L10 PROGRAMMING GUIDE

Augmentation Research Center
Stanford Research Institute
Menlo Park, California 94025

CONTENTS

Section 1. INTRODUCTION TO L10

3

Introduction

3a

This document describes a subset of the L10 programming language used at ARC on the PDP10. The language contains some high level features for operations such as string analysis and manipulation which are implemented in the language as calls on library routines. In addition, L10 has basic constructions such as local variables which have been particularly useful. The L10 compiler was written using the compiler-compiler system Tree Meta. 3a1

The subset presented is offered primarily to satisfy the needs of the novice programmer interested in producing user programs for use in the analyzer formatter system of the NLS portrayal generator. 3a1a

The portrayal generator, its NLS relative the sequence generator, and the NLS commands used to compile users' programs and establish them as the filters used by the system are described in Section 7 and 8 below. 3a1b

CONVENTIONS USED IN DESCRIPTION OF L10 3b

The following conventions (syntax) are used in the description of the features of L10. 3b1

If there is more than one alternative allowed in any syntax rule, they are separated by slashes (/). 3b1a

Each alternative consists of a sequence of elements. 3b1b

All elements in the sequence must occur in the specified order. 3b1c

Any element enclosed in square brackets, *[* and *]*, is optional. 3b1d

The elements may be any of the following: 3b1e

the name of a rule; 3b1e1

a call on a basic recognizer which tests the input for one of the following 3b1e2

ID - recognizes a lower case identifier, 3b1e2a

NUM - recognizes a number, 3b1e2b

SR - recognizes a string enclosed in quotes ("), 3b1e2c

SR1 - recognizes a single character preceded by an apostrophe (') 3b1e2d

CHR - recognizes any character; 3b1e2e

a string enclosed in quotes ("); 3b1e3

a single character string indicated by an apostrophe (') followed by the character; 3b1e4

a list of alternatives enclosed in parentheses; 3b1e5

a dollar sign ($) followed by an element, which means an arbitrary number of occurrences (including zero) of the element. 3b1e6

Comments are enclosed in percent signs (%) and may be embedded anywhere in the rule. 3b1f

Rules are terminated by a semicolon (;). 3blg

DEFINITIONS 3c

identifier 3c1

a symbolic name used to identify procedures, executable statements, and variables. (When used to identify executable statements, identifiers are referred to as labels.) In L10 identifiers consist of any number of lowercase letters and/or digits the first of which must be a letter. 3c1a

label 3c2

an executable statement identifier enclosed in parentheses and followed immediately by a colon. 3c2a

variable 3c3

an identifier which represents a quantity whose value was previously defined, is not yet defined, or may change through the course of the program. L10 variables must be explicitly defined in program declaration statements, in procedure argument lists or LOCAL statements, or must be available as NLS globals. 3c3a

indexed variable 3c4

a multi-element variable or array. L10 permits arrays of one dimension only. 3c4a

global 3c5

pertaining to a variable whose address in memory is known and accessible throughout all parts of a program. Global variables may be declared in a program or be NLS globals, which the NLS environment defines and which are valid for any L10 program. Through the compiler's knowledge of the correspondence between the identifier and the memory address (contained in the system symbol table), the contents of the memory cell may be changed by program instructions. 3c5a

local 3c6

pertaining to a variable whose address in memory is known only to a specific portion of a program, i.e., local to a procedure. 3c6a

constant 3c7

a program element whose value remains unchanged through the programming process. A constant may be a number or literal text (string). 3c7a

string 3c8

a variable or constant consisting of any number of characters enclosed in double quotation marks or a single character preceded by a single quotation mark. 3c8a

comments 3c9

information enclosed in percent signs (%) which may appear anywhere in the program and are ignored when the program is compiled and executed. 3c9a

expression 3c10

in general, any variable, constant or combination of these joined by operators. L10 also provides some special expression constructions that are peculiar to L10. An expression always has a value. 3c10a

statement 3c11

the basic unit of L10 procedures. L10 statements may consist of many parts: expressions, L10 reserved words, other statements, etc. Unlike expressions, statements do not necessarily have values. L10 statements may be labeled or unlabeled. 3c11a

execute 3c12

to carry out an instruction or "run" a program. 3c12a

Section 2. PROGRAM STRUCTURE AND PROCEDURES

4

Introduction 4a

The structure of an L10 program is ALGOL like in its block arrangement. The formal syntax equations for the structure of L10 user programs described below are: 4a1

```
program = header $parts "FINISH";                                   4a1a

header = "PROGRAM" ID;                                              4a1b

  Where ID is the identifier of the first procedure to
  be executed.                                                      4a1b1

parts = procedure / declare;                                        4a1c

procedure = '( ID ') "PROCEDURE" ['( arglist ')] ';
body;                                                               4a1d

arglist = ID $(', ID);                                              4a1e

body =                                                              4a1f

  $("LOCAL" locd '; / "REF" idlist ';)
  labeled $('; labeled) "END." ;                                    4a1f1

labeled = ['(ID");"]statement;                                      4a1g

idlist = ID $(',ID);                                                4a1h

declare =  (decl/ext/equ/regdec/record/pgdec/refd) ';;              4a1i

decl = "DECLARE" ["EXTERNAL"]                                       4a1j

  (field / string / tp / stores / items);                           4a1j1

locd =                                                              4a1k

  "STRING" lstr $(', lstr) /
  "TEXT""POINTER" idlist /
  loco $(', loco);                                                  4a1k1

lstr = .ID '[ NUM ']';                                              4a1l
```

```
        NUM gives the maximum length of the local string
        being declared                                          4alll

    loco = .ID ['[ .NUM ']];                                    4alm

        Local declaration of an array of NUM words or a
        simple variable                                         4alml
```

USER PROGRAM STRUCTURE 4b

A user program in the NLS environment consists of various procedures and declarations that are prefaced and followed by statements that define the boundaries of the program's text. These elements of the L10 program, which must be arranged in a definite manner with strict adherence to syntactic punctuation, are: 4b1

The header - 4b1a

a statement consisting of the word "PROGRAM" followed by the ID of a procedure in the program. (Program execution will begin with a call to this procedure.) No punctuation occurs between the header and the program body. 4b1a1

The body - 4b1b

consists of any number of the following in any order: 4b1b1

declaration statements which specify information about the data to be processed by the procedures in the program and cause the data identifiers to be entered into the program's symbol table. 4b1b1a

procedures which specify certain execution tasks. Each procedure must consist of - 4b1b1b

the procedure identifier enclosed in parentheses followed by the word "PROCEDURE" and optionally an argument list containing names of variables that are passed by the calling procedure for referencing within the called procedure. This statement must be terminated by a semicolon. 4b1b1b1

the body of the procedure which may consist of LOCAL, REF, and/or statements which may optionally be labeled. 4b1b1b2

LOCAL is used for declaring data which is to be used only within the current procedure. 4b1b1b3

REF specifies that the named data elments contain references to other data and when used, the referenced data itself will actually be used. 4b1b1b4

The procedure terminal statement which consists of the word "END" followed by a period (.). 4b1b1b5

The program terminal statement which consists of the word "FINISH". 4b1c

Section 3. VARIABLES, OPERATORS, PRIMITIVES AND EXPRESSIONS

5

Introduction 5a

This section contains a discussion of the basic elements of the L10 language which when combined with the L10 reserved word commands discussed in the next section, are the building blocks of the L10 statements and hence of L10 programs. 5a1

VARIABLES 5b

Five types of variables are described in this document: global, local, referenced, unreferenced, and text pointers. 5b1

GLOBAL VARIABLES 5b2

A global variable is represented by an identifier and refers to a cell in memory which is known and accessible throughout the program. Global variables are defined in the program's declaration statements or in the NLS system environment. 5b2a

A global variable may be indexed, i.e., declared as an array. In this case the user must specify the number of elements of the array by following the ID with an expression in square brackets. For example, in a declaration statement sam[10] specifies an array of 10 elements. In an expression however, sam[10] specifies the tenth element of the array sam. 5b2b

LOCAL VARIABLES 5b3

A local variable is represented by an identifier and refers to a cell in memory which is known and accessible only to the procedure in which it appears. Local variables must appear in a procedure argument list or be declared in a prodecure's LOCAL declaration statement. 5b3a

Local variables in the different procedures may have the same identifier without conflict. A global identifier may not be declared as a local identifier and a procedure identifier may be used as neither. In such cases the ID is considered to be multiply defined and an error results. 5b3b

A local variable may be indexed, i.e., declared as an array. In a local array declaration the user must specify the number of elements of the array by following the ID with an expression in square brackets. For example, odd*[6]* specifies an array of 6 elements. 5b3c

REFERENCED VARIABLES 5b4

A variable which represents a pointer to something rather than the thing itself may be passed as an argument to a procedure. If, in the called procedure, one wishes to access the data referenced by the pointer, the pointer identifier may be declared to be a reference using the REF construction. 5b4a

A pointer to a cell in memory may be passed by a calling procedure. A convenient way to access the contents of the cell is to declare the variable to be "referenced" in the procedure through the use of the "REF" construction. 5b4a1

If a variable has been REF'd, within the scope of the reference (usually a procedure in which it occurs, although a variable may be REF'd through an entire file if desired), whenever the variable is used, that which is pointed to will actually be used. 5b4a2

UNREFERENCED VARIABLES 5b5

If it is desired to use again a pointer to a variable which has been REF'd, one may "unref" it by prefacing the relevant ID with an ampersand (&). 5b5a

TEXT POINTERS 5b6

A text pointer is an L10 feature used in string manipulation constructions. It is a multi-word entity which provides information for pointing to particular locations within text whether free standing strings or strings which contain the text for an NLS file statement. A text pointer consists of a string identifier and a character count. A string may be a declared string, literal string, or a string which contains text of an NLS statement or an NLS file 5b6a

The text pointer points between two characters in a statement or string. By putting the pointers between characters a single pointer can be used to mark both the end of one substring and the beginning of the substring starting with the next character thereby simplifying the string manipulation algorithms and the way one thinks about strings. 5b6a1

OPERATORS 5c

Logical operators 5c1

Every numeric value also has a logical value. A numeric value not equal to zero has a logical value of true; a numeric value equal to zero has a logical value of false. 5c1a

Operator	Evaluation	5c1a1
OR	a OR b = true if a = true or b = true = false if a = false and b = false	5c1a2
AND	a AND b = false if a = false or b = false = true if a = true and b = true	5c1a3
NOT	NOT a = false if a = true = true if a = false	5c1a4

Relational Operators 5c2

A relational operator is used in an expression to compare one quantity with another. The expression is evaluated for a logical value. If true, its value is 1; if false, its value is 0. 5c2a

Operator	Meaning	Example	
			5c2a1
=	equal to	4+1 = 3+2 (true, =1)	5c2a2
#	not equal to	6#8 (true, =1)	5c2a3
<	less than	6<8 (true, =1)	5c2a4
<=	less than or equal to	8<=6 (false, =0)	5c2a5
>	greater than	3>8 (false, =0)	5c2a6
>=	greater than or equal to	8>=6 (true, =1)	5c2a7
NOT	may precede any other relational operator	6 NOT > 8 (true, =1)	5c2a8

Interval operators 5c3

The interval operators permit one to check whether the value of a primitive falls in or out of a particular interval. 5c3a

IN intrel 5c3a1

OUT intrel %equivalent to NOT IN% 5c3a2

intrel = ('(/ '[) opexp ', opexp ('] / ')) 5c3a3

The opexps are values separated by operators against which the operand is tested to see whether or not it lies within (or outside of) a particular interval. Each side of the interval may be "open" or "closed". Thus the values which determine the boundaries may be included in the interval (by using a square bracket) or excluded (by using parentheses). 5c3b

Example: 5c3b1

x IN [1,100) 5c3b1a

is the same as 5c3b1b

(x >=1) AND (x < 100) 5c3b1c

Arithmetic operators 5c4

Operator	Meaning	
unary +	positive value	5c4b
unary -	negative value	5c4c
+	addition	5c4d
-	subtraction	5c4e
*	multiplication	5c4f
/	integer division (remainder not saved.)	5c4g
MOD	a MOD b gives the remainder of a / b	5c4h

.V	a .V b = bit pattern which has 1's wherever either an a or b had a 1 and 0 elsewhere.	5c4i
.X	a .X b = bit pattern which has 1's wherever either an a had a 1 and b had a 0, or a had a 0 and b had a 1, and 0 elsewhere.	5c4j
.A	a .A b = bit pattern which has 1's wherever both a and b had 1's, and 0 elsewhere.	5c4k

PRIMITIVES 5d

Primitives are the basic units which are used as the operands of L1O expressions. There are many types of elements that can be used as L1O primitives; each type returns a value which is used in the evaluation of an expression. 5d1

Each of the following is a valid primitive: 5d2

variable - 5d2a

any valid variable identifier 5d2a1

procname args - 5d2b

a procedure call with argument list 5d2b1

variable '← exp - 5d2c

an assignment statement 5d2c1

pointer - 5d2d

a pointer, possibly a text pointer or a reference to any other type of array 5d2d1

literal - 5d2e

a numeric constant or character constant 5d2e1

string = '* stringname '* / .SR; 5d2f

It is possible to compare variable or literal strings. 5d2f1

charclass - 5d2g

provides a simple way to test the common classes of characters; described in detail below 5d2g1

