SEMINAR NO. DA3800

NOS CYBIL

STUDENT HANDOUT

PROPRIETARY NOTICE

The ideas and designs set forth in this document are the property of Control Data Corporation and are not to be disseminated, distributed, or otherwise conveyed to third persons without the express written permission of Control Data Corporation.

REVISION RECORD

REVISION	DESCRIPTION
A	Manual Release.
(07-15-83)	
B	Manual Update. Replaced page 8-6.
(11-05-84)	

Publication No.

DA3800-1

REVISION LETTERS I, O, Q AND X ARE NOT USED

NOS CYBIL

Address comments concerning this manual to:

CONTROL DATA CORPORATION
National Coordinator
Bloomington Facility (MNA02B)
5001 West 80th Street
Bloomington, Minnesota 55437

or use Comment Sheet in the back of this manual.

©COPYRIGHT CONTROL DATA CORPORATION
All Rights Reserved

CONTENTS

	Introduction to the Course	iii
1	Introduction	1
2	Program Structure	9
3	Constants and Variables	17
4	Type	25
5	Statements and Operators	57
6	Procedures	89
7	Standard Procedures	105
8	Machine Dependencies	131
9	Input/Output	149
10	Compile Time Facilities	185
11	Interactive Debugger	203
	Appendix A ASCII Character Set	221
	Appendix B I/O Examples	227
	Appendix C Exercises	231
	Appendix D Project	237

INTRODUCTION TO THE COURSE

This course is designed to provide the student with a comprehensive introduction to CYBIL. The course will include language concepts, language syntax, and coding techniques.

OBJECTIVE

Upon completion of the course, the student will be prepared to implement software designs in CYBIL.

COURSE DESCRIPTION

This version is oriented primarily toward the student who has some prior programming experience. It consists of lectures, with a set of examples to illustrate concepts and constructs. It also provides exercises and a project which supports the lectures and allows time to practice the concepts presented.

NOS CYBIL COURSE OUTLINE

I. Introduction
   A. Course Organization
   B. Compilers
   C. Documentation
   D. Metalanguage
      1. Symbols
      2. Constructs

II. Program Structure
   A. Module
   B. Declarations
   C. Scope of Identifiers
   D. Structured Programming
   E. Elementary Constructs
      1. The alphabet
      2. Identifiers
      3. Basic symbols

III. Constants & Variables
   A. Constants
      1. Declarations
      2. Types
   B. Variables
      1. Declarations
      2. Types
      3. Lifetime
      4. Attributes
      5. Initialization

IV. Type
   A. Declaration
   B. Scalar Types
      1. Integer
      2. Character
      3. Boolean
      4. Ordinal
      5. Subrange
      6. Real

C. Structured Types
1. Sets
2. Arrays
3. Strings
4. Records
D. Pointer Type

V. Statements & Operators
A. Assignment
B. Operators
C. Structured
1. BEGIN
2. WHILE
3. REPEAT
4. FOR
D. Control
1. IF, CASE
2. EXIT, CYCLE

VI. Procedures
A. Declaration
1. Program
2. Procedure
3. Function
B. Parameters
C. Nested Procedures
D. Attributes
E. Pointer to Procedure

VII. Standard Procedures
A. Conversion
B. Adaptables
C. Storage Allocation
1. System heap
2. User heap
3. Sequence
4. PUSH
5. Adaptables

VIII. Machine Dependencies
A. Data Representation - CY170
B. CELL Data Type
C. **MACHINE DEPENDENT FUNCTIONS**
D. **ALIAS**
E. **GENERAL INTRINSICS**

IX. Input Output

1. Files
2. Open/Close
3. Read/Write
4. Utilities

X. Compilation Time Facilities
A. CYBIL Command
B. Compile-time statements
C. Layout Control
D. Toggles

XI. Interactive Debugger
A. Breakpoints
B. Traps
C. Procedures
D. Utilities

COURSE CHART

HOUR	DAY 1	DAY 2	DAY 3	DAY 4	DAY 5
1	I INTRODUCTION	REVIEW	REVIEW	EXERCISES — PROJECT — REVIEW	EXERCISES — PROJECT — REVIEW
2	II PROGRAM STRUCTURE	V OPERATORS — STATEMENTS	VII STANDARD PROCEDURES — ADAPTABLE TYPES — STORAGE ALLOCATION	VIII MACHINE DEPENDENCIES	X COMPILE TIME FACILITIES
3	III CONSTANTS — VARIABLES			IX INPUT/ OUTPUT	
4	IV BASIC TYPES — STRUCTURED TYPES	VI PROCEDURES — FUNCTIONS			XI INTERACTIVE DEBUG
5		IO INTRODUCTION	EXERCISES — PROJECT		
6	EXERCISES	EXERCISES — PROJECT		EXERCISES — PROJECT	

1

INTRODUCTION

COMPILERS

IMPLEMENTATION	SOURCE CODE COMPILATION ON:	OBJECT CODE EXECUTION ON:
CC	CYBER 170	CYBER 170
CM	CYBER 170	M68000
CP	CYBER 170	P-CODE
CS	CYBER 170	CYBER 200
SS	CYBER 200	CYBER 200

CC COMPILER

```
79/12/13. 08.25.14. L32T1
(16) CYBER 176 S/N 101 NOS CLSH.          NOS 1-9G06176/R4B
FAMILY:   ,JW2120
PASSWORD
TERMINAL:      25, NAMIAF
RECOVER/ CHARGE: CHARGE,3916,25008705


 READY.
ASCII
 READY.
BATCH
$RFL,0.


/SES.CYBIL I=FIRST CC
*     COMPILING  FIRST
*     END CYBIL    FIRST -> LISTING, LGO
/SES.LINK170 CYBCLIB
*     END LINK170    LGOB


/LGOB
LGOB.
/REWIND,LISTING
$REWIND,LISTING.
/LIST,F=LISTING
1           CYBIL/CC          V1.0 790712        SOURCE LISTING - ONE
                            79-12-13     08:42:05   PAGE      1
0        0      1 MODULE ONE;
         0      2   PROGRAM MAIN;
        10      3     VAR A,B,C:INTEGER;
        10      4
        10      5     A:=10;
        16      6     B:=20;
        17      7     C:=A+B;
        20      8   PROCEND MAIN;
        20      9  MODEND ONE

1           CYBIL/CC          V1.0 790712  COMPILATION SUMMARY - ONE
                            79-12-13     08:42:05   PAGE      1
0      9 LINES COMPILED.
       NO COMPILATION ERRORS.
 EOI ENCOUNTERED.
```

CYBIL DOCUMENTATION

- CYBIL CLASS NOTES
- CYBIL REFERENCE MANUAL (60455280)
- EXTERNAL SPECIFICATION FOR CYBIL I/O (ARH2739)
- CYBIL INTERACTIVE DEBUGGER (ARH3142)

METALANGUAGE

- A LANGUAGE USED TO DESCRIBE A LANGUAGE
- USED FOR CYBIL SPECIFICATION
- DEFINES STRUCTURE (SYNTAX)
- DOES NOT DEFINE MEANING (SEMANTICS)
- DOES NOT DEFINE LIMITS

METALANGUAGE

< >	Syntactic construct
:: =	Is defined as
[ ]	Zero or one occurrence
{ }	Zero or more occurrence
I	Alternate definition (or)
—	Underscore denotes language element

For example: <variable declaration>

2

PROGRAM STRUCTURE

MODULE STRUCTURE

```
MODULE SKELETON;

   { CONSTANT DECLARATIONS }

   { VARIABLE DECLARATIONS }

   { TYPE DECLARATIONS }

   { PROCEDURE DECLARATIONS }


      { DECLARATIONS }

      { STATEMENTS }


MODEND SKELETON
```

A CYBIL PROGRAM

```
MODULE SAMPLE;
  TYPE RANGE=0..20;
  VAR  R:INTEGER,
    COUNT:RANGE;

  PROGRAM MAIN;
    VAR  A:INTEGER;
    PROCEDURE POWER;
      VAR  I:RANGE;
      R:=1;
      FOR I:=1 TO COUNT DO;
        R:=2*R;
      FOREND;
    PROCEND POWER;

    { EXECUTABLE STATEMENTS }
    COUNT:=5;
    POWER;
    A:=R;

  PROCEND MAIN;
MODEND
```

SCOPE OF IDENTIFIERS

```
MODULE SCOPE;

  { MODULE DECLARATIONS }

  PROGRAM MAIN;

    { DECLARATIONS }

    PROCEDURE AAA;

      { AAA DECLARATIONS }

    PROCEND AAA;

  PROCEND MAIN;

  PROCEDURE BBB;

    { BBB DECLARATIONS }

  PROCEND BBB;

MODEND SCOPE;
```

BASIC CONSTRUCTS

- ALPHABET
- IDENTIFIERS
- SYMBOLS

IDENTIFIERS

- USER DEFINED
- DENOTE
  - — CONSTANTS
  - — VARIABLES
  - — TYPES
  - — PROCEDURE NAMES, etc.
- RULES
  - — 31 CHARACTERS
  - — LETTER FIRST
  - — THEN LETTER, DIGIT, OR # $ @ _
- EXAMPLES
  - — LINES_PER_PAGE
  - — REGISTER#3

BASIC SYMBOLS

- **RESERVED WORDS**

  ARRAY CASE CONST
  ELSE FOR INTEGER
  OF RECORD STRING VAR etc.

- **SPECIAL MARKS**

  + - ; { } = ∧ etc.

- **DIGRAPHS**

  : = < = > = < > ??

..
. SUPLEMENT of RECORD
[ ] ARRAY'S SUBSRIPT , VALUE CONSTRUCT
:
:
, ƀ (
*

3

CONSTANTS & VARIABLES

CONSTANTS

CONST constid = value;

- INTEGER
- CHARACTER
- BOOLEAN
- POINTER
- STRING
- REAL
- LONGREAL

CONSTANT DECLARATION

```
MODULE CONSTANTS;

  PROGRAM MAIN;

    CONST
      UNITY = 1,
      HEX255 = 0FF(16),
      LINES_PER_PAGE = 55,
      BITPATTERN = 11001010(2),
      PLUSSIGN = '+',
      HEADING = 'START OF PAGE',
      END_POINTER = NIL,
      DEBUG = FALSE,
      FIRST = 476872,
      LAST = 638964,
      LENGTH = (LAST - FIRST + 1) * 3,
      REAL_ONE = 1.0,
      REAL_TOO = 1.0E10,
      REAL_EXP = REAL_ONE + REAL_TOO;
```

VARIABLES

VAR varid : type;

- **TYPE**
- **INITIALIZATION**
- **LIFE TIME**
- **ATTRIBUTES**
- **REFERENCE**

VAR varid : [attributes] type : = initialization;

VARIABLES

```
MODULE CONSTANTS;

  PROGRAM MAIN;

    CONST
      BITPATTERN = 11001010(2),
      PLUSSIGN = '+',
      DEBUG = FALSE;

    VAR
      I: INTEGER;
      B: BOOLEAN;
      C: CHAR;

    I := BITPATTERN;
    C := PLUSSIGN;
    B := DEBUG;

  PROCEND MAIN;

MODEND CONSTANTS.
```

VARIABLE INITIALIZATION

```
MODULE VARIABLES;

  VAR {VARIABLE INITIALIZATION}
    I,
    J,
    K: INTEGER,
    RANGE: INTEGER := 30,
    TRUTH: BOOLEAN := FALSE,
    TSTCHAR: : CHAR := '5',
    REALLY: REAL := 0.0;

MODEND VARIABLES;
```

VARIABLE ATTRIBUTES

```
PROGRAM MAIN;

  VAR {VARIABLES WITH ATTRIBUTES}
    COUNTER: [STATIC] INTEGER := 0,
    TABLES: [XDCL] CHAR := 'J',
    KEY: [XREF] CHAR;
    SYSTVAL: [READ] BOOLEAN := TRUE;

PROCEND MAIN;
```

4

TYPE

TYPE DECLARATION

```
TYPE typeid = type;
```

A) FIXED TYPES

- SCALAR
- STRUCTURED
- POINTER

B) FIXABLE TYPES

- ADAPTABLE
- BOUND VARIANT RECORDS

= PREDECESSOR & SUCCESSOR CAN BE DETERMINED

SCALAR TYPES

- INTEGER
- CHARACTER
- BOOLEAN
- SUBRANGE
- ORDINAL
- ~~REAL~~
- ~~LONGREAL~~

SUBRANGE

```
TYPE { PREDEFINED }
   CH = CHAR;
   INT = INTEGER;
   TF = BOOLEAN;
   RL = REAL;

TYPE { SUBRANGE }
   LETTER = 'A' .. 'Z';
   ADDRESS = 0(16) .. 7FFFFFF(16);
   REGISTER = 0 .. 7;
   INDEX1 = 0 .. 100;
   INDEX2 = - 20 .. 20;
   LIMIT = 1.0E - 3 .. 1.0E + 3;
```

ORDINAL TYPE

```
TYPE (ORDINAL )
  COLOR = (RED, ORANGE, YELLOW, GREEN, BLUE);
  RONS_BOOLEAN = (YES, MAYBE, NO);
  TYPE_FORM = (NUMERIC, SYMBOLIC, POINTER);
  DEVICE_TYPE = (LP512, CR425, DK844, DK819,
                 MT667, NT669);
TYPE ( SUBRANGE OF ORDINAL )
  HOT_COLOR = RED .. YELLOW;
  DISK = DK844 .. DK819;
```

SCALAR TYPE

```
        0      1 MODULE SCALAR_INDEX_TYPES;
        0      2
        0      3   PROGRAM MAIN;
        0      4
        0      5     TYPE { SUBRANGE }
        0      6       LETTER = 'A' .. 'Z',
        0      7       ADDRESS = 0(16) .. 7FFFFFF(16),
        0      8       REGISTER = 0 .. 7,
        0      9       INDEX1 = 0 .. 100,
        0     10       INDEX2 = - 20 .. 20,
        0     11       LIMIT = 1.0E - 3 .. 1.0E + 3;
        0     12
        0     13     TYPE { ORDINAL }
        0     14       COLOR = (RED, ORANGE, YELLOW, GREEN, BLUE),
        0     15       RONS_BOOLEAN = (YES, MAYBE, NO),
        0     16       TYPE_FORM = (NUMERIC, SYMBOLIC, POINTER),
        0     17       DEVICE_TYPE = (LP512, CR425, DK844, DK819, NT669);
        0     18
        0     19     TYPE { SUBRANGE OF ORDINAL }
        0     20       HOT_COLOR = RED .. YELLOW,
        0     21       DISK = DK844 .. DK819;
        0     22
        0     23     VAR
        0     24       REQUEST: DISK,
        0     25       EXECUTIVE_DECISION: RONS_BOOLEAN,
        0     26       STARTING_CHAR: LETTER;
        0     27
        0     28     { BEGIN STATEMENTS }
        0     29     REQUEST := DK844;
*ERROR*     30     REQUEST := NT669;
        0     31     EXECUTIVE_DECISION := YES;
*ERROR*     32     EXECUTIVE_DECISION := GO;
*ERROR*     33     STARTING_CHAR := '5';
        0     34
        0     35   PROCEND MAIN;
        0     36 MODEND SCALAR_INDEX_TYPES;

                LINE    ERR  COL     TEXT

FATAL             30                 VALUE OUT OF RANGE.
FATAL             32                 UNDECLARED IDENTIFIER - GO.

FATAL             33                 VALUE OUT OF RANGE.
```

STRUCTURED TYPES

- SET
- STRING
- ARRAY
- RECORD

SET CONCEPTS

TYPE settypeid = SET OF type;

- **REPRESENTATION**

m_1	m_2	m_3	...

- **MEMBERSHIP**
- **INITIALIZATION**
- **ASSIGNMENT**

SET TYPE

```
TYPE (INITIALIZATION)
  COLOR = (RED, ORANGE, YELLOW, GREEN, BLUE);
  HOT_COLOR = RED .. YELLOW;
  RANGE = - 10 .. 10;

TYPE (SET)
  ACCESS = SET OF (N_READ, N_WRITE, N_EX);
  RAINBOW = SET OF COLOR;
  FLAME = SET OF HOT_COLOR;
  NZ_SET = SET OF 1 .. 10;
  NUM_SET = SET OF - 5 .. 5;
  ALF_SET = SET OF 'A' .. 'Z';
  VALSET = SET OF RANGE;
```

SET INITIALIZATION

```
 000000      1
 000000      2 MODULE STRUCTURED_TYPE;
 000000      3
 000000      4   TYPE (INITIALIZATION)
 000000      5     COLOR = (RED, ORANGE, YELLOW, GREEN, BLUE);
 000000      6     HOT_COLOR = RED .. YELLOW;
 000000      7     RANGE = - 10 .. 10;
 000000      8
 000000      9   TYPE (SET)
 000000     10     ACCESS = SET OF (N_READ, N_WRITE, N_EX);
 000000     11     RAINBOW = SET OF COLOR;
 000000     12     FLAME = SET OF HOT_COLOR;
 000000     13     NZ_SET = SET OF 1 .. 10;
 000000     14     NUM_SET = SET OF - 5 .. 5;
 000000     15     ALF_SET = SET OF 'A' .. 'Z';
 000000     16     VALSET = SET OF RANGE;
 000000     17
 000000     18   VAR
 000000     19     F: FLAME;
 000000     20     N: NZ_SET := [4, 5, 6];
 000000     21     A: ALF_SET := [];
 000000     22
 000000     23   PROGRAM MAIN;
 000000     24
*ERROR*     25     F := RED;
*ERROR*     26     F := 1;
 000000     27     F := $FLAME[RED];
 000000     28
*ERROR*     29     N := 1;
 000000     30     N := $NZ_SET[1, 7, 2];
 000000     31
*ERROR*     32     A := $ALF_SET[A, B];
 000000     33     A := $ALF_SET['A', 'B'];
 000000     34   PROCEND MAIN;
 000000     35 MODEND STRUCTURED_TYPE;

                                V0.1 770920        ERROR LISTING - STRUCTURE
D_TYPE                        78/07/26.   16:18:45.  PAGE      1
0  LINE   SEVERITY   ERROR
0     25  ERROR      INCOMPATIBLE TYPES ARE NOT ASSIGNABLE.
0     26  ERROR      INCOMPATIBLE TYPES ARE NOT ASSIGNABLE.
0     29  ERROR      INCOMPATIBLE TYPES ARE NOT ASSIGNABLE.
0     32  ERROR      TYPE OF VALUE ELEMENT DOES NOT AGREE WITH TARGET TYP
E .
8     32  ERROR      UNDECLARED IDENTIFIER - B.
```

