to any of the characters in the special
      string.  tipper does not warn the user if those characters
      appear inside the file.  so, for example, if the source code of
      tipper is transferred, the transfer is broken by tip
      at the point that the special string is detected in the
      code and the source code begins to spill out to the screen
      rather than a file under unix.  to fix this, simply reset the tip
      eofread variable to one which is not in the fin file.
      the whatch program can be used to determine a good character.

technical notes
      the special string is defined by constant 'eofread'.
*)
(* end module describe.tipper *)
version = 1.11; (* of tipper 1985 apr 26
```

```
(* begin module describe.titer *)
(*
name
      titer: analyse titertek optical density data

files
      titer(plates: in, result: out, verbose: out, output: out)

synopsis
      plates: output of the tk program; containing a header line,
         and a series of plates that describe the names of wells,
         their optical densities at various times and od620 data.
      result: a tabulation of the beta-galactosidase values.
      verbose: more detail on how the calculations were done.
      output: messages to the user.

description
      take data from titertek plates and do the analysis:
        get sample names from id plate, find duplicates
        read in volume values
        read in od620 values
        read in od414 values for each time point
                calculate best slope from all time points
                calculate activity for each sample
        report beta-galactosidase data
           st.dev. is the standard deviation for the samples
           % dev.  is 100*st.dev./activity.  If this is larger than 10%
                   for 4 samples, we usually redo the measurement.

examples
        titer.plates  example input file
        titer.result  example output file
        titer.verbose  example output file

see also
        beta-galactosidase assay protocol
        tk program (in advanced IBM basic)
        tkod.p program

author
        Gary Stormo

*)
(* end module describe.titer *)
const   version = 2.29;  (* of titer.p 1993 Jan 27
```

```
(* begin module describe.tkod *)
(*
name
   tkod: read od values from tk data

files
   tkod(input: in, xyin: out, xyplop: out, output: out)

synopsis
   input: output of the tk program; containing a header line,
      and a series of plates that describe the names of wells,
      their optical densities at various times and od620 data.
   xyin: the data from the input rearranged for xyplo
   xyplop: control file for xyplo
   output: messages to the user.

description
   Tkod takes OD620 data from 96 well plates as provided by the
   tk program and converts the data into a form that the xyplo
   program can use to plot with.

see also
   titer program
   beta-galactosidase assay protocol
   tk program (basic)
   xyplo

author
   Tom Schneider

(* end module describe.tkod *)
version = 1.11;  (* of tkod, 1987 august 4 *)
```

```
(*
name
   tod: to database format for sites program

synopsis
   tod(abi: in, thedate: in, ssb: in, todp: in,
       results: out, summary: out, db: out, output: out)

files
   abi: Raw sequences from the ABI sequencing machine.  The
      files called *_??.Seq.txt are manipulated under Unix by:

      more *_??.Seq.txt | cat > abi
      echo "" >> abi

      The more program puts each name followed by the contents, and it is smart
      enough to pipe it to cat which joins the results together.  Thus the abi
      file contains the sample names followed by the sequences.  The echo puts
      a single carriage return at the end of the file so that it ends cleanly.

   thedate:  Date that the sequences were run, output of makedate program.

   ssb:  This must be a copy of the "Sample_Sheet.bin" file from the ABI
      machine.  It contains:
           lane number
           plasmid name
           primer name
      in a funny non-ASCII format which this program extracts from.  (The
      program extracts the data from their known rigid locations.)  The sample
      name column of the sample sheet must contain the plasmid name.  Any
      number of spaces, slashes (/) or null characters are then skipped and the
      next non null word (ending in null or space) is taken as the primer
      name.  Thus the format is "plasmid/primer".  For example:  pTS421/pTS37f1

      There is a bug in the ABI code which will replace the first letter
      of the 24th lane with a null character sometimes.  To get around
      the bug, we will try to rewrite the sample sheet if this appears.

   todp: parameters to control the program
      first line:  a string of characters, called R1, which represents
         a restriction site or other sequence.
         NOTE: it should be self complementary.
      second line:  a string of characters, called R2, which represents
         a restriction site or other sequence.
         NOTE: it should be self complementary.
      following lines: Editing commands for sequences.  There is one editing
 command per line, and each consists of three integers (called N, P1
 and P2) followed by a string (S).  N is the lane number to edit.
 The P1 and P2 define two positions in the sequence.  The sequence
 between these positions is deleted and replaced by the string S.  The
 string must contain only the letters 'acgt' or the single letter 'd'.
 This allows one to insert sequence (make the P1 = P2 + 1, string is
 'acgt' form), to delete (P1 > P2 + 1, string is 'd') and to replace
```

```
 (make P1 = P2 + 1 + length of string in 'acgt' form).  Lines that
 begin with '*' are comments, copied to the results file.  Comments
        for each edit must be placed just below the edit command line.

   results: running commentary of the processing of the sequences.