"MIN" '(exp $(', exp) ')
"MAX" '(exp $(', exp) ') 5d2h

Select the minimum or maximum, respectively, of the values of a list of expressions. 5d2h1

"READC" - 5d2i

a character is read from the current character position and in the direction as set by the last scan. This facility is described later in this document under string manipulation. 5d2i1

"CCPOS" - 5d2j

the value of the index of the character to the right of the current character position. This facility is described later in this document under string manipulation. 5d2j1

"FIND" stringstuff - 5d2k

used to test text patterns and load text pointers for use in string construction (see the STRING MANIPULATION section); return the value TRUE or FALSE depending on whether or not the string tests within it succeed. 5d2k1

"POS" posrel - 5d2l

may be used to compare two text pointers 5d2l1

Procedure Calls 5d3

When a procedure call is used as a primitive, the value is that of the leftmost result returned by the procedure. 5d3a

procname args 5d3a1

Where 5d3b

procname = 5d3b1

ID, a procedure identifier 5d3b1a

args = 5d3b2

'([exp $(', exp)] [': var $(', var)] '); 5d3b2a

exp = 5d3b3

any valid L10 expression. A set of expressions separated by commas constitute the argument list for the procedure. 5d3b3a

```
var =                                                    5d3b4
```

any variable. All but the leftmost variables are used to store the results of the procedure. 5d3b4a

The argument list consists of an arbitrary number of expressions separated by commas. It is not necessary for the number of arguments to equal the number of formal parameters for the procedure (although this is generally a good idea). The argument expressions are evaluated in order from left to right. 5d3c

Following the arguments there may be a list of locations for multiple results to be returned. The list of variables for multiple results is separated from the list of argument expressions by a colon. The number of locations for results need not equal the number of results actually returned. If there are more locations than results, then the extra locations get an undefined value. If there are more results than locations, the extra results are simply lost. 5d3d

Example: 5d3d1

If procedure p ends with the statement 5d3d2

```
RETURN (a,b,c)                                           5d3d2a
```

then the statement 5d3d3

```
q ← p(:r,s)                                              5d3d3a
```

results in (q,r,s) ← (a,b,c). 5d3d4

Assignments 5d4

An assignment can be used as a primitive. 5d4a

The form a ← b has the effect of storing b into a and has the value of b as its value. 5d4b

Pointers 5d5

A string or an identifier preceded by a dollar sign ($) represents a pointer to that string or the variable represented by the identifier. 5d5a

```
pointer = '$ (ID / SR)                                   5d5a1
```

Literals 5d6

A literal is a constant which returns a numerical value. A literal may be any of the following: 5d6a

NUM 5d6a1

"TRUE" 5d6a2

"FALSE" 5d6a3

char 5d6a4

There are several ways in which numeric values may be represented. A sequence of digits alone or followed by a D is interpreted as base ten. If followed by a B then it is interpreted as base eight. A scale factor may be given after the B for octal numbers or after a D for decimal numbers. The scale factor is equivalent to adding that many zeros to the original number. 5d6b

Examples: 5d6b1

64 = 100B = 1B2 5d6b1a

144B = 100 = 1D2 5d6b1b

The words TRUE and FALSE are equivalent to the numbers 1 and 0 respectively. 5d6c

Characters may be used as literals as they are represented internally by numeric values. The following are synonyms for commonly used characters: 5d6d

SR1 - any single character preceded by an apostrophe e.g. 'a represents the code for the character a and is equal to 141B. 5d6d1

"ENDCHR" -endcharacter as returned by READC 5d6d2

"SP" -space 5d6d3

"EOL" -Tenex's version of CR LF 5d6d4

"ALT" -Tenex's version of altmode or escape (=33B) 5d6d5

"CR" -carriage return 5d6d6

```
        "LF" -line feed                                          5d6d7
        "TAB" -tab                                               5d6d8
        "BC" -backspace character                                5d6d9
        "BW" -backspace word                                     5d6d10
        "C." -center dot                                         5d6d11
        CA -Command Accept                                       5d6d12
        CD -Command Delete;                                      5d6d13
Character classes                                                5d7
    charclass =                                                  5d7a
        "CH" /
          %any character%                                        5d7a1
        "ULD" /
          %uppercase letter or digit%                            5d7a2
        "LLD" /
          %lowercase letter or digit%                            5d7a3
        "LD" /
          %lowercase or uppercase letter or digit%               5d7a4
        "NLD" /
          %not a letter or digit%                                5d7a5
        "UL" /
          %uppercase letter%                                     5d7a6
        "LL" /
          %lowercase letter%                                     5d7a7
        "L" /
          %lowercase or uppercase letter%                        5d7a8
        "D" /
          %digit%                                                5d7a9
        "PT" /
          %printing character%                                   5d7a10
```

"NP"
%nonprinting character%; 5d7a11

Example: 5d7a12

char = LD 5d7a12a

is true if the variable "char" contains a value which is a letter or a digit. 5d7a12b

MIN and MAX 5d8

Tnese primitives return the lowest/highest value expression in the expression list specified. 5d8a

Example; if a = 3, b = 2, c = 4 at time MIN and mAX called, then MIN(a,b,c) = b (=2) and MAX(a,b,c) = c (=4). 5d8a1

READC 5d9

The primitive READC is a special construction for reading characters from NLS statements or strings. 5d9a

A character is read from the current character position in the scan direction set by the last CCPOS statement or string analysis FIND statement or expression. This feature is explained in detail later in this document, under String Manipulation. 5d9a1

Attempts to read off the end of a string in either direction result in a special "endcharacter" being returned and the character position is not moved. This endcharacter is included in the set of characters for which system mneumonics are provided and may be referenced by the identifier "ENDCHR". 5d9a2

Example: 5d9a3

to sequentially process the characters of a string 5d9a3a

```
CCPOS *str*;
UNTIL (char ← READC) = ENDCHR DO process(char).
```
5d9a3b

(Note: READC may also be used as a statement if it is desired to read and simply discard a character). 5d9a4

CCPOS 5d10

When used as a primitive, CCPOS has as its value the index of the character to the right of the current character position. CCPOS is more commonly used to set the current character position for use in text pattern matching. This is discussed in detail in section 6 (7b) below. 5d10a

Examples: 5d10a1

If str = "glarp", then after CCPOS *str*, the value of CCPOS is 1 and after CCPOS SE(*str*) the value of CCPOS is 6 (one greater than the length of the string). 5d10a1a

To sequentially process the first n characters of a string (assumed to have at least n characters) 5d10a1b

```
CCPOS *str*;
UNTIL CCPOS > n DO process(READC).
```
5d10a1c

Text Pointer Comparisons 5d11

posrel = 5d11a

```
pos ["NOT"] ('= / '# / ">=" / "<=" / '> / '<) pos;
```
5d11a1

This may be used to compare two text pointers. 5d11a2

The pos is a character position pointer (text pointer) in a form discussed in (7b) below. 5d11a2a

If the pointers refer to different statements then all relations between them are false expect "not equal" which is written '# or "NOT" '=. If the pointers refer to the same statement, then the truth of the relation is decided on the basis of their location within the statement with the convention that a pointer closer to the front of the statement is "less than" a pointer closer to the end. 5d11a3

EXPRESSIONS 5e

Introduction 5e1

An expression is any constant, variable, special expression form, or combination of these joined by operators and parentheses as necessary to denote the order in which operations are to be performed. Special L10 expressions are: the FIND expression which is used for string manipulation; the conditional IF and CASE expressions which may be used to give alternative values to expressions depending on tests made in the expressions. Expressions are used where the syntax requires a value. While certain of these forms are similar syntactically to L10 statements, when used as an expression they always have values. 5e1a

ORDER OF OPERATOR EXECUTION-- BINDING PRECEDENCE 5e2

The order of performing individual operations within an equation is determined by the heirarchy of operator execution (or binding precedence) and the use of parentheses. 5e2a

Operations of the same heirarchy are performed from left to right in an expression. Operations in parentheses are performed before operations not in parentheses. 5e2b

The order of execution hierarchy of operators (from highest to lowest) is as follows: 5e2c

unary -, unary + 5e2c1
.A 5e2c2
.V, .X 5e2c3
*, /, MOD 5e2c4
+, - 5e2c5
relational tests (e.g., >=, <=, >, <, =, #, IN, OUT) 5e2c6
NOT relational tests (e.g., NOT >) 5e2c7
NOT 5e2c8
AND 5e2c9
OR 5e2c10

CONDITIONAL EXPRESSIONS 5e3

IF Expressions 5e3a

IF testexp THEN expl ELSE exp2 5e3a1

testexp is tested for its logical value. If testexp is true then expl will be evaluated. If it is false, then exp2 is evaluated. 5e3a2

Therefore, the result of this entire expression is EITHER the result of expl of exp2. 5e3a3

Example: 5e3a3a

```
y ← IF x IN[1,3] THEN x ELSE 4;          5e3a3a1

% if x = 1, 2, or 3  y←x; otherwise y←4%  5e3a3a2
```

CASE Expression 5e3b

This form is similar to the above except that it causes any one of a series of expressions to be evaluated and used as the result of the entire expression. 5e3b1

```
CASE testexp OF $(relist ': exp ';) "ENDCASE" exp
';                                       5e3b1a

relist = RELOP exp $(', RELOP exp);      5e3b1b
```

Where RELOP = any relational operator 5e3b2

In the above, the testexp is evaluated and used with the operator RELOPs and their respective exps in a relist to test for a value of true or false. If true in any instance the companion exp on the right of the colon is executed and taken to be the value of the whole expression. A value of false for a set of relist tests causes the next relist in the CASE expression to be tested against the testexp. If all relists are false, the ENDCASE expression is taken to be the value of the whole expression. 5e3b3

Example: 5e3b3a

```
CASE x1 OF                               5e3b3a1
```

```
                <4: x1+1;                                   5e3b3a1a

                =4; x1+2;                                   5e3b3a1b

                =5: x1;                                     5e3b3a1c

                ENDCASE x1*2;                               5e3b3a1d
```

Value of X1	Value of Expression	
		5e3b3a2
4	6	5e3b3a3
5	5	5e3b3a4
2	3	5e3b3a5
6	12	5e3b3a6

STRING EXPRESSIONS 5e4

L10 also provides several expression forms which are used for string manipulation and evaluation. These are identical to the string manipulation statements discussed in Section 6 of this document (7). Note that when using string manipulation statement forms as expressions, parentheses may be necessary to prevent ambiguities. 5e4a

Section 4. DECLARATIONS

6

Introduction 6a

L10 declarations are necessary to provide information to the compiler about the nature of the data that is to be accessed. Declarations are non-executable. 6a1

There are various types of declarations available; only the most frequently used are discussed here: DECLARE, REF, and LOCAL. 6a2

Program level declarations (DECLARE amd REF) may appear anywhere in the program. However, procedure level declarations (LOCAL and REF inside a procedure) must appear before any executable statements in the procedure. 6a3

GLOBAL DECLARATIONS 6b

Variables specified in these declarations are global (i.e., outside any procedure) and may be used by all procedures in the program. There are four versions depending on the type of entity to be defined: scalars, arrays, strings, and text pointers. The scalar, array, and string declarations allow the user to initialize the value of the variable(s) specified. 6b1

Declaring Scalar Variables 6b2

A scalar variables that is to be used throughout a program must be declared in a declaration at the program level. The quantity represented by the scalar variable may be a numeric value, a string, or an address. Optionally, the user may specify the initial value of the variable being declared. If a scalar variable is not initialized at the program level, it should be initialized in the first executed procedure in which it appears. 6b2a

To declare a scalar variable only: .Grab=6 6b2a1

"DECLARE" ID '; 6b2a1a

To declare and initialize a scalar variable: 6b2a2

"DECLARE" ID '= CONSTANT '; 6b2a2a

Where ID = the name of the variable being declared. 6b2a3

CONSTANT = 6b2a4

the initial value of ID. It may be any of the following: 6b2a4a

-a numeric constant optionally preceded by a unary minus sign (-) 6b2a4a1

-a string enclosed in quotation marks 6b2a4a2

-another identifier (causing the latter's address to be used as the value of the ID being declared) 6b2a4a3

Examples: 6b2a5

DECLARE x1; %x1 is not initialized% 6b2a5a

DECLARE x2=5; %x2 contains the value 5% 6b2a5b

DECLARE x3="OUT";%x3 contains the word OUT% 6b2a5c

DECLARE xx=x1; %xx contains the address of x1% 6b2a5d

Declaring Array Variables 6b3

If the user intends to use any array variables throughout the program, he must specify the number of elements of the array at the program level. Optionally, he may specify the initial value of each element of the array. If array values are not initialized at the program level, they should be initialized in the first executed procedure in which the array is used. 6b3a

