LLO Users' Guide

Augmentation Research Center

6 NOV 74

Stanford Research Institute
333 Ravenswood Avenue
Menlo Park, California 94025

TABLE OF CONTENTS

INTRODUCTION 2

NLS provides a variety of commands for file manipulation and viewing. Editing commands allow the user to insert and change the text in a file. Viewing commands (viewspecs) allow the user to control how the system prints or displays the file. Line truncation and control of statement numbers are examples of these viewing facilities. 2a

Occasionally one may need more sophisticated view controls than those available with the viewspec and viewchange features in NLS. 2b

For example, one may want to see only those statements that contain a particular word or phrase. 2b1

Or one might want to see one line of text that compacts the information found in several longer statements. 2b2

One might also wish to perform a series of routine editing operations without specifying each of the NLS commands over and over again. 2c

User-written programs may tailor the presentation of the information in a file to particular needs. Experienced users may write programs that edit files automatically. 2d

User-written programs currently must be coded in ARC's procedure-oriented programming language, L10. NLS itself is coded in L10. L10 is a high-level language which must be compiled into machine-readable instructions. 2e

This document describes three general types of programs:
--simple filters that control what is poTRAYED ON THE USER's teletype or display (Parts One and Two),
--programs that may modify the statements as they decide whether to print them (Parts Two and Three),
--those that, like commands, are explicitly given control of the job and interact with the user (Part Four). 2f

User programs that control what material is portrayed take effect when NLS presents a sequence of statements in response to a command like Print (or Jump in DNLS). 2f1

In processing such a command, NLS looks at a sequence of statements, examining each statement to see if it satisfies the viewspecs then in force. At this point NLS may pass the statement to a user-written program to see if it satisfies

the requirements specified in that program. If the user program returns a value of TRUE, the (passed) statement is printed and the next statement in the sequence is tested; if FALSE, NLS just goes on to the next statement. 2f1a

While the program is examining the statement to decide whether or not to print it, it may modify the contents of the statement. Such a program can do anything the user can do with NLS commands. 2f2

For more complicated tasks, control may be passed explicitly to the program. In this case, a user program appears as a special-purpose subsystem having (in addition to the supervisor commands) one or more commands. Once such a program is loaded, it can be used just like any of the standard subsystems. (The MESSAGE program is an example.) 2f3

This document describes the L10 programming language used at ARC. 2g

Part One is intended for the general user. 2g1

It is a primer on Content Analyzer Patterns. This does not involve learning the L10 language nor programming. This section can stand alone, and the general (if somewhat experienced) NLS user should find it useful. 2g1a

Part Two is intended for the beginning programmer. 2g2

It presents a hasty overview of L10 programming, with enough tools to write simple programs. This is intended as an introduction for the beginning L10 programmer, who we assume is reasonably familiar with NLS (its commands, subsystems, and capabilities) and has some aptitude for programming. 2g2a

Parts Three and Four are not included in this document. They are presently being updated. You can read these sections online by jumping to the link <userguides,L10-Guide,>. When completed: 2h

Part Three will include a more complete presentation of L10. 2h1

It is intended to acquaint a potential L10 programmer with enough of the language and NLS environment to satisfy most requirements for automated editing programs. Many of the concepts in Part Two are repeated in Part Three so that it may stand alone as an intermediate programmer's reference guide. 2h1a

Part Four will present more advanced L10 tools and an introduction to CML, allowing command syntax specification. 2n2

This should give the programmer the ability to write programs which work across files, which move through files in other than the standard sequential order, and which interact with the user. 2n2a

We suggest that those who are new to L10 begin with Section 1 and read this document one section at a time, pausing between sections to try out the concepts presented by actually writing patterns or programs that put the new ideas to experimental use. Hands-on experience is of at least as much value as this tutorial. If you have problems at any point, you should get help from ARC before proceeding to the next section. 2n3

More complete documentation can be found in (7052,1). For examples of user programs which serve a variety of needs, consult the User Programs Library Table of Contents (programs,-contents,1). For information about commands mentioned, ask for the programming subsystem with the NLS Help command. This document is available online in <userguides,L10-guide,>. 21

PART ONE: Content Analyzer Patterns 3

Section 1: Introduction 3a

Content analysis patterns cannot affect the format of a statement, nor can they edit a file. They can only determine whether a statement should be printed at all. They are, in a sense, a filter through which you may view the file. More complex tasks can be accomplished through programs, as described later in this document. 3a1