      The following changes are made:
      1.  Each sequence is edited according to instructions in todp.
      2.  Each sequence is converted to lower case.
      3.  The letter 'n' is converted to 'x'.
      4.  When there is exactly one copy of R1 and one copy of R2,
          the region between R1 and R2 is printed (including R1 and R2).
          Otherwise, the entire original sequence is printed.
      5.  The sequence complement is printed if necessary to assure that R1
          is printed before R2.  The program will print the original
          sequence if R1 and R2 cannot be found on the complement.

      The sequence is then joined to the data from ssb, and the results printed.

   summary: summary of the results.

   db: The sequences from abi are reformed into the database format needed
      by the sites program.

   output: messages to the user

description
   Convert output sequence from ABI sequencing machine into format usable
   by the sites program.

examples

documentation

see also
   sites.p, makedate.p, dotod

author
   Thomas Dana Schneider

bugs

technical notes

*)
(* end module describe.tod *)
version = 2.73; (* of tod.p 1993 January 26
```

```
(* begin module describe.todawg *)
(*
name
      todawg: change a book into dawg format

synopsis
      todawg(book: in, dp: out, output: out)

files
      book: a book from the delila system
      dp: a file of the sequences corresponding to the book, in the
         form needed for the dawg program
      output: messages to the user

description
      the dawg program needs a special format file to create a dawg.
      this program converts a book to that format.

documentation
      obtain from david haussler information about dawgs.

author
      thomas d. schneider

bugs
      none known
*)
(* end module describe.todawg *)
version = 1.05; (* of todawg 1986 nov 14
```

```
(* begin module describe.tstrnd *)
(*
name
      tstrnd: test random generator

synopsis
      tstrnd(output: out)

files
      output: the version of tstrnd is printed.  successful compilation
         and running of the program indicates that the modules are correct.

description
      test of a random number generator

author
      thomas d. schneider

bugs
      none known

technical notes
      the constant n in procedure randomtest determines how many times
      the random number generator will be in a series of tests.  if n
      is small, the the test will be poor, if it is large then the test may
      take a long time.

*)
(* end module describe.tstrnd *)
version = 'tstrnd 1.06 1988 October 17';
```

```
(* begin module describe.undel *)
(*
name
      undel: remove references to delman in modules

synopsis
      undel(fin: in, fout: out, output: out)

files
      fin: a text file containing modules
      fout: a copy of fin with any modules beginning with
            'delman.describe.' replaced by 'describe.'
      output: messages to the user

description
   from this point on, manual pages will be simply describe.name.
   this program removes the old convention.

author
      thomas d. schneider

bugs
      none known
*)
(* end module describe.undel *)
version = 1.15; (* of undel.p 1993 Jan 27
```

```
(*
name
      unixmod: specific module library for the unix operating system

synopsis
      unixmod(output: out)

files
      output: where the date and time will appear.

description
      unixmod contains modules that will replace corresponding modules in
      the other module libraries which are cyber-system dependent. this
      will allow easy transportation of the delila system to unix
      operating systems for the pyramid 90-x.

documentation
      moddef, delman.describe.module

see also
      delman.describe.delmod, moddef, delman.describe.module

see also
 delmods, prgmods, matmods, vaxmods

author
      tom schneider

bugs
      none known

technical notes
      the datetime package required a const 'namelength' and a type 'alpha'.
      these are part of the book.const and book.type modules of delmod, and
      are identical to those types and consts.  note:  programs which use
      the datetime package must have these types and consts either from
      delmod or manually declared.
*)
(* end module describe.unixmod *)
version = 'unixmod 1.13 86 feb 13';
```

```
(* begin module describe.unshi *)
(*
name
 unshi: remove first column of characters from a file

synopsis
 unshi(fin: in, fout: out, output: out)

files
 fin: the file to be unshifted
 fout: the unshifted file
 output: messages to the user

description
 the unshi program reverses the effects of the shift program
 by removing the first character of each line in a file.

see also
 shift

author
 patrick r. roche

bugs
 none known yet silly us

*)
(* end module describe.unshi *)
version = 1.04; (* of unshi 1985 apr 26
```

```
(* begin module describe.unsqz *)
(*
name
   unsqz: unsqueeze the input file

synopsis
   unsqz(fin: in, fout: out, output: out);

files
   fin: the output of the sqz program
   fout: the unsqueezed file.
   output: messages to the user

describe
   unsqz reverses the operation of sqz.
   The first character of the fin file is used to indicate where
   lines should be fused together, so no matter what character was used
   to sqz, unsqz will always work.

see also
   sqz

author
   thomas dana schneider

bugs
   none known

*)
(* end module describe.unsqz *)
version = 1.06; (* of unsqz.p 1993 Jan 27
```

```
(*
name
   untex: remove tex and latex constructs

synopsis
   untex(input: in, output: out)

files
   input: a tex or latex file
   output: the file with:
      '\xxx' command words converted to spaces,
      '{$}' converted to spaces
      free floating '.' ',' '(' ')' removed
      comments (%) removed

      multiple spaces are comressed to single spaces.
      multiple lines are compressed to 2 lines (to preserve the
      paragraph structure).
description
   This reduces the number of words counted by wc to something close to correct.

author
   Thomas D. Schneider

bugs
   citations and comments on lines by themselves leave a blank line.

