# CMU PDP-10
# INTRODUCTORY USERS MANUAL

Editors: Jack Dills
Art Farley
Mary Shaw

JANUARY 1973

Computer Science Department
Carnegie-Mellon University
Pittsburgh, Pennsylvania

## PREFACE

The following manual is intended to provide a usable introduction to computing on the CMU PDP-10. To accomplish this, a discussion of general computing procedures and the PDP-10 monitor is given, followed by descriptions of the available language systems. The manual does not provide full language descriptions (references are provided to necessary, useful language manuals); but through a short introduction, sample problems to try, and an annotated script, the manual hopes to impart to the user an introductory knowledge of what it is like, and what to expect, when using each of the discussed language systems on the CMU PDP-10.

Note that timely information can be found for many of the language systems in a printable text file <language>.DOC on the PDP-10. Information on which files are available can be found in DOC.DOC. To get a copy of a DOC file print SYS:<language>.DOC.

Pat Fahey
Sept. 1971

TABLE OF CONTENTS



NOTE: The FORTRAN and BASIC language systems are fully described in the PDP-10 Timesharing Handbook. Thus, no discussion of either is included.

GENERAL PROCEDURES

D. Bajzek, B. Anderson, H. Wactlar

USAGE NUMBERS

A computer usage application may be obtained from the Manager of Operations, Science Hall 3204, and should be returned there when completed. You will be notified by campus mail, probably within a week, as to your usage number. It will contain eight alphanumeric characters. The first four characters are your account number; this is used for departmental accounting and statistics. The last four characters are your man number; i.e. the initials of your first and last names with two digits appended. Your man number will be the first part of your dectape name(s), and is sufficient to identify you in most cases.

DECTAPES

DECtapes may be purchased in the CMU bookstore. Members of the Computer Science Department may borrow DECtapes for their personal use free of charge from the Manager of Operations, Science Hall 3204. For the benefit of students in the Immigration Course and other graduate courses in the Department, one DECtape will be assigned to each student in the course and will be filed in the machine room for your use during the duration of the course. If additional tapes are needed, see the Manager of Operations. Each DECtape is named with from five to seven alphanumeric characters. The first four characters will be your man number, with from one to three characters of your choice appended. This is the name which you will use when requesting that a DECtape be mounted.

TROUBLE REPORT FORMS

Hardware trouble report forms are located in the teletype room, Science Hall 5201. These are to be filled out when you encounter hardward trouble with Datels and teletypes. The yellow copy should be put in a conspicuous place on the terminal, and the white copy should be put in the container marked REPAIR REQUESTED.

Software bugs can be reported by running the cusp GRIPE described under the heading UTILITY PROGRAMS.

If a hardware or software problem is seriously impeding your work and should receive immediate attention, call the operator on extension 350. He will report the problem to the appropriate staff member.

TELETYPES

We currently have two dial-in lines for teletypes on the PDP-10/A: 687-3411 and 687-3412.

A knob is located on the right front panel of the teletype, with three positions: LINE, OFF, and LOCAL. Line indicates that the teletype is on-line to the computer; that is, typed characters are sent to the computer for interpretation and response by the system. Local indicates that the teletype is being used off-line from the system, as a typewriter.

To change teletype paper, insert the red spindle through the center of the roll of paper, and place the spindle in the appropriate grooves in the teletype, making sure that the paper unrolls from beneath the roll. Unroll a foot or so of paper, and tear the paper unevenly so that a corner protrudes when inserted into the carriage. Lift the view plate and pull forward the rubber tipped lever at the right of the platen (black roller). Tilt the metal paper bale toward you and insert the paper under the platen. Pull the paper through toward you, tilt the paper bale back again, and insert the paper under the metal paper holder. Align the paper and then push the rubber tipped lever back. Lower the view plate and pull the paper through over the paper bale. Tear along the edge of the view plate. The knob on the left of the tty will advance the paper manually.

To change the ribbon, examine another tty to see how the ribbon is inserted. Be sure to keep one of the spools which is already on the tty, as the ribbon replacement has only one spool.

CONTROL CHARACTERS

On the Teletype, there is a special key marked CTRL called the Control Key. If this key is held down and a character key is depressed, the Teletype types what is known as a control character rather than the character printed on the key. In this way, more characters can be used than there are keys on the keyboard. Most of the control characters do not print on the Teletype, but cause special functions to occur, as described in the following sections.

There are several other special keys that are recognized by the system. The system constantly monitors the typed characters and, most of the time, sends the characters to the program being executed. The important characters not passed to the program are also explained in the following sections.

Control - C

Control - C (↑C) interrupts the program that is currently running and takes you back to the monitor. The monitor responds to a control - C by typing a period on your Teletype, and you may then type another monitor command. For example, suppose you are running a program in BASIC, and you now decide you want to leave BASIC and run a program in AID. When BASIC requests input from your Teletype by typing an asterisk, type control - C to terminate BASIC and return to the monitor. You may now issue a command to the monitor to initialize AID (.R AID). If the program is not requesting input from your Teletype (i.e., the program is in the middle of execution) when you type control - C, the program is not stopped immediately. In this case, type control - C twice in a row to stop the execution of the program and return control to the monitor. If you wish to continue at the same place that the program was interrupted, type the monitor command CONTINUE. As an example, suppose you want the computer to add a million numbers and to print the square root of the sum. Since you are charged by the amount of processing time your program uses, you want to make sure your program does not take an unreasonable amount of processing time to run. Therefore, after the computer has begun execution of your program, type control - C twice to interrupt your program. You are now communicating witb the monitor and may issue the monitor command TIME to find out how long your program has been running. If you wish to continue your program, type CONTINUE and the computer begins where it was interrupted.

The RETURN Key

This key causes two operations to be performed: (1) a carriage-return and (2) an automatic line-feed. This means that the typing element returns to the beginning of the line (carriage-return) and that the paper is advanced one line (line-feed). Commands to the monitor are terminated by depressing this key.

The RUBOUT Key

The RUBOUT key permits correction of typing errors. Depressing this key once causes the last character typed to be deleted. Depressing the key n times causes the last n characters typed to be deleted. RUBOUT does not delete characters beyond the previous carriage-return, line-feed, or alt-mode. Nor does RUBOUT function if the program has already processed the character you wish to delete.

The monitor types the deleted characters, delimited by backslashes. For example, if you were typing CREATE and go as far as CRAT, you can correct the error by typing two RUBOUTS and then the correct letters. The typeout would be

```
CRAT\TA\EATE
```

Notice that you typed only two RUBOUTS, but \TA\ was printed. This shows the deleted characters, but in reverse order.

Control - U

Control - U (↑U) is used if you have completely mistyped the current line and wish to start over again. Once you type a carriage-return, the command is read by the computer, and line-editing features can no longer be used on that line. Control - U causes the deletion of the entire line, back to the last carriage-return, line-feed, or altmode. The system responds with a carriage-return, line-feed so you may start again.

The ALTMODE Key

The ALTMODE key, which is labeled ALTMODE, ESC, or PREFIX, is used as a command terminator for several programs, including TECO and LINED. Since the ALTMODE is a non-printing character, the Teletype prints an ALTMODE as a dollar sign ($).

Control - O

Control - O (↑O) tells the computer to suppress Teletype output. For example, if you issue a command to type out a 100 lines of text and then decide that you do not want the type-out, type control - O to stop the output. Another command may then be typed as if the typeout had terminated normally.

Control - Z

This is the end-of-file character when the input device is the teletype, similar to and end-of-file mark on a magtape.

Modifying the terminal characteristics

When you login to the system the teletype characteristics are defaulted to the appropriate set for that terminal. If you wish to modify them, there is a TTY command which declares special properites of the Teletype line to the scanner service. The command format is:

TTY dev: NO WORD

dev:= the device argument that is used to control a line other than the one where the command is typed. This argument is optional and is legal only from the operator's console. It may be used to modify the characteristics of any Teletype lines in the system.

NO = the argument that determines whether a bit is to be set or cleared. this argument is optional.

WORD = the various words representing bits that may be modified by this command. The words are as follows:

| | |
|---|---|
| TTY TAB | This terminal has hardware TAB stops set every eight columns. |
| TTY NO TAB | The monitor simulates TAB output from programs by sending the necessary number of SPACE characters. |
| TTY FORM | This terminal has hardware FORM (PAGE) and VT (vertical tab) characters. |
| TTY NO FORM | The monitor sends eight linefeeds for a FORM and four linefeeds for a VT. |
| TTY LC | The translation of lower-case characters input to upper case is suppressed. |
| TTY NO LC | The monitor translates lower-case characters to upper case as they are received. In either case, the echo sent back matches the case of the characters being sent. |
| TTY WIDTH n | The carriage width (the point at which a free carriage return is inserted) is set to n. The range of n is 17 (two TAB stops) to 200 decimal. |
| TTY NO CRLF | The carriage return normally outputted at the end of a line exceeding the carriage width is suppressed. |
| TTY CRLF | Restores the carriage return. |
| TTY NO ECHO | The Teletype line has local copy and the computer should not echo characters typed in. |
| TTY ECHO | Restores the normal echoing of each character typed in. |
| TTY FILL n | The filler class n is assigned to this terminal. The filler character is always DEL (RUBOUT, 377 octal). No fillers are supplied for image mode output. Teletypes are class 0, 30 character per second terminals use classes 1 and 2, and datels are class 3 fillers. |
| TTY NO FILL | Equivalent to TTY FILL 0. |

GETTING TAPES MOUNTED

The first thing to do is to get a unit assigned for your tape.

Type: .AS DTA (FOR DECTAPE) or

.AS MTA (FOR MAGTAPE)

The monitor will respond with:

DTA2 ASSIGNED

or, if no unit is available, it will respond:

NO SUCH DEVICE

After a unit is assigned to you, you will notify the operator to mount your tape by using the monitor command PLEASE. PLEASE is described under the heading UTILITY PROGRAMS. In your request specify the tape name, tape unit, and whether the tape should be enabled for writing. If you do not specify "write enabled," the operator will write lock the tape. Remain in PLEASE mode until the operator responds to your request. He may say

NNØØABC MOUNTED ON DTA2 ENABLED

or, since the monitor recognized eight DECtape units and eight magtape units, and we have only five DECtape drives and two magtape drives, there may not be a drive free for you even though you have a unit assigned. If this is the case, the operator will try to get a drive for you as soon as possible. The drives are allotted on a first-come-first-served basis. If you need a drive urgently or only for a minute, the operator can try contacting other users to see if someone can give up a drive. When a drive is free the operator will mount your tape and notify you.

The tape drives are very much in demand, so please be considerate of others. When you finish with a tape, be sure to tell the operator to dismount it immediately, thus freeing the drive for someone else. If you are logging off, the unit will be returned to the pool. If not, you can type

.DEAS DTA2

to make the unit available for others. If you are using the same unit number for more than one tape, be sure to reassign the unit between tapes

.AS DTA2

and so a fresh copy of the directory will be read into core and you will not be using the directory from the last tape.

A unit can be reassigned to another job without first being returned to the pool by typing

.REAS DTA2 n

where n is the job number.

LINE PRINTER OUTPUT

The line printer (LPT) is currently located at the far end of the machine room, behind the operator's console. The operator bursts output as soon as it comes off of the printer if possible; however, if he is busy mounting tapes it may take a few minutes. Output is filed alphabetically by man number just inside the door to 3103. This door will be left unlocked for users to retrieve their output from 0800-2400. It will be locked from 0000-0800.

UTILITY PROGRAMS

Two monitor commands, PLEASE and SEND, may be used for inter-console communication, including communication between your teletype and the CTY (console teletype).

PLEASE is a monitor command which puts the issuing terminal, and eventually the CTY, into a special communications mode. This mode is evoked by typing, when logged in and in monitor mode,

.PLEASE text <cr>

If the CTY is logged in, or running SYSTAT, or in another PLEASE, the message

OPERATOR BUSY, PLEASE HANG ON

will print on the teletype. You can terminate the PLEASE with a CONTROL C or wait until the CTY is free. When it is free your teletype will print

OPERATOR HAS BEEN NOTIFIED

and your message will print on the CTY along with identifying information about you and several "bells." Now both terminals are in PLEASE mode. Any line typed on either terminal, terminated by <cr> will print out on the other terminal and will otherwise be ignored by the system. Thus a two-way communication is established. This mode is terminated with a CONTROL C or an ALTMODE typed on either terminal. Both terminals will then be in monitor mode. The most frequent use of PLEASE is to request mounting of tapes, or to talk with the operator via teletype.

SEND provides a mechanism for one-way inter-console communication. One line of text is transmitted to another terminal, TTYn, by typing

.SEND TTYn    text    <cr>

SEND leaves the user in monitor mode. The format of the message on the receiving terminal is

TTYm:    text  <cr>

where m is the terminal where the message originated. If the sender or receiver of the message is the CTY, the message will be transmitted regardless of what the receiving terminal is doing. The message will print out, leaving the terminal in its former state. If CTY is not involved, a busy test is made to see if the receiving terminal is in monitor mode. If so, the message is transmitted; if the designated terminal is not in monitor mode, the sender will get the message

?BUSY

on his terminal. You can do a short SYSTAT

.SY S

to determine which terminals are in use by whom and what they are running.

Another monitor command, SYSTAT, will give you current running information about the system. To get all the information printed on your tty, type

.SY

Subsets of the STSTAT information are available by running variations of SYSTAT. To get a short version of SYSTAT, giving the current status of all users on the system, type

.SY S

To determine the status of a particular job, type

.SY n

where n is the job number. To find out which I/O devices are assigned to which users, type

.SY B

To list all jobs waiting in the line printer queue, type

.SY Q

Also

.SY H

lists all the SYSTAT commands, including those given above.

Two CUSPs (commonly used system programs), MAIL and GRIPE, may be used to write a message onto a file in another's disk area. MAIL will create or update a file called MAIL.BOX on another user's disk area.

To send mail to a user type

.R MAIL

The CUSP will respond

ENTER PPN:

After the colon, type the user number (all eight characters) of the user to whom you are sending mail, and the <cr>. MAIL responds

ENTER A MESSAGE TERMINATED WITH AN ALTMODE:

Type your message, followed by <cr> and ALTMODE. There is no need to identify yourself as this information will be recorded in the file. Your terminal will then be returned to monitor mode. When the user next logs onto the system, the message

MAIL PENDING

will print on his tty at the beginning of the logon message. He can read the message by listing his file MAIL.BOX; i.e.

• R PIP

*TTY:<-MAIL.BOX

GRIPE will create a file for your message on one of the system disk areas. If you have a comment or gripe about the hardware, software, operations, etc. of the system, you can run the GRIPE CUSP.

.R GRIPE

GRIPE will respond with

YES? (TYPE ALTMODE WHEN THROUGH)

Type your comments as instructed; that is, first type <cr>, then your message, another <cr> and ALTMODE. There is no need to identify yourself, as that information will be recorded along with your comments in the GRIPE file. Systems personnel regularly review the GRIPE files and an answer will be sent to you by campus mail if appropriate.

PRINT is another useful CUSP. PRINT can be used to print files on the line printer. Unlike printing with PIP, PRINT supplies the filename on the file header page, and enables the user to print several copies of the file if desired. To run the CUSP, type

```
.R PRINT
```

When PRINT prompts you with a *, type the names of the files to be printed separated by commas. If you want the file to be deleted after being printed, type /D after the filename; if you want several copies, type /n after the filename where n is a number from 2 through 9 indicating the number of copies wanted. An example follows:

```
.R PRINT
*FOO.LST/D,*.MAC/2,FOO.F4
```

Now FOO.LST will print on the line printer and then that disk file will be deleted. Two copies of all files with MAC extensions will be printed and FOO.F4 will be printed. If the files to be printed are on a device other than DSK, you must precede each filename with the device name on which it is located; i.e.,

```
.R PRINT
*DTA2:FILE1,DTA2:FILE2,FILE3
```

Now files FILE1 and FILE2 from DTA2 and FILE3 from DSK will print.

Another useful CUSP is SAVE. SAVE will save on magtape, or restore from magtape, all or selected disk files for a single user. For instructions on how to run SAVE, type

```
.R SAVE
*/H
```

The instructions will print on the TTY; or type

```
.R SAVE
*/2L
*/H
```

to get the typeout on the line printer.

LEARNING TO TYPE

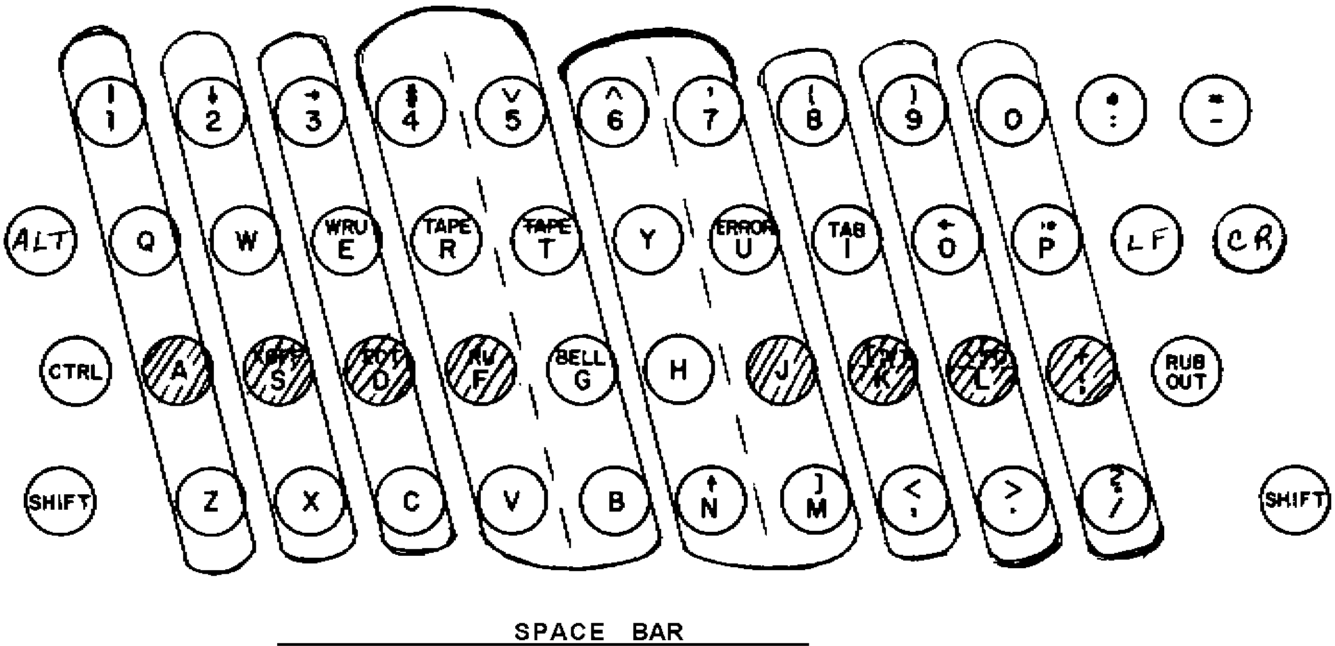
You will probably be spending many hours at the teletype. It will greatly increase your efficiency if you learn to type properly at the beginning. Following are a few brief instructions to get you started.

Study the keyboard chart below. Find the left-hand home keys on it; the left-hand home keys are "a - s - d - f Now find them on your teletype keyboard. Place your finders on them. Study the chart again. Find the right-hand home keys on it. Find them on your teletype keyboard. Place your finders on them. Take your fingers off the keys. Replace them. Repeat two or three times. Get the feel of these home keys. Curve your fingers. Hold them lightly just above the home keys. Drop your wrists slightly, but do not let them rest on the frame of the teletype. Strike the space bar with a quick inward motion of your right thumb.

Type the line of home keys shown below. Say each letter as you strike it. Repeat several times.

ff dd ss aa jj kk ll ;; ff dd ss aa jj kk ll ;; fj

TELETYPE KEYBOARD




SHrfbEb KEYS rttfE HOME hChYS

Carriage return is operated with the little finger of your right hand. Type each line twice. Double space after the second line.

ff jj dd kk ss ll aa ;; fj dk sl a; fdsa jkl; fjsl

a lad; a fall; a lad; a fall; a lad; a fall; a lad

all lads; all lads fall; a lad falls; a lad falls;

Regardless of what key you are typing, the other fingers should always remain just above their home keys. Operate h with the j finger; g with the f finger.

jhj fgf jhj fgf jhj fgf jhj fgf jhj fgf jhj fgf fj gf

Study the chart again. The a finger also operates the q and z keys. Similarly, each finger operates the keys in a line with its home key. Practice the exercises below.

aqaz aqaz swsx swsx dedc dedc frfv frfv gtgb gtgb

hyhn hyhn jujm jujm kik, kik, lol. lol. ;p;/ ;p;/

The six sets of exercises below will give you more practice in learning where the keys are. Do not go on to the next set until you are fairly sure of the current one.

fdsa jkl; fdsa jkl; gf hj gf hj fall hall glad had

juj juj uj uj full jug dull dud lugs hug hugs gulf

ded ded ed ed led fled he held she shed fell shell

lol lol ol ol old sold fold do so gold log loss go

keg jug she shall fog half log; he had a dull duel

- - - - - - - -

fdsa jkl; uj ed uj ed full fled dull fell jug held

frf kik rf ik rf ik fur fir furl fire ride hire or

lol dcd ol cd ol cd so sod sold cod code ice slice

jnj jnj nj nj fin fund and lend land gain sun sung

a large jug; and hold; did shake; and can fill all

sws sws ws ws will will with loss low how show who
jmj jmj mj mj mad made mar make am same me come me
ftf ftf tf tf to told the then them their lot late
karl saw the gold mine shaft. lou called. jouran

- - - - - - - -

fvf fvf vf vf five live strive move love have give
k,k k,k ,k ,k work, rack, trick, to give, for all,
jyj jyj yj yj yet yell year sly they lay flay gray
ws nj ws nj win wing wink drink won now know knows
they just like to drive down fog street in my car.

- - - - - - - -

;p; ;p; p; p; pled pledge help plain gulp tip trip
fbf fbf bf bf bug but bluff bring rub rib rob bold
aza aza za za zone size maze zones zeal doze dozed
yj vf yj vf live five yet they sly move love stray
jess dent gave buz a small pay check for his work.

- - - - - - - -

aqa aqa qa qa quit quip square squid squash squint
sxs sxs xs xs xs six fix hoax mix flax box tax box
p; bf p; bf pled bring trip blot gulp bold rip rub
gay quick foxes run and jump with bold vim or zip.

USING DATEL TERMINALS ON THE PDP-10

There are currently four dial-in lines for Datels on the PDP-10/a: 683-8330 to 683-8333. The procedure for getting onto the system on a Datel is:

1. Dial.

2. Place receiver on coupler, making sure the ON switch is lit.

3. Switch to remote.

The PDP-10 monitor has been modified to handle Datel terminals with the ASCII type head. Almost every character on the Datel keyboard has a direct ASCII equivalent in the PDP-10. However, some characters do need explanation. See the table below.

The ATTENTION key has two different functions depending on whether the keyboard is locked. If it is locked, ATTENTION unlocks the keyboard but does not result in any character being input. If the keyboard is unlocked, ATTENTION may be used to send an end-of-message; i.e., to release the keyboard control without inputting a carriage return.

The PDP-10 monitor can handle both lower and upper case characters from a Datel, and these terminals are initialized to have both cases. TTY commands can enable or disable this feature; that is, lower case characters will be mapped into upper case if the proper command is used. These commands are:

TTY LC (tells the monitor that the keyboard has a lower case keyboard so lower case letters are not mapped into upper case)

TTY NO LC (no lower case keyboard, therefore, mapping is necessary)

Remember that TTY LC is the initial state of the Datel when logging in.

Typing a CONTROL Q on the Datel puts the terminal into the non-standard APL mode, in which no characters can be input to the Datel. Exit from APL mode is by hitting four successive ATTENTIONs.

Monitor assumes that tabs are set to 8 print positions. If tabs are set to more than 8, early printing may occur.

CHARACTER TABLE

| PDP-10 INPUT | TYPE ON DATEL | PDP-10 OUTPUT ON DATEL |
|---|---|---|
| CONTROL C** | \|C | ,C |
| LINE FEED | INDEX | NONE |
| ALTMODE | $ (also ¬) | $ |
| ↑ | \|\| | \| |
| ] | \|) | ) |
| [ | \|( | ( |
| \ | \|/ | / |
| $ | \|$ | $ |
| ← | _ (underline) | _ |
| ' (grave) | ¢ | ¢ |

** similarly for all control characters

PDP-10 MONITOR

H. Wactlar

Commonly used monitor commands:

ASSIGN <physical device> <logical name>

allocates an I/O device (dectape, magtape) to the user's job and optionally assigns a logical name designated by the user to that device

```
e.g.,   .ASS  DTA3 IN
        DTA3  ASSIGNED
```

ATTACH <job no.>[project programmer No.]<password>
detaches the current job, if any, and connects the console to a detached job. Exclude <password> if attaching to a job detached during logout.

COMPIL <list of source file names separated by commas>

produces relocatable binary files for the specified program(s) by calling the appropriate compiler as determined by the source file name extension (ALG for ALGOL, MAC for MACRO, F4 for FORTRAN BLI for BLISS, SAI for SAIL)

e.g., .COMP TEST.MAC

CONT starts the program at the saved program counter address stored by a ↑C (halt) command

CREATE <filename>

calls the line editor to create a new file

e.g., .CREATE TEST.MAC

DDT saves the program counter and starts the program at the dynamic debugging module optionally loaded with the compiled program

DEASSIGN <logical or physical device name>

returns the I/O device to the system's available pool

e.g., .DEASS IN

DEBUG <list of file names separated by commas>
performs the compile and loading functions and in addition loads DDT which it enters on completion of loading
e.g., .DEBUG TEST.MAC, TEST2.F4

DELETE <list of file names or groups separated by commas>
automatically runs PIP to delete the specified files
e.g., .DELETE TEST.MAC,*.REL

DETACH Disconnects the console from the users job without affecting its status. Console is now free to control another job.

DIRECT <logical or physical device name>:
runs PIP to list the names and space occupied by files on that device (DSK is assumed if no device name given)
e.g., .DI DTA3:

EDIT <file name>
calls the live editor to edit an already existing file
e.g., .EDIT TEST.MAC

EXECUTE <list of file names separated by commas>
performs the compiling and loading functions and initiates program execution
e.g., .EXEC TEST.MAC

KJOB initiates log-off sequence

e. g.

```
.KJ
CONFIRM:  H

IN RESPONSE TO CONFIRM:,TYPE ONE OF: DFHIKLPQSU
D TO DELETE ALL FILES
(ASKS ARE YOU SURE?, TYPE Y OR CR)
F TO TRY TO LOGOUT FAST BY LEAVING ALL FILES ON DSK
H TO TYPE THIS TEXT
I TO INDIVIDUALLY DETERMINE WHAT TO DO WITH ALL EXCEPT TEMP FILES
  WHERE TEMP IS .LST, .CRF, .TMP, .TEM, .RPG
  AFTER EACH FILE NAME IS TYPED OUT, TYPE ONE OF: EKPQS
  E TO SKIP TO NEXT FILE STRUCTURE AND SAVE THIS FILE IF
    BELOW LOGGED OUT QUOTA ON THIS FILE STRUCTURE
  K TO DELETE THE FILE
  P TO PRESERVE THE FILE
  Q TO REPORT IF STILL OVER LOGGED OUT QUOTA, THEN REPEAT FILE
  S TO SAVE THE FILE WITH PRESENT PROTECTION
K TO DELETE ALL UNPRESERVED FILES
L TO LIST ALL FILES
P TO PRESERVE ALL EXCEPT TEMP FILES
Q TO REPORT IF OVER LOGGED OUT QUOTA
S TO SAVE ALL EXCEPT TEMP FILES
U SAME AS I BUT AUTOMATICALLY PRESERVE FILES ALREADY PRESERVED

IF A LETTER IS FOLLOWED BY A SPACE AND A LIST OF FILE STRUCTURES
  ONLY THOSE SPECIFIED WILL BE AFFECTED BY THE COMMAND.  ALSO
  CONFIRM WILL BE TYPED AGAIN.

A FILE IS PRESERVED IF ITS ACCESS CODE IS GE 1ØØ

CONFIRM:
```

LOAD <list of file names separated by commas>

perform the compiling and loading functions to execute core image of runnable program

LOG initiates log-in sequence; prompts for password

Passwords may be modified during login by typing altmode (ESC) after the password instead of a carriage-return. Prompting for the new password will follow.

