ESD-TR-66-309

MTR-222

USERS' MANUAL FOR THE EDITOR

NOVEMBER 1966

B. Isquith

Prepared for
DEPUTY FOR ENGINEERING AND TECHNOLOGY
DIRECTORATE OF COMPUTERS
ELECTRONIC SYSTEMS DIVISION
AIR FORCE SYSTEMS COMMAND
UNITED STATES AIR FORCE
L. G. Hanscom Field, Bedford, Massachusetts

Project 508F
Prepared by
THE MITRE CORPORATION
Bedford, Massachusetts
Contract AF19(628)-5165

AD0646717

ESD-TR-66-309

MTR-222

USERS' MANUAL FOR THE EDITOR

NOVEMBER 1966

B. Isquith

Prepared for
DEPUTY FOR ENGINEERING AND TECHNOLOGY
DIRECTORATE OF COMPUTERS
ELECTRONIC SYSTEMS DIVISION
AIR FORCE SYSTEMS COMMAND
UNITED STATES AIR FORCE
L. G. Hanscom Field, Bedford, Massachusetts

Project 508F
Prepared by
THE MITRE CORPORATION
Bedford, Massachusetts
Contract AF19(628)-5165

FOREWORD

This program was originally designed and implemented by Emerson Griswold, formerly of The Mitre Corporation, for the PHOENIX simulator on the IBM 7030.

REVIEW AND APPROVAL

This technical report has been reviewed and is approved.

[illegible]

CHARLES A. LAUSTRUP
Colonel, USAF
Director of Computers

ABSTRACT

EDITOR I is an on-line program within the initial PHOENIX computer software system which enables the system user to create, destroy, or modify his collection of symbolic data, organized as files.

The actions of EDITOR I are user-controlled by means of a one-pass assembler, herein described.

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

GLOSSARY OF TERMS

actor	a symbol which has a computed numeric value which is some line number of the open file.
argument	an atom or group of atoms used as control parameters by EDITOR commands.
argument separator	a type of symbol which must appear between arguments--the EDITOR initially defines the character "," (comma) as such.
atom	the lowest logical and informational group processed by the control language.
binary connector	a type of symbol used to concatenate expression elements--the EDITOR initially defines the characters "+" and "-" as such.
block	a group of one or more lines which have contiguous line numbers.
cln	an actor whose value is the current line number in the open file.
closed file	one of two file states, in which a file can only be read.
command	a user request for a specific action to be performed by the EDITOR.
command separator	a type of symbol which must end the argument string of a command, including the null argument string--the EDITOR initially defines the character ";" (semi-colon) as such.
config	an actor whose value is the line number in which a specified literal argument is located.
constant	a type of symbol which has a numeric value.
directory	a list of file names of all existing files.

expression	a type of argument consisting of one or more expression elements connected by binary connectors.
expression element	a numeric atom or a constant or an actor.
file	a block of lines accessed through a file name.
file name	a type of symbol which must be an atom of type variable, and is an unique reference to a file.
file state	the two states in which a file can exist, either open or closed.
file structure	a general term meaning the entire character string being handled by the EDITOR.
flag	an actor whose value is the line number of an uniquely specified line, wherever it appears in the open file.
independent atom	a type of atom which is a self-delimiting atom; an atom which stands by itself and acts as a delimiter for other types of atoms--all characters of type punctuation are such.
line	a string of characters in a file, the last character, and only the last, of which must be a carriage return.
line number	the numeric quantity associated with each line; the first line in a file is line 0 (zero).
literal argument	a type of argument which is a literal atom.
literal atom	a contiguous string of any characters, except carriage return, end of message, start of message, erase, and partial message, bounded by a closed, balanced pair of left and right quotes.
lln	an actor whose value is the line number of the last line in the open file.

<u>new</u>	a type of argument which is an undefined symbol.
<u>numeric atom</u>	a type of atom consisting of a contiguous string of numeric characters.
<u>open file</u>	one of two file states, in which a file can be modified--or that file (only one file may be open at any given time) which is in that state.
<u>operator</u>	a type of symbol which is a request for a specific action by the EDITOR.
<u>symbol</u>	an atom of type variable or type independent which appears in the symbol table, either initially or as a result of the user's commands, and carries with it an associated type and value.
<u>symbol argument</u>	a type of argument which is any symbol which appears in the symbol table, and which is not of type undefined.
<u>symbol table</u>	a segment of the EDITOR which stores the type, value, and character representation of all symbols used by the EDITOR in the translation of the control language.
<u>symbol type</u>	the logical category in which the symbol belongs or is classified; the EDITOR recognizes eight types of symbols.
<u>symbol value</u>	the value of a symbol depends upon the type of the symbol and may be either a subroutine call to perform an action upon the user's file, or a subroutine call to perform an action upon the symbol table itself, or a number used during the execution of an EDITOR action, or a subroutine call to compute a number, or a pointer to a file location.
<u>undefined symbol</u>	either a symbol which does not appear in the symbol table, or a symbol which appears in the symbol table as being of type undefined--such a symbol has no value associated with it.

variable atom	a type of atom consisting of a contiguous string of numeric or alphabetic characters, at least one of which is of type alphabetic.

SECTION I

INTRODUCTION

EDITOR I is an on-line program within the PHOENIX computer software system which gives the system user complete file manipulation capabilities upon symbolic data, by the issuance of commands through a PHOENIX typewriter.

This report serves as a user's guide or manual to EDITOR I, and is generally concerned with those features which remain invariant, no matter under what monitor system EDITOR I is operating.

The user's data will be referenced by the program as a character string delineated and ordered into lines by carriage returns. Such a collection of lines is called a file and is referenced by a symbolic file name.

The actions of the EDITOR are user-controlled by means of a one-pass assembler, described in sections IV and V of this report.

SECTION II

BRIEF DESCRIPTION

2.1 Properties

EDITOR I is an on-line, user-controlled, file manipulator program, providing symbol definition capability and complete editing functions.

2.2 Limitations

The present machine-implemented version of the EDITOR subjects the user's data to the following restrictions:

a) a single file may contain no more than 4,0946 lines;

b) the symbol table will accommodate approximately 200 user-defined, variable-length, symbol entries;

c) the number of separate files allowed, at any one time, depends upon the room for entries in the symbol table; and

d) all the files together, existing in the system at any one time, may contain no more than 184,320 characters.

2.3 Current Status

The EDITOR is operational.

2.4 Operation

The EDITOR will perform manipulative functions upon data and data printouts, as requested by the user, in sequence and through commands typed on a PHOENIX typewriter attached to the PHOENIX computer, subject to legality checks.

SECTION III

DATA STRUCTURE

3.1 Lines and Line Numbers

A line is a string of characters, the last, and only the last, character of which must be a carriage return.

Each line has associated with it a numeric quantity called the line number. The line number is the user's primary means of accessing a line within a given file. The line number of a given line may be computed at any time by counting the number of lines between the given line and the start of the file. The first line in a file is line 0 (zero).

