List of Contents
MULTICS PROGRAMMERS' MANUAL

Author-Maintained Library
4/30/75

Commands

(over)

Subroutines

2/25/74	ask_ (installed in maintenance tools)
3/12/75	attach_fortran_file_
10/21/74	basic_plot_
7/27/74	check_basic_file_
2/25/74	check_msf_
9/8/70	cv_ (installed in maintenance tools)
3/12/75	detach_fortran_file_
7/9/74	fillin_dprint_str_
9/6/74	fixed_to_english_
4/13/73	get_caller_ptr_
9/6/74	get_line_length_
4/11/73	get_mydir_
4/30/73	get_seg_ptr_ (installed in maintenance tools)
11/6/74	IMSL Library
2/21/74	linear_q_hash
9/30/73	qd
	release_seg_ptr_ (refer to get_seg_ptr_)
1/18/71	reverse_index_
5/24/71	scan_
2/25/74	tek_
2/25/74	tek_dim_
9/30/73	xcom
9/30/73	xpl_file
9/30/73	xpl_loader_
2/25/74	xpl_operators_

Active Functions

12/7/71	all
2/20/74	bit_count
9/6/74	center
10/31/73	dwd
2/20/74	exist_any
1/18/71	ld
11/15/74	translate

Command
Author-Maintained Library
Dave Moon
545 Technology Square
Room 501, Ext. 3-6013
10/29/73

Name: archive_entries, are
archive_calls, arc

The command archive_entries prints listings of the entry points of the segments within an archive, and archive_calls prints listings of the entry points called by the segments in an archive.

Usage:

are pathname1 ... pathnamen

This will produce a list of all the entry points in all the segments of archives pathname1.archive to pathnamen.archive.

arc pathname1 ... pathnamen

This will produce a list of all the entry points called by all the segments in archives pathname1.archive to pathnamen.archive.

Notes:

Upon encountering bad arguments, are and arc will comment and then continue with evaluation of the next argument.

By design, arc suppresses printing of calls to pl1_operators.

Active Function / Command
Author-Maintained Library
Student Information Processing Board
Room 39-200, Ext. 3-7788
4/13/73

Name: assoc

This procedure implements an associative memory, and is useful primarily for implementing exec_com variables.

Usage:

[assoc name]

1) name is a variable name which has been set to some value by a prior call to assoc_set. (Input)

The returned value is a varying character string representing the value associated with the supplied name. If the name was not found in the memory, the null string is returned.

Entry: assoc_set

This entry is used to associate values with names.

Usage:

assoc_set namel valuel ... namen valuen

1) namei is a character string name of up to 32 characters.

2) valuei is a character string value of up to 168 characters.

Entry: assoc_clear

This entry clears the associative memory

Usage:

assoc_clear

There are no arguments.

Entry: assoc_list

This entry lists the contents of the associative memory.

Usage:

assoc_list

There are no arguments.

Entry: assoc_seg

This entry is provided to allow assoc to reference an associative memory in a permanent segment rather than the default per-process memory.

Usage:

assoc_seg segname

1) segname is the pathname of the segment to be used. If the segment is not found, it will be created. If segname is missing, or is the null string, assoc will revert to the per-process associative memory table.

Command
Author-Maintained Library
Robert F. Mabee
545 Technology Square
Room 518, Extension 5871
2/27/71

Name: bcpl

BCPL is a language designed for but not limited to system building and compiler writing. It is easy to read, but is recommended primarily for experienced programmers.

The bcpl command invokes the BCPL compiler to translate a source segment into a Multics object segment. A listing segment may optionally be produced. The object and listing segments are placed in the current working directory.

Usage:

bcpl pathname option1 option2 ... optionn

1) pathname — is the pathname of the source segment. The compiler will append the suffix ".bcpl".

2) Option*i* — are chosen from the following set:

- source — produces a line numbered listing of the input text, including text inserted by get.
- xref — produces a cross reference table in the listing segment.
- alist — produces a complete assembly-like listing for the object program.
- list — combines the effects of the source, xref, and alist options.
- tree — produces a listing of the syntax tree in the listing segment.
- check — inhibits generating the object segment.
- time — causes timing information to be printed on the console after each pass.
- crep — is a debugging tool. It compiles input from the console and directs the listing output to the console.

pprep is a debugging tool. It lists canonical symbols as they are encountered.

Notes:

The entry name portion of pathname is used as the name for the object segment. For the listing segment ".list" is appended to the entry name. The name of the source segment is obtained by appending ".bcpl" to pathname.

The 'get' preprocesser convention is implemented as follows:

> The reserved word get must be followed by a string constant. All text remaining on the line after the string constant is ignored. The string identifies a segment of BCPL source which is to be logically inserted into the program. The name is formed by appending ".bcpl" to the string constant. The segment (or a link to it) must be in the current working directory.

A BCPL manual and Multics implementation guide will be available soon.

Command
Author-Maintained Library
Thomas Casey
575 Technology Square
CISL, Fifth Floor
491-6300, Ext. 237
1/8/74

Name: blip

The command blip types a short character string (a "blip") on the console every few seconds of CPU time used by the user's process. Both the time between blips and the type of blip may be specified by the user. A sequence of different blips that are to be cycled through also may be specified by the user. Blips will be typed out until the blip_off command is given.

Usage:

blip time -options and/or blips-

This command takes an unlimited number of arguments. The first argument is the time in CPU seconds between blips. Because this is a floating-point number, times of less than one second may be specified. This time will be adhered to exactly. If 50 blips are given during the course of working, 50*time seconds of CPU time were used by the process. The command will even try to remain faithful if the time specified is less than the amount of time required to execute the blip program. If the time is about two seconds or more, it is estimated that the overhead for the blip program is about 0.1 second per blip.

The rest of the arguments are either options or character strings to be used as blips after they have been modified as specified by the options. Because the arguments are processed from left to right, blip strings can be affected only by options to the left of the string. The options are:

-red	A red ribbon shift is added at the beginning of the blip string and a black ribbon shift is added at the end.
-black (-bl)	No ribbon shifts are added to the blip string.
-nla (-nl)	A newline character is added at the end of the blip string (newline after).
-nlb	A newline character is added to the beginning of the blip string (newline before).
-nnl	No newlines are added to the blip strings.

If no options are specified, the options -red and -nnl are in effect. If no blip strings are specified, then the 10 decimal digits from 0 through 9 will be the 10 blip strings to be cycled through. The first blip is typed out during the blip command, and it is the first blip string. The second blip is the second blip string. If n blip strings have been specified, then the n+1*st* blip will be the first blip string.

Blips are always typed on the console, even when using file_output.

Entry: blip_off

This command turns off the blips from the previous use of the blip command. If a blip is being given, another use of the blip command will turn off the previous blip automatically.

Examples:

```
blip 2
0r 1133 1.302 4.108 93
pl1 new_program
1PL/1
2345678r 1140 15.137 29.763 749
new_program
901
QUIT                          /*got into an infinite loop*/
r 1141 8.548 5.296 123
```

Command
Author-Maintained Library
Joseph W. Dehn III
Room 39-200, Ext. 3-7788
1/18/75

Name: calc_costs, cc

This command calculates the dollar cost of a given amount of cpu time, memory units, and connect time.

Usage:

calc_costs cpu1 mem1 . . . cpun memn -connect X -shift K

1) cpui specifies cpu time in seconds. The total cpu time is calculated by adding together all of the cpu times that are given.

2) memi specifies memory usage in the units given in a ready message. The total memory usage is calculated by adding together all of the memory usage figures that are given.

3) -connect X specifies the connect time in minutes.

4) -shift K specifies the shift whose prices are to be used in the calculation of the dollar cost. If this control argument is not given, a table will be printed showing the cost for all shifts that are currently defined.

Examples:

calc_costs 1.007 6.728 -shift 2

calc_costs 2.3 5.7 1.1 3.5 -connect 3

Command
Author-Maintained Library
Overlap Project Staff
NE40-500, M.I.T. Ext. 3-2053
2/5/75

Name: card_input

This routine may be used to read a deck of cards into a segment created in the user's working directory at the time the request is submitted. It supports the conventions about reading via links, and provides the following services:

1) The user need not be concerned about a "temporary name" and a "permanent name" -- the name appearing in the segment name field of the control card (1) placed at the beginning of the deck is the name of the segment which will be created and into which the card images will be copied.

2) The user need not be concerned with eliminating a link which may later cause problems, because the program removes the link when the cards are copied.

Use of this program greatly simplifies the task of reading cards on Multics, to the extent that it may be used with little instruction, even by a new user. Such instruction might be succinctly stated, "Put the name you wish the segment to have in the segment name field of the first (Control) card, submit the deck for reading, and then execute this program from the directory in which you want the segment to reside, using the same name that appears on the control card as the command argument."

The program may also be used to copy an arbitrary segment via a link to it, effectively changing the name on the link to the name on the segment.

---

(1) For details concerning the format of and information required to use control cards, please see the M.I.T. Information Processing Center publication MS-1, "Bulk I/O on Multics." Note, however, that this publication describes a series of naming rules that this command makes unnecessary.

Usage:

card_input card_deck_name

card_deck_name is the name that appears in the segment name field on the first (control) card of the deck.

Note:

This command must be invoked from within the same directory as that appearing in the directory name field on the first (control) card of the deck.

Method:

The experienced Multics programmer may desire to know the steps undertaken by the program during execution. An outline of the steps is:

1) The name supplied in the command line argument is checked to ensure that it exists and is a link, and that the link refers to an existing segment.

2) A uniquely named segment is created in the working directory; the segment referred to by the link is copied into the uniquely named segment.

3) The link is unlinked, and the uniquely named segment is renamed to the name that was on the link.

If any error occurs, the program displays a message and returns.

Command
Author-Maintained Library
Student Information Processing Board
Room 39-200, Ext. 3-7788
8/04/74

Name: clean_up

The clean_up command enables the user to clean a directory or directory subtree of worthless entries: segments that have not been used or modified since some specified time; links that point to a non-existent target, and directories that contain no entries at all.

Usage:

clean_up -path- -options- -control_args-

1) path is the pathname of the directory that is to be cleaned up or "-working_directory" ("-wd"). If neither is given then the current working directory is assumed.

2) options indicate which entries are to be deleted. They can be selected from the following list:

-date_time_modified, -dtm
specifies that segments found in the directory that have not been modified since a particular time are to be deleted. This argument can be followed by a string giving the desired time in format acceptable to convert_date_to_binary_. If a time is not given by the user, one month prior to the current time is assumed.

-date_time_used, -dtu
specifies that segments found in the directory that have not been used since a particular time are to be deleted. As above, the user can specify the time he desires, or let one month before be assumed.

-link, -lk specifies that links found in the directory that point to a non-existent target are to be unlinked.

-directory, -dr
specifies that any inferior directories found in the directory that have no entries themselves are to be deleted.

If no options are specified, segments not used in the last month will be deleted.

3) control_args are selected from among the following:

-long, -lg produces a listing of the directories searched and entries deleted. This listing is written on user_output and includes the date/time dumped for segments and directories and the target of a link. This is the default.

-brief, -bf suppresses the listing described above. ("-long" is the default).

-force specifies that entries are to be deleted even if their safety switch is set. If this argument does not appear, an error message will be printed when an attempt is made to delete such an entry.

-walk, -wk specifies that clean_up is to walk through all inferior directories of the directory specified looking for entries to be deleted. Note that the contents of a directory are examined for deletion before a check is made to see if the directory is empty. As a result, if "-dr" has also been specified and if all the entries are deleted from an inferior directory, then that directory will also be deleted.

-walk_force, -wf
specifies that inferior directories are to be walked as above, and that clean_up is to ensure that the user has "sma" access to each inferior directory.

Examples:

To delete all segments in the current working directory that have not been modified in the last month:

```
clean_up -date_time_modified
```

To remove all links that point to a non-existent target in the directory temp and all directories under it:

```
clean_up temp -link -walk
```

To delete all segments that have not been used since July 1, 1974 and all directories that are empty:

```
clean_up -wd -dtu "July 1, 1974" -dr
```

To purge the entire directory subtree under "old" of old segments, null links, and empty directories (including those made empty by clean_up):

```
clean_up old -dtu -dtm -lk -dr -wf
```

Command
Author-Maintained Library
J.B. O'Connor
Room 39-473, Ext. 6321
11/12/70

Name: convert_360_pl1, c360p

This command converts an IBM 360 PL/I program read in by the Multics card reader to a form acceptable to the Multics PL/I compiler. Specifically it:

1. maps all upper case letters in the file to lower case,

2. changes all apostrophes (') to quotes ("),

3. deletes any characters appearing in card columns 73 through 80 (sequence numbers), and

4. deletes trailing blanks on a line.

Usage:

convert_360_pl1 pathname1 pathname2

Pathname1 is the name of the segment to be converted and pathname2 is the name of a new segment to be created to contain the converted program. If pathname2 is omitted, pathname1 will be rewritten.

Notes:

The translation does not take into account the 48-character set used in IBM 360 PL/I. For example, if the operator "LT" is used in the 360 program, the user must change this to the character "<" with a Multics editor.

If character constants start on one card and end on another card in the 360 program, the effect of trimming off trailing blanks may change the value of the constant.

Example:

c360pl1 ibm.pl1 multics.pl1

This command can be applied to segments that it had previously produced without effect. It cannot be applied to segments entered on the console, however, unless tabs have not been used and care has been taken to follow IBM 360 card conventions (because characters after the 72nd column on a line will be deleted).

Command
Author-Maintained Library
Joseph W. Dehn III
Room 39-200, Ext. 3-7788
7/29/74

Names: convert_line_feeds, cvlf

This command converts an ASCII segment which contains carriage-return and line-feed codes as used by other systems into a segment which uses the Multics new-line character. A combination of line-feed and carriage-return is replaced with a new-line character, a carriage-return by itself is left as is, and an isolated line-feed may be treated in one of three ways: deletion, replacement, or "simulation" by new-line and spaces.

Usage:

convert_line_feeds path1 path2 -cntlargs-

path1 is the pathname of the segment to be converted

path2 is the name of the segment to be produced. If this argument is omitted, the conversion will be done in place (the new segment will replace the original).

Control arguments may be used to specify the handling of isolated line-feeds:

-delete, -dl — isolated line-feeds are to be removed

-replace xxx — isolated line-feeds are to be replaced by the string xxx, which may be up to 32 characters in length

-space — isolated line-feeds are to be replaced by a new-line and spaces to effect the carriage motion of a line-feed

-tab N — specifies the width of a tabulation character when using the -space option. If this option is not given, tabs are assumed to be ten columns wide.

Note:

The ASCII carriage-return code is octal 015. The octal code for line-feed is 012. Multics uses the code 012 for new-line. A carriage-return/line-feed combination is treated by removing the code 015 from the text.

Command
Author-Maintained Library
Thomas Casey
575 Technology Square
CISL, Fifth Floor
491-6300, Ext. 237
1/8/74

Name: convert_multics_fortran, cmf

This command converts Multics FORTRAN programs into a form acceptable by IBM 360 FORTRAN. The command:

1) changes lower-case characters to upper-case,

2) changes quotes (") to apostrophes ('),

3) changes the continuation card convention from a "&" to an "X" in column 6 of the next card,

4) changes horizontal tabs to spaces,

5) puts the first four characters of the output pathname into columns 73-76,

6) sequences lines in columns 77-80,

7) does not handle implied continuations,

8) will continue lines greater than 72 characters in length onto the next card(s), and

9) always will put the segment following a continuation character on the next card.

Usage:

```
cmf pathname1 pathname2
```

Where pathname1 is the name of the segment to be converted and pathname2 is the name of the segment to contain the converted program. If pathname2 is omitted, the output will be under pathname1".360".

Example:

```
cmf multics.fortran 360.fortran

        OR

cmf multics.fortran
```

Pathname2 defaults to multics.fortran.360 in the latter case.

This command cannot be applied to segments which it had previously produced. Non-FORTRAN lines in the program could conceivably cause incorrect operation.

Command
Author-Maintained Library
Joseph W. Dehn III
Room 39-200, Ext. 3-7788
8/7/73

Name: convert_radix, cvr

This command will convert the character string representation of an integer in one radix into its representation in another radix and print it on the console. The input and output radices may be from 2 to 64, and the characters to be used as numerals may be specified.

Usage: convert_radix input input_radix output_radix numerals

1) input is the number to be converted.

2) input_radix is the radix to convert from.

3) output_radix is the radix to convert to.

4) numerals is a character string of up to 64 numerals.

The "output_radix" and "numerals" arguments may be omitted. The former defaults to ten, and the latter defaults to, in order: the arabic numerals, the capital letters, and the lower case letters. Note that if the numerals are to be specified, the output radix must be specified.

If several numbers are to be converted in the same manner (same input and output radices and numerals), specifying "*" as the first argument will put the program into a loop reading input from the keyboard, one number per line. This mode may be terminated by entering a blank line.

Command
Author-Maintained Library
Student Information Processing Board
Room 39-200, Ext. 3-7788
4/7/73

Name: decimal_to_octal, dto

The decimal_to_octal command will convert a decimal argument to octal and print it on the console.