PJOB — types job number and project programmer number of job running on terminal on which this command is typed

R — <CUSP name>

executes the named commonly used system program

e.g., .R PIP

RENAME — <new file name> = <old file name>

runs PIP to change a file name

```
e.g.,   RENAME   TEST1 .MAC = TEST.MAC
        FILES RENAMED:
        TEST.MAC
```

RUN — <file name>

runs the core image previously loaded and SAVE'd with that file name

e.g., RUN DSK:TEST.SAV

SAVE — <file name>

Copies the core image currently loaded in core onto the specified file so that it can be RUN at a later time

```
e.g.,    .LOAD  TEST.MAC
         .SAVE  TEST
```

SYS — runs a CUSP to provide system status information

e.g.

```
.SYS H
SYSTAT INSTRUCTIONS:
TYPE "SYS<C.RET.>" TO LIST THE ENTIRE STATUS, OR
TYPE "SYS " FOLLOWED BY ONE OR MORE LETTERS AS FOLLOWS—
"<STRING>"
        <STRING> IS AN ACCOUNT NO. ,MAN NO.,STRUCTURE,DEVICE,CUSP
        THIS OUTPUTS THE SYSTEM STATUS OF <STRING>
B  BUSY DEVICE STATUS
D  DORMANT SEGMENT STATUE
F  FILE STRUCTURE STATUS
H  THIS MESSAGE
```

J JOB STATUS
L OUTPUT TO LPT
N NON-JOB STATUS
P DISK PERFORMANCE
Q PRINT QUEUE
S SHORT JOB STATUS

TYPE "SYS" FOLLOWED BY A JOB NUMBER FOR THAT JOB'S STATUS

TIME <job no>

causes typeout of total runtime since last TIME command, total runtime since login, and integrated product of runtime and core size

TYPE <file name>

runs PIP to type on the terminal the specified file

e.g., TYPE TEST.MAC

Note:

PIP and the two editing systems TECO and SOS are discussed separately as language systems in this manual.

## Extended Command Forms

The commands previously explained are adequate for the compilation and execution of a single program or a small group of programs at one time. However, the assembly of large groups of programs, such as the FORTRAN library or the Timesharing Monitor, is more easily accomplished by one or more of the extended command forms.

Indirect Commands(@ Construction) - When there are many program names and switches, they can be put into a file; therefore, they do not have to be typed in for each compilation. This is accomplished by the use of the @ file construction, which may be combined with any COMPIL-class commands.

The @ file may appear at any point after the first word in the command. In this construction, the word file must be a filename, which may have an extension and project-programmer numbers. If the extension is omitted, a search is made for the command file with a null extension and then for a command file with the extension .CMD. The information in the command file specified is then put into the command string to replace the characters @ file.

MONITOR

For example, if the file FLIST contains the string

```
FILEB,FILEC/LIST,FILED
```

then the command

```
.COMPILE FILEA,FILEB,FILEC/LIST,FILED,FILEZ
```

could be replaced by

```
.COMPILE FILEA,@FLIST,FILEZ
```

Command files may contain the @ file construction to a depth of nine levels. If this indirection process results in files pointing in a loop, the maximum depth is rapidly exceeded and an error message is produced.

The following rules apply in the handling of format characters in a command file.

a. Spaces are used to delimit words but are otherwise ignored. Similarly, the characters TAB,VTAB, and FORM are treated like spaces.

b. To allow long command strings, command terminators (CARRIAGE RETURN, LINE FEED, ALTMODE) are ignored if the first nonblank character after a sequence of command terminators is a comma. Otherwise, they are treated either as commas by the COMPILE, LOAD, EXECUTE, and DEBUG commands or as command terminators by all other COMPIL-class commands.

c. Blank lines are completely ignored because strings of returns and line-feeds are considered together.

d. Comments may be included in command files by preceding the comment with a semicolon. All text from the semicolon to the line-feed is ignored.

e. If command files are sequenced, the sequence numbers are ignored.

The + Construction† - A single relocatable binary file may be produced from a collection of input source files by the "+" construction. For example: a user may wish to compile the parameter file, S.MAC, the switch file, FT50S.MAC, and the file that is the body of the program, COMCON.MAC. This is specified by the following command:

```
.COMPILE S+FT5ØS+COMCCN
```

The name of the last input file in the string is given to any output (.REL, .CRF, and/or .LST) files. The source files in the "+" construction may each contain device and extension information and project-programmer numbers.

The = Construction† - Usually the filename of the relocatable binary file is the same as that of the source file, with the extension specifying the difference. This can be changed by the "=" construction, which allows a filename other than the source filename to be given to the associated output files. For example: if a binary file is desired with the name BINARY.REL from a source program with the name SOURCE.MAC, the following command is used.

```
.COMPILE BINARY=SOURCE
```

This technique may be used to specify an output name to a file produced by use of the "+" construction. To give the name WHOLE.REL to the binary file produced by PART1.MAC and PART2.MAC, the following is typed.

```
.COMPILE WHOLE=PART1+PART 2
```

†Used in COMPILE, LOAD, EXECUTE, and DEBUG commands only.

Although the most common use of the "=" construction is to change the filename of the output files, this technique may be used to change any of the other default conditions. The default condition for processor output is DSK: source.REL[self]. For example: if the output is desired on DTA3 with the filename FILEX, the following command may be used:

```
EXECUTE DTA3:FILEX=FILE1.F4
```

The < > Construction[†] - The < > construction causes the programs within the angle brackets to be assambled with the same parameter file. If a + is used, it must appear before the < > construction. For example: to assemble the files LPTSER.MAC, PTPSER.MAC, and PTRSER.MAC, each with the parameter file S.MAC, the user may type

```
.COMPILE S+LPTSER, S+PTPSER, S+PTRSER
```

With the angle brackets, however, the command becomes

```
.COMPILE S+<LPTSER,PTPSER,PTRSER>
```

The user cannot type

```
.COMPILE <LPTSER,PTPSER,PTRSER>+S
```

Compile Switches[†]

The COMPILE, LOAD, EXECUTE, and DEBUG commands may be modified by a variety of switches. Each switch is preceded by a slash and is terminated by any non-alphanumeric character, usually a space or a comma. An abbreviation may be used if it uniquely identifies a particular switch.

These switches may be either temporary or permanent. A temporary switch is appended to the end of the filename, without an intervening space, and applies only to that file.

Example:

```
.COMPILE A,B/MACRO,C      (The MACRO assembler applies only
to file B.)
```

---

[†]Used in COMPILE, LOAD, EXECUTE, and DEBUG commands only.

A permanent switch is set off from filenames by spaces, commas or any combination of the two. It applies to all the following files unless modified by a subsequent switch.

Example:

```
.COMPILE /MACRO A,B,C
.COMPILE A /MACRO B,C
.COMPILE A,/MACRO,B,C
.COMPILE A,/MACRO B,C
```

Compilation Listings† - Listing files may be generated by switches. The listings may be of the ordinary or the cross-reference type. The operation of the switch produces a disk file with the extension.LST ,queues it, prints it, and then deletes it.

The compile-switches LIST and NOLIST cause listing and nonlisting of programs and may be used as temporary or permanent switches.

Listings of all three programs are generated by

```
.COMPILE /LIST A,B,C
```

A listing only of program A is generated by

```
.COMPILE A/LIST,B,C
```

Listings of programs A and C are generated by

```
.COMPILE /LIST A,B/NOLIST,C
```

The compile-switch CREF is like LIST, except that a cross-reference listing is generated (FILE.CRF), processed later by the CREF CUSP which generates the .LST file, queues, prints and deletes it. Unless the /LIST or /CREF is specified, no listing file is generated.

Since the LIST, NO LIST, and CREF switches are commonly used, the switches L,N, and C are defined with the corresponding meanings, although there are (for instance) other switches beginning with the letter L. Thus, the command

```
.COMPILE /L A
```

†Used in COMPILE, LOAD, EXECUTE, and DEBUG commands only.

produces a listing file A.LST (and A.REL).

Standard Processor - The standard processor is used to compile or assemble programs that do not have the extensions .MAC, .CBL, .F4, or .REL. A variety of switches set the standard processor. If all source files are kept with the appropriate extensions, this subject can be disregarded. If the command

```
.COMPILE A
```

is executed and there is a file named A. (that is, with a blank extension), then A. will be translated to A.REL by the standard processor. Similarly, if the command

```
.COMPILE FILE.NEW
```

is executed, the extension .NEW, although meaningful to the user, does not specify a language; therefore, the standard processor is used. The user must be able to control the setting of the standard processor which is FORTRAN IV at the beginning of each command string.

Forced Compilation - Compilation (or assembly) occurs if the source file is at least as recent as the relocatable binary file. The creation time for files is kept to the nearest minute. Therefore, it is possible for an unnecessary compilation to occur. If the binary is newer than the source, the translation does not usually have to be performed.

There are cases, however, where such extra translation may be desirable (e.g., when a listing of the assembly is desired). To force such an assembly, the switch COMPILE is provided, in temporary and permanent form. For example:

```
.COMPILE /CREF/COMPILE A,B,C
```

will create cross-reference listing files A.CRF, B.CRF, and C.CRF, although current .REL files may exist. The binary files will also be recreated.

ALGOL

T. Teitelbaum, L. Snyder, J. Dills
(Revised Jan. 1973)

Algol 60 is an algebraic programming language developed by an international committee in 1960. Algol was designed at a time when many computer installations had their own _ad hoc_ algebraic programming languages. Algol was intended to be a machine independent standard for the communication (and execution) of algorithms. Most of the arbitrary restrictions found in languages such as FORTRAN were eliminated. Algol was the first language for which a complete and precise syntactic and semantic definition was attempted. The terminology used in this definition (in the Algol Report) has come into wide use in computer science. Algol is characterized by dynamic array allocation, recursive procedures, block structure, and a generalized parameter passing mechanism.

REFERENCES

_Manual_

[1] Digital Equipment Corp. _PDP-10 Algol Manual_.

_Definition_

[2] Naur, P. (ed.) Revised Report on the Algorithmic Language ALGOL 60. Comm.ACM 6 (Jan 63).

[3] Knuth, D. E. The remaining trouble spots in ALGOL 60. Comm.ACM 10 (Oct 67).

[4] Abrahams, P. W. A final solution to the dangling _else_ of ALGOL 60 and related languages. Comm.ACM 9 (Sept 66).

[5] Knuth, D. E., Merner, J. N. ALGOL 60 Confidential. CACM, Vol. 4, 1961.

_Philosophy_

[6] Perlis, A. J. The synthesis of algorithmic systems. J.ACM 14 (Jan 1967).

History-Bibliography

[7] Bemer, R. W. — A politico-social history of ALGOL. Annual Review In Automatic Programming, 5 Pergamon Press, 1969.

[8] Sammet, Jean — Programming Languages: History and Fundamentals, Prentice-Hall, 1969.

Introductory

[9] Bottenbruch, H. — Structure and use of ALGOL 60. J.ACM 9 (Apr 62).

[10] Higman, B. — What everybody should know about ALGOL. Computer Journal 6 (1963) p. 50.

[11] Ekman, T. and Froberg, C. — Introduction to ALGOL programming. Oxford University Press, (1967).

[12] Dijkstra, E. W. — A Primer of ALGOL 60 Programming, Academic Press, London, 1962.

Implementation

[13] Evans, A. — An ALGOL 60 Compiler. Annual Review in Automatic Programming, 4. Pergamon Press (1964).

[14] Randell, B. and Russell, L. J. — ALGOL 60 Implementation. Academic Press, (1964), 418 pp.

[15] Dijkstra, E. W. — "Making a Translator for ALGOL 60," Annual Review of Automatic Programming, Vol. 3., MacMillan, 1963, pp. 347-356.

Extensions

[16] Wirth, N. — A Generalization of ALGOL. Comm.ACM 6 (Sept 63).

[17] Perlis, A. J. and Iturriaga, R. — An extension to ALGOL for manipulating formulae. Comm.ACM 7 (Feb 64).

[18] Wirth, N. and Weber, H. — EULER: A generalization of ALGOL and its formal definition. Comm.ACM 9 (Jan, Feb 66).

[19] Wirth, N. and Hoare, C. A. R. — A contribution to the development of ALGOL. Comm.ACM 9 (June 66).

[20] Dahl, O. J. and Nygaard, K. — SIMULA - An ALGOL-based simulation language. Comm.ACM 9 (Sept 66).

[21] Hoare, C. A. R. — Record Handling, in F. Genays (Ed.) Programming Languages, Academic Press, 1968, pp. 291-347.

SAMPLE PROBLEMS

1. Continued Fractions

Let $Q_1 = \frac{1}{1+1}$, $Q_2 = \cfrac{1}{1+\cfrac{1}{1+1}}$, $Q_3 = \cfrac{1}{1+\cfrac{1}{1+\cfrac{1}{1+1}}}$ etc.

As $i \to \infty$, $Q_i \to Q = 0.61803...$

Write an ALGOL 60 function procedure Phi (n) that will return the value $Q_n$. For example, Phi (2) = 0.6666. Write two versions of Phi, one <u>recursive</u> and the other <u>iterative</u>.

2. <u>Palindromes</u>

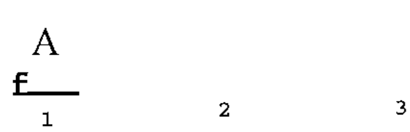
A palindrome is a vector V of values such that V = XY where X = reversal of Y. E.g., 110011.

Write a Boolean function that determines if a vector is a palindrome.

Write another which determines if a vector consists of a list of palindromes; e.g., 110110.

3. <u>Tower of Hanoi</u>

Write an ALGOL program to print the solution sequence to the towers of Hanoi puzzle. Given,




Move the stack of disks on pin 1 to pin 2 (possibly using pin 3 as intermediate storage) so that (1) the disks finally end up in the same order as they started (as shown); (2) at no time is a large disk on top of a smaller disk; and (3) only one disk at a time is moved. Your program should allow an arbitrary number of disks.

4. Partitions

Write an ALGOL procedure PART(X) which prints the partitions of the integer X. A partition is defined as a sequence of positive integers which sum to X. If that's too easy, find the unique partitions of X.

5. Pascal's Triangle

Recall that Pascal's triangle begins:

```
        1
       1 1
      1 2  1
     1 3  3  1
    1  4  6  4  1
```

Write an ALGOL procedure, PASCAL(N) , which prints the Nth row of Pascal's triangle. It should be possible to compute the result without a factorial routine and with only a single vector for a data structure.

6. Pattern of Primes

Write a program which fills an N x N array A with the integers 1 through $N^2$ arranged in a spiral.

E.g., when N = 3, then A =

```
7 8 9
6 1 2
5 4 3
```

The pattern of primes in this arrangement (for large N) has been of some interest (to some people). Try a printout where primes are and non-primes blank.

Can you think of a more efficient storage arrangement for the pattern of primes when N is large?

7. How well do you understand call-by-name and call-by-value?

```
BEGIN REAL A,B;
  REAL PROCEDURE INCV(X);VALUE X;REAL X;
        BEGIN X←X+1; INCV←X END;
  REAL PROCEDURE INCN(X);REAL X;
        BEGIN X←X+1; INCN←X END;
  REAL PROCEDURE ADDV(Y);VALUE Y;REAL Y;
        ADDV←Y+Y;
  REAL PROCEDURE ADDN(Y);REAL Y;
        ADDN←Y+Y;

  A←1; B←ADDV(INCV(A));
  COMMENT A IS NOW ------, B IS NOW -----;

  A←1; B←ADDV(INCN(A));
  COMMENT A IS NOW ------, B IS NOW -----;

  A←1; B←ADDN(INCV(A));
  COMMENT A IS NOW ------, B IS NOW -----;

  A←1; B←ADDN(INCN(A));
  COMMENT A IS NOW ------, B IS NOW -----;
END;
```

8. Exchange

Write a procedure EXCH(A,B) that exchanges A and B. This is not as easy as it seems. Consider the problems exchanging I and A[I].

Answers for odd number problems follow the Algol Script.

*MGOL* **SCRIPT**

**Delete** ^ . ^ ^ ^ ^ T .

*SOS,* e. 5os ^ ^

```
•CKEA1E  F IB*ALG
00100    BEGIN
00200    INTEGER  PROCEDURE  FIBONACCI(N)JVALUE  NI INTEGER  NI
00300       BEGIN  IF  N<=1  THEN  FIBONACCI:=1
00400               ELSE  FIBONACCI:=FIBONACCI(N-1>+FIBONACCICN-2);
00500       ENDI
00600    READ(K);
00700    J:=FIBONACCICK>;
00600    PRINK  J,6)i
00900    END
01000
*E

EAIf


•R  ALGOLY
*FIB>TTY:-FIB.


DECS ICS I EM  10  ALGOL-60>  V.  2B<146>:
1b-JAN-73         14:24:07


00100    BEGIN
00200    INTEGER  PROCEDURE  FIBONACCI <N > i VALUE  N J.INTEGER  N*
00300       BEGIN  IF  N<=1  THEN  FIBONACCI:=1
00400               ELSEFIBONACCI:=FIBONACCI(N-1)+FIBONACCI(N-2);
00i>00      END;
00600    KEAD(K)I
*  4c  4c  4t  4c  4c  *  t
600  UNDECLARED  IDENTIFIER)—
REL  FILE  DELETED                  > T f
00700    J: =F I BONACC I ( K ) *
*******     t
700  UNDECLARED  IDENTIFIER)^
00800    PRINK J#6>J
00900    END



?2  ERRORS

*tC
```

*•Ct'll IV4S»V lii–td on Try -Ilm*

*Con+sol* C

```
.EDIT
*I150
00150   INTEGER J,K;
*E

EXIT
```

(.EDIT) No need to name the file because it is already in core.

```
.EX FIB
ALGOL: FIB
LOADING

LOADER 1K CORE
EXECUTION
10
      89
END OF EXECUTION - 2K CORE

EXECUTION TIME: 0.17 SECS.

ELAPSED TIME:    11.88 SECS.
```

(.EX FIB) CCL command — causes compiling, loading and execution of program.

(10) I enter a 10

(89) Program returns 89, the 10th Fibonacci number.

The following procedure is a pseudo random number generator. If you wish to know more about this type of algorithm see KNUTH, D.E. Vol. 2, Seminumerical Algorithms, The Art of Computer Programming.

```
.CREATE RAND.ALG
00100   INTEGER PROCEDURE RAND(LESS);INTEGER LESS;
00200   COMMENT THIS PROCEDURE RETURNS AN INTEGER BETWEEN 0 AND
00300       LESS-1;
00400   BEGIN OWN INTEGER SEED,MUL,MOD;
00500   IF SEED=0 THEN
00600           BEGIN
00700           COMMENT SEED,MUL&MOD MUST BE LESS THAN 185364
00800           TO PREVENT OVERFLOW.  THE FOLLOWING
00900           NUMBERS ARE 7↑6,5↑7,2↑17 RESPECTIVELY;
01000           SEED←117649;
01100           MUL←78125;
01200           MOD←131072;
01300           END;
01400   SEED←(SEED*MUL) REM MOD;
01500   RAND←(LESS*SEED) DIV MOD;
01600   END
01700   $
*E
```

(OWN INTEGER ... / IF SEED=0 THEN / BEGIN) In this Algol compiler, Own variables are initialized to 0. Thus this block is performed only the first time the procedure is called.

(SEED←(SEED*MUL) REM MOD;) This line is really the random number generator.

This program will read numbers and print random numbers less then the numbers just read until a 0 is read. The random numbers are generated by the above procedure which is called externally.

```
.CREATE TESTR.ALG
00100   BEGIN INTEGER I,R;
00200   EXTERNAL INTEGER PROCEDURE RAND;
00300   READ(I);
00400   WHILE I#0 DO
00500           BEGIN
00600           R:=RAND(I);
00700           PRINT(R,3);
00800           NEWLINE;
00900           BREAKOUTPUT;
01000           READ(I);
01100           END;
01200   END
01300   $
*E
```

(NEWLINE;) Newline causes a carriage return and a line feed to be placed in the output buffer.

(BREAKOUTPUT;) Breakoutput causes the output buffer to be dumped to the output device, in this case the TTY.

```
.EX TESTR,RAND
LOADING

LOADER 1K CORE
EXECUTION
100
  26
100
   4
100
  93
100
   2
0


END OF EXECUTION - 2K CORE

EXECUTION TIME: 0.03 SECS.

ELAPSED TIME:   42.78 SECS.

.EDIT TESTR.ALG
*A100
00100   BEGIN INTEGER L,I,J,R;
*I250
00250   WRITE("RANGE:");BREAKOUTPUT;
*I325,25
00325   WRITE("NUMBER:");BREAKOUTPUT;
00350   READ(L);
00375   $
*R400
00400   FOR J:=1 UNTIL L DO
00425   $
*D800
*D1000
*E

EXIT

.EX TESTR,RAND
ALGOL: TESTR
LOADING

LOADER 1K CORE
EXECUTION
RANGE:100
NUMBER:50
  26   4  93   2  84  27  55  30   3  86  92  27  75  60  21  96  43  75
   7  90  89  12  66  13  62  96  64  17  50   8  15   6  94  75  87  32
  84  74  93  99  61  35   1  60  16  35  25  19  89   2

END OF EXECUTION - 2K CORE

EXECUTION TIME: 0.35 SECS.

ELAPSED TIME:   32.10 SECS.
```

If you don't understand the A command use a R command.

The program is altered so that it will produce an arbitrary number of random numbers, all in the same range.

Here I delete NEWLINE, note the effect below.

I enter the range and number.

```
.EDIT TESTR.ALG
*I360,10
00360    OUTPUT(4,"DSK");
00370    SELECTOUTPUT(4);
00380    OPENFILE(4,"RAND.DAT");
00390    $
*E

EXIT
```

*Assigns the disk to channel 4.*

*Now, the program is altered so that the output will go to a disk file called RAND.DAT.*

*The number 4 in the above statements is the channel number.*

```
.EX TESTR,RAND
ALGOL: TESTR
LOADING

LOADER 1K CORE
EXECUTION
RANGE:100
NUMBER:50


END OF EXECUTION - 2K CORE

EXECUTION TIME: 0.22 SECS.

ELAPSED TIME:   11.60 SECS.

.TYPE RAND.DAT
  26   4  93   2  84  27  55  30   3  86  92  27  75  60  21  96  40  75
   7  90  89  12  66  13  62  96  64  17  50   8  15   6  94  75  87  32
  84  74  93  99  61  35   1  60  16  35  25  19  89   2
.EDIT TESTR.ALG
*D250
*D325
*I225,25
00225    INPUT(3,"DSK");
00250    SELECTINPUT(3);
00275    OPENFILE(3,"RANGE.NUM");
*E

EXIT
```

*Selects channel 3 for the input.*

*Here the program is altered so that the input will come from a disk file called RANGE.NUM.*

*opens file RANGE.NUM for channel 3.*

```
.CREATE RANGE.NUM
00100    100
00200    50
00300    $
*E

EXIT

.R PIP
*RANGE.NUM/N←RANGE.NUM
*↑C
.EX TESTR,RAND
ALGOL: TESTR
LOADING

LOADER 1K CORE
```

*Strips the line numbers off the data file.*

*Algol can not handle line numbers on data files.*

```
EXECUTION

FATAL RUN-TIME ERROR AT ADDRESS 000167
MORE HEAP SPACE REQUIRED FOR I-O BUFFERS
?ACTION (H FOR HELP)? F

END OF EXECUTION - 2K CORE

EXECUTION TIME: 0.05 SECS.

ELAPSED TIME:   17.33 SECS.

.R ALGOL
*TESTR,←TESTR/1000D
*↑C
.EX TESTR,RAND
LOADING

LOADER 1K CORE
EXECUTION

END OF EXECUTION - 2K CORE

EXECUTION TIME: 0.13 SECS.

ELAPSED TIME:   3.13 SECS.

.TYPE RAND.DAT
 26   4  93   2  84  27  55  30   3  86  92  27  75  60  21  96  40  75
  7  90  89  12  66  13  62  96  64  17  50   8  15   6  94  75  87  32
 84  74  93  99  61  35   1  60  16  35  25  19  89   2
.
```

This was caused because I/O Buffers are put into the Heap and the default size is too small to do both input and output onto disk.

Causes the Heap size to become 1000 words.

END OF ALGOL SCRIPT

NOTE: There is useful information on the file SYS:ALGOL.DOC.

Solutions to Sample Problems

```
1.   REAL PROCEDURE PHIR(N);VALUE N;INTEGER N;
          PHIR←IF N=0 THEN 1.0 ELSE 1.0/(1.0+PHIR(N-1));
     REAL PROCEDURE PHII(N);VALUE N;INTEGER N;
          BEGIN REAL P;P←0.0;
          WHILE (N←N-1) > 0 DO P←1.0/(1.0+P);
          PHII←P;END;

3.   PROCEDURE HANOI(N,START,OTHER,FINISH);
       VALUE N,START,OTHER,FINISH;INTEGER N,START,OTHER,FINISH;
          BEGIN IF N=1 THEN BEGIN
               WRITE("MOVE DISC    1 FROM");PRINT(START,3);
               WRITE(" TO");PRINT(FINISH,3);NEWLINE;END
          ELSE BEGIN
               HANOI(N-1,START,FINISH,OTHER);
               WRITE("MOVE DISC");PRINT(N,3);WRITE(" FROM");
               PRINT(START,3);WRITE(" TO");PRINT(FINISH,3);NEWLINE;
               HANOI(N-1,OTHER,START,FINISH);END;
          BREAKOUTPUT;END;

5.   PROCEDURE PASCAL(N);VALUE N;INTEGER N;
          BEGIN INTEGER ARRAY P[1:N];INTEGER I,J;
          P[1]:=1;
          FOR I:=2 UNTIL N DO
               BEGIN P[I]:=0;
               FOR J:=I STEP -1 UNTIL 2 DO P[J]:=P[J]+P[J-1];
               END;
          FOR I:=1 UNTIL N DO PRINT(P[I],4);
          END;

7.   1.0  4.0
     2.0  4.0
     1.0  4.0
     3.0  5.0
```

*A reprint from* **COIYipUTtQPS "and automation** *November, 1969 vol 18, No. 12*

# APL: A PERSPICUOUS LANGUAGE

*Garth H. Foster*
*Department of Electrical Engineering*
*Syracuse University*
*Syracuse, N. Y. 13210*

***"In APL, a great many highly useful functions which are required in computing have been defined and given a notation consisting of a single character."***

The news and promotion copy now beginning to appear in many computer-related publications proclaiming APL *(A* Programming language) to be everything from a successor to PL/I (Programming Language One) to the most powerful interactive terminal system available, has no doubt been widely noticed. Such copy has led many to wonder what APL is, and after seeing its notation, many wonder about its clarity.

This article is not intended to a tutorial on APL, for that would take more space than is warranted here. However, let us discuss some of the aspects of APL which have excited the academic communities at a number of colleges and universities and at least one high school system, and which have triggered a number of implementation efforts in Canada, France, and the United States. The interested reader may then investigate further the many features of APL which cannot all be covered here. To assist in this direction, a rather complete bibliography of APL source material is appended to this article.

### Definition

The initials APL' derive from the title of the book "A Programming Language" by K.E. Iverson, published by John Wiley and Sons in 1962; and it was that publication which served as the primary vehicle for the publication of the initial definition of APL. Subsequent development of the language by Iverson has been done in collaboration with A.D. Falkoff at IBM's Thomas J. Watson Research Center, Yorktown Heights, New York.

The present form of APL is the APL\360 Terminal System, the implementation of APL on the system 360. Although there are implementations for the IBM 1130 and 1500 computers, when we speak of APL we shall mean APL\360.