If line 'n' is deleted from a file, the line numbers of all lines from 'n' through the last line are decreased by 1. If a line is inserted into a file at line number 'n', all line numbers from 'n' through the last line number are increased by 1.

3.2 Blocks

A block is a group of one or more lines which have contiguous line numbers. A block is specified by the line numbers of the first and last lines in the block, in that order.

3.3 Files

A file is a block or blocks of lines which the user may access through a file name (see 3.3.2). A single file may contain no more than 4,096 lines.

3.3.1 File Structure

The <u>file structure</u> is a general term meaning the entire character string being handled by the EDITOR, and may comprise both accessible files and blocks of lines which are no longer accessible, due to the user's commands. The entire file structure, at any one time, may contain no more than 184,320 characters. (Facilities for garbage collection upon the file structure are available in the command repertoire.) Inaccessible lines in the file structure are caused by destroyed files and the rewriting of altered files.

3.3.2 File Names

A <u>file name</u> is an unique alphanumeric character sequence by which a file is accessed by the system user. The user may change the name of any file or combine the contents of several files in a straightforward manner by means of the command repertoire.

3.3.3 File States

A file can only be in one of two states, <u>open</u> or <u>closed</u>. A file may be modified only when it is in the open state, and only one file may be open at any given time.

3.4 Syntactical Chart

Meta-linguistically, we restate the above definitions of the data structure as follows (in Backus normal form*):

*See Appendix V for a description of Backus normal form.

```
< ↵ >::= carriage return

<character>::= any typewriter character, red or black,

               upper or lower case, except < ↵ >

<line>::= <character> < ↵ >|<character> <line>

<block>::= <line>|<block> <line>

<file>::= <block>|<file> <block>
```

SECTION IV

THE CONTROL LANGUAGE

This section contains a description of the syntax or rules of formation of the control language through which the user issues commands to the EDITOR. A word of caution is in order at this point--in no way should the syntax of the data structure or user's files, as described in Section III, be confused with the syntax of the control language through which the EDITOR is commanded.

As characters are typed and placed into the input buffer, the EDITOR segments the characters according to their properties and groups them into larger and larger logical units, called atoms, symbols, arguments, actors, and commands. These logical units or subdivisions of the control language are analogous to the parts of speech of natural language.

Basically, there are four types of atoms which can form eight types of symbols which, in turn, give rise to the various arguments, actors, and commands.

Associated with a symbol are the two characteristics, "type" and "value." The type and value of a symbol should not be confused with each other. The type of a symbol is the logical category in which it belongs or is classified (much as a word is either a noun or adverb or another one of the parts of speech), while the value of a symbol is the meaning associated with it, or bound to it, by convention or decree.

For example, the word "quarter" denotes a certain <u>type</u> of coin which has a value of twenty-five cents, while the word "two bits" denotes any number of coins whose total <u>value</u> is twenty-five cents.

In much the same way, the character string "three" in the EDITOR can be defined by the user to be a symbol of type constant with the value of 3, while the character string "constant" is defined by the EDITOR to be a symbol of type constant with the value of 3.

4.1 <u>Character Set</u>

The character set of the input string to the control language is that of the MITRE ball 2 for the PHOENIX typewriter.

The following notation symbolizes the non-printing function keys of the PHOENIX typewriter (in Backus normal form):

< ↵ >::= carriage return

< ϶ >::= erase

< ↦ >::= tabulate

< ↧ >::= index

< Ǝ >::= end of message

< Σ >::= start of message

< ⁋ >::= partial message

< □ >::= space

< ᗺ >::= backspace

The following characters are recognized by the EDITOR by class:

Class	Member
numeric	0 1 2 3 4 5 6 7 8 9
punctuation	+ - * ( ) , ; $ ` ´
spacing	□ ↦
carriage return	⤸
start of message	Σ
partial message	ᑫ
erase	϶
undefined	∂ ∃ and all illegal codes
alphabetic	all others

4.2 Character Source Mechanism

The character source mechanism of the EDITOR processes the input character string into commands (according to rules of formation to be described) and the EDITOR executes these commands in sequence as they appear. However, the typewriter will remain in input status until a carriage return is struck, and more than one command may be typed across the page, properly separated by command separators, of course. No processing of the commands will occur until the user types a carriage return. If there is no typeout from the typewriter to indicate that a command has been performed, a carriage return will be performed. If an illegal command or an incorrectly formed command appears, appropriate error messages will be typed out, and the character source mechanism will go to the next command, or, if there is none, return the typewriter to input status awaiting a new input from the user. Meaningless alphabetic input character strings cause a "?" to be typed out.

The character source mechanism will trap all carriage returns, removing them from the input string, and then return the typewriter to the user for a new input. This has been done to allow the user flexibility in formatting in certain commands. In the case where a literal character string is to be processed as a line image (for adding to or inserting into a file, for example), the character source mechanism will insert the carriage return, at the end of the line image, for the user.

The detection of an undefined character in the input string will cause an error message to be typed out.

During the operation of the EDITOR, any striking of the start of message key during typeout will cause the EDITOR to "shut up" and return to the character source mechanism.

When the user is typing inputs, hitting the partial message key will remove the last character typed into the input buffer and echo it back in red; if all characters are removed, subsequent depressing of the partial message key will cause a carriage return to be performed. Hitting the erase key removes the entire line from the input buffer and also causes a carriage return to be performed.

4.3 Typewriter Dialect

At present, the "typewriter dialect" expected by the EDITOR in its dialogue with the user is "lower case black" except where user definitions and syntax override it. These exceptions are duly described.

The EDITOR responds in "lower case red" except when typing out files, symbols, and so forth, where the character color is considered to be part of the "name" or line "image," in which case the actual color of the character is typed out.

4.4 Segmentation

The characters of the input string are segmented into units, called atoms, according to the rules described below. Atoms are the lowest logical and informational groups processed by the control language.

4.5 Atom Types and Rules of Formation

4.5.1 Variable Atoms

A variable atom is a contiguous string of characters of type numeric or alphabetic, at least one of which is of type alphabetic. A variable atom is delimited by a character of type punctuation or type spacing.

4.5.2 Numeric Atoms

A numeric atom is a contiguous string of characters of type numeric, delimited by a character of type punctuation or type spacing.

4.5.3 Independent Atoms

All characters of type punctuation are <u>independent atoms</u>. An independent atom is a self-delimiting atom, an atom which stands by itself and which also acts as a delimiter for other types of atoms.

4.5.4 Literal Atoms

A <u>literal atom</u> is a contiguous string of any characters, except carriage return, end of message, start of message, erase, and partial message, bounded by a closed, balanced pair of left and right quotes (' and '), the left quote being to the left of the literal, the right quote being to the right of the literal. All characters between the quotes, including spaces and quotes, are part of the literal. The EDITOR keeps count of left and right quotes and will consider the literal ended when the number of right quotes equals that of the left quotes.

Note that a literal atom follows the syntax of the data structure (see Section 3.4) as far as character case and color are concerned.

(A future implementation of the EDITOR will change the rules of formation for literal atoms but at present the above is the one that is operational.)