To declare an array variable only: 6b3a1

"DECLARE" ID '[NUM '] '; 6b3a1a

To declare and initialize an array variable: 6b3a2

"DECLARE" ID '=!(CONSTANT $(,CONSTANT) ') '; 6b3a2a

Where ID = the name of the variable being declared. 6b3b

NUM = the number of elements in the array if the array is not being initialized. 6b3c

```
CONSTANT = the initial value of each element of
           the array. The number of constants
           implicitly define the number of elements
           in the array.  They may be any of the
           following:
           -a numeric constant optionally preceded by a
            unary minus (-)
           -a string enclosed in quotation marks
           -another identifier (causing the
            latter's address to be used as the
            value of the ID being declared)                  6b3d

Note: there is a one-to-one correspondence between the
first constant and the first element, the second
constant and the second element, etc.                        6b3e

Examples:                                                    6b3f

   DECLARE sam(10);                                          6b3f1

      %declares an array named sam containing 10
      elements which are not initialized%                    6b3f1a

   DECLARE numbs=(1,2,3);                                    6b3f2

      declares an array named numbs containing 3
      elements which are initialized such that:              6b3f2a

         numbs = 1                                           6b3f2a1

         numbs(1) = 2                                        6b3f2a2

         numbs(2) = 3                                        6b3f2a3

   DECLARE motley=(10,words);                                6b3f3

      declares an  array named motley containing 2
      elements which are initialized such that:              6b3f3a

         motley = 10                                         6b3f3a1

         motley(1) = the address of the variable words       6b3f3a2
```

Declaring Many Scalars and/or Arrays in One Statement 6b4

One may avoid putting several individual declarations of items (i.e., several statements each beginning with the word DECLARE) by putting items and arrays to be declared, initialized or not, in a list in one statement following a single DECLARE separated by commas and terminated by a semi-colon. 6b4a

Example: 6b4a1

```
DECLARE x, y[10], z = (1, 2, -5);
```
6b4a1a

Declaring Strings 6b5

The DECLARE STRING enables the user to declare a global string variable by initializing the string and/or declaring its maximum character length. Any number of strings may be declared in the same statement. 6b5a

To declare a number of strings: 6b5a1

```
"DECLARE STRING" ID '[NUM'] $(',ID'[NUM']) ';
```
6b5a1a

To declare and initialize a number of strings: 6b5a2

```
"DECLARE STRING" ID'=STRING $(',ID'=STRING)';
```
6b5a2a

Where ID = the name of the string being declared 6b5a3

NUM = the maximum number of characters allowed for the string 6b5a4

STRING = a string constant enclosed in double quotation marks. The length of this string defines the maximum length of the corresponding ID. 6b5a5

Strings have two associated values, maximum length and current length. When strings are simply declared, maximum length is specified by NUM and current length is 0; when strings are initialized in a declaration statement, maximum length is equal to current length. 6b5a6

These numbers may be accessed by specifying the name of the string followed by a period and the letters M or L respectively. 6b5a6a

Examples: 6b5a7

```
DECLARE STRING lstring[100];                                    6b5a7a
```

declares a string named lstring with a maximum length of 100 characters and a current length of 0 characters 6b5a7a1

```
DECLARE STRING message="RED ALERT",warn="WARNING",
help[50];                                                       6b5a7b
```

declares three strings message, warn, and help such that: 6b5a7b1

message has an actual and maximum length of 9 characters and contains the text "RED ALERT" 6b5a7b1a

warn has an actual and maximum length of 7 characters and contains the text "WARNING" 6b5a7b1b

help has a actual length of 0 and a maximum length of 50 characters, i.e. help.M = 50 and help.L = 0 6b5a7b1c

Declaring Text Pointers 6b6

The DECLARE TEXT POINTER declaration enables the user to declare global variables as text pointers that are used in string manipulation and construction. 6b6a

```
"DECLARE TEXT POINTER" ID $(',ID) ':                            6b6a1
```

REFERENCE DECLARATIONS 6c

Unlike the other declarations discussed here, the REF statement does not allocate storage; it simply defines the use of the variable(s) specified as references. 6c1

A variable which contains a pointer to something rather than the thing itself may be passed as an argument to a procedure. If, in the called procedure, one wishes to access the thing itself, the pointer identifier may be declared to be a reference using the REF construction. 6c1a

If a variable has been REF'd, within the scope of the reference (usually a procedure in which it occurs, although a variable may be REF'd through an entire file if desired) when the variable is accessed as a normal variable, the value of the cell being pointed to is actually used. 6c1a1

Example: 6c1a1a

If x contains the address of y and x has been REF'd, then: 6c1a1a1

```
z ←x; (=z←Y)                                   6c1a1a1a
x ← z (=y←z)                                   6c1a1a1b
```

This is equivalent (without REF'ing) to: 6c1a1a2

```
z ←[x];                                        6c1a1a2a
[x] ←z;                                        6c1a1a2b
```

Referenced variables may be "unreferenced" by preceding their identifiers by the ampersand character "&". Unreferencing a variable causes it to be interpreted as a pointer. Thus, any variable name may serve a dual function of pointing to an address as well as designating the contents at that address. 6c2

```
"REF" ID $(',ID) ';                            6c2a
```

Local variables may be declared as references by a REF declaration among declarations in a procedure (see below). 6c3

LOCAL DECLARATIONS 6d

The LOCAL declaration consists of several forms that are equivalent to those of the global DECLARE forms except that variables declared in a LOCAL declaration may be used only by the procedure in which they appear. Also, LOCAL declarations do not provide for the initialization of variables. 6d1

Any LOCAL declarations must precede the executable statements in a procedure. 6d2

To declare a local scalar variable only: 6d2a

```
"LOCAL" ID ';
```
6d2a1

To declare a local array variable only: 6d2b

```
"LOCAL" ID '[ NUM '] ';
```
6d2b1

Again lists of items separated by commas may be declared locally. 6d2c

To declare a local string only: 6d2d

```
"LOCAL STRING" ID '[NUM'] $(',ID'[NUM']) ';
```
6d2d1

To declare a local text pointer: 6d2e

```
"LOCAL TEXT POINTER" ID $(',ID) ';
```
6d2e1

Section 5. STATEMENTS

7

ASSIGNMENT 7a

ASSIGN STATEMENT 7a1

In the ASSIGN statement the expression on the right side of the "←" is evaluated and stored in the variable on the left side of the statement. 7a1a

```
var '← exp ';
```
7a1a1

where var = any global, local, referenced or unreferenced variable. 7a1b

MULTIPLEASSIGN STATEMENT 7a2

In the MULTIPLEASSIGN statement the expressions are evaluated and the values pushed on a stack provided by the system. Then the values are popped from the stack and stored into the appropriate left hand side. The order of evaluation of the expressions is left to right. 7a2a

```
'( var $(', var) ') '← '( exp $(', exp) ');
```
7a2a1

where var = any global, local, referenced or unreferenced variable. 7a2b

Naturally, the number of expressions must equal the number of var's. 7a2c

Example: 7a2c1

```
(a, b) ← (a+b, a-b)
```
7a2c1a

the expression a+b is evaluated and stacked, expression a-b is evaluated and stacked, the value of a-b is popped and stored into b, and finally, the value of a+b is popped and stored into a. 7a2c2

DIVIDE STATEMENT 7b

The divide statement permits both the quotient and remainder of a division to be saved. The syntax for the divide statement is as follows: 7b1

"DIV" exp ', quotient ', remainder 7b1a

The central connective in the expression must be '/. Quotient and remainder are the identifiers in which the respective values will be saved upon the division. 7b2

BLOCK 7c

The BLOCK construction enables the user to group several (labeled) statements into one syntactic statement entity. A block construction of any length is valid where a statement is required. 7c1

"BEGIN" $(statement ';) "END" 7c1a

Where statement = any executable L10 statement, labeled or unlabeled. 7c2

Example: 7c2a

```
BEGIN
a←b;
c←d+5;
xx←yy;
(nono):d←a+c;
END
```
7c2a1

is equivalent to: 7c2b

```
a←b;
```
7c2b1

```
c←d+5;
```
7c2b2

```
xx←yy;
```
7c2b3

```
(nono):d←a+c;
```
7c2b4

but may be used in an instance in which the syntax requires one statement. (See, for example, the LOOP constructon below.) 7c2c

CONDITIONAL 7d

There are two types of conditional statements described below-- the common IF statement with optional ELSE and the CASE statement. 7d1

IF Statement 7d2

This form causes execution of a statement (which may be a block) if a tested expression is true. If it is false and the optional ELSE part is present, the statement following the ELSE is executed. If no ELSE part is present, control passes to the statement immediately following the IF statement. 7d2a

```
"IF" testexp "THEN" labeledstatement [("ELSE"
labeledstatement)]
```
7d2a1

testexp is tested for its logical value. If testexp is true then the statement following the THEN will be executed. If it is false and an optional ELSE part is present, then the statement following the ELSE will be executed; otherwise the next statement after the IF statement will be executed. 7d2b

CASE Statement 7d3

This form is similar to the above except that it causes any one of a series of statements to be executed depending on the result of a series of tests. 7d3a

```
CASE testexp OF $( relist ': labeledstat ';)
"ENDCASE" labeledstat ';
```
7d3a1

```
relist = RELOP exp $(', RELOP exp);
```
7d3a2

Where RELOP = any relational operator (>=, <, =, IN, etc.) 7d3b

The CASE-statement provides a means of executing one statement out of many. The expression after the word "CASE" is evaluated and the result left in a register. This is used as the left-hand side of the binary relations at the beginning of the various cases. Several relations may be listed at the start of a single statement; the statement will be executed if any of the relations is satisfied. If none of the relations is satisfied, the statement following the word "ENDCASE" will be executed. 7d3c

Example: 7d3c1

```
CASE c OF
   = a,<d: x ← y;      %Executed if c = a or c < d%
   > b: (x, y) ← (x+y, x-y);   %Executed if c > b%
   ENDCASE v ← x;          %Executed otherwise%      7d3c1a
```

ITERATIVE 7e

The statements described here enable the user to alter the normal sequence of execution within a procedure and/or to cause the repeated execution of a set of statements until some condition is met. 7e1

LOOP STATEMENT 7e2

The statement following the word "LOOP" is repeatedly executed until control leaves by means of some transfer instruction within the loop. 7e2a

"LOOP" statement; 7e2a1

where statement = any executable L1O statement (including a block), labeled or unlabeled. 7e2b

Example: 7e2b1

```
LOOP                                    7e2b1a
   BEGIN                                7e2b1a1
   a ← a * a + 1;                       7e2b1a2
   b ← a + b;                           7e2b1a3
   IF a > 200 THEN EXIT;                7e2b1a4
   END;                                 7e2b1a5
```

It is assumed that a and b have been initialized before entering the loop. The EXIT construction is described below. 7e2b1b

WHILE...DO STATEMENT 7e3

This statement causes a statement (or block of statements) to be repeatedly executed as long as the expression immediately following the word WHILE has a logical value of true or control has not been passed out of the DO loop by some explicit transfer. 7e3a

"WHILE" exp "DO" statement 7e3a1

exp is evaluated and if true the statement following the word DO is executed; exp is then reevaluated and the statement continually executed until exp is false. In this event control will pass to the next sequential statement. 7e3b

Example: 7e3b1

```
WHILE alpha DO                          7e3b1a
    BEGIN                               7e3b1a1
    zygo ← b+b:                         7e3b1a2
    alpha ← alpha-1;                    7e3b1a3
    END:                                7e3b1a4
```

If alpha has a value of +5 (logically true) when this statement is executed, the statement following "DO" will be executed 5 times as alpha is decremented by one each time the statement is executed. Once alpha is equal to zero (false) the next statement will be executed. 7e3b2

UNTIL...DO STATEMENT 7e4

This statement is similar to the WHILE...DO statement except that statement(s) following DO are executed until exp is true. As long as exp has a logical value of false the statement(s) will be executed repeatedly. 7e4a

"UNTIL" exp "DO" statement 7e4a1

DO...UNTIL/WHILE STATEMENT 7e5

This statement is like the preceding statement, except that the logical test is made after the statement has been executed rather than before. 7e5a

"DO" statement ("UNTIL" / "WHILE") exp; 7e5a1

Thus the specified statement is always executed at least once (the first time, before the test is made). 7e5b

FOR STATEMENT 7e6

The FOR statement causes the repeated execution of the statement following "DO" until a specific terminal value is reached. 7e6a

```
"FOR" var ['← exp1] ("UP" / "DOWN") [exp2]
"UNTIL" (relop) exp3 "DO" statement;
```
7e6a1

Where var = the variable whose value in incremented/decremented each time the FOR statement is executed 7e6b

exp1 = an optional initial value for var. If exp1 is not specified, the current value of var is used. 7e6c

exp2 = an optional value by which var will be incremented (if UP specified) or decremented (if DOWN specified). If exp2 is not specified, a value of one will be assumed. 7e6d

relop = any relational operator 7e6e

exp3 = when combined with relop determines whether or not another iteration of the FOR statement will be performed. 7e6f

Note that exp2 and exp3 are recomputed on each iteration. 7e6g

Example: 7e6h

```
FOR k ← n UP j UNTIL > m*3 DO x[k] ← k;
```
7e6h1

is equivalent to 7e6h2

```
k ← n;
GOTO test;
(loop): k ← k + j;
(test): IF k > m*3 THEN GOTO out;
x[k] ← k;
GOTO loop;
(out):
```
7e6h3

TRANSFER 7f

These statements in general cause the unconditional transfer of control from one part of a program to another part. 7f1

