15

COMPOSE/COMPOSE
AN EXPERIMENTAL TEXT EDITOR

by

Robert G. Herriot

Computer Sciences Technical Report #18

May 1968

Compose/Compose

Compose/Compose is a text editing program designed with the philosophy of being fail-safe and extremely versatile. This manual is intended to describe all of the features of this program.

Starting Up

To initiate the program, the user types in ?EXECUTE COMPOSE←. After the BOJ, the program types out TYPE .COMMAND OR .OP FOR OPTIONS. The program then expects a string of commands to be typed in. If more than one command is typed in, then the commands are executed sequentially. All commands must be preceded by a period. The command .OP can always be typed in order to discover which commands are locally available. The following is a description of the available commands.

Options

<option> ::= .OP

This command lists all of the options which are available on this level, which is the main level to which all options return.

Example:
```
.OP←
OP  OPTIONS TYPED
CR  CREATE FILE
LD  LOAD FILE
PU  PUNCH FILE
PR  PRINT FILE
AS  ASSEMBLE FILE
AP  APPEND FILES
QT  QUIT
TV  TYPE VALUE
```

Quit

<quit> ::= .QT

This command terminates the program.

Create File

<create> ::= .CR <file>
<file> ::= see appendix 1

This command creates a file with the name given. If the file is unspecified, the program types out FILE UNSPECIFIED. If there is already a file of the same name on disk, the program types out: file name DUPLICATE. Otherwise, it types out: file name CREATED. The rest is the same as the description of Load File.

Examples:
```
.CR SMITH/NEWFILE←      create file SMITH/NEWFILE
.CR 1←                  create file defined as 1
```

Load File

<load> ::= .LD <file>

This command loads the file which is specified. If no file is specified, the program types out: FILE UNSPECIFIED. If there is no such file, the program types out: file name UNAVAILABLE. Otherwise, it types out: file name LOADED.

After a few seconds, the program types out: TYPE PROGRAM. On the next line it types a zero (0) and leaves the type head on the space after the 0. This zero is the presumed first sequence number of the program. The programmer may then either enter a card image or a string of commands. If a card already exists, then it can not be overwritten unless a flag is changed (see DS command). Thus the user is protected from accidently destroying a card image. The user will also receive a punched deck of all the changes which he makes to his file unless he turns off a flag (see PU command). It is possible to input characters which are not on the teletype keyboard. Appendix 2 explains the conversions which affects all input through the teletype. This option is terminated by typing .SV. The program then types out: WAIT FOR PROCESSING OF FILE. If the updated cards are to be punched, the program next types out: PUNCHING INITIATED ON punch file name. Finally the program types out: PROGRAM ON DISK NOW. The program is then ready for another command from the main level.

Appendix 3 explains all of the commands which are available under this option.

Examples:	.LD SMITH/NEWFILE=1←	load file SMITH/NEWFILE and define it as file 1
	.LD 1←	load file SMITH/NEWFILE which is file 1

Punch File

<punch> ::= .PU <file>

This command produces a punched deck of the file specified. One of the three messages is typed out: FILE UNSPECIFIED, file name UNAVAILABLE, or PUNCHING INITIATED ON file name.

Example:	.PU SMITH/NEWFILE←	punch file SMITH/NEWFILE

Print File

<print> ::= .PR <file>

This command is similar to .PU except that it prints the file on the line printer.

Example:	.PR SMITH/NEWFILE←	print file SMITH/NEWFILE

Resequence File

<resequence> ::= .RS <file><first number><step part>
<first number> ::= <empty>|<unsigned integer>
*<empty> = 0
<step part> ::= <empty>| BY <increment>
<empty> = 100
<increment> ::= <empty>|<unsigned integer>
<empty> = 100

This command resequences the specified file. Both the first number and the increment can be specified. The default condition assumes that the first card will be 00000000 and that the increment will be 100. On completion of the resequencing, the program types out: FILE file name RESEQUENCED. LAST SEQ. NO.: integer.