Usage:

decimal_to_octal decimal_number

1) decimal_number is a decimal number to be converted to octal.

Entry: octal_to_decimal, otd

This entry will convert an octal argument to decimal and print it on the console.

Usage:

octal_to_decimal octal_number

1) octal_number is an octal number to be converted to decimal.

Command
Author-Maintained Library
John C. Klensin
575 Technology Square
Fifth Floor, Ext. 3-6217
2/20/74

Name: file_output_unique, fou

This command facilitates dprinting the results of other commands.

Usage:

```
file_output_unique  [dp -options]
  command 1
  ...
  command n
  q
```

where dp -options are zero or more of the options normally passed to the dprint command. These options include "-ds," "-he," and "-cp."

The differences between the sequence

```
fou
command-line 1
command-line 2
...
...
command-line n
q
```

and the sequence

```
file_output name1;  command-line 1;  console_output
file_output name2;  command-line 2;  console_output
...
...
file_output namen;  command-line n;  console_output
dp -del name1 name2 ... namen
```

are as follows:

1. Each file produced by file_output_unique begins with a header consisting of the command line responsible and a time and date message.

2. Each file produced by file_output_unique ends with a distinctive trailer line consisting of "END*END*...*END."

Notes:

1. file_output_unique interprets command lines with the user's command interpreter, whatever that may be.

2. This version of file_output_unique does not require that the I/O daemon have any special privileges in the working directory.

3. This routine uses fillin_dprint_str_ to evaluate "dprint" options. See the current description of that subroutine for an exact list of the options accepted

Example:

```
fou
pwd;  ls -dtm
pli big_mother_bound_archive
q
```

Command
Author-Maintained Library
John C. Klensin
575 Technology Square
Fifth Floor, Ext. 3-6217
7/9/74

Name: get_current_charges, gcc

This command is used to provide a project administrator with a variety of information in compact form about a particular user (or list of users) on a particular project. It can display amounts spent per shift during the current month, amounts spent absentee, and on io, and the number of pages in the user's directory. It can display the limits on the above, if any. In addition, it can display the user's attributes and the date and device on which he last logged in.

Usage:

get_current_charges project-name user-name user-name ... options

Where:

project-name — is the name of the project for which information is to be obtained.

user-name — is the name of a user on that project for whom information is to be obtained. Up to 20 user names may be specified.

options — are control arguments for the program. These may occur in any order, and may be intermixed with user names. Any options that occur apply to the entire command invocation. These options are:

-total, -tt — Display only the dollar total (and limit, if that option is used).

-limit — Display the limits, in addition to the charges.

-attributes — Display the user's current attributes.

-last_login, -ll — Display the time, date and console id of the user's last login.

-absolute, -abs — Display information about "absolute" spending and, if "-limit" is specified, limit and cutoff information.

Output:

If no options are specified, a single line will be displayed containing the user name, the total charge, the four shift charges, the io daemon and absentee charges, and the page usage on the user's directory. If -total is specified, only the total charge will appear.

The other options cause additional per-user lines to be displayed. These are as follows:

For "-limit", a line showing the total limit, the four shift limits, and the user's page quota. If "-total" was specified, then only the total limit is printed.

For "-attributes" a line is displayed showing the attributes as text strings.

For "-last_login", a line is printed containing the time, date, and console id, of the user's last login.

Notes:

1) If errors occur for a given user, messages will be printed at the end of the information for that user. If the person using this program does not have access to get quota information, those fields will be set to zero and a message printed after other information.

2) This command requires read access to the project definition table of the project for which information needs to be obtained. As a consequence, this command is of no use to any Multics user who is not a project administrator.

Command/Active Function
Author-Maintained Library
Student Information Processing Board
Room 39-200, Ext. 3-7788
9/6/74

Names: get_object_info, goi
get_component_info, gci

These commands/active functions give information about the compilation of an object segment.

Entry: get_object_info, goi

Usage:

(as a command) get_object_info path key

(as an active function) [get_object_info path key]

1) path is the path of the object segment.

2) key specifies the information to be printed/returned, and may be one of the following:

author
at — the userid of the user who produced the object segment.

date_time_compiled
dtc — the date and time of the compilation.

compiler — the name of the translator that produced the object segment.

version — the version of the translator that produced the object segment.

comment
options — a translator inserted comment. The PL/I compiler uses this field to indicate the options that were specified at compile time.

Entry: get_component_info, gci

This command/active function gives information about the compilation of a component of a bound object segment.

Usage:

(as a command) get_component_info path component_name key

(as an active function) [get_component_info path component_name key]

1) path same as above.

2) key same as above.

3) component_name is the name of the component of the object segment for which the information is to be returned.

Notes:

1) If the "component" argument to get_component_info is omitted, it will be assumed to be the same as the entryname portion of the pathname.

2) If "get_object_info" is used on a bound segment, it will return information about the binding.

Examples:

```
get_object_info xyz author

get_component_info bound_xyz_ xyz dtc

get_component_info xyz version
```

Command
Author-Maintained Library
B. E. Hampson
Room 38-644, DL 9204
03/11/74

Names: lock_console, lkc

This command can be used to protect your process from unauthorized use in the event you have to leave your terminal for any length of time. Lock_console has two modes of operation: one in which your process is simply put to sleep awaiting your return, and one in which lock_console will execute a command line under its protection.

When in command-line-execute mode, lock_console will (by default) establish an on unit for the condition "any_other". The effect of this will be to "catch" any condition signalled by any procedure invoked by the command line. Lock_console captures the signal and prints a message to the effect that the signal occurred; but, unlike the system default on unit, the process is not returned to command level. Rather, lock_console calls itself recursively with a basic lock time and grace (see below) of 24 hours. If the user does not reclaim the terminal in 48 hours, he is either logged out, or lock_console returns to its caller. If the terminal is reclaimed, the signalling of the condition will continue just as if lock_console had not intercepted it. Note that, among other things, this feature prevents someone else answering a question for you (provided the question is asked in such a way as to cause a condition to be signalled, e.g., command_query_).

There is one exception to the above "signal-catching" scheme: if a quit is done while executing the command line, the user's password is requested. If correct, the quit is passed on and the process gets to command level. If incorrect, the quit is simply returned from.

Protection is achieved by means of an 8-character password. Lock_console will request this password when invoked (see the listed exceptions below) either under overprint or with printer disabled. To reclaim a terminal, the procedure is to strike the QUIT key, and to enter the same password when it is requested.

Usage:

lock_console -control1- ... -controln-

The controli are optional control arguments, and may be chosen in any order from the following list:

-time TTT
-tm TTT
Sets the basic lock time to TTT minutes. This value is the time after which lock_console will take further action (in the absence of a grace period), if the terminal has not yet been reclaimed. The default basic lock time is 10 minutes.

-no_logout
-nol
Specifies that, at the expiration of the basic lock time (and the grace period, if any), lock_console is to return to its caller instead of logging the user out. The default is to log the user out.

-prev_pw
-ppw
Specifies that the password from the previous invocation is to be used (a new password will not be requested if this argument is used). This argument will be ignored (with consent) if it is used in the initial invocation in a process.

-defer_messages
-dm
Specifies that the command "defer_messages" is to be executed (before executing the command line, if any). The command "immediate_messages -print" will also be executed, just before lock_console returns to its caller.

-set_pw PPP
-spw PPP
Sets the password for this invocation to PPP. This argument overrides -ppw if both are used. PPP is limited to eight characters.

-call CCC
-cl CCC
Requests that the command line CCC be passed to the user's command processor for execution under the protection of lock_console. The real time required to execute the command is counted against the basic lock time, but the grace period does not begin until after the command line has finished executing.

-grace GGG
-gr GGG
Sets the grace period to GGG minutes. Lock_console will neither return nor log out, after the basic lock time has been exceeded (or after the command line has finished executing, whichever is longer), until this grace period expires. The default grace is 0 minutes (no grace).

-no_catch_signals
-ncs
Specifies that during the executing of the command line, lock_console is not to establish an on unit for the condition any_other. The default is to provide such an on unit. A command line running in "no catch" mode cannot be quit out of by someone who does not know the password; however, if some error or other condition is signalled and the user has no on unit for it, the process will get to command level via the system default on unit, and then be vulnerable to unauthorized use.

-brief
-bf
Causes lock_console to be less verbose.

-no_print_off
-npf
Specifies that the user's terminal does not have the printer off feature, and that therefore the user desires his passwords be requested under overprint.

Example:

```
lock_console -tm 25 -no_logout -grace 2 -cl "ls -p <" -bf
```

Command
Author-Maintained Library
J. R. Steinberg
Room 39-427, Ext. 3-7184
1/8/74

Name: lower_case

This command, given the pathname of an ASCII file, maps all upper-case letters in the file to lower case. It is intended for use on card-punched decks which have been input to Multics and which contain source code for a compiler which expects lower-case keywords (e.g., PL/I, FORTRAN).

Usage

lower_case pathname

Where pathname is the pathname of an ASCII file which is to be mapped to lower-case.

Active Function/Command
Author-Maintained Library
Student Information Processing Board
Room 39-200, Ext. 3-7788
5/10/74

Name: number_queued, nq

This procedure returns the number of requests in an I/O daemon queue.

Usage:

(as active function): [number_queued qnum dvc]

(as command): number_queued qnum dvc

1) qnum a positive integer which specifies the queue number

2) dvc the name of the device class (optional)

Example:

As an active function, the procedure may be used by an exec_com or absin control program to decide in which queue to submit a request:

```
&if  [greater [nq 3]  15]  &then dp -q 2 sample.list
                           &else dp -q 3 sample.list
```

Command
Author-Maintained Library
Student Information Processing Board
Room 39-200, Ext. 3-7788
4/7/73

Name: octal_to_float, otf

The octal_to_float command will print the floating point value of an octal argument on the console.

Usage:

octal_to_float octal_number

1) octal_number is an octal number to be converted to float.

Entry: float_to_octal, fto

This entry will print the octal representation of a floating binary argument on the console.

Usage:

float_to_octal float_number

1) float_number is a float number to be converted to octal.

Command
Author-Maintained Library
Joseph W. Dehn III
Room 39-200, Ext. 3-7788
7/29/74

Name: print_basic_file

The print_basic_file command may be used to print the contents of a file as interpreted by the BASIC language. The file may be terminal-format, random-string, or random-numeric.

Usage:

print_basic_file file_spec

where file_spec is a string specifying the file to be printed. If the string contains blanks, it must be enclosed in quotes. The type of the file will be printed, followed by the contents.

Command
Author-Maintained Library
John C. Klensin
575 Technology Square
Fifth floor, Ext. 6217
1/18/71

Name: print_login_dir, pld

The print_login_dir command causes the name of the original login directory to be printed on the console.

Usage: print_login_dir

Command
Author-Maintained Library
Room 39-2009, Ext. 3-7788
5/10/74

Name: print_string, ps
print_string_nnl, psnnl
print_string_ht, psht
print_string_tb, pstb

This procedure prints on the console the line formed by concatenating its arguments, with one blank between each argument. The entries differ only in the character appended at the end of the line, which is as follows:

Entry	Abbreviation	Trailing Character
print_string	ps	newline
print_string_nnl	psnnl	(none)
print_string_ht	psht	horizontal tab
print_string_tb	pstb	space

Usage:

print_string arg1 ... argn

Example:

print_string [path &1] as of [date]

Command
Author-Maintained Library
Joseph W. Dehn III
Room 39-200, Ext. 3-7788
5/15/73

Name: punch_paper_tape, ppt

This command punches an ascii segment on the paper tape punch of a model 33, 35, 37, or 38 teletype. The segment may be punched as is, or carriage returns may be inserted at the end of each line. The file is preceeded by a punch-on (device control 2) character and a leader of nulls, and followed by a trailer and a punch-off (device control 4).

Usage: punch_paper_tape <optl> ... <optn> <pathl> ... <pathn>

Options take effect as found in the argument list, and should normally preceed all path arguments.

-insert_cr
-icr
enables the insertion of carriage returns and null padding at the end of each line.

-override
causes the fact that you are not on a teletype to be ignored.

Notes:

When in -icr mode, each \012 is replaced by \015, \012, \000. In addition, a punch-on character is added after each punch-off, to allow the punching to continue. When not in -icr mode, no processing is done at all. In either mode, all other characters are sent as is, including \004 and \005, whose effects on the particular terminal should be considered when trying to punch out strange segments. Note that this command cannot be used to punch non-ascii files since the high-order bit of each byte is lost.

Examples:

```
ppt -icr foo.fortran test.basic

punch_paper_tape -override sample_file
```

Command
Author-Maintained Library
Student Information Processing Board
Room 39-200, Ext. 3-7788
9/6/74

Names: push_wdir, pop_wdir

These commands may be used to change the working directory while pushing the previous working directory onto a "stack". The previous working directory may then be restored.

Entry: push_wdir

This entry changes the working directory and pushes the previous working directory.

Usage:

push_wdir path

1) path is the pathname of the new working directory

Entry: pop_wdir

This entry restores the working directory to the value pushed by the push_wdir command.

Usage: pop_wdir -control_arg-

The control argument "-all" specifies that the working directory should be restored to what it was before the first push_wdir command, i.e., the "stack" is popped all the way.

Command
Author-Maintained Library
Joseph W. Dehn III
Room 39-200, Ext. 3-7788
7/27/74

Names: read_dartmouth_tape, rdt

The read_dartmouth_tape command may be used to read all or some of the files on a Dartmouth library tape into Multics segments. The tape is assumed to be a 9-track image of a 7-track "format 1" ASCII tape, with the first file being a directory.

Usage:

read_dartmouth_tape -control_args- -file_names-

1) -tape xxxxx — specifies the volume_id of the tape to be read. If this argument is not specified, it is assumed that the stream "dart_tape" is already attached to the appropriate device.

2) -hold — specifies that the tape is not to be detached after the files have been read.

3) -all — specifies that all files on the tape are to be read. If this option is given, no file names may be specified.

4) -trace — specifies that as each file is encountered on the tape, its number and name are to be printed.

5) -list — specifies that a file is to be created with the name "tapeid.dliblist" listing the name and type of all files on the tape.

6) -noconversion — specifies that no conversion is to be done for "random-string" and "random-numeric" files.

7) -name — specifies that the following argument is to be treated as a file name, even if its first character is "-".

Notes:

1) Files on the tape are in one of three formats. "TERMINAL-FORMAT" files are read with no conversion. "RANDOM" files are converted into a form suitable for use with the Multics BASIC language, unless the -noconversion option is specified.

2) Darmouth file names are usually composed of upper-case letters. When specifying file names, be sure to spell them exactly as listed in the tape directory.

Examples:

To read all files and create a directory listing:

```
rdt -tape 12345 -list -all
```

To read two files and hold the tape:

```
rdt -tape 54321 -hold RATINV TRUTHTAB
```

Command
Author-Maintained Library
Elaine Franklin
575 Technology Square
Cambridge Project
Fifth Floor, Ext. 2054
9/9/71

Name: reformat_line, rfl

The reformat_line command may be used to scan a segment line-by-line, forcing each line to fit into a specified number of columns.

Usage:

reformat_line pathname1 [pathname2][-start cc1][-end cc2][-wrap cc3]

or

rfl pathname1 [pathname2][-start cc1][-end cc2][-wrap cc3]

Where:

pathname1	is the pathname of the input segment.
pathname2	is the pathname of the output segment. If pathname2 is not supplied, then the segment pathname1 will be replaced upon completion of the command.
cc1	is the column in which a line normally starts. If cc1 is not supplied, then it is assumed to be 1. cc1 may not be less than 1 nor greater than 132 or cc2. If supplied, cc1 must be preceded by the string "-start".
cc2	is the last column incorporated in a line before it wraps. If cc2 is not supplied, then it is assumed to be 80. cc2 may not be greater than 132 nor less than cc1. If supplied, cc2 must be preceded by the string "-end".
cc3	is the column at which a wrapped line is to start. If cc3 is not supplied, then it is assumed to be 1. cc3 may not be less than 1 nor greater than 132 or cc2. If supplied, cc3 must be preceded by the string "-wrap".

The program reformat_line is designed to scan a segment line-by-line and reformat it as follows. Any horizontal tabs which are encountered during processing will be converted to the appropriate number of blanks before further processing of the line is done. Lines will be reformated so that they will begin at column ccl and continue up to and including column cc2. If a line is longer than cc2-ccl+1 characters, it will be broken up or reformated to continue onto the next line, beginning in the column specified for cc3 and continuing through column cc2 of that line. If ccl or cc3 is greater than 1, blanks will be inserted in columns 1 through ccl-1 or cc3-1, respectively.

Examples:

```
reformat_line alpha
```

would take the segment alpha in the current working directory and reformat it using the default values for ccl, cc2 and cc3 (1, 80 and 1, respectively). The old segment alpha would be replaced by the new segment alpha in the current working directory.

```
reformat_line -end 132 beta -wrap 60 gamma
```

would take the segment beta in the current working directory and reformat it using the default value of 1 for ccl and the values 132 and 60 for cc2 and cc3, respectively. An output segment named gamma would be formed in the current working directory.