4.6 Syntactical Chart

Meta-linguistically, we restate the above definitions for atom formation as follows:

<n-char>::= 0|1|2|3|4|5|6|7|8|9

<p-char>::= +|-|*|(|)|,|;|$|`|´

<s-char>::= □|⊢

< ⅄ >::= carriage return

< Σ >::= start of message

< ꟼ >::= partial message

< ϶ >::= erase

<u-char>::= 8|∃| and any illegal code

<a-char>::= all others

<i-atom>::= <p-char>

<n-atom>::= <n-char>|<n-char> <n-atom>

<v-atom>::= <a-char> <n-atom>|<a-char>|<a-char> <v-atom>|
<n-char> <v-atom>

<l-atom>::= `any string of characters except < ⅄ > and < Σ > and
< ∃ > and < ϶ > and < ꟼ >´|`<l-atom>´

4.7 Symbols and the Symbol Table

A symbol is an atom of type variable or type independent which appears in the symbol table, either ab initio or as a result of the user's commands. (Symbols in the symbol table, ab initio, are considered to be reserved symbols and cannot be removed or modified.)

The symbol table is a segment of the EDITOR which stores the type, value, and character representation of all symbols used by the EDITOR in the translation of the control language.

The value of a symbol depends upon the type of the symbol and may be either a subroutine call to perform an action upon the user's file, or a subroutine call to perform an action upon the symbol table itself, or a number used during the execution of an EDITOR action, or a subroutine call to compute a number, or a pointer to a file location.

4.8 Symbol Types, Values, and Definitions

4.8.1 Symbol Types

The EDITOR recognizes eight basic types of symbols:

1) operator;
2) binary connector;
3) actor;
4) constant;
5) argument separator;
6) command separator;
7) file name; and
8) undefined.

4.8.2 Operators

A symbol of type operator is a request for a specific action by the EDITOR. A complete list of operators will be found in Section VI of this report.

4.8.3 Binary Connector

The characters "+" and "-" are symbols of type binary connector and have the normal arithmetic meaning.

4.8.4 Actor

An actor is a symbol which has a computed numeric value which is some line number (see Section 3.1) of the open file, and is computed each time the actor appears. Actors have a value only when there is an open file. At other times, actors are treated as undefined symbols.

The EDITOR recognizes the following actors with their associated meanings:

lln	the value is the line number of the last line in the open file.
cln	the value is the line number of the current line in the open file ($cln < lln + 1$), and represents the contents of a register pointing to some line in the open file.
flag(k)	locates a given flagged line, and the value is equal to the line number of the flagged line (k must be either a symbol of type constant, see below, or a number $0 \leq k \leq 31$). See Section 6.25 for a description and example of a flagged line.
config ('literal')	locates a line number by a character-by-character search of the open file, seeking the first appearance of the literal argument. The search begins with the cln and wraps around from the last line to the first, stopping the search at the cln. The value of the config actor is the line number of the first line encountered which contains the character string literal. Note that the parentheses and quote marks are not part of the literal. If no value can be found, an appropriate error message is typed out.

4.8.5 Constant

A constant is a symbol which has a numeric value. The value of a constant may be set by the user with the appropriate command. The following constants have been defined, *ab initio*:

Symbol	Numeric Value
operator	0
binaryconnector	1
actor	2
constant	3
operandsep	4
commandsep	5
filename	6
undefined	7
(	20
)	19

These constants may be used as arguments for those commands requiring constants as arguments, and are used internally by the EDITOR.

4.8.6 Argument Separator

The character "," (comma) is defined as a symbol of type argument separator, and must appear between arguments in those commands which take more than one argument.

4.8.7 Command Separator

The character ";" (semi-colon) is defined as a symbol of type command separator, and must be the last character in the argument string of a command, including the null argument string.

4.8.8 File Name

A file name is a symbol which must be an atom of type variable and which is used to reference the user's file. File names must be unique. A symbol may be defined as a file name through the use of certain commands in the command repertoire.

4.8.9 Undefined Symbols

There are two types of undefined symbols. The EDITOR does not distinguish between them.

1) A symbol which does not appear in the symbol table is undefined.

2) A symbol which appears in the symbol table as type undefined is also an undefined symbol.

Undefined symbols will appear, either directly or indirectly, as a result of the user's commands, and may be listed and garbage-collected out of the symbol table.

4.9 Syntactical Chart

Meta-linguistically, the preceding definitions for symbols are restated as follows:

```
<symbol character string>::= <v-atom>|<i-atom>
<operator>::= <symbol character string> defined as type operator
<binary connector>::= +|-| <symbol character string> defined
          as type binary connector
<constant>::= operator|binaryconnector|actor|constant|operandsep|
          commandsep|filename|undefined|(|)|<symbol character
          string> defined as type constant
<actor>::= lln|cln|flag(<constant>)|flag(<n-char>)|flag(<n-char>
          <n-char>)|config(<l-atom>)|<symbol character string>
          defined as type actor
<argsep>::= ,|<symbol character string> defined as type argument
          separator
<commandsep>::= ;|<symbol character string> defined as type
          command separator
<file name>::= <v-atom> defined as type file name
<defined symbol>::= <operator>|<binary connector>|<actor>|
          <constant>|<argsep>|<commandsep>|<file name>
<undefined symbol>::= <symbol character string> not in the
          symbol table |<symbol character string> in the
          symbol table defined as type undefined
<symbol>::= <defined symbol>|<undefined symbol>
```

SECTION V

COMMAND STRUCTURE

5.1 Definitions of Command

A command is a user request for a specific action to be performed by the EDITOR. The acceptable formats are:

1) operator;

2) operator argument; and

3) operator argument1, argument2,..., argumentn;

The ";" (semi-colon) is necessary. It is a signal to the EDITOR to begin execution of a command. The "," (comma) must be used to separate arguments.

5.2 Arguments

An argument is an atom or group of atoms used as a control parameter by EDITOR commands.

5.3 Argument Types

5.3.1 Expressions

An expression element is a numeric atom, a constant, or an evaluable actor, two or more of which may be concatenated by binary connectors to form expressions. A binary connector may begin, but never end, an expression.

5.3.2 Examples of Expressions

-24 + a + b is an expression

-24 + a + b - is not an expression

5.3.3 File Names

The characteristics of file names have been discussed in Section 4.8.8.

5.3.4 New

New arguments are symbols of type undefined (see Section 4.8.9).

5.3.5 Symbols

An argument of type symbol is any symbol which appears in the symbol table and is not of the type undefined.

5.3.6 Literals

A literal argument is a literal atom (see Section 4.5.4).

5.4 Syntactical Chart

Meta-linguistically, the preceding definitions for argument formation can be restated:

```
<expression element>::= <n-atom>|<constant>|<actor>

<expression>::= <expression element>|<binary connector>
        <expression>|<expression> <binary connector> <expression
        element>

<new>::= <undefined symbol>

<argument>::= <expression>|<file name>|<new>|<defined symbol>|
        <l-atom>

<argument sequence>::= <argument>|<argument sequence> <argsep>
        <argument>

<command>::= <operator> <commandsep>|<operator> <argument sequence>
        <commandsep>
```

SECTION VI

EDITOR COMMANDS

6.1 Command Notation

The following notation is used to describe the EDITOR's command repertoire. Any deviation from the notation will be justified as being *prima facie* understandable.