*)
(* end module describe.untex *)
version = 1.26; (* of untex.p 1991 Mar 19
```

```
(* begin module describe.untitle *)
(*
name
   untitle: remove titles from bbl file

synopsis
   untitle(input: in, output: out)

files
   input: a bbl file from bibtex.
   output: a bbl file without titles.

description
   Titles are removed by deleting between the two copies of the
   '\newblock' strings, leaving the second one.  If the first '\newblock'
   contains the italics indicator, '{\it', then the program realizes
   that this must be a book title, and it keeps the title.

author
   Thomas D. Schneider

*)
(* end module describe.untitle *)
version = 1.17; (* of untitle, 1988 july 2
```

```
(*
name
   unverb: remove verbatim sections from a latex file

synopsis
   unverb(input: in, output: out)

files
   input: a latex file
   output: the file with verbatim sections removed

description
   Removing verbatim sections helps to reduce the number of words counted
   by wc to something approximately correct.  To be used in conjunction
   with untex.

see also
   untex

author
   Thomas D. Schneider

bugs
   none known

*)
(* end module describe.unverb *)
version = 1.17; (* of unverb, 1988 September 14
```

```
(*
name
      vaxmod: specific module library for the vax computer

synopsis
      vaxmod(output: out)

files
      output: where the date and time will appear.

description
      vaxmod contains modules that will replace corresponding modules in
      the other module libraries which are cyber-system dependent. this
      will allow easy transportation of the delila system to vax computers
      running under vms.

documentation
      moddef, delman.describe.module

see also
      delman.describe.delmod, moddef, delman.describe.module

see also
 delmods, prgmods, matmods

author
      patrick r. roche

bugs
      none known

technical notes
      the datetime package required a const 'namelength' and a type 'alpha'.
      these are part of the book.const and book.type modules of delmod, and
      are identical to those types and consts.  note:  programs which use
      the datetime package must have these types and consts either from
      delmod or manually declared.
*)
(* end module describe.vaxmod *)
version = 1.10; (* of vaxmod 1991 Mar 19
```

```
(* begin module describe.ver *)                                    347
(*
name
      ver: look at the version of a program

synopsis
      ver(input: in, output: out)

files
      input: a program source code
      output: the line that contains "version = " in input

description
      this program lets one look at the version number of a program
      source code.

author
      thomas schneider

see also
      verbop

bugs
      none known

*)
(* end module describe.ver *)
version = 2.01; (* of ver.p 1990 Dec 13
```

```
(* begin module describe.verbop *)
(*
name
      verbop: increment the version number of a program

synopsis
      verbop(source: inout, output: out)

files
      source: a program source code, with a version constant in the form
          "version = " followed by a real number.
         the version number is incremented by 0.01.
      output: the new version number is reported.

description
      if you are too lazy to change the version number of a program
      every time you alter the code, then you have no excuses any longer,
      because this program will do it for you automatically...

author
      Thomas Schneider

see also
      ver, code

bugs
      none known

*)
(* end module describe.verbop *)
version = 2.08; (* of verbop.p 1990 june 20
```

```
(* begin module describe.vernum *)
(*
name
      vernum: print the version number of a program

synopsis
      vernum(input: in, output: out)

files
      input: a program source code, with a version constant in the form
          "version = " followed by a real number.
      output: the new version number is reported.
         If there is none, the program reports 0.

description
      the program finds the version number of a file and reports it to
      output for the purpose of saving copies.

author
      thomas schneider

see also
      ver, verbop, code

bugs
      none known

*)
(* end module describe.vernum *)
version = 1.04; (* of vernum 1988 feb 19
```

```
(*
name
   versave: save the file under the version number

synopsis
   versave(input: in, output: out)

files
   input: a text file, with a version constant in the form
       'version = ' followed by a real number.  The name of the
       file (including dot extensions) must be found after the word 'of '.
   output: Four lines are produced:
       file
       (name of text file found after the 'of')
       version
       (the real number found after 'version = ')

description
   Generate commands for worcha on how to change a script for saving
   the file.  A script is then passed through worch to produce the
   executable commands.

example
   For an input file containing:
      version = 1.00; (@ of versave.p 1989 April 4
   The output is:
      file
      versave.p
      version
      1.00
   This is to be placed in the worcha parameter file, worchap.