Examples:

.RS SMITH/NEWFILE←	resequence SMITH/NEWFILE starting with 0 and incrementing by 100
.RS A/B 2000BY1000←	resequence A/B starting with 2000 and incrementing by 1000

Append Files

<append> ::= .AP <file>

This command will create a file in the same way that .CR will create a file (same messages), but it will then expect a different set of commands. There are four commands available to build a new file from parts of old files.

Options

<option> ::= .OP

This command lists all of the options which are available at this sublevel.

Example:

```
.OP←
OP  OPTIONS TYPED
SV  SAVE FILE
OF  OLD FILE (.OF FILE NAME NUMBER TO NUMBER)
TV  TYPE VALUE
```

Save File

<save> ::= .SV

This command terminates the .AP option and saves the new file. The program types out: FILES APPENDED. It then waits for another command on the main level.

---

* The construct <empty> = ... will henceforth be used to specify the condition which is assumed if nothing is given. <empty>, as well as several other undefined items, have the usual Algol meaning.

Old File

<old file> ::= .OF <file><first seq. no.> TO <last seq. no.>
<first seq. no.> ::= <unsigned integer>
<last seq. no.> ::= <unsigned integer>

This command moves pieces of already existing files into the newly created file. The first and last sequence numbers provide the lower and upper bounds for these pieces. If the first sequence number does not exist, an error message: FIRST SEQ. NO. NOT ON FILE, is printed and nothing is transferred to the new file. If the end-of-file is encountered before finding the last sequence number, an error message: LAST SEQ. NO. NOT ON FILE, is printed, but the file has been correctly transferred to the new file.

Examples:

```
.OF  A/B 1200T02000←
```
move cards 1200 thru 2000 of file A/B to the new file.

```
.OF  2 0 TO 35600←
```
move cards 0 thru 35600 of file defined by 2 to the new file.

Type Value

<type value> ::= .TV <file list>
<file list> ::= <file list><file number>|<file number>
<file number> ::= FL | F<digit>

This command types out the files which have been defined by one digit numbers (see appendix 1). FL lists all files 0 thru 9 and F<digit> lists the file specified by the digit.

Example:

```
.TVF3F1←
3: SMITH  /NEWFILE
1: A      /B
```

Example:

```
.AP  SMITH/NEW←
SMITH  /NEW      CREATED.
OLD FILES:
.OF SMITH/FILE=1 300T0700←
.OF 1 4500 TO 5300←
.SV←
FILES APPENDED.
```
create file SMITH/NEW

put cards 300 thru 700 and cards 4500 thru 5300 of file SMITH/FILE into file SMITH/NEW and save it.

Type Value

<type value> ::= .TV <file list>

This command is described a few lines above and is identical to it, except that this command is on the main level.

Appendix 1

Specification of Files

Definition

<file> ::= <name>/<name><define part>|<digit><define part>
<name> ::= <ident>|<digit>
<ident> ::= <ident><letter>|<ident><digit>|<letter>|<digit>
An <ident> can not have more than 7 characters.
<define part> ::= <empty>| =<digit>

Explanation

The basic form of the file name is <ident>/<ident> (e.g. SMITH/FILE01). The additional complexity comes from being able to associate file names with one digit numbers (0 thru 9). Thus SMITH/FILE=1 associates the digit 1 with the file SMITH/FILE. The file JONES/1 is JONES/FILE and the file 1/NEW is SMITH/NEW.

In order to find out what files are associated with each digit, the command .TV (type value) is available on all levels (see page 4 and page 12).

Examples:

```
.CR SMITH/FILE44=1←
.CR 1/FILE = 2←
.CR JOE/2 = 3←
.CR 3/1 = 4←
.LD 4 = 5←
.PR 5←
.TVFL←
0: 0000000/0000000
1: SMITH   /FILE44
2: SMITH   /FILE
3: JOE     /FILE
4: JOE     /FILE44
5: JOE     /FILE44
6: 0000000/0000000
7: 0000000/0000000
8: 0000000/0000000
9: 0000000/0000000
```

1 is associated with SMITH/FILE44
2 is associated with SMITH/FILE
3 is associated with JOE/FILE
4 is associated with JOE/FILE44
5 is associated with JOE/FILE44
print JOE/FILE44
type out all files which are defined

Note: the above commands do not show all of the action which would really occur. The order of definition of files is the point of importance.