The Content Analyzer filter is created by typing in (or selecting from the text in a file) a string of a special form. This string is called the "Content Analyzer Pattern". Each statement is checked against the pattern before it is printed; only statements that are described by the pattern will be printed. 3a2

Some quick examples of Content Analyzer Patterns: 3a3

'(SLD ') will show all statements whose first character is an open parenthesis, then any number of letters or digits, then a close parenthesis. 3a3a

/"blap"/ will show all statements with the string "blap" somewhere in them. 3a3b

SINCE (3-JUN-73 00:00) will show all statements edited since June 3, 1973 3a3c

The next part of this section will describe the elements which make up Content Analyzer Patterns, followed by some examples. The final subject of this section is how to put them to use. 3a4

Section 2: Patterns 3D

Elements of Content Analyzer Patterns 3D1

Content Analyzer Patterns describe certain things the system must check before printing a statement. It may check one or a series of things. The Content Analyzer searches a statement from the beginning, character by character, for described elements. As it encounters each element of the pattern, the Content Analyzer checks the statement for the occurrence of that pattern; if the test fails, the whole statement is failed (unless there was an "or" condition, as described later) and not printed; if the test is passed, an imaginary marker moves on to the next character in the statement, and the next test in the pattern is considered. 3D1A

The pattern may include any sequence of the following elements, the Content Analyzer moves the marker through the statement checking for each element of the Pattern in turn: 3D1B

Literal Strings 3D1C

'c	the given character (e.g. a lower case c)
"string"	the given string (may include non-printing characters, such as spaces)

Character classes 3D1D

CH	any character
L	lowercase or uppercase letter
D	digit
UL	uppercase letter
LL	lowercase letter
ULD	uppercase letter, or digit
LLD	lowercase letter, or digit
LD	lowercase or uppercase letter, or digit
NLD	not a letter nor digit
PT	any printing character
NP	any non-printing character (e.g. space)

Special characters 3D1E

SP	a space
TAB	tab character
CR	a carriage return
LF	line feed character
EOL	TENEX EOL character
ALT	altmode character

Special elements 3D1F

ENDCHR	beginning and end of every statement; can't scan past it

```
TRUE          is true without checking anything
              in statement
ID= id        statement created by user whose
              ident is given
ID# id        statement not created by user whose
              ident is given
BEFORE (d-t)  statement edited before given date and time
SINCE (d-t)   statement edited since given date and time
        e.g.  BEFORE (1 OCT 1974 00:00) ;
```

The date and time must both appear, in the parentheses. It accepts almost any reasonable date and time syntax.

Examples of valid dates:

17-APR-74	17 APRIL 74
APR-17-74	17/5/1974
APR 17 74	5/17/74
APRIL 17, 1974	

Examples of valid times:

1:12:13	1234:56
1234	1:56AM
1:56-EST	1200NOON
16:30 (4:30 PM)	
12:00:00AM (midnight)	
11:59:59AM-EST (late morning)	
12:00:01AM (early morning)	

Scan direction 3D1B

```
<             set scan direction to the left
>             set scan direction to the right
```

The default, re-initialized for each new statement, is scan to the right.

Combining Elements 3D2

These elements may be combined in any order. Spaces within the pattern are ignored (except in literal strings) so they may be used to make reading easier for you. Several operators can modify the elements: 3D2A

NUMBER -- multiple occurrences 3D2C

A number preceding any element other than one of the "Special elements" means that the test will succeed only if it finds exactly that many occurrences of the element. If there aren't that many, the statement will be rejected. Even though there may be more, it will stop after that many and go on to check the next element in the pattern.

3U1 means three upper case letters

$ -- range of occurrences 3D2c

A dollar sign ($) preceding any element other than the "special elements" means "any number of occurrences of". This may include zero occurrences.

```
$'-   means any number of dashes
```

A number in front of the dollar sign sets a lower limit.

```
3$D   means three or more digits
```

A number after the dollar sign sets an upper limit for the search. It will stop after that number and then check for the next element in the pattern, even if it could have found more.

```
$3LD  means from zero to three letters or digits
5$7PT means from 5 to 7 (inclusive) printing
      characters
```