   An example script is:
      cp file old/file.version
      echo saved file in old/file.version

   Using worcha with the script would become:

      cp versave.p old/versave.p.1.00
      echo saved versave.p in old/versave.p.1.00

   When executed, this will save the text.

author
   thomas schneider

see also
   worcha, verbop, ver, code

bugs
   none known

*)
```

```
version = 1.09; (* of versave.p 1989 May 8
```

```
(* begin module describe.vfilt *)
(*
name
   vfilt: vector filter

synopsis
   vfilt(data: in, fines: out, output: out)

files
   data: the output of the scan program
   vfiltp: paramters to control the program.  one integer, the
      lowest value to pass through the filter
   fines: the same form as data, but low values removed
   output: messages to the user

description
   the program eliminates the lowest values in a scan of a matrix
   against a sequence.

see also
   scan

author
   Thomas Dana Schneider

bugs
   none known

*)
(* end module describe.vfilt *)
version = 1.02; (* of vfilt 1988 jan 6
```

```
(* begin module describe.whatch *)
(*
name
      whatch: what characters are in a file?

synopsis
      whatch(fin: in, fout: out, output: out)

files
      fin: the file to be studied
      fout: an alphabetic list of the characters in the file, giving:
         the character,
         the ordinal number of the character (pascal ord function),
         how many such characters are in the file,
         and the percent of the character in the file.
      output: messages to the user

description
      sometimes it is necessary to determine what characters are in a file.
      if the file is very large, it is not possible to do this by hand.

author
      thomas schneider

bugs
      none known

technical notes
      the constant maxchars determines the number of characters accepted.

*)
(* end module describe.whatch *)
version = 1.11; (* of whatch 1985 apr 20
```

```
(* begin module describe.winfo *)                                                           354
(*
name
   winfo: window information curve

synopsis
   winfo(data: in,  winfop:in, xyin: out, output: out)

files
   data:   output of rseq
   winfop: parameters to control the program
      First line: window size
   xyin:   input to xyplop
   output: messages to the user

description
   Make a sliding window average of an information curve

examples

documentation

see also
   rseq.p xyplo.p

author
   Thomas Dana Schneider

bugs
   not yet!

technical notes
   Constant maxwin is the largest window size allowed.

*)
(* end module describe.winfo *)
version = 1.08; (* of winfo.p 1989 November 28
```

```
(* begin module describe.wl *)
(*
name
   wl: wrap lines in a file

synopsis
   wl(input: in, output: out)

files
   input:  text to be wrapped
   output: wrapped text

description
   This Pascal program takes ASCII text and filters it.  Lines longer than the
constant maxline are altered by inserting carriage returns.

author
   Thomas Dana Schneider

see also
    ww.p

bugs
   the constant maxline is fixed at compile time.

*)
(* end module describe.wl *)
version = 1.00; (* of wl
```

```
(* begin module describe.woco *)
(*
name
   woco: word counting program

synopsis
   woco(input: in, output: out)

files
   input:  a file to find the number of words in
   output: number of words in the file

description
   The program knows about latex constructs a little.
   A word is defined to be any contiguous string of A-Z, a-z, 0-9,
   excluding those that begin with a \.

author
   Thomas Dana Schneider

bugs
   none known

*)
(* end module describe.woco *)
version = 1.08; (* of woco 1988 July 5
```

```
(* begin module describe.worcha *)
(*
name
      worcha: word changing program

synopsis
      worcha(fin: in, fout: out, worchap: in, output: out)

files
      fin:     the file in which words need to be changed to other words.
      fout:    the file where the copy of fin with the words changed is written.
      worchap: the parameter file containing the words that need to
               be replaced and their replacements.  Worchap must be
               constructed as follows:

               a word that needs to be changed is on the first line, the
               following line contains the replacement word,
                  next line:  word to be replaced,
                  following line: replacement word,
               and so on....etc.

               so, the odd numbered lines, (1,3,5....), have the words
               from fin that will be replaced, and the even numbered
               lines, (2,4,6...), contain the replacement words.
      output:  where error messages will appear.

description
      This program was designed to go through a pascal program and
      locate and replace 'words', (pascal identifiers).
         Worcha will sort through a file and look for the words that need to be
      changed, ignoring comments and both single and double quotes.  Upon
      finding the old words, worcha will substitute the specified new words from
      worchap when copying the input file onto the specified output file.  As
      many words as necessary may be changed at one time.  Worcha produces a
      list of the changes within a comment at the end of the fout file.

documentation
      delman.assembly.worcha

author
      Patrick R. Roche

bugs
      The program will yell if word length is equal to wdlgthmax.

technical notes
      Worcha uses linked-lists for storing the words to be changed and their
      replacements.  Thus as many words as desired may be changed at one time.