The terminal system was designed by Falkoff and Iverson with additional collaboration from L.M. Breed, who,, with R.D. Moore (LP. Sharp Associates, Toronto) developed the implementation. Programming was by Breed, Moore, and R.H. Lathwell, with continuing contributions by L.J. Woodrum (IBM, Poughkeepsie), and C.H. Brenner, H.A. Driscoll, and S.E. Krueger (SRA, Chicago). Experience had been gained from an earlier version which was created for the IBM 7090 by Breed and P.S. Abrams (Stanford U., Stanford, California).

A computer language which is classified as algebraic is generally, but not exclusively, used to program problems requiring reasonably large amounts of arithmetic. Generally such languages have available, as formalized arithmetic operators with a notation, the operations of addition, subtraction, multiplication, division, and exponentiation; and there the list ends. To achieve other arithmetic operations either calls to pre-written subroutines must be made or the user must supply his own.

This is not true of APL; a great many highly useful functions which are required in computing have been defined and given a single character notation (some of these require 3 keystrokes, striking a key, backspacing and then striking another key; but usually only a single keystroke is required.)

### The APL Keyboard

Figure 1 shows the APL keyboard. The letters and numbers all appear in their usual places on a typewriter, except that the capital letters are in the lower case positions (the lower case letters do hot appear). The up-shift positions on the keyboard are occupied by symbols used to represent the powerful set of APL operators.

'APL should not be confused with "ABL — A Language for Associative Data Handling in PL/I/' by George G. Dodd, General Motors Research, 1966 Fall Joint Computer Conference.

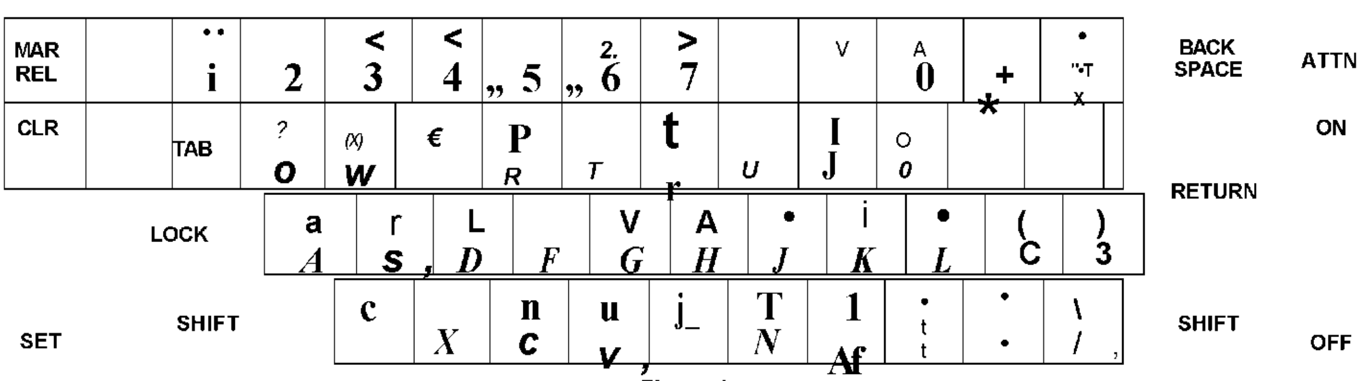



Figure 1

Besides +, -, x, ÷, (the familiar symbols for addition, subtraction, multiplication, and division located on the two right-most keys on the top row) and the symbol * assigned to represent exponentiation (the star over the P as in raising to a power), there are distinct single character notations for the operations of: negation; signum; reciprocal; logarithms (to both natural and arbitrary base); combinations and factorials; base e raised to a power; the residue of a number modulo any divisor. There are characters which represent taking: PI times a number; sines; cosines; tangents; hyperbolic sines, cosines, and tangents; and the inverse functions for the six preceding functions. Available too are: floor (truncating a number to the largest integer less than or equal to the number); ceiling (rounding up to the smallest integer greater than or equal to the number); and maximum or minimum of a pair of numbers.

APL also provides the relations which test whether two numbers are: less than; less than or equal to; greater than or equal to; greater than; equal; or not equal. The last two relations are also applicable to characters. These relations check to see, for example, if a relation is true and produce 1 (representing TRUE) or 0 (FALSE); these binary quantities may be operated upon by the logical functions of: OR; AND; NOT; NOR; and NAND. All these are also available as standard functions in APL, and are designated by a single character graphic. These operations are all summarized in Figure 2.

<table>
<tr><th colspan="2">Monadic form fB</th><th>f</th><th colspan="2">Dyadic form AfB</th></tr>
<tr><th>Definition or example</th><th>Name</th><th></th><th>Name</th><th>Definition or example</th></tr>
<tr><td>+B ←→ 0+B</td><td>Plus</td><td>+</td><td>Plus</td><td>2+3.2 ←→ 5.2</td></tr>
<tr><td>-B ←→ 0-B</td><td>Negative</td><td>-</td><td>Minus</td><td>2-3.2 ←→ ¯1.2</td></tr>
<tr><td>×B ←→ (B>0)-(B<0)</td><td>Signum</td><td>×</td><td>Times</td><td>2×3.2 ←→ 6.4</td></tr>
<tr><td>÷B ←→ 1÷B</td><td>Reciprocal</td><td>÷</td><td>Divide</td><td>2÷3.2 ←→ 0.625</td></tr>
<tr><td rowspan="2">B | ⌈B | ⌊B<br>3.14 | 4 | 3<br>¯3.14 | ¯3 | ¯4</td><td>Ceiling</td><td>⌈</td><td>Maximum</td><td>3⌈7 ←→ 7</td></tr>
<tr><td>Floor</td><td>⌊</td><td>Minimum</td><td>3⌊7 ←→ 3</td></tr>
<tr><td>*B ←→ (2.71828..)*B</td><td>Exponential</td><td>*</td><td>Power</td><td>2*3 ←→ 8</td></tr>
<tr><td>⍟*N ←→ N ←→ *⍟N</td><td>Natural logarithm</td><td>⍟</td><td>Logarithm</td><td>A⍟B ←→ Log B base A<br>A⍟B ←→ (⍟B)÷⍟A</td></tr>
<tr><td>|¯3.14 ←→ 3.14</td><td>Magnitude</td><td>|</td><td>Residue</td><td>Case | A|B<br>A≠0 | B-(|A)×⌊B÷|A<br>A=0, B≥0 | B<br>A=0, B<0 | Domain error</td></tr>
<tr><td>!0 ←→ 1<br>!B ←→ B×!B-1<br>or !B ←→ Gamma(B+1)</td><td>Factorial</td><td>!</td><td>Binomial coefficient</td><td>A!B ←→ (!B)÷(!A)×!B-A<br>2!5 ←→ 10   3!5 ←→ 10</td></tr>
<tr><td>?B ←→ Random choice from ⍳B</td><td>Roll</td><td>?</td><td>Deal</td><td>A Mixed Function</td></tr>
<tr><td>○B ←→ B×3.14159...</td><td>Pi times</td><td>○</td><td>Circular</td><td>See Table at left</td></tr>
<tr><td>~1 ←→ 0   ~0 ←→ 1</td><td>Not</td><td>~</td><td></td><td></td></tr>
<tr><td colspan="2" rowspan="2">(-A)○B | A | A○B<br>(1-B*2)*.5 | 0 | (1-B*2)*.5<br>Arcsin B | 1 | Sine B<br>Arccos B | 2 | Cosine B<br>Arctan B | 3 | Tangent B<br>(¯1+B*2)*.5 | 4 | (1+B*2)*.5<br>Arcsinh B | 5 | Sinh B<br>Arccosh B | 6 | Cosh B<br>Arctanh B | 7 | Tanh B<br><br>Table of Dyadic ○ Functions</td><td>∧<br>∨<br>⍲<br>⍱</td><td>And<br>Or<br>Nand<br>Nor</td><td>A | B | A∧B | A∨B | A⍲B | A⍱B<br>0 | 0 | 0 | 0 | 1 | 1<br>0 | 1 | 0 | 1 | 1 | 0<br>1 | 0 | 0 | 1 | 1 | 0<br>1 | 1 | 1 | 1 | 0 | 0</td></tr>
<tr><td><<br>≤<br>=<br>≥<br>><br>≠</td><td>Less<br>Not greater<br>Equal<br>Not less<br>Greater<br>Not Equal</td><td>Relations<br>Result is 1 if the relation holds, 0 if it does not:<br>3≤7 ←→ 1<br>7≤3 ←→ 0</td></tr>
</table>

**Figure 2**

## Order of Operations

Of course when such a host of generalized and powerful operations are at the disposal of the programmer, there is immediate concern as to the order or precedence of operations in an arithmetic expression written without parentheses.

Traditionally in algebraic languages, exponentiations were performed before multiplications and divisions, and these were done before additions and subtractions. One of the reasons for this choice (of hierarchy of operations) was that normal conventions in algebraic notation provided that the expression

$$5.6y^3 + 8y^2 + 2.84y + 9.06$$

could be written as

5.6 * y * * 3 + 8 * y * * 2 + 2.84 * y + 9.06

without the use of parentheses.

If one wanted to make the compiler work more efficiently when programming in the higher order language, then parens (parentheses) *were* used and the polynomial was "nested", so that in the above example one coded:

( ( 5.6 * y + 8) * y + 2.84) * y + 9.06

That is to say, one discarded the built-in precedence order.

Clearly, in APL having all the functions shown in Figure 2, the establishment of any hierarchy of operators would be arbitrary and open to question at best; and more than likely it would border on the impossible to justify the hierarchy in any reasonable way.

Thus in APL there is only *one* rule for evaluating all unparenthesized expressions (or within a pair of parens), and that rule is:

> Every operator takes as its right-hand argument the value of everything to the right of it (up to the closing parenthesis).

Now such a rule may seem strange and unfamiliar to someone who is now programming, but it has advantages:

(1) Uniformity–it is applied in the same way for all standard or primitive functions provided by the APL system as well as all functions (programs) written in APL by the user;

(2) Utility–this approach, for example, allows the nested polynomial to be written without parentheses as:[2]

9.06 + Y x 2.84 + Y x 8 + Y x 5.6

It is also possible to write continued fractions without parentheses and the rule given provides other interesting and useful results as a by product.

## Sum Reduction

Another area in which looping (of computer instructions) is explicitly required in most programming languages but not in APL is that of summing the components of a vector, which we will call for the sake of example, X. The usual approach is to initialize the sum to zero and then use a running index variable of a DO or FOR loop, and then take the summation by an expression like

SUM = SUM + Z(I).

In APL we use what is called *sum reduction.* This is the name for *conceptually* taking the vector X, inserting plus signs between each of its components, and then evaluating the resulting expression; its notation is simply +/X. If we had wanted to take the product of the elements of a vector Q, then in APL we write x/Q and this provides the *times reduction.*

[2]There are even more powerful ways to evaluate a polynomial expression in APL, but the availability of such methods does not reduce the effectiveness of the right to left rule just described.

### The Value of Powerful Operators

Thus the first area in which APL provides clarity in programming is by providing a large set of powerful functions. Now one may ask whether writing A ⌈ B in APL is only marginally more compact than say writing MAX(A,B). However, in APL we are allowed to use A!B to denote the combinations of taking B things A at a time. Such an operation in languages other than APL generally require the user to write his own program, perhaps calling upon routines to provide the factorials and if they in turn are not available, writing that routine also. The claim is that the presence of the APL operator ! in a program provides much more clarity than the presence of the equivalent routine in another programming language.

Of course one may argue that factorials and combinations are not needed all that much anyway. In many cases such a point of view may be correct; however, the fact still remains that the need for, say, the FORTRAN Library of subroutines indicates a need for arithmetic computations which are more complex than the operations included in the language as primitives. What APL has done therefore is to move in the direction of a library increasing the sophistication of the language, and at the same time simplifying the notation for using a much more powerful set of operators.

### Extending the Scope of Functions

The next step forward which APL has taken is to extend the scope of those functions shown in Figure 2, in the following way. In most languages extant today, if one writes A + B, then one commands the computer to add the number A to the number B. In APL the command still produces the addition of the single numbers, called scalars, if that is the nature of the variables A and B. If on the other hand, A and B are each names for a collection or string of numbers, called a vector, then the addition takes place on an element by element basis, with the first element of A being added to the first element of B, the second to the second, and so forth. The requirement is that either A or B may be a scalar while the other is a vector, but if they are both vectors, then they must have the same number of elements, that is, they must be of the same size.

If A and B are matrices of the same size (having the same number of rows and columns), then A + B in APL adds, on an element by element basis, matrix A to matrix B. To perform equivalent operations in most computer languages requires a DO or a FOR loop when adding vectors, or nested loops when adding matrices.

Two comments are relevant here. First, the explicit loops embodied in the DO or FOR loops are required by the language, but they are ancillary to communicating the process to be performed, say adding two matrices. Second, the utility of providing an extension of this nature, where the system assumes additional responsibility, is borne out, for example, in the MAT commands of BASIC. APL extends such ideas and applies them uniformly to all data structures treated in the language. In fact, from the programmer's point of view, one does not care in what sequence the operations in the loops implied in such an APL command take place. They could just as well be done all in parallel; the fact that the computer does not process the matrix elements in parallel does not matter. The extension of scope of the notation allows the algorithm to be thought of as acting on the data in parallel. Thinking about the computing process in this way gives new insight into the way the programs manipulate or transform the data.

### Allocating Space for Arrays

The philosophy is that the system should perform the tasks which are required by the computer but not essential to the algorithm. A useful extension is to have the computer assume the burden of allocation of space for arrays on a dynamic basis. This is done in the APL terminal system; for example, if one creates the vector X having components 2, 5, and 10, then X ← 2 5 10 is the ***specification*** or ***assignment*** of those constants to be the value of the variable X. No dimensioning is required. Later if we wish to respecify

| Name | Sign[1] | Definition or example[2] |
|---|---|---|
| Size | ρA | ρP ↔ 4   ρE ↔ 3 4   ρ5 ↔ ι0 |
| Reshape | VρA | Reshape A to dimension V   3 4ρι12 ↔ E<br>12ρE ↔ ι12   0ρE ↔ ι0 |
| Ravel | ,A | ,A ↔ (×/ρA)ρA   ,E ↔ ι12   ρ,5 ↔ 1 |
| Catenate | V,V | P,ι2 ↔ 2 3 5 7 1 2   'T','HIS' ↔ 'THIS' |
| Index[3,4] | V[A] | P[2] ↔ 3   P[4 3 2 1] ↔ 7 5 3 2 |
| | M[A;A] | E[1 3;3 2 1] ↔ 3 2 1<br>11 10 9 |
| | A[A;...;A] | E[1;] ↔ 1 2 3 4   E[;1] ↔ 1 5 9   'ABCDEFGHIJKL'[E] ↔ ABCD<br>EFGH<br>IJKL |
| Index generator[3] | ιS | First S integers   ι4 ↔ 1 2 3 4<br>ι0 ↔ an empty vector |
| Index of[3] | VιA | Least index of A in V, or 1+ρV   Pι3 ↔ 2   PιE ↔ 5 1 2 5<br>3 5 4 5<br>5 5 5 5<br>4 4ι4 ↔ 1 |
| Take | V↑A | Take or drop \|V[I]\| first (V[I]≥0) or last (V[I]<0) elements of coordinate I   2 3↑X ↔ ABC<br>EFG |
| Drop | V↓A | ¯2↓P ↔ 5 7 |
| Grade up[3,5] | ⍋A | The permutation which would order A (ascending or descending)   ⍋3 5 3 2 ↔ 4 1 3 2 |
| Grade down[3,5] | ⍒A | ⍒3 5 3 2 ↔ 2 1 3 4 |
| Compress[5] | V/A | 1 0 1 0/P ↔ 2 5   1 0 1 0/E ↔ 1 3<br>5 7<br>9 11<br>1 0 1/[1]E ↔ 1 2 3 4 ↔ 1 0 1⌿E<br>9 10 11 12 |
| Expand[5] | V\A | 1 0 1\ι2 ↔ 1 0 2   1 0 1 1 1\X ↔ A BCD<br>E FGH<br>I JKL |
| Reverse[5] | ⌽A | ⌽X ↔ DCBA<br>HGFE<br>LKJI   ⌽P ↔ 7 5 3 2   ⌽[1]X ↔ ⊖X ↔ IJKL<br>EFGH<br>ABCD |
| Rotate[5] | A⌽A | 3⌽P ↔ 7 2 3 5 ↔ ¯1⌽P   1 0 ¯1⌽X ↔ BCDA<br>EFGH<br>LIJK |
| Transpose | V⍉A | Coordinate I of A becomes coordinate V[I] of result   2 1⍉X ↔ AEI<br>BFJ<br>CGK<br>DHL<br>1 1⍉E ↔ 1 6 11 |
| | ⍉A | Transpose last two coordinates   ⍉E ↔ 2 1⍉E |
| Membership | A∊A | ρW∊Y ↔ ρW   P∊ι4 ↔ 1 1 0 0   E∊P ↔ 0 1 1 0<br>1 0 1 0<br>0 0 0 0 |
| Decode | V⊥V | 10⊥1 7 7 6 ↔ 1776   24 60 60⊥1 2 3 ↔ 3723 |
| Encode | V⊤S | 24 60 60⊤3723 ↔ 1 2 3   60 60⊤3723 ↔ 2 3 |
| Deal[3] | S?S | W?Y ↔ Random deal of W elements from ιY |

**Notes:**

1. Restrictions on argument ranks are indicated by: *S* for scalar, *V* for vector, *M* for matrix, *A* for Any. Except as the first argument of *S*ι*A* or *S*[*A*], a scalar may be used instead of a vector. A one-element array may replace any scalar.
2. Arrays used in examples: *P* ↔ 2 3 5 7   *E* ↔ 1 2 3 4 / 5 6 7 8 / 9 10 11 12   *X* ↔ ABCD / EFGH / IJKL
3. Function depends on index origin.
4. Elision of any index selects all along that coordinate.
5. The function is applied along the last coordinate; the symbols ⌿, ⍀, and ⊖ are equivalent to /, \, and ⌽, respectively, except that the function is applied along the first coordinate. If [*S*] appears after any of the symbols, the relevant coordinate is determined by the scalar *S*.

**Figure 3**

```
    ∇ R←AVERAGE V
[1]   R←(+/V)÷ρV
    ∇
```

**Figure 4**

```
    ∇ R←STATS X;SD;VAR;MEAN
[1]   R←MEAN,VAR,SD←(VAR←(+/(X-MEAN←AVERAGE X)*2)÷¯1+ρX)*
      0.5
    ∇
```

**Figure 5**

X to be all of those elements currently comprising X followed by the numbers 1.5 and 20.7, then X«-X, 1.5 20.7 *catenates* the constant vector 1.5 20.7 to X and *respecifies* X. The variable X is now a data object with 5 elements where X[1 J is 2 X[4] is 1.5 and X[5] is 20.7. We may query the system as to the size (number of components) of X by use of the function denoted by the Greek letter Rho. Thus, pX produces 5. The functions of *size* and *catenate are* summarized together with the rest of the mixed APL dyadic functions in Figure 3.

We will not here treat further the powerful functions of data manipulation illustrated there. However, we have now exposed the reader to a sufficient amount of detail in APL to understand Figure 4. This shows the listing of a user-written function, the name of which is AVERAGE. The first or *header line* of AVERAGE declares the syntax for that function, that is, it indicates that the explicit result will be called R and the vector of data to be averaged will be denoted by V. The line numbered [1] is the algorithm; and it is self explanatory, even at this point.

Figure 5 shows how AVERAGE is called within the function STAT to calculate the mean, variance, and standard deviation of a set of values. Here the variable names of MEAN, VAR, and SD refer to the result of the AVERAGE program and the calculated variance and standard deviation.

We do not illustrate the comparable programs in other languages; we leave to the reader the task of noting the coding compression achieved by APL. The APL array operations obviously provide both brevity and clarity in expression, and in that sense the programs may be thought of as somewhat self documenting.

The symbolic nature of APL makes it multilingual.

### Evaluation of APL

In these pages we have only scratched the surface of APL. The availability of a powerful set of functions having a generality and a sense of uniformity in definition is important in providing capability to program complex algorithms. The extension of operations uniformly to strings of quantities or tables of numbers is a step forward in programming, because a great deal of computing in science, government, and business may be cast in terms of those data structures. Also it is important to relieve the computer user of the burden of bookkeeping and housekeeping operations in computer programming in higher level languages, particularly in an interactive environment.

Enthusiastic supporters of APL have claimed that rather than standing for either *A* Programming Language or Another Programming Language, the initials APL stands for *A* Permanent Language. APL was first conceived of as a means of communication; and it will have importance in that regard independent of the availability of APL on a terminal system. The heart of communicating, describing, or programming a process is to make clear what is to be done. In fact I might suggest that Ken Iverson and his colleagues meant APL to be a tool so that we all could program lucidly. •

## An APL Bibliography


1. Abrams, P. S., **An** *Int9rpr9t%r* **for "Iverson Notation".** Stanford, Calif.: Computer Science Department, Stanford University, Tech. Report CS47, August 17, 1966.
2. Anscombe, F. J., **Use of Ivorson's Language APL for Statistical Computing.** New Haven: Department of Statistics, Yale University, July, 1968. TR-4 (AD 672-557).
3. Berges, G. A. and F. W. Rust, **APL/MSU Refer*** Manual.** Bozeman, Montana: Department of Electrical Engineering, Montana State Univ., September 26, 1968.
4. Berry, P. C, **API/1130 Primer.** IBM Corporation, 1968. (C20-1697-0).
5. Berry, P. C, **APLX360 Primer Student Text.** IBM Corporation, 1969. (C20-1702-0).
6. Breed, L. M. and R. H. lathwell, "The Implementation of APIA 360", **Interactive Systems for Applied Mathematics.** New York and London: Academic Press, 1968, pp. 390-399.
7. Calingaert, P., **Introduction to A Programming Language.** Chicago: Science Research Associates, field test edition, October, t967.
8. Creveling, Cyrus J. (Ed.), **Experimental Use of A Programming Language (APL) at the Goddard Space Flight Center.** Greenbelt, Maryland: Goddard Space Flight Center, Report No. x-560-68-420, November, 1968.
9. Charmonman, S., S. Cabay and M. I. Louie-Byne, **Use of APLX360 in Numerical Analysis.** Edmonton, Alberta, Canada: Department of Computing Science, University of Alberta, December, 1967.
10. FalkofT, A. D. and K. E. Iverson, **APLX360 User's Manual.** Yorktown Heights, N.Y.: T. J. Watson Research Center, IBM Corporation, 1968.
11. FalkofT, A. D. and K. E. Iverson, "The APL 360 Terminal System", **Interactive Systems for Applied Mathematics.** New York and London: Academic Press, 1968, pp. 22-37. (Also Research Note RC 1922, October 16, 1967, T. J. Watson Research Center.)
12. FalkofT, A. D., K. E. Iverson and E. H. Sussenguth, "A Formal Description of System/360". **IBM Systems Journal,** III, No. 3 (1964), pp. 193-262.
13. Gilman, L. I. and A. J. Rjse, **APLX360 An Interactive Approach.** IBM Corporation, 1969.
14. Hellerman, H., **Digital Computer System Principles.** New York: McGraw-Hill, 1967.
15. Iverson, I. E., "A Common Language for Hardware, Software and Applications". Eastern Joint Computer Conference, December, 1*62, pp. 121-129 (RC 749).
16. Iverson, K. E., "The Description of Finite Sequential Processes", **Information Theory, 4th London Symposium,** Colin Cherry (Ed.). London: Butterworth's 1961.
17. Iverson, K. E., **Elementary Functions: An Algorithmic Treatment.** Chicago: Science Research Associates, 1966.
18. Iverson, K. E., **Formalism in Programming Language.** Yorktown Heights, N.Y.: T. J. Watson Research Center, IBM Corporation, July 2, 1963. (RC-992).
19. Iverson, K. E., **A Programming Language.** New York: John Wiley and Sons, Inc., 1962.
20. Iverson, K. E., "A Programming Language". Spring Joint Computer Conference, May, 1962, pp. 245-351.
21. Iverson, K. E., "Recent Applications of a Universal Programming Language". New York: IFIP Congress, May 24, 1965. (Also Research Note NC-511, T. J. Watson Research Center.)
22. Iverson, K. E., **The Role of Computers in Teaching.** Kingston, Ont., Canada: Queen's University, Queen's Papers on Pure and Applied Mathematics, No. 13, 1968. Also issued as **The Use of APL in Teaching,** IBM Corporation, 1969. (320-0996-0).
23. Kolsky, H. G., "Problem Formulation Using APL". **IBM Systems Journal,** 8, 3(1969), pp. 204-217.
24. Krueger, S. E. and T. P. McMurchie, **A Programming Language.** Chicago: Science Research Associates, 1968.
25. Lathwell, R. H., **APLX360: Operations Manual.** IBM Corporation, 1968.
26. Lathwell, R.H., **APLX360: System Generation and Library Maintenance.** IBM Corporation, 1968.
27. MacAuley, Thomas, **CAL/APL: Computer Aided Learning/A Programming Language,** Author's Manual. Costa Mesa, Calif.: Information Services and Computer Facility, Orange Coast Junior College.
28. Pakin, Sandra, **APLX360 Reference Manual.** Chicago: Science Research Associates, 1968. (No. 17-1).
29. Rose, A. J., **Teaching the APLX360 Terminal System.** Yorktown Heights, N.Y.: T. J. Watson Research Center, IBM Corporation, August 28, 1966. (RC 2184.)
30. Rose, A. J., Videotaped APL Course. IBM Corporation, 1967.
31. Simillie, K. W., **STATPACK II: An APL Statistical Package.** Edmonton, Alberta, Canada: Department of Computer Science, University of Alberta, Publication No. 17, February 1969.
32. Woodrum, L. J., "Internal Sorting with Minimal Comparing". **IBM Systems Journal,** 8, 3(1969) pp. 189-203.

## Selected Bibliography for APL

[1] Berry, P.C., APL/360 Primer Student Text. IBM Corporation, 1969. (C20-1702-0).
An excellent introduction to the fundamentals of APL.

[2] Falkoff, A.D. and K.E. Iverson, APL/360 User's Manual. Yorktown Heights, N.Y.: T.J. Watson Research Center, IBM Corporation, 1968.

[3] Gilman, L.I. and A.J. Rose, APL/360 An Interactive Approach. IBM Corporation, 1969.
A textbook on APL (used in advanced undergraduate programming course at C-MU). Discusses some extensions to basic APL/360.

[4] Iverson, K.E., A Programming Language. New York: John Wiley and Sons, Inc., 1962.
The original definition of the notational scheme. Excellent in its own right, but not directly useful in learning one of the APL implementations.

[5] Pakin, Sandra, APL/360 Reference Manual. Chicago: Science Research Associates, 1968.
The definitive work on APL (as of 1968): explains each operator (with many examples). Note: this book is a reference manual, not a primer.

* Documentation for APL/10 system at C-MU can be found on the file APL.DOC. This file explains the differences between APL/10 and APL/360 and discusses the extensions implemented in APL/10, as well as how to get onto the APL/10 system at C-MU. *

# APL

## Simple Examples and Problems

Write APL expressions to perform the following:

1. Remove all duplicate elements from a vector V, and call the resulting compressed vector RES.

2. Determine which vowels ('AEIOU') and how many of each appear in a given character string C.

3. Given a vector V, whose components are decimal integers, determine how many decimal places each component has.

Write APL functions to perform the following:

4. Write a function PRI to list the prime numbers that lie between the integers R and S, inclusive.

5. Let X be a vector whose components are arranged in ascending order. Define a function MERGE which will insert the components of a vector V so that the resulting vector R is still in ascending order.

6. Write a <u>one-line</u> function to determine if a square matrix M is symmetric or not and have it print out either 'THE MATRIX IS SYMMETRIC' or 'THE MATRIX IS NOT SYMMETRIC'.

7. Without using the <u>array</u> catenation extension of the ravel operator, write a function to:
   a. catenate a vector R rowwise to a given matrix M.
   b. catenate a vector C columnwise to a given matrix M.

   Do not assume that the lengths of R or C are proper.