[] -- floating scan 3D2d

To do other than a character by character check, you may enclose an element or series of elements in square brackets []. The Content Analyzer will scan a statement until the element is found. (If the element is not in square brackets, the whole statement fails if the very next character or string fails the test of the next element.) This test will reject the statement if it can't find the element anywhere in the statement. If it succeeds, it will leave the marker for the next test just after the string satisfying the contents of the square brackets.

```
"start"     means check to see if the statement
            begins with the string "start" (or,
            if it is in the middle of a pattern,
            check the next 5 characters to see
            if they are s t a r t).

["start"]   means scan until it finds the
            string s t a r t.
[3D]        means scan until it finds
            three digits.
[ 3D ':]    means scan until it finds three
            digits followed by a colon
```

- -- negation 3D2e

If an element is preceded by a minus sign -, the statement will pass that test if the element does not occur.

-LD means anything other than a letter or digit, such as punctuation, invisibles, etc.

You may put together any number of any of these to form a pattern. 3b2f

e.g. 1$PT /".NL3;" 1$D/ -SP

Logic in Patterns 3b3

More sophisticated patterns can by written by using the logic features of LIO. Generally, an expression is executed left to right. The following operations are done in the given order:

()
/
NOT
AND
OR 3b3a

() 3b3b

Parentheses (and square brackets for floating scans) may be used to group elements. It is good practice to use parentheses liberally.

/ 3b3c

/ means "either or"; the element will be true if either element is true.

(3D I / 4D) means either three digits and a letter or four digits.

Sometimes you may want want the scan to pass your marker over something if it happens to be there (an optional element). "TRUE" is true without testing the statement. If the other tests fail, the imaginary marker is not moved.

(D / TRUE) looks for a digit and passes the imaginary marker over it. If the next character is not a digit it will just go on to the next test element in the pattern without moving the marker. This test always passes.

i.e. It is used to scan past something(s) which may or may not be there.

Since expressions are executed from left to right, it does no good to have TRUE as the first option. (If it is first, the test will immediately pass without trying to scan over any elements.)

NOT 3030

NOT will be TRUE if the element or group of elements enclosed in parentheses following the NOT is false.

NOT LD will pass if the next character is neither a letter nor a digit.

Since the slash is executed first, NOT D / 'h will be true if the next character is NEITHER a digit nor the letter "h". It is the same as NOT (D/'h).

AND 3b3e

AND means both of the two separated groups of elements must be true for the statement to pass.

SINCE (3/6/73 00:00) AND ID#NDM means statements written since March 6, 1973 by someone other than NDM.

OR 3b3f

OR means the test will be true if either of the separated elements is true. It does the same thing as slash, but after "AND" and "NOT" have been executed, allowing greater flexibility.

D AND LLD OR UL means the same as (D AND LLD) OR UL
D AND LLD / UL means the same as D AND (LLD / UL)

While such patterns are correct and succinct, parentheses make for much clearer patterns. Elements within parentheses are taken as a group; the group will be true only if the statement passes all the requirements of the group. It is a good idea to use parentheses whenever there might be any ambiguity.

Section 3: Examples of Content Analyzer Patterns 3C

```
D 2$LD / ["CA"] / ["Content Analyzer"]
```
3C1

This pattern will match any of three types of statements: those beginning with a numerical digit followed by at least two characters which may be either letters or digits, and statements with either the patterns "CA" or "Content Analyzer" anywhere in the statement. 3C1a

Note the use of the square brackets to permit a floating scan -- a search for a pattern anywhere in the statement. Note also the use of the slash for alternatives.

```
BEFORE (25-JAN-72 12:00)
```
3C2

This pattern will match those statements created or modified before noon on 25 January 1972. 3C2a

```
(ID = HGL) OR (ID = NDM)
```
3C3

This pattern will match all statements created or modified by users with the identifiers "HGL" or "NDM". 3C3a

```
[(2L (SP/TRUE) / 2D) D '- 4D]
```
3C4

This pattern will match characters in the form of phone numbers anywhere in a statement. Numbers matched may have an alphabetic exchange followed by an optional space (note the use of the TRUE construction to accomplish this) or a numerical exchange. 3C4a

Examples include DA 6-6200, DA6-6200, and 326-6200.

```
[ENDCHR] < "cba"
```
3C5

This will pass those statements ending with "abc". It will go to the end of the statement, change the scan direction to left, and check for the characters "cba". Note that since you are scanning backwards, to find "abc" you must look for "cba". Since the "cba" is not enclosed in square brackets, it must be the very last characters in the statement. 3C5a

Section 4: Using the Content Analyzer 3d

Content Analyzer Patterns may be entered in two ways: 3d1

1) From the BASE subsystem, use the command: 3d1a

   Set Content (pattern) To PATTERN OK

2) From the PROGRAMS subsystem, use the command: 3d1b

   Compile Content (pattern) PATTERN OK

   OK means "Command Accept", a control-D or, in TNLS (by default) a carriage return.