ARRAY CONCEPTS

TYPE artypid = ARRAY [index type] OF type;

- **INDEX**
- **HOMOGENEOUS COMPONENTS**
- **MULTIDIMENSIONED ARRAYS**
- **PACKING**
- **INITIALIZING**
- **REFERENCE**

ARRAY TYPE

```
TYPE
  INDEX = 1 .. 8;
  TOKENS = (ALPHA, NUMERIC, SPECIAL, UNUSED);
  TYPICAL = ARRAY[1 .. 10] OF INTEGER;
  MEMORY = ARRAY[0 .. 4095] OF 0 .. 4095;
  CODES = ARRAY[CHAR] OF TOKENS;
  PPUS = ARRAY[1 .. 10] OF MEMORY;
  TRANS = ARRAY['A' .. 'Z'] OF INTEGER;
  CHARS = ARRAY[INDEX] OF CHAR;
  BOOLA = PACKED ARRAY[1 .. 3] OF BOOLEAN;
```

ARRAY TYPE

```
 000000      1
 000000      2 MODULE ARRAY_TYPE;
 000000      3
 000000      4   TYPE
 000000      5     INDEX = 1 .. 8,
 000000      6     TOKENS = (ALPHA, NUMERIC, SPECIAL, UNUSED),
 000000      7     TYPICAL = ARRAY[1 .. 10] OF INTEGER,
000000       8     MEMORY = ARRAY[0 .. 4095] OF 0 .. 4095,
 000000      9     CODES = ARRAY[CHAR] OF TOKENS,
 000000     10     PPUS = ARRAY[1 .. 10] OF MEMORY,
 000000     11     TRANS = ARRAY['A' .. 'Z'] OF 0 .. 255,
 000000     12     CHARS = ARRAY[INDEX] OF CHAR,
 000000     13     BOOLA = PACKED ARRAY[1 .. 3] OF BOOLEAN;
 000000     14
 000000     15   VAR
 000000     16     X: PPUS,
 000000     17     B: BOOLA := [TRUE, FALSE, FALSE],
 000000     18     Y: TRANS := [REP 26 OF 0],
 000000     19     Z: CHARS := ['1', '2', 'A', 'B', REP 4 OF * ],
 000000     20     R_B: BOOLEAN,
 000000     21     R_Y: INTEGER,
 000000     22     R_Z: CHAR;
 000000     23
 000000     24   PROGRAM MAIN;
 000000     25
 000000     26     X[3][4000] := 0;
*ERROR*     27     X[4000][3] := 1;
*ERROR*     28     X[3, 4000] := 10;
 000000     29
*ERROR*     30     Y[A] := 1;
*ERROR*     31     Y[1] := 1;
 000000     32     Y['A'] := 1;
 000000     33
 000000     34     Z[7] := '+';
 000000     35     Z[8] := '-';
 000000     36
 000000     37     R_B := B[2];
 000000     38     R_Y := Y['Z'];
 000000     39     R_Z := Z[4];
 000000     40
 000000     41   PROCEND MAIN;
 000000     42 MODEND ARRAY_TYPE.
```

```
                          V0.1 770920          ERROR LISTING - ARRAY_TYP
E                       78/07/26.    16:44:06.   PAGE      1
0  LINE    SEVERITY    ERROR
0     27   ERROR       CONSTANT SUBSCRIPT OUTSIDE INDEX TYPE OF ARRAY.
0     28   WARNING     ARRAY REFERENCES MUST USE '][' INSTEAD OF A COMMA.

0     30   ERROR       UNDECLARED IDENTIFIER - A.
0     31   ERROR       SUBSCRIPT DOES NOT CONFORM TO INDEX TYPE OF ARRAY.

0     31   ERROR       LEFT PART OF ASSIGNMENT STATEMENT MUST BE A VARIABLE
                       OR FUNCTION IDENTIFIER.
```

STRING CONCEPTS

TYPE strtypeid = STRING (length);

- **REPRESENTATION**

c_1	c_2	c_3	c_4	------	c_n

n = length

- **SUBSTRING REFERENCE**
- **ASSIGNMENT RULES**

STRING TYPE

```
TYPE
  SIMP = STRING (11);
  LIST = ARRAY[1 .. 80] OF CHAR;
  C_AR = PACKED ARRAY[1 .. 80] OF CHAR;
  CARD = STRING (80);
  MESS = ARRAY[1 .. 10] OF STRING (11);
```

STRING REFERENCE

```
VAR
  S : STRING (n);
```

S	Refers to entire string
S (I)	Refers to the Ith character in the string
S (I, J)	Refers to the string which starts at the Ith character and is J characters long
S (I, *)	Refers to the string which starts at the Ith character and continues to the end of S

STRING REFERENCE

```
MODULE STRING_TYPE;

  PROGRAM MAIN;

    TYPE
      SIMP = STRING (11),
      LIST = ARRAY[1 .. 80] OF CHAR,
      C_AR = PACKED ARRAY[1 .. 80] OF CHAR,
      CARD = STRING (80),
      MESS = ARRAY[1 .. 10] OF STRING (11);


    VAR
      S: [STATIC] SIMP := ' ELEVEN 11 ',
      T: STRING (3),
      C: CHAR,
      X: MESS;

    X[1] := '123456789AB';
    X[2] (1) := '*';
    X[3] (1, 3) := 'XYZ';

    T := S (2, 3);
    X[5] (2, 3) := T;

    C := S (10);
    S (4) := C;
    S (8, * ) := 'FOUR';

  PROCEND MAIN;
MODEND STRING_TYPE.
```

RECORD CONCEPTS

TYPE id = RECORD fid_1 :$type_1$,...RECEND;

- NONHOMOGENOUS COMPONENTS
- RECORD TYPES
  - INVARIANT
  - VARIANT
  - BOUND VARIANT
  - ADAPTABLE
- INITIALIZATION
- REFERENCE
- PACKING

RECORD TYPE

```
MODULE RECORD_TYPE;

  PROGRAM MAIN;

    TYPE
      CREDIT = RECORD
        NAME: STRING (10),
        ADDR: STRING (41),
        STATE: STRING (26),
        INCOME: 0 .. 500000,
        RATING: 'A' .. 'Z'
      RECEND;

    VAR
      Q: CREDIT;

    { BEGIN STATEMENTS }
    Q.RATING := 'A';
    Q.NAME := ' PASCAL-X ';

  PROCEND MAIN;
MODEND RECORD_TYPE;
```

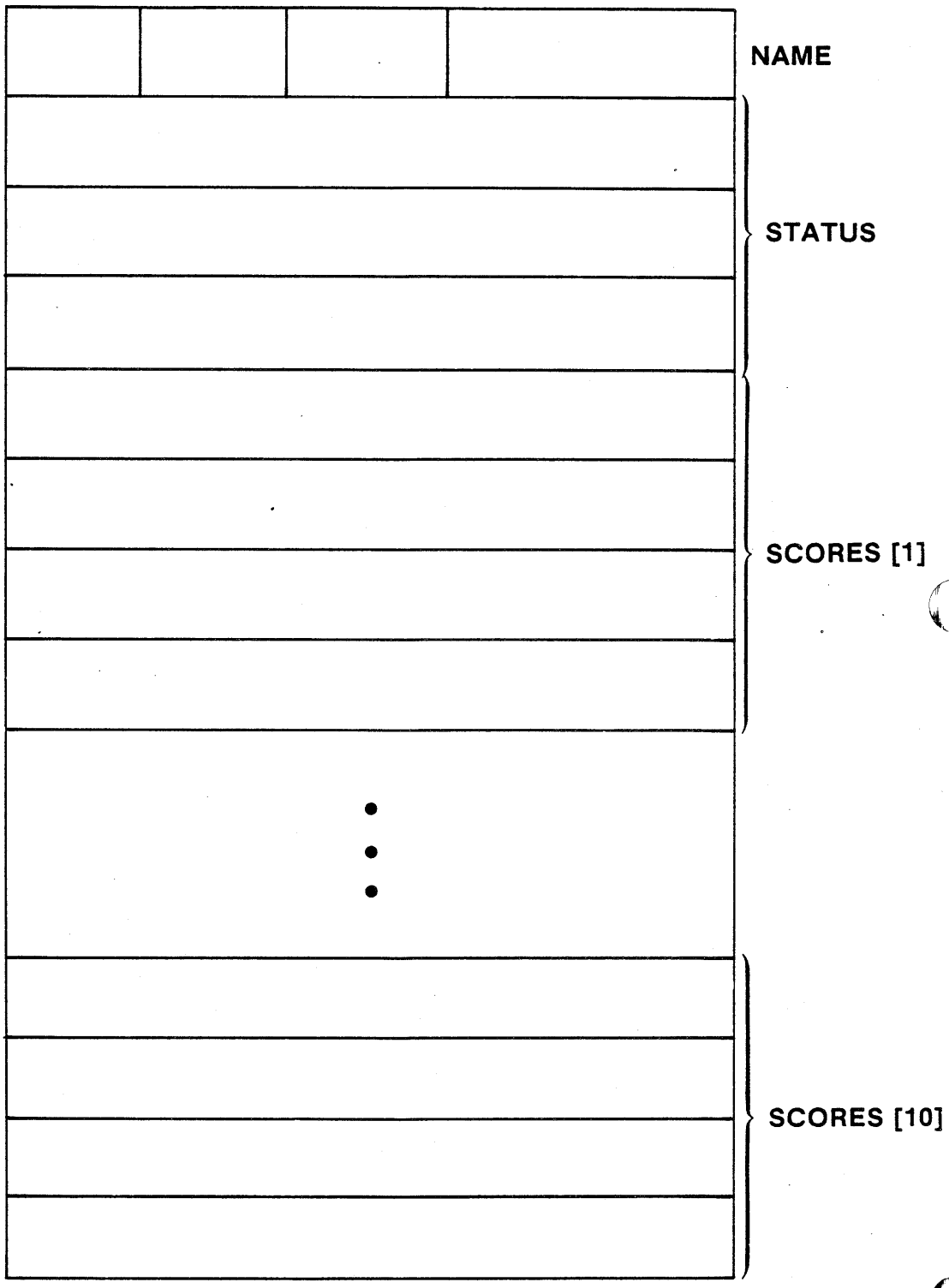
RECORD INITIALIZATION

```
MODULE VARIABLE_INITIALIZATION;

  VAR { RECORD INITIALIZATION }
    R: RECORD
       AGE: 6 .. 66,
       MARRIED,
       SEX: BOOLEAN,
       DATE: RECORD
          DAY: 1 .. 31,
          MONTH: 1 .. 12,
          YEAR: 70 .. 90,
       RECEND
    RECEND := [23, TRUE, TRUE, [3, 5, 73]];

MODEND VARIABLE_INITIALIZATION;
```

FIELD REFERENCE

```
MODULE RECORD_WITH_ARRAY_FIELDS;

  PROGRAM MAIN;

    TYPE
      DATE = RECORD
        DAY: 1 .. 31,
        MONTH: ARRAY[1 .. 4] OF CHAR,
        YEAR: 1900 .. 2100,
        SCORE: INTEGER,
      RECEND,
      STAT = RECORD
        AGE: 6 .. 66,
        MARRIED,
        SEX: BOOLEAN,
      RECEND,
      BOOK = RECORD
        NAME: STRING (3),
        STATUS: STAT,
        SCORES: ARRAY[1 .. 10] OF DATE,
      RECEND;

    VAR
      Q: BOOK;


      Q.NAME := 'JHW';
      Q.STATUS.AGE := 30;
      Q.SCORES[1].SCORE := 92;


  PROCEND MAIN;
MODEND.
```

BOOK



PACKED RECORD

```
TYPE
  COST = PACKED RECORD
    MFG_COST: 0 .. 9,
    INT_COST: 0 .. 9,
    EXT_COST: 0 .. 9,
  RECEND;

  IDNT = PACKED RECORD
    ITEM_NUMBER: INTEGER;
    QTY_ORDERED: INTEGER;
  RECEND;

  DATA = PACKED ARRAY[1 .. 100] OF RECORD
    REC1: IDNT;
    REC2: COST;
  RECEND;
```

VARIANT RECORD

```
CONST
  MAXSIZE = 4095;

TYPE
  SHAPE = (TRIANGLE, PARALLELOGRAM, SQUARE),
  SIDE_RANGE = 0 .. MAXSIZE,
  ANGLE = - 180 .. 180;

TYPE { VARIANT RECORD }
  FIGURE = RECORD
    X,
    Y: SIDE_RANGE,
    CASE S: SHAPE OF
    =TRIANGLE=
      SIDE: SIDE_RANGE,
      A1,
      A2: ANGLE,
    =PARALLELOGRAM=
      SKEW: ANGLE,
      S1,
      S2: SIDE_RANGE,
    =SQUARE=
      EDGE: SIDE_RANGE,
    CASEND
  RECEND;
```

VARIANT REFERENCE

```
 1
 2 MODULE VARIANT_RECORD_TYPE;
 3
 4   PROGRAM MAIN;
 5
 6     CONST
 7       MAXSIZE = 4095;
 8
 9     TYPE
10       SHAPE = (TRIANGLE, PARALLELOGRAM, SQUARE),
11       SIDE_RANGE = 0 .. MAXSIZE,
12       ANGLE = - 180 .. 180;
13
14     TYPE ( VARIANT RECORD )
15       FIGURE = RECORD
16         X,
17         Y: SIDE_RANGE,
18         CASE S: SHAPE OF
19         =TRIANGLE=
20           SIDE: SIDE_RANGE,
21           A1,
22           A2: ANGLE,
23         =PARALLELOGRAM=
24           SKEW: ANGLE,
25           S1,
26           S2: SIDE_RANGE,
27         =SQUARE=
28           EDGE: SIDE_RANGE,
29         CASEND
30       RECEND;
31
32     VAR
33       Q,
34       R: FIGURE;
35
36     BEGIN
37       Q.X := 500;
38       Q.Y := 600;
39       Q.S := SQUARE;
40       Q.EDGE := 50;
41
42       R.S := TRIANGLE;
43       R.SIDE := 22;
44     END
45
46   PROCEND MAIN;
47 MODEND.
```

POINTER CONCEPTS

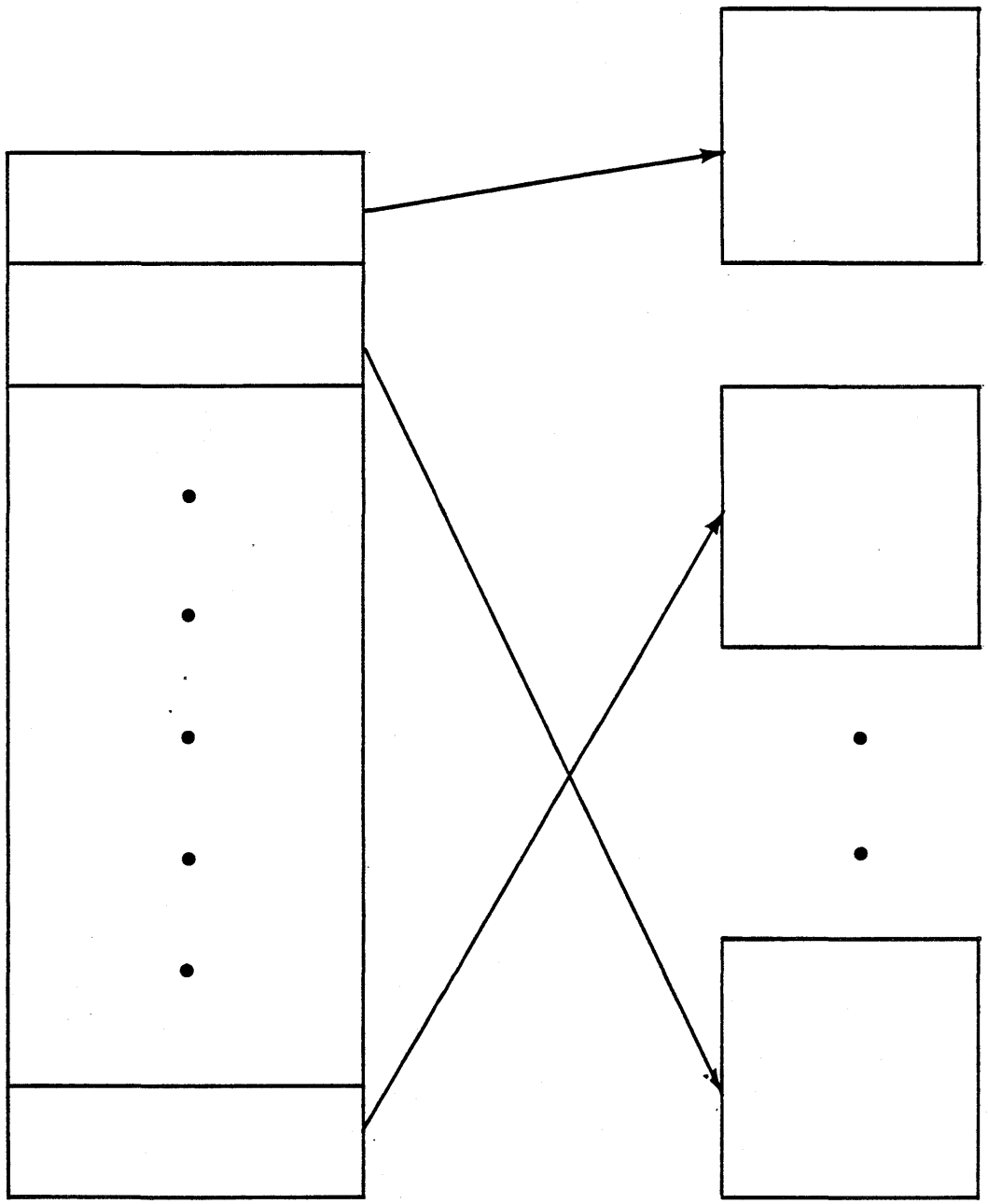
TYPE ptrtypeid = ∧ type

- POINTER TO TYPE
- LIFETIME
- REFERENCE/DEREFERENCE
- ASSIGNMENT
- APPLICATIONS
  - — LINKED LIST
  - — TABLE OF POINTERS

POINTER REFERENCE

```
MODULE POINTER_TYPE;

  PROGRAM MAIN;

    TYPE
      COLOR = (RED, ORANGE, YELLOW);
      TABLE = ARRAY [1 .. 30] OF COLOR;
      C_PTR = ^COLOR;

    VAR
      C: COLOR;
      T: TABLE;
      PC: C_PTR;
      PT: ^TABLE;

    C := RED;
    PC := ^C;
    PC^ := YELLOW;
    C := PC^;
    PT := ^T;
    PT^ [1] := ORANGE;
    C := PT^ [1];
    PT := NIL;
  PROCEND MAIN;
MODEND POINTER_TYPE;
```