## APL

### ANSWERS TO SIMPLE EXAMPLES AND PROBLEMS

```
1.     RES←((⍳V)=V⍳V)/V

2.     +/'AEIOU'∘.=C

3.     1+⌊10⍟|V

4.            ∇Z←R PRI S;T
       [1]    Z←(R≤T)/T←(2=+/[1]0=(⍳S)∘.|⍳S)/⍳S
              ∇


5.            ∇ X MERGE V
       [1]    R←R[⍋R←X,V]
              ∇


6.            ∇ SYM M
       [1]    'THE MATRIX IS ';(0∊M=⍉M)/'NOT ';'SYMMETRIC.'
              ∇


7.            ∇ M PLUSROW R
       [1]    (1 0+⍴M)⍴(,M),R,((⍴M)[2]⍴0)
       [2]    ⍝ NOTE--NUMERIC INPUT IS ASSUMED SINCE R IS
       [3]    ⍝        EXTENDED BY 0'S IF TOO SHORT.
              ∇

              ∇ M PLUSCOL C
       [1]    ⍉(1 0+⍴⍉M)⍴(,⍉M),C,((⍴M)[1]⍴0)
       [2]    ⍝ NOTE--NUMERIC INPUT IS ASSUMED SINCE C IS
       [3]    ⍝        EXTENDED BY 0'S IF TOO SHORT.
              ∇
```

*APLSS\APL*

**TELETYPE SYSTEM MNEMONICS**

| TTY | *APL* | ALTERNATE TTY | TTY | *APL* |
|---|---|---|---|---|
| .AL | ⍺ | @A | .CB | ⍀ |
| .DE | ⊥ | @B | .CR | ⊖ |
| .DU | ∩ | @C | .CS | ⌿ |
| .FL | ⌊ | @D | .DQ | ⌹ |
| .EP | ∊ | @E | .GD | ⍒ |
| .US | _ | @F | .GU | ⍋ |
| .DL | ∇ | @G | .IB | ⌶ |
| .LD | ∆ | @H | .IQ | ⍇ |
| .IO | ⍳ | @I | .LG | ⍟ |
| .SO | ∘ | @J | .NN | ⍲ |
| ' | ' | @K | .NR | ⍱ |
| .BX | ⎕ | @L | .OQ | ⍈ |
| .AB | \| | @M | .OU | ⌀ |
| .EN | ⊤ | @N | .PD | ⍫ |
| .LO | ○ | @O | .QD | ⍔ |
| * | * | @P | .QQ | ⍞ |
| ? | ? | @Q | .RV | ⌽ |
| .RO | ⍴ | @R | .TR | ⍉ |
| .CE | ⌈ | @S | .XQ | ⍣ |
| .NT | ∼ | @T | .ZA | A̲ |
| .DA | ↓ | @U | .ZB | B̲ |
| .UU | ∪ | @V | .ZC | C̲ |
| .OM | ⍵ | @W | etc. | etc. |
| .LU | ⊃ | @X | | |
| ↑ | ↑ | @Y | | |
| .RU | ⊂ | @Z | | |

| TTY | *APL* | TTY | *APL* |
|---|---|---|---|
| .DD | ¨ | " | ⍝ |
| .GE | ≥ | & | ∧ |
| .GO | → | # | × |
| .LE | ≤ | % | ÷ |
| .NE | ≠ | ! | ! |
| .NG | ¯ | $ | $ |
| .OR | ∨ | / | / |
| | | ( | ( |
| | | ) | ) |
| | | etc. | etc. |

Script

R. Fennel, F. Pollack

```
.R APL                                  Gets you into APL
CHARACTER SET..
TTY                                     type tty if you are at teletype
APL-OLS                                    or APL if at datel
TTY100) 19:11:16  8/19/71 [65,10]       You are now in APL
CLEAR WS
      3#4                               entry is automatically indented
12                                      response is not
      X←3#4                             X is assigned the value of  3 times 4
      X
12                                      value of x typed out
      Y←-5                              y assigned -5
      X+Y                               the sum of x plus y
7
      144E.NG2                          exponential form. .ng is special minus
1.44                                    for constants.  It is not an operator
      P←1 2 3 4                         assign the vector 1 2 3 4 to p
      P#P                               multiply p by itself
1  4  9  16
      P#Y                               scalar is apllied to all elements
-5  -10  -15  -20
      Q←'CATS'                          assign q a 4 element character vector
      Q
CATS
      3+4#5+2                           evàluation is from right to left
31                                        with no operator precedence
      X←3
      Y←4
      (X#Y)+4
16
      X#Y+4
24
      XY                                the variable xy has not been defined
VALUE ERROR
      XY
      ↑

      X←@I5                             index generator function
      X
1  2  3  4  5
      @I 0                              the vector of 0 elements

      Y←5-X                             all scalar functions extend to vectors
      Y
4  3  2  1  0
```

```
      X<Y                          result of relational operator is 0 or 1
1 1 0   0   0
      §0  1                        Pi times 1
3.141592
      3X2                          3 divided 2
1.5
      •0  X  1  2                  pi divided by 1 2
3*141592    1.570796
      1fO  1                       sin 1
0.8414709
      2  §01  2                    cos 1 2
0.5403022    -0.4161468
```

```
                                   Function Definition
      •DL  Z-X  F  Y               function header, result plus 2 parameters
C13   Z-C<X*2)+Y*2>*.5             function body
C23   .DL                          close of function
      3  F  4                      executing function
S                                  result
      P-7
      Q-CP+15F   P-I               function call with expressions
      0                            value assigned to q
10
      4#3  F  4
20
      •6  B-G  A                   g is signum function. IA and B are
C13   B-<A>0)-A<0                  locals, function is monadic.
C2D   86
      G  4                         monadic function call
1
      G  .N66
-1
      6  X—6                       assignments may be anywhere in statement
-1
      •DL  H  A                    same as G but no result
C13   P*-<A>0>-A<0
C23   §G
      H  .NG6
      P
-1
      Y-H  «NG6
VALUE ERROR                        value error since function call returns
      Y-H  -6                      no explicit result
      t
      .DL  Z-FAC  NJ1              PAC is factorial function
cn    Z-I
C23   I-0
C33   LI.I-I+1                     LI becomes 3 at entrance into function
C43   •GO  0#  tl  I>N             11 is local.
C53   Z-Z#I
C63   •GO  LI
C73   •G
      FAC  3
6
      FAC  5
120
```

```
      T@HFAC←3 5               set to trace lines 3 and 5 of FAC
      X←FAC 3
FAC[3] 1                       Trace of FAC
FAC[5] 1
FAC[3] 2
FAC[5] 2
FAC[3] 3
FAC[5] 6
FAC[3] 4
      X
6
      T@HFAC←0                 set trace off

      @G G←M GCD N             Greatest common divisor
[1]   G←N
[2]   M←M @M N
[3]   .GO 4#M .NE 0
[4]   [1]G←M                   correction of line 1
[2]   [4]N←G                   resume with line 4
[5]   [1.BX]                   display line 1
[1]   G←M
[1]   [.BX]                    display entire function
    .DL  G←M GCD N
[1]   G←M
[2]   M←M.AB N
[3]   .GO 4#M.NE 0
[4]   N←G
    .DL
[5]   .GO 1                    enter new line
[6]   @G                       close of function. @g and .dl are the same
      36 GCD 44
4
      .DL GCD                  reopen definition
[6]   [4.1]M,N                 insert new line
[4.2] [.BX]                    display function
    .DL  G←M GCD N
[1]   G←M
[2]   M←M.AB N
[3]   .GO 4#M.NE 0
[4]   N←G
[4.1] M,N
[5]   .GO 1
    .DL
[6]   .DL                      close function.
      36 GCD 44
8  36
4  8
4
```

```
      ●G GCD[•BX]●G                        reopen,display, and close function
    •DL  G←M GCD N                         notice that when function is closed,
[1]    G←M                                 the lines are automatically renumbered.
[2]    M←M•AB N
[3]    •GO 4●M•NE 0
[4]    N←G
[5]    M,N
[6]    •GO 1
    •DL
      •DL GCD
[7]    [●H5]                               delete line 5 of function
[5]    ●G
       •DL Z←ABC X                         to demonstrate line editing
[1]    Z←(33●Q+(R●5)-6
[2]    [1•BX 8]                            edit line 1, print line and space in 8
[1]    Z←(33●Q+(R●5)-6
          /    1 /1                        / for delete, number for leave space.
[1]    Z←(3●Q)+(T●5)-6                     enter ) and t in proper place
[2]    •DL
       FAC 5                               FAC still defined
120
       )ERASE FAC                          Erase it

       FAC 5                               FAC no longer defined
SYNTAX ERROR
       FAC 5
      ↑
       )FNS                                List defined function in this workspace
ABC      F        G        GCD      H

       P←2 3 5 7                           assign p the vector 2 3 5 7
       ●RP                                 dimension of p
4
       T←'OH MY'                           character vector
       ●R T                                dimension of t
5
       P,P                                 catenation of two numeric vectors
2  3  5  7  2  3  5  7
       T,T                                 catenation of two character vectors
OH MYOH MY
       P,T                                 catenation of numbers with characters
DOMAIN ERROR                               not permitted
       P,T
      ↑

       N←5
       'NOTE: •IO';N;' IS ';•IO N          Mixed output
NOTE: •IO5 IS 1  2  3  4  5
```

```
      M-2 3tR 2 3 5 7 11 13         create matrix of dimension 2 3
      M

  2    3    5
  7   11   13
      2 4tR T                       reshape t into 2 4 matrix

OH M
YOH
      69RM                          reshape matrix into vector
2   3   5   7   11   13
      • BX«-P-#M                    ravel in row major order
2   3   5   7   11   13
      PC33                          indexing
5
      PC1 3 53                      indexing by a vector
2   5   11
      PC0I33                        first 3 elements of p
2   3   5
      PC§RP3                        last element of P
13
      MC 1123                       element in row 1 column 2 of m
3
      MCU3                          row 1 of M
2       5
      MCI U 3 23                    rows 1 and 1, columns 3 2

    5    3
    5    3
      A«-•ABCDEFGHIJKLMNOPQRSTUVWXYZ•
      ACM3                          A matrix index produces a matrix result

BCE
GKM
      ACMC1   193   233

EC
EC
      MC1M-15  3  12                respecifying the first row of M
      M

  15    3   12
   7   11   13
      Q«-3 1 5 2 4 6
      PCQ3
5   2   11   3   7   13
      0CQ3
5   3   4   1  2   6
      PC33
5
      )ORIGIN 0                     set origin to 0
WAS 1
      PC33                          fourth element of P
7
      PC0 1 23                      first 3 elements of P
2   3   5
      ©15
0   1  2   3   4
      )ORIGIN  1
WAS 0
```

```
      V←?3●R9                 get random 3 element vector whose elements
      M←?3 3●R9               are less than 10. and 2 random matrices
      N←3\3\?3 3●R 9
      V
6  8  1
      M

   8   1   5
   8   4   8
   6   6   6
      N

   4   7   2
   9   6   4
   4   6   4
      M+N                     sum element by element

  12   8   7
  17  10  12
  10  12  10
      M ●D N                  Minimum

   4   1   2
   8   4   4
   4   6   4
      M<N                     comparison(result 0 to 1)

  0  1  0
  1  1  0
  0  0  0
      +/V                     sum reduction of v
 15
      #/V                     product reduction
 48
      +/[1]M                  sum over first co-ordinate of m
 22  11  19
      +/[2]M                  sum over 2nd co-ordinate of M
 14  20  18
      +/M                     sum over last co-ordinate of m
 14  20  18
      ●S/M                    max over last co-ordinate of M
 8  8  6
```

```
      M+.#N                               ordinary matrix inner product

   61   92   40
  100  128   64
  102  114   60
      M+,<N                               inner product

  1  2  1
  1  1  0
  1  1  0
      M+.#V                               +.# inner product with vector right
61  88  90                                argument
      V
6  8  1
      V@J.#@I5                            Outer product (times)

     6   12   18   24   30
     8   16   24   32   40
     1    2    3    4    5
      V @J.<@I5                           Outer product with 1 2 3 4 5(less than)

   0  0  0  0  0
   0  0  0  0  0
   0  1  1  1  1
      .RO V@J.#M                          Outer product of rank 3
3  3  3
      Q←?10@R5                            random 10 element vector(1-5)
      Q
4  3  2  2  5  2  5  5  1  4
      +/[1]Q@J.=@I5                       Ith element of result is number of
1  3  1  2  3                             occurences of the value I in Q
      2 1.TR M                            ordinary transpose

    8   8   6
    1   4   6
    5   8   6
      .TR M                               same as monadic transpose

    8   8   6
    1   4   6
    5   8   6
```

```
      Q
4  3  2  2  5  2  5  5  1  4
      3 •RV Q                    rotate q to left by 3
2  5  2  5  5  1  4  4  3  2
      •NG3•RV Q                  Rotate Q to right by 3
5  1  4  4  3  2  2  5  2  5
      -3 •RV Q                   negative of rotate Q to left by 3
-2  -5  -2  -5  -5  -1  -4  -4  -3  -2
      Ø 1 2•RV[1]M               Rotate columns by different amounts

   8   4   6
   8   6   5
   6   1   8
      •NG2•RV[2]M                rotation of all rows by 2 to right

   1   5   8
   4   8   8
   6   6   6
      1 2 3•RV M                 Rotation of rows

   1   5   8
   8   8   4
   6   6   6
      •RV Q                      Reversal of Q
4  1  5  5  2  5  2  2  3  4
      •RV[1]M                    Reversal of M along first co-ordinate

   6   6   6
   8   4   8
   8   1   5
      •RVM                       Reversal along last co-ordinate of M

   5   1   8
   8   4   8
   6   6   6
      U←Q>4
      U
Ø  Ø  Ø  Ø  1  Ø  1  1  Ø  Ø
      U/Q                        Compression of Q by logical vector U
5  5  5
      (•T U)/Q                   compression by not U
4  3  2  2  2  1  4
      +/U/Q
 15
```

```
      _1 Ø 1[1]M                type-in error
SYNTAX ERROR
       1  Ø  1[1]M
      ↑
       [1.BX 9]                 editing of immediate line
       1 Ø 1[1]M
             1
       1 Ø 1/[1]M               insert `/´

   8   1   5                    compression along first co-ordinate of M
   6   6   6
       (,M>5)/,M
8  8   8  6  6  6               all elements of M which exceed 5
       V←1 Ø 1 Ø 1
       V\@I3                    expansion of iota 3
1  Ø   2  Ø  3
       V\M                      expansion along last co-ordinate of M

   8   Ø   1   Ø   5
   8   Ø   4   Ø   8
   6   Ø   6   Ø   6
       V\'ABC'                  expansion of character inserts blanks
A B C
       10@B 1 7 7 6             base 10 value of  1 7 7 6
1776
      _8 PB1 7 7 6              typing error
SYNTAX ERROR
       8 PB1 7  7  6
      ↑
       [1.BX7]
       8 PB1 7 7 6
         /1                     P should be @
       8 @B1 7 7 6              base 8 value of 1 7 7 6
1Ø22
       1Ø 1Ø 1Ø 1Ø@N1776        4 digit base 10 representation of 1776
1  7   7  6
       1Ø 1Ø@N1776              2 dgit base 10 representation of 1776
7  6
       24 6Ø 6Ø@B1 3 25         mixed base value
38Ø5
       24 6Ø 6Ø@N38Ø5
1  3   25
       2@B1 Ø 1 1 Ø             base 2 value
22
```

```
      P
2  3  5  7  11  13
      P .IO 7                least index of 7 in p
4
      P .IO 6                6 not in p, result is 1+.ro P
7
      P .IO 4 5 6 7          least index of 4 5 6 7 in p
7  3  7  4
      Q←5 1 3 2 4
      R←Q.IO .IO⊗RQ
      R
2  4  3  5  1
      Q[R]
1  2  3  4  5
      A←'ABCDEFGHIJKLMN'
      A←A,'OPQRSTUVWXYZ'
      A
ABCDEFGHIJKLMNOPQRSTUVWXYZ
      A⊗I'CAT'               rank of c a t in alphabet
3  1  20
      J←A⊗I'CAT'
      A[J]
CAT
      3?5                    random choice of 3 out of 5 with no repeat
2  4  1
      6?3
RANGE ERROR
      6?3
      ↑
      X←8?8                  a random permutation vector
      X
7  1  3  2  8  6  5  4
      .GUX                   the grade up of X
2  4  3  8  7  6  1  5
      X[.GU X]               X in ascending order
1  2  3  4  5  6  7  8
      X[.GD X]               X in descending order
8  7  6  5  4  3  2  1
      U←A ⊗E 'NOW IS THE TIME'  Membership
      (.BX←U)/A
0  0  0  0  1  0  0  1  1  0  0  0  1  1  1  0  0  0  1  1  0  0  1  0
      0  0
EHIMNOSTW
      (⊗I9)⊗E3 6 2 9
0  1  1  0  0  1  0  0  1
```

```
      •DL   Z-BIN  N
 cn     z-i
 C23    LA»Z*-<Z#0>+0*Z
 C33    •GO LA#N.6E  .RO  Z
      • DL
       >FNS                          List of functions in workspace
ABC        BIN        ENTERTEXT        F        6        6CD       H
MULTDRILL
       )VARS                         List of variables in workspace
A         D                    LA     M        N        P        0        R
T         U                    X      Y
       A«" *)FNS                     string containing two APL statements
)VARS*
       A
>FNS
)VARS
       B<-tE  A                      Execution of string, value of first printed
ABC        BIN        ENTERTEXT        F        G        GCD       H
MULTDRILL                            Second assigned to B
       B
A         D                    LA     M        N                 0
T         U                    X      Y
       A-'BIN  3*
       B~#E  A                       Execute string value returned in b
       B                             print value of function call
1 3   3   1
       B*-8N  'BIN *                 get  lines of function BIN
       B
•DLZ-BIN  N
t+l
LAIZ*-<Z#0)+0*Z
•GOLA#N.GE.ROZ
•DL
       )ERASE  BIN                   erase function

       BIN  3
SYNTAX  ERROR
       BIN  3
      t
       0E  B                         execute will redefine function

       BIN  3                        try it out
1 3   3   1
       INV-.DQ  M                    get inverse of matrix
       M+.#INV                       result should be identity matrix

     1»000000E0        2.980232E-8      0.0
     0.0               1*000000E0       0.0
     0*0               2.980232E-8      1«000000E0
       >OFF.HOLD                     sign off APL
TTY100)  20:52J05_  8/19/71
CONNECTED    1:40:48  CPU TIME    0:00:17
```

BLISS

C. Geschke (Revised, 6/29/72, C. Weinstock)

INTRODUCTION

BLISS-10 is a language specifically designed for writing software systems such as compilers and operating systems for the PDP-10. While much of the language is relatively "machine independent" and could be implemented on another machine, the PDP-10 was always present in our minds during the design; and as a result, BLISS-10 can be implemented very efficiently on the 10. This is probably not true for other machines.

We refer to BLISS-10 as an "implementation language." This phrase has become quite popular lately, but apparently does not have a uniform meaning. Hence, it is worthwhile to explain what we mean by the phrase and consequently what our objectives were in the language's design. To us the phrase "implementation language" connotes a higher level language suitable for writing production software; a truly successful implementation language would completely remove the need and/or desire to write in assembly language. Furthermore, to us, an implementation language need not be machine independent--in fact, for reasons of efficiency, it is unlikely to be.

Many reasons have been advanced for the use of a higher level language for implementing software. One of the most often mentioned is that of speeding up its production. This will undoubtedly occur, but it is one of the less important benefits, except insofar as it permits fewer, and better programmers to be used. Far more important, we believe, are the benefits of documentation, clarity, correctness, and modifiability. These were the most important goals in the design of BLISS-10.

Some people, when discussion the subject of implementation languages, have suggested that one of the existing languages, such as PL/I, or at most a derivative of one, should be used; they argue that there is already a proliferation of languages, so why add another. The only rational excuse for the creation of yet another new language is that existing languages are unsuitable for the specific applications in mind. In the sense that all languages are sufficient to model a Turing machine, <u>any</u> of the existing languages, LISP for example, would be adequate as an implementation language. However, this does not imply that each of these languages would be equally convenient. For example, FORTRAN <u>can</u> be used to write list processing programs, but the lack of recursion coupled with the requirement that the programmer code his own primitive list manipulations and storage control makes FORTRAN vastly inferior to, say, LISP for this type of programming.

What, then, are the characteristics of systems programming which should be reflected in a language especially suited for the purpose? Ignoring machine dependent features (such as a specific interrupt structure) and recognizing that <u>all</u> differences in such programming characteristics are only one of degerr, three features of systems programming stand out:

1. Data structures. In no other type of programming does the variety of data structures nor the diversity of optimal representations occur.
2. Control structures. Parallelism and time are intrinsic parts of the programming system problem.*
3. Frequently, systems programs cannot presume the existence of large support routines (for dynamic storage allocation, for example).

---

* Of course, parallelism and time are intrinsic to real time programming as well.

These are the principal characteristics which the design of BLISS-10 attempts to address. For example, taking point (3), the language was designed in such a way that no system support is presumed or needed, even though, for example, dynamic storage allocation is provided. Thus, code generated by the compiler can be executed directly on a "bare" machine. Another example, taking point (1), is the data structure definition facility. BLISS contains no implicit data structures (and hence no presumed representations for structures), but rather provides a method for defining a representation by giving the explicit accessing algorithm.

CMU I.O. and Peripherals

There are several peripheral packages built around the BLISS-* 10 language. Here is a list of the packages and their implementations, which can provide more detailed information:

IO/DYIO:

The BLISS-10 language has no I/O facilities. This package provides a library of routines which can be used to build I/O handling capabilities within BLISS-10 programs.

Documentation: IO.DOC Also in Bliss Reference Manual
Implementor: J. Newcomer

HELP:

This is a set of routines useful in augmenting the DDT debugging facility which unfortunately is not geared to stacks, block-structured symbol tables, etc.

Documentation: HELP.DOC
Implementor: W. Wulf

TIMER:

A package which can be loaded with your BLISS-10 to provide statistics on the run-time of routines in your BLISS-10 program. Extremely useful in the design-implementation cycle of an efficient programming system.

Documentation: TIM.DOC

Implementor: J. Newcomer

POOMAS:

"Poor-Mans-Simulation-Package." An adjoint to BLISS-10 of the same flavor as the union of SIMULA and ALGOL.

Documentation: POOMAS.DOC

Implementor: A. Lunde

SIX12:

A high level debugging package. Since it knows about the Bliss-10 run time environment it is useful in interactive Bliss deburring.

Documentation: SIX12.DOC

Implementors: C. Weinstock
W. Wulf

REFERENCES


[1] Wulf, Russell, Habermann, Geschke, Apperson, Wile, Brender, "BLISS Reference Manual," Computer Science Department Report, CMU, 1970.

[2] Wulf, Russell, Habermann, "BLISS: A Language for Systems Programming," DECUS Proceedings, Spring, 1970.

[3] Wile, Geschke, "Efficient Data Accessing in the Programming Language BLISS," SIGPLAN Conf. on Data Structures in Programming Languages, SIGPLAN Notices, February, 1971.

[4] Wulf, Geschke, Wile, Apperson, "Reflections on a Systems Programming Language," SIGPLAN Conf. on Implementation Languages, SIGPLAN Notices, October, 1971.

[5] Wulf, Russell, Habermann, "BLISS: A Language for Systems Programming," C.A.C.M. (to be published).

Some fairly extensive examples have been prepared as an appendex to the BLISS-10 Reference Manual. Anyone interested in these can see the BLISS-10 implementors for a copy.

SIMPLE EXAMPLES

1)
```
! find index of first space in a line
! image of 80 characters (one per word)
! index = -1 implies none found

    index ← incr j from 0 to 79 do
              if . line [.j] eql #40 then
                    exitloop .j;
```

2)
```
! find last item of simple list
    link ← . beginning of linked list;
    while .. link neg 0 do link ← ..link;
! link contains address of last item
```

3)
```
! add the first N numbers
    sum ← 0;
    incr j from 1 to .n do sum ← .sum+.j;
```

4)
```
! routine to compute factorial
    routine factorial (n) =
        if .n eql 0
            then 1
            else .n* factorial (.n-1);
```

---

THE FOLLOWING IS AN EXAMPLE OF A TERMINAL SESSION USING BLISS10. COMMENTS ARE DISTINGUISHED FROM ACTUAL MACHINE INTERACTION BY BEING ENCLOSED IN ----'ED LINES. SINCE BLISS10 HAS NO BUILT-IN I/O FACILITIES, YOU WILL FIND THE USE OF A FILE IOPRE.BLI WHICH WAS CREATED USING TECO. ITS CONTENTS ARE:

---

```
.TYPE IOPRE.BLI

MODULE TTIO(STACK)=

BEGIN

MACHOP TTCALL=#51;

MACRO   INC= (REGISTER Q; TTCALL(4,Q); .Q)$,
        OUTC(Z)= (REGISTER Q; Q←(Z); TTCALL(1,Q))$,
        OUTSA(Z)= TTCALL(3,Z)$,
        OUTS(Z)= OUTSA(PLIT ASCIZ Z)$,
        OUTM(C,N)= DECR I FROM (N)-1 TO 0 DO OUTC(C)$,
        CR= OUTC(#15)$,  LF= OUTC(#12)$,  NULL= OUTC(0)$,
        CRLF= OUTS('?M?J?0?0')$,
        TAB= OUTC(#11)$;


ROUTINE OUTN(NUM,BASE,REQD)=
  BEGIN OWN N,B,RD,T;
    ROUTINE XN=
      BEGIN LOCAL R;
        IF .N EQL 0 THEN RETURN OUTM("0",.RD-.T);
        R←.N MOD .B; N←.N/.B; T←.T+1; XN();
        OUTC(.R+"0")
      END;

    IF .NUM LSS 0 THEN OUTC("-");
    B←.BASE; RD←.REQD; T←0; N←ABS(.NUM); XN()
  END;


MACRO   OUTD(Z)= OUTN(Z,10,1)$,
        OUTO(Z)= OUTN(Z,8,1)$,
        OUTDR(Z,N)= OUTN(Z,10,N)$,
        OUTOR(Z,N)= OUTN(Z,8,N)$;
```

---

TY

NOW WE WILL BUILD A PROGRAM TO PRINT THE FACTORIALS FROM 0 TO 12 AT THE TTY. WE HAVE ALREADY CREATED THE FILE FACT.BLI USING TECO. ITS CONTENTS ARE:

---

```
.TYPE FACT.BLI


ROUTINE FACTORIAL(N)=
  IF .N EQL 0 THEN 1 ELSE .N*FACTORIAL(.N-1);

  CRLF; TAB; OUTS('N'); TAB; OUTS('N!'); CRLF; CRLF;
  INCR I FROM 0 TO 12 DO
    BEGIN
      TAB;
      OUTD(.I);
      TAB;
      OUTD(FACTORIAL(.I));
      CRLF;
    END;

END ELUDOM
```

---

NOTICE THAT THE FILES IOPRE.BLI AND FACT.BLI WHEN CONCATENATED WILL FORM A SYNTACTICALLY VALID BLISS10 MODULE. NOW WE ARE READY TO COMPILE THE PROGRAM. BLISS10 ACCEPTS THE STANDARD DEC COMMAND STRING ALONG WITH A LARGE NUMBER OF OPTIONAL (AND DEFAULTED) SWITCHES WHICH ARE DESCRIBED IN THE MANUAL. IN THIS EXAMPLE WE ARE NOT GOING TO USE ANY OF THE CCL COMMANDS ALTHOUGH THE CMU MONITOR DOES RECOGNIZE THE .BLI EXTENSION AND WILL HANDLE BLISS10 FILES.

THE COMMAND STRING WILL PRODUCE A .REL FILE NAMED FACT.REL.