In either case: 3d2

1) Patterns may be typed in from the keyboard, or 3d2a

2) they may be addressed from a file. 3d2b

   In this case, the pattern will be read from the first character addressed and continue until it finds a semicolon (;) so you must put a semicolon at the end of the pattern (in the file).

Viewspec j must be on (i.e. Content Analyzer off) when entering a pattern. 3d2c

Entering a Content Analyzer Pattern automatically does two things: 3d3

1) compiles a small user program from the characters in the pattern, and 3d3a

2) takes that program and "institutes" it as the current Content Analyzer filter program, deinstituting any previous pattern. 3d3b

   "Instituting" a program means selecting it as the one to take effect when the Content Analyzer is turned on. You may have more than one program compiled but only one instituted.

   When a pattern is deinstituted, it still exists in your program buffer space and may be instituted again at any time with the command in the PROGRAMS subsystem:

      Institute Program PROGRAM-NAME (as) Content (analyzer) OK

The programs may be refered to by number instead of name. They are numbered sequentially, the first entered being number 1.

All the programs you have compiled and the one you have instituted may be listed with the command in the PROGRAMS subsystem:

Show Status (of programs buffer) OK

Programs may build up in your program buffer. To clear the program buffer, use the PROGRAMS subsystem command:

Delete All (programs in buffer) OK

We recommend that you do this before each new pattern, unless you specifically want to preserve previous patterns.

To invoke the Content Analyzer: 3d4

When viewspec i is on, the instituted Content Analyzer program (if any) will check every statement before it is printed (or displayed). 3d4a

If a statement does not pass all of the requirements of the Content Analyzer program, it will not be printed.

In DNLS, if no statements from the top of the screen on pass the Content Analyzer, the word "Empty" will be displayed.

Note: You will not see the normal structure since one statement may pass the Content Analyzer although its source does not. Viewspec m (statement numbers on) will help you determine the position of the statement in the file.

When viewspec k is on, the instituted Content Analyzer search program will check until it finds one statement that passes the requirements of the pattern. Then, the rest of the output (branch, plex, display screen, etc.) will be printed without checking the Content Analyzer. 3d4b

When viewspec j is on, no Content Analyzer searching is done. This is the default state; every statement in the output (branch, plex, display screen, etc.) will be printed. Note that i, j, and k are mutually exclusive. 3d4c

Notes on the use of Content Analyzer filters: 3a5

Some NLS commands are always affected by the current viewspecs (including i,j, or k): 3a54

Output

Jump (in DNLS)

Print (in TNLS)

Most NLS commands ignore the Content Analyzer in their editing. The following BASE subsystem commands offer the option of specifying viewspecs, or "Filters", (which may turn on the Content Analyzer) which apply only for the purpose of that one command and affect what statements the command works on: 3a56

Copy

Delete

Move

Substitute

At this point, it would be wise to practice until you become proficient at Content Analyzer patterns. You might begin by trying to use some of the patterns given in the above examples, and then try writing a few patterns of your own. These patterns are both a useful NLS tool and a basic component of many L10 programs. 3a6

PART TWO: Introduction to L1O Programming 4

Section 1: Content Analyzer Programs 4a

Introduction 4a1

When you specify a Content Analyzer Pattern, the PROGRAMS subsystem constructs a program which looks for the pattern in each statement and only displays the statement if the pattern matching succeeds. You can gain more control and do more things if you build the program yourself. The program will be used just like the simple pattern program and has many of the same limitations. Programs are written in NLS just like any other text file. They then can be converted to executable code by a compiler. This code resides (or is loaded) in your programs buffer space; it can be instituted as the current Content Analyzer filter program like a Content Analyzer Pattern. 4a1a