Expressions are abbreviated in a command description as En, where n is a number indicating the order in which the various expressions appear.

When a command requires arguments of type file name, the arguments are abbreviated as NAMEn, where n is a sequence number.

New arguments are indicated in the argument list of a command by the word NEW and are symbols of type undefined.

Arguments of type symbol (defined) are indicated as SYMBOL, with the exception of file names.

The use of a literal argument is indicated by the appearance of the word LITERALn, where n is a sequence number.

Two broad classes of legality under which most of the commands are either operable or inoperable depend upon whether there is or is not an open file. The EDITOR will always advise the user, through appropriate error messages, if a command is not acted upon for this reason.

The following sections comprise a complete list of all user commands to the EDITOR.

6.2 <u>listdirectory;</u>

(list the directory)

1) Arguments

   No arguments are necessary.

2) Actions

   All file names of existing files are typed out.

6.3 <u>listallfiles;</u>

(list all files)

1) Arguments

   No arguments are necessary.

2) Actions

   a) FILE OPEN

      'listallfiles' is an illegal command when there is an open file.

   b) FILE CLOSED

      The complete contents of all defined files are typed out with their associated file names and line numbers.

6.4 listfile NAME;

(list file)

1) Arguments

'NAME' must be a symbol of type file name.

2) Actions

a) FILE OPEN

'listfile' is an illegal command when there is an open file.

b) FILE CLOSED

The file referenced by the symbol 'NAME' is typed out with the proper line numbers.

6.5 listsymboltable;

(list the symbol table)

1) Arguments

No arguments are required.

2) Actions

Every symbol in the symbol table and its type, as well as value in the case of constants, is typed out.

6.6 listsymbols TYPE;

(list symbols)

1) Arguments

As shown in the list below, 'TYPE' must be either an expression whose value is in the range from 0 through 7, or a predefined constant:

Constant	Numeric Value
operator	0
binaryconnector	1
actor	2
constant	3
operandsep	4
commandsep	5
filename	6
undefined	7

2) Actions

All symbols of the type specified by 'TYPE' are typed out. Constants have their values typed out as well.

3) Example

The commands 'listsymbols 2;' and 'listsymbols actor;' are identical commands to the EDITOR, and will cause all symbols of type actor to be typed out.

6.7 <u>purgesymboltable;</u>

(purge the symbol table)

1) Arguments

No arguments are required.

2) Actions

a) FILE OPEN

The 'purgesymboltable' command is illegal when there is an open file.

b) FILE CLOSED

All undefined symbols are removed from the symbol table. The EDITOR will type out an appropriate warning message whenever the symbol table is full and an attempt is made to add another symbol.

6.8 <u>openfile NAME;</u>

(open file)

1) Arguments

'NAME' must be a symbol of type file name.

2) Actions

a) FILE OPEN

The currently open file is closed, proceed to b).

b) FILE CLOSED

The file referenced by the symbol 'NAME' is placed in the open (read and write) state.

6.9 <u>closefile;</u>

(close file)

1) Arguments

No arguments are required.

2) Actions

a) FILE OPEN

The currently open file is closed (placed in read only status).

b) FILE CLOSED

The 'closefile' command is ignored.

6.10 <u>typecln;</u>

(type current line number)

1) Arguments

No arguments are required.

2) Actions

a) FILE OPEN

The contents of the current line number register are typed out as shown below:

```
.. the current line number is n
```

b) FILE CLOSED

'typecln' is an illegal command when there is no open file.

6.11 <u>setcln E1;</u>

(set current line number)

1) Arguments

'E1' must be an evaluable expression.

2) Actions

a) FILE OPEN

The value of 'E1' replaces the contents of the current line number register.

b) FILE CLOSED

'setcln' is an illegal command when there is no open file.

6.12 dcon NEW, E1;

(define constant)

1) Arguments

   a) 'NEW' must be a symbol of type constant (which is not a reserved symbol) or a symbol of type undefined.

   b) 'E1' must be an evaluable expression.

2) Actions

   a) FILE OPEN

      If undefined, the symbol 'NEW' is added to the symbol table as a symbol of type constant, whose value is equal to the value of 'E1'. If 'NEW' is already defined as a symbol of type constant, the value of 'E1' replaces the current value of the constant.

   b) FILE CLOSED

      Identical to section a).

3) Example

```
dcon test,2;

dcon test, test+test;

setcln test;

typecln;

.. the current line number is 4.
```

6.13 <u>typelln;</u>

(type last line number)

1) Arguments

No arguments are required.

2) Actions

a) FILE OPEN

The line number of the highest numbered line in the open file is typed out in the following form:

```
.. the last line number is n
```

b) FILE CLOSED

'typelln' is an illegal command when there is no open file.

6.14 <u>setudf SYMBOL;</u>

(set undefined)

1) Arguments

'SYMBOL' must not be a reserved symbol.

2) Actions

'SYMBOL' is defined or redefined as a symbol of type undefined.

<u>CAUTION</u> - through this command, file names can be declared to be of type undefined, and access to the file can thus be lost.

6.15 <u>dsynon NEW, SYMBOL;</u>

(define synonym)

1) Arguments

a) 'NEW' must not be a reserved symbol.

There are no restrictions as to the type of 'NEW'.

b) 'SYMBOL' must not be a file name.

2) Actions

The symbol 'NEW' is defined to have the same meaning (type and value) as 'SYMBOL'.

3) Example

```
dsynon num, typecln;

dsynon D, dsynon;

D*,;;

num*

.. the current line number is 24 .
```

6.16 <u>rename OLDNAME, NEWNAME;</u>

(rename file)

1) Arguments

   a) 'OLDNAME' must be a file name.

   b) 'NEWNAME' must be a symbol of type undefined.

2) Actions

   a) FILE OPEN

      The name of the file 'OLDNAME' is changed to 'NEWNAME'. The symbol 'OLDNAME' is redefined to be a symbol of type undefined. If 'OLDNAME' is the name of the open file, the name of the open file is changed as above.

   b) FILE CLOSED

      Identical to a) above.

6.17 <u>destroy NAME;</u>

(destroy file)

1) Arguments

'NAME' must be a symbol of type file name.

2) Actions

The access to the file which is referenced by the file name 'NAME' is destroyed. The symbol 'NAME' is redefined to be of type undefined.

6.18 <u>purgefiles;</u>

(purge files)

1) Arguments

No arguments are required.

2) Actions

a) FILE OPEN

The currently open file is closed, proceed to b).

b) FILE CLOSED

All lines not associated with a file name are removed from the file structure. Lines not associated with a file name are caused by destroyed files and the rewriting of altered files.

The EDITOR will type out an appropriate warning message whenever the space allotted for the file structure is nearly filled.