Warning:

This command considers anything of the form "character backspace character" (e.g., A) to be three separate character positions. If this action inconveniences anyone, please contact Elaine Franklin, Ext. 2054.

Command
Author-Maintained Library
Student Information Processing Board
Room 39-200, Ext. 3-7788
4/7/73

Name: repeat

The repeat command makes it possible to have a command line repeated any number of times.

Usage:

repeat [-times n] command_line

1) n — is the number of times to repeat the command. If the -times option is omitted, the command will be repeated indefinitely.

2) command_line — is the command line to be repeated. It need not be enclosed in quotes.

Command
Author-Maintained Library
Paul Green
Honeywell - DSO
575 Technology Square
491-7300
1/9/74

Name: return_to, rt

This command will cause a return to a procedure which called out from a specified stack frame, thus simulating a normal return from that call.

The return_to command is simply an interface to the system subroutine "unwinder_"; refer to the MPM Subsystem Writer's Guide for complete information.

Usage:

return_to frame

The command argument "frame" is the octal offset of the stack frame (in the current ring) to be returned to. This offset may easily be determined by using the ".t" request to "debug," or the "trace_stack" command.

Example:

return_to 5540

will return to the procedure owning the stack frame at octal offset 5540 in the current ring's stack.

Active Function/Command
Author-Maintained Library
Student Information Processing Board
Room 39-200, Ext. 3-7788
5/10/74

Name: ring

This procedure returns the current ring of execution.

Usage:

(as active function): [ring]

(as command): ring

Example:

```
&if [equal [ring] 5] &then subsys_who &1 &2 &3 &4
                     &else who &1 &2 &3 &4
```

Command
Author-Maintained Library
Gary M. Palter
Room 39-200, Ext. 3-7788
11/29/73

Names: subtree_status, stst
directory_status, dst
global_status, gstat

This command is related to the status command. It will produce a listing of a subtree of the hierarchy (or just a single directory) with complete information about all (or some) of the entries in each directory. It also will list the quotas and Initial Access Control Lists of each directory.

Options exist to specify the type of sort to be performed on entries in a directory, to specify which classes of entries are to be listed (similar to the list command), and how multi-segment files are to be treated.

Output is in a format suited almost solely for dprinting. The field width is 136 characters, and there are form-feeds at the start of each directory's listing.

Usage: subtree_status, stst

subtree_status -directory- option1- ... -optionn-

1) directory — is the relative pathname of the directory which is the top node of the hierarchy to be listed. If the directory is omitted, the working directory will be used.

2) optioni — is chosen from the following list of options (see also Notes below):

-brief, -bf — will cause any messages which might be expected not to be reported through com_err_. See the Notes below.

-check, -ck — will cause subtree_status to walk any multi-segment files that do not conform to the rules for multi-segment files (in the write-up of check_msf_) like any other directory.

-name, -nm specifies that the entries in a directory are to to be sorted by primary entry name in the listing.

-date_time_used, -dtu
specifies that the entries are to be sorted by date time used in the listing.

-date_time_modified, -dtm
specifies that the entries are to be sorted by date time modified.

-segment, -sm specifies that segments are to be included in the listing (see Notes).

-directory, -dr specifies that directories are to be included in the list.

-multisegment_file, -msf
specifies that multi-segment files are to be included.

-link, -lk specifies that links are to be included.

Notes:

1) The messages suppressed by the -brief option are those concerning insufficient access to a directory, or a directory containing none of the entries selected for listing.

2) If any one of the arguments for selection of entry classes for the listing is present, then information only about those classes for which an option appears will be given.

3) Only one of the sorting options may be given per command.

4) The defaults for the options are:
-brief off
-check off
sorting none
selection all entry classes

5) An example of the output produced by this command appears below.

6) The "Records" field of a segment or directory entry is the number of non-zero pages for that branch. The "Blocks" field is the length of the segments in blocks of 1024. (These two numbers will be different only if there are pages containing all zeroes.)

7) The "Bit Count" field of a multi-segment file entry is the number of segments inferior to the multi-segment file. The "Records" field is the number of pages used by the directory portion of the multi-segment file. The "Total" field is the number of records in the multi-segment file (the same value that is returned by the list command).

8) If the target of a link does not exist, an asterisk will be placed in the "Null" field of the link's entry.

Usage: directory_status, dst

directory_status -directory- -option1- ... -optionn-

The arguments are identical to those for subtree_status. This entry differs from subtree_status in that it does not walk the subtree of the specified directory, but produces a listing of only that directory. As a consequence, the -check option is meaningless for this command.

Usage: global_status, gstat

global_status -directory- -option1- ... -optionn-

The arguments are identical to those for directory_status. This entry differs from directory_status in that the default sorting option is date time used (instead of no sort). This entry is provided as a replacement for an older version of the gstat command.

11/28/73 0328.72 est Wed

>udd>Janus>Palter>subtree_status_example_directory

Records Used: 69

Initial Access Control List - Segments

Ring 4:	rew	Palter.*.*	r w	*.SysDaemon.*	re	*.SIPBADMIN.*
	r	*.*.*				
Ring 5:	r	A_very_long_name_.Project.*	r w	VanVleck.*.*	r w	Roach.*.*

Segments: 1

Primary Name / Secondary Name(s)	Ring Brackets	Author / Bit Count Author / Mode / Access ID	Used Modified	Dumped Entry Modified	Bit Count Max Length	Records Blocks	Safety Copy
empty_segment	(4,4,4)	Palter.Janus.a	11/28/73 0313	11/28/73 0326	0	0	off
		Palter.Janus.a	11/28/73 0313	11/28/73 0315	65536	0	off
a_second_name		r w Palter.Janus.*		rew Palter.*.*			
		r w *.SysDaemon.*		re *.SIPBADMIN.*			
		r *.*.*					

No Directories.

Multi-Segment Files: 1

Primary Name / Secondary Name(s)	Ring Brackets	Author / Bit Count Author / Mode / Access ID	Used Modified	Dumped Entry Modified	Bit Count Max Length	Records Total	Safety Copy
a_multisegment_file	(7,7)	Palter.Janus.a	11/28/73 0328	11/28/73 0325	5	1	off
		Palter.Janus.a	11/28/73 0310	11/28/73 0310	65536	69	off
a_msf		sma Palter.Janus.*		sma *.SysDaemon.*			

Links: 1

Primary Name / Secondary Name(s)	Author / Null Pathname	Modified	Dumped
empty	Palter.Janus.a	11/28/73 0311	01/01/01 1900
	* >user_dir_dir>Janus>Palter>empty		

Command
Author-Maintained Library
Peter B. Bishop
545 Technology Square
Room 536, Ext. 6213
03/26/72

CHAPTER 1

TECO

TECO (Text Editor and COrrector) is a character oriented text editor modelled after the TECO in general use on the Digital Equipment Corp. PDP-10, which was originally written at M.I.T.'s Artificial Intelligence project. TECO allows many simple editing requests, macro definitions, iterations, and conditional statements. These permit the user to do simple "manual" editing of ASCII files or to write complex macros which do "automatic" editing. Although this implementation is modeled after the TECO in general use, many new commands and features have been added that make the macro facility really powerful and easy to use. Some of the additions include adding if ... then ... else ... statements, allowing the contents of Q-registers to be used as quoted strings, allowing numeric and string arguments to be passed to macros, and allowing macros that reside in files to be called directly from TECO.

1.1 GENERAL DESCRIPTION

TECO is basically a character oriented editor, whereas editors like edm and qedx are line oriented editors. In edm and qedx it is only possible to position the pointer to the beginning of the line. The pointer is then considered to point at the whole line. These editors then supply commands (the substitute or change command) to edit the current line. In TECO such a complicated command is unnecessary because the pointer can point between any two characters in the buffer. The fundamental character oriented commands are insert, delete, search, and moving the pointer. With these commands it is very easy to do what would be complicated operations in a line oriented editor. The concept of a line as an important entity is not unknown in TECO, however. There are many commands that can be line oriented. These are the L, T, K, X, and S commands.

TECO reads command lines from the user's console (actually it reads from the stream "user_input") line by line until a line ending with "$" is typed. Execution of the complete command string is started when this last line is read. TECO will type "*" when it is waiting for a new command string.

1.2 ENTERING TECO

TECO may be called from the Multics command level by the Multics command:

teco -pathname- or TECO -pathname-

If pathname is specified, TECO automatically reads in the file by effectively executing the string "EI/pathname/J" upon entry. If no pathname is specified, the buffer will initially be empty. To create a new file, one should enter TECO (without specifying a pathname) and then use the "I" request to insert text.

(See section 3.1.8.4(5))

1.3 EXITING FROM TECO

One may exit from TECO by typing the EQ command (followed by "$" and a newline).

1.4 TECO DEFINITIONS

A. TECO uses four storage areas:

(1) The buffer is the area where text to be edited is examined and modified. At all times it contains a (possibly null) character string. There is a pointer into the buffer, denoting the current position. This pointer does not point to a character; it points between two characters. The pointer may assume any value between 0 and Z, where "Z" is the number of characters currently in the buffer. 0 indicates that the pointer is to the left of the first character, and Z would represent the position to the right of the last character in the buffer. The value of the pointer is represented by ".".

(2) Commands to TECO are written as a character string which is read into the Command String Area. TECO interprets the characters in the command string as a series of commands. Upper and lower case letters may be used interchangeably in commands.

(3) The Q-Registers are locations for storing either numeric quantities or strings of text for later use. Each Q-Register is designated by a single character name. There are 95 Q-Registers, one for each printing ASCII character. Each Q-Register may contain a positive or negative integer or a character string.

(4) The Q-Register pushdown list is a last-in-first-out (LIFO) list which may be used to temporarily store the contents of a Q-Register. It is cleared (i.e., the contents are lost) every time one returns to command level, i.e., a "⊠" is typed.

B. TECO uses numeric expressions for many of its operations. These may consist of any combination of decimal or octal numbers, the unary operator "-", the binary operators "+", "-", "*", "/", "|" (boolean or), "&" (boolean and), and the special valued commands and symbols. All operators are of equal precedence and expressions are evaluated from left to right. Note, however, that parentheses may be used in their normal manner. Spaces are ignored (except to terminate decimal numbers). If two numeric quantities are given with no operator between them, the default operator "+" is used. Note that a string of digits followed immediately by a "." is interpreted as an octal rather than a decimal number. Division using the "/" operator is integer division, i.e., the remainder is ignored. The special symbols allowed in an expression at any point are:

B (Beginning) equivalent to 0
Z equivalent to the number of characters in the buffer
. (pointer) equivalent to the number of characters to the left of the pointer, i.e., the current value of the pointer.

There is another special symbol related to the symbols above and this is the H (wHole) symbol. This symbol is equivalent to "0,Z". It is the only symbol in TECO that has two values. It is useful for referring to the whole buffer.

Commands which return values may also be used in expressions, but they may not appear immediately to the right of an operator. This is because the command will assume that everything to its left is part of one of its arguments. If a command appears within parentheses, it will assume that its arguments are entirely between the last "(" and the command. Therefore a command will not read parts of an expression wich are outside the parentheses in which it appears.

The plus and minus binary operators (excluding the unary minus) assume a right operand of 1 if none is given.

EXAMPLES

Assume that the current value of the pointer is 500.

	expression	value
(1)	(7 12)/3	= 6
(2)	9+	= 10
(3)	b-	= -1
(4)	-	= -1
(5)	4+8/2	= 6
(6)	101.	= 65

```
(7)  3|10                          = 11
(8)  1++++ ++ +++     +            = 11
(9)  9*-2                          = -18
(10) 9*--2                         = 18
(11) .10                           = 510
(12) 10.                           = 8
```

C. Quoted strings are strings of text delimited by a quoting character. The quoting character may be any character not contained in the string except a letter or a digit. The contents of a Q-register may be used as a quoted string if the letter "q" followed immediately by the letter specifying the Q-register is typed instead of the first quoting character.

(See section 3.1.4)

EXAMPLES

(1) "hello"
(2) /This is a quoted string/
(3) ,This string is delimited by the comma character and contains 2 new-line characters.
,
(4) q1

1.5 ERROR MESSAGES

TECO types out error messages in one of two modes, long or short. Short error messages are less than 9 characters long while long error messages are less than 50 characters long. The default mode is short. To change the error mode TECO is using, give the following Multics command:

```
teco$teco_error_mode long
```
or
```
teco$teco_error_mode short
```

If a short error message, such as "/: ?" cannot be understood, the following Multics command will type out the long error message:

```
teco_error "/: ?"
```

The above holds for all error messages except those informing the user that a file could not be found.

1.6 IMPLEMENTATION RESTRICTIONS

The maximum number of characters allowed in a Q-register is 262143. The maximum number of characters allowed in a quoted string is 262143, as is the maximum number of characters in a TECO command line. Note that these sizes are all one segment long. When the Multics segment size changes, these restrictions

will also change. The maximum number of items in the pushdown list is 20. The maximum depth of macro calls is 20. The maximum depth of parentheses is 20.

1.7 LEARNING TECO

This manual contains three logical sections. In the first section (Chapter 2) commands are described which:

(1) read and write files
(2) examine text within a file
(3) make deletions and insertions
(4) search for strings of text

Examples of using the commands are given at the end of the chapter. After reading Chapters 1 and 2 the reader should be able to use most of the common editing requests.

In the second section (Chapter 3), more sophisticated TECO commands are described, including use of Q-registers, macros, iterations, conditionals. The commands listed in chapter 3 transform TECO from just another editor to one of the most powerful general purpose text editors in existence.

The third section contains a summary of all the TECO commands in alphabetical order. This is intended to be used as a reference section.

CHAPTER 2

BASIC TECO COMMANDS

2.1 The most general form of a TECO command would be:

m,nX/string/

where m and n are optional numeric arguments, X is the command to be executed, and /string/ is a quoted string. In most cases the command will be just one character, though in some cases it may be two characters. Not all of the commands take arguments. Those that do generally have default values for missing arguments. Only a few commands expect quoted strings. The string must not be omitted if the command expects one. Some commands also return values, this will be discussed in Chapter 3.

The letter chosen for a command generally has some mnemonic meaning, which is indicated in the description of the command. Unfortunately, TECO has a fairly long history, having originally been developed for editing paper tapes, and so some of the mnemonic meanings are almost lost now. As many commands as one wishes may be typed at a time. Execution of the commands will not start until after the "$" followed by a newline character is typed. Spaces may be inserted anywhere (except in the middle of numbers) and newline characters may be inserted anywhere <u>except</u> between a command and its arguments.

Remember that upper case and lower case letters may be used interchangeably as commands.

2.1.1 <u>ENTERING TECO</u>

See section 1.2.

2.1.2 <u>READING A FILE</u>: - EI (<u>E</u>xternal <u>I</u>nput)

EI/pathname/ — reads in the file specified by pathname, which is assumed to be a standard Multics pathname. The contents of the file are inserted in the buffer at the current pointer position and then the pointer is moved to the right of the text just inserted.

2.1.3 <u>WRITING A FILE</u>: - EO (<u>E</u>xternal <u>O</u>utput)

EO/pathname/ — writes the contents of the buffer to the file specified by pathname. This command takes arguments similar to the "T" command; it writes out that part of the buffer which

would be typed by "T". Note, however, that if no arguments are given, "EO" assumes "B,Z" as the default rather than "1".

Note: The pointer is never moved by the "EO" command.

2.1.4 TYPING THE BUFFER - T (Type)

T		equivalent to "1T"
nT	+n	types out the string of characters beginning at the current pointer position and terminating after n newline characters have been encountered. T types out the rest of the current line, and 2T types out the rest of the current line and the next line. The last character typed by T is a newline unless there aren't that many lines in the file.
	-n	types out starting just after the (n+1)th newline to the left of the pointer and finishing at the pointer. 0T types out the beginning of the line up to the current pointer. Usually two T commands are given at once, such as 0TT, which types out the entire line the pointer is in. When 0T is useful, the last character it types out is not a newline. -T types out the previous line and the beginning of the current line. If the pointer is at the beginning of a line, -T types out the previous line, the newline at the end of that line, and nothing more.
m,nT		Types out the (m+1)th through the nth characters of the buffer.

Note: The pointer is never moved by the "T" command.

2.1.5 MOVING THE POINTER - J (Jump), C (Characters), R (Reverse), and L (Lines)

nJ	Moves the pointer to the right of the nth character in the buffer, i.e., sets "." to the value of n. If n is not specified, 0 is assumed. That is, the pointer is moved to the left of the first character in the buffer.
nC	Moves the pointer n characters to the right of its current position (equivalent to .+nJ). If n is omitted, 1 is assumed.
nR	Like nC except it moves the pointer to the left (equivalent to -nC). If n is omitted, 1

is assumed.

nL +n Moves the pointer to the right, stopping after it has passed over n newline characters. If n is omitted, 1 is assumed. L moves the pointer to the beginning of the next line.

-n Moves the pointer to the left, stopping after it has passed over n+1 newline characters and then moving it to the right of the last newline character passed over. 0L moves the pointer to the beginning of the current line, and -L moves the pointer to the beginning of the previous line.