---

```
. R BLISS
*FACT,-IOPRE,FACT


MODULE LENGTH =91+16

COMPILATION COMPLETE
```

NOW WE ARE READY TO LOAD THE PROGRAM.

```
 .LOAD FACT
LOADING

LOADER 2+IK CORE

EXIT

.START


        N        N!

        0

        2        2
        3        6
        4        24
        5         120
        6        720
        7        5040
        8        40320
        9        362880
        10       3628800
        11       39916800
        12       479001600

EXIT
```

## LISP

D. Waterman

The following quote from the introduction to the LISP 1.5 Primer by Clark Weissman will serve to introduce the language:

> "LISP is an unusual language in that it is both a formal mathematical language, and (with extensions) a convenient programming language. As a formal mathematical language, it is founded upon a particular part of mathematical logic known as recursive function theory. As a programming language, LISP is concerned primarily with the computer processing of symbolic data rather than numeric data.
>
> From childhood we are exposed to numbers and to ways of processing numerical data, such as basic arithmetic and solutions to algebraic equations. This exposure is based upon a well-established and rigorously formalized science of dealing with numbers. We are also exposed to symbolic data--such as names, labels, and words--and to ways of processing such data when we sort, alphabetize, file, or give and take directions. Yet the processing of symbolic data is not a well-established science. In learning al algebraic programming language, such as FORTRAN or ALGOL, we call upon our experience with numbers to help us understand the structure and meaning (syntax and semantics) of the language.
>
> In learning a symbolic programming language such as LISP, however, we cannot call upon our experience, because the formalism of symbolic data processing is not part of this experience. Thus, we have the added task of learning a basic set of formal skills for representing and manipulating symbolic data before we can study the syntax and semantics of the LISP 1.5 programming language.
>
> LISP is designed to allow symbolic expressions of arbitrary complexity to be evaluated by a computer. To achieve a thorough understanding of the meaning, structure, construction, and evaluation of symbolic expressions, is to learn how to program in LISP."

REFERENCES

[1] Quam, Lynn, Stanford LISP 1.6 Manual, Stanford AI Project, September, 1969.

[2] McCarthy, John, Paul W. Abrahams, Daniel J. Edwards, Timothy P. Hart, and Michael I. Levin, LISP 1.5 Programmer's Manual, Cambridge, Massachusetts, The MIT Press, 1962.

[3] Hart, Timothy P., and Thomas G. Evans, "Notes on Implementing LISP for the M-460 Computer," in Edmund C. Berkeley and Daniel G. Bobrow (eds.), The Programming Language LISP: Its Operation and Applications, 2nd ed., Cambridge, Massachusetts, The MIT Press, 1966, p. 191.

[4] Weissman, Clark, LISP 1.5 Primer, Dickenson Publishing Co., 1967.

The first reference, the Stanford LISP 1.6 Manual, contains most of the special features of the CMU LISP and outlines the differences between CMU LISP and the LISP described in the last three references. Reference 3 contains an excellent set of LISP exercises with solutions, pp. 73-92.

RECURSIVE EXAMPLE

A simple example of a recursive LISP program to sum the digits in a list is shown below.

```
(DEFPROP SUM (LAMBDA (L) (COND
((NULL L) 0)
(T (PLUS (CAR L) (SUM (CDR L))))
)) EXPR )
```

Executing (SUM (QUOTE(1 9 7 1))) produces 18.

SAMPLE PROBLEMS

Write LISP functions for the following purposes:

1. to determine whether an atom is a member of a list.

   e.g. member [B;(A B C)] = T
        member [X;(A B C)] = F
        member [A;(B (A B) C) = F

2. to produce a tale (list of dotted pairs) given two lists,

   one of the references, and other of values.

   e.g. pair [(ONE TWO THREE);(1 2 e)] = ((ONE . 1)(TWO . 2)(THREE . 3))
        pair [(PLANE SUB);(B47 THRESHER)] = ((PLANE . B47)(SUB . THRESHER))

3. to append one list onto another.

   e.g. append [(A B C);(D E F)] = (A B C D E F)
        append [(A B) C (D E)));((A))] = ((A B) C (D (E)) (A))

4. to delete an element from a list.

   e.g. delete [Y;(X Y Z)] = (X Z)
        delete X;((UV) X Y)] = ((U V) Y)

5. to reverse a list. (Hint: use <u>append</u>.)

   e.g. reverse [(A B C)] = (C B A)
        reverse [(A (B C) D)] = (D (B C) A)

6. to produce a list of all the atoms which are in either of two lists.

   e.g. union[(U V W);(W X Y)] = (U V W X Y)
        union[(A B C);(B C D)] = (A B C D)
        union[(A B C);(A B C)] = (A B C)

7. to produce a list of all the atoms in common to two lists.

   e.g. intersection [(A B C);(B C D)] = (B C)
        intersection [(A B C);(A B C)] = (A B C)
        intersection [(A B C);(D E F)] = NIL

8. to find the last element on a list.

   e.g. last[(A B C)] = C
        last[((A B)(C))] = (C)

9. to reverse all levels of a list.

   e.g. superreverse[(A B (C D))] = ((D C) B A)
   superreverse[((U V)((X Z) Y))] = ((Y (Z X))(V U))

10. to determine whether a given atomic symbol is some part of an S-expression.

   e.g. part[A;A] = T
   part[A;(X . (Y . A))] = T
   part[A;(U V (W . X) Z)] = F

Lisp Script

## Examples: CMU LISP

```
.login
JOB 1 CMU10A 6.N /DEC 5S02.C   TTY33
#a330dw28
PASSWORD: [redacted]

 0035       20-Aug-71
WELCOME BACK.
READ SYS:NOTICE FOR INFORMATION ABOUT YOUR DISK FILES.
FOUR DIAL UP TTY LINES (687-3411) AND FOUR
DATEL LINES (683-8330) NOW IN SYSTEM.
NEW BLISS IN SYSTEM. OLD VERSION CALLED OLDBLI....BA03

 .tty no lc     <------ LISP wants upper case, this tells the system no lower case

 create file1 <------ Creating a file using SOS
00100      (defprop factorial
00200      (lambda (x)(cond
00300      ((zerop n) 1)
00400      (t (times n (factorial (sub1 n))))
00500      )) expr)
00600      $ <------ altmode
$
 *e

 EXIT
```

```
.r lisp 15

ALLOC?

AUXILIARY FILES?
DECIMAL?y
STANFORD AI LISP 1.6 AT CMU   9-SEP-70
SEE LISP.DOC FOR HELPFUL (SIC) HINTS

*(setq *nopoint t)

T
*(grindef factorial)

NIL
*(inc (input dsk: file1))

NIL

FACTORIAL
```

ALLOC?, AUXILIARY FILES?: if answer is 'no' type a space (for TTY) or a space, attention (for 2741 or datel)

DECIMAL?y: If answer is (yes) type a y (for TTY) or a y, attention (for 2741 or datel)

*(setq *nopoint t): when this is T, the decimal point in integers is not printed

*(grindef factorial): this lists a LISP function, if the function does not exist it returns NIL

*(inc (input dsk: file1)): the function FACTORIAL is not in the system, so it is real in from the DISK file just created

```
*(grindef factorial)

(DEFPROP FACTORIAL
 (LAMBDA(X)
  (COND ((ZEROP N) 1) (T (TIMES N (FACTORIAL (SUB1 N))))))
EXPR)

NIL
*(ed)

*g factorial

*r(x)¬(n)¬
$*!
 **p factorial

*!!

NIL
*(grindef factorial)

(DEFPROP FACTORIAL
 (LAMBDA(N)
  (COND ((ZEROP N) 1) (T (TIMES N (FACTORIAL (SUB1 N))))))
EXPR)

NIL
```

the function is now in the system, however it is clear that it contains an error: (X) should be (N)

call the LISP editor to correct the error

inside the editor, 'get' FACTORIAL

'replace' (X) with (N), first occurrence

'put' corrected version into system with the name FACTORIAL

exit from the LISP editor (this is ↑ on the TTY)

the function has indeed been corrected

```
•(factorial 3)

6
•(factorial k)

lk
*(defprop prints (lambda (xMcond
*((nu11 xHterpri))
*(t (and (princ (car x))(princ (quote " "))
•(prints (cdr x)))))) expr)

(RINTS
*(gr indef pri nts)


fIDEFPROP PRINTS
 (LAMBDA(X)
  (COND ((NULL X) (TERPR I) )
      (T
       (AND (PRINC (CAR X))
            (PRINC (QUOTE " ")>
            (PRINTS (CDR X))))))

EXPR)

ML
•(prints (quote (this is a print test))) ^
THIS IS A PRINT TEST
```

PHIiN)TS pvt*ts O- fat UidUu-t p*W**«*

```
NIL
*(print (quote (this is a print test)))   <-- PRINT, a system function, prints the parentheses

(THIS IS A PRINT TEST)
(THIS IS A PRINT TEST)
*(ed)   <-- back to the LISP editor

*f (factorial prints) dsk: file1   <-- 'file' FACTORIAL and PRINTS on DISK with
                                       filename FILE1
*|c
/c

..type file1   <-- list FILE1

(DEFPROP FACTORIAL
 (LAMBDA (N) (COND ((ZEROP N) 1) (T (TIMES N (FACTORIAL (SUB1 N))))))
EXPR)

(DEFPROP PRINTS
 (LAMBDA (X) (COND ((NULL X) (TERPRI)) (T (AND (PRINC (CAR X)) (PRINC (QUOTE " ")) (PRINTS (CDR X))))))
EXPR)

|c
```

L*

G. Robertson and D. McCracken

L* is a system on the PDP-10 for constructing software systems, which is under development at CMU by A. Newell, D. McCracken, G. Robertson, and P. Freeman. The current version, L* (G), is the seventh to be designed for the PDP-10 and the fourth to become a running system. There are also three running systems on the PDP-11, the most current version being L*11 (C). A running system on 360 TSS also exists, L*360.

The design rationale for L* is discussed in the article, "The Kernel Approach to Building Software Systems," which appears in the 1970 Computer Science Research Review. This guide makes brief references to the principles set forth in that article.

L* is intended to be a complete system for running and constructing software systems. Completeness implies that one should be able to perform, and to construct systems for performing, the following:

a) Processing of arbitrary data types, e.g., symbolic structures, lists, numbers, arrays, bit strings, tables, text

b) Editing

c) Compiling and assembling

d) Language interpreting

e) Debugging

f) Operating systems, e.g., resource allocation, space and time accounting, exotic control (parallel and supervisory control)

g) Communication between user and system, e.g., external languages, dynamic syntax, displays, etc.

L* is a kernel system. It starts with a small kernel of code and data and is grown from within the system. Thus, L* does not perform all the functions above when it exists only as a kernel. It does have means to construct systems for them all.

L* is designed for the professional programmer. It assumes someone sophisticated in systems programming who wants to build up his own system and who will modify any presented system to his own requirements and prejudices. Thus, L* is intended to be transparent. All mechanisms in the total system are open for understanding and modification. No mechanisms are under the floor.

One of the design goals of the L*(G) system was that it should be entirely self-documenting on-line to the maching, but this goal was not fully realized. The listings of the system which are available on the [AllJBLGjdjif] disk area may be used as documentation. There is available an interactive script which teaches L*L, the simple list processing language at the heart of the L*(G) system.

Getting into L*(G) is very simple. All that is neccessary is:

R LSGA

HELP

The response of the HELP command will be sufficient to get you started in the system.

There is also a file SYS: LSG.DOC which contains a few helpful hints on using L*(G)

There is a new (and hopefully final) version of L*, called L*(H) which should be completed during the fall of 1972. Along with L*(H) there will also be a new PDP-11 version of L*.

L*(H) will have complete facilities for assembly, translation, filing and documentation, and will be written up in final form for publication.

As soon as L*(H) becomes available for use, documentation on getting into the system will appear on file SYS: LSH.DOC

MACRO 10

D. Bajzek

MACRO 10 is the symbolic assembly language for the PDP-10 machine language. It is characteristic of most machine languages in that is is most useful in fully utilizing the facilities of a PDP-10.

The PDP-10 Reference Handbook is a complete reference guide for the MACRO 10 assembler since no special CMU features have been added to this processor. Chapters 1 and 2 contain a complete description of the PDP-10 instruction set and the MACRO 10 assembler.

Chapter 3 contains detailed information on communication with the TOPS-10 monitor. Section 4.10 of this chapter is very important since it describes all the input/output operators. In particular, this section describes the use of the directory devices, disk and DECtape, which are most commonly used since they provide random access data storage. Also included are diagrams and explanations of data structures and programming examples on

1) how to create data files and transfer data in buffered mode (pp. 3-197),
2) how to transfer data in unbuffered mode (pp. 3-199),
3) a general subroutine to input one character (pp. 3-200),
4) and a general subroutine to output one character (pp. 3-201).

In general, to create or update a data file on disk or DECtape, it is necessary to understand the following operators:

{OPEN
{INIT (pp. 3-189) The OPEN and INIT programmed operators initialize a file by specifying a device (or data channel), logical device name, initial file status, and the location of the input and output buffer headers.

{INBUF
{OUTBUF (pp. 3-193) can be used to establish buffer data storage areas.

LOOKUP (pp. 3-194) selects a file for input on the specified channel.

ENTER (pp. 3-195) selects a file for output to a specified channel.

RENAME (pp. 3-196) is used to

a. alter the filename, filename extension, and the protection, or

b. delete a file associated with a specified channel on a directory device.

{INPUT
{IN (pp. 3-198) transmits data from the file selected on the specified channel to the user's core area.

{OUTPUT
{OUT (pp. 3-198) transmits data from the user's core area to the file selected on the specified channel.

CLOSE (pp. 3-203) terminates data transmission on the specified channel.

RELEASE(pp. 3-205) releases the channel.

The following is an example of a MACRO 10 program which merely reads a string on one-digit octal numbers, ignoring all other characters, from an ASCII text file called DATA.FIL. It then sums these digits and prints out their octal sum on the TTY.

```
        TITLE ADDER
; GIVE ACCUMULATORS SYMBOLIC NAMES
        A=1
        A1=2
        DIGIT=3
        SUM=4
        COUNT=5
        PNT=6

; DEFINE I/O CHANNEL
        INCHN=1

;NOW BEGIN

START:  INIT    INCHN,1                 ;INITIALIZE INPUT CHANNEL IN
                                        ;ASCII LINE MODE
        SIXBIT  /DSK/                   ;LOGICAL DEVICE NAME IS DSK
        XWD     0,IBUF                  ;NEED TO GIVE NAME OF INPUT
                                        ;BUFFER HEADER ONLY, SINCE WE
                                        ;ONLY WISH TO INPUT FROM THIS
                                        ;DEVICE.
        JRST    NOTAVL                  ;GO TO ERROR ROUTINE IF DEVICE
                                        ;IS NOT AVAILABLE.
        INBUF   INCHN,1                 ;SMALL AMOUNT OF DATA, WE ONLY
                                        ;NEED 1 BUFFER IN RING.
        LOOKUP  INCHN,INNAME            ;LOOK UP FILE WHICH IS DESCRIBED
                                        ;IN INNAME.
        JRST    NOTFND                  ;ERROR IF FILE NOT FOUND

; PREPARE TO START SUMMING
        SETZM   SUM                     ;INITIALIZE SUM TO ZERO

LOOP1:  JSR     GETCHR                  ;GETCHR RETURNS WITH ASCII CHAR
                                        ;IN DIGIT
        CAIG    DIGIT,67                ;MAKE SURE ASCII CHAR IS REALLY
        CAIGE   DIGIT,60                ;AN OCTAL DIGIT
        JRST    LOOP1                   ;IF IT'S NOT, IGNORE IT AND GO
                                        ;GET ANOTHER CHAR
        SUBI    DIGIT,60                ;GET ACTUAL VALUE OF ASCII OCTAL
                                        ;DIGIT.
        ADDM    DIGIT,SUM               ;ADD DIGIT TO SUM
        JRST    LOOP1                   ;GO GET NEXT DIGIT

INEOF:  ;WHEN THE END OF FILE IS REACHED ON THE INPUT FILE
        ;THE GETCHR SUBROUTINE WILL TRANSFER CONTROL TO HERE.

;NOW  THE VALUE IN SUM MUST BE CONVERTED TO AN ASCII STRING
;OF OCTAL DIGITS TO BE OUTPUT TO THE TTY.

        MOVE    PNT,OUTPNT              ;LOAD PNT WITH A BYTE POINTER
                                        ;INTO THE AREA THE RESULT IS TO
                                        ;BE STORED INTO.
        MOVSI   COUNT,-14               ;MAXIMUM OF 12 DIGIT RESULT
                                        ;(ASSUMING NO OVERFLOW)
        MOVE    A1,SUM                  ;THE OCTAL DIGITS CAN BE
                                        ;OBTAINED BY SIMPLY SHIFTING THE
                                        ;SUM 3 BITS AT A TIME INTO
                                        ;REGISTER A.
LOOP2:  SETZM   A                       ;INITIALIZE A
        LSHC    A,3                     ;MOVE LEFT 3 BITS OF A1 INTO A
        CAMN    PNT,OUTPNT              ;IF POINTER HAS CHANGED, SKIP
```

```
                                            I OVER TEST FOR LEADING ZERO
        JUMPE      AiLEND2                  lir LEADING HERO JUST INCREMENT
                                            (COUNTER BUT DON'T OUTPUT,
        ADOI       A,6®                     I MAKE INTO ASCtI CHAR
        IDPB       A i PNT                  (PUT CHAR INTO TTV OUTPUT BUFFER
        AOBJN      COUNT,L00P2              J IF THERE ARE MORE DIGITS LEFT,
                                            IGO GET THEM TOO
        MOVEI      A|0
        IDPB       A,PNT                    ISTORE AN ASCII NULL AT END OF
                                            •STRING
        OUTSTR     OUTMSG                   (THIS SPECIAL PROGRAMMED
        OUTSTR     OUtWRO                   I OPERATOR OUTPUTS AN ASCII
                                            (STRING TO A TTYtSTRING IS
                                            (TERMINATED BY A NULL)
EXITTI  CALL        CSIXBIT /EXIT/3         (SPECIAL FUNCTION TO GRACE*
                                            (FULLY TERMINATE THE EXECUTION
                                            (OF A PROGRAM,

1TME FOLLOWING SUBROUTINE IS USED TO INPUT ONE ASCII CHAR

GETCHRI 3                                   (RETURN ADDRESS IS STORED HERE
GETNXTI SOSLE      IBUF*2                   (OECREMENT THE BYTE COUNT
        JRST       GETOK                    (NON-ZERO RESULT MEANS MORE
                                            (CHARS LEFT IN BUFFER
        IN         INCHM,                   (GET NEXT BUFFER PROM MONITOR
        JRST       GETOK                    (RETURN WHEN BUFFER IS FULL
        STATZ      INCHN,740000             JIN OOFS A SKIP RETURN IF THERE
                                            (WAS AN ERROR ON INPUT. THE
                                            (STATUS BITS MUST BE TESTED TO
                                            JDETERMINE WHAT KJMO OF ERROR,
        JRST       INERR                    (NOT END"OF"FILE, GO PROCESS THE
                                            I ERROR
        JRST       INEOF                    (END OF FILE RETURN TO NEXT
                                            IPHASE OF PROGRAM,

GETOKI  ILDB       DIGIT,!BUP*1             (GET CHAR FROM BUFFER
        JUMPN      DIGIT,»GETCHR            (IF NOT NULL CHAP, RETURN TO
                                            (CALLING PLACE WHOSE ADDRESS IS
                                            (STORED IN GETCHR,
        JRST       GETNXT                   (IGNORE NULL AND GET NEXT CHAR,


(NEXT COME SOME ERROR ROUTINES WHICH TYPE OUT ERROR MESSAGES
jTO EXPLAIN ERRORS RECEIVED BY THE PROGRAM,

INERR!  OUTSTR     INPMSG                   (OUTPUT MESSAGE AND
        JRST       EXITT                    (EXIT FROM PROGRAM

INPMSGI ASCIZ      /ERROR WHILE READING INPUT FILE/


NOTAVLI OUTSTR     AVLMSG
        JRST       EXITT

AVLMSGI ASCIH      /DEVICE NOT AVAILABLE/


NOTFNDI OUTSTR     FIL^SG
        JRST       EXITT

FILMSGt ASCI*      /FILE WAS NOT FOUND/
```

```
;-----------------------------------

;NOW TO DEFINE SOME CONSTANTS AND DATA

IBUF:   BLOCK   3               ;THIS IS THE INPUT BUFFER HEADER
INNAME: SIXBIT  /DATA/          ;NAME OF DATA FILE
        SIXBIT  /FIL/           ;EXTENSION OF DATA FILE
        0                       ;
        0                       ;IF THIS IS LEFT "0" THE OWNER
                                ;OF THE FILE IS ASSUMED TO BE
                                ;THE USER RUNNING THIS PROGRAM.
                                ;THIS NUMBER CAN BE OBTAINED BY
                                ;RUNNING THE "PPN" CUSP.

OUTPNT: POINT   7,OUTWRD        ;POINTER TO OUTWRD WHERE THE
                                ;ASCII REPRESENTATION OF THE SUM
                                ;OF THE DIGITS IS TO BE STORED.
OUTMSG: ASCII   /***THE SUM OF THE DIGITS IS  /
OUTWRD: BLOCK 4

        END     START
```

The following is an example of a terminal session in which a data file for the ADDER.MAC example program is created, and the example program (assumed to exist on dsk) is assembled, loaded, and executed.

```
.LOG
JOB 17 CMU10A 6.U10/DEC 5S02.C/D TTY40
#Z799ID00                             ;Your usage number goes here.
PASSWORD:                             ;Type your password here. It will not be echoed.
2141     17-JUL-72                    ;The system will respond with a greet message.
MON 7-17...ALL STRUCTURES IN SYSTEM...SYS:NEWS (7-7)


                                      ;To run the ADDER program which we assume is on disk
                                      ;from a previous secession, the data file must fir
                                      ;first be created.
.CREATE DATE.FIL
00100   1 2 3,4,,5,6 7 8,9,E
00200   $
*E

EXIT                                  ;Now that the data file has been created, we can
                                      ;execute the ADDER program.
.
                                      ;We can assemble,load,and execute ADDER in
                                      ;three seperate steps, or we can simply use the
                                      ;EXECUTE command to do all three.

.EXCCUTE ADDER.MAC
MACRO: ADDER                          ;This statement indicates that the MACRO 10
                                      ;assembler is now assembling.
LOADING                               ;The loader is now loading the relocatable file
                                      ;produced by the assembler.
LOADER 1K CORE
EXECUTION                             ;Begin execution.
FILE WAS NOT FOUND                    ;This message is coming from the ADDER program.
EXIT                                  ;It says there is no file called DATA.FIL. If
                                      ;we look back we see a spelling error in the
                                      ;CREATE.


                                      ;We can correct this error by using the RENAME
                                      ;command to change the name of the data file.

.RENAME DATA.FIL=DATE.FIL
FILES RENAMED:
DATE.FIL        05
```

```
.EX                                       ;Try executing ADDER again.
LOADING                                   ;Since the relocatable file already exists, the
                                          ;assembly step has been skipped.
LOADER 1K CORE
EXECUTION
***THE SUM OF THE DIGITS IS  35           ;If we look back to the data file, the sum of the
EXIT                                      ;octal digits should be 34(we ignore the 8,9,and E).
                                          ;But notice, there is an octal non-zero digit in
                                          ;the line number—the line number was included as
                                          ;part of the data string.


.R PIP                                    ;We can use "PIP" to remove the line numbers
                                          ;from the file.
*DSK:DATA.FIL/N←DSK:DATA.FIL
                                          ;This command simply causes the file to be
*↑C                                       ;rewritten without the line numbers.

.

.EX                                       ;Again execute the program.
LOADING

LOADER 1K CORE
EXECUTION
***THE SUM OF THE DIGITS IS  34           ;The sum is now correct.
EXIT
                                          ;To get off the system it's necessary to execute
                                          ;the "KJOB" command which returns all I/O devices
                                          ;to the system device pool. In addition, if there
                                          ;are any files on our disk area, the monitor
                                          ;responds with "CONFIRM:" to which we have
                                          ;several options, described on page 2-17 of the
.KJOB                                     ;Timesharing Handbook.
CONFIRM: U
                                          ;"U" says list all unprotected files so they
                                          ;can either be protected or deleted.

DSKA:
DSKB:
DATA  .FIL  <055>  5. BLKS  : P           ;Here we've said protect the data file ,


ADDER .REL  <055>  5. BLKS  : K           ;but delete the relocatable ADDER file.

                                          ;The monitor responds with some file statistics
                                          ;and accounting information.
JOB 2, USER 27991D00     LOGGED OFF TTY1     1420   20-AUG-71
DELETED 1 FILES (5. DISK BLOCKS)
SAVED 7 FILES (85. DISK BLOCKS)
RUNTIME 0 MIN, 03.75 SEC        KILOCORE SEC: 23
CONNECT TIME 0 HR, 4 MIN, 37 SEC      TOTAL CHARGE: $0.33
```

**MLISP**

**M. Rychener**

**The following is from the MLISP Manual by D.C. Smith (Stanford AIM-135, October, 1970).**

Most programming languages are designed w|th the Idea that the syntax should be structured to produce efficient code for the computer. Fortran and Algol are outstanding examples, Yet, It Is apparent that HUMANS spend more t|me with any given program than the COMPUTER. Therefore, It has been our Intention to construct a language which Is as transparently c|ear and understandable to a HUMAN BEING as possible, Considerable effort has been spent to make the syntax concise and uncluttered. It reduces the number of parentheses required by LISP, Introduces a more mnemonic and natural notation, clarifies the flow of control and permits comments. Some "meta-express Ions" are added to Improve the I I st-processIng power of LISP. Strings and string manipulation features, particularly useful for Input/output, are Included, In addition, a substantial amount of redundancy has been built Into the language, permitting the programmer to choose the most natural way of writing routines from a varlety of poss|b|I I ties.

LISP Is a I Ist-processIng and symbo|-man IpuI at I on language created at MIT by John McCarthy and h|s students (McCarthy, 1965), The outstanding features of LISP are: (1) the simplest and most elegant syntax of any language In existence, (2) high-level symbol manipulation capabilities, (3) an efficient set of list-processing primitives* and (4) an easily-usable power of recursion. Furthermore, LISP automatically handles all Internal storage management, freeing the user to concentrate on problem solving, This Is the single most Important Improvement over the other major list-processing language, IPL-V, LISP has found applications **In** many Important artificial InteI I Igenee I nvest i gat|ons, IncIudIng symboI Ic mathrratlcs, natura I - I anguage handling, theorem proving and logic.

Unfortunately, there are several Important weaknesses In LISP. Anyone who has attempted to understand a LISP program written by another programmer (or even by himself a month earlier) quickly becomes aware of several difficulties:

A, The flow of control Is very difficult to follow. In fact, It is about as difficult to follow as machine language or Fortran, This makes understanding the purpose of routines (**J**.e, what do they do?) difficult. Since comments are not usually Permitted, the programmer Is unable to provide written assistance.

B, An Inordinate amount of time must be spent balancing parentheses, whether In writing a LISP program or trying to understand one, It Is frequently difficult to determine where one expression ends and another begins, Formatting utility routines pretty-pr |nt") he|p; but every LISP programmer knows the dubious pleasure of laboriously matching left and right parentheses In a function, when all he knows Is that one is missing somewhere!!

C, The notation of LISP (prefix notation for functions, parentheses around all functions and arguments, etc.), while uniform from a logician's point of view, is far from the most natural or mnemonic for a language. This clumsy notation also makes it difficult to understand LISP programs. Since MLISP programs are translated into LISP s-expressions, all of the elegance of LISP is preserved at the translated level; but the unpleasant aspects at the surface level are eliminated.

D. There are important omissions in the list-processing capabilities of LISP. These are somewhat remedied by the MLISP "meta-expressions", expressions which have no direct LISP correspondence but instead are translated into sequences of LISP instructions. The MLISP meta-expressions are the FOR expression, WHILE expression, UNTIL expression, index expression, assignment expression, and vector operations. The particular deficiency each of these attempts to overcome is discussed in the subsection of SECTION 3 describing the meta-expression in detail.

MLISP was written at Stanford University by Horace Enea for the IBM 360/67 (Enea, 1968). The present author has implemented MLISP on the PDP-10 time-shared computer. He has rewritten the translator, expanded and simplified the syntax, and improved the run-time routines. All of the changes and additions are intended either to make the language more readable and understandable or to make it more powerful.

MLISP programs are first translated into LISP programs, and then these are passed to the LISP interpreter or compiler. As its name implies, MLISP is a "meta-LISP" language; MLISP programs may be viewed as a superstructure over the underlying LISP processor. All of the underlying LISP functions are available to MLISP programs, in addition to several powerful MLISP run-time routines. The purpose of having such a superstructure is to improve the readability and writeability of LISP, long (in)famous for its obscurity. Since LISP is one of the most elegant and powerful symbol-manipulation languages (but not one of the most readable), it seems appropriate to try to facilitate the use of it.