Program Structure 4a2

If you specify a Content Analyzer Pattern, NLS compiles a small program that looks like this (with the word "pattern" standing for whatever you typed in): 4a2a

```
PROGRAM name

   (name) PROCEDURE;

      IF FIND pattern THEN RETURN(TRUE) ELSE RETURN(FALSE);

      END.

FINISH
```

All L1O programs must begin with a header statement, the word PROGRAM (all caps) followed by the name of the first procedure to be executed (all lower-case). This name is also the name of the program. If the program is being compiled into a file (to be described at the end of this section), the word FILE should be substituted for the word PROGRAM. 4a2 B

```
E.g.  PROGRAM first
      FILE deldir
```

(Note: the Content Analyzer makes up a program name consisting of UP#1xxxxx , where

is a sequential number, the first pattern being number one, and

xxxxx is the first five characters of your pattern.)

The body of a program consists of a series of DECLARATION statements and PROCEDURES (in any order). In the above case, the program consisted of only one small procedure and no declarations. When the program is loaded into your programs buffer space, the declarations reserve space in the system to store information (variables). When the program is used as a Content Analyzer filter program, the first procedure is called for each statement. It may in turn call other procedures and access variables in the program or in the NLS system. 4a2c

e.g. DECLARE x, y, z (described below)
(first) PROCEDURE;
...

The end of the program is delimited by the word "FINISH" (in all upper case). 4a2d

Comments may be enclosed in percent signs (%) anywhere in the program, even in the middle of L10 statements. The L10 compiler will ignore them. 4a2e

Except within literal strings, variable names and special L10 words, spaces are ignored. It is good practice to use them liberally so that your program will be easy to read. Also, NLS file structure is ignored. Structure is, however, very valuable in making the program readable, and it is good practice to use it in close correlation to the program's logical structure. For instance, the programmer usually makes each of the elements of a program (declarations, procedures, and FINISH) separate statements, below the header statement in file structure. This point will be discussed further later. 4a2f

So far, we have file which looks something like: 4a2g

```
PROGRAM name1

    DECLARE ... ;

    DECLARE ... ;
```

```
(name1) PROCEDURE ;

(name2) PROCEDURE ;

FINISH
```

Procedure Structure 4a3

Each procedure must begin with its header statement. This header statement is a name enclosed in parentheses followed by the word PROCEDURE, and terminated by a semicolon. 4a3a

e.g. (name) PROCEDURE ;

The body of the procedure may consist of Local declarations, then L1O statements. An L1O statement is any program instruction, terminated by a semicolon. The body must at some point return control to the procedure that called it. All this will be discussed more later. 4a3b

The procedure must end with the terminal statement: 4a3c

END.

Example: 4a4

PROGRAM compare 4a4a

```
% Content analyzer.  Displays statement if first two
visibles are the same.  %
DECLARE TEXT POINTER pt1, pt2, pt3, pt4;  %reserves
                          space for ("declares") four
                          text pointers named "pt1"
                          through "pt4"%
DECLARE STRING vis1[100], vis2[100];  %reserves 100
                          characters of space for each
                          of two string variables named
                          "vis1" and "vis2",%
(compare) PROCEDURE ;
   IF FIND $NP ↑pt1 1$PT ↑pt2 $NP ↑pt3 1$PT ↑pt4 THEN
                          %set pointers around first
                          two visibles (strings of
                          printing characters)%
      BEGIN               %if it found two visibles%
      *vis1* ← pt1 pt2 ;  %put visibles in strings%
      *vis2* ← pt3 pt4 ;
      IF *vis1* = *vis2* THEN RETURN(TRUE);  %compare
                          contents of strings, return
                          and display the statement
                          if identical%
      END;
   RETURN (FALSE) ;       %otherwise, return and don't
                          display%
   END.
FINISH
```

Declaration Statements 4a5

As you may have guessed from the above example, Content Analyzer programs can deal with variables (like text pointers and strings), while patterns cannot. 4a5a

Text Pointers 4a5c

A text pointer points to a particular location within an NLS statement (or into a string, as described later).

The text pointer points between two characters in a statement. By putting the pointers between characters, a single pointer can be used to mark both the end of one string and the beginning of the string starting with the next character.

Text pointers are declared with the following Declaration statement:

DECLARE TEXT POINTER name ;

Strings 4a5c

String variables hold text. When they are declared, the maximum number of characters is set.