*)
(* end module describe.worcha *)
version = 2.48; (* of worcha.p 1989 April 5
```

```
(* begin module describe.wordlist *)
(*
name
   wordlist: lists words in a file

synopsis
   wordlist(input: in, output: out)

files
   input:  a file to find the words in
   output: the words of the file listed one per line

description
   The program knows about latex constructs a little.
   A word is defined to be any contiguous string of A-Z, a-z, 0-9,
   excluding those that begin with a \.

author
   Thomas Dana Schneider

bugs
   none known

*)
(* end module describe.wordlist *)
version = 1.13; (* of wordlist 1993 January 26
```

```
(*
name
   ww: word wrap

synopsis
   ww(input: in, output: out)

files
   input:  text to be wrapped
   output: wrapped text

description
   This Pascal program takes ASCII text and filters it.  Lines longer than the
constant maxline are altered by replacing the first space after position
maxline with a carriage return.  This has the effect of wrapping the lines
between 'words'.
   The original purpose was to get around a design flaw in another program.
The program fig produces graphics for X and NeWS windows.  The graphics
is converted to PostScript by another program, f2ps.  Unfortunately f2ps
was poorly designed: the PostScript produced has many lines longer than
70 characters.  When this PostScript code is sent to the (latest as
of 1988) Apple NTX LaserWriterII, the printer dies.  By running this filter,
the problem is bypassed.  Moral: never make lines longer than 80 characters!

author
   Thomas Dana Schneider

bugs
   the constant maxline is fixed at compile time, of course.

*)
(* end module describe.ww *)
version = 1.01; (* of ww 1988 September 14
```

```
(*
name
   xyplo: plot x, y data

synopsis
   xyplo(xyin: in, xyout: output, xyplop: in, output: out)

files
   xyin:  A set of header lines that begin with asterisk ('*') are copied to
       output.  Remaining lines are the data in columns, ending with end of
       file.  Do not use tabs to separate data, as the tabs will be recognized
       as tokins!  Missing columns are not allowed.  See the demonstration file
       xyin.demo for an example.  Once the first data line has been read,
       lines that begin with an '*' will be ignored.  This allows one
       to place comments or other information deeper into the file withou
       having xyplo object.

   xyplop:  Parameters to control the plot, on lines as shown.  The major
       sections of the parameter file are separated by lines that are used by
       the program as separators.  A separator line may begin with blanks, and
       these must be followed by asterisks, as shown below.  These lines simply
       make the file easier to deal with, but you must have them in the file!
       The easiest way to create a xyplop file is to copy the demonstration
       file (xyplop.demo) and modify that to suite your needs.

       xzero yzero         amounts to move the graph origin (inches)
       zx min max          (character, real, real) if zx='x' then set xaxis
       zy min max          (character, real, real) if zy='y' then set yaxis
                           These two lines set the minimum and maximum range of
                           the data to graph.  Other characters mean the
                           program automatically uses the range of the data.
       xinterval yinterval number of intervals on axes to plot
       xwidth    ywidth    width of numbers in characters
       xdecimal  ydecimal  number of decimal places
       xsize     ysize     size of axes in inches
       xlabel              the x axis label
       ylabel              the y axis label
       zc                  if zc='c' then a crosshairs put on zero of x and y
                                 'x' then only X axis is plotted
                                 'X' then only X axis and crosshairs
                                 'y' then only Y axis is plotted
                                 'Y' then only Y axis and crosshairs
                                 'n' then neither axis nor crosshairs
                                 'N' then neither axis with crosshairs
                           Otherwise, both axes are plotted without crosshairs.
       zxl base            if zxl='l' then convert the x axes to a log scale
                           using the indicated base
       zyl base            if zyl='l' then convert the y axes to a log scale
                           using the indicated base
       * define columns to read data from **********************************
                           This section defines which column of xyin contains
                           what kind of data.  You can use a column only once.
```

```
    xcolumn   ycolumn   columns of xyin that determine the
                        location of the symbol
    symbol-column       the xyin column to read symbols from
                        if zero, then use the first symbol defined below
    xscolumn  yscolumn  columns of xyin that determine the size  of the
                        symbol.  If zero, then no data is expected.
```

NOTE: for most symbols this is the entire size of
the symbol. For the I beam symbol, the yscolumn is
half of the total size plotted. Thus one may use
standard deviations and obtain a symbol of 2
standard deviations high centered on the y
coordinate.

```
    hucolumn sacolumn brcolumn       hue saturation brightness columns.
                        These control the color of the rectangle symbol.
                        1 0 0 is black (assumed if columns are all zero)
                        1 0 1 is white
    * define one or more symbols ****************************************

                        Each of these sections defines one of the symbols by
                        specifying what to do for each symbol flag seen in
                        the symbol column.  There may be as many symbols as
                        will fit in memory.
```

The last of these sections must contain just a '.'
as the 'symbol-to-plot'. This is required to end
the symbol definition section since there are an
indefinite number of symbols.