MLISP has been running for several years on the Stanford PDP-10 time-shared computer. It has been distributed to the DEC User Services Group (DECUS). The MLISP translator and run-time routines are themselves compiled LISP programs. The Stanford version runs under the Stanford LISP 1.6 system (Quam, 1969). Some effort has been made to keep the translator as machine independent as possible; in theory MLISP could be implemented on any machine with a working LISP system by making only minor changes. The one probable exception to this is the MLISP scanner; to enable scanning (where most of the time is spent) to be as efficient as possible, the translator uses machine language scanning routines. While these routines have greatly increased translation speed (MLISP now translates at a rate of 3000-5000 lines per minute.), their use means that someone wishing to implement MLISP on a system without LISP 1.6 will have to use an equivalent scanner package. For this reason, a whole section of this manual (SECTION 7) is devoted to presenting an equivalent scanner.

While LISP was created with the goal of being machine independent, it has turned out that most LISP systems have unique features. The situation is so difficult that Anthony Hearn has attempted to define "a uniform subset of LISP 1.5 capable of assembly under a wide range of existing compilers and interpreters," called STANDARD LISP (Hearn, 1969). MLISP helps to alleviate this situation by introducing another level of machine independence: to implement MLISP on a given LISP system, one changes the underlying translator rather than the surface syntax. Dr. Hearn has also constructed an MLISP-like language called REDUCE (HEARN, 1970).

For sample exercises, see the Lisp section of this document. There also is a program available on SYS:, written in MLisp, called MEXPR. In that program, the function convert takes a lisp source-filename, reads the file, and writes an MLisp equivalent of the file. A slightly augmented version of MEXPR is available from M. Rychener, SCH 4211.

```
.
.
.  : MLISP SCRIPT
.
.TY ICSCR.MLI
   BEGIN
   EXPR MLISTN();  % READ - EVAL MLISP EXPRS %
     WHILE T DO BEGIN
       TERPRI(NIL); PRINC(":");
       PRINT EVAL MTRANS();
       END;
     END.

.R MLISP

*(MLISP (ICSCR.MLI))


*
MLISTN
*
0. SECONDS TRANSLATION TIME
0. ERRORS WERE DETECTED
0. FUNCTIONS WERE REDEFINED



***-END-OF-RUN-***
*(MLISTN)

:* % NOW WE'RE TALKING TO MTRANS, WHICH TRANSLATES AN MLISP EXPRESSEION
*   INTO LISP. THIS LISP IS TAKEN BY THE ROUTINE MLISTN ABOVE AND
*    EVAL'D, THEN PRINT'D, MUCH LIKE THE TOP LEVEL OF LIST\T\P %
*EXPR FACT(N); IF ZEROP N THEN 1 ELSE N*A\A\FACT(N-1);

FACT
NIL
:*FACT(3);

6.
:*FACT(7);

5040.
:* % COMPARE THAT DEFINITION OF FACT TO  THE FOLLOWING LISP EQUIVALENT%
* EVAL '(GRINDEF FACT) ;
```

```
(DEFPROP FACT
 (LAMBDA (1M) (COND ((ZEROP N) 1.) (T (TIMES N (FACT (SUB1 N))))))
EXPR)

NIL
•i* % NOW TRY AN ITERATIVE VERSION %
*EXPR FACT2(N); BEGIN NEW Mi M_l. I DO BEGIN M_M*N;

FACT2 * N_N-i; END UNTIL ZEROP N; RETURN M END;

NIL

:*FACT2(3);                    '

6.

:*FACT2(7);

3040.

:*FACT2 4;

24.
:* %PARENTH ER\ R\ S E S ARE NOT NECESSARY AROUND UNARY FUNCTIONS %
*EVL\L\AL '(GRINDEF FACT2)
(DEFPROP FACT2
 (LAMBOA(N)
  (PROG (M)
        (SETQ M1.)
        (ADO (QUOTE PROG2)
             (QUOTE
              (PROG NIL (SETQ M (TIMES M N)) (SETQ N (SUB1 N))))
             (QUOTE (ZEROP N)>)
        (RETURN M)))          '

EXPR)

nIL
»* % ADO IS AN MLISP FUNCTION TO PERFORM THE INDE\E\ICATED• ACTION.
*    WITHIN THE MLISP INTERPRETER %

* % THE ABOVE CALLS TO GRINDEF ILLUSTA\A\RATE HOW TO SPEAK IN LISP
*    TO THE FUNCTION MTRAN'S, IN CASE IT DOESN'T. LIKE WHAT YOU'VE TRIED
*    TO TYPE IN MLISP. FOR INSTANCE *\*\s %
•GRINDEF(FACT);
```

```
*** ERROR IN TOP-LEVEL
*** ILLEGAL SYMBOL BEGINNING A SIMPLE EXPRESSION
*** CURRENT SYMBOL IS )
*** SKIPPING TO NEXT SEMICOLON

*** ERROR IN TOP-LEVEL
*** ILLEGAL ARGUMENT
*** CURRENT SYMBOL IS ;
*** SKIPPING TO NEXT SEMICOLON


(DEFPROP (FACT NIL)
 (NIL)
VALUE)

NIL
:* % SOMEHOW, MTRANS DOS\S\ESN'T LIKE TO SEE GRINDEF %
*CDR '(G\G(\GRINDEF ;

(FSUBR #6164 PNAME (#50763 #50764))
:* % PERHAPS BECAUSE IT'S AN FSUBR % EVAL '(GRINDEF FACT) ; %DOES WORK,
HOWEVER %


(DEFPROP FACT
 (LAMBDA (N) (COND ((ZEROP N) 1.) (T (TIMES N (FACT (SUB1 N))))))
EXPR)

NIL
:* % MLISP DOESN'T REQUIRE PARENTHESE FOR BINARY FUNCTIONS EITHER :%
*'X MEMBER '(A B C D X Y Z) ;

T
:*'(A B C) CONS '( D E F) ;

((A B C) D E F)
:*'(A B C) @ '(D E F) ;

(A B C D E F)
:* % @ IS USED FOR APPEND %  <A,,B\B,,A<\       <A,B,C> @ <D,E,F>;

A
UNBOUND VARIABLE - EVAL
BACKTRACE
MAPLIST-? LIST-*EVAL PRINT-EVALARGS PROG-*EVAL &WHILE-*EVAL ?-*EVAL

*(MLISTN)
```

```
:* %LISP ERRORS TAKE US BACK TO LISP READ-EVAL LLOOP
*    THE ERROR WAS OMMITTING QUOTES ABOVE % <'A,'B> @ <'C,'D>;

(A B C D)
:* %ANGE\E\LE BRACKETS DENOTE THE LIST FUNCTION IN MLISP %
* MTRANS();
* <'A,'B>;

(LIST (QUOTE A) (QUOTE B))
:*EXPR  HAVE(X); X;

HAVE
NIL
:*EXPR WII\I\TH(X,Y); ^U
FEXPR WITH(X,Y); REU\U\TURN 'OK!! ;

WITH
NIL
:*HAVE FUN WITH MLISP;

82835.
:*FEXPR WIH\H\TH(X,Y); 'OK ;

WITH    *** WARNING, FUNCTION REDEFINED

NIL
:*HAVE FUN WITH MLISP! ;

OK
:*FEXPR WIH\H\TH(X,Y); 'OK!! ;

WITH    *** WARNING, FUNCTION REDEFINED

NIL
:*HAVE UN\WU\^U
^U
HAVE FUN WITH MLISP!! ;

OK!!
:*^C
^C
```

PIP - Peripheral Interface Program

B. Anderson

PIP is a basic systems program of the PDP-10 which provides the user with the necessary facilities for handling existing data files. Actions possible, among others, are transferring files from one standard I/O device to another standard I/O device, listing and deleting directories, simple editing, changing protection codes, and controlling magnetic tape functions. The following script shows the typical uses and efficient methods for handling such uses.

REFERENCES

[1] PDP-10 Reference Handbook, pp. 585-596.

```
•AS DTA
DTA2 ASSIGNED                          v                    *
iAS DTA              AJtt:        w\\        c w y v   4c.MxceS          i
DTA4 ASSIGNED                                            **0*\     \><-       ^ ^ ^ 1
•PLEASE MOUNT BA03BC ON DTA2 ENABLED AND BA03DF ON DTA4 ENABLED
OPERATOR HAS BEEN NOTIFIED
BA03BC IS ON DTA2 AND BA03JDF IS ON DTA4 BOTH ENABLED
THANK YOU
tC

•R PIP
```

```
                    J COPYING FILES

                    ;UNDERLINING DENOTES SYSTEM TYPEOUT
                    ;SEMI COLONS DENOTE COMMENTS


*DSK:A.EXT-DSK:B.EXT                        ;A COPY OF B.EXT CALLED A.EXT
                                            ;IS MADE ON DISK.  ANY FORMER
                                            ;CONTENTS OF A*EXT ARE LOST

*DTA2:A.EXT-DSK:B.EXTCQ502BA033             JB.EXT FROM O502BA03'S DISK
                                            JAREA IS COPIED ONTO THE USER'S
                                            ;DISK AREA WITH THE NAME A.EXT,
                                            ;PROVIDED B.EXT IS NOT READ
                                            ;PROTECTED

*LPT;-DSK:B.EXT                             ;LPT IS A NON-DIRECTORY DEVICE
                                            ;AND SO A FILENAME IS NOT
                                            ;REQUIRED.  IF ONE IS GIVEN, IT
                                            ;iS IGNORED

*DTA2:A.EXT-DSK:B.EXT, CEXT                 ;B.EXT AND C EXT FROM DISK APE
                                            ;COPIED ONTO DTA2 IN THE ORDER
                                            >SPECIFIED AND COMBINED IN THE
                                            J FILE A.EXT

*DTA2: A. EXT«-DTA4: * .MAC                 ;ALL FILES WITH MAC EXTENSIONS
                                            ;ON DTA4 ARE COMBINED IN A.EXT
                                            ;ON DTA2

*DTA2: A.EXT*-DTA4: FILE.*                  , ALL FILES WITH THE FILENAME
                                            ;FILE, REGARDLESS OF EXTENSION,
                                            ;ARE COMBINED IN A.EXT ON DTA2

*jDTA2: A. EXT«-DTA4:* .*                   ;ALL FILES ON DTA4 APE COPIED
                                            ;iMTO A.EXT ON DTA2

*DSK: FILE1-DTA2: A.EXT, DTA4: FILE.MAC     ;A.EXT FROM DTA2 AND
                                            ;FILE.MAC FROM DTA4 ARE COPIED
                                            ; INTO FILE1 ON DI SK
```

```
                ;END OF FILE ON TTY IS DENOTED BY ↑Z (CONTROL Z)

*DSK:A.EXT←TTY:                              ;THE TEXT OF THE TTY FILE IS
THE TEXT OF THE FILE GOES HERE               ;COPIED INTO A.EXT ON DISK
AND HERE
↑Z

                ;COPYING SPECIFIED FILES WITHOUT COMBINING THEM.
                ;ORDINARILY ONLY ONE DESTINATION FILE IS PERMITTED BY
                ;PIP. THE X SWITCH ALLOWS FILES TO BE COPIED AS THEY
                ;ARE, KEEPING THEIR NAMES AND INDIVIDUAL FILE STATUS

*DSK:/X←DTA2:A.EXT                           ;A.EXT IS COPIED TO DISK WITH
                                             ;THE SAME NAME

*DSK:/X←DTA2:A.EXT,DTA4:FILE.MAC             ;A.EXT AND FILE.MAC ARE EACH
                                             ;COPIED TO DISK WITH THE SAME
                                             ;NAMES

*DTA2:(DX)←DSK:A.EXT,B.EXT                   ;(DX) DENOTES TO COPY ALL FILES
                                             ;EXCEPT THOSE SPECIFIED.  ALL
                                             ;FILES EXCEPT A.EXT AND B.EXT
                                             ;ARE COPIED TO DTA2

                ;DELETING FILES

*DSK:/D←DSK:FILE.MAC                         ;FILE.MAC IS DELETED FROM THE
FILES DELETED:                               ;DISK.  PIP TELLS YOU SO
FILE.MAC        05

*DTA2:/X←DSK:FILE.MAC                        ;FILE.MAC HAS BEEN DELETED.  PIP
? NO FILE NAMED FILE.MAC                     ;TELLS YOU IT IS NOT THERE
```

```
                    ;RENAMING FILES

*DSK:FILE2/R←DSK:B.EXT                    ;B.EXT IS RENAMED AS FILE2.
FILES RENAMED:                            ;IF /R WERE LEFT OUT, ANOTHER
B.EXT    Ø5                               ;COPY OF B.EXT WOULD BE MADE
                                          ;CALLED  FILE2

*DSK:A.EXT/R←DSK:B.EXT                    ;B.EXT WAS RENAMED ABOVE AND SO
? NO FILE NAMED B.EXT                     ;NO LONGER EXISTS UNDER THAT
                                          ;NAME

*DSK:B.*/R←DSK:A.*                        ;ALL FILES WITH THE FILENAME A
FILES RENAMED:                            ;ARE RENAMED WITH FILENAME B
A.EXT    Ø5                               ;AND THE SAME EXTENSIONS


                    ;CHANGING FILE PROTECTIONS

*DSK:/R<155>←DSK:B.EXT                    ;B.EXT'S PROTECTION IS MADE 155.
FILES RENAMED:                            ;DEFAULT PROTECTION IS Ø55
B.EXT    Ø5

*DSK:A.EXT<155>←DSK:B.EXT                 ;B.EXT IS COPIED INTO A.EXT
                                          ;WITH THE PROTECTION 155.  B.EXT
                                          ;KEEPS ITS OLD PROTECTION

*DSK:*.*<155>/R←DSK:*.*                   ;ALL FILES ARE RENAMED TO THEIR
FILES RENAMED:                            ;SAME NAMES, BUT THEIR
FILE2    Ø5                               ;PROTECTIONS ARE CHANGED TO 155
C.EXT    Ø5
B.EXT    Ø5
FILE1
A.EXT    Ø5

*DSK:*.*<155>/R←DSK:*.EXT                 ;ALL FILES WITH EXT EXTENSIONS
FILES RENAMED:                            ;GET THE PROTECTION CODE 155
C.EXT    Ø5
B.EXT    Ø5
A.EXT    Ø5
```

;ZEROING A DECTAPE DIRECTORY

```
*DTA2:/Z←                                ;DTA2'S DIRECTORY IS ZEROED OUT

*DTA2:A.EXT/Z←DSK:B.EXT                  ;FIRST THE DIRECTORY IS ZEROED
                                         ;AND THEN B.EXT IS COPIED INTO
                                         ;A.EXT
```

;GETTING A DIRECTORY LISTING

```
*TTY:/L←DSK:*.*                                            ;A DIRECTORY OF YOUR DISK
DIRECTORY            Q502BA03   14:36    25-AUG-71   ;AREA PRINTS ON THE TTY.
                                                           ;THIS IS EQUIVALENT TO
DSKB:                                                      ;THE NEXT EXAMPLE
FILE2                      05   <155>    25-AUG-71
C         EXT              05   <155>    25-AUG-71
B         EXT              05   <155>    25-AUG-71
FILE1                      05   <155>    25-AUG-71
A         EXT              05   <155>    25-AUG-71

TOTAL BLOCKS               25
DSKA:


*TTY:/L←
DIRECTORY            Q502BA03   14:37    25-AUG-71

DSKB:
FILE2                      05   <155>    25-AUG-71
C         EXT              05   <155>    25-AUG-71
B         EXT              05   <155>    25-AUG-71
FILE1                      05   <155>    25-AUG-71
A         EXT              05   <155>    25-AUG-71

TOTAL BLOCKS     25
DSKA:


*LPT:/L←DTA2:*.*                         ;A DIRECTORY OF ALL FILES ON
                                         ;DTA2 PRINTS ON THE LINE PRINTER

*TTY:/L/F←                               ;A SHORT DIRECTORY, LISTING ONLY
                                         ;FILENAMES, PRINTS ON THE TTY
DSKB:
FILE2
C         EXT
B         EXT
FILE1
A         EXT
DSKA:
```

```
                ;INSERTING OR ELIMINATING SEQUENCE NUMBERS

*DSK:/X/S←DSK:A.EXT                     ;RESEQUENCES OR ADDS SEQUENCE
                                        ;NUMBERS, INCREMENTED BY 10,
                                        ;TO A.EXT

*DSK:/X/N←DSK:A.EXT                     ;ANY SEQUENCE NUMBERS IN A.EXT
                                        ;ARE DELETED

*LPT:/N←DSK:A.EXT                       ;COPY THE FILE ON THE
                                        ;LPT WITHOUT SEQUENCE NUMBERS


                ;LESS FREQUENTLY USED SWITCHES, INCLUDING MAGTAPE
                ;CONTROL SWITCHES, CAN BE FOUND IN THE REFERENCE
                ;HANDBOOK, PAGES 6-9 TO 6-23.
                ;AFTER FINISHING WITH YOUR DECTAPE, ALWAYS HAVE THE
                ;OPERATOR DISMOUNT THE TAPE.  THEN DEASSIGN THE UNIT

.PLEASE DISMOUNT BAØ3BC AND BAØ3DF FROM DTA'S 2 AND 4
OPERATOR HAS BEEN NOTIFIED
TAPES DISMOUNTED
↑C

.DEAS DTA2                              ;WAIT UNTIL THE TAPES ARE
                                        ;DISMOUNTED DEFORE DEASSIGNING
.DEAS DTA4                              ;THE UNITS

.
```

PPL

S. Gerhart

PPL (Polymorphic Programming Language) was developed by Tim Standish, formerly of CMU and now at Harvard. PPL is a conversational, extensible language, in many respects like APL. Conversational features include line and character editing of functions, trace and suspension, and I/O to teletypes. Also, functions may be written onto files and edited by TECO or SOS.

PPL is a typeless language with extensibility for operator and data definitions. Built-in types includes integer, real, double precision, Booleau, and string, with the usual operators for atomic data. Data definitions for structures, variadic sequences, fixed sequences, and alternates may be given, each having association construction, predicate, and selection operations. New operators are defined by associating user-defined functions with strings. Other features are Iversonian precedence, structure sharing, and both call by reference and call by value.

PPL is not supported here but is fairly stable. A good users' manual is available in the Computer Science Department Library. PPL is recommended for programs which require variability of data structures, structured data representation, and conversation.

Script



```
.R PPL
PPL.26 31-JAN-71
        READ("PROTO")

        WRITE()


        BINARY("&",CATAND)
        UNARY("@",RETURN)

        $LIST = [1: ] GENERAL


        $CATAND(A,B)
    [1]  AND((A==LIST),B==LIST)-->CAT.OP
    [2]  -->@CATAND←AND(A,B)
    [3]  CAT.OP: CATAND←CONCAT(A,B)
         $


        $RETURN($A)
    [1]  RETURN←0
         $


        $APPEND(A,L)
    [1]  NOT(L==LIST)-->ERROR   ...   == IS THE "INSTANCE OF" OPERATOR
    [2]  -->@APPEND←L&LIST(A)
    [3]  ERROR: PRINT("APPEND TRIED ON NONLIST")
    [4]  ?
         $


        $EXPLANATION
    [1]  ...@ IS A DEFINED OPERATOR WHICH COMPUTES AND EXPRESSION
    [2]  ...THEN RETURNS FROM A FUNCTION(BRANCH TO 0 IS AN EXIT).
    [3]
    [4]  ...& IS THE BUILT-IN SYMBOL FOR THE "AND" OPERATOR
    [5]  ...HERE, & IS REDEFINED TO HAVE THE MEANING CATENATION WHEN
    [6]  ...ITS OPERANDS ARE BOTH LISTS.
    [7]
    [8]  ... A LIST IS DEFINED AS A VARIADIC SEQUENCE WITH ELEMENTS OF
    [9]  ...ANY TYPE IN THE SYSTEM.
         $

        X←LIST(1,2,3)
        Y←LIST(6,5)
        X&Y
[1,2,3,6,5]
        APPEND(X,Y)
[6,5,[1,2,3]]
        APPEND(X,LIST())
[[1,2,3]]
        X[2]←Y
        X
[1,[6,5],3]
        X==LIST
TRUE
        X[1]==LIST
FALSE
```

SAIL

J. Nugent

INTRODUCTION

SAIL is a high-level programming system for the PDP-10 computer, developed at the Stanford AI Project to be the major language for the hand-eye robot project. It includes an extended Algol compiler and a companion set of execution-time routines. A non-standard Algol 60 compiler is extended to provide facilities for describing manipulations of an associative data structure. This structure contains information about items, stored as unordered collections of items (sets) or as ordered triples of items (associations). The algebraic capabilities of the language are linked to the associative capabilities by menas of the datum operator, which can associate an algebraic datum with any item.

The associative data structure is a slightly reworked version of the LEAP language, which was designed by J. Feldman and P. Rovner, and implemented on Lincoln Laboratory's TX-2. This language is described in some detail in an article entitled "An Algol-Based Associative Language" in the August, 1969, issue of the ACM Communications (Feldman and Rovner). The implementation was modified to tolerate the non-paging environment of the PDP-10.

SAIL in a sense has something for everyone. For those who think in Algol, SAIL has Algol. For those who want the most from the PDP-10 and the time-sharing system, SAIL allows flexible linking to hand-coded machine language programs, as well as inclusion of machine language instructions in SAIL source programs. For those who have complex input/output requirements, the language provides complete access to the I/O facilities of the PDP-10 system. For those who aspire to speed, SAIL generates fairly good code.

The user should, however, be warned that SAIL falls several man-decades short of the extensive testing and optimization efforts contained in the histories of most commercial compilers.

COMPILER OPERATION

SAIL accepts commands in the same format as other DEC processors, i.e.,

<Binary>, <listing> ← <source 1>, <source 2>, . . .

where <Binary>, <listing>, <source 1>, etc., are of the form

<Device>: <file name>. <extension> [<PPN>].

If <Device> is omitted, the last device specified will be used. If none has been given, DSK will be used.

If <device> is not a disectory device, it is the only specification necessary.

If <extension> is omitted, the following will be assumed:

.REL for binary

.LST for listing

.CRF for CREF listing

.SAI for source file

(See DEC reference manual for explanations of CREF.)

If [<PPN>] is omitted, the user's PPN will be used.

Switches, if given, should follow the listing file name. See section 14 of the SAIL manual for a description of valid switches.

For example,

```
.R SAIL
* MYPROG ← MYPROG
```

would compile the program MYPROG.SAI and place the output file MYPROG.REL on the user's disk space.

The following:

```
* MYPROG, MYPROG ← MYPROG.NEW [A7ØØHUØØ]
```

would compile the program MYPROG.NEW on HUØØ's disk area, again generating output MYPROG.REL, but also creating a listing of the program in MYPROG.LST. Also:

```
*DTA2:  MYPROG, MTAØ:  /C ← PTR:
```

would compile a program read in from paper tape, place output file MYPROG.REL on DTA2 (dectape), a CREF listing on MTAØ (magtape).

The SAIL compiler can be invoked in the same ways as FORTRAN or MACRO. The Default extension for SAIL SOURCE PROGRAMS is .SAI.

The COMPILE, EXECUTE, LOAD, or DEBUG commands may be used. For example:

```
.EX PRGRAM.SAI
.DEB PRGRAM            (where the extension is the default for SAIL)
.EX PROG1, SUB1, SUB2  (where SUB1 and SUB2 are separately compiled
                        procedures)
```

For details on these commands, see the PDP-10 Reference Manual.

If a CREF listing is to be generated, AICREF must be used instead of CREF, i.e.,

```
.R AICREF
* _ _ _ _ _ _ _        (commands are the same as for DEC's CREF.)
```

To load a SAIL program, use AILOAD, as above. The correct DDT to use is (what else?) AIDDT.

If you use DEBUG, EXECUTE, LOAD, etc., they will do the above things correctly automatically upon seeing the .SAI extension.

NOTE:

Since SAIL is a very fast (one pass) compiler, it is generally a good idea to delete .REL files after using them. This will save space and avoid possible confusion in the effects of the load, debug and execute commands*

## REFERENCES


[1] Swinehart, D. and R. Sproull, SAIL Manual, CMU version of May, 1970, available from Computer Science Department.

[2] Most recent CMU manual update, available from Computer Science Department.

[3] Erman, L., SAIL Pocket Guide (Sailing Chart), available from Computer Science Department.

[4] Feldman, J. and F. Rovnar, "An Algol-Based Associative Language," CACM, 12(8), August, 1969, pp. 439-449.

EXERCISES

1. Write a SAIL program to merge two SOS files, according to sequence numbers.

2. You are given an M x N matrix of numbers where M and N can be very large. The values of the entries are 0 - 15. In order to conserve DISK space, it is desirable to pack the data (each number can be represented in 4 bits) nine entries to a PDP-10 word before writing the matrix onto a DISK file. Write a SAIL program which does this packing, writes out the file, reads it in, and "unpacks" it.

## SOME SIMPLE PROGRAMMING EXAMPLES

[1]

```
BEGIN "FACTORIAL"
COMMENT THIS PROGRAM READS NUMBERS FROM THE TELETYPE AND
        TYPES BACK THEIR FACTORIALS;
DEFINE !="COMMENT";                   ! COMMENT IS TOO LONG;
DEFINE CR="'15",LF="'12";             ! ASCII FOR CR AND LF;
INTEGER PROCEDURE FACT(INTEGER N);
      BEGIN "FACT"
      INTEGER I;
      I←1;                     ! INITIAL VALUE FOR THE LOOP;
      FOR N←N STEP -1 UNTIL 1 DO
      I←I*N;                   ! NOTE THAT FOR N=0, I WILL BE 1;
      RETURN(I);
      END   "FACT";
INTEGER X;
WHILE TRUE DO
      BEGIN "INFINITE LOOP"
      ! WHEN FINISHED WITH THE PROGRAM, TYPE C TO BREAK OUT;
      OUTSTR(CR&LF&"NUMBER, PLEASE:");
      X←CVD(INCHWL);           ! READ THE NUMBER;
      OUTSTR(IF X<0 THEN "NOW REALLY" ELSE CVS(FACT(X)));
      END  "INFINITE LOOP";
END "FACTORIAL";
```

[2]

```
      BEGIN "FIXER"
    COMMENT THIS PROGRAM READS A FILE AND REPLACES ALL OCCURRENCES
            OF OLDCHR WITH NEWCHR. THIS IS ESPECIALLY USEFUL FOR
            FIXING UP FILES ORIGINALLY DESTINED FOR THE LPT,
            WHICH CONTAIN SPECIAL PRINTER CONTROL CHARACTERS
            INSTEAD OF REGULAR LINE FEED CHARACTERS (SUCH
            CHARACTERS CAUSE SPECIAL PRINTER ACTION, BUT ARE IGNORED
            BY A TELETYPE, MAKING IT IMPOSSIBLE TO PRINT THEM ON
            A TELETYPE;
     DEFINE OLDCHR="'23",NEWCHR="'12";
     DEFINE !="COMMENT",NOTE="COMMENT";
     LABEL ST;
     STRING S,S2,S3,S4;
     INTEGER EEOF,BRK,DSKIN,DSKOUT,EEEOF;
     OUTSTR("INPUT FILE");                           ! TYPE PROMPT MESSAGE;
     S4←INCHWL;                                      ! READ INPUT FILE NAME;
     DSKIN←GETCHAN;                                  ! CHANNEL FOR INPUT;
     OPEN(DSKIN,"DSK",0,4,4,400,BRK,EEOF);           ! OPEN DSK ON CHANNEL;
     LOOKUP(DSKIN,S4,EEOF);                          ! LOOK UP THE  FILE;
     IF EEOF THEN USERERR(0,0,"FILE NOT FOUND");     ! IF EEOF IT FAILED;
     OUTSTR("OUTPUT FILE");                          ! DITTO FOR OUTPUT;
     S4←INCHWL;
     DSKOUT←GETCHAN;
     OPEN(DSKOUT,"DSK",0,4,4,400,BRK,EEEOF);
     ENTER(DSKOUT,S4,EEEOF);
     IF EEEOF THEN USERERR(0,0,"CANNOT ENTER FILE!!");
     BREAKSET(1,OLDCHR,"IS");                        ! INPUT  BREAK ON
    OLDCHR;
     WHILE NOT EEOF DO
          BEGIN "READ FILE"
          NOTE = THIS LOOP WILL CONTINUE UNTIL END OF FILE IS REACHED;
          S←INPUT(DSKIN,1);
          OUT(DSKOUT,S&                              ! INPUT ENDED ON EITHER;
          (IF BRK=OLDCHR THEN NEWCHR ELSE BRK));     ! OLDCHR OR 400 CHARS;
          END   "READ FILE";
      RELEASE( DSKOUT); RELEASE (DSKIN);             ! RELEASE I/O DEVICES;
      END "FIXER";                                   ! AND CLOSE FILES;
```

SNOBOL4

Script: S. Schlesinger

SNOBOL4 is a computer language, developed at Bell Telephone Laboratories, which contains many features not commonly found in other programming languages. The basic data element is the string. The language has operations for joining and separating strings, testing their contents, and making replacements within them. Strings can be broken down and reassembled differently. Also, examination of a string for a desired structure of characters, an operation called pattern matching, is possible and most powerful. Because SNOBOL4 is mainly character oriented, the numerical capabilities with both integers and reals exist, but are limited. Array variables also exist.