:L (See section 3.1.1)

2.1.6 DELETING TEXT - D (Delete) and K (Kill)

nD Deletes n characters. If n is positive the characters are deleted to the right of the pointer. If n is negative the characters are deleted to the left of the pointer. If n is omitted, 1 is assumed.

K Takes arguments like the "T" command except that it deletes that text which "T" would type. The pointer is moved to where the deletion took place. If no arguments are specified, "1K" is assumed.

+n deletes all the characters beginning at the current pointer position and terminating after n newline characters have been encountered. K deletes the rest of the current line and the newline character at the end of the line, while 2K deletes the rest of the current line and the next line.

-n deletes all the characters starting just after the (n+1)th newline to the left of the current pointer and ending at the current pointer. 0K deletes the beginning of the current line without deleting the newline character at the end of the previous line. -K deletes the previous line and the beginning of the current line. To ensure that only the previous line is deleted, the command sequence "0L-K" can be used.

m,nK Deletes the (m+1)th through the nth characters of the buffer.

2.1.7 INSERTING TEXT - I (Insert)

I/text/	Inserts the text of the quoted string at the current pointer position and moves the pointer to the right of the inserted text.
nI	Inserts the character whose ASCII code value is n. It moves the pointer to the right of the inserted character.
:I	(See section 3.1.3.2)

2.1.8 SEARCH FOR TEXT - S (Search)

S/string/	equivalent to 1S/string/
nS/string/	Searches for the nth occurrence of the quoted string. If n is positive the text is searched from the current pointer through the end of the buffer for the nth occurrence of the string. If found, the pointer is set to the right of the matching string. Otherwise the pointer is not moved and an error message is typed. If n is negative the text is searched from the current pointer position to the beginning of the buffer for the nth occurrence of the quoted string. The pointer is set to the left of the matched string. If the string is not found the pointer is not moved and an error message is typed out.
m,nS/string/	Instead of searching the entire buffer for n occurrences of the quoted string, only m lines from the current pointer are searched. If m is positive, the only part of the buffer that will be searched will be from the current pointer to just after the mth newline character after the current pointer. If m is 0 or negative, the only part of the buffer that will be searched will be from the current pointer to just after the (m+1)th newline before the current pointer. 1,1S/text/ will only search the rest of the current line. 0,-1S/text/ will only search the beginning of the current line. If m is less than or equal to 0, n must be negative. If m is greater than zero, n must be positive.
:S/string/	(See section 3.1.1)

2.1.9 TYPING OUT VALUES - = (Equals)

n= or m,n= types out the decimal value of all the arguments separated by spaces and ending with a newline.

2.1.10 LEAVING TECO - EQ (External Quit)

EQ returns to the caller of TECO (e.g., Multics command level)

Note: don't forget to do an EO command before the EQ.

2.1.11 RESTARTING TECO AFTER A "QUIT"

If one "quits" out of TECO in order to abort a command string, one may use the "program_interrupt" ("pi") command to restart TECO. It will not abort the entire command string; only those commands which have not yet been executed. The current command is aborted if the effect of doing so would be identical to that of not starting the command in the first place. TECO keeps track of what it is doing, so that if the sequence:

```
(quit)
program_interrupt (or "pi")
```

is given, it will not abort the current operation if it would leave TECO in an inconsistent state. In other words, the sequence will only interrupt between TECO commands, not in the middle of a command.

At times it is desirable to get around this feature. When doing an "EO", for instance, TECO will not allow the user to "pi" back to TECO command level once the EO has started until it has completed writing the file. To get around this, one should type:

```
(quit)
teco$abort          or          TECO$ABORT
```

When TECO$ABORT is called, the most recent invocation of TECO aborts its current operation without checking for consistency of states. Note that TECO will be in a consistent state whenever it actually accesses a file, and so there should be no problems encountered if this feature is used to get out of a runaway "E" command. Under other circumstances, however, it is wise for the user to type:

```
-5t5t
```

to make sure that things are OK. Except for the case of a runaway EO command, this feature is probably totally unnecessary in normal use.

2.2 STAND-ALONE EXAMPLES

2.2.1 ENTERING TECO

a) teco source.pl1 — enter TECO and read in the file source.pl1 from the working directory.

b) TECO <x>y>z>a.ec — enter TECO and read in the file specified.

c) teco — enter the buffer initially empty.

2.2.3 READING A FILE

a) EI/source.pl1/ — Insert the text contained in source.pl1 at the current point in the buffer.

2.2.3 WRITING A FILE

a) EO/new_source.pl1/ — Write the whole buffer out into new_source.pl1.

b) .,zEO/bottom/ — Write out the buffer from the current pointer to the end into the file "bottom".

c) 2EO/lines/ — Write out two lines starting at the current pointer position to the file "lines".

2.2.4 TYPING TEXT

a) 2T — Type out from . to the end of the next line.

b) 0T — Type out the current line from its beginning to ".".

c) 0TT — Type out all of the current line.

d) 25,100T — Type out the 25+1 (26th) through the 100th character of the buffer.

2.2.5 MOVING THE POINTER

a) J — Position the pointer at the beginning of the buffer.

b) ZJ — Position the pointer at the end of the buffer.

c) L — Position the pointer at the beginning of the next line in the buffer.

d) 0L — Position the pointer at the beginning of the current line.

e) -L — Position the pointer at the beginning of the current line.

f) R — Back up the pointer by one character position.

g) 812-388C — Move the pointer ahead 812-388 (424) character positions.

2.2.6 DELETING TEXT

a) 19,22K — Delete the 19+1 (20th) through the 22nd character of the file. Set the pointer to 19.

b) 19J 3D — Move the pointer to the right of the 19th character and then delete the next three characters (20-22).

c) HK — Delete the whole buffer.

d) -D — Delete the character just to the left of the pointer.

2.2.7 INSERTING TEXT

a) I/abc
/ — Insert the line "abc" followed by a new-line character at the current pointer position.

b) I.abc. — Insert the string "abc" without a new-line character.

c) 65I — Insert the character with ASCII code 65 ("A") at the current pointer position.

2.2.9 TYPING VALUES

a) Z =

Type out how many characters are in the buffer.

b) Z,.=

Type out how many characters are in the buffer followed by the current pointer position.

c) =

Type just a blank line.

d) Q6+53 =

Type out 53 plus the value contained in Q-register 6.

2.2.8 SEARCHING FOR TEXT

a) J S/Hello/

Position the pointer just to the right of the first occurence of the string "Hello" in the buffer.

b) ZJ -S"Hello"

Position the pointer just to the left of the last occurence of the string "Hello" in the buffer.

c) J 3S"*
"

Position the pointer just after the third occurence of a line ending with a "*".

d) J 1,1S/hello
/

Position the pointer just after the first line in the buffer if it ends in "hello". If the first line does not end in "hello" type out an error message.

2.3 EXAMPLES OF BASIC EDITING REQUESTS

Note: In the following examples, underlined text is produced by TECO.

TECO abc.pl1	Enter TECO and read in the segment abc.pl1.
5LT$	Move to the 6th line and type it out.
dcl a fixed bin; S/a/-DI/b/0LT$	Change the "a" to a "b" and retype the line.
dcl b fixed bin; S/dcl d/0LKT$	Search for the declaration of d and delete the line that contains it. Then type out the next line.
dcl f fixed bin; KI/dcl g char(2); /$	Delete this line and then insert a declaration of g.
EO/abc.pl1/EQ$	Write the edited text out to the file and then return from TECO.

CHAPTER 3

ADVANCED TECO COMMANDS

3.1 In Chapter 2 the general form of a TECO command was given. Some items were left out, however. The actual format is:

m,nXq/string1//string2/.../stringn/

The q indicates a Q-register on which the command is to act.

It should also be noted that more than one string may be given. Although no TECO command currently accepts more than one quoted string, a macro may be called with multiple string arguments which may be retrieved inside the macro by the :X command.

In Chapter 1 we specified that expressions may be built from numbers, special valued commands and symbols. Examples of valued commands will be given in Chapter 3. Care should be taken to notice that commands with values may appear only on the left side of the first operator, or within parentheses. Otherwise the part of the expression preceding the command will be considered an argument to the command.

3.1.1 The effect of many commands may have their function changed by preceding the command with a ":". The colon has no fixed meaning - it is defined for each command individually. The following commands given earlier may be used as follows.

:Iq/string/ or n:Iq like the I command except that the specified string is inserted into Q-register q. The former contents of Q-register q are lost.

n:L Equivalent to nLR. Thus TECO moves to the end of the line rather than the beginning.

:S/string/, n:S/string/, or m,n:S/string/ like S except that it returns a value. The value is 0 if the search fails and -1 if it succeeds. Even if the search fails, TECO continues execution.

:T/string/ types the specified string on the user's console.

:vw (See section 3.1.10)

:x (See section 3.1.8.3)

3.1.2 Numeric Q-Registers

Q-Registers may be used, as mentioned in section 1.4, to hold numeric values. These values may be used in expressions which are arguments to other commands.

3.1.2.1 SAVING A VALUE - U (Update (or what comes after Q?))

Uq	sets Q-register q to a very large positive number.
nUq	sets Q-register q to n.
m,nUq	sets Q-register q to n and returns m as its value.

3.1.2.2 READING Q-REGISTERS - Q (Q-Register (Don't ask me why "Q"))

Qq	Return the number stored in Q-register q as the value. Note that Q is not really a command - it is a special symbol (as in section 1.4.2). Thus, in the expression "5+Q3" the "5+" is not considered an argument to Q; the result is the sum of Q3 and 5. Note if Q-register q contains text, the length of the text in characters is returned.

3.1.2.3 INCREMENTING Q-REGISTERS - % (You figure out the mnemonic)

%q	Add 1 to Q-register q and return the new number as the value. Q-register q may not contain text. Note that %, like Q, is a special symbol, not a command.

3.1.3 Text Q-Registers

Q-Registers may also be used to hold character strings. They may be used to move text from one place in the buffer to another, to save command lines for execution as macros, or to provide quoted strings for commands which expect them.

3.1.3.1 EXTRACTING TEXT TO A Q-REGISTER - X (eXtract)

Xq	takes arguments like the "T" command, but copies the text that T would type into Q-register q. The former contents of Q-register q are deleted. The text is not

deleted from the buffer and the current pointer is not moved.

nXq +n copies all the text from the current pointer to just past the nth newline character to the right of the pointer into Q-register q. X1 copies the rest of the current line including the newline at the end of the line into Q-register 1. 2Xa copies the text on the rest of the current line and all of the next line into Q-register "a".

-n copies all the text from just to the right of the (n+1)th newline that is to the left of the current pointer to the current pointer into Q-register q. 0X/ copies the beginning of the current line into Q-register "/". No newline characters will be put into Q-register "/". -Xa puts the previous line and the beginning of the current line into Q-register "a".

m,nXq copies character number (m+1) through character number n into Q-register q.

:X (See section 3.1.8.3)

3.1.3.2 INSERTING TEXT DIRECTLY INTO A Q-REGISTER - :I (Insert)

:Iq/string/ This command is identical to the normal "I" command except that the text is inserted into Q-register q rather than the buffer. The former contents of Q-register q are deleted. The main text buffer is not affected.

n:Iq is like :I except that it puts the character corresponding to n into the Q-register q.

3.1.3.3 GETTING TEXT FROM A Q-REGISTER - G (Get)

Gq inserts the text contained in Q-register q into the buffer to the left of the current pointer. If the Q-register contains a number, the decimal representation of the number is inserted.

3.1.4 Obtaining quoted strings from Q-registers.

Whenever TECO expects a quoted string, it is possible to indicate that the string is in a Q-register. Normally letters

and digits are considered illegal quoting characters. If, however, the letter "Q" is found where a quoted string is expected, the next character after the Q will be considered a Q-register name. Whenever a quoted string is retrieved by any command, it is loaded into Q-register ". As an example, SQ" immediately after another search will search again for the same string. This notation is illegal if the specified Q-register contains a number.

3.1.5 The Q-register pushdown stack.

There is one Q-register pushdown stack (not one per Q-register) in which the values of Q-registers may be saved. It is organized as a pushdown (Last-In, First-Out) list. It is emptied every time TECO waits for a new command string, i.e., a "⊠" is typed.

3.1.5.1 PUSHING A VALUE ONTO THE STACK - [(opposite of])

[q — pushes the current value of Q-register q onto the top of the stack. The Q-register is not affected.

3.1.5.2 POPPING A VALUE FROM THE STACK -] (opposite of [)

]q — pops the top value on the stack into Q-register q. The previous contents of the Q-register are lost. It is an error to do a "]" command if the stack is empty.

3.1.6 Loops

TECO has the ability to execute a command string repeatedly, much as Fortran or PL/1 provides "do-loops".

3.1.6.1 BEGINNING A LOOP - < and > (opposite of each other)

< — is equivalent to n< except that n is set to a very big number which is for all practical purposes infinite.

n< — causes TECO to take note of the fact that a loop is beginning. The value of n and the position of the "<" in the command string are saved.

> — causes execution to return to just after < if the string has not yet been executed n times.

n<...> this causes the string between the angle brackets to be executed n times.

3.1.6.2 TERMINATING A LOOP BEFORE n EXECUTIONS - ; (think about it)

n; if n is less than 0 then nothing is done. Otherwise execution of the current loop is aborted and TECO skips to just after the closing >. If n is not specified, the result of the most recent S command is used (terminate loop if search failed). The ; command may not appear outside of a loop.

3.1.7 Goto's

TECO provides the ability to transfer control to a different part of the command string.

3.1.7.1 GOTO - O (gOto)

O/string/ causes TECO to search the current macro (or, if we are not in a macro, the command line) for the string "!string!". If it is found, TECO begins interpreting commands just after the label found. If not found, but execution is currently in a macro, the search is repeated in the previous execution level, i.e., the caller of the macro. This is repeated until TECO has checked all the way down to the command line typed by the user. Note that although TECO may exit a macro using an O command, it may not use that command to exit a loop. Only ";" may be used to terminate a loop.

3.1.8 Macros

TECO has the ability to execute strings of text (macros) other than those read from the user's console. The associated commands are:

3.1.8.1 EXECUTING A MACRO IN A Q-REGISTER - M (Macro)

Mq causes the contents of Q-register q to be executed as a command string. Note that if the M command is given any numeric arguments they are passed to the first command inside the macro. String arguments may be fetched

by the :X command.

3.1.8.2 EXECUTING A MACRO IN A FILE - EM (External Macro)

EM/string/ is just like the M command except that the command string is found in a file named "string.teco". This file is looked for in three places: 1) the working directory, 2) the user's login directory, 3) the TECO library. TECO$teco_ssd is a command that takes a single argument, an absolute pathname for a search directory, and changes the TECO search rules so that instead of searching the user's login directory, the search directory specified is searched. TECO$teco_search is an external subroutine (See section 3.1.16) that follows the same search rules used to find a macro.

3.1.8.3 OBTAINING A STRING ARGUMENT TO A MACRO

:Xq causes TECO to suspend execution of the current macro, return to its caller to fetch a quoted string into Q-register q, and then restore the macro that was being executed. Note that each :X command in a macro fetches another quoted string. Note that the U command(s) should be the first command in a macro if one wishes to fetch numeric arguments in a macro.

3.1.8.4 A few notes about macros:

1) Loops may not cross macro boundaries, i.e., a loop may not start in one macro and end in another. This does not, however, prohibit the M command from being used within a loop.

2) A macro may modify itself if it is in a Q-register. Note, however, that the current invocation of the macro will not be affected; only future accesses to the Q-register. If the macro is invoked by the EM command, the results of modifying the file are hard to predict: TECO reads the command string directly from the file.

3) When a macro is invoked by the EM command it should be noted that the name of the macro will be found in the Q-register named ". Thus one can put several macros in one segment with the first command in the segment being OQ". (Don't forget to put all the appropriate names on

the segment.)

4) If an M or EM command is given as the last command in one macro, the command is interpreted as a goto rather than a call. Thus one may do unlimited M's in this manner although there is an implementation-defined limit to the depth of calls.

5) When TECO is entered, a macro named "start_up" is searched for. If it is found, the arguments to TECO are put onto the pushdown stack and the start_up macro is executed. If no start_up macro is found, the string "EI/filename/J" is executed, where filename is the first argument to TECO. At the present time there is a start_up macro in the TECO library. When the start_up macro is called, the first thing on the pushdown list is the number of arguments TECO was called with. The remaining items in the list are the actual string arguments to TECO going from left to right on the command line.

3.1.8.5 CODING CONVENTIONS FOR MACROS

Since there are only a small number of Q-registers (95), each with a one character name, there are serious problems in writing a set of macros that are compatible. A set of macros become incompatible if one macro uses a Q-register for long-term storage that any other macro uses at all. There are two ways this effect can be combated. First, by establishing certain coding conventions, and second, by use of a documented macro library. Probably the most important coding convention is the specification of which Q-registers may be used inside a macro for temporary storage. Many macro writers now use the ten Q-registers 1,2,3,4,5,6,7,8,9, and 0 for temporary storage. If one macro wants to call another macro that will clobber one of these registers, the calling macro may save the value of the Q-register in the pushdown list and then restore it after the other macro has been called.