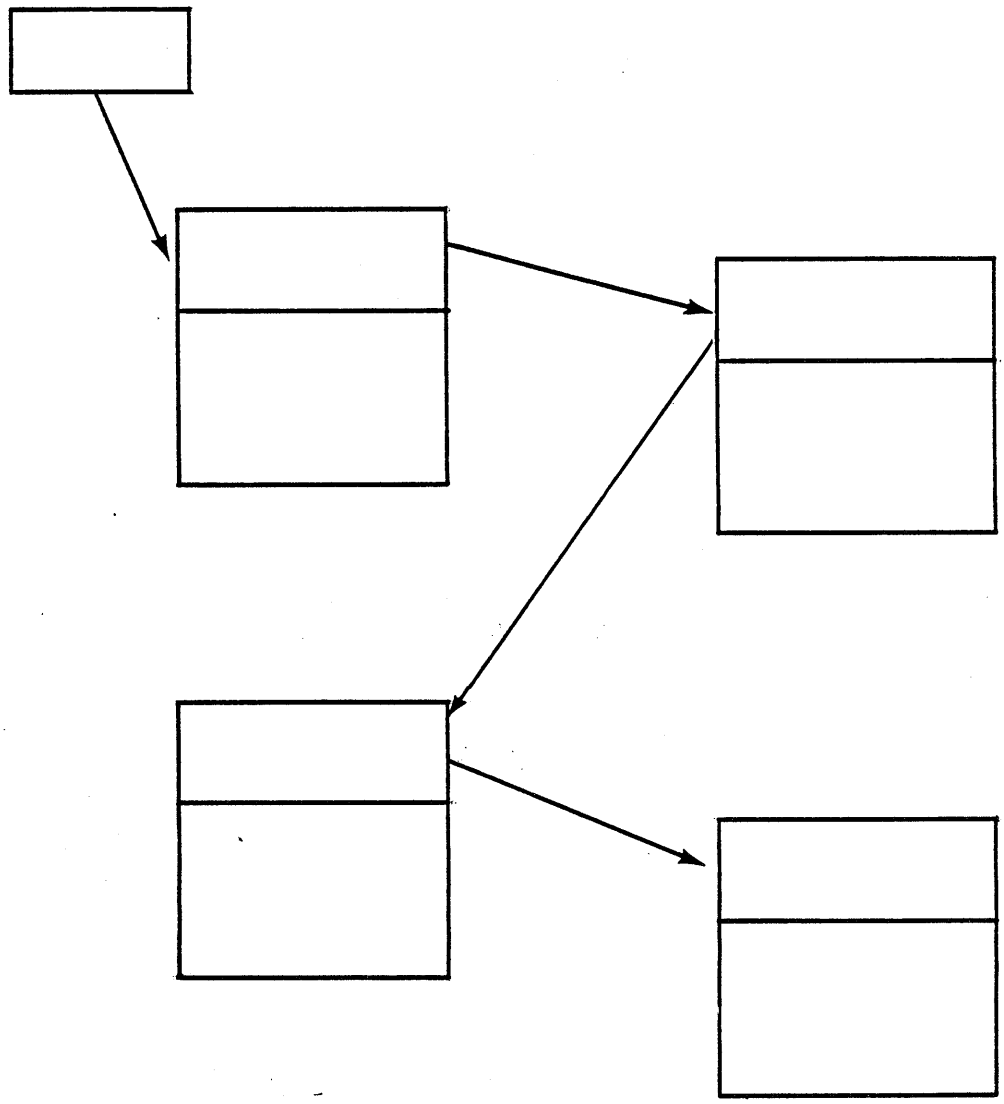
ARRAY OF POINTERS



ARRAY OF POINTERS

```
MODULE ARRAY_OF_POINTERS;

  TYPE
    STATS = PACKED RECORD
      NAME: STRING (20),
      AGE: 1 .. 100,
      MARRIED,
      SEX: BOOLEAN
    RECEND;

  VAR
    POINTER_ARRAY: ARRAY[1 .. 1000] OF ^STATS;

MODEND;
```

LINK-LIST



POINTER EXAMPLE

```
MODULE LINK_LIST;

  TYPE
    LINK = ^REC;
    REC = RECORD
      VALUE: INTEGER;
      PTR: LINK
    RECEND;

  VAR
    FIRST: LINK:= NIL;

  PROGRAM MAIN;

    VAR
      ELEMENT: REC;
      POINTER: LINK;

{ LINK ELEMENT TO THE TOP }
{ OF THE LIST }


    POINTER := ^ELEMENT;
    POINTER^.VALUE := 3;
    POINTER^.PTR := FIRST;
    FIRST := POINTER;

{ DEREFERENCE THE POINTER; I.E.,
{ REFERENCE THE LINKED RECORD. }

    ELEMENT := FIRST^;
    FIRST^.VALUE := 6;
  PROCEND MAIN;
MODEND LINK_LIST;
```

REFERENCE

- **POINTERS**

  :=ptr∧
  ptr ∧ :=
  :=∧ variable

- **ARRAY**

  arr [index]:=
  arr [$index_1$] [$index_2$]:=

- **STRING**

  str:=
  str (start):=
  str (start, length):=
  str (start, *):=

- **RECORD**

  rec:=
  rec. field:=

- **SET**

  set:=

5

STATEMENTS AND OPERATORS

OPERATOR PRECEDENCE

- NOT
- MULTIPLYING
  * , /
  DIV , MOD
  AND
- SIGN OPERATORS
  + , -
- ADDING
  + , -
  OR
  XOR
- RELATIONAL
  < , < =
  > , > =
  = , < >
  IN

MULTIPLYING OPERATORS

OPERATOR	OPERATION	OPERANDS	RESULT
*	multiplication	INTEGER REAL, LONGREAL	INTEGER REAL, LONGREAL
	set intersection	SET	SET
DIV /	integer quotient real quotient	INTEGER REAL, LONGREAL	INTEGER REAL, LONGREAL
MOD	remainder function	INTEGER	INTEGER
AND	logical "and"	BOOLEAN	BOOLEAN

MULTIPLYING

```
MODULE EXPRESSIONS;

  VAR
    x: INTEGER := 7,
    y: INTEGER := 2,
    z: INTEGER,

    b1: BOOLEAN := TRUE,
    b2: BOOLEAN := FALSE,
    b3: BOOLEAN,

    xx: REAL := 7.0,
    yy: REAL := 2.0,
    zz: REAL;

  PROGRAM MAIN;
{ MULTIPLYING OPERATORS }
    z := x * y; { 14 }
    z := x DIV y; { 3 }
    z := x MOD y; { 1 }
    zz := xx * yy; { 14.0 }
    zz := xx / yy; { 3.5 }

    b3 := b1 AND b2; { FALSE }
    b3 := b1 AND b1; { TRUE }

  PROCEND MAIN;
MODEND EXPRESSIONS
```

ADDING OPERATORS

OPERATOR	OPERATIONS	OPERANDS	RESULT
+	addition	INTEGER REAL, LONGREAL	INTEGER REAL, LONGREAL
	set union	SET	SET
-	subtraction	INTEGER REAL, LONGREAL	INTEGER REAL, LONGREAL
	boolean difference	BOOLEAN	BOOLEAN
	set difference	SET	SET
OR	logical "or"	BOOLEAN	BOOLEAN
XOR	exclusive "or"	BOOLEAN	BOOLEAN

ADDING

```
MODULE EXPRESSIONS;

  VAR
    X : INTEGER := 7,
    Y : INTEGER := 2,
    Z : INTEGER ,

    B1 : BOOLEAN := TRUE,
    B2 : BOOLEAN := FALSE,
    B3 : BOOLEAN ,
    B4 : BOOLEAN ;

  PROGRAM MAIN;

      { ADDING OPERATORS }
    Z := X + Y;                    {  9 }
    Z := X - Y;                    {  5 }

    B3 := B1 - B2;               {TRUE }
    B3 := B2 - B1;               {FALSE}

    B4 := B1 OR B1;              {TRUE }
    B4 := B2 OR B2;              {FALSE}

  PROCEND MAIN;

MODEND EXPRESSIONS
```

RELATIONAL OPERATORS

OPERATOR	OPERATION	LEFT OPERAND	RIGHT OPERAND	RESULT
<	- less than	SCALAR	SCALAR	BOOLEAN
<=	- less than or equal to			
>	- greater than	STRING (n)	STRING (n)	BOOLEAN
>=	- greater than or equal to	S(k) CHAR	CHAR S(k)	BOOLEAN BOOLEAN
=	- equal to			
<>	- not equal to			
IN	set membership	SCALAR	SET	BOOLEAN
= <> <= >=	- identity - different - is contained in - contains	SET	SET	BOOLEAN
= <>	- equal to - not equal to	RECORD	RECORD	BOOLEAN
		POINTER	POINTER	BOOLEAN

RELATIONAL

```
MODULE EXPRESSIONS;

  VAR
    X : INTEGER := 7,
    Y : INTEGER := 2,
    B3 : BOOLEAN ,
    B4 : BOOLEAN ,

    CH : CHAR := 'M',
    S_ : STRING(3) := 'LMN',
    P : ^STRING(3);

  PROGRAM MAIN;
      { RELATIONAL OPERATORS }
    B3 := X <= Y;                {FALSE}
    B3 := CH = 'M';              {TRUE }

    B4 := CH <> S_(2);           {FALSE}
    B4 := S_(3) > 'A';           {TRUE }

    B3 := P = NIL;               {TRUE }
  PROCEND MAIN;

MODEND EXPRESSIONS
```

EXPRESSIONS

```
X := 7;
Y := 2;
B1 := TRUE;
B2 := FALSE;
CH := 'M';

Z := ( X + Y - 3 ) * ( X + 2 ); { 54 }

Z := (( X*X) + (Y*Y)) DIV 5;   { 10 }


B3 := (X <= Y) OR B1;          {TRUE}

B4 := ( CH = 'L' ) AND ( B2 = FALSE ); {FALSE}
```

OPERATIONS ON SETS

```
MODULE SET_ASSIGNMENT;
PROGRAM MAIN;

  TYPE
    B = SET OF 0 .. 5;

  VAR
    SETA:   B,
    SETB:   B,
    SETC:   B,
    BOO:BOOLEAN,
    THREE:[STATIC]INTEGER := 3;

      { OPERATIONS ON SETS }
    SETB   := $B[0,1,2];          {0,1,2}
    SETC   := SETB + $B[0,1,4];   {0,1,2,4}
    SETA   := SETB  +  SETC;      {0,1,2}
    BOO    := 5 IN SETA;          {FALSE}
    BOO    := SETA = SETC;        {FALSE}
    BOO    := SETA <> SETC;       {TRUE}
    BOO    := SETA <= SETC;       {TRUE}
    BOO    := SETA >= SETC;       {FALSE}
    SETB   := SETC - SETA;        {4}
    SETB   := SETA - SETC;        { }
    SETB   := -SETA;              {3,4,5}
    SETB   := -$B[];              {FULL}
    SETA   := SETA + $B[THREE];   {0,1,2,3}
PROCEND MAIN;
MODEND SET_ASSIGNMENT.
```

STRUCTURED STATEMENTS

- SEQUENTIAL

  BEGIN

- REPETITIVE

  WHILE

  REPEAT

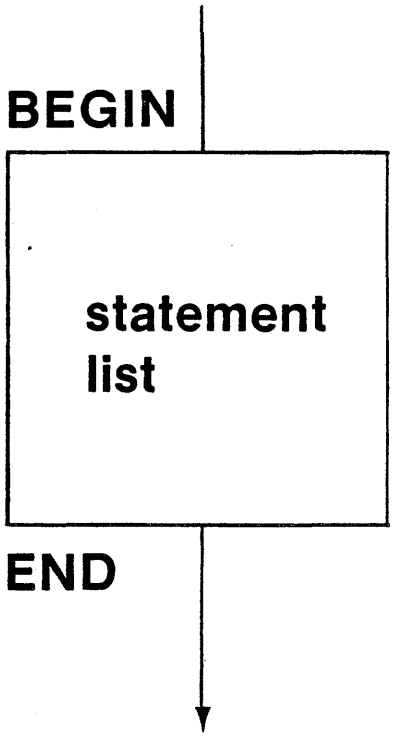
  FOR

- STRUCTURED STATEMENTS

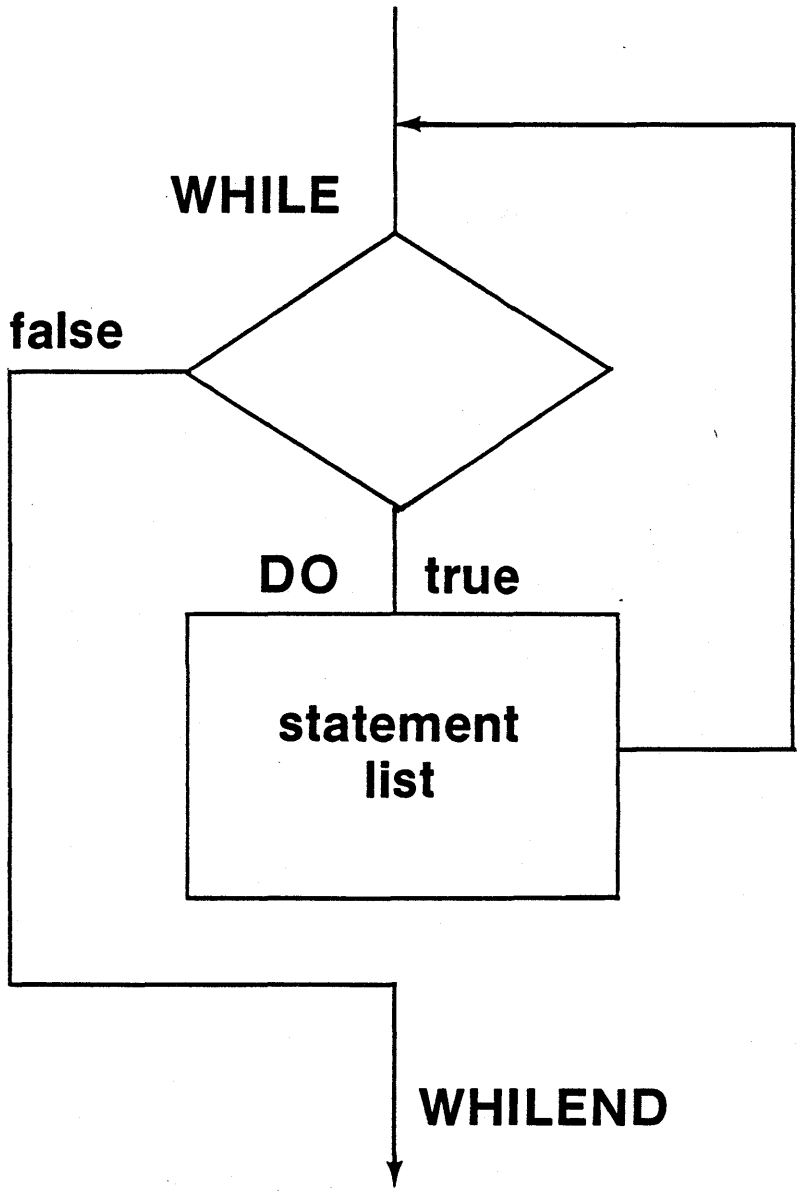
  MAY HAVE LABELS

BEGIN STATEMENT

- USED TO GROUP STATEMENTS
- MAY HAVE A LABEL



WHILE STATEMENT

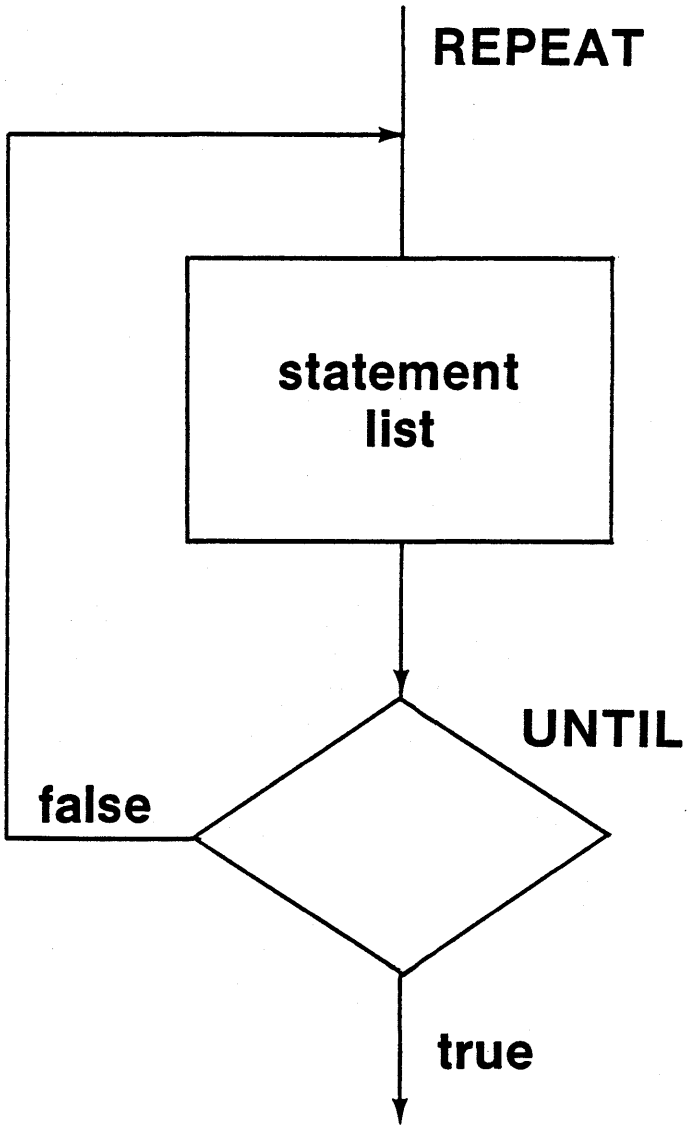


WHILE

```
MODULE WHILE_STATEMENT;

  PROGRAM MAIN;

    VAR
      I: INTEGER;
      A: ARRAY[0 .. 999] OF BOOLEAN;

    I := 0;
    WHILE A[I] = TRUE DO
      I := I + 1;
    WHILEND;

  PROCEND MAIN;
MODEND WHILE_STATEMENT.
```

NOTE: A has not been initialized.