PROCEDURE CALL STATEMENT 7f2

This statement is used to direct program control to the procedure specified. 7f2a

```
procname args                                              7f2a1
```

Where procname = ID, a procedure identifier 7f2b

```
args = '( [exp $(',exp)] [': var $(',var)]'):              7f2c
```

exp = any valid L10 expression. The set of expressions separated by commas is the argument list for the procedure. 7f2d

var = any variable. The set of variables is used to store the results of the procedure if there is more than one result. 7f2e

The argument list consists of an arbitrary number of expressions separated by commas. It is recommended (although not necessary) for the number of arguments to equal the number of formal parameters for the procedure. The argument expressions are evaluated in order from left to right. 7f2f

Following the arguments there may be a list of locations for multiple results to be returned. The list of variables for multiple results is separated from the list of argument expressions by a colon. The number of locations for results need not equal the number of results actually returned. If there are more locations than results, then the extra locations get an undefined value. If there are more results than locations, the extra results are simply lost. 7f2g

Example: 7f2g1

If procedure p ends with the statement 7f2g2

```
RETURN (a,b,c)                                             7f2g2a
```

then the statement 7f2g3

```
q ← p(:r,s);
```
7f2g3a

results in (q,r,s) ← (a,b,c). 7f2g4

A procedure call may just exist as a statement alone without returning a value: 7f2g5

```
z();
```
7f2g5a

RETURN STATEMENT 7f3

This statement causes a procedure to return an arbitrary number of results. The order of evaluation of results is from left to right. 7f3a

"RETURN" [' (exp $(', exp) ')] 7f3a1

GOTO STATEMENT 7f4

Goto provides for unconditional transfer of control to a new location. 7f4a

"GO""TO" ID 7f4a1

The ID is the name of a label elsewhere in the program. 7f4b

EXIT STATEMENT 7f5

This construction provides for forward branches out of CASE or iterative statements. The optional number (NUM) specifies the number of lexical levels of CASE or iterative statements respectively that are to be exited. If a number is not given then 1 is assumed. All of the iterative statements (LOOP, WHILE, UNTIL, DO, FOR) can be exited by the EXIT LOOP construct. 7f5a

"EXIT" ("CASE" [NUM] / ["LOOP"] [NUM]) 7f5a1

EXIT and EXIT LOOP have the same meaning. 7f5b

Examples: 7f5b1

```
LOOP
   BEGIN
   ........
   IF test THEN EXIT;
   %the EXIT will branch out of the LOOP%
   ........
   END;
```
7f5b1a

```
UNTIL something DO
   BEGIN
   ........
   WHILE test1 DO
      BEGIN
      ........
      IF test2 THEN EXIT;
      %the EXIT will branch out of the WHILE%
      ........
      END;
   ........
   END;                                                    7f5b1b

UNTIL something DO
   BEGIN
   ........
   WHILE test1 DO
      BEGIN
      ........
      IF test2 THEN EXIT 2;
      %the EXIT 2 will branch out of the UNTIL%
      ........
      END;
   ........
   END;                                                    7f5b1c

CASE exp OF
   =something:
      BEGIN
      ........
      IF test THEN EXIT CASE;
      %the EXIT will branch out of the CASE%
      ........
      END;
   ........                                                7f5b1d
```

REPEAT STATEMENT 7f6

This construction provides for backward branches to the front of CASE or conditional statements. The optional number (NUM) has the same meaning as in the EXIT statement. 7f6a

"REPEAT" ("LOOP" *[NUM]* / *["CASE"] [NUM]* *['(exp ')]*) 7f6a1

If an expression is given with the REPEAT CASE, then it is evaluated and used in place of the expression given at the head of the specified CASE statement. If the expression is not given, then the one at the head of the CASE statement is reevaluated. 7f6b

It is worth noting that the availability of EXIT and REPEAT statements has resulted in clearer programs which are generally without labels and GOTO's. The EXIT and REPEAT replace GOTO's to the start or end of the most common compound forms. By providing implicit labels in these positions for use with EXIT or REPEAT, explicit labels are avoided. 7f6c

REPEAT and REPEAT CASE have the same meaning. 7f6d

Examples: 7f6e

```
CASE exp1 OF
   =something:
      BEGIN
      ........
      IF test1 THEN REPEAT;
      %REPEAT with a reevaluated exp1%
      ........
      IF test2 THEN REPEAT(exp2);
      %REPEAT with exp2%
      ........
      END;
   ........                                          7f6e1

LOOP
   BEGIN
   ........
   IF test THEN REPEAT LOOP;
   %REPEAT LOOP will go to the top of the LOOP%
   ........
   END;                                              7f6e2
```

NULL STATEMENT 7g

The NULL statement may be used as a convenience to the programmer. It is a no-op. 7g1

```
null = "NULL";
```

7g1a

Section 6. STRING TEST AND MANIPULATION 8

INTRODUCTION 8a

The following special statements allow for complex string analysis and construction. The three basic elements of string manipulation discussed here are the Current Character Position (ccpos) and text pointers which allow the user to delimit substrings within a string, patterns that cause the system to search the string for specific occurrences of text and set up pointers to various textual elements, and actual string construction. 8a1

The content analysis facility of NLS may be invoked using similar search patterns without the pointer-loading capabilities. 8a1a

CURRENT CHARACTER POSITION (CCPOS) AND TEXT POINTERS 8b

The Current Character Position is similar to the TNLS CM (current marker) in that it specifies the location in the string at which subsequent operations are to begin. All L10 string tests start their search from the current character position. 8b1

"CCPOS" (pos / '* stringname '* *[* '*[* exp '*]]*); 8b1a

pos is a position in a statement or string that may be expressed as any of the following: 8b2

A previously declared and set text pointer ID 8b2a

The scan direction over the text will remain unchanged. The direction of scanning may be set implicitly using the string front of string end facilities or explicitly using the direction setting "<" or ">" in an earlier pattern. (See "Other parameters" under PATTERNS below.) 8b2a1

String Front -- left of the first character 8b2b

"SF(" stspec ') 8b2b1

When SF is specified scanning will take place from left to right within the string. 8b2b2

"stspec" is a string specification that may be expressed as a previouly declared text pointer ID or previously declared string ID enclosed in asterisks. 8b2b3

String End -- right of the last character 8b2c

"SE(" stspec ') 8b2c1

When SE is specified scanning will take place from right to left within the string. 8b2c2

A text pointer points between two characters in a string. 8b3

The variable holding a text pointer is declared by a DECLARE TEXT POINTER or LOCAL TEXT POINTER statement. There is a special declaration for these because text pointers require more than a single word of storage. The identifier used as a text pointer may be such a variable or a reference, defined by a REF statement, to such a variable. 8b4

If a text pointer is given after CCPOS, then the character position is set to that location. 8b5

If a stringname ('* stringname'*) is given after CCPOS, then the position is moved to that string. The scan direction is set left to right. 8b6

Indexing the stringname (by specifying '[exp ']) simply specifies a particular position within the string. Thus *str*[3] puts the current character position between the second and third characters of the string "str". If the scan direction is left to right, then the third character will be read next. If the direction is right to left, then the second will be read next. 8b6a

If no indexing is given, then the position is set to the left of the first character in the string. This is equivalent to an index of 1. 8b6b

PATTERNS - the FIND statement and CONTENT ANALYSIS patterns 8c

FIND Statements and Expressions 8c1

This statement specifies a string pattern to be tested and text pointers to be manipulated and set starting from the current character position. If the test succeeds the character position is moved past the last character read. If the test fails the character position is reset to the position prior to the test and the values of all text pointers set within the pattern will be reset. 8c1a

"FIND" $strentity; 8c1a1

FINDs may be used as expressions as well as free-standing elements. 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 one of the elements is false. 8c1b

Content Analysis Patterns 8c2

Content analysis patterns are simply string pattern entities followed by a semi-colon. When placed in an NLS file and "compiled" using the Execute Content Analyzer command, the pattern may be invoked using a special viewspec to search through an NLS file for statements satisfying the patterns. (The process is described in detail in sections 7 and 8 below.) 8c2a

Implicit in Content Analysis patterns is the notion that they will start a pattern matching search at the beginning of each NLS text statement. 8c2a1

Certain of the arguments are valid only in the context of complete L10 programs. These are noted below. 8c2a2

Because text pointers may not be loaded in Content Analysis patterns and because strings may not be reconstructed in them, they may only be used effectively in relatively simple cases. In more complex situations, full L10 programs are necessary. 8c2a2a

String pattern entities-- (strentities) 8c3

A string entity (strentity) may be any valid combination of the following: logical operators, testing arguments. and other non-testing parameters which in general cause repositioning within the current string. 8c3a

Logical Operators-- These combine and delimit groups of patterns. Each compound group is considered to be a single pattern with the value TRUE or FALSE. If text pointers are set within a test pattern and the pattern is not true, the values of those text pointers are reset to the values they had before the test was made. (See examples below.) 8c3a1

"OR" - 8c3a1a

Either of the two separated groups must be true for the pattern to be true. 8c3a1a1

"AND" - 8c3a1b

Both of the two separated groups must be true for the pattern to be true. 8c3a1b1

"NOT" - 8c3a1c

The following pattern group must not be true for the pattern to be true. 8c3a1c1

"/" - 8c3a1d

Either of the two separated groups must be true for the pattern to be true. Has lower precedence than OR, i.e., binds less tightly than "OR". 8c3a1d1

Pattern Matching Arguments-- (each of these can be true or false) 8c3a2

These may appear in Content Analysis patterns: 8c3a2a

SR 8c3a2a1

string constant, e.g. "ABC" 8c3a2a1a

It should be noted that if the scan direction is set right to left the pattern string constant pattern should be reversed. In the above example, one would have "CBA". 8c3a2a1a1

char 8c3a2a2

any character 8c3a2a2a

charclass 8c3a2a3

look for a character of a specific class (see Primitives for a list of character classes) If found, = true, otherwise false. 8c3a2a3a

'(strentity ') 8c3a2a4

look for an occurrence of the pattern specified by strentity. If found, = true, otherwise false. 8c3a2a4a

'- parameter 8c3a2a5

True only if the parameter following the dash does not occur. 8c3a2a5a

'[strentity '] 8c3a2a6

true if the pattern specified by strentity can be found anywhere in the remainder of the string. First searches from current position. If the search failed, then the current position is incremented by one and resets. Incrementing and searching continues until the end of the string. The value of the search is false if the testing string entity is not matched before the end of the string is reached. 8c3a2a6a

NUM argument 8c3a2a7

find (exactly) the specified number of occurrences of the argument. 8c3a2a7a

NUM1 '$ NUM2 argument 8c3a2a8

Tests for a range of occurrences of the argument specified. If the argument is found at least NUM1 times and at most NUM2 times, the value of the test is true. 8c3a2a8a

Either number is optional. The default value for NUM1 is zero. The default value for NUM2 is 10000. Thus a construction of the form "$3 CH" would search for any number of characters (including zero) up to and including three. 8c3a2a8a1

"ID" ('#/'=) UID 8c3a2a9

if the string being tested is the text of an NLS statement then the identifier of user who created the statement is tested by this construction. 8c3a2a9a

"SINCE" datim 8c3a2a10

if the string being tested is the text of an NLS statement, this test is true if the statement was created after the date and time (datim, see below) specified. 8c3a2a10a

"BEFORE" datim 8c3a2a11

if the string being tested is the text of an NLS statement, this test is true if the statement was created before the date and time (datim, see below) specified. 8c3a2a11a

These may not appear in Content Analysis patterns: 8c3a2b

'* stringname '* 8c3a2b1

string variable 8c3a2b1a

"BETWEEN" pos pos (strentity ') 8c3a2b2

Search limited to between positions specified. Scan character position is set to first position before the pattern is tested. 8c3a2b2a

Format of date and time for pattern matching 8c3a2c

datim = '(date time ') 8c3a2c1

Acceptable dates and times follow the forms permitted by the TENEX system's IDTIM JSYS described in detail in the JSYS manual. It accepts "most any reasonable date and time syntax." 8c3a2c1a

Examples of valid dates: 8c3a2c1a1

17-APR-70 8c3a2c1a1a
APR-17-70 8c3a2c1a1b
APR 17 70 8c3a2c1a1c
APRIL 17, 1970 8c3a2c1a1d
17 APRIL 70 8c3a2c1a1e
17/5/1970 8c3a2c1a1f
5/17/70 8c3a2c1a1g

Examples of valid times: 8c3a2c1a2

1:12:13 8c3a2c1a2a
1234 8c3a2c1a2b
16:30 (4:30 PM) 8c3a2c1a2c
1234:56 8c3a2c1a2d
1:56AM 8c3a2c1a2e
1:56-EST 8c3a2c1a2f
1200N00N 8c3a2c1a2g
12:00:00AM (midnight) 8c3a2c1a2h
11:59:59AM-EST (late morning) 8c3a2c1a2i
12:00:01AM (early morning) 8c3a2c1a2j

Other Arguments-- (these do not involve tests; ratner, they involve some execution action. They are always TRUE for the purposes of pattern matching tests.) 8c3a3