To declare a string:

DECLARE STRING name(num) ;

num is the maximum number of characters allowed for the string.

e.g. DECLARE STRING lstring(100);

declares a string named "lstring" with a maximum length of 100 characters and a current length of 0 characters (it's empty).

You can refer to the contents of a string variable by surrounding the name with asterisks.

e.g. *lstring* is the string stored in the variable named "lstring".

You can put the text between two text pointers in a string variable with the L1O statement:

lstring ← ptr1 ptr2 ;

where ptr1 and ptr2 are the names of previously declared and set text pointers, and lstring is a previously declared string variable.

These variables will retain their value from one statement to the next. Other types of variables and their use will be discussed in detail in Part Three, Section 3. 4a5d

Body of the Procedure 4a6

RETURN Statement 4a6a

No matter what it does, every procedure must return control

to the procedure that called it. The statement which does this is the RETURN statement.

e.g. RETURN;

A RETURN statement may pass values to the procedure that called it. The values must be enclosed in parentheses after the word RETURN.

e.g. RETURN (1,23,47);

A Content Analyzer program must return either a value of TRUE or of FALSE. If it returns the value TRUE (1), the statement will be printed; if it returns FALSE (0), the statement will not be printed.

i.e. RETURN (TRUE); will print the statement
RETURN (FALSE); will not print the statement

The RETURN statement often is at the end of a procedure, but it need not be. For example, in the middle of the procedure you may want to either RETURN or go on depending on the result of a test.

Other than the requirement of a RETURN statement, the body of the procedure is entirely a function of the purpose of the procedure. A few of the many possible statements will be described here; others will be introduced in Part Three of this document. 4a6b

FIND Statement 4a6c

One of the most useful statements for Content Analyzer programs is the FIND statement. The FIND statement specifies a Content Analyzer pattern to be tested against the statement, and text pointers to be manipulated and set, starting from the Current Character Position (that invisible marker refered to in Section 1). If the test succeeds, the character position is moved past the last character read. If the test fails, the character position is left at the position prior to the FIND statement and the values of all text pointers set within the statement will be reset.

FIND pattern ;

The Current Character Position is initialized to BEFORE THE FIRST CHARACTER, and the scan direction is initialized to

left to RIGHT, FOR EACH NEW STATEMENT passed to the Content Analyzer program.

Any simple Content Analyzer pattern (as describe above) is valid in a FIND statement. In addition, the following elements can be incorporated in the pattern:

stringname

the contents of the string variable

↑ptr

store current scan position into the text pointer specified by ptr, the name of a declared text pointer

←NUM ptr

back up the specified text pointer by the specified number (NUM) of characters. If NUM is not specified, 1 will be assumed. Backup is in the direction opposite to the current scan direction.

ptr

Set current character position to this position. ptr is the name of a previcusly set text pointer.

SF(ptr)

The Current Character Position is set to the front of the statement in which the text pointer ptr is set and scan direction is set from left to right.

SE(ptr)

The Current Character Position is set to the end of the statement in which the text pointer ptr is set and scan direction is set from right to left.

BETWEEN ptr ptr (pattern)

Search limited to between positions specified. ptr is a previously set text pointer; the two must be in the same statement or string. Current Character Position is set to first position before the pattern is tested.

e.g. BETWEEN pt1 pt2 (2D [./] *NP)

FINDS may be used as expressions as well as free-standing statements. If used as an expression, for example in IF statements, it has the value TRUE if all pattern elements within it are true and the value FALSE if any one of the elements is false.

e.g. IF FIND pattern THEN ... ;

Complicated example:

```
IF FIND †sf $NP '( $(LD/'-) ') (".  " *str*/ SE(sf) $NP
'. THEN RETURN(TRUE) ELSE RETURN(FALSE);
```

IF Statement 4a6d

IF causes execution of a statement if a tested expression is TRUE. If it is FALSE and the optional ELSE part is present, the statement following the ELSE is executed. Control then passes to the statement immediately following the IF statement.

IF testexp THEN statement ;

IF testexp THEN statement1 ELSE statement2 ;

The statements within the IF statement can be any valid L10 statement, but are not followed by the usual semicolon; the whole IF statement is treated like one statement and followed by the semicolon.

e.g.