REPEAT STATEMENT

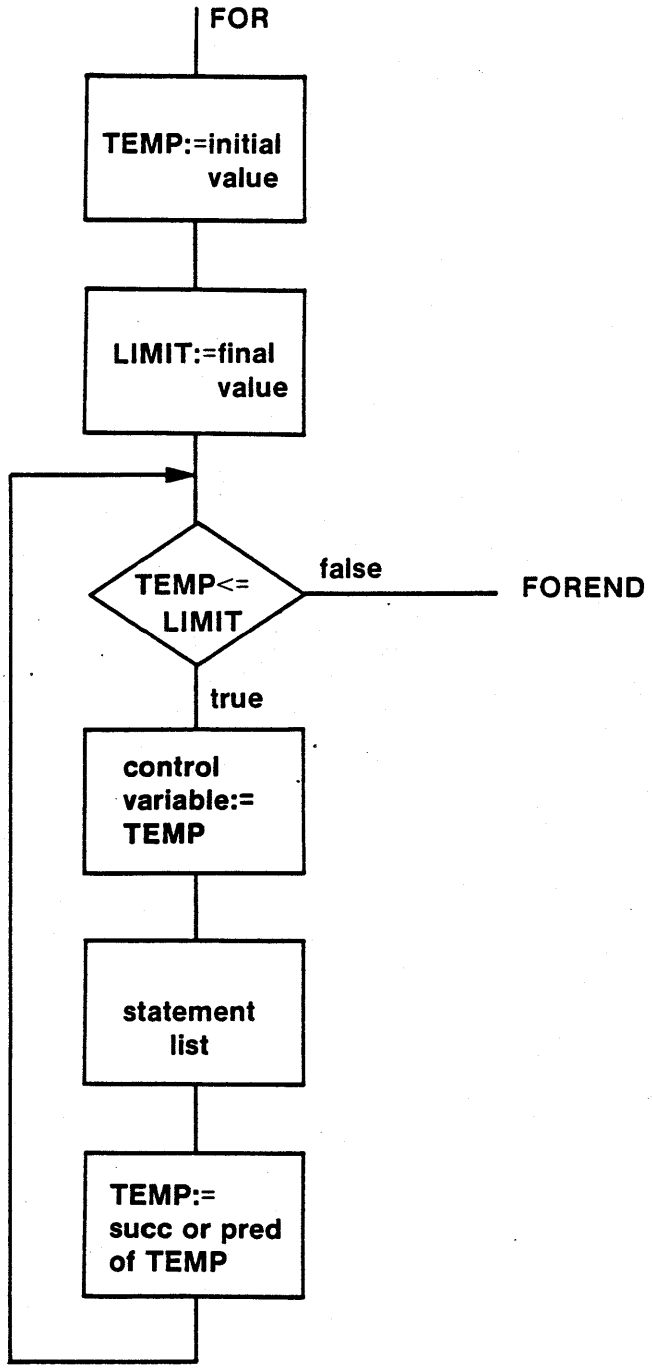


REPEAT

```
MODULE REPEAT_STATEMENT;

  PROGRAM MAIN;

    VAR
      X: INTEGER;
      Y: INTEGER;
      STR: STRING (100);

    X := 0;
    Y := 0;
    REPEAT
      X := X + 1;
      Y := Y + 2;
    UNTIL STR (X) <> ' ';

  PROCEND MAIN;
MODEND REPEAT_STATEMENT.
```

NOTE: STR has not been initialized.

FOR STATEMENT



FOR (integer)

```
MODULE FOR_STATEMENT;

  PROGRAM MAIN;

    CONST
      INITIAL = 1,
      FINAL = 100;

    VAR
      W: INTEGER,
      A: ARRAY[INITIAL .. FINAL] OF INTEGER;

    FOR W := INITIAL TO FINAL DO
      A[W] := 0;
    FOREND;

  PROCEND MAIN;
MODEND FOR_STATEMENT.
```

FOR (character)

```
MODULE FOR_STATEMENT_2;

  PROGRAM MAIN;

    { COUNT CHARACTERS DOWN }

    VAR
      W: CHAR;
      A: ARRAY['A' .. 'Z'] OF INTEGER;

    FOR W := 'Z' DOWNTO 'M' DO
      A[W] := 0;
    FOREND;

  PROCEND;

MODEND
```

FOR (ordinal)

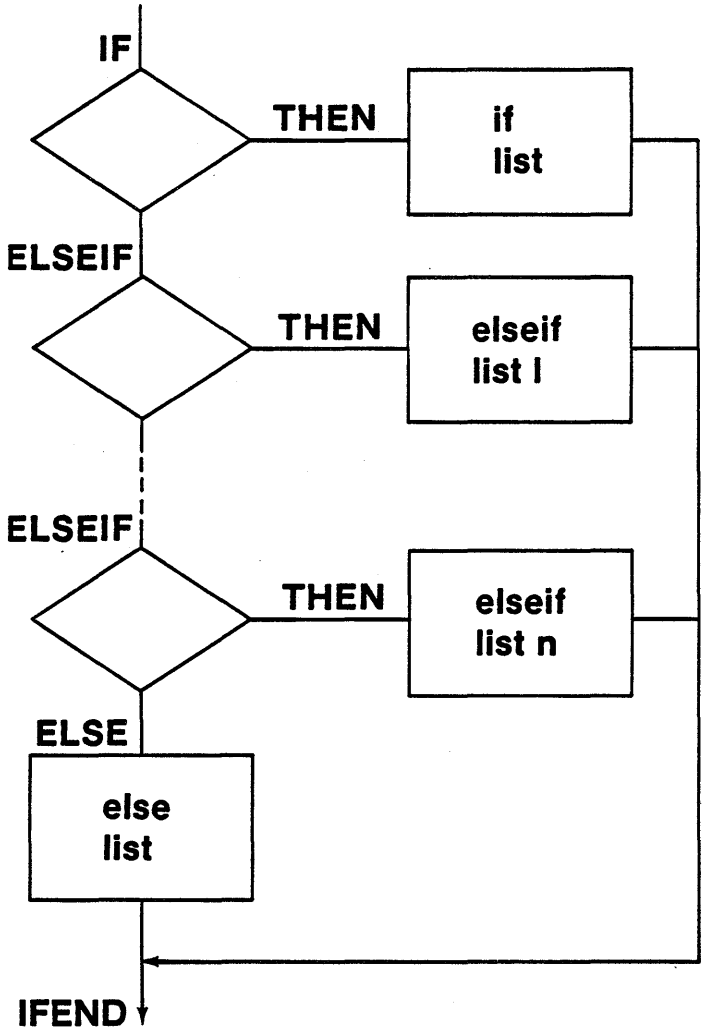
```
MODULE FOR_STATEMENT_3;

  PROGRAM MAIN;

    { ORDINAL COUNTING }

    TYPE
      GRAIN = (WHEAT, RYE, OATS, BARLEY);

    VAR
      W: GRAIN;
      I: INTEGER;
      A: ARRAY[GRAIN] OF INTEGER;
      V: ARRAY[0 .. 3] OF INTEGER;

    I := 0;
    FOR W := WHEAT TO BARLEY DO
      A[W] := A[W] + V[I];
      I := I + 1;
    FOREND;

  PROCEND MAIN;
MODEND FOR_STATEMENT_3.
```

NOTE: A and V have not been initialized.

CONTROL STATEMENTS

- IF
- CASE
- CYCLE
- EXIT
- Procedure Call
- RETURN

IF STATEMENT

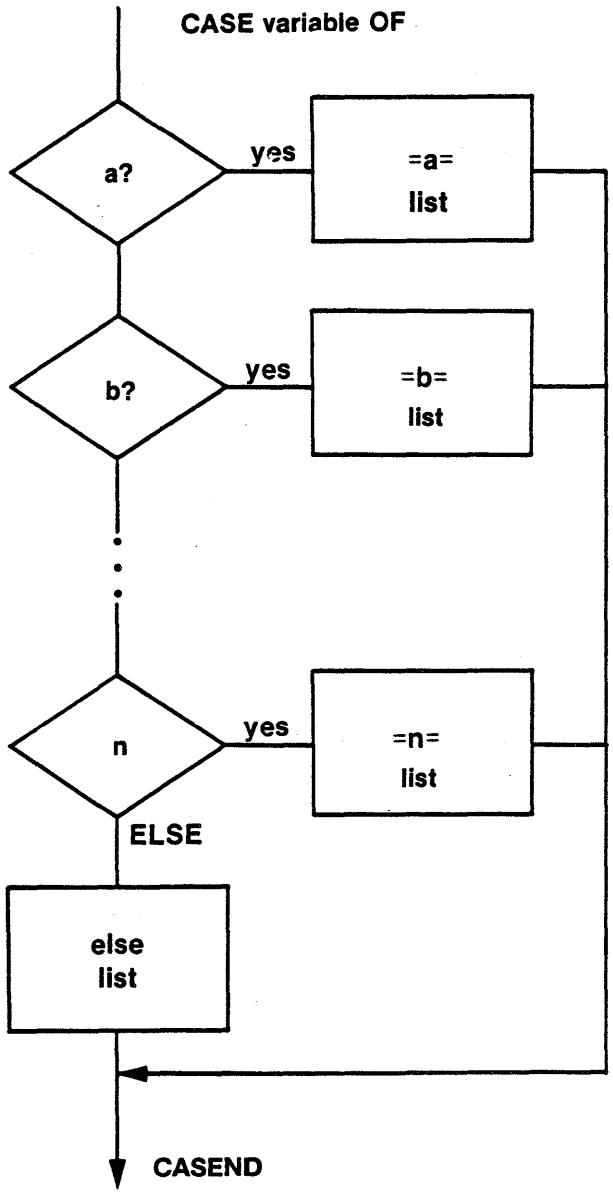


IF

```
MODULE IF_STATEMENT;

  PROGRAM MAIN;

    VAR
      A:
      B: INTEGER;

    IF A < B THEN
      A := B;
    IFEND;

    IF A <= 5 THEN
      B := 1;
    ELSEIF A > 30 THEN
      B := 2;
    ELSE
      B := 0;
    IFEND;
  PROCEND MAIN;
MODEND IF_STATEMENT;
```

NOTE: A AND B have not been initialized.

CASE STATEMENT



CASE (integer)

```
MODULE CASE_STATEMENT;

  PROGRAM MAIN;

    VAR
      X: 0 .. 5,
      Y: INTEGER;

    X := 3;
    CASE X OF
    =0=
      Y := 99;
    =1=
      Y := X;
    =2=
      Y := X + X;
    =3=
      Y := X * X;
    =4=
      Y := 2 * X + 3;
    =5=
      Y := X * X + 2 * X + 9;
      X := 0;
    CASEND;

  PROCEND MAIN;
MODEND CASE_STATEMENT;
```

CASE (ordinal)

```
MODULE CASE_STATEMENT;

  PROGRAM MAIN;

    TYPE
      COLOR = (RED, YEL, BLU, GRN);

    VAR
      N: INTEGER,
      Q: COLOR;

    Q := BLU;
    CASE Q OF
    =RED=
      N := 99;
    =GRN=
      N := 66;
    =BLU=
      N := - 99;

    CASEND;

  PROCEND MAIN;
MODEND CASE_STATEMENT;
```

CASE (subrange)

```
MODULE CASE_STATEMENT;

  PROGRAM MAIN;

    VAR
      Z: INTEGER;
      I: INTEGER;

    Z := 0;
    CASE Z OF
    =1 .. 10=
      I := 99;
    =11 .. 100=
      I := Z;
    = - 100 .. - 1=
      I := Z * Z;
    ELSE
      I := 2 * Z + 3;
    CASEND;

  PROCEND MAIN;
MODEND CASE_STATEMENT;
```

CYCLE STATEMENT

```
/label/
    start repetitive statement
    ----------CYCLE/label/
    end repetitive statement
```

- USED WITHIN REPETITIVE STATEMENT:

  FOR

  REPEAT

  WHILE

- CYCLE TO NEXT ITERATION, IF ANY

- CYCLE STATEMENT MUST BE IN SCOPE OF /label/

CYCLE

```
MODULE CYCLE_STATEMENT;

  PROGRAM MAIN;

    VAR
      I: INTEGER;
      J: [STATIC] INTEGER := 1000;
      A: ARRAY[1 .. 10] OF INTEGER;

  /LOOP1/
    FOR I := 1 TO 10 DO
      J := J + 1;
      IF I = 3 THEN
        CYCLE /LOOP1/ ;
      IFEND;
      A[I] := A[I] + J;
    FOREND /LOOP1/;

  PROCEND MAIN;
MODEND CYCLE_STATEMENT.
```

NOTE: A has not been initialized.

EXIT STATEMENT

EXIT /label/

- Used within a structured statement
  BEGIN
  FOR
  REPEAT
  WHILE

- Exits to following statement

- Exit statement must be in the scope of the /label/

EXIT name

- Used to exit a statically encompassing
  PROCEDURE
  FUNCTION

- Exits to the successor of the named procedure or function

EXIT

```
MODULE EXIT_STATEMENT_1;

  PROGRAM MAIN;

    VAR
      I,
      KEY: INTEGER;
      A: ARRAY[1 .. 10] OF INTEGER;
      KEY_FOUND: BOOLEAN;

    KEY := 7;
    FOR I := 1 TO 10 DO
      A[I] := KEY;
      KEY := KEY - 1;
    FOREND;

    I := 1;
    KEY := 4;

  /L200/
    REPEAT
      IF KEY = A[I] THEN
        EXIT /L200/ ;
      IFEND;
      I := I + 1;
    UNTIL I >  10;

    IF I <= 10 THEN
      KEY_FOUND := TRUE;
    ELSE
      KEY_FOUND := FALSE;
    IFEND;

  PROCEND MAIN;
MODEND EXIT_STATEMENT_1;
```

6

PROCEDURES

PROCEDURES

- **DEFINITION**
  PROGRAM name (params);
  PROCEDURE name (params);
  FUNCTION name (params):result_type;

- **CALL**
  Name (actual parameters);

- **PARAMETERS**
  Read-Only
  Read-Write (VAR)

- **ATTRIBUTES**
  PROCEDURE [attribute] name (params);

- **POINTER TO PROCEDURE**

GLOBAL VARIABLES

```
MODULE GLOBAL_PARAMS;

     VAR  I,J,K:INTEGER;

     PROCEDURE ADD;
       K := I + J;
     PROCEND ADD;

     PROCEDURE SUB;
       K := I - J;
     PROCEND SUB;

     PROGRAM MAIN;
       I := 6;
       J := 7;
       IF I<J THEN ADD
              ELSE SUB
       IFEND;
     PROCEND MAIN;

MODEND GLOBAL_PARAMS.
```

PASSED PARAMETERS

```
MODULE PASSED_PARAMETERS;
  PROCEDURE ADD ( VAR L:INTEGER; M,N:INTEGER );
    L := M + N;
  PROCEND ADD;

  PROCEDURE SUB (S,T:INTEGER; VAR R:INTEGER );
    R := S - T;
  PROCEND SUB;

  PROGRAM CALLER;
    VAR I,J,K:INTEGER;
    I := 20;
    J := 13;
    IF I<J THEN ADD( K,I,J );
           ELSE SUB( I,J,K );
    IFEND;
  PROCEND CALLER;
MODEND PASSED_PARAMETERS
```

PROCEDURE PARAMETERS

	COMMUNICATION	PROTECTION	CALL EFFICIENCY	REFERENCE EFFICIENCY
CALL BY reference	GOOD	POOR	AVERAGE	GOOD
CALL BY value	POOR	GOOD	AVERAGE	GOOD
GLOBAL VARIABLE	BEST	WORST	BEST	BEST

PARAMETERS

```
  0      1 MODULE PARAMETERS;
  0      2
  0      3   PROCEDURE COUNT (VAR N_OF_CHAR: 0 .. 80;
  0      4         STRING_TO_SEARCH: STRING (80),
  0      5         FIRST_CH: 1 .. 80,
  0      6         SEARCH_LENGTH: 1 .. 80,
  0      7         SEARCH_CH: CHAR);
  0      8
 11      9     VAR
 11     10       I: 1 .. 80;
 11     11
 11     12     N_OF_CHAR := 0;
 25     13     FOR I := FIRST_CH TO FIRST_CH + SEARCH_LENGTH - 1 DO

 33     14       IF STRING_TO_SEARCH (I) <> SEARCH_CH THEN
 46     15         N_OF_CHAR := N_OF_CHAR + 1;
 52     16       ELSE
 54     17         RETURN;
 55     18       IFEND;
 55     19     FOREND;
 55     20
 57     21   PROCEND COUNT;
 57     22
 57     23   PROGRAM MAIN;
 57     24
 61     25     VAR
 61     26       CARD: STRING (80),
 61     27       I,
 61     28       J: 1 .. 80,
 61     29       K: 0 .. 80,
 61     30       CH: CHAR;
 61     31
 61     32 { ASSUME THAT THE CARD (80 CHARACTERS) }
 61     33 { HAS BEEN READ INTO THE VARIABLE CARD }
 61     34
 61     35     COUNT (K, CARD, 1, 80, '+');
 61     36
 61     37
 61     38
 72     39     I := 14;
 73     40     J := 3;
 74     41     CH := ' ';
 75     42     COUNT (K, CARD, I, J, CH);
101     43   PROCEND MAIN;
101     44 MODEND PARAMETERS;
```

FUNCTIONS

STRUCTURE:

```
FUNCTION name (params): result_type;
   {declarations}
   {statements}
   name:=
FUNCEND name;
```

SIDE EFFECTS:

- Illegal Assignments
  Nonlocal variables
  Read/write Parameters
  Pointer Variables

- Illegal Contents
  User_defined procedure calls
  Nonlocal variables to standard procs
  Procedures as parameters

FUNCTION CALL

```
MODULE USE_FUNCTION;

  FUNCTION sum (x,
    y: INTEGER): INTEGER;
    sum := x + y;
  FUNCEND sum;

  PROGRAM CALLER;

    VAR
      a,
      b: INTEGER;

    a := 10;
    b := - 3;
    IF sum (a, b) >= 0 THEN
      a := - a;
      b := - b;
    IFEND;
  PROCEND CALLER;
MODEND USE_FUNCTION;
```