```
    symbol-to-plot      (character) Most symbols are plotted at the
```

coordinates given in xcolumn and ycolumn.

```
                        'c' plot a circle
                        'b' plot a box
                        'x' plot an x
                        '+' plot a plus
                        'I' plot an I beam symbol
                        'd' plot a box with central dot
                        'p' point (or dot) alone.
```

'R' plot a filled rectangle in color. Unlike the
other symbols, which are centered on the data, the
lower right hand corner of this rectangle is placed
on the data. This allows the user more control on
placement.

```
                        'r' like 'R' but gray scale.  The brightness
                        column is used for controling the brightness.
```

'f' Means to plot the symbol-flag (defined below).
The 'f' type allows several symbols to be made each
with its own regression and connection lines, but
plotted with the entire flag string in xyin. The

symbols are distinguished by their first character.

The symbol-flag in xyplop should be set to the string that one desires to be recognized.

'g' Means 'grab bag'. The 'g' type has lower priority than any other symbol. Xyplo searches through all the available symbols looking for a match to the symbol-flag. If a symbol-flag cannot be found, then the data are assigned to the 'grab-bag'. The program uses the symbol-flag on the graph.

The symbol-flag in xyplop can be anything.

The symbol underscore (_) in xyin is converted to a blank to allow the appearance of separated words.

One can do grab-bag connected curves without symbols by setting g and the symbol-flag to ' '. One can also set the symbol-to-plot to blank (or other unrecognized symbol) to get specific connected curves. In this case, the symbols MUST be connected or the program will object (invisible symbol and invisible connection means data loss).

symbol-flag The string of characters that indicates that this symbol should be plotted. Eg, if the 'symbol-to-plot' is I and the flag is x, then whenever an x is seen in the symbol column, an I beam will be plotted. The flag can be more than one character long, but (unfortunately) it cannot contain blanks.

symbol-sizex Side in inches on the x axis of the symbol. If this value is negative, the data in xscolumn is used to determine the size. For circles, sizex determines the radius, sizey is ignored.

symbol-sizey Side in inches on the y axis of the symbol. If this value is negative, the data in yscolumn is used to determine the size. For circles, sizeX determines the radius but a positive number is still required for sizey.

connection linetype size If the first character is 'c' then the symbols will be connected by lines of linetype as defined below. (Linetype must follow the c immediately, without blanks.)

linetype size linetype is a character defining the kind of regression line to plot for this symbol:
'l' means do regression line
'i' invisible,
'.' dotted

```
                        '-' dashed
                        'n' means no line.

   '-' and '.' require a size in inches for the
   spacing.  The others also require a number, but it
   is ignored.

       * end the symbol definitions with a period (left justified!) *********
.
       * define zero or more user defined lines *****************************
       linetype m b size   One or more lines to be drawn on the plot, m and b
   are slope and intercept.  Linetype and size are
   define as for the symbol connection lines.

   xyout: regression results, ready for PostScript input.  (See technical
          notes.)

   output: messages to the user
```

description

The data in the xyin file are converted to graphics in the PostScript language on the xyout file, under control of the parameters set in xyplop. There are several distinct sections of the parameters:

1. The first set of parameters determine the overall characteristics of the graph.
2. The second set of parameters defines the columns of xyin to be read.
3. The next section of the parameter file defines one or more symbols to be plotted on the graph. If desired, a linear regression is performed between the data columns, and this may be graphed for each symbol. The invisible option allows one to obtain the regression data without the graph.
4. A section with just a period ends the symbols section.
5. The last section contains lines you define.

Recommended procedure for using xyplo: obtain a copy of xyplop.demo and xyin.demo, set permission to read them for yourself (on a Unix system use chmod), and copy them to the names xyplop and xyin. Try them out as is. If you don't get a graph, doing your own data will not do any good! Then convert the xyplop to your own use by changing the xyplop.demo file and substitute your xyin file. This way the complexity of xyplop can be held at bay.

see also

xyplop.demo, xyin.demo,
xyplop.test, xyin.test,
xyplop.mul, xyin.mul,
doodle

author

Thomas Schneider

technical notes

The program originally generated output in the pic format. One could then

```
   run this through pic and troff to produce a graph.  However, the program has
   been modified to eliminate the pic notation (by substituting modules from
   dops rather than domods).  All lines outside the graphics now are preceeded
   by a %, which is beginning of a comment in PostScript.  Thus the output of
   the program can be run directly into a PostScript interpreter.  This saves
   on both memory and speed of graphing since the intermediate file is no
   longer created.

bugs
   Minor unobvious things have prevented people from getting graphs.  Most
   problems occur when badly formed xyplop files are used, and the program has
   no way to tell what the difficulty is.  Recently, more checks have been put
   it, so the program can detect most oddly formed xylop and xyin files.  Check
   your xyplop carefully.