Note: blanks are ignored everywhere except when getting a file name. A blank or non-alphanuneric character is needed after the second <name>. If a file is being associated with a digit, only one digit is picked up. The rest are ignored.

Appendix 2

Input and Output Conversion

Because the teletype keyboard is incapable of transmitting all of the 64 Algol characters, the program must do conversion for input and output with the teletype. For input, the user can either use the special set accepted by the Algol compiler or he can use the following set which this program converts into the desired character.

Input	Converted to	Alternate input (no conversion)
:=	←	none
_EQ_	_=_	=
_NQ_	_≠_	↑ (upper case N)
_GR_	_>_	> (upper case .)
_GQ_	_≥_	none
_LS_	_<_	none
_LQ_	_≤_	' (upper case 7)

The two letter mnemonics must be delimited by at least one blank on each side, and these blanks are preserved on conversion. The "_" stands for a blank. There are three more characters which should be mentioned: the [is upper case K, the] is upper case M, and the ✗ is upper case L and prints as \.

This conversion will normally apply only to input of card images and strings which contain parts of card images. Although card images are limited to 72 columns (a message is typed out and characters after column 72 are lost) and strings are limited to 63 characters, this length applies to the length after conversion and not the length before conversion.

When card images are typed out on the teletype, a similar conversion occurs.

Output	Converted to
=	:=
≠	NQ
>	GR
≥	GQ
<	LS
≤	LQ

Appendix 3

Commands Local to .CR and .LD

If the user desires to enter a command instead of a card image, he types a period (.) in column 1, followed by the command. If more than one command appears on the line, then each is preceded by a period (.). Blanks are ignored entirely, and may be omitted. The commands will be discussed below.

Options

<option> ::= .OP

This command lists all of the options which are available on this sublevel.

Example:

```
.100.OP
NU  NUMBER
OK  CARD OK
ST  STEP
BS  BASIC STEP
NS  NO STEP
TY  TYPE
TU  TYPE UNNUMBERED
TA  TYPE ABSOLUTE
TD  TYPE DURING
TB  TYPE BACK
TV  TYPE VALUE
CH  CHANGE STRING
IN  INSERT STRING
DE  DELETE STRING
SL  SHIFT LEFT
SR  SHIFT RIGHT
FF  FIND FIRST
FA  FIND ALL
IC  INTERCHANGE CARDS
RM  REMOVE CARD
CL  COLUMN (BUILD A CARD)
CP  COPY CARDS
SI  SET INDENTATION
II  INCREASE INDENTATION
DS  DESTROY OLD CARDS
PU  PUNCH UPDATES
WC  WHAT COMUMN
SV  SAVE FILE
OP  OPTIONS TYPED
```

Save File

<save> ::= .SV

This command terminates the .CR or .LD option and returns the program to the main level.

Number

<new number> ::= .NU <number>
<number> ::= <empty>|<unsigned integer>|-<unsigned integer>
<empty> = 0

This command is used to specify the sequence number of the next desired card. If <number> MOD <basic step> ≠ 0 then the increment used to guess at the next sequence number will become the result of the foregoing modulus operation, and it will remain so until it passes the next multiple of the basic step.

Examples:		
	.NU	sequence number is now 0
	.NU 1000	sequence number is now 1000
	.NU-2300	sequence number is now 2300
	.NU 220	if the basic step is 100, then the program will automatically type out 220, followed by 240, 260, 280, 300 and will then resume counting by 100

OK

<ok> ::= .OK <number>
<empty> = 1

The .NU command specifies the absolute sequence number and the .OK command specifies a relative sequence number. The number in the .OK command is multiplied by the current increment and added to the current sequence number.

Examples: Assume that the program is pointing to card 2500 with an increment of 100 before each of the following commands is executed.