6.19 <u>type E1;</u> or <u>type E1,E2;</u>

(type lines)

1) Arguments

a) 'E1' and 'E2' must be less than or equal to the last line number.

b) 'E2' must be equal to or greater than 'E1'.

2) Actions

a) FILE OPEN

The lines in the open file whose numbers are in the range from 'E1' through 'E2' are typed out with their associated line numbers. If 'E1' is the only argument, then that single line is typed out.

b) FILE CLOSED

'type' is an illegal command when there is no open file.

6.20 replace E1, LITERAL1;

(replace lines)

or

replace E1, LITERAL1, LITERAL2,...,LITERALn;

or

```
replace E1, LITERAL1,
            LITERAL2,
              ...
            LITERALn;
```

1) Arguments

a) 'E1' must be equal to or less than the last line number +1.

b) 'LITERAL1', 'LITERAL2',..., 'LITERALn' must be atoms of type literal (see Section 4.5.4).

NOTE: the argument sequence of literals may be typed across the page, or typed underneath each other in linear mode. This flexibility in formatting is due to the fact that the EDITOR traps all carriage returns, and also inserts them for the user as the last character in a literal that is to be processed as a line image (see Section 4.2). No literal may contain a carriage return.

This flexibility of formatting and the rules in the cited sections as to literal atom formation apply to all commands using literals as arguments.

2) Actions

a) FILE OPEN

The value of 'E1' replaces the contents of the current line number register. 'LITERAL1' is converted into a line which replaces the line whose line number is equal to the contents of the current line number register. The contents of the current line number register are then incremented by 1, and the above process is repeated with the next literal, until a symbol of type commandsep is detected (in this case, it is the ";"). The EDITOR then proceeds to the next command.

b) FILE CLOSED

'replace' is an illegal command when there is no open file.

c) Example

```
        type 10,13;
10      file line 10
11      file line 11
12      file line 12
13      file line 13
        replace 11, 'replace line 1',
                    'replace line 2';
        type 10,13;
10      file line 10
11      replace line 1
12      replace line 2
13      file line 13
```

6.21 create NEW;

(create file)

1) Arguments

'NEW' must be a symbol of type undefined.

2) Actions

a) FILE OPEN

The open file is closed, and b) is executed.

b) FILE CLOSED

The symbol 'NEW' is defined to be a symbol of type file name, and space is allocated for a new file with that name.

WARNING - the next command *must* be

replace 0, LITERAL and so forth

(see Section 6.20) in order to put something into the new file, starting at line zero. Failure to observe this rule will cause the EDITOR to write 'EMPTY FILE' as line zero. The new file is left in the open state at the conclusion of the 'create' command.

6.22 wipelastline;

(wipe out the last line)

1) Arguments

No arguments are required.

2) Actions

a) FILE OPEN

The highest numbered line of the open file is destroyed.

b) FILE CLOSED

The 'wipelastline' command is ignored.

6.23 merge NEW,NAME1,E11,E12, NAME2,E21,E22,...,NAMEn,En1,En2;

(merge lines)

1) Arguments

   a) 'NEW' must be a symbol of type undefined.

   b) All 'NAMEi' must be the names of defined files. A given 'NAMEi' may appear any number of times in the argument sequence.

   c) The expressions 'Ei1' and 'Ei2' must define blocks of lines contained in the file named 'NAMEi'.
   Note that 'Ei1' and 'Ei2' may not contain actors, as there is no open file during the merge operation.
   Note also that the command format permits the same flexibility as the 'replace' command (see Section 6.20).

2) Actions

   a) FILE OPEN

   The open file is closed, proceed to b).

   b) FILE CLOSED

   The blocks of lines defined by 'Ei1' and Ei2' are copied from the file 'NAMEi' into the newly defined file 'NEW', in order of their appearance in the argument sequence.
   At the end of the 'merge', the file 'NEW' is in the open state.

6.24 swap E1,E2,E3;

(swap lines)

1) Arguments

   a) 'E2' must be equal to or greater than 'E1'.

   b) The block defined by 'E3', ('E3' + 'E2' - 'E1') as well as the block defined by 'E1', 'E2', must be within the line number range of the open file. The two blocks may not share any line numbers.

2) Actions

   a) FILE OPEN

      The lines in the block defined by 'E1', 'E2' are exchanged with the lines in the block defined by 'E3', ('E3' + 'E2' - 'E1').

   b) FILE CLOSED

      'swap' is an illegal command when there is no open file.

3) Example

   (See section 6.25.)

6.25 setflag K,E1;

(set flag)

1) Arguments

   a) 'K' must be a symbol of type constant or an atom of type numeric.

      'K' is taken modulo 32.

   b) 'E1' must be an evaluable expression.

2) Actions

   a) FILE OPEN

      The flag whose flag number is 'K' is bound to, and identified with, the line whose line number is 'E1', regardless of previous definitions.

   b) FILE CLOSED

      'setflag' is not a legal command when there is no open file.

3) Example

   a) Sample Open File

```
00000  dac alpha,6
    1  last line
```

b) User Command Sequence

```
        setflag 1,1;
        type flag(1);
     1  last line
        swap 0,0,1;
        type lln-1,lln;
 00000  last line
     1  dac alpha,6
        type flag(1);
 00000  last line
```

6.26 typeflags;

(type out flags)

1) Arguments

No arguments are required.

2) Actions

a) FILE OPEN

A table of all 32 flag numbers is typed out with the current state of each flag. If a flag is set and the associated line exists, then the line and its current line number are also typed out.

b) FILE CLOSED

'typeflags' is an illegal command when there is no open file.

6.27 <u>delete E1; or delete E1,E2;</u>

(delete lines)

1) Arguments

   a) 'E2' must be equal to or greater than 'E1'.

   b) 'E1' and 'E2' must both be equal to or less than the highest line number.

2) Actions

   a) FILE OPEN

   The block of lines specified by 'E1' and 'E2' are removed from the open file. The line numbers of the lines from ('E2' +1) through the last line number are decremented by ('E2' - 'E1' + 1). If 'E1' is the only argument, then that is the only line deleted.

   b) FILE CLOSED

   'delete' is an illegal command when there is no open file.

6.28 insert E1,LITERAL1,LITERAL2,...,LITERALn;

(insert lines)

1) Arguments

   a) 'E1' must be equal to or less than the last line number.

   b) 'LITERAL1', 'LITERAL2',...,'LITERALn' must be atoms of type literal (see sections 4.2, 4.5.4, and 6.20).

2) Actions

   a) FILE OPEN

      The value of 'E1' replaces the contents of the current line number register, so that insertion of the lines into the file occurs just before 'E1'. The line numbers of the lines from 'E1' through the last line number are increased by 1, and the literal 'LITERAL1' is converted into a line whose line number equals the contents of the current line number register. The current line number register is then incremented by 1, and the above process is repeated with the next literal, until a symbol of type commandsep is detected (in this case, it is the ";"). The EDITOR then processes the next command.

   b) FILE CLOSED

      'insert' is an illegal command when there is no open file.