*)
(* end module describe.xyplo *)
version = 7.75; (* of xyplo.p 1992 December 4
```

```
(*
name
   zipf: Monte Carlo simulation for Peter Shenkin's problem

synopsis
   zipf(zipfp: in, data: out, xyin: out, output: out)

files
   zipfp:  parameters to control the program
      first line: integer, number of correlation coefficients to create
      second line: integer, number of symbols for each correlation coefficient.
         eg, 20 means amino acids.
      third line: character.  't' means use Tom's method, 'p' means use Peter's.
      fourth line: character.  'g' means to graph the simplex.
   data:  a list of correlation coefficients.  This is to be input
      to the genhis program.
   xyin:  data for graphing the simplex.  The graph is generated with the
      xyplo program.
   output: messages to the user

description

   1992 Jan 13  Returned call to Stephen Altschul 496-2475.  He suggested that
   Peter Shenkin's results of rank versus log of probability are due to random
   effects.  This is easy to test with a Monte Carlo simulation:

   Tom's method
      chose s (eg 20) random numbers
      find their sum
      divide each number by the sum to produce s random numbers which
         sum to 1.
      sort the numbers
      take the log versus the rank
      determine the correlation coefficient
      repeat to get distribution of correlation coefficients.

   Peter's method
      chose s-1 random numbers between 0 and 1
      sort the numbers
      take the differences to produce 20 numbers that sum to 1
      resort the numbers
      take the log versus the rank
      determine the correlation coefficient
      repeat to get distribution of correlation coefficients.

   Graph of simplex.  The numbers all add to 1 for either method.  They are
   points in an s dimensional space.  The volume they fit into is a hyper plane
   of s-1 dimensions since they sum to 1, called a simplex.  The distribution
   of the points can be visualized by projecting onto a plane and graphing with
   the xyplo program.  The projection is done by using polar coordinates.
   There is a vector P from the center of the simplex to each point to graph.
   There is a vector, A, from the center of the simplex to the point where the
   first coordinate has value 1 and all others are zero.  The magnitude of P is
```

determined, and the angle between P and A determines an angle. These numbers are in polar coordinates. They are converted to rectangular coordinates in the xyin file. If s = 3, then the simplex is a simple plane reaching between the three points A=(1,0,0), B=(0,1,0) and C=(0,0,1). The projection takes this equilateral triangle onto the xy plane. In higher dimensions, the points are collapsed to the xy plane, so high dimensional effects are expected. This means that the center should tend to become empty, and the distribution will become spherical. 366

examples

```
zipfp file:
***********************************************************
10000 10000      1000 Number of correlation coefficients to print out
3 16             Number of symbols being simulated
p              t= tom's, else peter's
g              g = graph the symplex, otherwise not

zipfp:  parameters to control the zipf program.
***********************************************************

genhisp file for use with genhis
***********************************************************
x n 50
r -1 -0.5
***********************************************************

xyplop file for use with xyplo
***********************************************************
2 2       zerox zeroy         graph coordinate center
x -1 1 zx 0 25    zx min max (character, real, real) if zx='x' then set xaxis
y -1 1 zy 0 250   zy min max (character, real, real) if zy='y' then set yaxis
10 10     xinterval yinterval number of intervals on axes to plot
6 6       xwidth    ywidth    width of numbers in characters
1 1       xdecimal  ydecimal  number of decimal places
5 5       xsize     ysize     size of axes in inches
x
y
c         zc                  'c' crosshairs, axXyYnN
n 2       zxl base            if zxl='l' then make x axis log to the given base
n 2       zyl base            if zyl='l' then make y axis log to the given base
          *****************************************************************
1 2       xcolumn   ycolumn   columns of xyin that determine plot location
0         symbol column       the xyin column to read symbols from
0  0      xscolumn  yscolumn  columns of xyin that determine the symbol size
0 0 0     hue saturation brightness   columns for color manipulation
          *****************************************************************
p         symbol-to-plot      c(circle)bd(dotted box)x+Ifgpr(rectangle)
0         symbol-flag         character in xyin that indicates that this symbol
0.05      symbol sizex        side in inches on the x axis of the symbol.
0.05      symbol sizey        as for the x axis, get size from yscolumn
nl 0.05   no connection (example for connection is c- 0.05 for dashed 0.05 inch)
n  0.05   linetype  size      linetype l.-in and size of dashes or dots
          *****************************************************************
```

```
          ***********************************************************************
************************************************************

documentation

see also
   genhis.p, xyplo.p

author
   Thomas Dana Schneider

bugs

technical notes
   The non-standard random number generator is used (rand).
   This could be replaced by a portable one, but with the danger
   of it not giving good results.