Fortunately, calling a macro is a very inexpensive operation in TECO if the macro is in a Q-register. The EM command is much more expensive, however. This leads to the practice of creating a macro in a macro library that will only load a Q-register with a useful macro. When the user realizes that he wants the macro, he gives the EM command that will load the macro he wants into a Q-register, where he may then call it whenever he wishes. It now becomes necessary to have coding conventions that specify which registers may be loaded permanently with macros. Since it should be easy to type the macro names, the lower case alphabetic letters should be used for this purpose. Sometimes a macro will use a Q-register for long-term storage. If the user will not have

to type the name of this Q-register, names that must be escaped on a 2741 are good, otherwise other special characters may be used. This leaves the upper case alphabetic letters entirely to the user for him to use to store intermediate results in editing. Also the special characters "-", ",", ".", "/", space, tab, and newline should be reserved for the user since these are all lower case letters on both a 2741 and a Model 37 teletype.

An extremely useful feature of TECO is that the last quoted string is loaded into Q-register ". To allow this to continue to be useful, all macros should make sure that Q-register " either contains the last quoted string argument to the macro, if there are any, or contains what it contained before the macro was called. Q-register " can be saved on the pushdown list on entry to a macro and then restored just before leaving the macro. Use of the pushdown list is very inexpensive.

3.1.8.6 RELATIVE COSTS IN TECO

TECO stores the buffer in two pieces. The first piece, all the characters from the beginning of the buffer to the current pointer, is stored at the beginning of one buffer segment, while the second piece, all the characters from the current pointer to the end of the buffer, is stored at the end of another buffer segment. An insert merely adds text to the end of the first buffer segment and increases the number of legitimate characters in the first buffer segment. A D or X command merely changes the number of legitimate characters in one of the buffer segments. In order to move the pointer, a string copy from one buffer segment to the other must be performed. It does not matter to TECO which direction the pointer is moved, although a reverse search is somewhat slower than a forward search, since the Pl1 index built-in function can only be used for a forward search.

Any operation that does not move text is less expensive than an operation that does move text, where the cost of the operation that does move text is proportional to the amount of text moved. For the most part, performing input or output is the major cost involved in editing. This cost can be decreased by using more sophisticated commands, such as loops or macros, and performing the same editing operation with fewer interactions. The cost of i/o operations is comparable to a medium length search (5,000 characters).

Each text Q-register is presently kept in its own segment. This means that if a start_up macro loads many Q-registers with macros, then entering TECO for the first time in a process will be somewhat slow since all these segments must be created. TECO has its own segment manager (get_temp_seg_), that allows it to re-use segments without calling hardcore to create and delete segments when the values of Q-registers are changed. Whenever a

string is quoted, or a Q-register loaded with text, a new segment is retrieved from get_temp_seg_ and loaded with the value. If the string that is being loaded into the Q-register is in another Q-register, the new Q-register is just made to point to the same copy of the text in the first Q-register. :IAQB is therefore a very simple operation, as are [(Push) and] (Pop). The feature of keeping the last quoted string in Q-register " lets the user take advantage of this scheme.

If the user wants to write a macro that must do some editing on another file, it is much cheaper if he saves the value of . and Z-. , inserts the text to be edited, edits it, writes it out or copies it into a Q-register, and then deletes what he was just editing from the buffer. The net change to the buffer by all these operations is zero, but the text that the user was editing was never moved. This method is much cheaper than storing the entire buffer in one Q-register, the value of the pointer in another, and then using the buffer for the editing within the macro.

There are four ways to transfer control in TECO, by the > command, the ; command, the " or :' command, and the O command. Of these the > command is the fastest since TECO already knows exactly where to transfer to. The ;, ", and :' commands are next, since they merely search from where they are forward. Although the > command and the ; command cannot change macro levels, the ", and :' commands can. This adds a small expense. The ;, ", and :' commands all have to check so that a ; command will completely skip over another nested loop and look beyond it for a >. Similarly the " transfer will skip over nested if statements, as will the :' command. Usually the matching ' or > is not far from the transfer, so this only causes a short search. O is the most general and most expensive transfer of control in TECO. It must search the entire macro from the beginning, then the entire macro that called the present macro, etc. until it finds it or finishes searching the command line and gives an error. Although this is the most expensive transfer, its cost is proportional to the distance of the label from the beginning of the macro.

3.1.9 Conditionals

TECO has the ability to conditionally execute strings. The " command corresponds to the PL/1 statement "if ... then do;" The ' command corresponds to the PL/1 statement "end;". " and ' are matched much like (and) and may be nested. The letter following the " determines what test will be made.

3.1.9.1 NUMERIC COMPARISONS - "E (Equals), "N (Not equal), "G (Greater than),"L (Less than)

m,n"E	if m=n then execution continues; otherwise execution skips to just after the corresponding '.
n"E	identical to n,0"E.
m,n"N	like m,n"E except it tests for m¬=n.
n"N	identical to n,0"N.
m,n"G	like m,n"E except it tests for m>n.
n"G	identical to n,0"G.
m,n"L	like m,n"E except it tests for m<n.
n"L	identical to n,0"L.

3.1.9.2 TESTING FOR A SYMBOL CONSTITUENT - "C (symbol Constituent)

n"C	if n is the ASCII code for either a letter, a digit or one of the characters ".", "_", or "$" then execution continues; otherwise execution skips to the corresponding '.

3.1.9.3 TERMINATING A CONDITIONAL DO - ' (matches ")

'	is ignored when executed in normal execution. It is used to close a conditional statement.
:'	This command causes a transfer to the next ', just as a 1"e does. Since this command looks like a ', it can serve to close a conditional statement. This is useful if an if ... then ... else ... statement is desired. The if expression is a " statement, the then expression is terminated by the :' command and the else expression is terminated by the ' command.

3.1.10 Reading input from the user's console. - VW (V then Wait for input)

VW	does a V command (presently does nothing on Multics) and then reads one character from the user's console. The ASCII value of the character is returned as the value of the command. Note that Multics escape/kill

processing is not effected because only one character is read at a time.

:VWq — does a V command and then reads one line from the user's console. The line is put into Q-register q. The newline is the last character read in.

3.1.11 Passing a command to the command processor - EC (External Command)

EC/string/ — passes the specified string to the Multics command processor for execution.

3.1.12 Examining a character in the buffer - A (Ascii)

nA — The ASCII code for the (.+n)th character in the buffer is returned as the value of the command. n must be specified. (Note that 1 indicates the character just to the right of the current pointer, 0 indicates the character just to the left.)

3.1.13 Tracing command execution - ? (Asking what happens)

? — turns tracing on. When tracing is on, each command executed by TECO is printed on the user's console just before it is executed.

?? — turns off tracing.

3.1.14 Translating numbers to ASCII and vice versa - ¢ (You figure it out)

¢ — reads the decimal number found to the right of the current pointer and returns its value as the value of the command. The pointer is moved to the right of the number. The number may be signed and may be preceeded by any number of blanks or tabs. It is an error if no number is found.

n¢ — inserts the decimal interpretation of n into the buffer to the left of the current pointer.

m,n¢ — inserts the decimal interpretation of m into the buffer to the left of the current

pointer. The interpretation is padded on the left to be at least n characters wide.

3.1.15 Null command - W (Wipe out?)

W — does nothing. It is most useful for throwing away unneeded numeric arguments.

new_line — has the same effect as W.

$ — has the same effect as W.

3.1.16 Subroutines

TECO has the ability to communicate with programs written by users on Multics. In particular, TECO macros can get information from programs written to interface with those macros.

3.1.16.1 CALLING A SUBROUTINE - ES (External Subroutine)

m,nESq/prog_name/ — prog_name is a relative pathname of a segment with an optional entry point name. The search rules are used to find the segment. The entry point specified is called in the following way: dcl prog_name ext entry(char(*) aligned, fixed bin, fixed bin, fixed bin); call prog_name (Q_register, m, n, value); q must be a text Q-register. prog_name will be called with the Q_register as the first argument. If this string is changed, the Q_register will be changed. This command has a numeric value that can be set by changing the fourth argument to prog_name. This is initialized to zero. If the ES command is called with one numeric argument, it will be passed as the second argument to prog_name and the third argument will be a very large positive number. If no numeric arguments are given to ES, both the second and third arguments to prog_name will be very large positive numbers.

3.1.16.2 A few notes about subroutines:

1) TECO$teco_search is an external subroutine that can be used. It takes a Q-register that contains the name of an external macro followed by a blank followed by enough characters to hold an absolute pathname in the whole register. TECO$teco_search searches for the macro (it adds ".teco" to the name) using the TECO search

rules. If it is not found it returns with the value 0. If it is found, it changes the Q-register to hold an absolute pathname to the macro followed by enough blanks to fill up the Q-register, as long as it is not longer than 256 characters. The value returned is the number of non-blank characters at the beginning of the Q-register.

2) TECO$teco_no_ES is an entry point in TECO that is very similar to the main entry point of TECO, except the ES command is not an implemented feature of TECO$teco_no_ES. This entry point is meant to be used by subsystems that cannot allow a user to make a call to an arbitrary procedure.

3.2 EXAMPLES OF MACROS

3.2.1 A WRITING MACRO

This macro writes out the entire buffer into a file whose name is in Q-register *. The file being edited can be changed merely by doing :i*/new_name/.

EOQ*	This macro assumes that the name of the file we are editing is in Q-register *. It writes out the entire buffer into this file.

3.2.2 A RESTART MACRO

This macro zeroes out the buffer, changes Q-register * to be a new file name and reads the file into the buffer.

```
:x* hk eiq*j
```

:X*	This macro takes one string argument and loads it into Q-register *.
HK	Since we are restarting the editing, we delete all the text in the current buffer.
EIQ*J	We now read the new file into the buffer and put the pointer at the beginning of the buffer.

3.2.3 A START UP MACRO

This macro only uses the first argument to TECO. It treats it as a file name, loads it into Q-register * and reads the file into the buffer. It also loads the writing macro into Q-register w.

```
]1 :iw|eoq*| q1"n ]* eiq*j '
```

]1	Pop the top item off the pushdown list and put it into Q-register 1. This will be the number of arguments TECO was called with.
:iw|eoq*|	Load Q-register w with the writing macro given in Example 3.2.1.
q1"n	If the contents of Q-register 1 are not zero, then execute the following statements, otherwise transfer to the ' that ends the macro.

]*	Pop the first argument to TECO off the pushdown list and into Q-register *.
eiq*j	Read the file into the buffer and move the pointer to the beginning of the buffer.
'	This point is transferred to if there are no arguments given to TECO.

3.2.4 A SUBSTITUTE MACRO

This macro takes two string arguments. The first string argument is searched for, then it is deleted and the second string inserted.

```
:x1 :x2 sq1 -q1d g2
```

:x1	Load the first string argument into Q-register 1.
:x2	Load the second string argument into Q-register 2.
sq1	Search for the first string.
-q1d	Delete the first string when it is found.
g2	Replace the string found with the second string argument.

When the macro returns, Q-registers 1 and 2 contain the first and second strings, respectively. Q-register " contains the second quoted string.

CHAPTER 4

A TECO SUMMARY

NAME	MNEMONIC	SECTION	USE	EXPLANATION
a	Ascii	3.1.12	nA	The value of the command is the ASCII code for the (.+n)th character in the buffer.
b	Beginning	1.4B	B	The value of this symbol is always zero.
c	Characters	2.1.5	nC	Moves the pointer n characters to the right. If n is omitted, 1 is assumed.
d	Delete	2.1.6	D	deletes the one character to the right of the pointer.
			+nD	deletes n characters to the right of the pointer.
			-nD	deletes n characters to the left of the pointer.
ec	External Command	3.1.11	EC/command/	passes the string to the Multics command processor.
ei	External Input	2.1.2	EI/file/	reads the file into the buffer to the left of the current pointer.
em	External Macro	3.1.8.2	EM/macro_name/	searches for the file "macro_name.teco", first in the working directory, then the login directory, then the TECO library. If found, it executes it as a macro.
eo	External Output	2.1.3	EO/file_name/	writes out the entire buffer into the file specified.
			+nEO/file_name/	writes out the next n lines.
			(0 or-n)EO/file_name/	writes out the last n lines.
			m,nEO/file_name/	writes out the (m+1)th through the nth characters.
eq	External Quit	2.1.10	EQ	TECO returns to its caller after zeroing out all Q-registers.
es	External Subroutine	3.1.16	m,nESq/prog_name/	Calls subroutine progname with arguments Q-register q, m, n, value. The numeric value of the command is set by prog_name and the contents of Q-register q may also be changed.

NAME	MNEMONIC	SECTION	USE	EXPLANATION
g	Get Q-register	3.1.3.3	Gq	inserts the text contained in Q-register q into the buffer to the left of the pointer. If Q-register q contains a number, it is converted to a character string and inserted.
h	wHole	1.4B	H	This symbol is equivalent to 0,Z . It is the only symbol that has two values.
i	Insert	2.1.7	I/string/	inserts the quoted string to the left of the pointer.
			nI	n is the ASCII code for a letter that is inserted.
:i		3.1.3.2	:Iq/string/	inserts the quoted string into Q-register q.
			n:Iq	inserts the single character whose code is n into register q.
j	Jump	2.1.5	nJ	moves the pointer to the right of the nth character in the buffer. If n is omitted, 0 is assumed.
k	Kill buffer	2.1.6	K	deletes the rest of the current line from the buffer.
			+nK	deletes the next n lines from the buffer.
			(0 or-n)K	deletes the last n lines from the buffer.
			m,nK	deletes the (m+1)th through the nth characters from the buffer.
l	Lines	2.1.5	L	moves the pointer to the beginning of the next line.
			+nL	moves the pointer to the beginning of the next nth line.
			(0 or-n)L	moves the pointer to the beginning of the last nth line.
:l		3.1.1	:L	moves the pointer to the end of the current line.
			+n:L	moves the pointer to the end of the next (n-1)th line.
			(0 or-n):L	moves the pointer to the end of the last (n+1)th line.
m	Macro	3.1.8.1	m,nMq/string1//string2/.../stringn/	starts executing the text in Q-register q as a macro. m and n are numeric

NAME	MNEMONIC	SECTION	USE	EXPLANATION
				arguments to the first command in the macro. string1 through stringn are string arguments to the macro that can be retrieved with the :X command. EM also takes all these arguments.
o	gOto	3.1.7.1	o/label/	transfers control to just after !label! in the current macro, its caller, etc., or the command string.
q	Q-register	1.4C	Qq	the value of this command is the value of Q-register q if it is a numeric Q-register or the number of characters in Q-register q if it contains text. This command can also replace any quoted string if Q-register q contains text. The contents of the Q-register are used as the quoted string. (See also sections 3.1.2.2 and 3.1.4)
r	Reverse	2.1.5	R	moves the pointer one character to the left.
			nR	moves the pointer n characters to the left.
s	Search	2.1.8	S/string/	searches from the current pointer to the end of the buffer for "string", if found it moves the pointer to the right of the string.
			+nS/string/	searches for n occurrences of the string. Moves the pointer to the right of the nth occurrence.
			-nS/string/	searches for n occurrences of "string" from the current pointer to the beginning of the file. If found, it moves the pointer to the left of the nth occurrence.
			+m,+nS/string/	only searches from the current pointer to the beginning of the next mth line.
			(0 or-m),-nS/string/	only searches from the current pointer to the beginning of the last mth line.
:s		3.1.1		Takes arguments in all the ways S does, except that if S does not find the string it types out an error message and returns to TECO command level. :S does not. Instead, :S has the value -1 if the search succeeds and 0 if the search fails.