Execution of SNOBOL4 is interpretive. This allows easy tracing of variable values, and the ability to redefine functions during execution. The language can be extended by using data type definition facilities and defining operations on these through function definition (i.e., lists, complex numbers).

REFERENCES

[1] Griswold, R. E., J. F. Poage, and I. P. Polonsky, The SNOBOL4 Programming Language, Prentice Hall, 1968.

[2] Modified Chapter 8 of above, for local PDP-10, I.O. conventions, available from Computer Science Department.

[3] SNOBOL.DOC, a printable text file on the PDP-10.

CMU PDP-10 I/O Notes - SNOBOL

SNOBOL4 I/O is similar to FORTRAN I/O as described in Griswold, et al.[1] The following list is the current device assignments as used for input and output.

The SNOBOL 10 list of device numbers:

| UNIT | DEVICE |
|---|---|
| 1 | DSK |
| 2 | TTY |
| 3 | PTR |
| 4 | PTP |
| 5 | DSK1Ø - Program input file. |
| 6 | DSK11 - .LST file |
| 7 | CDP |
| 8 | CDR |
| 9 | LPT |
| 1Ø | DTAØ |
| 11 | DTA1 |
| 12 | DTA2 |
| 13 | DTA3 |
| 14 | DTA4 |
| 15 | DTA5 |
| 16 | DTA6 |
| 17 | DTA7 |
| 18 | PLT |
| 19 | FORTR |
| 2Ø | DSKØ |
| 21 | DSK1 |
| 22 | DSK2 |
| 23 | DSK3 |
| 24 | DSK4 |
| 25 | DSK5 |
| 26 | DSK6 |
| 27 | DSK7 |
| 28 | DSK8 |
| 29 | DSK9 |
| 3Ø | MTAØ |
| 31 | MTA1 |
| 32 | MTA2 |
| 33 | MTA3 |
| 34 | MTA4 |
| 35 | MTA5 |
| 36 | MTA6 |
| 37 | MTA7 |
| 99 | TTCALL |

To perform input and output from within a SNOBOL4 program, variables are associated with devices or file names. If a variable is associated in an output relation with a device or file then each time the variable is assigned a value, a copy of the value is written to the device or file. Similarly each time an input variable is used, a new value is read from the associated device or file to become the value of the variable.

The function

OUTPUT (variable name, unit number, format)

[e.g. OUTPUT ('DONE', 23, '(1X,20 A5)')]

associates the variable DONE with unit 23 which is a disk device. Output data will be written in the indicated FORTRAN IV format. Unit 23 may be associated with a particular file by coding the function.

OFILE (unit number, file name)

Input associations are similarly accomplished using

INPUT (variable name, unit number, length)

IFILE (unit number, file name)

where length is the number of characters to be read into the input variable each time it is referenced. Files may be closed using ENDFILE (unit number).

Other I/O functions and an extended discussion of those named here appears in reference [2]. Examples of these functions appear in the following script.

There does exist a SNOBOL4 system which permits saving of SNOBOL programs and variables during execution in order to restart them at a later date. Documentation on this version of SNOBOL may be obtained from the system file SNOBLX.DOC.

SAMPLE PROBLEMS

Write SNOBOL programs to do the following:

1. Read and print cards, removing all blanks before printing.
2. Read cards and print those beginning with '/'.
3. Read cards and print those not containing '*'.
4. Reverse the order of characters in a string.
5. Count all the vowels in the input text.
6. Read left-justified text; print it centered on the line.
7. Alphabetize the characters of a string.
8. Count the occurrences of pronouns in English text.
9. Read a deck. For each card, if a vowel appears in the first five columns, print the card as it was read. If not, and if '$' or '*' appears between columns 60 and 70, reverse the card, prefix two slashes, and print the result.
10. Read numbers in free form (e.g., separated by commas). Every time you have read ten numbers, print them in columnar format. Assume that no number is more than ten characters long.
11. Devise a simple cipher (e.g., letter substitution). Write programs to encode and decode messages using this cipher. Generalize to accept a description of the cipher as an input. How complex can you make the cipher?

SNOBOL Script

```
.MAKE REV.SNO
```
– Creates file REV.SNO and calls TECO ready for editing.

```
*I      DEFINE('REVERSE(X)A')                 :(REVEND)
REVERSE X LEN(1) . A =                        :F(RETURN)
        REVERSE = $$REVERSE A                 :(REVERSE)
REVEND
        DATA = TRIM(INPUT)                    :((F(END)
        OUTPUT = DATA ' REVERSED IS ' REVERSE(DATA)
                                              :(REVEND)
END
$HT$$
        DEFINE('REVERSE(X)A')                 :(REVEND)
REVERSE X LEN(1) . A =                        :F(RETURN)
        REVERSE = REVERSE A                   :(REVERSE)
REVEND
        DATA = TRIM(INPUT)                    :F(END)
        OUTPUT = DATA ' REVERSED IS ' REVERSE(DATA)
                                              :(REVEND)
END
*IABCDEFG
1234567890
$EX$$

EXIT
```

REVERSE should reverse its input parameter string

Note data is at the end of program file.

```
.R SNOBOL 41
*REV
*TTY:=REV
```

*REV — This creates a file REV.LST containing output of SNOBOL Processor & program output.

*TTY:=REV — This causes new execution with REV.LST listed on TTY, as follows:

```
SNOBOL4 (VERSION 3.4.3, JAN. 16, 1971)

DIGITAL EQUIPMENT CORP., PDP-10

1               DEFINE('REVERSE(X)A')              :(REVEND)
2       REVERSE X LEN(1) . A =                     :F(RETURN)
3               REVERSE = REVERSE A                :(REVERSE)
4       REVEND
5               DATA = TRIM(INPUT)                 :F(END)
6               OUTPUT = DATA ' REVERSED IS ' REVERSE(DATA)
7                                                  :(REVEND)
8       END

NO ERRORS DETECTED IN SOURCE PROGRAM
```

```
ABCDEFQ REVERSED IS ABCDEFO
1234567890 REVERSED IS 1234567890
```

90J- r°v av " ^ ?

TV. >f«'sC ^ v 1.

```
NORMAL TERMINATION AT LEVEL  0
LAST STATEMENT EXECUTED VAS    5



SN0B0L4 STATISTICS SUMMARY-

             700 MS* COMPILATION TIME

             417 MS. EXECUTION TIME

              44 STATEMENTS EXECUTED*        3 FAILED

               0 ARITHMETIC OPERATtO
*tC
•TECO REV.SNO

•9LS- SIA SS AS-8D0TTSS
        REVERSE - A REVERSE                 t(REVERSE)
•OLStSRI  SOL-TTSS
RUVERSE X LEN(l) • A -                      «F( RETURN)
        REVERSE - A REVERSE                   t(REVERSE)
• SH-8D0L-TTSS
REVERSE X LEtf<l> • A •                     • F( RETURN )
        REVERSE • A REVERSE                   (REVERSE)
•SCSRItSOTTSS
        REVERSE - A REVERSE                  t(REVERSE)
*TSS
(REVERSE)
•R-DOL-TTSS
REVERSE X LEN(1> • A •                      t F(RETURN)
                                            • (REVERSE)
        REVERSE - A REVERSE
*EXSS
EXIT
```

```
.R SNOBOL 41
*REV
```

— Create full REV.LST as above.

```
*TTY:-REV/U
```

— /U turns off processor listing of program text.

```
NO ERRORS DETECTED IN SOURCE PROGRAM




ABCDEFG REVERSED IS GFEDCBA
1234567890 REVERSED IS 0987654321




NORMAL TERMINATION AT LEVEL  0
LAST STATEMENT EXECUTED WAS    5
```

REV.LST using /U switch

statistics omitted (by editors)

```
.MAKE REV.DAT
```

Create an input data file

```
*I ABCDEFGHJKLMN
0987654321
$EX$$
```

```
SNOBOL4 (VERSION 3.4.3, JAN. 16, 1971)

DIGITAL EQUIPMENT CORP., PDP-10

                  DEFINE('REVERSE(X)A')                   :(REVEND)
         REVERSE X LEN(1) . A =                           :F(RETURN)
                  REVERSE = A REVERSE                     :(REVERSE)
         REVEND
                  INPUT('FILE',20,72)
                  IFILE(20,'REV.DAT')
                  DATA = TRIM(FILE)                       :F(END)
                  OUTPUT = DATA ' REVERSED IS ' REVERSE(DATA)
                                                          :(REVEND)
         END

NO ERRORS DETECTED IN SOURCE PROGRAM
```

Using INPUT and IFILE, attempting to get input from disk file.

```
ABCDEFGHJKLMN REVERSED IS NMLKJHGFEDCBA
ABCDEFGHJKLMN REVERSED IS NMLKJHGFEDCBA
ABCDEFGHJKLMN REVERSED IS NMLKJHGFEDCBA
ABCDEFGHJKLMN REVERSED IS NMLKJHGFEDCBA
ABCDEFGHJKLMN REVERSED IS NMLKJHGFEDCBA
ABCDEFGHJKLMN REVERSED IS NMLKJHGFEDCBA
ABCDEFGHJKLMN REVERSED IS NMLKJHGFEDCBA
ABCDEFGHJKLMN REVERSED IS NMLKJHGFEDCBA
ABCDEFGHJKLMN REVERSED IS NMLKJHGFEDCBA
ABCDEFGHJKLMN REVERSED IS NM↑O
```

– Note error in intended execution as the file is reopened during every loop – thus good idea to put all IFILE, INPUT, OFILE, and OUTPUT specifications at head of program – as follows

```
↑C
↑C
```

```
         INPUT('FILE',20,72)
         IFILE(20,'REV.DAT')
         DEFINE('REVERSE(X)A')              :(REVEND)
REVERSE  X LEN(1) . A =                     :F(RETURN)
         REVERSE = A REVERSE                :(REVERSE)
REVEND
         DATA = TRIM(FILE)                  :F(END)
         OUTPUT = DATA ' REVERSED IS ' REVERSE(DATA)
                                            :(REVEND)
END
*EX$$

EXIT

.RE\E\ SNOBOL 41
*TTY:=REV/U

NO ERRORS DETECTED IN SOURCE PROGRAM
```

```
 ABCDEFGHJKLMN REVERSED IS NMLKJHGFEDCBA
0987654321 REVERSED IS 1234567890
```
] – Program output, as desired.

```
1               OUTPUT! «OUTFILE*»ei» »<1X#&7A5)•)
8               OFILEfgU »REV.OUT*>  "
3               INPUT!*FILE*#80*78>
4               IPILE<80#*REV*DAT*)
S               DEFINE!•R£VERSE!X)A'>              t!REVEND)
6       REVERSE X LENC1) . A -                     IFCRETURN)
7               REVERSE - A REVERSE                t<REVERSE)
8       REVEND
9               DATA- TRIM!FILE)                   1F<END)
10              (OUTFILE^)" DATA * REVERSED IS • REVERSE!DATA)
11                                                 I<REVEND)
12      END

NO ERRORS DETECTED IN SOURCE PROGRAM


NORMAL TERMINATION AT LEVEL  0
LQST STATEMENT EXECUTED VAS   9


tO
*tC

p»TY REV.OUT

^.TECO REV*SINO                                                    AAv<

• SF< END)S4R3DEFILESOTTSS
        DATA - TRIMCFILE)                  •FCLE)
*0L8S<S1EFISOTTSS
        DATA - TRIMCFILE)                  •F<EFILE)
•3LIEFILE        ENDFILEC21)
SEXSS
7N0 FILE FOR OUTPUT                                  <x9\**      A^^;,.
•» in
```

```
        OUTPUT('OUTFILE',21,'(1X,27A5)')
        OFILE(21,'REV.OUT')
        INPUT('FILE',20,72)
        IFILE(20,'REV.DAT')
        DEFINE('REVERSE(X)A')              :(REVEND)
REVERSE X LEN(1) . A =                     :F(RETURN)
        REVERSE = A REVERSE                :(REVERSE)
REVEND
        DATA = TRIM(FILE)                  :F(EFILE)
        OUTFILE = DATA ' REVERSED IS ' REVERSE(DATA)
                                           :(REVEND)
EFILE   ENDFILE(21)
END
*EXS$

EXIT

.R SNOBOL 37
*REV/U
*:C

.

 TY REV.OUT
 ABCDEFGHIJKLMN REVERSED IS NMLKJIHGFEDCBA
0987654321 REVERSED IS 1234567890

.TY REV.LST

NO ERRORS DETECTED IN SOURCE PROGRAM
```

Note: OFILE will create REV.OUT.

— and REV.OUT has program output as intended

Note REV.LST also exists as before, minus program output.

```
        OUTPUT('TTYOUT',2,'(1X,14A5)')
        INPUT('TTYIN',2,72)
        DEFINE('REVERSE(X)A')            :(REVEND)
REVERSE X LEN(1) . A =                   :F(RETURN)
        REVERSE = A REVERSE              :(REVERSE)
REVEND
        TTYOUT = 'ENTER DATA: '
        DATA = TRIM(TTYIN)               :F(END)
        TTYOUT = DATA ' REVERSED IS ' REVERSE(DATA)
                                         :(REVEND)
END
*EX$$

EXIT

.



 R SNOBOL 37
*REV1
ENTER DATA:
ASDFG HJKL
ASDFG HJKL REVERSED IS LKJH GFDSA
ENTER DATA:
/.,MN BVC DE3 678
/.,MN BVC DE3 678 REVERSED IS 876 3ED CVB NM,./
ENTER DATA:

 REVERSED IS
ENTER DATA:
↑C

.TY REV1.LST
? NO FILE NAMED REV1.LST
 ..
```

Have no end character - thus mu[st] terminate with ↑C.

This as REV was termin[ated] by control C (↑C).

The above script hopefully conveyed the essential ideas of the SNOBOL system and I.O

# SOS Primer

Joseph M. Newcomer

## Introduction

This document is merely intended as an introduction to the SOS editor. For further explanations and a more complete set of commands, consult the SOS manual.

SOS is a teletype-oriented text editor written by Bill Weiher and Stephen Savitzky of the Stanford Artificial Intelligence Laboratory. In addition to the common editing capabilities of inserting, deleting, and shifting of lines of text, SOS includes string search and substitute commands, an intra-line edit capability, text-justifying features, and a few other assorted bells and whistles.

SOS does not edit a file "in place", as some editors do. Changes are made on a temporary copy of the file, and ordinarily are made permanent only upon completion of the edit. However, you may request at any time that all changes up to that point be made permanent. This is an especially recommended practice for beginners, as it insures all changes made in the file since the EDIT command or the last save request against loss due to system failure or user inexperience.

SOS is oriented towards full-duplex devices, such as the teletype, the ARDS display, the Infoton display, and other such devices. Before attempting to use it from a half-duplex device such as an IBM 2741 terminal or a Datel terminal, you should become thoroughly familiar with using it from the teletype or similar full-duplex device. You must then familiarize yourself with the conventions for using half-duplex devices on the PDP-10 as implemented here at C-MU. In general, it is not worthwhile for the novice to learn how to use SOS from half-duplex devices, since the effort involved in using them does not really make up for the 50% faster typeout.

Basic commands

The basic operation in a file-oriented system is the creation of a file. To invoke the editor and request it to create a file, give the CREATE command when the console is in monitor mode i.e., the computer has typed a period.

In all examples, the computer output is underscored.

Example _ Creating a file:

```
.CREATE BLAT.DOC
00100  THIS IS AN EXAMPLE OF HOW TO CREETE
00200  A FILE USING THE EDITOR.
00300  IN ORDER TO GET OUT OF NUMBERING MODE, TYPE
00400  AN ALTMODE (ESCAPE( CHARACTER, WHICH ECHOES
00500  AS A DOLLAR SIGN.
00600  «
```

When the asterisk is typed, you may enter any editor commands you want. The E command (End) terminates the edit, saves the file, and returns to the monitor.

Example 2 Terminating an edit:

```
*E
EXIT
```

The file now exists and you may access it in any of the normal modes in which files are accessed. For example, you may type it:

Example 3 Typing a file:

```
.TYPE BLAT.DOC
00100  THIS IS AN EXAMPLE OF HOW TO CREETE
00200  A FILE USING THE EDITOR.
00300  IN ORDER TO GET OUT OF NUMBERING MODE, TYPE
00400  AN ALTMODE (ESCAPEt CHARACTER. WHICH ECHOES
00500  AS A DOLLAR SIGN.
```

If upon examining the typeout, you find there are some errors (as in the typeout above) you may invoke the editor with the EDIT command to make the corrections. The set of commands for simple editing is:

I - Insert

D - Delete

R - Replace

P - Print

L - List

The Replace command is used to replace lines of the file. In its simplest form it is the single letter R followed by the line number to be replaced. The editor then types the line number out and new text may be typed in. This new line replaces the previous contents of the line.

The Delete command is used to delete lines from the file. In its simplest form it is the single letter D followed by the line number of the line to be deleted. The editor deletes the line and returns control with the asterisk. There is normally no other typeout. To delete a group of contiguous lines, a range may be specified; see "Specifying Ranges", below.

The Insert command is used to insert new lines in a file. Its basic format is the letter I followed by the line number of the line to be inserted.

<u>Example</u> <u>4</u> Simple editing

```
.EDIT BLAT.DOC
*R100
00100  THIS IS AN EXAMPLE OF HOW TO CREATE
*D400
*I400
00400  AN ALTMODE (ESCAPE) CHARACTER, WHICH ECHOES
*
```

Note that the Replace command has the same effect as a Delete command followed by an Insert command. In order to use Insert to replace a line, the line must first be deleted. The Insert command by itself does not replace the line specified if it already exists, as in some editing systems, but instead creates a new line whose number is equal to the line given plus the line increment (normally 100). The Insert command will always insert a new line in a file, never replace an old one. If the line following the specified line has a line number less than or equal to the computed insertion line number, then the insertion is given a number which is halfway between the line specified and the next line.

Example 5 Interpolated insertion

```
*I200
00250  SINCE THE INCREMENT IS 100, THIS LINE IS HALFWAY
*I250
00275  BETWEEN TWO LINES, AS THIS LINE ALSO IS.
*
```

In order to see what your file now looks like, you can use the Print command to print it on the teletype. The Print command is the letter P followed by the line number of the line to be printed. The letter P by itself will print the current line and 15 following lines. To specify a range of lines, a colon may be used to indicate a beginning and ending line number specification; see "Specifying Ranges", below, for more details on this.

Example 6 Printing part of a file

```
*P100:500
00100  THIS IS AN EXAMPLE OF HOW TO CREATE
00200  A FILE USING THE EDITOR.
00250  SINCE THE INCREMENT IS 100, THIS LINE IS HALFWAY
00275  BETWEEN TWO LINES, AS THIS LINE ALSO IS.
00300  IN ORDER TO GET OUT OF NUMBERING MODE, TYPE
00400  AN ALTMODE (ESCAPE) CHARACTER, WHICH ECHOES
00500  AS A DOLLAR SIGN.
```

In addition to the P command, two keys on the teletype will also cause printing. A linefeed (in this text, ↓), will print the next line, and an altmode (escape, shown as a "$") will print the previous line.

Example 7 Linefeed and Altmode commands

```
*P300
00300  IN ORDER TO GET OUT OF NUMBERING MODE, TYPE
*↓
 00400  AN ALTMODE (ESCAPE) CHARACTER, WHICH ECHOES
*$00300  IN ORDER TO GET OUT OF NUMBERING MODE, TYPE
*
```

If there is too much information to conveniently type on the teletype, the L (List) command may be used to output the lines on the printer. Its format is precisely the same as the P command, except that if just "L" is specified the entire file is listed. Note that the file may not come out immediately on the printer, as print files are queued waiting for the printer to become available. Consequently, your file may not be printed for some time after the L command completes. You may continue editing the file, however, since the information is copied into a temporary buffer and held until printed. The file name on the listing printed will be of the form "nnn.LPT", where "nnn" is a number assigned by the monitor, and "LPT" indicates a print buffer file. You should not then be looking for a listing with the file name printed on the front.

Example 8 Listing a file

```
*L
*
```

This has printed the entire file on the line printer.

Specifying Ranges

Whenever you wish to specify more than a single line, you may specify a range. This is done by using a colon to separate the two line numbers (where the second must be higher than the first). Thus 100:600 specifies lines 100 to 600. Most commands accept a range of lines to be operated upon, and this is one way of giving that range. However, in some cases it is easier or more appropriate to specify a quantity of lines (5 lines, 17 lines, etc.) regardless of the line number of the last line. This is indicated by using an exclamation point (!) to specify the range: 100!3 is line 100 and the following two lines (so "100!1" is the same as "100").

Example 9 The exclamation point

```
*P100!4
00100  THIS IS AN EXAMPLE OF HOW TO CREATE
00200  A FILE USING THE EDITOR.
00250  SINCE THE INCREMENT IS 100, THIS LINE IS HALFWAY
00275  BETWEEN TWO LINES, AS THIS LINE ALSO IS.
*D250!2
*P100!4
00100  THIS IS AN EXAMPLE OF HOW TO CREATE
00200  A FILE USING THE EDITOR.
00300  IN ORDER TO GET OUT OF NUMBERING MODE, TYPE
00400  AN ALTMODE (ESCAPE) CHARACTER, WHICH ECHOES
*
```

Intermediate commands

The intermediate editing commands are:

C - Copy

T - Transfer

N - Number

W - save World

M - Mark page

G - Go

The Copy command copies lines from one place in the file to another. The first location specified is the "destination" line number. The second location (which may be a range) is the "source" location. The editor will choose an increment which will allow all the specified lines to be copied to the destination without overflowing; this increment is printed out in the message "INC1=nnnnn". If the editor cannot compute an increment such that all lines will fit, then an error message will be typed and appropriate action will be taken by the editor (see the SOS manual, page 28).

The Copy command can also copy from another file, so that portions of program files can be extracted to form a new file. Again for details, consult the SOS manual (page 28).

The Transfer command is much the same as the Copy command, except that the lines which are copied into the specified destination in the file are then deleted from the source location.

Note: In the SOS manual it states that Copy and Transfer behave as Insert, i.e., "C200,500" would copy line 500 to somewhere after line 200 (the exact number depending on the line number following line 200). This is not true! The Copy (or Transfer) command will copy line 500 and put it after line 200, but will also number it 200, giving two line 200's. To get out of this problem, use the N command to renumber the file. The extra line 200 will be numbered correctly.

Example 10 Copy & Transfer commands

```
.CREATE COPY.DOC
00100  THIS IS
00200  A SHORT
00300  FILE
00400  $
*C150,300
INC1=00050
```

```
*P100:300
00100   THIS IS
00150   FILE
00200   A SHORT
00300   FILE
*T350,100
*P100:400
00150   FILE
00200   A SHORT
00300   FILE
00350   THIS IS
*
```

The Number command is used to renumber files. This is usually done after a number of insertions have been made and no more room exists between line numbers for further insertions. The simplest form of the Number command is simply the letter N, which renumbers the entire file with an increment of 100. For more information on the Number command, see the SOS manual, page 13.

Example 11 The Number command

```
*P100:400
00150   FILE
00200   A SHORT
00300   FILE
00350   THIS IS
*N
*P100:99999
00100   FILE
00200   A SHORT
00300   FILE
00400   THIS IS
```

The W command is particularly useful to the beginner. The W command makes permanent all changes made in the file up to the time it is given. Changes made in a file are temporary until either a W or an E command is given. There are two reasons you should do a W command often: 1) The system could crash, and all editing done would be lost when it came back up, or 2) you might attempt using some new command (say, "substitute", a somewhat tricky one), and confuse your file to the point where you cannot recover the text you started with. In either case, the loss will be back to the last "EDIT" command to the monitor, or the last W command to the editor. By giving permanence to those changes whose accuracy you are certain of, you will avoid losing time in re-creation of those changes, or perhaps the entire file.

## Pages

Files can be divided into logical subunits termed "pages". A page in the SOS editor is merely a collection of lines. It may be less than one physical printer page, or it may be several physical printer pages. When we need to make a distinction, we will call the SOS pages "logical pages" and the printer pages "physical pages". We will use the term "page" ordinarily to mean a logical page. To indicate the separation into logical pages, a "page mark" is inserted into the file by the Mark page command. The Mark page command places a page mark immediately before the line number specified. Each page is numbered separately, and hence you may have several line 100*s in a file. In order to specify what page you are on, use the slash (/) in the line number specification, with the page number following the slash. Line 100 on page 1 is then designated as "100/1". To minimize the amount of typing required, the editor remembers what the current page is, and subsequent commands need only specify the line number on the current page.

Example 12 Multipage file

```
*P 100:400
00100   FILE
00200   A SHORT
00300   FILE
00400   THIS IS
*M300
*P 100/1:400
00100   FILE
00200   A SHORT
*P100/2:400
00300   FILE
00400   THIS IS
*N
*P 100/1:400/2
00100   FILE
00200   A SHORT

PAGE 2

00100   FILE
00200   THIS IS
```

When listed on the line printer with an L command, each page has the page number printed in the upper left. The form of this page number is the logical page number followed by a hyphen followed by the physical page number (recall that logical pages can be longer than physical pages). The physical page number is reset for each logical page, so that the numbers proceed as "1-1, 1-2, ... , 1-n, 2-1, 2-2, When using a listing as a guide to editing, remember that the first number is the page number that SOS uses, e.g. when correcting page 4-15 specify "/4" for the page number.

There are two other special characters which you can use to designate lines in the file. The period (.) is used to designate either the current line or the current page, depending on where it is used. If it is used in the line position, it is the current line; if in the page position it is the current page. If page 2 is the current page, and line 100 is the current line, then "./2" is "100/2", "./1" is "100/1", "200/." is "200/2" and of course "./." is the current line, 100/2. The asterisk is always the last line on the page indicated. If the current line is 100/2 in the file of example 12, then "*" is "200/2" and "*/1" is "200/1". If the line number is omitted but a page number is given, it means the entire page, e.g., "P/2" is the same as "P0/2:*/2". For more details on specifying ranges, see the SOS manual, page 7.

Example 13 Period and asterisk designators

```
*P100/1:*
00100   FILE
00200   A SHORT
*I*/2
00300   NEW LINE
00400   $
*P/2
00100   FILE
00200   THIS IS
00300   NEW LINE
*P*/1:*/2
00200   A SHORT

PAGE 2

00100   FILE
00200   THIS IS
00300   NEW LINE
*P/1:/2
00100   FILE
00200   A SHORT

PAGE 2

00100   FILE
00200   THIS IS
00300   NEW LINE
*P.
00300   NEW LINE
*P100/1
00100   FILE
*I.
00150   INSERTION
*I.
00175   ANOTHER
*P.:*
00175   ANOTHER
00200   A SHORT
*P/1
00100   FILE
00150   INSERTION
00175   ANOTHER
00200   A SHORT
```

*

The Go command is equivalent to the End command in that it terminates the edit; however, it also causes the last COMPILE, EXECUTE, LOAD, or DEBUG monitor command to be re-executed. This is a great convenience when debugging programs.