NESTED PROCEDURES

```
MODULE NESTED_PROCS;
  PROCEDURE COMPUTE;                    --+
    VAR                                   |
                                          |
    PROCEDURE TEST;          --+          |
      VAR                      |          |
      { STATEMENTS }           |          |
    PROCEND TEST;            --+          |
                                          |
    { STATEMENTS }                        |
  PROCEND COMPUTE;                      --+

  PROGRAM MAIN;                         --+
    { DECLARATIONS }                      |
    { STATEMENTS }                        |
                                          |
    COMPUTE;                              |
                                          |
    { STATEMENTS }                        |
  PROCEND MAIN;                         --+
MODEND NESTED_PROCS;
```

RECURSIVE FUNCTION

```
MODULE RECURSIVE_FUNCTION;

  FUNCTION FACT (m: INTEGER): INTEGER;
{ RECURSIVE FACTORIAL }
{ N! = N*(N-1)! }

    IF m > 1 THEN
      FACT := m * FACT (m - 1);
    ELSE
      FACT := 1;
    IFEND;
  FUNCEND FACT;

  PROGRAM MAIN;

    VAR
      N:
      VALUE: INTEGER;

    N := 4;
    IF N >= 0 THEN
      VALUE := FACT (N);
    IFEND;
  PROCEND MAIN;
MODEND RECURSIVE_FUNCTION;
```

RECURSIVE PROCEDURE

```
MODULE RECURSIVE_PROC;

  PROCEDURE ITOS { INTEGER TO STRING }
    (N: INTEGER;
    VAR S: STRING (20);
    VAR I: 0 .. 20);

    VAR
      M: INTEGER;

    I := 0;
    IF N >= 0 THEN
      M := N;
    ELSE
      S (1) := '-';
      M := - N;
      I := 1;
    IFEND;
    IF M >= 10 THEN
      ITOS (M DIV 10, S, I);
    IFEND;
    I := I + 1;
    S (I) := $CHAR (M MOD 10 + $INTEGER ('0'));
  PROCEND ITOS;

  PROGRAM CALLER;


    VAR
      N_CHAR: 0 .. 20;
      STR: STRING (20);
      NUM: INTEGER;

    NUM := 250;
    ITOS (NUM, STR, N_CHAR);
  PROCEND CALLER;
MODEND RECURSIVE_PROC;
```

PROCEDURE ATTRIBUTES

```
MODULE FIRST;

  PROCEDURE [XREF] COMPUTE;

  PROGRAM MAIN;
     .
     .
     COMPUTE;
     .
     .
  PROCEND MAIN;

MODEND FIRST;
```

```
MODULE SECOND;

  PROCEDURE [XDCL] COMPUTE;
     .
     .
  PROCEND COMPUTE;

MODEND SECOND;
```

PROCEDURE POINTERS

- ALLOWS CALLS TO PROCEDURES BY POINTER (proc_ptr ∧;)
- CAN BE PASSED AS PARAMETERS
- POINTS TO OUTER LEVEL PROCEDURES ONLY
- INITIALIZATION
  WITHIN PROGRAM (∧ name)
  OPERATING SYSTEM CALLS (future)

POINTERS TO PROCEDURES

```
MODULE POINTER_TO_PROCEDURE;

  PROCEDURE [XREF] ERROR (P: ^PROCEDURE);

  PROCEDURE [XREF] A;

  PROCEDURE [XREF] B;

  PROCEDURE [XREF] C;


  PROGRAM M;

    TYPE
      P = ARRAY[1 .. 3] OF ^PROCEDURE;

    VAR
      I: 1 .. 3,
      JUMP_TABLE: P;

    { INITIALIZE TABLE }

    JUMP_TABLE[1] := ^A;
    JUMP_TABLE[2] := ^B;
    JUMP_TABLE[3] := ^C;

    { REFERENCE THE TABLE }

    I := 2;
    JUMP_TABLE[I]^;
    ERROR (JUMP_TABLE[I]);

  PROCEND M;
MODEND
```

POINTER TO PROCEDURE

```
MODULE SHOW;

  VAR

    PROC_PTR: ^PROCEDURE (A,
    B: INTEGER);

  PROCEDURE [XREF] TEST1 (XVAL,
        YVAL: INTEGER);

  PROCEDURE [XREF] TEST2 (PTR: ^PROCEDURE (C, D: INTEGER),
        M,
        N: INTEGER);


  PROGRAM MAIN;

    VAR

      MN1,
      MN2: INTEGER;


    MN1 := 50;
    MN2 := 75;

     .
     .
     .

    PROC_PTR := ^TEST1;

     .
     .
     .

    PROC_PTR^ (MN1, MN2);

    TEST2 (^TEST1, MN1, MN2)

  PROCEND MAIN;

MODEND SHOW;
```

7

STANDARD PROCEDURES

CONVERSION

SUCC (X)	Returns the successor of X
PRED (X)	Returns the predecessor of X
$CHAR (X)	Returns the character represented by integer X
$REAL (X)	Returns real representation of the integer or longreal X.
$LONGREAL (X)	Returns longreal representation of the integer or real X.
$INTEGER (X)	Returns the integer representation of a scalar, real or longreal X.

ORDINAL CONVERSION

```
MODULE ORD_CONVERSION;

  PROGRAM MAIN;

    { ORDINAL COUNTING }

    TYPE
      GRAIN = (WHEAT, RYE, OATS, BARLEY);

    VAR
      W: GRAIN;
      I: INTEGER;
      A: ARRAY[GRAIN] OF INTEGER;
      V: ARRAY[0 .. 3] OF INTEGER;

    FOR W := WHEAT TO BARLEY DO
       A[W] := A[W] + V[$INTEGER (W)];
    FOREND;

  PROCEND MAIN;
MODEND ORD_CONVERSION.
```

Note: A & V have not been initialized

STRINGREP

STRINGREP(s, l, p_1, p_2, ..., p_n)

s = result string

l = result length

p_i = concatenation element

TYPE	RESULT	Pi SPECIFICATION	DEFAULT	JUSTIFY
integer	~~b~~ nn...(r)	exp:length:#(radix)	val:n+1:#(10)	R
ordinal	~~b~~ nn...(r)	exp:length:#(radix)	val:n+1:#(10)	R
subrange	~~b~~ nn...(r)	exp:length:#(radix)	val:n+1:#(10)	R
character	c	exp:length	val:1	L
boolean	~~b~~ TRUE,FALSE	exp:length	val:5	L
string	cc...	exp:length	val:n	L
pointer	①	exp:length	val:n:#(r) ①	R
~~real~~ FLOATING PT	~~b~~ n.nn...E±nnn	exp:length	val:n+4 ①	R
~~real~~ FLOATING PT DOUBLE	~~b~~ n.nn...D±nnn	exp:length:~~fraction~~	val:n[redacted] +4	R
FIXED POINT	~~b~~ n.nn	exp: len : FRACT	NONE	R

① machine dependent

STRINGREP EXAMPLE-1

SOURCE

```
MODULE SEE;
?? SET (LIST := OFF) ??
*CALLC PXIOTYP
*CALLC LGZOPEN
*CALLC LGZCLOS
*CALLC LGZPUT
*CALLC LGZGET
?? RESET ??

  TYPE
    COLOR = (RED, ORANGE, YELLOW, GREEN, BLUE),
    A_M_ALPHA = 'A' .. 'M',
    M_ARRAY = ARRAY [1 .. 10] OF STRING (10);

  VAR
    OUT,
    INN: FILE,
    LENGTH: INTEGER,
    INITIAL: INTEGER := 1,
    FINAL: INTEGER := 10,
    CONTROL_LOOP: INTEGER := 1,
    INT_VAL: INTEGER,
    HEX_VAL: INTEGER := - 0A3(16),
    FLOAT_PT_VAL: REAL,
    FIXED_PT_VAL: REAL,
    MESSAGE: M_ARRAY,
    BIG_MESSAGE: STRING (80),
    CHAR_VAL: CHAR := 'T',
    SUBRANGE_VAL: A_M_ALPHA := 'D',
    BOOL_VAL: BOOLEAN := TRUE,
    COLOR_VAL: COLOR;

  PROGRAM MAIN;

    LG#OPEN (OUT, 'OUTPUT', OLD#, OUTPUT#, ASIS#);
    LG#PUT (OUT, 'START OF JOB');

    FLOAT_PT_VAL := 123.456;
    FIXED_PT_VAL := - 12.3456;
    INT_VAL := 25;
    COLOR_VAL := ORANGE;

{ CONVERT TO STRING REPRESENTATION }

    STRINGREP (MESSAGE [1], LENGTH, INT_VAL);
    STRINGREP (MESSAGE [2], LENGTH, INT_VAL: 10: #(16));
    STRINGREP (MESSAGE [3], LENGTH, HEX_VAL);
    STRINGREP (MESSAGE [4], LENGTH, HEX_VAL: 10: #(16));
    STRINGREP (MESSAGE [5], LENGTH, CHAR_VAL);
    STRINGREP (MESSAGE [6], LENGTH, SUBRANGE_VAL);
    STRINGREP (MESSAGE [7], LENGTH, BOOL_VAL);
    STRINGREP (MESSAGE [8], LENGTH, FLOAT_PT_VAL: 10);
    STRINGREP (MESSAGE [9], LENGTH, FIXED_PT_VAL: 8: 3);
    STRINGREP (MESSAGE [10], LENGTH, COLOR_VAL);
```

STRINGREP EXAMPLE-2

```
{ OUTPUT THE STRINGS SEPARATELY }

    LG#PUT (OUT, 'STRINGS OUTPUTTED SEPARATELY');

    FOR CONTROL_LOOP := INITIAL TO FINAL DO
      LG#PUT (OUT, MESSAGE [CONTROL_LOOP]);
    FOREND;

{ CONVERT AND CONCATENATE }

    STRINGREP (BIG_MESSAGE, LENGTH, INT_VAL, INT_VAL: 10: #(16), HEX_VAL
,
            HEX_VAL: 10: #(16), CHAR_VAL, SUBRANGE_VAL, BOOL_VAL, FLOAT_PT
_VAL:
            10, FIXED_PT_VAL: 8: 3, COLOR_VAL);

{ OUTPUT THE CONCATENATED STRING }

    LG#PUT (OUT, 'CONCATENATED STRING OUTPUTTED');

    LG#PUT (OUT, BIG_MESSAGE);

    LG#PUT (OUT, 'END OF JOB');
    LG#CLOSE (OUT, ASIS#);
  PROCEND MAIN;
MODEND SEE.
```

OUTPUT

```
/SES.GENCOMP SF=TEST4 CYBCOMN
◆    GENERATING COMPILE FILE  COMPILE
◆    END GENCOMP    COMPILE <- BASE
/SES.CYBIL CC
◆    COMPILING  COMPILE
◆    END CYBIL    COMPILE -> LISTING, LGO
/SES.LINK170 CYBCLIB
◆    END LINK170    LGOB
/LGOB
START OF JOB
STRINGS OUTPUTTED SEPARATELY
 25
        19
-163
       -A3
T
 68
 TRUE
  1.2E+002
 -12.346
 1
CONCATENATED STRING OUTPUTTED
 25        19-163       -A3T 68 TRUE  1.2E+002 -12.346 1
END OF JOB
LGOB.
```

DETERMINE LIMITS

UPPERVALUE (X)

Returns the largest value of a scalar variable or type

LOWERVALUE (X)

Returns the smallest value of a scalar variable or type

EXAMPLE:

```
FOR w := UPPERVALUE (GRAIN) DOWNTO LOWERVALUE (GRAIN) DO
.
.
.
FOREND;
```

ADAPTABLE TYPES

- Size is unknown at Compile time
- Size is determined at execution time
- Used as parameters
  STRING (*)
  ARRAY [*] OF type
  RECORD
- Allocated
  STRING
  ARRAY
  RECORD
  BOUND VARIANT RECORD
  HEAP
  SEQUENCE

SIZE DETERMINATION FUNCTIONS

- **STRLENGTH (X)**

  Returns the length of the string X

- **UPPERBOUND (X)**

  Returns the highest value of an array index

- **LOWERBOUND (X)**

  Returns the lowest value of an array index

ADAPTABLE ARRAY

```
MODULE ex1;

  TYPE

    adaptarray = array [*] OF integer;

  PROCEDURE square_it (VAR data : adaptarray);

    VAR

      index : integer;


    FOR index := LOWERBOUND(data) TO UPPERBOUND(data) DO

      data[index] := data[index] * data[index];

    FOREND;

  PROCEND square_it;

  PROGRAM main;

    VAR

      vector1 : ARRAY[-25 .. 50] OF integer,
      vector2 : ARRAY[1 ..100] OF integer;


    {  FILL ARRAYS WITH VALUES  }


      square_it(vector1);


      square_it(vector2);


  PROCEND main;

MODEND ex1;
```

ADAPTABLE RECORD

```
TYPE
     AREC = RECORD
          X = INTEGER
          Y = STRING(*)
     RECEND;

PROCEDURE BLANKCOUNT(INREC : AREC; VAR NBLANKS : INTEGER);
     VAR POSITION : INTEGER;
     NBLANKS := 0;
     FOR POSITION := 1 TO STRLENGTH(INREC.Y) DO
          IF INREC.Y(POSITION) = " " THEN
               NBLANKS := NBLANKS + 1;
          IFEND;
     FOREND;
PROCEND BLANKCOUNT;

PROGRAM CALLER;
     VAR
          R : RECORD
               S : INTEGER
               T : STRING(10)
               RECEND.
          COUNT : INTEGER;
     R.T := "123 56 890";
     BLANKCOUNT(R,COUNT);
PROCEND CALLER;
```

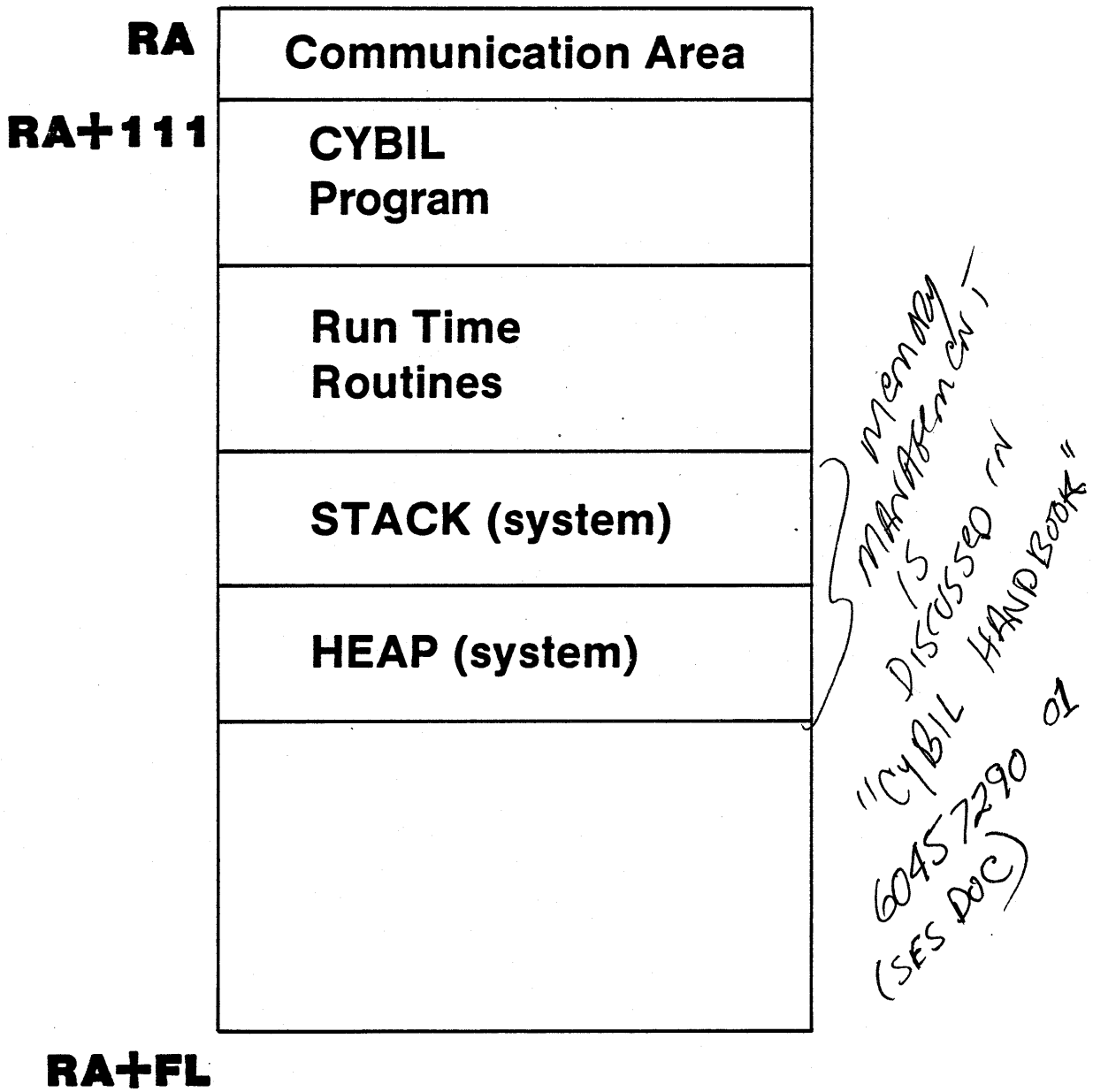
ADAPTABLE STRING

```
MODULE ADAPTABLE_STRING;

  TYPE
    ASTRING = STRING ( * );

  PROCEDURE BLANK_COUNT (IN_STR: ASTRING;
    VAR NBLANKS: INTEGER);

    VAR
      POSITION: INTEGER;

    NBLANKS := 0;
    FOR POSITION := 1 TO STRLENGTH (IN_STR) DO
      IF IN_STR (POSITION) = ' ' THEN
        NBLANKS := NBLANKS + 1;
      IFEND;
    FOREND;

  PROCEND BLANK_COUNT;

  PROGRAM CALLER;

    VAR
      S: STRING (10),
      L: INTEGER;

    S := 'AB CD EF G';
    BLANK_COUNT (S, L);

  PROCEND CALLER;

MODEND ADAPTABLE_STRING;
```

STORAGE ALLOCATION

- STACK (system)
- HEAP (system & user)
- SEQUENCE (user)
- ADAPTABLE STORAGE
  HEAP
  SEQUENCE
  ARRAY
  STRING
  RECORD
  BOUND VARIANT RECORD

MEMORY ORGANIZATION
(CYBER 170)



STORAGE ALLOCATION
(SYSTEM HEAP)

ALLOCATE P;	Allocates space for T; returns pointer P.
ALLOCATE P:[length];	Allocates space for an adaptable string; returns pointer.
ALLOCATE P:[low..high];	Allocates space for an adaptable array; returns pointer.
ALLOCATE P:[t_1,...,t_n];	Allocates space for a bound variant record; returns pointer.
FREE P;	Frees the space pointed to by P; sets P to nil.