NAME	MNEMONIC	SECTION	USE	EXPLANATION
t	Type	2.1.4	T	Type out the rest of the current line on the console.
			+nT	Type out the buffer from the current pointer to the beginning of the next nth line.
			(0 or-n)T	Type out the buffer from the beginning of the last nth line to the current pointer.
			m,nT	Type out the (m+1)th through the nth characters of the buffer.
:t		3.1.1	:T/string/	Type out the quoted string on the console.
u	Update	3.1.2.1	Uq	sets Q-register q to a very large positive number.
			nUq	sets Q-register q to n.
			m,nUq	sets Q-register q to n and returns m as its value. This may be used inside a macro to get the numeric arguments to the macro.
vw	VieW	3.1.10	VW	When this command is executed, one character is read from the console. The ASCII code for the character read is the value of the VW command.
:vw		3.1.10	:VWq	Reads in an entire line from the console and puts it into Q-register q. The newline is the last character in the register.
w	Wipe out	3.1.15	W	This command does nothing. It is used for throwing away unwanted numeric arguments.
x	eXtract from buffer	3.1.3.1	Xq	loads the rest of the current line into Q-register q.
			+nXq	loads Q-register q with everything from the current pointer to the beginning of the next nth line.
			(0 or-n)Xq	loads Q-register q with everything from the beginning of the last nth line to the current pointer.
			m,nXq	load Q-register q with everything from the (m+1) character to the nth character.
:x		3.1.8.3	:Xq	loads Q-register q with the next string argument to the macro we are

NAME	MNEMONIC	SECTION	USE	EXPLANATION
				executing in.
z	Last Letter	1.4B	Z	This symbol's value is the total number of characters in the buffer. ZJ will move the pointer to the right of the last character in the buffer.
%	Increment	3.1.2.3	%q	If Q-register q contains a numeric value, this command increments the register by 1. The value of the command is the new value of the Q-register.
$		3.1.15	$	Throws away its arguments and does nothing.
newline		3.1.15	newline	Throws away its arguments and does nothing.
?	Whats happening?	3.1.13	?	Turns tracing on.
??		3.1.13	??	Turns tracing off.
¢	Number-character	3.1.14	¢	the value of this command is the decimal number immediately to the right of the pointer. It moves the pointer to just after the number. n¢ inserts the decimal representation of n to the left of the pointer. m,n¢ inserts the decimal representation of m to the left of the pointer. The representation is padded on the left to be at least n characters wide.
[	Push	3.1.5.1	[q	Pushes the contents of Q-register q onto the pushdown list.
]	Pop	3.1.5.2	]q	Pops the top element off the pushdown list and into Q-register q.
<	Begin a loop	3.1.6.1	<	This marks the place in the command string that will be transferred to by the > command. This loop can only be exited by the ; command. +n< This loop will only execute n times. It may be exited by the ; command.
>	End a loop	3.1.6.1	>	Transfer control to just after the last < command executed and decrement the loop count. If we have looped

NAME MNEMONIC	SECTION	USE	EXPLANATION
			enough times, this command does nothing. Nested loops are allowed.
; Terminate if positive	3.1.6.2	;	If the last :S command was unsuccessful, transfer to just after the next > and exit the present loop, otherwise do nothing.
		n;	If n is positive, transfer control to just after the next > command and exit the present loop, otherwise do nothing.
"C If Symbol Constituent	3.1.9.2	n"C	if n is the ASCII code for either a letter, a digit, ".", "_", or "$", do nothing, otherwise transfer to just after the next '.
"e If Equal	3.1.9.1	m,n"E	If m=n then do nothing otherwise go to just after the next '.
		n"E	if n=0
"g If Greater	3.1.9.1	m,n"G	if m>n.
		n"G	if n>0.
"l If Less than	3.1.9.1	m,n"L	if m<n.
		n"L	if n<0.
"n If Not equal	3.1.9.1	m,n"N	if m¬=n.
		n"N	if n¬=0.
' Matches "	3.1.9.3	'	marks the location a " command may transfer to. If executed as a command, it does nothing.
:'	3.1.9.3	:'	marks the location a " command may transfer to. If executed as a command, it transfers to just after the next '. If statements may be nested, but " characters in the command string are only matched with one ' character. (See section 3.1.9)
o gOto	3.1.7.1	o/label/	transfers control to just after !label!
! Label	3.1.7.1	!label!	This entire construct is ignored if it is executed.
. Pointer	1.4B	.	The value of this command is the value of the current pointer.

NAME	MNEMONIC	SECTION	USE	EXPLANATION
=	Equals	2.1.9	=	types out a newline.
			n=	types out n on the console followed by a newline.
			m,n=	types out m followed by a space, followed by n, followed by a newline.

Note: For descriptions of operators in numeric expressions see section 1.4B

(END)

Command
Author-Maintained Library
J. M. Broughton
Room 39-200, Ext. 3-7788
2/25/74

Name: xpl

The xpl command invokes the XPL compiler to translate an ASCII source segment into a Multics object segment. The segment will be placed in the user's current working directory, as will any listing segments produced. In general, this command behaves like standard system translators.

Usage:

xpl pathname -option1- ... -optionn-

1) pathname — is the relative pathname of the segment to be compiled. A suffix of ".xpl" will be assumed for the source segment if it does not appear.

2) optioni — may be selected from the following list of options:

-source, -sc — produces a line-numbered printable ASCII listing of the program. The default is no listing.

-symbols, -sb — gives a listing of the source as above, and all the names declared in the program with their attributes. The default is no symbols.

-map — provides a listing of the source and symbols followed by a map of the object code generated. The default is no map.

-list, -ls — yields all of the above information, plus an assembly-like listing of the compiled code. The default is no list.

-execute, -ex — allows the program to be executed despite severe errors detected during compilation. The default is to suppress execution.

-library1, -lib1 — causes the program to be compiled using a null string compaction routine. This is the default.

-library2, -lib2 causes the program to be compiled using a real string compaction routine. This mode should be used by programs doing a large amount of string manipulation.

-times prints on the stream user_output information regarding the time used by various phases of the compiler.

The XPL Language

The XPL language was developed at Stanford University as part of a translator writing system. It is a simple dialect of PL/1 supporting only automatic variables; all data types, except floating, are supported. Strings are implicitly varying, and arithmetic elements are of fixed scale and precision. It has certain features not found in PL/1, for instance, a limited macro facility, and built-in functions for performing shifts on full words. The best reference for the XPL language is found in A Compiler Generator by McKeeman, Horning, and Wortman (Prentice-Hall, 1970), Chapter 6.

Notes on the Multics Implementation

References to the functions input(i) and output(i) are used to do I/O. These functions cause data to be read or written on the streams xpl_input_i and xpl_output_i. The streams for i = 0 are special-cased and the input comes from the stream user_input and output is directed to the stream user_output. The stream xpl_output_1 is by default attached to error_output. To use any other stream, it must first be attached to a device by an appropriate iocall.

References to the function file(i) may be used to move large blocks (one record, 1024 words) of data to and from files. The files written or read are xpl_file_i. By default, these files will be created in the process directory; to use a permanent file, one should put a link in his process directory to the desired file.

All XPL programs on Multics are subject to certain restrictions: they cannot be bound, and they cannot be called recursively. In the former case, the binder will refuse to bind them, and though the latter is possible, unpredictable results will occur.

Certain built-in functions do not appear in this implementation: addr, clock_trap, interrupt_trap, monitor_link, trace, and untrace. The built-in functions, arg_count, and argument(i) have been added however. They give the number of arguments and the ith argument respectively. The functions corebyte and coreword have slightly different meanings. Corebyte is overlayed, not on core, but on the string data area; coreword is overlayed on the arithmetic data area.

Subroutine
Author-Maintained Library
Tom Van Vleck
Room 39-513, Ext. 1749
2/25/74

Name: ask_

The ask_ module provides the programmer with a flexible terminal-input facility for whole lines, strings delimited by blanks, or fixed-point and floating-point numbers. Special attention is given to prompting the terminal user.

Entry: $ask_

This entry returns the next string of characters delimited by blanks or tabs from the line typed by the user. If the line buffer is empty, "ask_" formats and types out a prompting message and reads a line from "user_input".

Usage:

```
call ask_(ctl,ans,ioa_args...);
```

1)	ctl char(*) (input)	This is a control string in the same format as that used by "ioa_".
2)	ans char(*) (output)	The return value.
3)	ioa_args (input)	Any number of arguments to be converted according to "ctl".

Entry: $ask_clr

This entry clears the internal line buffer. Because the buffer is internal static, one program's input may accidentally be passed to another unless the second begins with a call to this entry. "ask_$ask_clr_" also can be called if a value typed by the user is incorrect and if the program wishes to ask for the line to be retyped.

Usage:

```
call ask_$ask_clr;
```

Entry: $ask_int

This entry works the same as "ask_$ask_" except that the next item on the line must be a number. An integer value is returned. Numbers may be fixed-point or floating-point, positive or negative. A leading dollar sign or a comma will be ignored. If the value typed is not a number, the program will type

"string" non-numeric. Please retype:

and wait for the user to retype the line.

Usage:

```
call ask_$ask_int(ctl,int,ioa_args);
```

1)	ctl char(*) (input)	As above.
2)	int fixed bin (output)	The return value.
3)	ioa_args (input)	As above.

Entry: $ask_flo

This works like "$ask_int" except that it returns a floating value.

Usage:

```
call ask_$ask_flo(ctl,flo,ioa_args);
```

1)	ctl char(*) (input)	As above.
2)	flo float bin (output)	The return value.
3)	ioa_args (input)	As above.

Entry: $ask_line

This entry returns the remainder of the user-typed line. Leading blanks are removed. If there is nothing left on the line, the program will prompt and read a new line.

Usage:

```
call ask_$ask_line(ctl,line,ioa_args);
```

1)	ctl char(*) (input)	As above.
2)	line char(*) (output)	The return value.
3)	ioa_args (input)	As above.

Entry: $ask_c

This entry tests if there is anything left on the line. If so, it returns the next symbol, as in "ask_$ask_", and sets a flag non zero. Otherwise, it sets the flag to zero and returns.

Usage:

```
call ask_$ask_c(ans,flag);
```

1)	ans char(*) (output)	The symbol, if any.
2)	flag fixed bin (output)	=1 if symbol returned. =0 if none there.

Entry: $ask_cint

Conditional entry for integers. If an integer is available on the line, it will be returned and "flag" set to 1. If the line is empty, "flag" will be set to 0. If there is a symbol on the line, but it is not a number, it will be left on the line and "flag" will be set to -1.

Usage:

```
call ask_$ask_cint(int,flag);
```

1)	int fixed bin (output)	Return value.
2)	flag fixed bin (output)	=1 if "int" returned. =0 if line empty. =-1 if no number.

Entry: $ask_cflo

This entry works like "$ask_cint" but returns a floating value if one is available.

Usage:

```
call ask_$ask_cflo(flo,flag);
```

1)	flo float bin (output)	Return.
2)	flag fixed bin (output)	=0 if line empty. =1 if value returned. =-1 if not a number.

Entry: $ask_cline

Conditional ask for rest of line.

Usage:

```
call ask_$ask_cline(line,flag);
```

1)	line char(*) (output)	Return value.
2)	flag fixed bin (output)	=1 if line returned. =0 if line empty.

Entry: $ask_n

This entry "peeks" at the line and returns the next symbol without changing the line pointer. A call to "$ask_" later will return the same value.

Usage:

```
call ask_$ask_n(ans,flag);
```

1)	ans char(*) (output)	Return symbol.
2)	flag fixed bin (output)	=0 if line empty. =1 if symbol returned.

Entry: $ask_nint

Peek entry for integers. The second argument will be returned as -1 if there is a symbol on the line but it is not a number, as 1 if successful, and as 0 if the line is empty.

Usage:

```
call ask_$ask_nint(int,flag);
```

Arguments as above.

Entry: $ask_nflo

Peek entry for floating.

Usage:

```
call ask_$ask_nflo(flo,flag);
```

Entry: $ask_nline

Peek entry for rest of line.

Usage:

```
call ask_$ask_nline(line,flag);
```

Entry: $ask_setline

This entry sets the internal static buffer in "ask_" to the given input line in order that the line may be scanned.

Usage:

```
call ask_$ask_setline(line);

dcl line char(*)
```

Trailing blanks will be removed from line. A carriage return is optional at the end of line.

Entry: $ask_prompt

This entry scraps the current contents of the internal line buffer and prompts for a new line. The line is read in, and the entry returns.

Usage:

```
call ask_$ask_prompt(ctl,ioa_args...)
```

1)	ctl char(*) (input)	A control string similar to that typed by "ioa_".
2)	ioa_args (input)	Any number of arguments to be converted according to "ctl".

attach_fortran_file_

Subroutine Call
Author-Maintained Library
Joseph W. Dehn III
Room 39-200, Ext. 3-7788
3/12/75

Name: attach_fortran_file_

This subroutine allows a program written in FORTRAN to specify the attachment for an I/O "file number" or "unit number".

Usage: call attach_fortran_file_(ifile,astring,ierr)

1) ifile is a FORTRAN file reference number, which must be between 1 and 99.

2) astring is a character constant or variable which specifies the attachment. See Chapter 14 of the FORTRAN manual for the format of attachment specifications.

3) ierr is a standard Multics error code. If the attachment was successful, ierr will be zero.

call attach_fortran_ssfile_(ifile,path,ierr)

2) path is a character constant or variable which specifies the pathname of a storage system file (segment or MSF).

Notes:

1) The second entry is for convenience only. The following two statements have the same effect:

```
call attach_fortran_ssfile_(n,"a>b",ierr)
call attach_fortran_file_(n,"vfile_ a>b",ierr)
```

2) When the program is done with the file, it should be closed using the "endfile" statement. The endfile statement will not, however, detach the file if it was attached using these subroutines or with the io_call command.

3) If the file specified by "ifile" is already attached, it will be closed and detached automatically when attach_fortran_file_ is called.

Examples:

```
        call attach_fortran_file_(1,"syn_ user_output",ierr)

        call attach_fortran_ssfile_(6,"output_seg",ierr)

        character*32 pathname
        read(5,10)pathname
10      format(a32)
        call attach_fortran_ssfile_(kunit,pathname,ierr)
```

Subroutine
Author-Maintained Library
Joseph W. Dehn III
Room 39-200, Ext. 3-7788
10/21/74

Name: basic_plot_

This subroutine is an interface which allows programs written in the BASIC language to use the "plot_" facility of the Multics Graphic System. For details of the operation of plot_, please see the MPM documentation.

Usage:

Initialization

```
call "basic_plot_$init":x$,y$,l,b,g,s
```

x$	label for the X axis
y$	label for the Y axis
l	=1 for linear-linear =2 for log-linear =3 for linear-log =4 for log-log
b	logarithm base for l>1
g	=0 for tic marks and values =1 for dotted grid and values =2 for solid grid and values =3 for no grid or values
s	=0 for normal scaling =1 for equal scaling

Setting Scales

```
call "basic_plot_$scale":x1,x2,y1,y2
```

x1	minimum X
x2	maximum X
y1	minimum Y
y2	maximum Y

Plotting

```
call "basic_plot_":x(),y(),n,v,s$
```

x()	array of X values
y()	array of Y values
n	number of elements in x() and y()
v	=1 for plotting with vectors =2 for plotting with vectors and symbol =3 for plotting with symbol only
s$	the symbol for v>1

Notes:

1) If the entry "basic_plot_$scale" is not called, automatic scaling will be performed.

2) BASIC programs use zero-origined arrays. Therefore, to pass N points to basic_plot_, the values should be stored in array elements zero through N-1.

check_basic_file_

Subroutine
Author-Maintained Library
Joseph W. Dehn III
Room 39-200, Ext. 3-7788
7/27/74

Name: check_basic_file_

This subroutine, intended primarily for use by BASIC programs, may be used to get information on a given file specification. By calling this routine before using the string in a "file" statement, the program can catch certain errors that otherwise would terminate the program.

Usage:

```
call "check_basic_file_": a$,c
```

where a$ is the file specification, and c is an integer as follows:

1	specification was null or blank
2	specification was "*"
3	pathname: bad syntax
4	pathname: does not exist
5	pathname: may exist, but no access
6	pathname: exists but is zero length
7	pathname: exists (segment)
8	pathname: exists (msf)
9	pathname: exists (directory)
10	pathname: is the ROOT
11	ioswitch: exists
12	ioswitch: does not exist
13	ioswitch: exists and attachment specified
14	ioswitch: does not exist, but attachment specified
15	ioswitch: bad syntax
16	undefined error

Note:

The interpretation of the code "c" is left to the calling program. Values of (1,3,5,9,10,12,15,16) almost certainly indicate an error. Other codes may indicate an error or not, depending on how the file is to be used. For example, c=4 means segment does not exist, which would be an error if the program were going to read from the file, but not necessarily an error if the program were to write to it.

Example:

```
1 print "Name of file";
2 input a$
3 call "check_basic_file_": a$,c
4 on c goto 5,5,5,5,5,5,7,7,5,5,7,5,7,7,5,5
5 print "Bad file name"
6 go to 2
7 file #1: a$
  (rest of program)
```

Subroutine Call
Author-Maintained Library
Gary M. Palter
Room 39-200, Ext. 3-7788
2/25/74

Name: check_msf_

This subroutine will determine if a specified directory is a multi-segment file. The definition of a multi-segment file used by this subroutine is given below.

Usage:

```
dcl     check_msf_ entry (char(*), char(*), area(*) aligned, fixed
        bin(35));

call    check_msf_ (directory, entry, user_area, status_code);
```

1) directory — is the directory portion of the name of the branch to be tested. (Input)

2) entry — is the entry portion of the above name. (Input)

3) user_area — is an area to be used for allocations. (Input)

4) status_code — is a returned status code. (Output)

The possible values of status_code are:

0 — The branch is a multi-segment file.

error_table_$bad_ms_file — The branch is a directory, but is not a multi-segment file.

error_table_$nondirseg — The branch is not a directory.

Any error code which may be returned by hcs_$star_list_ and hcs_$status_long.

Notes:

A directory is considered to be a multi-segment file by this subroutine only if it satisfies all of the following conditions:

1) The directory has a non-zero bit count.

2) The directory does not contain any links or directories.

3) The number of entries in the directory is equal to its bit count, and each entry has exactly one name which is the character string representation of a number satisfying the condition $0 \leq number \leq bit\_count-1$.

Subroutine
Author-Maintained Library
Tom Van Vleck
Room 39-513, Ext. 1749
9/8/70

The procedure cv_ performs several useful number conversion calculations.