IF FIND /5D/ THEN RETURN(FALSE) ELSE RETURN(TRUE) ;

Programming Style: File Structure 4a7

You may remember that the compiler which converts your NLS text to code ignores file structure. This allows you to use structure to make your program text easier to read and understand. Logical use of structure often facilitates the actual programming task as well. Some conventions have developed at ARC in this respect. All of these should seem obvious and logical to you. 4a7a

All declarations and PROCEDURE statements should be one level below the PROGRAM statement.

All local declarations (not yet described) and code should be one level below the PROCEDURE statement.

It is good style, and makes for much easier programming, to list what you want to do as comment statements (in percent signs) at the level below the PROCEDURE statement. Then you can go back and fill in the code that accomplishes the task described in each comment statement. The code should go one level below the comment.

We will later describe how to block a series of statements where one is required. These blocks should go a level below the statement of which they are a part.

File structure should follow the logical structure of the program as closely as possible.

e.g.
```
IF FIND [5D]
    THEN RETURN(TRUE)
    ELSE RETURN(FALSE);
```

Using Content Analyzer Programs 4a6

Once the Content Analyzer program has been written (in an NLS file), there are two steps in using it. First, the program must be "compiled," i.e. translated into machine-readable code; the compiled code is "loaded" into a space reserved for user programs (the user programs buffer)? Secondly, the loaded program must be "instituted" as the current Content Analyzer program. 4a6a

There are two ways to compile and load a program: 4a6b

1) You may compile a program and load it into your programs buffer all in one operation. The program header statement must have the word PROGRAM in it. When the user resets his job or logs off, the compiled code will disappear.

First, enter the Programs subsystem with the command:

Goto Programs OK

Then you may compile the program with the command:

Compile L10 (user program at) SOURCE OK

SOURCE is the address of the PROGRAM statement.

2) You may compile a program into a file and then load it into your buffer as a separate operation. The program can then be loaded from the file into your user programs buffer at any time without recompiling. The header statement must use the word FILE instead of PROGRAM. Use the PROGRAMS subsystem command:

Compile File (at) SOURCE (using) LLO (to file) FILENAME OK

The FILENAME must be the same as the program's name.

The code file is called a REL (RELocatable code) file. Whenever you wish to load the program code into the user programs buffer, use the PROGRAMS subsystem command:

Load REL (file) FILENAME OK

Once a compiled program has been loaded (by either route), it must be instituted. This is done with the PROGRAMS subsystem command: 4a0c

Institute Program PROGRAM-NAME
(as) Content (analyzer program) OK

The named program will be instituted as the current Content Analyzer program, and any previous program will be deinstituted (but will remain in the buffer).

Again, the programs in the buffer are numbered, the first in being number one. You may use the number instead of the program's name as a shorthand for PROGRAM-NAME.

To invoke the Content Analyzer using whatever program is currently instituted, use the viewspec i, j, or k, as described in Part One, Section 4 (3d4). 4a0d

Problems 4a9

Given these few constructs, you should now be able to write a number of useful Content Analyzer programs. Try programming the following: 4a9a

1) Show those statements which have a number somewhere in the first 20 characters.

2) Show those statements where the first visible in the statement is repeated somewhere in the statement.

Sample solutions:

Problem 1

```
PROGRAM number
   DECLARE TEXT POINTER ptr1, ptr2 ;
   (number) PROCEDURE ;
      FIND ↑ptr1 $20CH ↑ptr2 ;
      IF FIND BETWEEN ptr1 ptr2 ( /D/ )
         THEN RETURN(TRUE)
         ELSE RETURN(FALSE);
      END.
   FINISH
```

Problem 2

```
PROGRAM vis
   DECLARE TEXT POINTER ptr1, ptr2 ;
   DECLARE STRING str/500/ ;
   (vis) PROCEDURE ;
      FIND $NP ↑ptr1 1$PT ↑ptr2 ;
      *str* ← ptr1 ptr2 ;
      IF FIND ptr2 /NP *str* NP/
         THEN RETURN(TRUE)
         ELSE RETURN(FALSE);
      END.
   FINISH
```

Section 2: Content Analyzer Programs: Modifying Statements 4b

Introduction 4b1

Content Analyzer programs may edit the statements as well as decide whether or not they are printed. They are very useful where a series of editing operations has to be done time and time again. This section will introduce you to these capabilities. All these constructs will be covered in detail in Part Three. 4b1a

A Content Analyzer program has several limitations. It can manipulate only one file and it can look at statements only in sequential order (as they appear in the file). It cannot back up and re-examine previous statements, nor can it skip ahead to other parts of the file. It cannot interact with the user. Part Four provides the tools to overcome these limitations. 4b1b

String Construction 4b2

Statements and the contents of string variables may be modified by either of the following two statements: 4b2a

```
ST ptr ← strlist ;
```

The whole statement in which the text pointer named "ptr" resides will be replaced by the string list (to be described in a minute).

```
ST ptr ptr ← strlist ;
```

The part of the statement from the first ptr to the second ptr will be replaced by the string list.

ptr may be a previously set text pointer or SF(ptr) or SE(ptr).