3) Example

```
    type 10,12;
10  file line 10
11  file line 11
12  file line 12
    insert 11, 'insert line 1',
               'insert line 2';
    type 10,14;
10  file line 10
11  insert line 1
12  insert line 2
13  file line 11
14  file line 12
```

6.29 dupinsert E1,E2,E3;

(duplicate and insert)

1) Arguments

a) Definitions

1 - 'E1' and 'E2' are, respectively, the lowest and highest line numbers of block 1.

2 - 'E3' and ('E3' + 'E2' - 'E1') are, respectively, the lowest and highest line numbers of block 2.

b) Restrictions

1 - block 1 and block 2 may not share any line numbers.

2 - the line numbers of block 1 must be in the range from 0 through the highest line number.

3 - 'E3' must be equal to or less than the last line number.

2) Actions

a) FILE OPEN

The lines currently occupying block 1 are duplicated and written into a gap in the open file, created by incrementing the line numbers of the lines from 'E3' through the last line number, by the quantity ('E2' - 'E1' + 1), so that insertion occurs just before 'E3'.

b) FILE CLOSED

'dupinsert' is an illegal command when there is no open file.

3) Example

```
        type 0,lln;
00000   test line 1
    1   test line 2
    2   test line 3
    3   test line 4
    4   test line 5
    5   test line 6
        dupinsert 0,1,4;
        type 0,lln;
00000   test line 1
    1   test line 2
    2   test line 3
    3   test line 4
    4   test line 1
    5   test line 2
    6   test line 5
    7   test line 6
```

6.30 dupreplace E1,E2,E3;

(duplicate and replace)

1) Arguments

a) Definitions (same as 'dupinsert')

1 - 'E1' and 'E2' are, respectively, the lowest and highest line numbers of block 1.

2 - 'E3' and ('E3' + 'E2' - 'E1') are, respectively, the lowest and highest line numbers of block 2.

b) Restrictions

1 - block 1 and block 2 may not share any line numbers.

2 - the line numbers of block 1 must be in the range from 0 through the highest line number.

3 - 'E3' must be equal to or less than the last line number+1.

2) Actions

a) FILE OPEN

The lines currently occupying block 1 are duplicated and written into the open file, replacing the lines contained in block 2.

b) FILE CLOSED

'dupreplace' is an illegal command when there is no open file.

3) Example

```
        type 0,11n;
00000   test line 1
    1   test line 2
    2   test line 3
    3   test line 4
    4   test line 5
    5   test line 6
        dupreplace 0,1,4;
        type 0,11n;
00000   test line 1
    1   test line 2
    2   test line 3
    3   test line 4
    4   test line 1
    5   test line 2
```

6.31 <u>transinsert E1,E2,E3</u>;

(transmit and insert)

1) Arguments

a) Definitions (same as 'dupinsert')

1 - 'E1' and 'E2' are, respectively, the lowest and highest line numbers of block 1.

2 - 'E3' and ('E3' + 'E2' - 'E1') are, respectively, the lowest and highest line numbers of block 2.

b) Restrictions (same as 'dupinsert')

1 - block 1 and block 2 may not share any line numbers.

2 - the line numbers of block 1 must be in the range from 0 through the highest line number.

3 - 'E3' must be equal to or less than the last line number.

2) Actions

a) FILE OPEN

The lines currently occupying block 1 are duplicated and written into a gap in the open file, created by incrementing the line numbers of the lines from 'E3' through the last line number, by the quantity ('E2' - 'E1' + 1) so that insertion occurs just before 'E3'. The lines in block 1 are then destroyed and all the line numbers are adjusted accordingly.

b) FILE CLOSED

'transinsert' is an illegal command when there is no open file.

3) Example

```
        type 0,11n;
00000   test line 1
    1   test line 2
    2   test line 3
    3   test line 4
    4   test line 5
    5   test line 6
        transinsert 0,1,4;
        type 0,11n;
00000   test line 3
    1   test line 4
    2   test line 1
    3   test line 2
    4   test line 5
    5   test line 6
```

6.32 transreplace E1,E2,E3;

(transmit and replace)

1) Arguments

a) Definitions (same as 'dupinsert')

1 - 'E1' and 'E2' are, respectively, the lowest and highest line numbers of block 1.

2 - 'E3' and ('E3' + 'E2' - 'E1') are, respectively, the lowest and highest line numbers of block 2.

b) Restrictions

1 - block 1 and block 2 may not share any line numbers.

2 - the line numbers of block 1 must be in the range from 0 through the highest line number.

3 - 'E3' must be equal to or less than the last line number+1.

2) Actions

a) FILE OPEN

The lines currently in block 1 are duplicated and written into the open file replacing the lines in block 2. The lines in block 1 are then destroyed.

b) FILE CLOSED

'transreplace' is an illegal command when there is no open file.

3) Example

```
      type 0,11n;
00000 test line 1
    1 test line 2
    2 test line 3
    3 test line 4
    4 test line 5
    5 test line 6
      transreplace 0,1,4;
      type 0,11n;
00000 test line 3
    1 test line 4
    2 test line 1
    3 test line 2
```

6.33 recover;

(recover file)

1) Arguments

No arguments are required.

2) Actions

a) FILE OPEN

The open file is restored to the state it was in immediately following the most recent action that put it into the open state. All destroyed lines originally present are reinserted into the file, all added lines are removed, and the lines are placed in their original sequence. WARNING - the 'recover' command must not be used if the open file is a newly created one, as a result of a 'create' operation, as such a file actually exists as a list of line pointers and will not be put into the file structure until it is closed. Failure to observe this rule will cause an empty file as described in the 'create' command. To avoid the loss of the new file, after a substantial number of lines have been entered, it is recommended that the file be closed and then re-opened. In this way, subsequent use of the 'recover' command will cause no loss of the file.

b) FILE CLOSED

'recover' is an illegal command when there is no open file.

6.34 space;

(space lines)

1) Arguments

No arguments are required.

2) Actions

All printouts of any file, whether open or closed, will be changed from single spacing to double spacing, or double spacing to single spacing, depending upon what spacing mode was being followed, initially.

3) Example

```
        type 0,1ln;

00000   first line
    1   last line

        space;

        type 0,1ln;

00000   first line

    1   last line

        space;

        type 0,1ln;

00000   first line
    1   last line
```

6.35 <u>numbers;</u>

(numbers)

1) Arguments

No arguments are required.

2) Actions

All printouts of any file, whether open or closed, will be single spaced with line numbers deleted, if they are present, or vice versa. This command overrides 'space' (see section 6.34) as far as spacing is concerned.

3) Example

```
        type 0,lln;

00000   lac alpha

    1   add beta

    2   dac gamma

        numbers;

        type 0,lln;

        lac alpha
        add beta
        dac gamma

        numbers;

        type 0,lln;

00000   lac alpha

    1   add beta

    2   dac gamma
```

6.36 <u>terminate;</u>

(terminate)

1) Arguments

No arguments are required.

2) Actions

If a file is open, it is closed. The directory of file names is saved, thus preserving current status of the files and symbol table, and control is then transferred to whatever monitor is running the system.

6.37 <u>reinitialize;</u>

(reinitialize system)

1) Arguments

No arguments are required.