NOTE: P is a pointer to type T.
If space cannot be allocated, P:=NIL.

LINKED LIST

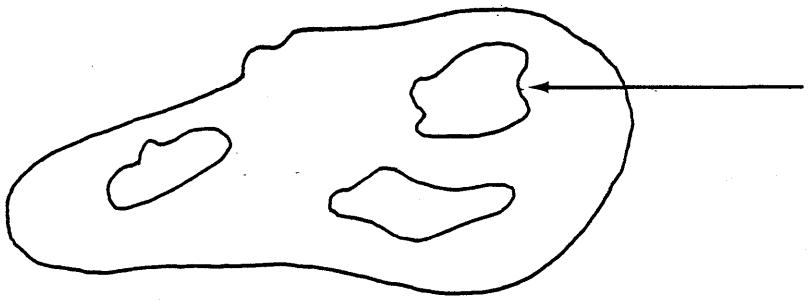
```
MODULE LINK_LIST;

  TYPE
    LINK = ^REC;
    REC = RECORD
      VALUE: INTEGER;
      PTR: LINK
    RECEND;

  VAR
    FIRST: LINK := NIL;

  PROGRAM MAIN;

    VAR
      I: 1 .. 10;
      POINTER: LINK;

    FOR I := 1 TO 10 DO
      ADD (POINTER);
      POINTER^.VALUE := I;
    FOREND;
  PROCEND MAIN;

  PROCEDURE ADD (VAR P: LINK);

    PROCEDURE [XREF] ERROR;

    { ADD A RECORD TO THE BEGINNING }
    { OF THE LIST }

    ALLOCATE P;
    IF P <> NIL THEN
      P^.PTR := FIRST;
      FIRST := P;
    ELSE
      ERROR;
    IFEND;
  PROCEND ADD;

MODEND;
```

BOUND VARIANT ALLOCATION

```
 1
 2 MODULE BOUND_VARIANT_RECORD;
 3
 4   PROGRAM ALLOCATE_VARIANT;
 5
 6     TYPE
 7       PART = BOUND RECORD
 8         NAME: STRING (20),
 9         NUMBER: 1 .. 1000,
10         VENDOR: 1 .. 100,
11         CASE T: 1 .. 2 OF
12         =1= (STOCKED PART)
13           QUANTITY: 1 .. 4000,
14           LOCATION: 1 .. 10,
15         =2= (UNSTOCKED PART)
16           OBSOLETE: BOOLEAN,
17         CASEND
18       RECEND;
19
20     VAR
21       P1,
22       P2: ^PART;
23
24     PROCEDURE [XREF] ORDER (P: ^PART);
25
26   /L10/
27     BEGIN
28       ALLOCATE P1: [1];
29       P1^.NAME := 'UNDERWATER CORKSCREW';
30       P1^.NUMBER := 481;
31       P1^.QUANTITY := 3998;
32
33       ALLOCATE P2: [2];
34       P2^.NAME := 'ZITHER G-STRING      ';
35       P2^.VENDOR := 3;
36
37       IF P2^.OBSOLETE THEN
38         EXIT /L10/
39       ELSE
40         ORDER (P2)
41       IFEND;
42
43     END;
44   PROCEND;
45 MODEND;
```

STORAGE ALLOCATION
(USER HEAP)

TYPE heaptypid = HEAP (REP exp OF type);



- STATIC OR IN STACK
- CONTROL

  RESET heap_variable;

  ALLOCATE ptr_variable IN heap_variable;

  FREE ptr_variable IN heap_variable;
- ADAPTABLE

  TYPE id = HEAP (*)
- SPACE FOR POINTER IS NOT ALLOCATED

USER HEAP

```
MODULE USER_HEAP;

  PROCEDURE [XDCL] SAVE (CARD: STRING (80));
    VAR N: 1 .. 80,
      P: ^STRING ( * ));

    TYPE
      CIH = HEAP (REP 100 OF STRING (80));

    VAR
      COMPRESSED_IMAGES: [XREF] CIH;

    { SAVE THE COMPRESSED STRING (CARD) }
    { IN THE HEAP }

    WHILE CARD (N) = ' ' DO
      N := N - 1;
    WHILEND;
    ALLOCATE P: [N] IN COMPRESSED_IMAGES;
    IF P <> NIL THEN
      P^ := CARD (1, N);
    IFEND;
  PROCEND;
MODEND
```

SUBSTRUCTURE POINTERS

variable := ∧ allocated_ptr ∧ [n];

variable := ∧ allocated_ptr ∧.field;

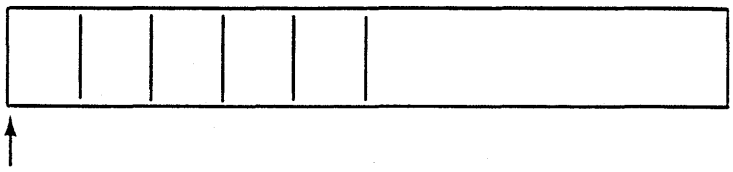
- POINT TO ANY COMPONENT OF A DYNAMICALLY ALLOCATED ARRAY OR RECORD
- POINTER TYPE CHECKING IS PRESERVED
- CAUTION—POINTER LIFETIME CAN EXCEED DATA LIFETIME

SUBSTRUCTURE POINTERS

```
MODULE SUBSTRUCTURE_POINTER;

  TYPE
    AAA = ARRAY[1 .. 9] OF INTEGER;

  PROCEDURE SQUARE_IT (P: ^INTEGER);
    P^ := P^ * P^;
  PROCEND;

  PROGRAM M;

    VAR
      Q: ^AAA;
      QI: ^INTEGER;

    ALLOCATE Q;
    Q^[1] := 3;
    SQUARE_IT (^Q^[1]);
  PROCEND M;
MODEND
```

STORAGE ALLOCATION
(SEQUENCES)

TYPE seqtypid = SEQ (REP exp OF type);

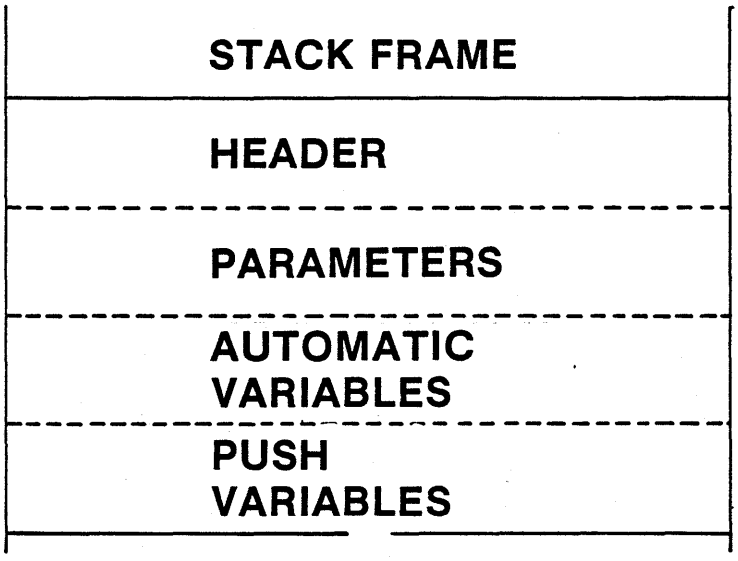


- STATIC OR IN STACK
- CONTROL
  RESET ptr_to_seq_variable;
  NEXT ptr_variable IN ptr_to_seq_variable;
  RESET ptr_to_seq_variable TO ptr_variable
- CAN BE USED TO AVOID TYPE CHECKING
- ADAPTABLE
  TYPE id = SEQ(*)
- SPACE FOR POINTER IS ALLOCATED

SEQUENCE

```
MODULE sequences;

  TYPE

    aseq = SEQ (REP 10 OF integer);

  VAR

    seqstorage : aseq,
    pseq : ^aseq,
    p : ^integer,
    value : 0 .. 10 := 0;

  PROGRAM s;

    {  PLACE VALUES IN SEQSTORAGE  }


    pseq := ^seqstorage;
    RESET pseq;
    NEXT p IN pseq;
    WHILE p <> NIL DO
      value := value + 1;
      p^ := value;
      NEXT p IN pseq;
    WHILEND;

    {  RETRIEVE VALUES FROM SEQSTORAGE  }

    RESET pseq;
    NEXT p IN pseq;
    WHILE p <> NIL DO

      {  PROCESS VALUE  }
      {  P^ REFERS TO INTEGER VALUE  }

      NEXT p IN pseq;

    WHILEND;

  PROCEND s;

MODEND sequences;
```

STORAGE ALLOCATION
(PUSH)



PUSH ptr;

- SPACE ALLOCATED AT END OF STACK FRAME. "UNAUTOMATIC" VARIABLE.
- SPACE IS FREED WHEN THE PROCEDURE RETURNS.

PUSH

```
MODULE PUSHY;

  PROGRAM MAIN;

    PROCEDURE A;

      PROCEDURE X (Q: ^INTEGER);
      PROCEND X;

      VAR
        P: ^INTEGER;

      PUSH P;
      IF P <> NIL THEN
        P^ := 482;
        X (P);
      ELSE
        RETURN;
      IFEND

    PROCEND A;
  PROCEND MAIN;
MODEND
```

ADAPTABLE RECORD

```
MODULE ALLOCATE_ADAPTABLE_RECORD;

  TYPE
    REC = RECORD
      KEY: 1 .. 1000,
      NAM: STRING ( * ),
    RECEND;

  VAR
    POINTER_TABLE: ARRAY[1 .. 1000] OF ^REC;

  PROCEDURE [XDCL] SAVE_STRING (S: STRING ( * ),
    K: 1 .. 1000);

    VAR
      P: ^REC;

    ALLOCATE P: [STRLENGTH (S)];
    P^.KEY := K;
    P^.NAM := S;
    POINTER_TABLE[K] := P;
  PROCEND SAVE_STRING;
MODEND
```

ALLOCATING ADAPTABLE TYPES

```
MODULE FIXERS;

  PROGRAM A;

    VAR
      P_A_ARRAY: ^ARRAY[ * ] OF INTEGER,
      P_A_STRING: ^STRING ( * ),
      P_A_HEAP: ^HEAP ( * ),
      P_A_SEQUENCE: ^SEQ ( * );

    VAR
      P_I: ^INTEGER,
      P_CH: ^CHAR;

    ALLOCATE P_A_ARRAY: [1 .. 20];
    P_A_ARRAY^[3] := - 3;

    ALLOCATE P_A_STRING: [10];
    P_A_STRING^ := '$$$$$$$$$$';

    ALLOCATE P_A_HEAP: [[REP 1000 OF INTEGER]];
    RESET P_A_HEAP;
    ALLOCATE P_I IN P_A_HEAP;
    P_I^ := 16;

    ALLOCATE P_A_SEQUENCE: [[REP 1000 OF CHAR]];
    RESET P_A_SEQUENCE;
    NEXT P_CH IN P_A_SEQUENCE;
    P_CH^ := 'X';

  PROCEND;
MODEND;
```

"HEAPS"

1. ALLOCATE - ALLOCATES SPACE IN "SYSTEM HEAP"

2. ALLOCATE IN - ALLOCATES SPACE IN "USER HEAP".

3. RESET - RELEASES ALL VARIABLES IN "USER HEAP". DOES NOT CHANGE POINTERS.

4. FREE - RELEASES A PARTICULAR VARIABLE IN "SYSTEM HEAP" OR "USER HEAP". SETS POINTER TO "NIL".

"SEQUENCE"

1. NEXT - ALLOCATES SPACE IN THE "SEQUENCE".

2. RESET - RETURNS "POINTER TO SEQUENCE" TO BOI. DOES NOT RELEASE VARIABLES.

3. RESET TO - RETURNS "POINTER TO SEQUENCE" TO AN INTERMEDIATE POSITION. DOES NOT RELEASE VARIABLES.

TO THINK ABOUT

1. WHETHER "USER HEAP" OR "SEQUENCE" GOES INTO "STACK" OR WITH "STATIC VARIABLES" DEPENDS ON IF "HEAP VARIABLE" OR "SEQUENCE VARIABLE" IS "STATIC" OR "AUTOMATIC".

2. A. WHEN YOU DEFINE "USER HEAP" OR "SEQUENCE" VARIABLE IS WHEN THE SPACE IS ACTUALLY CREATED FOR A "USER HEAP" OR "SEQUENCE".
   B. RESET THE "USER HEAP VARIABLE" OR "POINTER TO SEQUENCE VARIABLE" PRIOR TO FIRST ALLOCATE_IN_ OR NEXT.
   C. THIRD, THE ALLOCATE_IN_ AND NEXT GIVE THE ABILITY TO PUT STRUCTURES INTO "USER HEAP" AND "SEQUENCE".
   D. LAST, USER PUTS DATA INTO THESE STRUCTURES.

3. A. ALWAYS HAVE A "SYSTEM HEAP".
   B. ALLOCATE GIVES ABILITY TO PUT STRUCTURES IN "SYSTEM HEAP".
   C. LAST, USER PUTS DATA INTO THESE STRUCTURES.

4. IF YOU PUSH A VARIABLE OF TYPE "POINTER TO TYPE", ONE COPY (SPACE) OF THAT TYPE IS CREATED ON THE "SYSTEM STACK" AT THAT TIME.

8

MACHINE DEPENDENCIES

DATA REPRESENTATION

- SCALARS
- STRUCTURED DATA
- PACKED ARRAY/RECORDS
- POINTERS

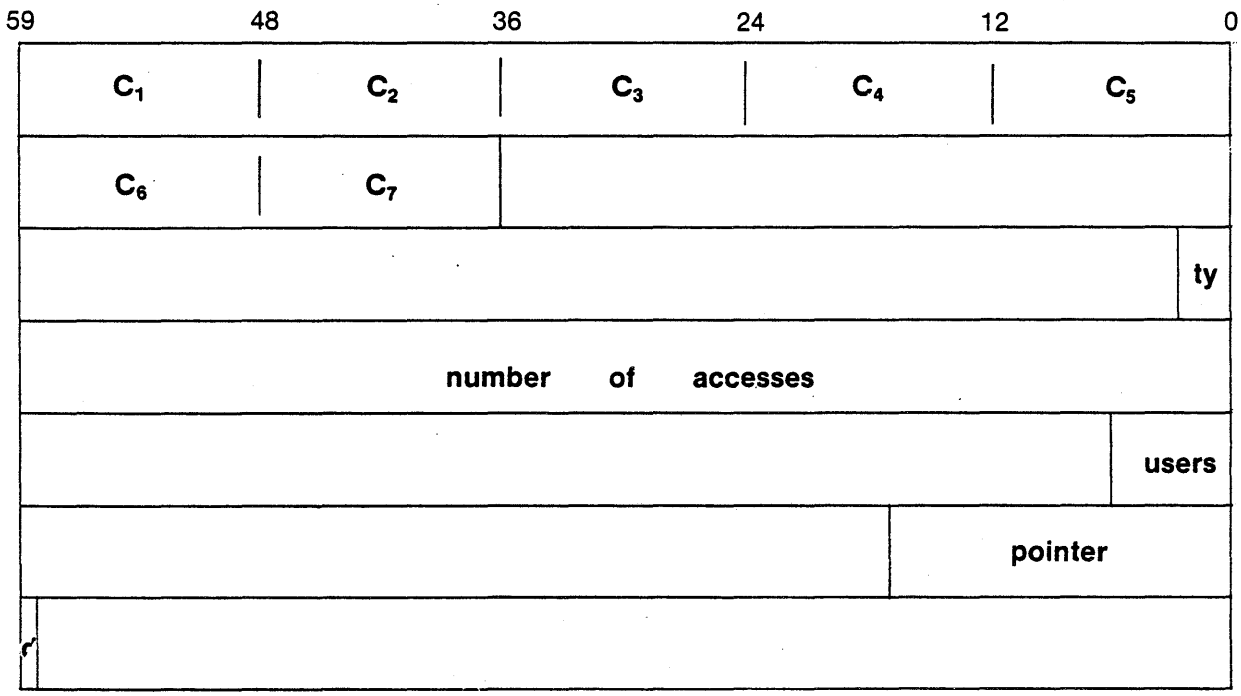
CY 170 DATA REPRESENTATION

TYPE	SIZE	ALIGNMENT	
		UNPACKED	PACKED
BOOLEAN	bit	LJ word	bit
INTEGER	word	word	word
SUBRANGE	as needed	RJ word	bit
ORDINAL	as needed	RJ word	bit
CHARACTER	12 bits/ 8 bits	RJ word	bit
REAL	word	word	word
LONGREAL	2 words	word	word
STRING	n* 12 bits	LJ word	12 bit
SET	as needed	LJ word	bit
ARRAY/RECORD	component dependent	word	unaligned components
FIXED POINTFR	18 bits	RJ word	bit

CY 170 DATA REPRESENTATION

UNPACKED

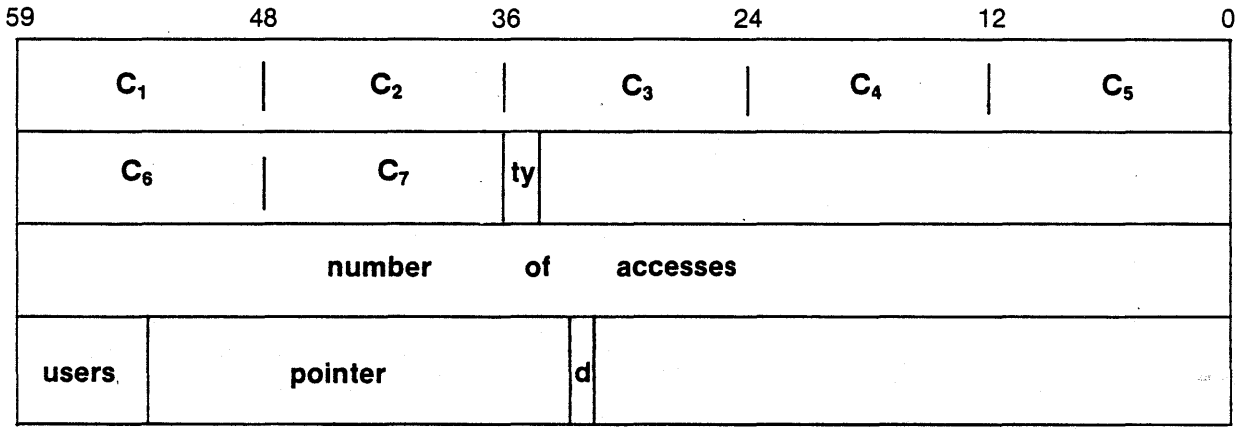
```
TYPE
  FT = RECORD     { FILE TABLE }
    FT_NAME: STRING(7),
    FT_TYPE: (BI,DI,LG,PR),
    FT_ACCESSES: INTEGER,
    FT_USERS: 0..100,
    FT_P_IOREQ: ^IOREQ,
    FT_DISK: BOOLEAN
  RECEND;
```