*)
(* end module describe.zipf *)
version = 1.32; (* of zipf.p 1993 January 26
```

```
This is a list of the Delila programs as of
Wed Jan 27 20:42:01 EST 1993

program name<:> a one-line description of the program.
alist: aligned listing of a book
alpro: frequency and information of aligned protein sequences
alword: frequency and information of aligned words
aran: aligned random sequences
asciicode: converts ascii table to Pascal code
auxmod: modules for auxiliary programs
av: average integers
biglet: text enlargement program
binhex: convert binary to hex
binomial: produce the binomial probabilities for a found black to white ratio
binplo: produce the binomial probabilities for a found black to white ratio
bkdb: convert a book to database format for the sites program
calc: a calculator that propagates errors
calhnb: calculate e(hnb), var(hnb), ae(hnb), avar(hnb), e(n)
calico: character and line counts of a file
cap: put capital letters inside quotes of a program
catal: cataloguer of delila libraries, the catalogue program
censor: removes code from a program
cerf: complement of the error function
chacha: changes characters in a file
chi: estimates chi squared from degrees of freedom
cisq: circle to square
ckhelix: check that the helix location is where one wants
cluster: cluster indana subindexes into groups of duplicate entries
coda: composition file to data for genhis
code: find the comment density of a pascal program
column: pull defined column from input
comp: determine the composition of a book.
compan: composition analysis.
concat: concatenate files together
copy: copy one file to another file
count: counts the amount of sequence in a book
cybmod: specific module library for the cyber computer
da3d: diana da file to 3d graphics
dalvec: converts Rseq rsdata file to symvec format
dbbk: database to delila book conversion program
dbcat: database catalog production and sorting program.
dbfilter: filter GenBank databases to remove unwanted entries
dbinst: extract Delila instructions from a GenBank database
dblo: look at the catalogue of a genbank/embl database
dbpull: database extraction program.
decat: break a file into 10 files
decom: remove comment starts from within a comment
delila: the librarian for sequence manipulation
delmod: delila module library
diana: dinucleotide analysis of an aligned book
difint: differences between integers
digrab: diagonal grabs of diana data
dirty: calculate probabilities for dirty DNA synthesis
```

module: module replacement program
mstrip: remove control m's from a file
nocom: remove comments
normal: generate normally distributed random numbers
notex: remove tex and latex constructs
nulldate: modules to neutralize the date-time functions
number: add line numbers to a file
odti: munch od and time plates together for xyplo
palinf: find palindromes, based on information theory
parse: breaks a book into its components
patana: pattern analysis
patlrn: pattern learning
patlst: lister of patlrn output.
patser: pattern searcher
patval: pattern evaluations of aligned sequences
pbreak: breaks a file into pages at a certain trigger phrase
pcs: partial chi squared
pemowe: peptide molecular weights
prgmod: programming modules for the delila system
rawbk: make a raw sequence into a book
ref2bib: refer to bibtex converter
refer: print the references in the pieces of a book
reform: raw sequences reformatted
rembla: remove blanks from ends of lines in a file
rep: records repeats between sequences in two books
repro: make multiple copies of a file
rf: calculate Rfrequency
ri: Rindividual is calculated for every site in the aligned book
riden: ring density graph
ring: z space ring
rndseq: generate random dna sequences
rseq: rsequence calculated from encoded sequences
rsgra: rsequence graph
rsim: Rsequence simulation
same: counts the number of lines that are identical in two files
scan: scan a book with a wmatrix and generate a vector
search: search a book for strings
sepa: separates delila instruction sets
shell: basic outline for a program
shift: copy one file to another file, with a blank in front of each line
short: find locations of short lines in a file
shortline: make short lines out of long lines
show: show modules in a module library
shrink: reduce size of postscript graphics
sites: analyse sites from randomized sequence data base
siva: site information variance
sortbibtex: sort a bibtex database
sorth: sort helix list
spec: analyse two spectra from the camspec
sphere: plot density of shannon spheres
split: split a wide file into printable pages
sqz: squeeze the input file to fit into fewer characters per line
ssbread: read a sample sheet from the ABI sequencer
stirling: test of Stirling's formula

tipper: copy a file to the output file with special symbols at end
titer: analyse titertek optical density data
tkod: read od values from tk data
tod: to database format for sites program
todawg: change a book into dawg format
tstrnd: test random generator
undel: remove references to delman in modules
unixmod: specific module library for the unix operating system
unshi: remove first column of characters from a file
unsqz: unsqueeze the input file
untex: remove tex and latex constructs
untitle: remove titles from bbl file
unverb: remove verbatim sections from a latex file
vaxmod: specific module library for the vax computer
ver: look at the version of a program
verbop: increment the version number of a program
vernum: print the version number of a program
versave: save the file under the version number
vfilt: vector filter
whatch: what characters are in a file?
winfo: window information curve
wl: wrap lines in a file
woco: word counting program
worcha: word changing program
wordlist: lists words in a file
ww: word wrap
xyplo: plot x, y data
zipf: Monte Carlo simulation for Peter Shenkin's problem
(* end module describe.program-list *)

index

index

index

index

index