2) Actions

The file structure and directory of file names are restored to the original condition they were in following the last 'purgefiles' command. All modifications to all files since then are canceled. All new files are destroyed. All user definitions are removed from the symbol table. It is obvious that this command should be used with due respect and caution.

6.38 listlinepointers;

(list line pointers)

1) Arguments

No arguments are required.

2) Actions

a) FILE OPEN

A list of all lines and their associated addresses in storage (not line numbers) of the open file is typed out. This command is not intended to be of value to the user but is a system debugging command.

b) FILE CLOSED

'listlinepointers' is an illegal command when there is no open file.

6.39 listfilestructure;

(list the file structure)

1) Arguments

No arguments are required.

2) Actions

All lines in the file structure are listed with octal pointers to the first character in each line. This command is not normally of any interest to the user and is a system debugging command.

APPENDIX I

THE EDITOR UNDER PEST CONTROL

Through simple typewriter commands, PEST will allow the user to call either the EDITOR or PAT into core from the drum--the drum having been initially loaded through a PEST tape.

The EDITOR has, as its exclusive province, certain drum fields for its files.

Upon a user command to PEST for magnetic tape input of files, PEST will write the files onto these drum fields and hand down to the EDITOR a drum list, giving the starting address and length of each file, up to a maximum of four files.

The very first time the EDITOR is loaded into core and activated by pressing the start of message key, the EDITOR will check the drum list, and, if correct, use it to build its directory of files. In either case, appropriate messages will be typed out.

Subsequent to the initial activation, the EDITOR, upon being loaded and started (by pressing the start of message key), refers to its already existing directory and merely indicates that it is ready, thus preserving its current status.

If during the interim since the last 'terminate' command (see Section 6.36) there has been a magnetic tape input of new files, the EDITOR must be alerted by the 'reinitialize' command (see Section 6.37), being the first command issued. Under PEST, the 'reinitialize' command should be used for nothing else, since using it during a system run will send the EDITOR to the outdated drum list.

It should be noted that the maximum of four files applies only to magnetic tape input. The number of separate files allowed, at any one time within the EDITOR, is a function of the symbol table and file structure (see Section 2.2).

After receiving and checking the drum list from PEST, the EDITOR will build its directory of files and, arbitrarily, name the first four files, in the order of their presence, "filea", "fileb", "filec", "filed". (Complete renaming, merging, file creating, and destroying facilities for the user are present in the EDITOR.)

When issued the user command 'terminate', the EDITOR will transfer control to PEST, after first preserving the current status of its files and symbol table. It will also write out onto a fixed drum location a copy of the most recently opened file, filled out with right parentheses (to signal the end of the file) for eventual input to PAT.

At the end of the run, PEST will allow the user to perform a tape dump of the entire system, which can be reloaded for the next run, thus preserving the current status of programs and files.

APPENDIX II

LIST OF PROGRAM RESERVED SYMBOLS

+

-

,

;

operator

binaryconnector

actor

constant

operandsep

commandsep

filename

undefined

lln

cln

flag

config

(

)

and all command names listed in Appendix III.

APPENDIX III

ALPHABETIC LIST OF COMMANDS

Command	Section	Page
closefile	6.9	25
create	6.21	35
dcon	6.12	27
delete	6.27	42
destroy	6.17	31
dsynon	6.15	29
dupinsert	6.29	45
dupreplace	6.30	47
insert	6.28	43
listallfiles	6.3	21
listallfiles	6.2	21
listfile	6.4	22
listfilestructure	6.39	57
listlinepointers	6.38	57
listsymbols	6.6	23
listsymboltable	6.5	22
merge	6.23	37
numbers	6.35	55
openfile	6.8	24
purgefiles	6.18	31
purgesymboltable	6.7	24

ALPHABETIC LIST OF COMMANDS (Cont.)

APPENDIX IV

TABLE OF COMMANDS BY FUNCTION

File State*	Command	Arguments	Section	Page
	File Status Commands			
A	closefile		6.9	25
A	create	NEW;	6.21	35
A	destroy	NAME;	6.17	31
A	merge	NEW, NAME1,E11,E12,..., NAMEn,En1,En2;	6.23	37
A	openfile	NAME;	6.8	24
O	recover		6.33	53
A	rename	OLDNAME, NEWNAME;	6.16	30
	File Printouts			
C	listallfiles		6.3	21
C	listfile	NAME;	6.4	22
O	type	E1; or E1,E2;	6.19	32

* O = open file
C = closed file
A = any state

TABLE OF COMMANDS BY FUNCTION (Cont.)

File State*	Command	Arguments	Section	Page
	Open File Manipulation			
O	delete	E1; or E1,E2;	6.27	42
O	dupinsert	E1,E2,E3;	6.29	45
O	dupreplace	E1,E2,E3;	6.30	47
O	insert	E1,LITERAL1,...,LITERALn;	6.28	43
O	replace	E1,LITERAL1,...,LITERALn;	6.20	33
O	swap	E1,E2,E3;	6.24	38
O	transinsert	E1,E2,E3;	6.31	49
O	transreplace	E1,E2,E3;	6.32	51
A	wipelastline		6.22	36
	Symbol Definition			
A	dcon	NEW,E1;	6.12	27
A	dsynon	NEW,SYMBOL;	6.15	29
A	setudf	SYMBOL;	6.14	28
	System Information			
A	listdirectory		6.2	21
A	listfilestructure		6.39	57

* O = open file
  C = closed file
  A = any state

TABLE OF COMMANDS BY FUNCTION (Cont.)

File State*	Command	Arguments	Section	Page
	System Information			
O	listlinepointers		6.38	57
A	listsymbols	TYPE;	6.6	23
A	listsymboltable		6.5	22
O	setcln	E1;	6.11	26
O	setflag	K,E1;	6.25	39
O	typecln		6.10	25
O	typeflags		6.26	41
O	typelln		6.13	28
	Format Commands			
A	numbers		6.35	55
A	space		6.34	54
	Garbage Collection			
A	purgefiles		6.18	31
C	purgesymboltable		6.7	24
	System Status			
A	reinitialize		6.37	56
A	terminate		6.36	56

* O = open file
  C = closed file
  A = any state

APPENDIX V

BACKUS NORMAL FORM

Backus normal form (B.N.F.) is a type of notation developed to describe unambiguously the syntax or rules of formation of a programming language. It is, essentially, a language for describing languages, or a meta-language. The most important aspect of Backus normal form is its recursive definition ability, which allows description of all the possible and syntactically correct character strings of the particular language being described.

The symbology of the notation consists of

" < ", " > ", "::=", and "|".

"|" is the "exclusive or" operator and is used on the right-hand side of a B.N.F. statement to distinguish between the different possible character strings.

" < " and " > " are used as the left and right delimiters, respectively, of a B.N.F. symbol.

"::=" is the B.N.F. "equals" or assignment operator, which sets the left-hand side of a B.N.F. statement equal to the right-hand side, with the provision that a B.N.F. symbol may appear on either side, thus providing recursive capability.