CY 170 DATA REPRESENTATION

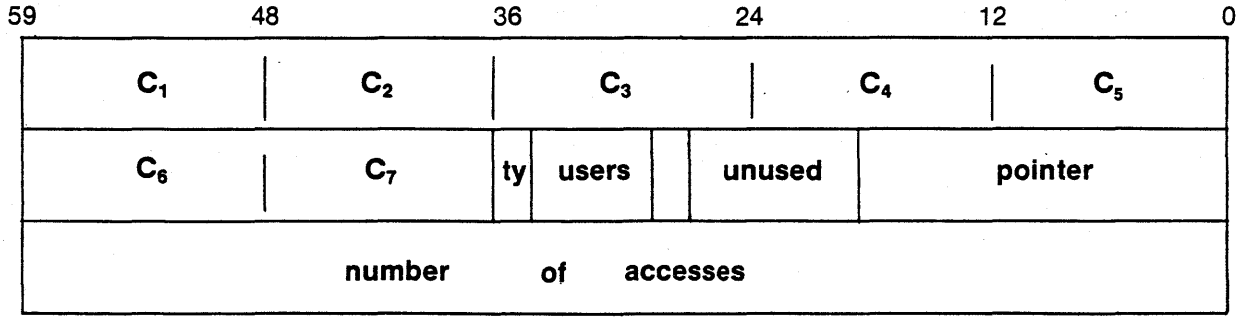
PACKED

```
TYPE
  FT = PACKED RECORD     { FILE TABLE }
    FT_NAME: STRING(7),
    FT_TYPE: (BI,DI,LG,PR),
    FT_ACCESSES: INTEGER,
    FT_USERS: 0..100,
    FT_P_IOREQ: ^IOREQ,
    FT_DISK: BOOLEAN
  RECEND;
```



BEST CASE

```
TYPE  { 170 BEST }
  FT = PACKED RECORD     { FILE TABLE }
    FT_NAME: STRING(7),
    FT_TYPE: (BI,DI,LG,PR),
    FT_USERS: 0..100,
    FT_DISK: BOOLEAN,
    FT_UNUSED: 0 .. 255,
    FT_P_IOREQ: ^IOREQ,
    FT_ACCESSES: INTEGER
  RECEND;
```



CELL

- CELL DATA TYPE
- POINTER TO CELL
- APPLICATIONS
  - — INPUT/OUTPUT PROCEDURES
  - — AVOID TYPE CHECKING

```
/COPF $USER.CELL_EXAMPLE
MODULE A;

  PROGRAM CELL_EXAMPLE;

    VAR
      PTR: ^CELL,
      INT: ^INTEGER,
      STRING_1: [STATIC] STRING (8) := 'MNOPQRST',
      CHARACTER_1: CHAR,
      C: CELL,
      I: INTEGER;

    PTR := ^STRING_1;
    CHARACTER_1 := PTR^ (4);
    PTR := ^STRING_1 (4);
    C := PTR^;
    CHARACTER_1 := PTR^;

    PTR := INT;
    INT := PTR; (ONLY IF PTR CONTAINS ADDR OF AN INTEGER)

    C := I;
    I := C;
    C := FALSE;
    C := CHARACTER_1;

  PROCEND CELL_EXAMPLE;
MODEND A;
```

ALIAS

(CY 170)

- ALLOWS "NON-CYBIL" IDENTIFIERS
- XDCL/XREF RELATED
- EXAMPLES

```
MODULE TEST ALIAS 'SYS.123';

  PROCEDURE SORT ALIAS 'SRTMRG9'  [XREF]
     ( VAR X: STRING(*));

  VAR
    DEVICE_ALLOCATION_TABLE ALIAS 'DAT':
      [XDCL] SET OF Ø..999;

  PROCEND SORT;

MODULE TEST
```

MACHINE-DEPENDENT FUNCTION

- **FUNCTION**

#SIZE (arg)	**Returns the size in cells of a variable or type**

#SIZE FUNCTION

```
 1
 2 MODULE VARIANT_RECORD_TYPE;
 3
 4   PROGRAM MAIN;
 5
 6     CONST
 7       MAXSIZE = 4095;
 8
 9     TYPE
10       SHAPE = (TRIANGLE, PARALLELOGRAM, SQUARE),
11       SIDE_RANGE = 0 .. MAXSIZE,
12       ANGLE = - 180 .. 180;
13
14     TYPE ( VARIANT RECORD )
15       FIGURE = BOUND RECORD
16         X,
17         Y: SIDE_RANGE,
18         CASE S: SHAPE OF
19         =TRIANGLE=
20           SIDE: SIDE_RANGE,
21           A1,
22           A2: ANGLE,
23         =PARALLELOGRAM=
24           SKEW: ANGLE,
25           S1,
26           S2: SIDE_RANGE,
27         =SQUARE=
28           EDGE: SIDE_RANGE,
29         CASEND
30       RECEND;
31
32     VAR
33       S1,
34       S2: INTEGER,
35       Q_P,
36       R_P: ^FIGURE;
37
38     BEGIN
39       ALLOCATE Q_P: [SQUARE];
40       S1 := #SIZE (Q_P^);
41       ALLOCATE R_P: [TRIANGLE];
42       S2 := #SIZE (R_P^);
43     END;
44
45   PROCEND MAIN;
46 MODEND.
```

GENERAL INTRINSICS

#CONVERT_POINTER_TO_PROCEDURE (p,9)
Converts a pointer to a procedure with no parameters (p) to a pointer to a procedure with an arbitrary parameter list (q).

#KEYPOINT (class, data, id)
Generates the keypoint hardware instruction.

#SCAN (select, string, index, found)
Scans the string until a match is found with the select mask.

#TRANSLATE (table, source, destination)
Translates a source string into a destination string using a translation table.

#UNCHECKED_CONVERSION (source, target)
Copy source to target. Presumably source and target are different types.

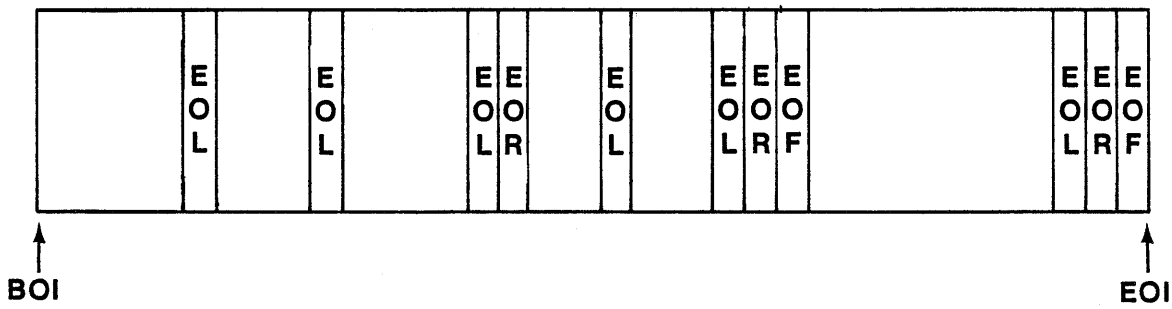
NOTE: SEE REFERENCE MANUAL FOR FURTHER DETAILS !

9

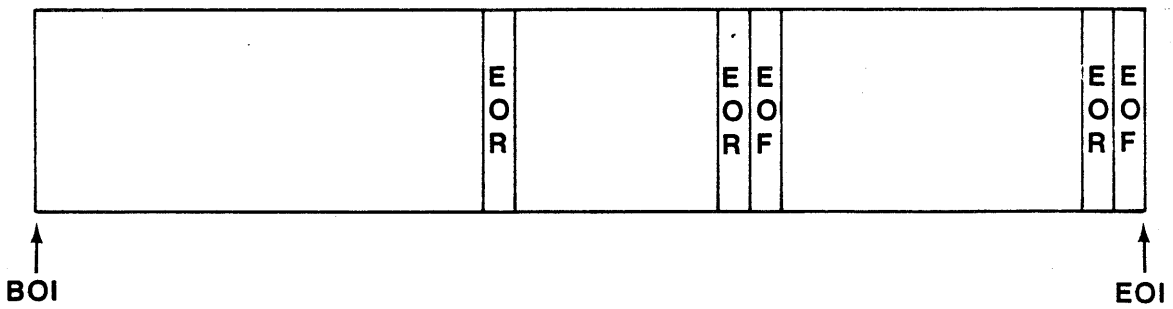
INPUT/OUTPUT

FILE STRUCTURE

LEGIBLE/PRINT



BINARY/DIRECT



CYBIL INPUT/OUTPUT

- PREDEFINED PROCEDURES
- ACCESS METHODS
  LEGIBLE (LG #...)
  PRINT (PR #...)
  BINARY (BI #...)
  DIRECT (DI #...)
- PREDEFINED TYPE DECLARATIONS
  FILE:∧CELL
  ORDINALS

WRITE LEGIBLE

```
/SES.GENCOMP SF=WLG CYBCOMN
*     GENERATING COMPILE FILE  COMPILE
*     END GENCOMP     COMPILE <- BASE
/SES.CYBIL CC
*     COMPILING  COMPILE
*     END CYBIL     COMPILE -> LISTING, LGO
/SES.LINK170 CYBCLIB
*     END LINK170     LGOB
/LGOB
START OF JOB
END OF JOB
LGOB.
/REWIND,WLG
$REWIND,WLG.
/COPY,WLG
MODULE WRITE_LEGIBLE;

*CALL PXIOTYP
*CALL LGZOPEN
*CALL LGZCLOS
*CALL LGZPUT

PROGRAM A;

  VAR
    OUT: FILE;

  LG#OPEN(OUT,'OUTPUT',OLD#,OUTPUT#,ASIS#);
  LG#PUT(OUT,'START OF JOB');
  LG#PUT(OUT,'END OF JOB');
  LG#CLOSE(OUT,ASIS#);

PROCEND A;

MODEND
 EOI ENCOUNTERED.
```

OPEN/CLOSE PROCEDURES

xx# OPEN (f,name, status, mode, position)

PR#OPEN (f, name, status, position)

xx #CLOSE (f, disposition)

F#SABF (f)

LG#CODESET (f, encoding)

PR#CODESET (f, encoding)

PR#LIMIT (f, lines_per_page)

PR#SETPGNO (f, page_number)

WRITE

LEGIBLE

LG#PUT (f, source)

LG#PUTPART (f, eol, source)

LG#WEOL (f)

PRINT FILE

PR#PUT (f, source)

PR#PUTPART (f, eol, source)

PR#WEOL (f)

NOTES— source, target: STRING (*)
eol : boolean
chars_read : integer

WRITE LEGIBLE FILE

```
 0     1 MODULE WRITE_LEGIBLE;
 0     2
 0     3          TYPE
 0     4            FILE = ^CELL,
 0     5            FILE_STATUS = (NEW#, OLD#),
 0     6            FILE_MODE = (INPUT#, OUTPUT#, CONCURRENT#),
 0     7            FILE_ENCODING = (ASCII64#, ASCII612#, ASCII#),
 0     8            FILE_MARK = (DATA#, EOR#, EOF#, EOI#),
 0     9            FILE_POSITION = (FIRST#, ASIS#, LAST#);
 0    10          CONST
 0    11            RETURN# = LAST#;
 0    12          TYPE
 0    13            FILE_DISPOSITION = FIRST# .. RETURN#;
 0    14                { I.E. (FIRST#, ASIS#, RETURN#) }
 0    15          PROCEDURE [XREF] LG#OPEN
 0    16               (VAR LEGIBLE_FILE : FILE;
 0    17                    FILE_NAME : STRING (*);
 0    18                    STATUS : FILE_STATUS;
 0    19                    MODE : FILE_MODE;
 0    20                    POSITION : FILE_POSITION);
 0    21          PROCEDURE [XREF] LG#CLOSE
 0    22               (    LEGIBLE_FILE : FILE;
 0    23                    DISPOSITION : FILE_DISPOSITION);
 0    24          PROCEDURE [XREF] LG#PUT
 0    25               (    LEGIBLE_FILE : FILE;
 0    26                    LINE : STRING (*));
 0    27
 0    28 PROGRAM A;
 0    29
17    30   VAR
17    31     OUT: FILE;
17    32
17    33   LG#OPEN(OUT,'OUTPUT',OLD#,OUTPUT#,ASIS#);
32    34   LG#PUT(OUT,'START OF JOB');
37    35   LG#PUT(OUT,'END OF JOB');
44    36   LG#CLOSE(OUT,ASIS#);
44    37
46    38 PROCEND A;
46    39
46    40 MODEND
```

PARTIAL WRITE

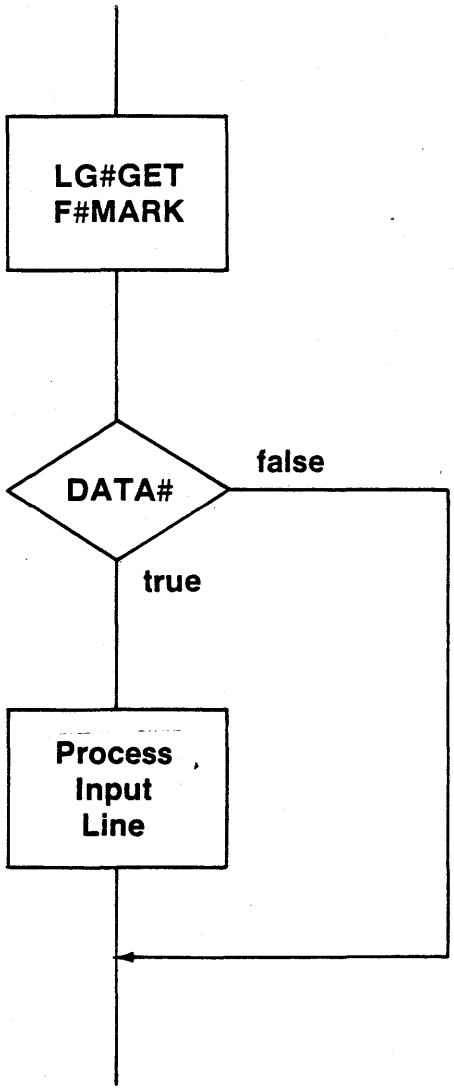
```
MODULE WRITE_LEGIBLE;

?? SET (LIST := OFF) ??
*CALL PXIOTYP
*CALL LGZOPEN
*CALL LGZCLOS
*CALL LGZPUT
*CALL LGZPUTP
*CALL LGZWEOL
?? RESET ??

  PROGRAM A;

    VAR
      OUT: FILE,
      XARRAY: [STATIC] ARRAY[1 .. 5] OF INTEGER := [5, 10, - 5, 100, 0],

      I: 1 .. 5,
      S: STRING (10),
      L: INTEGER;

    LG#OPEN (OUT, 'OUTPUT', OLD#, OUTPUT#, ASIS#);
    LG#PUT (OUT, 'START OF JOB');

    FOR I := 1 TO 5 DO
      STRINGREP (S, L, XARRAY[I]);
      LG#PUTPART (OUT, FALSE, S);
    FOREND;
    LG#WEOL (OUT);

    LG#PUT (OUT, 'END OF JOB');
    LG#CLOSE (OUT, ASIS#);

  PROCEND A;

MODEND
```

READ LEGIBLE

LG#GET (f, chars_read, target)

LG#GETPART (f, eol, chars_read, target)

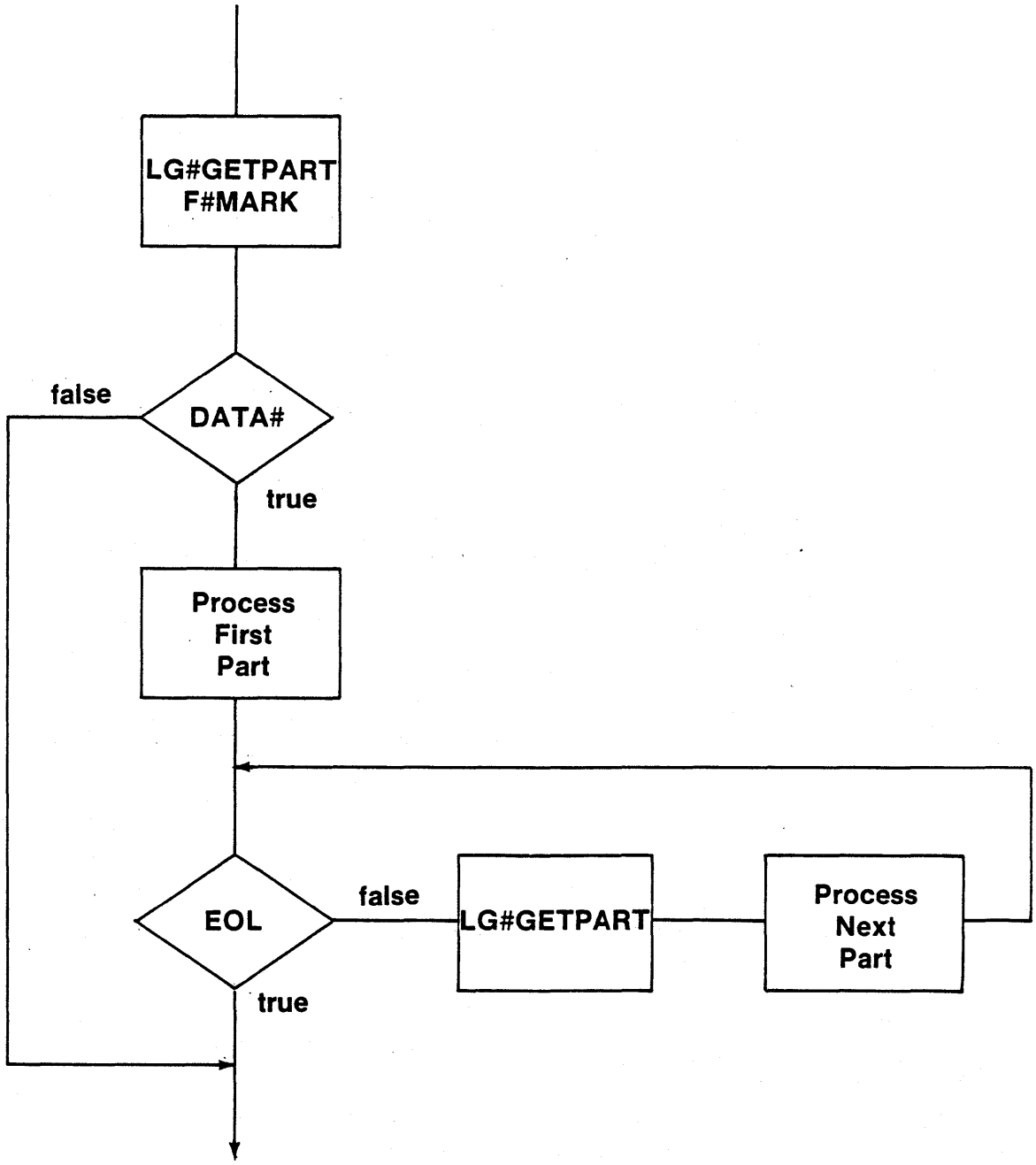
F#MARK (f, mark)