Entry: $hrmin

The entry point hrmin accepts a time in seconds and converts it to a time in hours and minutes, suitable for printing. The converted time is rounded up to the next minute, so that any nonzero input will produce a nonzero time.

Usage:

```
call cv_$hrmin(ss,hr,min);
```

1)	ss fixed (input)	Input time in seconds.
2)	hr fixed (output)	Hours.
3)	min char(*) (output)	Minutes, 4 chars.

Entry: $absdat

The entry point absdat returns the number of days since January 1, 1901, given the month, day and year as input.

Usage:

```
call cv_$absdat(mo,da,yr,abs);
```

1)	mo fixed (input)	Month
2)	da fixed (input)	Day
3)	yr fixed (input)	Year
4)	abs fixed (output)	Absolute date

Entry: $shift

The routine shift computes the current accounting shift numbers.

Usage:

```
call cv_$shift(mo,da,yr,hr,sh);
```

1)	mo fixed (input)	Month
2)	da fixed (input)	Day
3)	yr fixed (input)	Year
4)	hr fixed (input)	Hour
5)	sh fixed (output)	Shift number

Entry: $mwvf

The routine mwvf converts a floating dollar amount to ASCII characters.

Usage:

```
v = cv_$mwvf(flo);
```

1) flo float (input)

2) v char(15) (output)

The converted value will have a floating dollar sign and commas every three digits. Blanks will be returned if the number is all zero. The value is right-justified. Fifteen characters will be returned.

Example:

Input	Returned
1234.567e0	$1,234.57
0.001e0	$.00
0.0e0	
-5.7e0	$-5.70
1.234e10	$**,***,***.**

Entry: $cdate

The entry cdate converts a character-string date into a double-precision integer in system calendar clock format, i.e., in micro-seconds since 0000GMT, January 1, 1901. The date may be expressed flexibly. An invalid date will convert to 0. A date of "*" returns the current time.

Usage:

```
call cv_$cdate(date,time);

1)  date char(*) (input)            Date

2)  time fixed bin(71) (output)     Time
```

Example:

The date June 1, 1970, may be expressed as:

```
060170
6/1/70
0601
6/1
06/1
6/01
061/70
6/0170
```

detach_fortran_file_

Subroutine Call
Author-Maintained Library
Joseph W. Dehn III
Room 39-200, Ext. 3-7788
3/12/75

Name: detach_fortran_file_

This subroutine allows a program written in FORTRAN to detach a file that was attached using the "io_call" command or using the "attach_fortran_file_" subroutine.

Usage: call detach_fortran_file_(ifile,ierr)

1) ifile is a FORTRAN "file number" or "unit number" in the range 1 to 99.

2) ierr is a standard Multics error code. If the detach was successful, ierr will be set to zero.

Example:

```
call detach_fortran_file_(1,ierrcode)
```

Subroutine
Author-Maintained Library
Edward McCabe
575 Technology Square
Fifth Floor, Ext. 3-1533
7/9/74

Name: fillin_dprint_str_

Usage: declare fillin_dprint_str_ entry (char(*) varying, pointer, fixed binary (35));

call fillin_dprint_str_(optstring, dpap, ercd);

Where:

optstring — is the string used to fill in the dprint_arg structure based on dpap.

dpap — is the pointer to the dprint_arg structure. Only structures of version = 1 are accepted by this subroutine.

ercd — is an error code returned by this subroutine which can assume the following values:

0 no errors were encountered.
1 the structure was not version = 1.
2 an error was encountered in the parsing of the options.

This routine assumes that the caller will choose and control the defaults of the dprint_arg structure as required by the dprint_ subroutine. As a result, only those values which are explicitly specified in optstring will be affected by this subroutine.

Structures which are not understood, i.e., version =1 will be rejected. Errors encountered in the parsing of optstring will cause a message to be printed (via com_err_), but any of the rest of the structure which is appropriate will be filled in.

Finally, if 'dpunch', 'dprint', or 'dp' appear at the beginning of the string, they will be ignored (since it may be convienent to pass a string containing them, rather than parsing them out).

This subroutine accepts the following options:

-print — specifies that both dprint_arg.pt_pch and dprint_arg.output_module are to be set to 1 (for printed output).

-punch — specifies that both dprint_arg.pt_pch and dprint_arg.output_module are to be set to 2 (for punched output). Note that the default is 7punch.

The use of any of the three following options results in setting dprint_arg.pt_pch to 2 (for punched output).

-mcc specifies that dprint_arg.output_module is to be set to 3 for MCC punch.

-raw specifies that dprint_arg.output_module is to be set to 4 for raw punch.

-7punch, -7p specifies that dprint_arg.output_module is to be set to 2 for 7punch.

-cp n, -copy n specifies that dprint_arg.copies is to be set to n.

-dl, -delete specifies that dprint_arg.delete is to be set to 1.

-header header, -he header
specifies that dprint_arg.header is to be set to " for " header. Note that this results in the correct header for the output.

-ds dest, -destination dest
specifies that dprint_arg.dest is to be set to dest.

-q n, -queue n specifies that dprint_arg.queue is to be set to n.

-dvc class, -device_class class
specifies that dprint_arg.class is to be set to class.

Any option can be overridden by a succeeding option except the -delete, -dl option.

Subroutine
Author-Maintained Library
Student Information Processing Board
Room 39-200, Ext. 3-7788
9/6/74

Name: fixed_to_english_

This subroutine returns the spelled-out English representation of a fixed binary number.

Entry: fixed_to_english_

This entry converts a fixed binary value and uses the prefix "minus" to indicate a negative number.

```
declare fixed_to_english_ entry(fixed bin(35)) returns(char(*));

output_string=fixed_to_english_(number);
```

Entry: fixed_to_english_$own_minus

This entry converts a fixed binary value and uses a user-specified prefix to indicate a negative number.

```
declare fixed_to_english_$own_minus entry(fixed bin(35), char(*))
     returns(char(*));

output_string=fixed_to_english_$own_minus(number,minus_word);
```

Subroutine Call
Author-Maintained Library
Student Information Processing Board
Room 39-200, Ext. 3-7788
4/13/73

Name: get_caller_ptr_

This subroutine is an ALM utility module which returns pointers to the text sections of various ancestors of the calling program.

Usage:

```
declare get_caller_ptr_ external entry returns (pointer);

   caller_ptr = get_caller_ptr_ ();
```

1) caller_ptr is a pointer to the text section of the direct ancestor of the procedure calling get_caller_ptr_. (Output)

Entry: get_caller_ptr_$my_ptr

This entry returns to the calling program a pointer to its own text section.

Usage:

```
declare get_caller_ptr_$my_ptr external entry returns
          (pointer);

  my_ptr = get_caller_ptr_$my_ptr ();
```

1) my_ptr is a pointer to the text section of the procedure calling get_caller_pointer_$my_ptr. (Output)

Entry: get_caller_ptr_$backstack

This entry returns to the calling program a pointer to the text section of its nth ancestor, where the 0th ancestor is defined as the program calling get_caller_ptr_$backstack.

Usage:

```
declare get_caller_ptr_$backstack external entry (fixed
          binary) returns (pointer);

  any_ptr = get_caller_ptr_$backstack (n);
```

1) n is the ancestor for which a pointer to the text section is desired. If n is zero, the pointer returned is the same as would be returned by get_caller_ptr_$my_ptr. (Input)

2) any_ptr is the pointer to the text section of the nth ancestor of the program calling get_caller_ptr_$backstack. (Output)

Subroutine
Author-Maintained Library
Student Information Processing Board
Room 39-200, Ext. 3-7788
9/6/74

Name: get_line_length_

This subroutine may be used to find the line_length of a specified io-switch. It should normally only be used on an io-switch that is associated with a terminal.

Usage:

```
declare get_line_length_ entry(char(*)) returns(fixed bin);

line_len=get_line_length_(switch_name);
```

1) switch_name is the name of the io-switch.

2) line_len is the number of characters per line; if this value cannot be determined, it will be zero.

Subroutine
Author-Maintained Library
Richard H. Gumpertz
Senior House, Ext. 2893
11/5/73

Names: get_seg_ptr_
get_seg_ptr_arg_
get_seg_ptr_full_path_
get_seg_ptr_search_
release_seg_ptr_

This procedure consists of entries to initiate and terminate data segments. It also is capable of creating, truncating and setting the bit count on segments. It is more useful than the current Multics file primitives because it expands pathnames, creates segments and initiates all in one call by the user. Similarly, it sets the bit count, truncates the segment and terminates all in one call.

Entry: get_seg_ptr_

This entry initiates a segment given a relative pathname and checks access to the segment. If the segment does not exist, it will be created if the user so requests.

Usage: declare get_seg_ptr_ external entry

```
(char(*), bit(6) aligned, fixed bin(24), ptr,
fixed bin(35));

call get_seg_ptr_ (pathname, wanted_access, bit_count,
                   return_ptr, return_code);
```

Where:

pathname — is a relative pathname to the segment. (Input)

wanted_access — is the requested access to the segment. The first five bits are considered to indicate the standard Multics access control bits. They are "t" (trap, not currently implemented), "r" (read), "e" (execute), "w" (write) and "a" (append), respectively. If the segment exists, then an error code (error_table_$moderr) will be returned if the user does not have at least the access requested. Note, however, that return_ptr will contain a valid pointer even if this error occurs. The sixth bit is interpreted as a "c" (create) bit -- if the segment does not exist, it will be created if this bit is on. If a segment is created, it is given the access bits specified by the t, r, e, w, and a bits. (Input)

bit_count is the bit count of the segment. (Output)

return_ptr is a pointer to the segment. If the segment is not initiated, this pointer will be returned containing the null pointer. (Output)

return_code is the standard Multics error code. The only condition under which this code will be non-zero when the return_ptr is non-null is if the error is error_table_$moderr. (Output)

Entry: release_seg_ptr_

This entry terminates a segment initiated by one of the entries to get_seg_ptr_. If a bit count is specified, the bit count of the segment is set and the segment is truncated to the corresponding length.

Usage: declare release_seg_ptr_ external entry (ptr, fixed bin(24), fixed bin(35));

```
call release_seg_ptr_ (seg_ptr, bit_count, return_code);
```

Where:

seg_ptr is a pointer to the segment to be terminated. (Input)

bit_count is the bit count to be set on the segment. If this argument is negative, it is assumed that the bit count should remain the same and no truncation should take place. (Input)

return_code is a standard Multics error code. (Output)

Entry: get_seg_ptr_arg_

This entry is identical to get_seg_ptr_ except that it obtains the pathname of the segment to be initiated from the caller's argument list. It saves a call to cu_$arg_ptr_.

Usage: dcl get_seg_ptr_arg_ external entry(fixed bin, bit(6) aligned, fixed bin(24), ptr, fixed bin(35));

```
call get_seg_ptr_arg_(arg_number, wanted_access, bit_count,
                      return_ptr, return_code_;
```

Where:

arg_number is the number of the caller's argument to be used. (Input)

All other arguments are identical to get_seg_ptr_.

Entry: get_seg_ptr_full_path_

This entry is identical to get_seg_ptr_ except that the pathname is specified as an absolute pathname in directory/entry form.

Usage: declare get_seg_ptr_full_path_ external entry

```
(char(*), char(*), bit(6) aligned, fixed bin(24), ptr,
fixed bin(35));

call get_seg_ptr_full_path_(dir_name, entry_name,
                            wanted_access, bit_count,
                            return_ptr, return_code_;
```

Where:

dir_name — is the absolute pathname of the directory of the segment. (Input)

entry_name — is the entry name of the segment. (Input)

All other arguments are identical to get_seg_ptr_.

Entry: get_seg_ptr_search_

This entry is identical to get_seg_ptr_ except that just an entry name is specified. The directory is determined by Multics search rules. If the segment is not found and if the "c" (create) bit is on, then the segment is created in the process directory. Note, however, that if the entry name is not known as a reference name before a call to get_seg_ptr_search_, this entry will not cause it to be made known. This procedure initiates the segment with a null reference name. This has the net effect that full search rules will be followed each time this routine is called.

Usage: declare get_seg_ptr_search_ external entry

```
(char(*), bit(6) aligned, fixed bin(24), ptr, fixed bin(35));

call get_seg_ptr_search_(entry_name, wanted_access, bit_count,
                         return_ptr, return_code);
```

Where:

entry_name — is the entry name of the segment to be found. (Input)

All other arguments are identical to get_seg_ptr_.

Examples:

1. To read a segment

```
declare file aligned char(262144) based (file_ptr),
r_access bit(6) aligned int static init ("010000" b);

call get_seg_ptr_ (file_name, r_access, count, file_ptr,
                   error_code);

if error_code ¬= 0 then go to error;

count = divide (count, 9, 17, 0);

string_variable = substr (file, 1, count);

call release_seg_ptr_ (file_ptr, -1, error_code);

if error_code ¬= 0 then go to error;
```

2. To write a segment

```
declare file aligned char(262144) based (file_ptr),
rwac_acess bit(6) aligned int static init ("010111" b);

call get_seg_ptr_ (file_name, rwac_access, count, file_ptr,
                   error_code);

if file_ptr = null then go to error;

count = length (string_variable);

substr (file, 1, count) = string_variable;

call release_seg_ptr_ (file_ptr, count*9, error_code);

if error_code ¬= 0 then go to error;
```

3. To search for a segment

Program "x," whenever entered, does a search for a segment called "init.x" which, if found, is used to initialize x. This could be done as follows:

```
call get_seg_ptr_search_("init.x", "010000" b, count, file_ptr,
                         error_code);
```

```
    if file_ptr¬= null then /*initiate only if found*/
init:  do;
       count = divide (count, 9, 17, 0);
            .
            .
            .
       /*do initialization*/
            .
            .
            .
       call release_seg_ptr_ (file_ptr, -1, error_code)
       if error_code¬= 0 then go to error;
       end init;
/* rest of program*/
```

Subroutine Call
Author-Maintained Library
Student Information Processing Board
Room 39-200, Ext. 3-7788
4/11/73

Name: get_mydir_

This subroutine allows a procedure to ascertain the directory in which it resides.

Usage:

```
declare get_mydir_ external entry (char (*));

   call get_mydir_ (dir_name);
```

1) dir_name is the name of the directory in which the procedure calling get_mydir_ resides. (Output)

Entry: get_refdir_

This entry allows a procedure to ascertain the directory in which its caller resides.

Usage:

```
declare get_refdir_ external entry (char (*));

   call get_refdir_ (dir_name);
```

1) dir_name is the name of the directory in which the caller of the procedure calling get_refdir_ resides. (Output)

Subroutine Library
Author-Maintained Library
Overlap Project Staff
NE40-500, 253-2053
11/19/74

Name: IMSL Library

The IMSL Library is a collection of approximately 300 mathematical and statistical subroutines, written in FORTRAN. These routines have been compiled on the MIT Multics machine as part of work on the Consistent System. They are being made available to the community in subroutine form as well.

The bulk of this description is intended to give the reader an idea of the range of capabilities of the library. Documents describing it are on file in the IPC Reading Room (39-430), and can be purchased from

IMSL
6200 Hillcroft, Suite 510
Houston, Texas 77036

Anyone interested should contact the Overlap Project Staff at the address above (NE40 is 575 Technology Square) for more information.

These subroutines are in ">libraries>imsl" which must be in the user's search path. To do this, enter the command "ssa >libraries>imsl" or "asr >libraries>imsl".

RESTRICTIONS and DISCLAIMER

The library is proprietary. Its use is licensed to the Overlap Project and to the MIT Multics. Attempting to carry the library to another machine is prohibited by the license agreement. The lease is on a year by year basis, and the Overlap Project can make no guarantee to the community that it will renew the lease for any given period. Finally, while parts of the library have been tested in the Multics environment and found to be of very high quality we can make no guarantees about the correctness of the routines. IMSL does certify them and is of considerable help if a user of the library gets into substantive mathematical trouble with it; requests for such assistance should be directed to the Overlap Project as holder of the lease. A summary of

modifications we have made to the library, and for which IMSL bears no responsibility, appears below.

Capabilities

The library is arranged within subgroups, called "chapters". These chapters are titled as follows: (1)

Analysis of Experimental Design Data
(includes analysis of variance and classification routines)
Basic Statistics
(includes elementary bayesian inference, data screening, and elementary classical inference)
Categorized Data Analysis
Differential Equations; Quadrature; Differentiation
Eigenanalysis
Forcasting; Econometrics; Time Series
Generation and Testing of Random Numbers; Goodness of Fit
Interpolation; Approximation; Smoothing
Linear Algebraic Equations
Mathematical and Statistical Special Functions
(includes probability distribution functions and special functions of mathematical physics)
Non-Parametric Statistics
(includes analysis of variance, binomial or multi-nomial bases, kolmogorov-smirnov tests, randomization bases)
Observation Structure
Regression Analysis
Sampling
Utility Functions
(see Note 1)
Vector, Matrix Arithmetic
Zeros and Extrema; Linear Programming

Modifications

The following list is a summary, some knowledge of the library may be required to understand it.