These may appear in simple Content Analysis Patterns: 8c3a3a

'< - 8c3a3a1

set scan direction to the left 8c3a3a1a

In this case, care should be taken to specify patterns in reverse, that is in the order which the computer will scan the text. 8c3a3a1a1

'> - 8c3a3a2

set scan direction to the right 8c3a3a2a

"TRUE" - 8c3a3a3

has no effect; it is generally used at the end of FIND when a value of true is desired even if all tests fail. 8c3a3a3a

These may not appear in simple Content Analysis Patterns: 8c3a3b

pos - 8c3a3b1

set current character position to this position. If the SE pointer is used, set scan direction from right to left. If the SF pointer is used, set scan direction from left to right. 8c3a3b1a

'↑ ID - 8c3a3b2

store current scan position into the textpointer specified by the identifier 8c3a3b2a

'← [NUM] ID - 8c3a3b3

back up the specified text pointer by the specified number (NUM) of characters. Default value for NUM is one. Backup is in the opposite direction of the current scan direction. 8c3a3b3a

STRING CONSTRUCTION 8d

String constructions allow the replacement of one string (substring) by another string. 8d1

```
("ST" (pos / substr) '← stlist /                                    8d1a

'* stringname '* [ '[ exp "TO" exp' ]] ) '← stlist;                  8d1b
```

The string to which pos or stringname refers is replaced by the string specified to the right of the arrow. A substring is replaced if a substr or an indexed stringname is specified. 8d2

Examples: 8d2a

```
ST p1 p2 ← string;
is equivalent to
ST p1 ← SF(p1) p1, string, p2 SE(p2);                               8d2a1

*str*[lower TO upper] ← string;
is equivalent to
*str* ← *str*[1 TO lower-1], string, *str*[upper+1 TO
str.L];                                                             8d2a2

stlist = stprim $(', stprim);                                       8d3

stprim =                                                            8d4

  "NULL" /                                                          8d4a

    represents the zero length string                               8d4a1

  SR /                                                              8d4b

    for string constant, e.g. "ABC"                                 8d4b1

  substr /                                                          8d4c

    substring                                                       8d4c1

  '+ substr /                                                       8d4d

    substring capitalized                                           8d4d1

  '- substr /                                                       8d4e

    substring in lower case                                         8d4e1
```

```
'$ substr /                                                    8d4f

   If it is preceded by a dollar sign ($), then the
   substring is copied without moving any associated
   markers to the new position.  This element is
   relevant only if the string is the text of an NLS
   statement.                                                  8d4f1

'* stringname '* /                                             8d4g

   for string variables                                        8d4g1

'* stringname '* '[ exp '] /                                   8d4h

   for character variables                                     8d4h1

'* stringname '* '[ exp "TO" exp '] /                          8d4i

   substring by indices                                        8d4i1

      A construction of the form *str*[i TO j] refers to
      the substring starting with the ith character in
      the string up and including the jth character.
      Thus *str*[i TO i+10] is the eleven character
      substring starting with the ith character of str.
      and *str*[i TO str.L] is the string str with the
      first i-1 characters deleted.                            8d4i1a

exp /                                                          8d4j

   value of a general L10 expression taken as a
   character; i.e., the character with the ASCII code
   value equivalent to the value of the expression             8d4j1

"STRING" '( exp [', exp] ');                                   8d4k

   gives a string which represents the value of the
   expression as a signed decimal number.  If the second
   expression is present, a number of that base is
   produced instead of a decimal number.                       8d4k1

substr = pos pos;                                               8d5

  This is the substring bounded by the two positions.          8d5a
```

Example: 8d6

Let a "word" be defined as an arbitrary number of letters and digits. The two statements in this example delete the word pointed to by the text pointer "t", and if there is a space on the right of the word, it is also deleted. Otherwise, if there is space on the left of the word it is deleted. 8d6a

The text pointers x and y are used to delimit the left and right respectively of the string to be deleted. 8d6b

LD is true if the character is a letter or a digit, and SP is true if the character is a space. 8d6c

```
FIND t < $LD ↑x t > $LD (SP ↑y / ↑y x < (SP ↑x / TRUE));
ST x y ← NULL;
```
8d6d

The reader should work through this example until it is clear that it really behaves as advertised. 8d6e

The new string or substring is specified as a concatenation of string primaries, with the primaries separated by commas.

8d7

Section 7. CONTENT ANALYSIS AND SEQUENCE GENERATOR PROGRAMS

9

Introduction 9a

NLS provides a variety of commands for file manipulation and viewing. All of the editing commands, and the print command with associated viewspecs (like line truncation and statement numbers) provide examples of these manipulation and viewing facilities. 9a1

But occasionally one may need more sophisticated view controls than those available with the viewspec and viewchange features in NLS. 9a2

For example, one may want to see only those statements that contain a particular word or phrase. 9a2a

Or one might want to see one line of text that compacts the information found in several longer statements. 9a2b

One might also wish to perform a series of routine editing operations without specifying each of the NLS commands over and over again. 9a3

The Network Information Center at ARC uses the ability to create text using the information from several different statements (and even different files) and the ability to insert this new text into a file to produce catalogues and indices. 9a3a

User written programs enable one to tailor the presentation of the information in a file to his particular needs. Experienced users may write programs that edit files automatically. 9a4

CREATION OF USER WRITTEN PROGRAMS 9b

User written programs must be coded in L10. They may call other user written routines and various procedures in the NLS program itself. 9b1

User programs that control the way material is portrayed take effect when NLS presents a sequence of statements in response to a command like Print Group. 9b2

In processing a command such as Print NLS looks at a sequence of statements, examining each statement to see if it falls within the range specified in the Print command and if it satisfies the viewspecs. At this point NLS may also 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. the next statement in the sequence is tested. 9b2a

User programs that modify files usually gain control at the same point in processing as those that control the view. 9b3

Typically, one wants such a program to operate on a sequence of statements chosen by a user when he decides to run the program. In addition, one usually wants to see the results of such an automated series of editing operations immediately after it happens. 9b3a

Although a user program may be called explicitly (using a special purpose NLS command), it is usually invoked when one asks to view a part of the file. 9b3b

CONTEXT OF USER WRITTEN PROGRAMS -- THE PORTRAYAL GENERATOR 9c

Generally, the user written program runs in the framework of the portrayal generator. It may be invoked in several ways, described below, whenever one asks to view a portion of the file, e.g., with a Print command in TNLS, with any of the output to printer commands, and with the Jump command in DNLS. 9c1

All of the portrayal generators in NLS have at least two sections -- the formatter and the sequence generator; if the user invokes a program of his own, the portrayal generator will have at least one, and possibly two, additional parts -- a user filter program and a user sequence generator. 9c2

FORMATTER 9c3

The formatter section arranges text passed to it by the sequence generator (described below) in the style specified by the user. The formatter observes viewspecs such as line truncation, length and indenting; it also formats the text in accord with the requirements of the output device. 9c3a

The formatter works by calling the sequence generator, formatting the text returned, then repeating this process until the sequence generator decides that the sequence has been exhausted or the formatter has filled the desired area (e.g. the display). 9c3b

SEQUENCE GENERATOR 9c4

The sequence generator looks at statements one at a time, beginning at the point specified by the user. It observes viewspecs like level truncation in determining which statements to pass on to the formatter. 9c4a

For example, the viewspecs may indicate that only the first line of statements in the two highest levels are to be output. The default NLS sequence generator will return pointers only to those statements passing the structural filters; the formatter will further truncate the text to only the first line. 9c4a1

When the sequence generator finds a statement that passes all the viewspec requirements, it returns the statement to the formatter and waits to be called again for the next statement in the sequence. 9c4b

One of the viewspecs that the sequence generator pays particular attention to is "i" -- the viewspec that indicates whether a user filter is to be applied to the statement. If this viewspec is on, the sequence generator passes control to a user filter program, which looks at the statement and decides whether it should be included in the sequence. If the statement passes the filter (i.e. the user program returns a value of true). the sequence generator sends the statement to the formatter; otherwise, it processes the next statement in the sequence and sends it to the user filter program for verification. (The particular user program chosen as a filter is determined by commands described below.) 9c4c

USER FILTERS 9c5

The user filter program may be either a content analysis pattern (compiled and invoked in the manner described below) or an L10 program which may contain what are essentially content analysis patterns as well as text modification elements which may edit the NLS file automatically. 9c5a

CONTENT ANALYSIS PATTERNS 9c5a1

Content analysis patterns describe characteristics that a statement must have to be included in the sequence being generated. For example, a content analysis pattern may stipulate that a statement must contain a particular phrase, or that it must have been written since a particular date. In general, content analysis patterns may use any of the pattern matching facilities permitted in L10 FIND statements. 9c5a1a

Content analysis patterns cannot affect the format of a statement, nor can they initiate editing operations on a file. They can only determine whether a statement should be viewed at all. 9c5a1b

Nevertheless, content analysis filters provide a powerful tool for user control of the portrayal of a series of statements. They are the most frequently used, and easily written, of the user programs. However, if one wishes to change the format of a statement, or to modify the file as it is displayed, he must use a user written L10 program. 9c5a1c

USER WRITTEN L10 PROGRAMS 9c5a2

A user written program may be given control by the sequence generator in exactly the same fashion that a content analysis program is initiated. Writing and using such programs effectively requires a thorough knowledge of NLS (content analysis, in particular) and a modicum of exposure to L10. 9c5a2a

Such a program may change the format of a statement being displayed and it may modify the statement itself (as well as other statements in the file). 9c5a2b

A user written program invoked by the sequence generator has several limitations. It can manipulate only one file and it can look at statements only in the order in which they are presented by the sequence generator. In particular, it cannot back up and re-examine previous statements, nor can it skip ahead to other parts of the file. A user-written sequence generator must be provided when one needs to overcome these restrictions. 9c5a2c

USER-WRITTEN SEQUENCE GENERATORS 9c6

A user may provide his own sequence generator to be used in lieu of the regular NLS sequence generator. (This is controlled by viewspecs O and P.) Such a program may call the normal NLS sequence generator, as well as content analysis filters and user-written L10 programs. It may even call other user-written sequence generators. 9c6a

This technique provides the most powerful means for a user to reformat (and even create) files and to affect their portrayal. However, since writing them requires a detailed knowledge of the entire NLS program, the practice is limited to experienced NLS programmers. 9c6b

Section 8. INVOCATION OF USER FILTERS AND PROGRAMS

10

Introduction. 10a

The user-written filters described in this document may be imposed in some cases through the NLS command "Execute Content Analyzer" and in other cases by an NLS subsystem accessed by the command "Goto Programs". The former method is easier but may be used only with simple Content Analyzer patterns. The latter method requires more of the user; furthermore, the several additional capabilities offered by general user-written programs may be invoked only through the "Goto Programs" submode. 10a1

User sequence generator programs for more complex editing among many files may be written. Additionally, programs may be written in this L10 subset to be used to generate sort keys in the NLS Sort and Merge commands. Descriptions of these more complicated types of user programs and of NLS procedures which may be accessed by such programs is deferred until a later document. In such examples, however, the user would still make use of the commands in the NLS "Goto Programs" subsystem. 10a1a

These TNLS commands are used to compile, institute and execute User Programs and filters. 10a2

Compilation-- 10a2a

is the process by which a set of instructions in a program is translated from a form understandable by humans (e.g., the L10 language) into a form which the computer can use to execute those instructions. 10a2a1

Institution-- 10a2b

is the process by which a compiled program is linked into the NLS running system for execution. 10a2b1

Execution-- 10a2c

is the process in which the computer carries out the instructions contained in a compiled and instituted program. 10a2c1

This section additionally presents, in detail, examples of the use of the L10 programming language to construct user analyzer filters and reformatters. These programs were written by members of ARC who are not experienced programmers. They do not make use of any constructions not explained in this manual. 10a3

SIMPLE CONTENT ANALYSIS PATTERNS 10b

The content analysis feature of NLS permits the user to specify a pattern of text content to be matched by statements in NLS files. Only those statements passed to the filter by the sequence generator satisfying the test will be sent to the formatter for display to the user. A simple content analyzer pattern is compiled by the Execute Content Analyzer command or through the Goto Programs submode, and is activated by a Viewspec parameter. 10b1

The NLS Portrayal Generator, made up of the formatter, the sequence generator, and user filters, is invoked whenever the user requests a new "view" of the file, for example through the use of the TNLS "Print" command or any of the output to printer commands. Thus if one had a user content filter compiled, instituted, and invoked, one could have a printout made (using "Output Quickprint", for example) containing only those statements in the file satisfying the pattern. Section 7 (8c) discusses these concepts in detail. 10b1a

Syntax of Simple Content Analysis Patterns 10b2

A simple content analyzer pattern is made up of any number of String Patterns to be matched terminated by a semi-colon. 10b2a

```
$strentity ';                                    10b2a1
```

It is thus similar to the FIND statement described in Section 6 (7c) of the L10 Primer. It is different because some of the pattern constructions, noted in that section, are neither valid nor relevant out of the context of a complete L10 user program including the constructions which manipulate text pointers. 10b2b

A pattern may be written as text anywhere in an NLS file. A file may thus contain any number of patterns. However, only one pattern may be instituted (or placed as the active program or pattern) at a time although any number of content analysis patterns may be compiled. Using commands in the Programs subsystem, one may switch back and forth between the invocation of any of them. 10b2c