.OK	is equivalent to	.NU 2600
.OK 1	is equivalent to	.NU 2600
.OK-1	is equivalent to	.NU 2400
.OK 5	is equivalent to	.NU 3000

Step

<step> ::= .ST <number>
<empty> = 0

This command allows the user to change to number which is used to increment the sequence number. This command in no way affects the basic step, and this new step remains in effect until another .ST or .NU command is given.

Examples:		
	.ST	the increment is set to 0
	.ST 30	the increment is set to 30

Basic Step

<basic step> ::= .BS <number>
<empty> = 100

This command is to be used if the sequence numbering in the program is not incremented by 100 most of the time. If the basic step is not changed, then the program will take longer to find the card images on disk. The program initially assumes that the basic step is 100.

Examples:		
	.BS	the basic step becomes 100
	.BS 1000	the basic step becomes 1000
	.BS 0	the basic step becomes 100 (0 is not allowed)

No Step

<no-step> ::= .NS

This command keeps the sequence number from being incremented as it normally would be after a series of commands. When several commands are put on one line, each command starts at the original sequence number, unless an .OK or .NU command is given. After the line is processed, the next sequence number typed is that of the card just following the last card processed. If .NS is the last command of the line, then the original sequence number is typed. .NS just amounts to a no-op.

Examples: Assume that the program is pointing to card 500, and the succeeding cards are 600, 650, 700 and 750.

.TY 700.TY	type lines 500, 600, 650, 700, and then 500 again. next seq. no.: 600
.TY.TY 700	type lines 500, 500, 600, 650, 700. next seq. no.: 750
.TY.TY700.NS	same as above except next seq. no.: 500

Type

<type> ::= .TY <number> | .TY TO <number>
<empty> = current sequence number

This command types out one or more card images on the teletype. The conversions are discussed in appendix 2. The first eight characters typed are the sequence number, followed by the first 64 characters of the card. If there is any overflow, it is typed on a second line. The first card typed is the one to which the program was pointing before the .TY command. The last card typed is specified in the command. Every card in between is printed, regardless of the current increment.

Examples: Assume that the program is pointing to card 2500

.TY or .TYTO	type card 2500
.TY 2700 .TY TO2700	type card 2500 and every card thereafter until card 2700 is encountered. Card 2700 is the last card typed (if it exists)

Type Unnumbered

<type unnumbered> ::= .TU <number> | .TU TO <number>
<empty> = current sequence number

This command is identical to .TY except that the sequence numbers are not typed and the line is shifted over to the far left. Only the first 72 columns of the card image are printed.

Examples:		
	.NU500.TU	type card 500 without the seq. nos.
	.NU500.TU600	type card 500 thru 600 without the seq. nos.

Type Absolute

<type absolute> ::= .TA <number><to part>
<to part> ::= <empty> | TO <number>
<empty> = 0
TO <empty> = TO the number specified by <number>

This command types out parts of files as specified by the absolute address bounds in the command (0 = first record). This command disregards the sequence numbers and types out only columns 1 thru 72. This command is mainly a system command.

Examples:		
	.TA	type out the first record
	.TA 35	type out the 36th record
	.TA 3 TO 10	type out records 4 thru 11

Type Back

<type back> ::= .TB <switch>
<switch> ::= <empty> | ON | OFF
<empty> = ON

This option allows the user to have every card he changes written back on the teletype. Cards which are removed with the .RM command are typed before they are removed. .TB OFF is the initial state.

Examples:		
	.TB ON .TB	all changes will be typed henceforth until .TB OFF
	.TB OFF	changes will not be typed out

Type During

<type during> ::= .TD

This command is similar to .TB except that it only affects the next command. If the next command doesn't change any cards, then .TD is a no-op. An implied .TB OFF follows .TD.

Examples:		
	.TD.DE"AA"TO3500	all cards on which AA is deleted will be typed

Punch Updates

<punch updates> ::= .PU <switch>
<empty> = ON

This command is like .TB except that each card which is changed goes out to the card punch, thus giving the user a hard copy of his changes. Initially the flag is ON under .LD and OFF under .CR. The card images are saved in a disk file and punched after .SV is typed. Thus if the flag is OFF when .SV is typed, then no cards will be punched even if there are cards in the punch disk file.