Example 14 The Go command

```
.CREATE TEST.ALG
          .
          .
          .
*E

EXIT

.COM TEST
ALGOL: TEST
200  INCORRECT STATEMENT
REL FILE DELETED
300  UNDECLARED IDENT↑C↑C
.ED
*P200
00200           INTEGRE I, J, K;
*R.
00200           INTEGER I, J, K;
*G

ALGOL: TEST

EXIT

.
```

Advanced commands

The advanced editing commands are:

A - Alter

J - Join

S - Substitute

F - Find

B - Beginning

The Alter command is one of the most useful features of the SOS editor. It allows editing individual lines much as the normal edit commands are used to edit files. You can alter a single letter in a line, i.e., change it, delete it, or even insert it. The full capabilities of the Alter command are explained in the SOS manual, page 14 ff; some examples will be given here.

Edit commands in intraline edit mode are not echoed by the teletype. We will indicate this in examples by showing the edit commands in lower case. One exception to this will be the altmode character, which will still be a dollar sign. Remember that in intraline edit mode it will not echo. The following notation will be used: "␣" will be a space, "■" will be a rubout, "⤶" will be a carriage return, and ↑U will be control-U (the control key and U key simultaneously).

The set of intraline edit commands is:

␣ - Accept the character under the pointer

■ - Backspaces the pointer

C - Change the character under the pointer

D - Delete the character pointed to

I - Insert new characters (terminated by altmode)

⤶ - Terminate intraline edit

Q - Quit intraline edit without making changes

↑U - Start over

S - Skip

K - Kill

R - Replace

L - Print remaining line and continue edit

P - Print remaining line and resume edit

For explanations of the commands, see the SOS manual, pp 15-17. With this as a guide, you may follow the examples below. In these examples, a ⤸ is a non-echoed carriage return; a ■ is a non-echoed rubout, and a ␣ is a non-echoed space.

Example 15 Intraline skip and insert

```
*P/1
00100   FILE
00150   INSERTION
00175   ANOTHER
00200   A SHORT
*A150
00150   seINSi***$⤸ERTION
*P.
00150   INS***ERTION
*
```

Example 16 Intraline delete and kill

```
*P150
00150   INS***ERTION
*A.
00150   ssINd\\S⤸\\***ERTION
*P.
00150   IN***ERTION
*A.
00150   s*INkr\\**E⤸\\RTION
*P.
00150   INRTION
*
```

You may precede a command with a number which causes it to be repeated, e.g. "2sa" is equivalent to "sa" followed by another "sa".

Example 17 Intraline skip and change

£1150
00175 THIS IS A (SMAPLE( LINE
*A.
00175 2s(THIS IS A (SMAPLEc)J LINE
*P.
00175 THIS IS A (SMAPLE) LINE

Example 18 Intraline accept and rubout
*P175
00175 THIS IS A (SMAPLE) LINE
#A.

00175 3ssTHIS IS A (2^SMffll\\M2c\\AMiPLE) LINE

00175 THIS IS A (SAMPLE) LINE

One of the most common errors made in using the Alter command is failure to type the altmode terminating an Insert within the line. This has the effect of terminating the line being edited and beginning a new line. Although a sometimes desired effect, such as in indenting Algol program files, it is more often just an error. Should you type a >} after an insertion, and get a new line number instead of the rest of the line, just type the altmode and i again. You now have two lines where you had one before, and the Join command can undo this. To use the Join command, type J followed by the original line number.

Example 19 The Join command

*P175
00175 THIS IS A (SAMPLE) LINE
*A.
00175 s)THIS IS A (SAMPLE^)i OF A)
00187 8J LINE
*P175!2
00175 THIS IS A (SAMPLE) OF A
00187 LINE
*J175
*P175
00175 THIS IS A (SAMPLE) OF A LINE

The Find command may be used to locate known strings in a file when their line numbers are not known, or to check a file for occurrences of strings. The basic format of the Find command is the letter F, followed by a string to be searched for, followed by a altmode, followed by a range specification. Again, more details may be found in the SOS manual, pp 23-25. When a string is located, the line containing it is typed out and search is suspended. To resume the search with the same string, only an F followed by an altmode is required.

Example 20 The Find command

```
.EDIT SOME.BLI
*FLOCAL$/1
*
```

(There were no occurrences of "LOCAL" on page 1)

```
*F$/2
00150          LOCAL A, B, C;
*F$
00300          LOCAL AARGH BLAT[5];
*F$.+1:/99

PAGE 6

00400          MEASURES LOCALIZED PHENOMENA SUCH AS
*F$
*
```

If you give further Find commands without specifying a range, no more strings will be found, since the current line position is the end of the file. To reset the file position, you could either specify the first line of the file as the lower bound of search, e.g., "Fstring$100/1:/999", which is clumsy, or, more simply, you could use the Beginning command to reposition the file.

If you are not interested in stopping at each line where the string is found, you can give a parameter to the Find command which tells how many occurrences to print and bypass before stopping. To find all occurrences in a file, use some large number such as 999 or 99999.

Example 21 The Begin and Find commands

Assume the file is in the state it was left in at the end of example 20.

```
*F$,
*B
*F$,999

PAGE 2

00150          LOCAL A, B, C;
00300          LOCAL AARGH BLAT[5];
```

PAGE 6

```
00400              MEASURES LOCALIZED PHENOMENA SUCH AS
*
```

The Substitute command is similar to the Find command, in the sense that a string is searched for; in addition, a second string is substituted for the one found. The format of the Substitute command is the letter S followed by the string to be searched for, followed by an altmode, followed by a string to replace it, followed by another altmode, followed by a range. For more details, see the SOS manual, pp 25-27.

Example 22 The Substitute command

Assume the file is in the state it was left in at the end of example 21.

```
*B
*SLOCAL$OWN$

PAGE 2

00150              OWN A, B, C;
00300              OWN AARGH BLAT[5];

PAGE 6

00400              MEASURES OWNIZED PHENOMENA SUCH AS
*
```

As you see, the string substitution also replaced the occurrence of "LOCAL" in line 400/6. This is one of the most common errors made with the Substitute command. In this example the Substitute command or the Alter command may be used to correct the problem; in another example it may be neither simple or even possible to undo a bad substitution. For this reason, we recommend giving a W command before doing a Substitute. If the Substitute command then destroys part of the file, abort the edit without making the changes permanent by typing ↑C (control-C), and typing EDIT again. Since you are editing the same file, the file name need not be given.

Example 23 Aborting an edit

Assume the file is in the condition it was in at the end of example 22.

```
*↑C

.EDIT
TEMPORARY EDIT FILE ALREADY EXISTS! DELETE? (Y OR N)
←Y
*P400/6
```

```
00400          MEASURES LOCALIZED PHENOMENA SUCH AS
*P150/2
00150          LOCAL A, B, C;
*
```

The message about the temporary edit file may not be typed if the editor was left in a state where the temporary file did not exist.

## Miscellany

In addition to the commands discussed here, there are several others of marginal interest. One of the most useful of these is the command, which types out information contained in the editor. Its format is followed by the name of the internal parameter to be displayed. The command is discussed more fully on pp 20-21 of the SOS manual. The most useful parameters to display are the current line (.), the number of pages in the file (BIG) and the current line increment (INC).

Along with the "=" command there is the complementary "set" command which is a left arrow (<-). This is used to change the values of the internal parameters. This is discussed on pp 19-20 of the manual. The most useful parameter to set is the line increment (INC).

Example 24 The - and *- commands

```
.EDIT  HUGE.BLI
*=BIG
62
*P100/41
00100              INCR I FROM 1 TO .N DO

100/41
*I.,25
00125  BEGIN A<-5; X<-.Y<3,2>;
00150   $
*=INC
00025
*<-INC=5
*I.
00130                  BLATO;THUD(.Q);
00135  END;
00140   $
*=
00135/41
```

## Removing line numbers

In some cases it is necessary to remove the line numbers which SOS places in the file. To do this, you may use PIP with the "/N" switch, as shown in the example below.

Example 25 Removing line numbers

```
.R PIP

*BLAT.DOC/N<-BLAT.DOC

*TC
```

Using terminals with both upper and lower case

Some terminals are available with both upper case and lower case letters, notably the ARDS display and the Western Union 300 terminals. The PDP-10 monitor, however, always translates lower case input into upper case unless instructed otherwise. SOS also assumes the terminal has only upper case letters unless instructed to the contrary (except for the ARDS display, which SOS knows has lower case). The example below shows the commands necessary to use such terminals.

Example 26 Using a terminal with lower case

```
.TTY LC
.edit garble.doc
*←m37
*p100
00100  This document describes the GARBLE system of
*
```

Note that when using the WU300 terminals, the "all caps" switch must be turned off, or the terminal will convert lower case letters to upper case letters before transmitting.

When in intraline edit mode, a "skip" or "kill" command will interpret its argument in the exact case it was typed in. Thus in the last example, a skip to "r" from the beginning of the line will stop in "describes", while a skip to "R" will go (from the beginning of the line) directly to the R in "GARBLE".

Using terminals with only upper case

Most terminals available are Teletype model 33 terminals, which have only upper case letters. Occasionally it is necessary to create or edit a file containing both upper case and lower case letters on one of these terminals. SOS allows the case of the input character to be shifted by preceding it with a question mark (?). In normal mode, for example, "A" represents "A", and "?A" represents "a". By changing the mode, "A" will represent "a" and "?A" will represent "A". This is shown in the example below.

Example 27 Lower case from a teletype

```
.EDIT GARBLE.DOC
*P100
00100  T?H?I?S ?D?O?C?U?M?E?N?T ?D?E?S?C?R?I?B?E?S ?T?H?E GARBLE
  ?S?Y?S?T?E?M ?O?F
*←LOWER
*P100
00100  ?THIS DOCUMENT DESCRIBES THE ?G?A?R?B?L?E SYSTEM OF
*
```

## LIST OF EXAMPLES
(See index for page numbers)

# INDEX

TECO - Text Editor and Corrector

Script: T. Teitelbaum

TECO edits files recorded in ASCII characters on any standard device. It can perform simple editing functions as well as sophisticated search, match, and substitute operations, and operate upon arbitrary length character strings under control of commands which are themselves character strings (and can exploit this recursiveness).

The following script will show the uses and methods of TECO.

REFERENCES

[1] PDP-10 Reference Handbook, pp. 501-523.

Script

# T E C O

TECO IS A TEXT EDITOR. THE TEXT BEING EDITED IS STORED AS A SINGLE CHARACTER STRING IN THE TECO BUFFER. THIS BUFFER IS ALWAYS JUST AS LONG AS THE STRING IT CONTAINS. THE BOUNDARIES OF THE BUFFER CELLS ARE NUMBERED STARTING TO THE LEFT OF
Z THE FIRST CHARACTER WITH ZERO. THE INDEX OF THE BOUNDARY TO THE RIGHT OF THE LAST CHARACTER IS KNOWN AS "Z". THUS, THE BUFFER CONTAINING THE STRING "ABCD" MAY BE PICTURED AS

[A|B|C|D]
Ø 1 2 3 4=Z .

M,N A SUBFIELD OF THE BUFFER IS DESIGNATED BY THE INTEGER PAIR "M,N" WHERE M<N. THUS, IN THE EXAMPLE ABOVE, THE SUBFIELD "1,3" CURRENTLY
H CONTAINS THE STRING "BC". WE MAY REFER TO THE WHOLE BUFFER BY "H" WHICH IS REALLY JUST AN ABREVIATION FOR "Ø,Z" .

↵ TEXT IN TECO HAS NO LINE NUMBERS, UNLIKE SOS OR EDITOR. THE RETURN KEY OF THE TELETYPE IS TREATED LIKE ANY OTHER SYMBOL, WITH THE EXCEPTION THAT IT IS INPUT TO THE BUFFER AS THE TWO CHARACTERS "CARRIAGE-RETURN" AND "LINE-FEED" . THUS, THE LINE

ABCD↵

WILL APPEAR IN THE BUFFER AS

[A|B|C|D|cr|↓]
Ø 1 2 3 4 5 6=Z .

. ASSOCIATED WITH THE BUFFER IS A CURSOR WHICH CAN BE MOVED TO POINT TO PLACES OF INSERTION, DELETION, ETC. THE CURRENT BOUNDARY POSITION OF THE CURSOR IS KNOWN AS "." .

* TECO SIGNALS THAT IT IS WAITING FOR COMMANDS BY TYPING A "*" . ARBITRARILY MANY COMMANDS
$$ MAY BE STRUNG TOGETHER IN A COMMAND STRING WHICH IS TERMINATED BY TWO ALTMODES (ESC ON SOME TELETYPES). NOTE THAT THE ALTMODE ECHOS AS A "$" . ON RECEIVING THE "$$" TECO WILL INTERPRET THE COMMAND STRING FROM LEFT TO RIGHT, THEN RETURN TO THE USER FOR MORE WITH A "*" .

LET US NOW USE TECO IN ORDER TO CREATE A NEW FILE NAMED "SCRIPT.TEC" .
REMARKS ADDED AFTER THE SESSION WILL APPEAR INTERMITTENTLY AND WILL BE INDENTED.

```
                             WE ENTER FROM PDP-10
                             MONITOR MODE WITH THE
                             CCL COMMAND "MAKE".
                             THIS IS USED WHEN A NEW
                             FILE IS BEING CONSTRUCTED.

.MAKE SCRIPT.TEC

*2=$$                        WHAT IS THE VALUE OF "2" ?
2
*.=$$                        WHERE IS THE CURSOR?
Ø
*.=Z=$$                      WHERE IS THE CURSOR AND WHERE IS
Ø                             THE END OF THE BUFFER?
Ø
*.=$Z=$$                     AN ALTMODE BETWEEN COMMANDS
Ø                             IS OPTIONAL TO IMPROVE CLARITY.
Ø
*HT$$                        TYPE THE WHOLE BUFFER. IT'S EMPTY.
*IABCD                       INSERT THE LINE "ABCD" AND
$HT$$                         TYPE WHOLE BUFFER. THE TEXT OF THE
ABCD                          INSERTION STOPS AT THE FIRST ALTMODE "$".
*.=Z=$$                      WHERE IS CURSOR AND END OF BUFFER?
6                             CURSOR IS AFTER LAST INSERTION.
6                            BUFFER SIX LONG (REMEMBER 2 FOR RETURN.)
*IEFGH                       INSERT SOME MORE LINES. INSERTION
IJKL                          ALWAYS MADE AT POINT OF CURSOR.
MNOP
$$
*HT$$                        TYPE WHOLE BUFFER.
ABCD
EFGH
IJKL
MNOP
*J.=$$                       MOVE CURSOR TO BEGINNING OF BUFFER.
Ø
*CC.=$$                      ADVANCE CURSOR TWO.
2
*-2C.=$$                     MOVE CURSOR TWO BACK.
Ø
*6D$$                        DELETE 6 CHAR TO RIGHT OF CURSOR AND LEFT
*.=$$                         ADJUST STRING IN BUFFER.
Ø
*ZJ-6DHT$$                   JUMP CURSOR TO END, DELETE 6 CHAR TO LEFT, &
EFGH                          TYPE WHOLE BUFFER.
IJKL
*Ø,6KHT$$                    KILL SUBFIELD BETWEEN Ø AND 6.
IJKL                          NOTE THAT Ø,6D  WON'T WORK.
*HK$$                        KILL THE WHOLE BUFFER.
```

| | |
|---|---|
| *IONE | WE INSERT SOME LINES SO |
| TWO | WE CAN EXHIBIT THF LINE |
| THEE | ORRIENTED COMMANDS• |
| SS | |
| *-2T$S | TYPE THE PREVIOUS 2 LINES* |
| TWO | |
| THEE | |
| *-LTS$ | MOVE CURSOR BACK A LINE |
| THEE | AND TYPE 1 LINE* |
| *CCTS$ | MOVE CURSOR FORWARD 2 CHARACTERS |
| EE | AND TYPE REST OF THE LINE. |
| *IR$$ | INSERT THE CORRECTION. |
| *0LTSS | RETURN CURSOR TO BEGINNING OF LINE |
| THREE | AND TYPE THE LINE. |
| *LIFOR | ADVANCE CURSOR A LINE |
| FIVE | AND CONTINUE INSERTING. |
| 55$ | |
| *JSFO$T$S | JUMP TO 0 AND SEARCH UNTIL "FO". MOTE |
| R | CURSOR PLACED AFTER PATTERN FOUND. |
| *IU$0LTS$ | INSERT CORRECTION AND TYPE LINE. |
| FOUR | |
| *ISIX | CONTINUE INSERTIONS. |
| SEVEN | |
| EIGHT | |
| $$ | |
| *HTS$ | TYPE WHOLE BUFFER. |
| ONE | |
| TWO | |
| THREE | |
| SIX | WE FORGOT TO MOVE THE CURSQR BEFORE |
| SEVEN | THIS INSERTION AND SO IT WAS |
| EIGHT | MISPLACED. |
| FOUR | |
| FIVE | |
| *JSSIXS0L. %% | USE SEARCH TO PLACE CURSOR |
| 17 | AT LINE "SIX. TYPE CURSOR POSITION. |
| *3LT$$ | PLACE CURSOR THREE LINES DOWN. |
| FOUR | |
| *1 7, *XAS$ | SAVE FROM 17 TO • IN REGISTER A. |
| *11, .KS$ | DELETE SAME SUBFIELD IN BUFFER. |
| *ZJGASS | JUMP CURSOR TO END AND GET (INSERT) |
| *HT$S | REGISTER A. TYPE WHOLE BUFFER. |
| ONE | THATS BETTER. |
| TWO | |
| THREE | |
| FOUR | |
| FIVE | |
| SIX | |
| SEVEN | |
| EIGHT | |
| | DELETE THE PREVIOUS TWO LINES. |
| *7,,J-PT?>$ | ASSURE CURSOR AT END AND TYPE |
| FIVE | PREVIOUS TWO LINES. |
| SIX | |
| *EX$S | EXIT. THIS WILL WRITE OUT THE BUFFER |
| | TO THE OPENED FILE "SCRIPT.TEC" |
| EXIT | AND RETURN US TO POP-10 MONITOR MODE. |

```
.TECO SCRIPT.TEC

*1ØØØ<A>$$
*HT$$
ONE
TWO
THREE
FOUR
FIVE
SIX
```

EDITING EXISTING FILES IS DONE WITH A TECO
COMMAND WHICH FETCHES THE FIRST FEW CHARS.
A BACKUP FILE (E.G. SCRIPT.BAK) IS ALSO MADE.
THE REMAINDER OF THE BUFFER IS FILLED USING
THE APPEND COMMAND. VALUES GREATER THAN
1ØØØ MAY BE NEEDED FOR LARGE FILES.
MAKE SURE YOUR BUFFER IS FULL BY TYPING IT OR
THE LAST FEW LINES OF IT.

```
*J5<S
$-2DI $>$$
*HT$$
ONE TWO THREE FOUR FIVE SIX
*J<S $;-DI
         $>$$
*HT$$
ONE
   TWO
      THREE
           FOUR
               FIVE
                   SIX
*J5<S
    $-DI
$>$$
*HT$$
ONE
TWO
THREE
FOUR
FIVE
SIX
```

HERE 'SPECIFIC' ITERATION IS USED TO CHANGE
THE FIRST 5 OCCURANCES OF CARRAGE-RET/
LINE FEEDS TO BLANKS. THE COMMANDS IN THE
BRACKETS ARE REPEATED AS MANY TIMES AS IS SPEC
'ARBITRARY' ITERATION (INDICATED BY THE
ABSENCE OF A NUMBER AND THE PRESENCE OF A ;)
ITERATES UNTIL THERE IS NO MATCH, THEN THE
BRACKETS ARE EXITED.

```
*J<SO$;ØLT$L>$$
ONE
TWO
FOUR
*HT$$
ONE
TWO
THREE
FOUR
FIVE
SIX
```

A FREQUENT USE OF ITERATION IS TO "PRINT ALL
OCCURANCES".

```
*ØUA$J<S
$;-2C$.-QAUB$QC-QB"LQBUC'$.+2UA$L>$ØUA$J$QC+1UC$<S
$;-2C.-QAUB$QA+QC+2UA$ØL$QC-QB<I $>L>HT$$
   ONE
   TWO
 THREE
  FOUR
  FIVE
   SIX
*EX$$

EXIT
```

INTERPRETATION OF THIS COMMAND STRING IS LEFT AS
AN EXERCISE TO THE READER.

# XCRIBL---A Hardcopy Scan Line Graphics System for Document Generation‡

R. Reddy, W. Broadley, L. Erman, R. Johnsson, J. Newcomer, G. Robertson and J. Wright

In certain areas of computer science research, conventional line printers and graphics terminals have proven to be inadequate output devices. Typical problems such as a display of digitized (speech or visual) data require either displaying a very large number of (flicker-free) vectors or simulating gray scale output. The need for a hardcopy computer output device capable of producing arbitrary type fonts, graphics, and gray scale images has been obvious. The XCRIBL system, developed at Carnegie-Mellon University (CMU), using a Xerox Graphic Printer (XGP) driven by a minicomputer represents an inexpensive solution to the problem. Careful design of data structures and interface permits the minicomputer to generate each scan line for the XGP as needed without having to resort to brute force solutions. Although the XGP was designed over ten years ago, it had not found wide acceptance as a computer output device because of the excessive processing time and memory requirements of scan-line generation.

The XGP is a facsimile copying machine originally designed for transmission of documents over high bandwidth telephone lines. It has adjustable resolution; the one described here is operated at 192 points per inch which is equivalent to an image of approximately 3.5 million bits for an 8½×11 page. Because of its high resolution each page can contain information equivalent to two pages of conventional computer listing. The XGP printer is a synchronous device, requiring a complete raster line every 5 milliseconds. In order to make the project economically reasonable, a decision was made to use a low-cost minicomputer, a Digital Equipment Corporation PDP-11, with a 28k (16 bit) memory. The limited computing power of the machine influenced many design decisions, such as the inclusion of "modes" of operation of the interface.

The usual Xerox process consists of reflecting light from a printed page onto a selenium drum. The change in electrical charge on the drum caused by the light is used to transfer the "toner" to paper, where a high temperature "fuser" makes the image permanent. Instead of reflected light, the XGP uses the image generated on a cathode-ray tube, one scan line at a time. The image on the CRT is produced by facsimile transmission or, in this case, under computer control. The image is transferred to unsensitized 8½×11 inch continuous roll paper at a speed of 1 inch/second; the paper may be cut to size automatically under computer control.

The PDP-11/XGP system operates as a peripheral device to the main computer, a PDP-10. The character set descriptions for various type fonts may be stored on a

‡This research was supported in part by Xerox Corporation and in part by the Advanced Research Projects Agency of the Department of Defense under contract no. F44620-70-C-0107 and monitored by the Air Force Office of Scientific Research. We would like to thank Bill Gunning, Dave Damouth, and Louis Mailloux of Xerox Corporation for their help and assistance.

small head-per-track disk connected to the PDP-11, or kept on the PDP-10. Text and graphic information are transmitted as needed from the PDP-10 across a high-speed data link (160,000 bits/sec). In addition to textual and graphic information, the data from the PDP-10 may also contain special purpose control information such as changes of type fonts, variations in margins, and special formatting requests such as line justification.

An interesting feature of the system is that every aspect of the output device now becomes a variable when compared with conventional line printers. The character sets, size, all margins, interline spacing, and page size are all variable, and can be changed dynamically during the output of a document.

### Representation of Information

Characters are represented internally as a rectangular bit matrix. Each row of the matrix requires an integral multiple of 8 bits (the byte size of the PDP-11), although not all the bits of the last byte may be used. Characters may be any width from 0 to 255 bits wide and (theoretically) up to $2^{15}-1$ bits high.

Vectors are represented in a conventional scan line format. This format is necessarily different from the ordinary representation of vectors, since for most graphics terminals the entire screen is randomly accessible. In video terminals and hard-copy scan line devices the data must be presented in the order that the scan lines are generated. A software solution to the problem of vector intersection with scan lines was chosen in order to retain the capability for flexible formatting of the output. Vectors are processed in real time, and the available computing power limits the number of vectors which can cross any scan line.

Gray scale representation is achieved by dividing the page into 1/25 inch squares (an area of .0016 square inches) in which an appropriate number of bits is set to black to represent darkness. This is achieved at present by using a rectangular spiral representation of increasing darkness. Generation of gray scale images thus turns out to be a special case of textual output in which a special gray scale type font is used.

The generation of a scan line which contains both textual and graphic information is not a problem for the PDP-11 if the text and graphics is non-overlapping. If the latter is not the case, then one has to resort to an off-line solution of generating the bit image on the PDP-10 or restricting the character set to only fixed-width characters. This is a restriction in the present system but may not be permanent.

## IMPLEMENTATION

In this section we provide a description of the overall implementaion of the system. More detailed descriptions of the various aspects of the system may be found in [1].

The Interface

The purpose of the interface between the PDP-11 and the XGP is to accept a coded scan line from the PDP-11 memory and decode it into a video signal, every 5 milliseconds. A scan line is a bit vector of about 1550 points, in which each point is either on (black) or off (white). There is no gray scale available at this level. The interface has facilities for handling three different modes of data and means for switching between modes, as well as providing control and interrupt functions. The modes available are "character mode", "vector mode", and "image mode".

In the character mode, the first byte sent to the interface represents the number of valid bits (and consquently, the number of following bytes) which contain the data. When the width count is given as zero, then the next byte represents a mode change (to either vector mode or image mode) or a stop code, indicating completion of the data.

In the vector mode, each pair of bytes represents a run-coding of (part of) the data. The first byte of the pair represents the number of white points and the second byte represents the number of black points. When two successive bytes are zero, the interface reverts to character mode.

In image mode, every bit is treated as video information until an error condition occurs, typically "overscan", at which point an interrupt is caused for restart of the next scan line. Because of the high data rate required, this is the only mode which cannot operate in real time from the PDP-10; for this mode, the scan line images are first sent to the PDP-11, where they are accumulated on the disk before being transferred to the XGP.

The support system

There are two components to the support system; one resides in the PDP-11; the other operates as a user program in the time-shared PDP-10. The purpose of the PDP-11 support system is to generate the scan line data needed by the XGP. The support system also services interrupts from the PDP-10/PDP-11 link, examines the incoming data for control information, and selects type fonts from the disk as needed. All of this is done subject to the real-time constraints of the XGP.

The part of the support system which resides in the PDP-10 provides the users with the facilities of sending text, vectors, and character sets across the link. It also provides for conversion of vectors from conventional format to scan line format.

The Character Set Design System

BILOS is a system for the creation and modification of character sets and has many facilities that are common to other interactive editing systems. Rather than manipulating lines of text, BILOS manipulates the rectangular bit matrices which define characters. Any bit of a character matrix may be set or reset by moving a cursor to the appropriate point on a grid and issuing a command.

In addition to these manipulations, the system has facilities for copying, substituting, translating, rotating, stretching, shrinking and reflecting characters. The system currently runs on a storage screen display terminal connected to the PDP-10.

Document Generation Languages

The XGP provides a powerful and flexible tool for the production of printed documents. Since there is a very low cost associated with producing a copy of a document, the user is free to experiment with type fonts, typographic style, physical arrangement of the text and illustrations, etc., until the desired document is produced. The flexibility of type fonts allows mathematical or technical notation to be used freely, without the necessity of typing or drawing the symbols on the final document. Furthermore, the output is "camera-ready"---a distinct advantage in light of rising publication costs.

Two languages for text preparation exist on the PDP-10 at CMU -- XOFF and PUB. Both have been modified to interface with the XGP and are documented in manuals available from the Computer Science Department.

INTRODUCTION TO LOOK

LOOK is a PDP-10 program which transmits information from the 10 to the PDP-11 controlling the XGP. Complete documentation of look is available on file LOOK.DOC[A730GR02]. Below is the sequence of commands used to print this document on the XGP. User input is underlined, comments in lower case.

.R LOOK

| | |
|---|---|
| *!OUTA NGR25.KST | file name for the a partition character set |
| *!OUTB NGRU25.KST | file name for the b partition |
| *←NL=55 | set the number of lines per page to 55 |
| *XCRIBL.XGO | name of the file to be printed |
| *↑C | |