Execute Content Analyzer 10b3

The TNLS command used to compile simple content analysis patterns is: 10b3a

e[xecute] co[ntent analyzer type in?] SP
CA
y[es]
n[o] 10b3a1

(if SP, CA, or y[es]) LIT CA 10b3a1a

(if n[o]) ADDR CA 10b3a1b

In response to the prompt "type in?" the user may respond with SP, CA, or "y" indicating that the pattern will be entered directly from the keyboard. Reponding by "n" indicates that the address of the pattern will be specified. 10b3b

ADDR is a TNLS address specification pointing to the first character in the pattern or non-printing characters immediately preceding the pattern. If the pattern is imbedded in the text of an NLS statement the process will read characters until the first semi-colon is read. 10b3c

If the semi-colon is omitted in this instance, an error will result. 10b3c1

Thus one may make use of parts of complex patterns by positioning the TNLS current position pointer at an appropriate place in the middle of the pattern text. 10b3c2

If a LIT is specified it is taken to be the text of a Content Analysis pattern. (The semi-colon may be omitted here; it will be appended by the system.) 10b3c3

When this command is given the pattern specified is compiled into the user program buffer, a name is assigned and put on the user program name stack, and it is instituted as a content analyzer program. 10b3d

When the CA is typed the message "Compiling User Program" will be put out. If the compilation was successful, the user will be left at the TNLS command specification level. If there were any errors in the compilation a list of the places in the pattern in which the error was discovered followed by the message "*[number]* error(s): Type CA". 10b3e

The description of the errors may be relatively cryptic. Syntax errors deal with some violation of acceptable language form. Compiler and system errors may relate to some more general (and perhaps more obscure) error in the compiler which the ordinary user cannot easily fix. 10b3e1

Remember that the L10 compiler does not do anything about misspelled words and misplaced punctuation marks. 10b3e1a

Content Analysis Via Goto Programs 10b4

Simple Content Analysis patterns may also be compiled using a command of the Programs subsystem described below. 10b4a

Execution and Effect 10b5

When applied to a proper pattern the "Execute Content Analyzer" command, in addition to compiling the user's pattern, institutes it as the current content analyzer filter deinstituting any existing content analyzer pattern program. 10b5a

Most users need not be aware of this fact. 10b5a1

Those, however, who may compile more than one content analyzer pattern in a session may wish to switch between them. 10b5a2

To provide a handle on Content Analyzer patterns they are assigned program names made up of the first 5 characters of the pattern preceded by the letters "UP" (for user program), a number referring to the order of compilation, and an exclamation mark (!). 10b5a3

Using this name one may institute and deinstitute patterns as content analyzer filters by using a command in the Programs subsystem described below. The patterns will appear under these names in the user program stack which may be examined with the Program Status command. 10b5a4

After compilation and institution a content analyzer pattern may be applied as a filter to any NLS file by using certain viewspecs and any command which causes the Portrayal Generator to examine the file, e.g., the TNLS Print commands. Simple content analyzer programs do not modify files. Rather, they just serve as "filters" for the Portrayal Generator (see Section 7 (8c)). Relevant viewspecs are: 10b5b

i-- show only statements with content which passes the filter. For example an Output Quickprint with viewspec i on would print only those statements passing the filter. If none satisfy the filter test, an "Empty" will be displayed on-line, a blank file will be printed by the Quickprint command. 10b5b1

j-- show all content. This is the default viewspec in NLS. The filter is not used in this case. 10b5b2

k-- show the first statement passing the filtej then all others. 10b5b3

Again we emphasize that the files are not modified by simple content analysis filters. L10 user programs must be used for this purpose. 10b5c

Examples of Simple Content Analysis Patterns 10b6

BEFORE (25-JAN-72 12:00); 10b6a

This pattern will match those statements created or modified (whichever happened most recently) before noon on 25 January 1972. 10b6a1

ID = HGL OR ID = MFA; 10b6b

```
    This pattern will match all statements created or
    modified (whichever happened most recently) by users
    with the identifiers "HGL" or "MFA".                    10b6b1

D 2$LD / [("CA" / "Content Analyzer")];                     10b6c

    This pattern will match any of three types of
    statements: those beginning with a numerical digit
    followed by two characters which may be either
    letters or digits, and statements with either the
    patterns "CA" or "Content Analyzer" anywhere in the
    statement.                                              10b6c1

        Note the use of the brackets to permit an
        unanchored search -- a search for a pattern
        anywhere in the statement.  Note also the use of
        the slash for alternations.                         10b6c1a

[(2L (SP/TRUE) /2D) D '- 4D];                               10b6d

    This pattern will match characters in the form of
    phone numbers anywhere in a statement,  Numbers
    matched may have a two digit alphabetic exchange
    followed by an optional space (note the use of the
    TRUE construction to accomplish this) or a numerical
    exchange.                                               10b6d1

        Examples include YU 4-1234, YU4-1234, and
        984-1234.                                           10b6d1a
```

PROGRAMS SUBSYSTEM 10c

Introduction 10c1

This NLS subsystem provides several facilities for the processing of user written programs and filters. It is entered by using the NLS "Goto" <subsystem name> command. This subsystem enables the user to compile L10 user programs as well as Content Analyzer patterns, control how these are arranged internally for different uses, define how programs are used, and interrogate the status of user programs. 10c1a

Programs subsystem commands 10c2

The Goto Programs subsystem is entered by the NLS command: 10c2a

g[oto] p[rograms]... 10c2a1

After the user types the above the system expects one of the following commands: 10c2b

Status of User Programs 10c2c

This sub-command prints out information concerning active user programs and filters which have been compiled and/or instituted. The system may be interrogated about this status with the command: 10c2c1

s[tatus of user programs] CA 10c2c1a

when this command is executed the system will print: 10c2c2

-- the names of all the programs in the stack, including those generated for simple content analysis patterns, starting at the bottom of the stack. This stack contains the symbolic names of all compiled programs and a pointer to the corresponding compiled code. The stack is arranged in order of compilation with the most recently compiled program at the head of the stack. 10c2c2a

-- the remaining free space in the buffer. The buffer contains the compiled code for all the current compiled programs. New compiled code is inserted at the first free location in this buffer. l0c2c2b

-- the current Content Analyser Program or "None" l0c2c2c

-- the current user sequence generator program or "None" l0c2c2d

-- the user key program or "None" l0c2c2e

Content Analyzer l0c2d

This command allows the user to specify a content analysis pattern as a content analyzer filter. l0c2d1

c[ontent analyzer type in?]

SP
CA
y[es]
n[o] l0c2d1a

(if SP, CA, or y[es]) LIT CA l0c2d1a1

(if n[o]) ADDR CA l0c2d1a2

In response to the prompt "type in?" the user may respond with SP, CA, or "y" indicating that the pattern will be entered directly from the keyboard. Reponding by "n" indicates that the address of the pattern will be specified. l0c2d2

ADDR must be the address of the first character or immediately preceding space of the program or pattern. l0c2d3

When this command is executed the pattern specified is compiled into the buffer, its name is put on the stack, and it is instituted as a content analyzer program. l0c2d4

The name assigned is generated in the same manner as those for patterns compiled by the "Execute Content Analyzer" command. l0c2d4a

This command is equivalent to the "Execute Content Analyzer" command in compilation error indications (9b3e) and execution (9b5a). 10c2d5

L1O Compile 10c2e

This command compiles the program specified. 10c2e1

l/lO compile at/ ADDR CA 10c2e1a

ADDR is the address of the first statement of the program. 10c2e2

This command causes the program specified to be compiled into the user program buffer and its name entered into the stack. The program is not instituted. 10c2e3

The name of the program is the visible following the word PROGRAM or FILE in the statement indicated by ADDR. 10c2e3a

Errors are indicated as above for the compilation of simple patterns in (9b3e). 10c2e4

The program may be instituted and executed by the appropriate commands. 10c2e5

Institute Program 10c2f

This command enables the user to designate a program as a content analyzer, sequence generator, or key extractor. 10c2f1

i/nstitute program/ PROGNAME CA /CR/
NUM
/as/ CA /content analyzer/ CA
c/ontent analyzer/ CA
k/ey extractor/ CA
s/equence generator/ CA 10c2f1a

PROGNAME is the name of a program which had been previously compiled with any of the Execute Content Analyzer, Program L1O, or Program Content Analyzer Commands. That is, PROGNAME must be in the stack when this command is executed. 10c2f2

Instead of PROGNAME the user may specify the program to be instituted by NUM, a numeric value indicating the nth program from the bottom of the stack. 10c2f3

The program on the bottom of the stack is the program compiled first. 10c2f3a

Execute Program 10c2g

This command transfers control to the specified program. 10c2g1

e/xecute program/ PROGNAME CA
NUM 10c2g1a

PROGNAME is the name of a program which had been previously compiled. That is, PROGNAME must be in the stack when this command is executed. 10c2g2

Instead of PROGNAME the user may specify the program to be instituted by NUM, a numeric value indicating the nth program in the stack. 10c2g3

Deinstitute Program 10c2h

This command deactivates the indicated program, but does not remove it from the stack and buffer. It may be reinstituted at any time. 10c2h1

d/einstitute program/ PROGNAME CA
NUM 10c2h1a

PROGNAME is the name of a program which had been previously compiled. That is, PROGNAME must be in the stack when this command is executed. 10c2h2

Instead of PROGNAME the user may specify the program to be instituted by NUM, a numeric value indicating the nth program in the stack. 10c2h3

This assumes one program will not be used for more than one purpose at one time. 10c2h3a

Pop Stack 10c2i

The Pop Stack command deletes the top (or most recent) program on the stack. The program is deinstituted, its name removed from the stack, and its space in the buffer marked as free. 10c2i1

p[op stack] CA 10c2i1a

Pop Stack program command (10c2i1) 10c2i2

Reset Stack 10c2j

This command clears all programs from the user program area. All programs are deinstituted, the stack is cleared, and the buffer is marked as empty. 10c2j1

r[eset stack] CA 10c2j1a

Note on Returning from User Analyzer-Formatter Programs 10c3

When a user writes an analyzer-formatter filter program, the main routine must RETURN to the Portrayal Generator. The RETURN must have an argument which is checked by the sequence generator. If the value of that argument is TRUE, the statement will be passed to the formatter to be displayed; if the value is FALSE, it will not be displayed. 10c3a

The user could thus use FIND statements and expressions to check for the presence of statements to be edited by the string construction elements and either display the edited statement or not, thereby saving the formatting time. 10c3b

A file could thus be edited quickly without any immediate feedback to the user with the i viewspec on. However, by turning viewspec j on afterwards, the user could then see the completely edited file. 10c3b1

Examples of Analyzer-Formatter Programs 10c4

The following are examples of user analyzer-formatter programs which selectively edit statements in an NLS file on the basis of text searched for by the pattern matching capabilities. Examples of more sophisticated user programs such as sort keys and user sequence generator programs will be presented in a later supplement with a description of NLS routines easily accessed by users. 10c4a

Example 1-- 10c4b

```
PROGRAM outname % removes statement names -- del= ()
--%                                                    10c4b1
   DECLARE TEXT POINTER sf, paf, pae;                  10c4b1a
   (outname)PROCEDURE;                                 10c4b1b
      IF FIND ↑sf $NP '( ↑paf [')] ↑pae THEN           10c4b1b1
         BEGIN                                         10c4b1b1a
         ST sf ← pae SE(sf);                           10c4b1b1b
         RETURN(TRUE);                                 10c4b1b1c
         END                                           10c4b1b1d
      ELSE RETURN(FALSE);                              10c4b1b2
      END.                                             10c4b1b3
   FINISH                                              10c4b1c
```

This program removes the text and delimiters of NLS statement names from the beginning of the statements. 10c4b2

Example 2-- 10c4c

```
PROGRAM changed;                                          10c4c1
(changed)PROCEDURE;                                       10c4c2
   LOCAL TEXT POINTER f, e;                               10c4c2a
   FIND ↑f SE(f) ↑e;                                      10c4c2b
   IF FIND SINCE (25-JAN-72 12:00) THEN                   10c4c2c
      BEGIN                                               10c4c2c1
      ST f ← "[CHANGED]", f e;                            10c4c2c2
      RETURN(TRUE);                                       10c4c2c3
      END                                                 10c4c2c4
   ELSE RETURN(FALSE);                                    10c4c2d
   END.                                                   10c4c2e
FINISH                                                    10c4c3
```

This program checks to see if a statement was written after a certain date. If it was, the string "[CHANGED]" will be put at the front of the statement. 10c4c4

INDEX

DRC 23-FEB-72 15:33 9248

DRC TELEPHONE/MESSAGE SERVICE

As we move and reorganize the ARC PSO/secretarial functions, we are addressing the problem of contacting staff members when they receive telephone or other messages. Audio and individual radio paging systems are NOT being considered. We plan to have some visual overview of the console area from the telephone call director, and in some cases can direct phone calls to console phones, or personally contact staff. An additional possibility is an array of lights indicating that a message awaits - perhaps blinking if it is considered urgent: Hence the TEST board. 1