Examples:	.PU ON .PU	all changes will be punched
	.PU OFF	changes will not be punched

Destroy Old Cards

<destroy> ::= .DS <switch>
<empty> = ON

This command determines whether cards are protected from destruction by overwriting. Initially the flag is OFF and card images which are type in directly and those which are moved in with the .CP (copy) command can not overwrite a card which already exists. Only the .CL command can overwrite a card. If the destroy flag is turned ON, the overwrite protection is removed.

Examples:	.DS ON .DS	overwriting of cards permitted
	.DS OFF	cards protected from overwriting

Set Indentation

<set indentation> ::= .SI <number>
<empty> = 1

This command specifies the column on which all subsequent cards will begin. Initially the program assumes 1. The indentation also affects the .CL command when no number is given. A .CL: causes the first character of the new card to start in the column specified by the indentation.

Examples:	.SI 1 .SI	all subsequent cards will start in column 1
	.SI 7	all subsequent cards will start in column 7 (until next .SI or .II)

Increase Indentation

<increase indentation> ::= .II <number>
<empty> = 3 (a common Algol paragraph indentation)

This command is similar in purpose to .SI, except that .II is relative to the value of the existing indentation. The number may be positive or negative. A positive number moves the indentation to the right and a negative number moves the indentation to the left.

Examples:

.II 3 .II	all subsequent cards will be indented 3 columns further to the right than the previous cards.
.II-3	all subsequent cards will be indented 3 columns further to the left than the previous cards

Column

<column> ::= .CL <number> : <any collection of characters> ←
<empty> = current indentation (see .SI above)

This command takes the characters between the colon (:) and the left arrow (←) and places them on a card starting at the column specified by <number>. The rest of the card is filled with blanks. If any characters are past column 72, they are lost and an error message is typed. This command must always be the last command of any line.

Examples: Assume .NU2500.TB.SI3 precedes the commands

.CL:START AT 3←	2500 START AT 3
.CL 1:START AT 1←	2500START AT 1

Type Value

<general type value> ::= .TV <value list>
<value list> ::= <value list><value element>|<value element>
<value element> ::= FL | F<digit> | ST | BS | SI | TB | PU | DS

This command types out the files which have been defined by one digit numbers (see appendix 1) and the values of various constants. FL lists all files 0 thru 9 and F<digit>lists the file specified by the digit. The remaining 6 mnemonics give the value of the constant set by the respective command.

Examples:

.TVF3PUSIBSDS←	input
3: SMITH /SAMPLE	output
PU IS ON	
SI = 1	
BS = 100	
DS IS OFF	

Change

```
<change> ::=  .CH <string> TO <string><on-to part>
<string> ::=  "<any collection of 0 to 63 characters> "
              the quote (") may be put into the string by using
              two quotes (""), and it is counted as one character
<on-to part> ::= <empty> | ON <number> | TO <number>
<empty> =  ON 1
ON <empty> =  ON 1
TO <empty> =  TO  current sequence number
```

This command edits one or more lines of a file. The first string is a string of characters which exists in the file and which is to be changed to the second string. The <on-to part> specifies how this operation is to be done. The ON construct specifies which occurrence of the string is to be changed. (ON 1 (ON the first occurrence) is substituted for <empty>). The TO construct specifies that all occurrences of the string from the first card thru the last card are to be changed to the second string. The ON and TO constructs are mutually exclusive and may not be used together in the same command.

Examples: Assume .NU2500.TB and the following card images before each command begins:

```
2500  AA A AA BB
2600  CC A AA
```

	Output:
.CH "AA" TO "****"	2500 **** A AA BB
.CH "AA" TO "****" ON1	2500 **** A AA BB
.CH "AA" TO "****" ON2	2500 AA A **** BB
.CH "AA" TO "****" ON3	string not found, card unchanged
.CH "AA" TO "****" TO	2500 **** A **** BB
.CH "AA"TO"****"TO2600	2500 **** A **** BB 2600 CC A ****

Note: The first and second strings need not be the same length, but if the second is longer, then it may attempt to slide characters off the right end of the card. If this happens, an error message is printed, indicating on which cards this occurred. The most recent card image before this error is saved.