The B.N.F. statement

```
<digit>::= 0|1|2|3|4|5|6|7|8|9
```

means that a digit has been defined as 0 or 1 or 2 or 3 or 4 or 5 or 6 or 7 or 8 or 9.

The B.N.F. statement

```
<number>::= <digit>|<number> <digit>
```

means that a number has been defined as a digit or a number followed by a digit. The symbols <number> <digit> expand to <digit> <digit> or <number> <digit> <digit>, since a number has been recursively defined as either a digit or a number followed by a digit. Thus, with a rather simple B.N.F. statement, the definition of a number has been expressed as any possible string of digits of any possible length, as <number> <digit> may be recursively substituted for <number> as many times as necessary.

APPENDIX VI

FURTHER ILLUSTRATIVE EXAMPLES

1. Repetitive use of the following command

```
delete config('xyz');
```

will delete all lines where "xyz" occurs, from the open file, and the EDITOR will type out a message indicating that there are no further occurrences.

2. The following example is included with no further comment:

```
create fun;

replace 0, 'ZERO',
'ONE',
'TWO',
'THREE',
'FOUR';

type 0, lln;
00000  ZERO
    1  ONE
    2  TWO
    3  THREE
    4  FOUR
```

```
setcln 0; transinsert lln,lln,cln;

setcln cln+1; transinsert lln,lln,cln;

setcln cln+1; transinsert lln,lln,cln;

setcln cln+1; transinsert lln,lln,cln;

type 0,lln;

00000  FOUR

    1  THREE

    2  TWO

    3  ONE

    4  ZERO
```

Security Classification

DOCUMENT CONTROL DATA - R&D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY *(Corporate author)* The MITRE Corporation Bedford, Massachusetts	2a. REPORT SECURITY CLASSIFICATION Unclassified
	2b. GROUP

3. REPORT TITLE

Users' Manual For The EDITOR

4. DESCRIPTIVE NOTES *(Type of report and inclusive dates)*

N/A

5. AUTHOR(S) *(Last name, first name, initial)*

Isquith, Ben

6. REPORT DATE November 1966	7a. TOTAL NO. OF PAGES 80	7b. NO. OF REFS 0

8a. CONTRACT OR GRANT NO. AF 19(628)-5165	9a. ORIGINATOR'S REPORT NUMBER(S) ESD-TR-66-309
b. PROJECT NO. 508F	
c.	9b. OTHER REPORT NO(S) *(Any other numbers that may be assigned this report)* MTR-222
d.	

10. AVAILABILITY/LIMITATION NOTICES

Distribution of this document is unlimited.

11. SUPPLEMENTARY NOTES	12. SPONSORING MILITARY ACTIVITY Deputy For Engineering and Technology, Directorate of Computers; Electronic Systems Division, L. G. Hanscom Field, Bedford, Mass.

13. ABSTRACT

EDITOR I is an on-line program within the initial PHOENIX computer software system which enables the system user to create, destroy, or modify his collection of symbolic data, organized as files.

The actions of EDITOR I are user-controlled by means of a one-pass assembler, herein described.

DD FORM 1 JAN 64 1473

Security Classification

14. KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
COMPUTORS						
Programming, on-line						
EDITOR I, user's manual						

INSTRUCTIONS

1. ORIGINATING ACTIVITY: Enter the name and address of the contractor, subcontractor, grantee, Department of Defense activity or other organization *(corporate author)* issuing the report.

2*a*. REPORT SECURITY CLASSIFICATION: Enter the overall security classification of the report. Indicate whether "Restricted Data" is included. Marking is to be in accordance with appropriate security regulations.

2*b*. GROUP: Automatic downgrading is specified in DoD Directive 5200.10 and Armed Forces Industrial Manual. Enter the group number. Also, when applicable, show that optional markings have been used for Group 3 and Group 4 as authorized.

3. REPORT TITLE: Enter the complete report title in all capital letters. Titles in all cases should be unclassified. If a meaningful title cannot be selected without classification, show title classification in all capitals in parenthesis immediately following the title.

4. DESCRIPTIVE NOTES: If appropriate, enter the type of report, e.g., interim, progress, summary, annual, or final. Give the inclusive dates when a specific reporting period is covered.

5. AUTHOR(S): Enter the name(s) of author(s) as shown on or in the report. Enter last name, first name, middle initial. If military, show rank and branch of service. The name of the principal author is an absolute minimum requirement.

6. REPORT DATE: Enter the date of the report as day, month, year; or month, year. If more than one date appears on the report, use date of publication.

7*a*. TOTAL NUMBER OF PAGES: The total page count should follow normal pagination procedures, i.e., enter the number of pages containing information.

7*b*. NUMBER OF REFERENCES: Enter the total number of references cited in the report.

8*a*. CONTRACT OR GRANT NUMBER: If appropriate, enter the applicable number of the contract or grant under which the report was written.

8*b*, 8*c*, & 8*d*. PROJECT NUMBER: Enter the appropriate military department identification, such as project number, subproject number, system numbers, task number, etc.

9*a*. ORIGINATOR'S REPORT NUMBER(S): Enter the official report number by which the document will be identified and controlled by the originating activity. This number must be unique to this report.

9*b*. OTHER REPORT NUMBER(S): If the report has been assigned any other report numbers *(either by the originator or by the sponsor)*, also enter this number(s).

10. AVAILABILITY/LIMITATION NOTICES: Enter any limitations on further dissemination of the report, other than those imposed by security classification, using standard statements such as:

(1) "Qualified requesters may obtain copies of this report from DDC."

(2) "Foreign announcement and dissemination of this report by DDC is not authorized."

(3) "U. S. Government agencies may obtain copies of this report directly from DDC. Other qualified DDC users shall request through
_______________________________."

(4) "U. S. military agencies may obtain copies of this report directly from DDC. Other qualified users shall request through
_______________________________."

(5) "All distribution of this report is controlled. Qualified DDC users shall request through
_______________________________."

If the report has been furnished to the Office of Technical Services, Department of Commerce, for sale to the public, indicate this fact and enter the price, if known.

11. SUPPLEMENTARY NOTES: Use for additional explanatory notes.

12. SPONSORING MILITARY ACTIVITY: Enter the name of the departmental project office or laboratory sponsoring *(paying for)* the research and development. Include address.

13. ABSTRACT: Enter an abstract giving a brief and factual summary of the document indicative of the report, even though it may also appear elsewhere in the body of the technical report. If additional space is required, a continuation sheet shall be attached.

It is highly desirable that the abstract of classified reports be unclassified. Each paragraph of the abstract shall end with an indication of the military security classification of the information in the paragraph, represented as *(TS), (S), (C), or (U)*.

There is no limitation on the length of the abstract. However, the suggested length is from 150 to 225 words.

14. KEY WORDS: Key words are technically meaningful terms or short phrases that characterize a report and may be used as index entries for cataloging the report. Key words must be selected so that no security classification is required. Identifiers, such as equipment model designation, trade name, military project code name, geographic location, may be used as key words but will be followed by an indication of technical context. The assignment of links, rules, and weights is optional.