No single approach will provide the optimal solution. Comments and suggestions will be appreciated. We want to provide the best communications possible for all concerned - adapted to each individual's desires and needs to the maximum extent feasible. 2

Donald R. Cone
Stanford Research Institute
333 Ravenswood Ave.
Menlo Park, California 94025

To:
Access Copy

9248

DRC 23-FEB-72 15:33 9248

ARC TELEPHONE/MESSAGE SERVICE

(J9248) 23-FEB-72 15:33; Title: Author(s): Donald R. Cone/DRC;
Distribution: Diane S. Kaye, Paul Rech, Michael D. Kudlick, Donald R. Cone, Don Limuti, William R. Ferguson, Priscilla Lister, Robert L. Dendy, Linda L. Lane, Carilyn F. Auerbach, Walter L. Bass, Cary S. Church, William S. Duvall, Douglas C. Engelbart, Beauregard A. Hardeman, Martin E. Hardy, L. D. Hopper, Charles H. Irby, Mil E. Jernigan, Harvey G. Lehtman, John T. Melvin, Leanne B. North, James C. Norton, Cindy Page, Bruce L. Parsley, William H. Paxton, Jeffrey C. Peters, Jake Ratliff, Barbara E. Row, Ed K. Van De Riet, Dirk H. van Nouhuys, Kenneth E. Victor, Don C. Wallace, Richard W. Watson, Don I. Andrews/SRI-ARC;
Sub-Collections: SRI-ARC; Clerk: LLL;
Origin: <LANE>BLANK.NLS;141, 23-FEB-72 14:26 LLL ; ;

Engineering Change A to Contract F30602-70-C-0219
Part One--Technical Proposal 1

I INTRODUCTION 2

This proposal is in response to Rome Air Development Center Request for Change A dated 1 December 1971. 2a

II DISCUSSION OF PROPOSED CHANGE 3

We propose the following changes to the Statement of Work (dated 6 February 1970) for the above contract: 3a

Page 4, add the following paragraphs: 3b

"4.2 The contractor shall provide three IMLAC Programmable Display Systems (PDS-1D) and the IMLAC software necessary to allow remote operations of TNLS over the ARPA network. The systems shall consist of the basic PDS-1D unit with the following features: 3b1

PDS-1D	(Display computer)	3b1a
MEM-1	(Incremental cost of 4k memory module)	3b1b
LVH-1	(Long vector generation hardware)	3b1c
CBS-1	(Read/write cassette including bootstrap ROM)	3b1d
HRC-1	(High contrast, high resolution CRT)	3b1e
TAB-2	(Extended modular support table in lieu of TAB-1)	3b1f
GMI-1	(Graphic mouse with keyset) "	3b1g

In addition, we suggest the addition of: 3b2

BEL-1 (Bell alarm) 3b2a

"4.2.1 In addition, one of the units shall be equipped with the programmer/maintenance control panel (CON-1) feature. The contractor shall include other features as required for interfacing to TNLS, or DNLS, and DEX software or the ARPA network hardware. 3b3

4.2.2 The IMLAC equipment shall include a mouse and binary keyset with each of the basic PDS-1D units. 3b4

4.2.3 The contractor shall procure four Execuport - 310 transceivers and four Termicette cassette digital recorders. These units shall be interfaced in a manner that

makes their operation compatible with the TNLS and DEX software on the contractor's PDP-10 computer." 3b5

The following is a change to Exhibit "A" dated 6 February 1970. 3c

Exhibit "A" add the following: 3c1

"Exhibit Line Item A005 shall be in accordance with the requirements of paragraph 4.2 of the Statement of Work entitled: "Network Information Center and Computer Augmented Team Interaction", dated 6 February 1970 and Engineering Change "A" as outlined in 4.2 of the Statement of Work." 3c1a

Engineering Change A to Contract F30602-70-C-0219
Part Two--Business Proposal 4

I ESTIMATED TIME AND CHARGES 5

It is proposed that the work outlined herein be performed during a period of 4 months. This estimated duration is dependent of the actual delivery of the IMLAC display equipment by IMLAC. They have quoted a delivery time of 90 - 120 days after receipt of the order. 5a

Pursuant to the provisions of ASPR 16-206.2, attached is a cost estimate and support schedule in lieu of the DD Form 633-4. Also enclosed is a signed form complete except as to the "Detail Description of Cost Elements." 5b

II CONTRACT FORM 6

It is requested that any contract resulting from this proposal be awarded on a cost-plus-fixed-fee basis as a modification to contract F 30602-70-C-0219. 6a

III ACCEPTANCE PERIOD 7

This proposal will remain in effect until 1 April 1972. If consideration of the proposal requires a longer period, the Institute will be glad to consider a request for an extension of time. 7a

Cost Estimate:

COST ESTIMATE

Direct Costs *	
Equipment Costs	$ 80,037
Shipping costs	450
Total Direct Costs	80,487
Total Estimated Cost	80,487
Fixed Fee	3,702
Total Estimated Cost Plus Fixed Fee	$ 84,189

* See supporting schedules

Cost schedules:

SCHEDULE A

EQUIPMENT COSTS

IMLAC displays $ 57,877

Item:	Unit price:
1. PDS-1D	$ 9,970
2. MEM-1	3,450
3. LVH-1	2,800
4. CBS-1	845
5. HRC-1	290
6. TAB-2	95
7. GMI-1	2,300
8. CON-1	1,250

Three each of items 1 - 6		$ 52,350
Discount for items 1 - 6		-2,618
Subtotal		49,732
Three each of item 7	6,900	
discount for item 7	-690	6,210
Subtotal		55,942
One each item 8		1,250
Subtotal		57,192
Installation		400
Subtotal		57,592

In addition to features selected by RADC technical personnel, we suggest:

Item:	Unit price:
9. BEL-1 (Bell alarm)	100

Added feature 9:	285
Three each less IMLAC 5% discount	
Total display equipment costs	57,877

As per IMLAC quote dated 3 February 1972 and price list dated 15 January 1972

(assumes all three systems are installed simultaneously at RADC without shipment to an intermediate point for temporary installation and/or modification)

Execuport terminals $ 14,360
Four at $ 3,590
As per Execuport sales brochures dated 5/1/71

Termicette cassette digital recorders	$ 7,800
Termicette model 3100-3	
This model covers 110 to 2400 baud range.	
Four @ $ 1,950	
As per Termicette letter quote to D.F. Mc Namara dated 1/21/72	
Total:	$ 80,037

SCHEDULE B

Shipping	
IMLAC displays	300
Execuport terminals	50
Termicette recorders	100
Total	450

<HJOURNAL>9249.NLS;1, 23-MAR-72 17:34 JCN ;
(J9249) 23-MAR-72 17:34; Title: Author(s): S.R.I. - Augmentation Research Center, James C. Norton/&SRI-ARC JCN; Distribution: Duane L. Stone/DLS; Sub-Collections: SRI-ARC; Clerk: JCN;
Origin: <NORTON>J9249.NLS;2, 6-MAR-72 6:48 JCN ;
.RTJ=0;

Primitive Text Macro Expander

I have written a primitive text macro expcnder which runs as a user program with the sequence generator. 1

With this program, a file may have a number of statements of the form: 2

"DEFINE" name '= string '# $(', name '= string#) '; 2a

name is a legal L10 name (up to 50 characters), 3.e. LL $LLD 2b

String is any string excluding the character '#. 2c

Each DEFINE statement must be in one NLS statement, and anything else in that statement is ignored. 2d

The DEFINE must be the first non-prnting string in the statement. 2e

Any subsequent usage of the word (not string) <name>, will result in the name being replaced by the string. 3

For Example: 3a

DEFINE blap = garple blork#; 3a1

Would result in a statemnt reading: "go blap" being changed to "go garple blork". 3a2

There is a current limit of 100 defines (which is easy to change if necessary), and the total number of characters in all defines (names and strings) is limited to (roughly) 3000. 4

Defines may be nested. The current limit is 8 deep, but this too is easy to change. 5

For Example: 5a

DEFINE blap = garple blork#, snik =go blap away#; 5a1

The statement "Let's snik" would be translated to "Lets go garple blork away". 5a2

In order to use the program, compile the program (starting at statemet 1) in (duvall, macro, 1), and invoke it as the Conan program. 6

When it is executed, it uses the SEND construct, and does not change the file. 7

Any DEFINE statemets are deleted (i.e. not sent). 7a

I am thinking about putting a new command in NLS-utilty such as M-Compile which will do all of this stuff automatically before compiling. 8

In this case, OACRO would probably be compiled within NLS. 8a

All of this is pretty easy to change, and I am open to suggestions. 9

WSD 23-FEB-72 20:13 9254

Primitive Text Macro Expcnder

(J9254) 23-FEB-72 20:13; Title: Author(s): William S. Duvall/WSD; Distribution: Diane S. Kaye, Paul Rech, Michael D. Kudlick, Donald R. Cone, Don Limuti, William R. Ferguson, Priscilla Lister, Robert L. Dendy, Linda L. Lane, Carilyn F. Auerbach, Walter L. Bass, Cary S. Church, William S. Duvall, Douglas C. Engelbart, Beauregard A. Hardeman, Martin E. Hardy, L. D. Hopper, Charles H. Irby, Mil E. Jernigan, Harvey G. Lehtman, John T. Melvin, Leanne B. North, James C. Norton, Cindy Page, Bruce L. Parsley, William H. Paxton, Jeffrey C. Peters, Jake Ratliff, Barbara E. Row, Ed K. Van De Riet, Dirk H. van Nouhuys, Kenneth E. Victor, Don C. Wallace, Richard W. Watson, Don I. Andrews/SRI-ARC; Sub-Collections: SRI-ARC; Clerk: WSD;

JBL 24-FEB-72 7:17 9255

First message

(J9255) 24-FEB-72 7:17; Title: Author(s): Joel B. Levin/JBL:
Fistribution: Lee R Talbert/LRT; Sub-Collections: NIC; Clerk: JBL;

JBL 24-FEB-72 7:17 9255

First message

Hello, Lee, this is your first message on the NIC.

1

William S. Duvall

Redowwd POD NOtes: 22FEB

notes from the Redwood Tree---feb 22 1

Barbara asked why should we continue to have meetings. 2

some discussion on the subject, with reference to the fact that other pods are fading away 2a

I don't think that there were any concrete points for or against. 2b

The discussion ten migrated onto the subject of why don't we cpply the results of augmentation inward. 3

For example, why isn't there a P4 Directive library, or a program Library, or... 3a

Some people are obviously bored 4

Talked about information dssemination and collection within the group (still related to applying augmentation inward) 5

Talked about file privacy, and concept of non-printable files. 6

Some discussion about pros and cons of privacy 6a

List of goals was distributed, and people scanned. 7

Some discussi merits and value of goals versus means. 7a

Agreed to meet next Tues at 1400 7b

Dirk H. van Nouhuys

WSD 24-FEB-72 9:05 9256

Redowwd POD NOtes: 22FEB

(J9256) 24-FEB-72 9:05; Title: Author(s): William S. Duvall/WSD; Distribution: Diane S. Kaye, Paul Rech, Oichael D. Kudlick, Donald R. Cone, Don Limuti, William R. Ferguson, Priscilla Lister, Robert L. Dendy, Linda L. Lane, Oarilyn F. Auerbach, Walter L. Bass, Oary S. Church, William S. Duvall, Douglas C. Engelbart, Beauregard A. Hardeman, Martin E. Hardy, L. D. Hopper, Charles H. Irby, Mil E. Jernigan, Harvey G. Lehtman, John T. Melvin, Leanne B. North, Lames C. Norton, Cindy Page, Bruce L. Parsley, William H. Paxton, Jeffrey C. Peters, Jake Ratliff, Barbara E. RoW, Ed K. Van De Riet, Dirk H. van Nouhuys. Kenneth E. Victor, Donald C. Wallace, Richard W. Watson, Don I. Andrews/SRI-ARC; Sub-Collections: SRI-ARC; Clerk: WSD;

DVN 24-FEB-72 10:16 9262

heading test

(J9262) 24-FEB-72 10:16; Title: Author(s): Dirk H. van Nouhuys/DVN; Distribution: Barbara E. Row/BER(Barbara, when you see this journal item, plese tell me ---Dirk); Sub-Collections: SRI-ARC; Clerk: DVN; Origin: <VANNOUHUYS>HEADTEST.NLS;1, 24-FEB-72 10:13 DVN ;

heading test

ABSTRACT <PBS> <LFH=13> <Center=1> <BRM=60> <BLM= 12> <Ilev=0>
<Text"Section/="Abstract"> 1

During 1970 SRI's Augmentation Research Center took part in preliminary operation of the ARPA network, made several important improvements in the ARC operating system's efficiency and features for users, and began installation of a new computer. 1a

Conversion from an XDS 940 to a DEC PDP-12, which was in process in February 1971, has delayed full operation on the ARPA network. 1a1

However, the network has been used both in software development and in trial runs of the Network Information Center. Initial software for the Network Information Center was completed and documents have been rapidly accumulating. Other new hardware includes UNIVAC drums and various remote terminals. New software includes redesign of the core of our NLS, development of higher level processes such as executable text, and ready use of content analysers in automated clerical procedures. New features for users include, among other things, an online journal comparable both to a daily periodical and to archival journals, and a both to a daily periodical and to archival journals, and a calculator. 1a2