Examples of strings:

""	the empty string
""""	ILLEGAL
""""""	the string containing "
"A=""-"""	the string containing A="-"

Insert

<insert> ::= .IN <string><where><string><on-to part>
<where> ::= A | B
see Change for empty defaults

This command can be obtained from the change command. It finds the occurrence(s) of the second string as specified by the <on-to part>, and then inserts the first string either after (<where> is A) or before (<where> is B) the second string. The <on-to part> is similar to that in the change command.

Examples: Assume .NU2500.TBON and the following card images before each command begins:

```
2500 AA A AA BB
2600 CC A AA
```

```
.IN "****" A "AA"          2500 AA**** A AA BB
.IN "****" B "AA"          2500 ****AA A AA BB
.IN "****" A "AA" ON2      2500 AA A AA**** BB
.IN "****" B "AA"TO2600    2500 ****AA A ****AA BB
                           2600 CC A ****AA
```

Note: we could accomplish the same thing with change commands. For example, the last example could be rewritten as: .CH "AA" TO "****AA" TO 2600.

Delete

<delete> ::= .DE <string><on-to part>
see Change for empty defaults

This command deletes a string from one or more cards, and it is equivalent to .CH string TO "" <on-to part>.

Examples: Assume the same as in the Insert example

```
.DE "AA"                   2500  A AA BB
.DE "AA" ON2               2500 AA A  BB
.DE "AA" TO 2600           2500  A  BB
                           2600 CC A
.DE " " TO                 2500AAAAABB
```

Note: the last example would be expected to produce an infinite loop according to the definitions given, but the program doesn't let it run away.

What Column

<what column> ::= .WC

This command types out the column of the first nonblank character on the current card.

Examples: Assume the following card images:

```
500START:
600     BEGIN
.NU500.WC                  Output:
.NU600.WC                  500COL:  1
                           600COL:  6
```

Shift Left

<shift left> ::= .SL <number><to part>
<empty> = 0
TO <empty> = TO current sequence number

This command shifts the contents of one or more cards left the number of columns specified. The shifting begins on the first card and shifts every card left through the last card specified. If an attempt is made to shift non-blank columns off the card, and error message is typed, indicating the cards on which the error occurred. The absolute value of the shift number is taken before the shift is executed.

Examples: Assume .NU2500.TBON and the following card images before each command begins:

```
2500     ABC
2600  CDEF
.SL 5      }                    2500ABC
.SL 5 TO   }
.SL 2 TO 2600                  {2500   ABC
                               {2600CDEF
```

Shift Right

<shift right> ::= .SR <number><to part>
see Shift Left for empty defaults

This command is identical to the shift left command except that it shifts right.

Examples: Assume .NU2500.TBON and the following card images before each command begins:

```
2500ABC
2600  CDEF
.SR 3      }                   2500   ABC
.SR 3 TO   }
.SR 3 TO 2600                  {2500   ABC
                               {2600     CDEF
```

Interchange

<interchange> ::= .IC <number><to part>
<empty> = current sequence number = no-operation
TO <empty> = TO <sequence number specified by number>.

This command interchanges a specified number of cards. The numbers typed give one set of lower and upper bounds, and the sequence number to which the program is pointing gives the other lower bound. The other upper bound is found during the interchange operation.

Examples: Assume .NU2500.TBON and the following card images:

```
2500AAA                        3500XXX
2600BBB                        3600YYY
```

```
                                    Output:
.IC 3500     }                     (2500XXX
.IC 3500 TO  }                     (3500AAA
.IC 3500 TO 3600                   (2500XXX
                                   (3500AAA
                                   (3600BBB
                                   (2600YYY
```

Remove

<remove> ::= .RM <number> | .RM TO <number>
<empty> ::= current sequence number

This command removes one or more card images from the file. The card to which the program is pointing is the first card removed, and the last card removed is the card <number>. The word TO is optional. This command should be used sparingly because it can be very slow. A $VOID card is punched is .PU is ON.