1) Chapter U, which contains Utility Functions, has not been implemented. It contains a series of functions for

---

(1) This material is taken, in large part from IMSL publication LIB-0004

input/output and the like. The maintainers are willing to supply the source to anyone seriously interested in bringing up this Chapter on Multics.

2) Routines that are indicated in the manual as being available in double or single precision have been provided in single precision only.

3) All calls to the IMSL error message printing routine, UERTST, have been removed. The routines still return error codes.

4) The names of all subroutines, functions, and entries into them have been changed to insure that conflicts will not occur with the dynamic linking mechanism on Multics. The renaming rule is as follows:

   a) Find the subprogram in the IMSL manual. At the head of the description is a listing of some comment cards, at the end of which is an eight character "card label". The first four characters of the "card label" is a four-character "deck label".

   b) For the main entry point, take the deck label, change its characters to lower case, and prefix them with "cs_i" to obtain the subroutine name.

   c) For any additional entry points, add the digit "1" to the subroutine name for the first one, "2" for the second, etc.

5) The library edition implemented on Multics was originally designed for the Univac 1108. Due to differences in Fortran compilers and machine structure, changes have been required to constants and the order of statements, and a special overflow procedure written in PL/I has been provided. Unless problems arise with them, these modifications should be transparent to the user. A complete list of these modifications is available from the Overlap Project, and with the Reading Room copy of the IMSL manuals.

Subroutine
Author-Maintained Library
Edward J. McCabe
575 Technology Square
Fifth floor, Ext. 3-1533
2/21/74

Name: linear_q_hash, lqh
linear_q_hash_, lqh_

The subroutine linear_q_hash is an implementation of the linear quotient hashing algorithm described by Bell and Kaman in the November, 1970, issue of the Communications of the Association for Computing Machinery (pp. 675-677).

Usage:

```
call linear_q_hash(result,residue,table_size,word);
```

Arguments:

```
declare (result,residue,table_size) fixed binary(35,0);
declare word char(32) varying;
```

Where:

result	is in the range 0<=result<table_size. (output)
residue	is in the range 1<=residue<(2**35). (output)
table_size	is a prime integer. (input)
word	is the source word to be hashed. (input)

Entry point for retries:

```
call lqh$retry(result,residue,table_size);
```

Although arguments are declared identically, their significance is different on retries.

result	is both input and output. On input, it is the most recently returned result from lqh or lqh$retry. On output, it is a new value of result.
residue	is the value of residue last obtained from lqh or lqh$retry. (input)
table_size	is the original prime that defines the range. (input)

This subroutine, given a character string (word) and a prime number (table_size), will produce a pseudo-random result in the range 0<=result<table_size, plus a residue of unpredictable size (less than 2**35, however). This result may be used to reference a location in a hash table. If this location is unsuitable, up to table_size - 1, retries may be made; complete coverage of the range is guaranteed. Retries must be attempted by returning the most recently obtained values for "result" and "residue," in order to ensure complete coverage of the range. The user must keep track of the number of retries attempted.

Note that the key is constructed using only the information-carrying bits of the first five characters of the source word (padded on the right with blanks, if necessary). On Multics, these are the right-most seven bits of each character.

Note that linear_q_hash, lqh, linear_q_hash_ and lqh_ are synonymous for both entry points.

Subroutine
Author-Maintained Library
J. M. Broughton
Room 39-200, Ext. 3-7788
9/30/73

Name: qd

This procedure is used to determine if an instruction with a given opcode modifies the contents of the q-register.

Usage:

```
declare  qd entry (fixed bin(35)) returns (fixed bin(35));

modified = qd (opcode);
```

1) modified is one if the instruction alters the q-register, zero otherwise.

2) opcode is the opcode of the instruction in question.

Miscellaneous Call
Author-Maintained Library
John C. Klensin
575 Technology Square
Fifth floor, Ext. 6217
1/18/71

Name: reverse_index_

This function searches a character string from right to left for a particular character string. The location returned is in characters from the left.

Entry: reverse_index_$reverse_index_

This entry returns the index position (from the left) of the first character string (from the right) equal to the specified string.

```
dcl reverse_index_ ext entry(char(*),char(*)) returns(fixed bin);

i = reverse_index_(string1,string2);
```

Where:

i	is the index position from the left of the string. It will be 0 if the string specified in string2 is not in string1. (output)
string1	is the string to be scanned for string2. (input)
string2	is the comparison string. (input)

Entry: reverse_index_$notequal

This entry returns the index position (from the left) of the first character not equal to the specified character. (This is useful for finding the last nonblank character in a word.)

```
dcl reverse_index_$notequal ext entry(char(*),char(1)) returns(fixed bin);

i = reverse_index_$notequal(string1,char);
```

Where:

i	is the index position from the left of the string. It will be 0 if the string is not found in string1. (output)
string1	is the string to be scanned for char. (input)
char	is the character to be searched for in string1. (input)

Note: reverse_index_ is similar to the PL/I "index" built-in function, except that it searches the string from the opposite direction; the position in the string is expressed in the same fashion.

Subroutine
Author-Maintained Library
J. Klensin
575 Technology Square
Fifth Floor, Ext. 6217
5/24/71

Name: scan_

The procedure scan_ contains a number of functions that scan across a supplied character string looking for an occurrence of any single character from a second string. Functions are supplied that scan the string from left to right and from right to left, and that look for the first character equal to and the first character not equal to any of those in the second string.

Entry: scan_$scan_

This entry is used to find the first character in one string that matches any character in a second string. It returns, as do all of the functions below, the index of the located character from the left of the first string.

Usage:

```
dcl scan_$scan_ ext entry(char(*),char(*)) returns(fixed bin);
i=scan_$scan_ (string1,string2);
```

string1 is the string to be scanned for the first occurrence of a character in the second string.

string2 is the string containing characters to be located in the first string.

i is the index (from the left) of the first character in string1 to match any character in string2. If no character in string1 matches any character in string2, "i" is set to zero.

Entry: scan_$scan_notequal

This entry is used to find the first character in one string that does not match any character in the second string. (Note: This entry point is equivalent to the VERIFY function existing in certain implementations of PL/I.)

Usage:

```
dcl scan_$scan_notequal ext entry(char(*),char(*)) returns(fixed bin);
i=scan_$scan_notequal (string1,string2);
```

string1 is the string to be scanned for the first occurrence of a character not in the second string.

string2 is the string containing characters to be located in the first string.

i is the left index of the first character in string1 that does not match any character in string2.

Entry: scan_$scan_rev

This entry is used in a fashion similar to scan_$scan_, except that it searches the string from the right. (Note: This is not the same as AML subroutine reverse_index_.)

Usage:

```
dcl scan_$scan_rev ext entry(char(*),char(*)) returns(fixed bin);
i=scan_$scan_rev (string1,string2);
```

string1 is the string to be scanned from the right for the first occurrence of a character in the second string.

string2 is the string containing characters to be located in the first string.

i is the index (from the left) of the first character in string1 to match any character in string2. If no character in string1 matches any character in string2, "i" is set to zero. Note that "i" is still a left index.

Entry: scan_$scan_notequal_rev

This entry is used in a fashion similar to scan_$scan_notequal, except that it searches the string from the right.

Usage:

```
dcl scan_$scan_notequal_rev ext entry(char(*),char(*))
returns(fixed bin);
i=scan_$scan_notequal_rev (string1,string2);
```

string1 is the string to be scanned from the right for the first occurrence of a character not in the second string.

string2 is the string containing characters to be located in the first string.

i is the left index of the first character in string1 that does not match any character in string2.

Entry: scan_$scan_ptr
scan_$scan_ptr_notequal
scan_$scan_ptr_rev
scan_$scan_ptr_notequal_rev

These entries correspond to the ones above, except that they accept a pointer and a length to designate the first string, rather than having the string passed directly.

Usage:

The first of these is typical of the others.

```
dcl scan_$scan_ptr(pointer,fixed bin,char(*)) returns(fixed bin);
i=scan_$scan_ptr(ptr_to_string1,length_of_string1,string2);
```

i and string2 are the same as defined above.

ptr_to_string1 is the pointer to the first string.

length_of_string1 is the length of the first string.

Disclaimer:

No claim is made that these functions are fast or that they could not be done more efficiently with in-line code. The functions have proved convenient to get some types of code working that could be optimized later or that did not require optimization.

For the reason mentioned in the note above, the function scan_$scan_notequal may be withdrawn when the PL/I verify BIF becomes available.

I/O System Device Interface Module
Author-Maintained Library
Student Information Processing Board
Room 39-200, Ext. 3-7788
2/25/74

Name: tek_

The tek_ DIM allows a user to perform graphic input and output on a Tektronix 4002, 4012, or 4013 type terminal.

Usage:

```
call ios_$attach (stream_name, "tek_", to_stream, mode,
     status);
```

1) stream_name is usually "graphic_output" or "graphic_input".

2) to_stream is usually "user_i/o", but may be any other stream. (Input)

3) mode is ignored.

4) status is a standard I/O system status code. (Output)

Permitted I/O System Calls:

The following I/O system calls are implemented by this DIM:

attach
detach
read
write
order

Returned Status:

With the exception of the error returned upon attempted multiple attachment, this DIM will only reflect status codes from downstream.

Order Request:

Only one order request is implemented by this DIM:

screen_size causes the output and input to be scaled so that the maximum square physical screen size of the terminal (760 x 760) represents a virtual screen size of (N x N), where N is a fixed binary number pointed to by the pointer argument to ios_$order. The default is standard for the Multics Virtual Terminal screen (1024 x 1024).

Graphic Input Format:

Any read call issued through tek_ will activate the crosshair cursors for graphic input. The graphic input portion of the DIM simulates the operation of an ARDS mouse or joystick. To send any graphic input command, the user must type one character followed by optional text, followed by a newline character. The coordinates of the crosshair intersection are sent with the first character, and the character itself specifies the action to be taken. Because of the terminal architecture, the desired constructs will be echoed on the screen after every request. Allowable characters are:

s (setposition) Causes a setposition to the current location to be sent.

v (vector) Causes a vector to be drawn from the last coordinate point to the current location.

i (shift, stands for "invisible vector") Causes a relative shift to be generated to the current position.

p (point) Causes a shift to the current position, and displays a visible point.

t (text) (followed by text before the newline) Causes a text string to be generated at the current position. If the current position differs from the position at the last command, a shift is generated to the new position.

e (escape) (followed by text before the newline) Signifies that the text string is to be treated as a normal, non-graphic ASCII string. If this does not occur as the first entity for any particular read call, it is ignored. This entity causes termination of the read call.

q (quit) Sends a setposition at the current position, and terminates the read call.

The input stream is in Multics Graphic Code format, and may be read and parsed by using the subroutine gf_input_ rather than having a program issue a read call directly.

I/O System Device Interface Module
Author-Maintained Library
Student Information Processing Board
Room 39-200, Ext. 3-7788
2/25/74

Name: tek_dim_

This is the PL/I procedure which implements the functions of the tek_ outer module. It is documented under the writeup for tek_.

Subroutine
Author-Maintained Library
J. M. Broughton
Room 39-200, Ext. 3-7788
9/30/73

Name: xcom

This is the compiler for the XPL language.

Usage:

```
declare xcom entry options (variable);

call xcom (entry, toggles, time);
```

1) entry — is the character string forming the entry name for the segment. (Input)

2) toggles — contains compiler toggles to be set initially on. (Input)

3) time — is a character string containing a date/time string for the listing of the program. (Input)

All the above variables should be declared character (*).

Subroutine
Author-Maintained Library
J. M. Broughton
Room 39-200, Ext. 3-7788
9/30/73

Name: xpl_file

This routine is used to perform file I/O for XPL programs. It moves data in 1024 word blocks. Files are kept in the process directory, and have names of the form xpl_file_n.

Usage:

```
declare  xpl_file (fixed bin(32), fixed bin(32), bit (1)
         aligned, fixed bin(32));

call  xpl_file (file, block, output, dummy);
```

1) file — is the file number of the segment to be read or written. If it does not exist, it will be created. Only zero to nine is allowed.

2) block — is the block number of the block to be read or written. It starts at zero.

3) output — if "1"b, the file will be written, if "0"b, it will be read.

4) dummy — is the first word of the block to be read from or written into.

```
declare  xpl_file$truncate entry;

call  xpl_file$truncate;
```

This truncates all files used by an xpl program and which are currently active.

Subroutine
Author-Maintained Library
J. M. Broughton
Room 39-200, Ext. 3-7788
9/30/73

Name: xpl_loader_

This routine is used to move the data portion of an xpl program into a scratch segment in the user's process directory. Scalar and arrary data are placed in the segment "xpl_arith_data_"; string data are placed in "xpl_string_data_". It is called with one argument: a pointer to a structure of pointers to the different areas that are to be filled in.

Usage:

```
declare xpl_loader_ entry (pointer);

call xpl_loader_ (regptr);
```

1) regptr is a pointer to the structure described above:

```
declare

     1 registers aligned based (regptr),
       2 data    pointer,
       2 string  pointer,
       2 array   pointer,
       2 text    pointer,
```

The pointer text is an input parameter, and points to the base of the object segment.

Note:

This routine is, and should only be, called by the prologue sequence of an xpl program.

Subroutine
Author-Maintained Library
J. M. Broughton
Room 39-200, Ext. 3-7788
2/25/74

Name: xpl_operators_

This routine is called by an xpl program in order to perform tasks that it cannot do for itself.

Usage:

```
declare xpl_operators_ entry (pointer);
call xpl_operators_ (stack_ptr);
```

1) stack_ptr is a pointer to the stack frame of the calling procedure.

Notes:

This procedure is, and should only be, called by code as generated by the xpl compiler.

Active Function
Author-Maintained Library
John C. Klensin
575 Technology Square
Fifth Floor, Ext. 6217
12/7/71

Name: all

The active function all returns the contents of a designated segment with all of the new line characters changed to blanks. It is useful when a list of things must be done by several commands in succession or when a file has been created which contains such a list of items.

Usage:

command [all path]

Where "path" is the name of a segment containing characters to be placed on the command line.

Note:

This active function, by its nature, makes it fairly easy to exceed the default command line length (see set_com_line) and the maximum number of arguments accepted by the standard command processor.

Active Function
Author-Maintained Library
John C. Klensin
575 Technology Square
Fifth Floor, Ext. 3-6217
2/20/74

Name: bit_count

This active function is used to obtain and return the bit count on a given segment. It is likely to be useful when the presence of information in a segment is more interesting than whether or not it exists.

Usage: bit_count segname

Where

segname is the relative pathname of the file whose bit count is to be returned.

Note: If the file is not present, a value of "-1" is returned. Otherwise the bit count is returned. This permits testing for "present and bit count greater than zero" by a single test that does not produce errors.

Example: Used in an exec com context, this function might appear in a statement as follows:

```
&if  [greater [bit_count] mailbox 0]
&then &print mail!
```

Active Function
Author-Maintained Library
Student Information Processing Board
Room 39-200, Ext. 3-7788
9/6/74

Name: center

This active function returns its first argument centered in a field of blanks whose length is specified by the second argument. The return value is enclosed in quotes.

Usage:

```
[center string length]
```

1) string is the string to be centered.

2) length is the size of the field of blanks.

Example:

```
dprint -he [center &1 13] &1.list
```

Active Function
Author-Maintained Library
John C. Klensin
575 Technology Square
Fifth Floor, Ext. 3-6217
10/31/73

Name: dwd

This active function returns the name of the default working directory (the home directory unless it has been changed by the command change_default_wdir (cdwd)) in a fashion similar to the active functions wd and pd for the working and process directories.

Usage: dwd

Active Function
Author-Maintained Library
John C. Klensin
575 Technology Square
Fifth Floor, Ext. 3-6217
2/20/74

Name: exist_any

This active function accepts one or more relative pathnames, possibly containing stars or question marks, as arguments. If any of the names is found (or any name is found) that matches one of the star names, the function returns "true." Otherwise, it returns "false."

Usage: exist_any -namel- ...

Where

-namei- are relative pathnames.

Note: The routine returns "true" as soon as a single name match is found, so it does not scan the rest of the input names.

Example: Used in an exec_com context, this function might appear in a statement as follows:

```
&if  [exist_any  **.list]
&then &print you have list segments
```

Active Function
Author-Maintained Library
John C. Klensin
575 Technology Square
Fifth floor, Ext. 6217
1/18/71

Name: ld

The active function ld returns the pathname of the original login directory of the process in which it is invoked.

Usage:

command [ld]

Active Function
Author-Maintained Library
Joseph W. Dehn III
Room 39-200, Ext. 3-7788
11/15/74

Name: translate

This active function may be used to translate a character string in a manner similar to the translate built-in function of PL/I.

Usage:

[translate string new old]

or

[translate string opt]

In the first form, the returned string will be the string formed by substituting for each character occurring in "old" the corresponding character in "new". This is identical to the PL/I function.

In the second form, the translation is specified by a control option, which may be:

-uc	to translate to upper case
-lc	to translate to lower case

Examples:

```
print_string [translate "Test String" -uc]
TEST STRING

print_string [translate 10/13/74 - /]
10-13-74
```