String variables may also be modified with the string assignment statement: 4b2b

```
*stringname* ← strlist ;
```

The string list (strlist) may be any series of string designators, seperated by commas. The string designators may be any of the following (other possibilities to be described later): 4b2c

a string constant, e.g. "ABC" or 'w

ptr ptr

the text between two text pointers previously set in either a statement or a string

stringname

a string name in asterisks, refering to the contents of the string

E.g.: 4020

```
ST p1 p2 ← *string* ;
    or
ST p1 ← SF(p1) p1, string, p2 SE(p2);
```

(Note: these have exactly the same meaning.)

Example: 403

PROGRAM delsp 403a

```
% Content analyzer.  Deletes all leading spaces from
statements. %
DECLARE TEXT POINTER pt; %reserves space for
                          ("declares") a text pointer
                          named "pt"%
(delsp) PROCEDURE ;
   IF FIND 1$SP ↑pt THEN %scans over leading spaces,
                          then sets pointer%
      ST pt ← pt SE(pt); %replaces statement with text
                          from pointer to statement end%
   RETURN (FALSE) ;      %return, don't display anything%
   END.
FINISH
```

More Than One Change per Statement 404

Part of a text pointer is a character count. This count stays the same until the text pointer is again set (to some other position), even though the statement has been edited. If, for example, you have the statement 404a

abcdefghijklmnopqrstuvwxyz

and if you have set a pointer between the "d" and the "e", it

Will always point between the fourth and fifth characters in the statement. If you then delete the character "a", your pointer will be between the "e" and the "f", now the fourth and fifth characters. For this reason, you probably want to do a series of edits beginning with the last one in the statement and working backwards through the statement. 4b4b

Controlling Which Statements are Modified 4b5

In TNLS, the Content Analyzer program will be called for commands which construct a printout of the file (Print and Output). The program will run on every statement for which it is called (e.g. every statement in the branch during a Print Branch command) which pass all the other viewspecs. Once you have written, compiled, and instituted a program which does some editing operation, the Print command is the easiest way to run the program on a statement, branch, plex, or group. 4b5a

In DNLS, the system will call the Content Analyzer program whenever the display is recreated (e.g. viewspec f and the Jump commands), and also for the Output commands. If the program returns TRUE, it will only run on enough statements to fill the screen. It is safer to have programs that edit the file return FALSE. Then when you set viewspec i, it will run on all statements from the top of the display on, and when it is done it will display the word "Empty". At that point, change to viewspec j and recreate the display with viewspec f, then all statements including the changes will be displayed. You can control which statements are edited with level viewspecs and the branch only (g) or plex only (l) viewspecs. 4b5b

After having run your program on a file, you may wish to Update to permanently incorporate the changes in the file. It is wise to Update before you run the program so that, if the program does something unexpected, you can Delete Modifications and return to a good file. 4b5c

Problems 4b6

Try writing the following programs: 4b6a

1) Remove any invisibles from the end of each statement.

2) Make the first visible a statement name (surrounded by parentheses) if it is a word (letters and digits).

Sample solutions: 4060

Problem 1

```
PROGRAM endinv
   DECLARE TEXT POINTER ptr ;
   (endinv) PROCEDURE ;
      IF FIND ↑ptr SE(ptr) 1$NP ↑ptr
         THEN ST ptr ← SF(ptr) ptr ;
      RETURN (FALSE) ;
      END.
   FINISH
```

Problem 2

```
PROGRAM makename
   DECLARE TEXT POINTER ptr1, ptr2 ;
   (makename) PROCEDURE ;
      IF FIND $NP ↑ptr1 1$LD ↑ptr2 NP
         THEN ST ptr1 ← '(, ptr1 ptr2, '), ptr2 SE(ptr2);
      RETURN(FALSE)
      END.
   FINISH
```