Examples: Assume .NU2500 precedes the following:

```
.RM      }                    removes card 2500 from the file
.RM TO   }
.RM 2700     }                removes all cards from card 2500
.RM TO 2700  }                thru 2700 from the file
```

Copy

<copy> ::= .CP <number><to part>
<empty> = current sequence number
TO <empty> = sequence number specified by <number>

This command copies one or more cards. The numbers typed give the lower and upper bounds for the set of cards to be copied, and the number to which the program is pointing is the location of the first card to be copied. Subsequent cards are copied into cards whose sequence number is obtained by using the current increment.

Examples: Assume .NU3500.ST100.TBON and the following cards:

```
2500AAA
2600BBB
.CP 2500     }                 3500AAA
.CP 2500 TO  }
.CP 2500TO2600                 (3500AAA
                               (3600BBB
.NU2700.CP2600TO2800           (2700BBB
                               (2800BBB
                               (2900BBB
```

Find First

<find first> ::= .FF <string><at part><to part>
<at part> ::= <empty> | AT <number>
<empty> = any column
AT <empty> = AT 1
<empty> = rest of file
TO <empty> = TO current sequence number

This command searches the file for the string specified starting at the current sequence number and continuing either until the string is found or the upper bound sequence number is encountered. If the string is found, the card image is typed out. If the <at part> is empty, the entire card is searched, but if the column is specified by the <at part>, the string must start in that column. The <to part> allows a section of the file to be searched. If the <to part> is empty, the entire remainder of the file is searched until the string is found.

Examples: Assume .NU500 and the following card images:

```
500  ABCD
600    ABCDE
```

	Output:
.FF "AB" } .FF "AB" TO 600 }	500 ABCD
.FF "AB" AT 5	600 ABCDE
.FF "DE"	600 ABCDE

Find All

<find all> ::= .FA <string><at part><to part>
see Find First for empty defaults

This command is identical to .FF except that it doesn't stop after it finds one occurrence of the string. Instead it keeps going until it hits the upper bound specified in the <to part>. It types out the image of every card on which the string occurs.

Examples: Assume the same as for .FF

.FA "AB" TO 600	{ 500 ABCD { 600 ABCDE
.FA "AB" AT 5 TO 600	600 ABCDE
.FA "DE"	600 ABCDE

Warning: the preceding two commands are extremely slow if a large number of card images must be searched. Thus they should be used sparingly unless the user has great patience.

Appendix 4

Error Recovery

The message: UNEXPLAINED ERROR. FILES CLOSED. means one of two things: the user goofed or the author goofed. In any case the user must try to go back to the place where the error occurred. If the error occurred in the .CR or .LD options, it is quite important that they exit thru the .SV command so that the files are cleaned up properly. Should a real snafu occur, the author should be notified.

This system is designed to withstand any number of halt-loads with minimum loss of cards. The only cards which may be lost are those which were just typed in. All other cards should be safe. If a halt-load does occur, the user merely starts the program running again and goes back to the option in which he was working before the halt-load (except that .CR then becomes .LD). If the user was in the .CR or .LD options with the .PU flag ON, then the program will give the last card which made it to the punch file before the halt-load.

Appendix 5

Sample Program

```
?EXECUTE COMPOSE←
5:COMPOSE/COMPOSE=5 BOJ 162# FROM 01/10
TYPE .COMMAND OR .OP FOR OPTIONS
.CR SMITH/FILE=1.PR1←
SMITH  /FILE    CREATED.
TYPE PROGRAM
       OBEGIN←
     100.II.CL:INTEGR I,J,;     I := 34;←
     200.II-3.CL:END.←
     300.NU 100.CH "GR" TO "GER".TD.DE "," ON 2←
     100   INTEGER I,J;     I := 34;
     200.SV←
WAIT FOR PROCESSING OF FILE.
PROGRAM ON DISK NOW.
PRINTING INITIATED ON SMITH  /FILE    .
.QT←
  COMPOSE/COMPOSE=5 EOJ 1630
```