CREDIT <PBS> <LFH=8> <Ilev=3> <Center=1><Text"section/="Credit"> 2

The<BRM=55> research reported here is the product of conceptual, design, and development work by a large number of persons; the program has been active as a coordinated team effort since 1965. 2a

1970's work involved the whole ARC staff: 2b

Walter L Bass, Roger D Bates,
Vernon R Baughman, Mary S Church,
William S Duvall, Douglas C Engelbart
Martin E Hardy J David Hopper,
Charles H Irby, Mildred E Jernigan,
Harvey G Lehtman, John T Melvin,
Jeffrey C Peters, Jeanne B North,
James C Norton, Dirk H van Nouhuys,
Cynthia Page, Bruce L Parsley,
William H Paxton, Jake Ratliff,
Barbara E Row, Edwin K Van De Riet,
and Kenneth E Victor. 2b1

heading test

in addition two consultants: 2c

Don I Andrews and James A Fadiman, 2c1

and the following former members of the staff: <BRM=70> 2d

Geoffrey H Ball, Frederick van den Bosch,
Mary G Caldwell, Roberta A Carillon,
David G Casseres, Ann R Geoffrion,
Jared H Harris, William K English,
Martha E Trundy, and John M Yarborough. 2d1

Kenneth E. Victor

KEV 24-FEB-72 14:37 9265

comments on 110 macros

why not expcnd macro facility to include parametric substitutions? 1

William S. Duvall

KEV 24-FEB-72 14:37 9265

comments on 110 macros

(J9265) 24-FEB-72 14:37; Title: Author(s): Kenneth E. Victor/KEV; Distribution: Diane S. Kaye, Paul Rech, Michael D. Kudlick, Donald R. Cone, Don Limuti, William R. Ferguson, Priscilla Lister, Robert L. Dendy, Linda L. Lane, Carilyn F. Auerbach, Walter L. Bass, Cary S. Church, William S. Duvall, Douglas C. Engelbart, Beauregard A. Hardeman, Martin E. Hardy, L. D. Hopper, Charles H. Irby, Mil E. Jernigan. Harvey G. Lehtman, John T. Melvin, Jeanne B. North, James C. Norton, Cindy Page, Bruce L. Parsley, William H. Paxton, Leffrey C. Peters, Jake Ratliff, Barbara E. Row, Ed K. Van De Riet, Dirk H. van Nouhuys. Kenneth E. Victor, Donald C. Wallace, Richard W. Watson, Don I. Andrews/SRI-ARC; Sub-Collections: SRI-ARC; Clerk: KEV;

Usage of Program Communication Flags 1

Flag #0 (password JLOCK): Used to control Journal access. 1a

When set, prevents anyone new from entering the Journal, but allows persons already using it to continue. 1a1

Flag #1 (Password JBFIL): Indicates a Bad File in the Journal System Files. 1b

This flag may be set either by the Journal, or by slinker. 1b1

It indicates that an error was found in one of the Journal files, and immediately stops any further use of the Journal. 1b2

persons currently using the Journal are bombed out to te TNLS command parser with the message: Global Journal File System Error--Call NIC Center. 1b2a

The flag will always be reset by running recovf, and it will be additionally reset by any successful running of slinker. 1b3

Note that slinker May also set this flag if it finds a bad file. 1b3a

Recovf should be used for recovering. 1b3b

Flag #2 (Password SLNKR): controls the automatic startup of Recovf (slinker, OLJDEL). 1c

If on, PLS will not function as NLS, but will reset it and start up recovf (including logging in as background) instead. 1c1

If found on and NLS is logged in, NLS executes an error after resetting it. 1c2

Flag #3 (Password NLSUT): Controls the automatic startup of NLS utilty. . 1d

If on, NLS will not function as NLS, but will reset it and start up Utilty (including logging in as background) instead. 1d1

If found on and NLS is logged in, NLS executes an error after resetting it. 1d2

Flag #4 (password OPNLK): Used by the routine (IOCTL, openlock) for preventing race conditions. 1e

Flag #5 (password EXPFG): If TRUE, SLINKER and NLSUTILTY will expwnge the directories under which they are running. If FALSE, the expunge will not be executed. 1f

Flag # 6 (Password WMEAS) If on, NLS will write some measurement stuff out on a file when Execute Quit is done. 1g

Flag # 7 (Password IDLOK): A Flag which, if set, will not allow entry into the identification system. 1h

Flag #8: (Password AUTOJ) 1i

Whenever Checkdisc runs, it turns this flag on. 1i1

After it has completed running and there were no errors, it turns it off. 1i2

If this flag is on, no auto-startup jobs will be started. 1i3

WSD 24-FEB-72 15:07 9266

Usage of TENEX Program Communication Flags

(J9266) 24-FEB-72 15:07; Title: Author(s): William S. Duvall/WSD; Fistribution: Diane S. Kaye, Don I. Andrews, Walter L. Bass, William S. Duvall, Oary S. Church, J. D. Hopper, Charles H. Irby, Harvey G. Lehtman, Lohn T. Melvin, Bruce L. Parsley, William H. Paxton, Donald C. Wallace, Kenneth E. Victor, William R. Ferguson, Robert L. Dendy/NPG DCW KEV WRF RLD; Sub-Collections: SRI-ARC NPG; Clerk: WSD;

WSD 24-FEB-72 15:18 9267

On parameters in macros

I didn't allow parameters because Because I wanted to keep it simple enough so as to be implemented in 2-3 hours. I welcome anyone else who wishes to do it. 1

Kenneth E. Victor

WSD 24-FEB-72 15:18 9267

On parameters in macros

(J9267) 24-FEB-72 15:18; Title: Author(s): William S. Duvall/WSD; Distribution: Kenneth E. Victor, Diane S. Kaye, Don I. Andrews, Walter L. Bass, William S. Duvall, Mary S. Church, L. D. Hopper, Charles H. Irby, Harvey G. Lehtman, John T. Melvin, Bruce L. Parsley, William H. Paxton/KEV NPG; Sub-Collections: SRI-ARC NPG; Clerk: WSD;

CHI 24-FEB-72 17:16 9268

New groups in the IDENT system

The following groups have been defined in the IDENT system to facilitate interaction concerning software teams and interest groups. 1

CIP.....Changes in Progress	1a
NLSBUGS.....NLS BUGS Team	1b
MPSDT.....MPS Development Team	1c
MPSIG.....MPS Interest Group	1d
JMAINT.....Journal Maintenance Team	1e
JMIG.....Journal Maintenance Interest Group	1f
JIG.....Journal Interest Group	1g
PNDT.....Primitive NLS Debugger Team	1h
PNDIG.....Primitive NLS Debugger Interest Group	1i
IDMAINT.....IDENT System Maintenance Team	1j
ISMIG.....IDENT System Maintenance Interest Group	1k
IDIG.....IDENT System Interest Group	1l
NCT.....NLS Cleanup Team	1m
NCIG.....NLS Cleanup Interest Group	1n
TLCT.....TENEX Literal Collection Team	1o
TLCIG.....TENEX Literal Collection Interest Group	1p
SDAT..... Sequential Display Area Team	1q
SDAIG.....Sequential Display Area Interest Group	1r
CEDT..... Control Environment Development Team	1s
CEIG.....Control Environment Interest Group	1t
NFFET.....NLS File Feature Extention Team	1u
NFFEIG.....NLS File Feature Extention Interest Group	1v

New groups in the IDENT system

NMAINT.....NLS Maintenance Team	1w
NMIG.....NLS Maintenance Interest Group	1x
BRST.....Baseline Record System Team	1y
BRSIG.....Baseline Record System Interest Group	1z
CST.....Catalog System Team	1a*
CSIG.....Catalog System Interest Group	1aa
BFST..... Basic File System Team	1ab
BFSIG.....Basic File System Interest Group	1ac
DEX2..... DEX-II Development Team	1ad
DEXIG.....DEX Interest Group	1ae
DEXMAINT.....DEX Maintenance Team	1af
LDOCT.....Language Documentation Team	1ag
LDOCIG.....Language Documentation Interest Group	1ah
NDOCT.....NLS Documentation Team	1ai
NDOCIG.....NLS Documentation Interest Group	1aj
SYSCAPT.....System Capacity Team	1ak
SYSCAPIG.....System Capacity Interest Group	1al
TENEXT.....TENEX Team	1am
TENEXIG.....TENEX Interest Group	1an

Dirk H. van Nouhuys

CHI 24-FEB-72 17:16 9268

New groups in the IDENT system

(J9268) 24-FEB-72 17:16; Title: Author(s): Charles H. Irby/CHI; Distribution: Diane S. Kaye, Paul Rech, Michael D. Kudlick, Donald R. Cone, Don Limuti, William R. Ferguson, Priscilla Lister, Robert L. Dendy, Linda L. Lane, Carilyn F. Auerbach, Walter L. Bass, Cary S. Church, William S. Duvall, Douglas C. Engelbart, Beauregard A. Hardeman, Martin E. Hardy, L. D. Hopper, Charles H. Irby, Mil E. Jernigan. Harvey G. Lehtman, John T. Melvin, Leanne B. North, James C. Norton, Cindy Page, Bruce L. Parsley, William H. Paxton, Jeffrey C. Peters, Jake Ratliff, Barbara E. Row, Ed K. Van De Riet, Dirk H. van Nouhuys. Kenneth E. Victor, Donald C. Wallace, Richard W. Watson, Don I. Andrews/SRI-ARC; Sub-Collections: SRI-ARC; Clerk: CHI;

<GJOURNAL>9271.NLS;1, 25-FEB-72 10:24 PL ;
(J9271) 25-FEB-72 10:23; .HJOURNAL="DVN 25-FEB-72 10:23 9271"; Title:
.HED="TNLS Course in March"; Author(s): Dirk H. van Nouhuys/DVN;
Distribution: Robert L. Fink, Karl C. Kelley, Schuyler Stevenson,
Charles Holland, Jeanne B. North, Charles Holland, George N. Petregal,
Steve D. Crocker, Thomas F. Lawrence, John W. McConnell, John F.
Heafner, Robert E. Long, Ari O. J. Ollikainen, James E. White, A. Wayne
Hathaway, Dan L. Murphy, Patrick W. Foulk, Richard A. Winter, Harold R.
Van Zoeren, Alex A. McKenzie, Robert L. Sundberg, Joel M. Winett, Abhay
K. Bhushan, Peggy M. Karp, Thomas N. Pyke, Abe S. Landsberg, B. Michael
Wilber, James A. Moorer, Edward A. Feigenbaum, Robert T. Braden, James
M. Pepin, Barry D. Wessler, John T. Melvin, Richard W. Watson, Ed K.
Van De Riet/NLG RWW EKV(fyi); Sub-Collections: SRI-ARC NLG; Clerk: PL;
.IGD=0; .SNF=72; .MCH=65; .TABSTOPS=8,16,24,32,40,48,56,64; .PGN=-1;
.SCR=2; .PES;
Origin: <LISTER>TNLSCOURSE.NLS;2, 25-FEB-72 10:19 PL ; .PST=1; .MCH=68;
.PGN=0; .PNO=1; .SCR=2; .SNF=73; .SN=0; .DIR=0; .HED=" TNLS Course in
March "; .PES;

1 We will offer a course in NIC TNLS on March 16 and 17th. A few places remain. If you want a place for someone from your site, please inform Mil Jernigan at ARC--(MEJ), (415) 326-6200, ex. 4775.

JBL 25-FEB-72 12:33 9272

Random Ident stuff

(J9272) 25-FEB-72 12:33; Title: uthor(s): Joel B. Levin/JBL:
Distribution: Barbara E. Row/BER; S b-Collections: NIC; Clerk: JBL;

JBL 25-FEB-72 12:33 9272

Random Ident stuff

Is there a place somewhere, like a ournal document or a memo or
a page in the User's Guide, which d scribes the desired formats
etc. and gives examples for enterin individuals or groups into
the ident file? I think that you t ld me yes or no, but I forget
already. 1

Also: I am now getting Author's co ies on-line as you know.
However I am also getting hardcopy opies by air-mail of
everything I send. Can you turn th s off for me? Eleven cents a
shot is an unnecessary expense, alt ough it probably pales into
insignificance beside the cpu time t takes to send a message. 2

Thanx for your help.
JBL 3

George E. Lindamood

WE HAVE DISCARDED UNUSED FILES TO HELP YOU WITH YOUR DISK SPACE SHORTAGE. YOUR SCHEDULED DOWNTIME FOR MAINTENANCE ON MARCH 1 INCONVENIENCES US AS IT HAS IN PAST MONTHS. WE WILL BE HAPPY TO RENDER TESTIMONY TO THIS EFFECT IF IT WILL HELP YOU GET THE MAINTENANCE RESCHEDULED TO ANOTHER (LESS DISRUPTIVE) TIME. WE UNDERSTAND THAT THE NIC IS NOT TO BLAME FOR THIS. 1

GEL 25-FEB-72 13:06 9273

(J9273) 25-FEB-72 13:06; Title: Author(s): George E. Lindamood/GEL; Distribution: Richard W. Watson/RWW; Sub-Collections: NIC; Clerk: GEL;