READ LEGIBLE

```
MODULE READ_LEGIBLE;

  PROGRAM B;

?? SET (LIST := OFF) ??
*CALL PXIOTYP
*CALL LGZOPEN
*CALL LGZCLOS
*CALL LGZGET
*CALL FZMARK
?? RESET ??

     VAR
       I: 1 .. 80,
       NCHARS: INTEGER,
       EOL: BOOLEAN,
       MARK: FILE_MARK,
       STR: STRING (80),
       INPUT: FILE;

     LG#OPEN (INPUT, 'INPUT', OLD#, INPUT#, ASIS#);

     LG#GET (INPUT, NCHARS, STR);
     F#MARK (INPUT, MARK);
     WHILE (STR (1) <> ')') AND (MARK = DATA#) DO

       { PROCESS INPUT STRING }

       LG#GET (INPUT, NCHARS, STR);
       F#MARK (INPUT, MARK);
     WHILEND;

     LG#CLOSE (INPUT, ASIS#);

  PROCEND B;
MODEND
```

READ PARTIAL LINE



PARTIAL READ

```
MODULE READ_LEGIBLE;

  PROGRAM B;

?? SET (LIST := OFF) ??
*CALL PXIOTYP
*CALL LGZOPEN
*CALL LGZCLOS
*CALL LGZGET
*CALL LGZGETP
*CALL FZMARK
?? RESET ??

    VAR
      I: 1 .. 80,
      NCHARS: INTEGER,
      EOL: BOOLEAN,
      MARK: FILE_MARK,
      STR: STRING (80),
      INPUT: FILE;

    NCHARS := 10;
    LG#OPEN (INPUT, 'INPUT', OLD#, INPUT#, ASIS#);
    LG#GETPART (INPUT, EOL, NCHARS, STR);
    F#MARK (INPUT, MARK);
    WHILE MARK = DATA# DO
      FOR I := 1 TO NCHARS DO
        { PROCESS STR (I) }
      FOREND;
      WHILE NOT EOL DO
        LG#GETPART (INPUT, EOL, NCHARS, STR);
        FOR I := 1 TO NCHARS DO
          { PROCESS STR(I)}
        FOREND;
      WHILEND;
      { PROCESS ANOTHER LINE }
      LG#GETPART (INPUT, EOL, NCHARS, STR);
      F#MARK (INPUT, MARK);
    WHILEND;
    LG#CLOSE (INPUT, ASIS#);

  PROCEND B;
MODEND
```

READ/WRITE BINARY

BI#PUT (f, source, length)

BI#GET (f, target, length)

NOTE: source, target: Λ cell

WRITE BINARY FILE

```
MODULE WRITE_BINARY;

?? SET (LIST := OFF) ??
*CALL PXIOTYP
*CALL BIZOPEN
*CALL BIZCLOS
*CALL BIZPUT
?? RESET ??

  PROGRAM C;

    VAR
      BIN: FILE;
      X: STRING (100);
      Y: ARRAY[1 .. 1000] OF INTEGER;

    { CONSTRUCT VALUES FOR X AND Y }

    BI#OPEN (BIN, 'BINOUT', NEW#, OUTPUT#, FIRST#);
    BI#PUT (BIN, #LOC (X), #SIZE (X));
    BI#PUT (BIN, #LOC (Y), #SIZE (Y));
    BI#CLOSE (BIN, FIRST#);

  PROCEND;

MODEND
```

READ/WRITE DIRECT

DI#PUT (f, key, source, length)

DI#PUTDIR (f, key, source, length)

DI#GET (f, key, target, length)

DI#GETDIR (f, key, target, length)

DI#LOCATE (f, key)

OPERATION	XFER	KEY	LENGTH
PUT	OUT	returned to program	supplied
PUTDIR	OUT	supplied by program	supplied
GET	IN	returned to program	returned
GETDIR	IN	supplied by program	returned
LOCATE	none	supplied by program	N/A

WRITE DIRECT FILE

```
MODULE CONSTRUCT_DIRECT_FILE;

?? SET (LIST := OFF) ??
*CALL PXIOTYP
*CALL DIZOPEN
*CALL DIZCLOS
*CALL DIZPUT
?? RESET ??

  PROGRAM D;

    TYPE
      RECS = RECORD
        NAME: STRING (20),
        JULIAN_DATE: 0 .. 99999,
      RECEND;

    VAR
      I: 1 .. 100,
      X: ARRAY[1 .. 100] OF RECS,
      KEY: ARRAY[1 .. 100] OF INTEGER,
      DIR: FILE;

    DI#OPEN (DIR, 'DIRFILE', NEW#, OUTPUT#, FIRST#);
    FOR I := 1 TO 100 DO
      DI#PUT (DIR, KEY[I], #LOC (X[I]), #SIZE (X[I]));
    FOREND;
    DI#CLOSE (DIR, FIRST#);

  PROCEND D;
MODEND
```

POSITIONING

xx#FIRST (f)

xx#LAST (f)

DI#LOCATE (f, key)

LG#TAB (f, column_number)

PR#TAB (f, column_number)

PR#LINE (f, line_number)

PR#SKIP (f, number of_lines)

PR#EJECT (f)

PR#PAGE (f)

END-OF-DATA HANDLING

xx#WEOR (f)	Write END-OF-RECORD
xx#WEOF (f)	Write END-OF-FILE
F#MARK (f, mark)	Return last file structure mark
F#WORDS (f, words)	Return length of last transfer

STATUS INTERROGATION

F#TERMINAL (f, is_term)

xx#OLDCODESET (f, encoding)

xx#COLNO (f, column_number)

PR#LINO (f, line_number)

PR#PGNO (f, page_number)

PR#OLDLIMIT (f, no_of_lines)

NOTES— is_term : boolean
xx=LG or PR only

10

COMPILE TIME FACILITIES

SES. CYBIL OPTIONS

I = Source file [COMPILE]

L = Output file [LISTING]

Suppress source (0)

B = Binary file [LGO]

CHK by itself = [NRST]

CHK = Nil pointer reference (N), range (R),

subscripts (S), none (0) [NRS] ← DEFAULT

, VARIANT RECORD TAG (T),

LO = Full listing - A, S, R (F)

Generated code (O),

Attributes (A),

List fatal diagnostics only (W),

Cross reference list (R or RA), ← INCLUDES UNREFERENCED

Source (S), ← DEFAULT

Use listing pragmats if LISTEXT is ON

and LO = X

[OFF] = DEFAULT

D = Symbol Table (SD) — ALLOW SYMBOLIC DEBUG

Debugging Statements (DS) ← TURN ON DEBUGGING STMTS DELIMITED BY NOCOMPILE/COMPILE

Full Debug (FD)

CYBFORM
(before)

```
ASCII
/BATCH
/GET:FOR
/COPY:FOR
```

```
module for_statement;
 program main;
   const
   initial=1,
   final=100;
   var
   w:integer,
   x:integer,
   a:array[initial..50] of integer;
x:=initial;
for w:= initial to final do
 x:=x+1;
 a[w] := a[w] +x;
forend;
procend main;
modend for_statement;
```

CYBFORM
(after)

```
/SES.CYBFORM FOR
*     END CYBFORM     FOR
/REWIND,FOR
$REWIND,FOR.
/COPY,FOR
```

```
MODULE for_statement;

  PROGRAM main;

    CONST
      initial = 1,
      final = 100;

    VAR
      w: integer,
      x: integer,
      a: array [initial .. 50] of integer;

    x := initial;
    FOR w := initial TO final DO
      x := x + 1;
      a [w] := a [w] + x;
    FOREND;
  PROCEND main;
MODEND for_statement;
```

```
/SES.CYBIL FOR CC LO=F
*    COMPILING  FOR
*    END CYBIL    FOR -> LISTING, LGO
/REWIND,LISTING
$REWIND,LISTING.
/COPY,LISTING
1SOURCE LISTING OF   FOR_STATEMENT                          NOS       CYBIL/CC
 1.0  8133 (     )            12/30/81   4:09 PM            PAGE 1

        0      1 MODULE FOR_STATEMENT;
        0      2
        0      3   PROGRAM MAIN;
        0      4
       10      5     CONST
       10      6       INITIAL = 1,
       10      7       FINAL = 100;
       10      8
       10      9     VAR
       10     10       W: INTEGER,
       10     11       X: INTEGER,
       10     12       A: ARRAY [INITIAL .. 50] OF INTEGER;
       10     13
       10     14     X := INITIAL;
       15     15     FOR W := INITIAL TO FINAL DO
       16     16       X := X + 1;
       20     17       A [W] := A [W] + X;
       30     18     FOREND;
       33     19   PROCEND MAIN;
       33     20 MODEND FOR_STATEMENT;
0
    **** NO DIAGNOSTICS
```

```
1CROSS REFERENCES OF FOR_STATEMENT                                  NOS       CYBIL/C
 1.0  8133 (      )            12/30/81    4:09 PM                  PAGE 2

 IDENTIFIER--------------------DEFINED----------------REFERENCES
                                ON LINE
 A                                  12                      17/M      17

 FINAL                               7                      15

 INITIAL                             6                      12        14
    15

 MAIN                                3                      19

 W                                  10                      15        17/S
    17/S

 X                                  11                      14/M      16/M
    16        17
-
-
-
-
-
-
-
-
-
-
-
-
-
-
0                          ***  REFERENCE ABBREVIATIONS : M=MODIFY, A=AT
TRIBUTE, S=SUBSCRIPT, I=I/O REF, R=READ, W=WRITE, P=PARAMETER
 EOI ENCOUNTERED.
```

```
1ATTRIBUTE LIST FOR  FOR_STATEMENT                          NOS       CYBIL/CC
 1.0  8133 (      )            12/30/81    4:09 PM          PAGE 1

 IDENTIFIER--------------------DEFINED-----SIZE UNIT--TYPE-------LOCATIO
N--BLOCK--NEST--CONTAINED OR DECLARED IN---------ATTRIBUTES
                                ON LINE.                                 SEC+OF
F          LEVEL
 A                                    12        50 WORD  STRUCTURE   AUTO+7
       2      1   MAIN
                                                         ARRAY

 FINAL                                 7         0       CONSTANT
       2      1   MAIN

 INITIAL                               6         0       CONSTANT
       2      1   MAIN

 MAIN                                  3         0       PROCEDURE
       1      0

 W                                    10         1 WORD  VARIABLE    AUTO+5
       2      1   MAIN
                                                         INTEGER

 X                                    11         1 WORD  VARIABLE    AUTO+6
       2      1   MAIN
                                                         INTEGER
```

LOADER MAP

```
/MAP,PART
MAP,PART.
/SES.LINK170 CYBCLIB
S
1          LOAD MAP - FOR_STA                                          CYBER LOADER 1.5
      80/01/02. 15.18.35.      PAGE      1


           FWA OF THE LOAD              111
           LWA+1 OF THE LOAD           5572

           WRITTEN TO FILE      LGOB

           TRANSFER ADDRESS -- SW=MAIN           121

           PROGRAM ENTRY POINTS --        FOR_STA           121


           PROGRAM AND BLOCK ASSIGNMENTS.

           BLOCK        ADDRESS   LENGTH    FILE      DATE      PROCSSR VER LEVEL  H
RE    COMMENTS

           FOR_STA          111       37    LGO       80-01-02 CYBIL   1.0     0

           SW=ERSC          150       44 UL-CYBCLIB  79/08/24 COMPASS
      PASCAL-X RUNTIME - ERROR EXITS
           SW=STST          214       63 UL-CYBCLIB  79/0
      PASCAL-X STACK MGR- PROLOG EXCEPTION + OBTAIN
           SW=MMST          277       27 UL-CY
      PASCAL-X RUNTIME - INITIALIZE PRO                                  ..OCSSR VER LEVEL  H
           SW=MAMA          326
                                          •
           SW=STSS                        •
      PASCAL-X ST                         •          ..IB  79/08/24 COMPASS 3.6 498
           .
      -                                  .06 SL-SYSLIB   79/11/25 COMPASS 3.6 508

                           5500       13 SL-SYSLIB   79/11/25 COMPASS 3.6 508
                   .ROCESSOR.
              ..SYS        5513       40 SL-SYSLIB   79/11/25 COMPASS 3.6 508
        ..ESS SYSTEM REQUEST.
           CPU.WTO         5553       17 SL-SYSLIB   79/11/25 COMPASS 3.6 508
      WRITE ONE WORD.


                .048 CP SECONDS                 21600B CM STORAGE USED
     2 TABLE MOVES


S
*     END LINK170    LGOB
```

ABORT DUMP

```
/LGO3
1- CYBIL POST MORTEM PROCESSOR
 - ABORT AT     213  STACK=   7501
 - CP ABORT FROM     135
 - SUBSCRIPTRANGE ERROR
-   EXCHANGE PACKAGE
   I     A[I]     B[I]            X[I]

   0     7575             000000000000000000000
   1     7506        1    000000000000000000063
   2     7571     7501    000000000000000000000
   3     7507     5573    000000000000000000063
   4        1       -1    000000000000000000000
   5     7560        2    777777777777777777764
   6        1     2341.   010224200001030000135
   7     7507       71    010224200000000000000
-  CALL NEST(AND STACK DUMP)
 ABS ADDR     STACKFRAME
     135         7501
     323         7572
- SUBSCRIPTRANGE ERROR
```

OBJECT CODE

```
/SES.CYBIL FOR CC LO=O
*    COMPILING  FOR
*    END CYBIL    FOR -> LISTING, LGO
/REWIND,LISTING
$REWIND,LISTING.
/COPY,LISTING
1SOURCE LISTING OF    FOR_STATEMENT
33 (     )            12/18/81    1:24 PM          PAGE 1

      10      1 MODULE FOR_STATEMENT;
      10      2
      10      3   PROGRAM MAIN;
      10      4
      10      5     CONST
      10      6       INITIAL = 1,
      10      7       FINAL = 100;
      10      8
      10      9     VAR
      10     10       W: INTEGER,
      10     11       X: INTEGER,
      10     12       A: ARRAY [INITIAL .. 50] OF INTEGER;
      10     13
      10     14     X := INITIAL;

      10 6170000071            L_8     SB7 B0+57                  * R_1 R_0
                  0100000000X          RJ  0                      * CIL#PGI
      11 0100000000X           L_9     RJ  0                      * CIL#SPE
                  6100000000           SB0 B0+0                   * R_0 R_0
      12 76020                 L_4     SX0 B2+B0                  * R_3 R_0 R_1
                  76010                SX0 B1+B0                  * R_2 R_0 R_1
                       36660           IX6 X6+X0                  * R_2A R_0 R_2
      31 7110000144                    SX1 B0+100                 * R_1 R_0
                  37116                IX1 X1-X6                  * R_5C R_0 R_2A
                       46000           NO                         *
      32 0321000016+                   PL  X1,14                  * R_0 L_5
                  6100000000           SB0 B0+0                   * R_0 R_0

      33     19   PROCEND MAIN;

                               L_6
      33 56120                 L_2     SA1 B2+B0                  * R_3 R_0 R_1
              63710                    SB7 X1+B0                  * R_67 R_0 R_1
                  6122000071           SB2 B2+57                  * R_3 R_0
      34 0270000000                    JP  B7+0                   * R_0
                  6100000000           SB0 B0+0                   * R_0 R_0
      15                 46000         NO                         *

      35     20 MODEND FOR_STATEMENT;
0
   **** NO DIAGNOSTICS
 EOI ENCOUNTERED.
```

COMPILE TIME STATEMENTS

VARIABLES

```
? VAR varid:boolean := exp ?
      .
      .
      .
? varid := exp ?
```

CONDITIONS

```
? IF exp THEN
      executable statements
? ELSE
      executable statements
? IF END
```

COMPILE TIME VARIABLES

```
 0      1 MODULE COMP_TIME;
 0      2
 0      3   TYPE
 0      4     REC = RECORD
 0      5       KEY: STRING (6),
 0      6       P: ^UNIT
 0      7     RECEND,
 0      8     UNIT = RECORD
 0      9       AMOUNT: INTEGER,
 0     10     RECEND;
 0     11
 0     12   ?VAR
 0     13     C170A: BOOLEAN := FALSE,
 0     14     C170B: BOOLEAN := TRUE,
 0     15     C170C: BOOLEAN := FALSE?;
 0     16
 0     17   ?C170A := TRUE?;
 0     18
 0     19   PROGRAM TEST_COMP_TIME;
 0     20
11     21     VAR
11     22       A: ARRAY [1 .. 1000] OF REC,
11     23       K: STRING (6),
11     24       LOC: ^UNIT;
11     25
11     26     PROCEDURE [XREF] A_PROC (PA: ARRAY [ * ] OF REC,
11     27           PK: STRING ( * );
11     28       VAR PL: ^UNIT);
11     29
11     30     PROCEDURE [XREF] B_PROC (PA: ARRAY [ * ] OF REC,
11     31           PK: STRING ( * );
11     32       VAR PL: ^UNIT);
11     33
11     34     PROCEDURE [XREF] C_PROC (PA: ARRAY [ * ] OF REC,
11     35           PK: STRING ( * );
11     36       VAR PL: ^UNIT);
11     37
11     38     ?IF C170B AND C170C THEN
       39       B_PROC (A, K, LOC);
11     40     ?IFEND;
11     41     ?IF C170A AND NOT C170B THEN
       42       A_PROC (A, K, LOC);
11     43     ?ELSE
15     44       C_PROC (A, K, LOC);
15     45     ?IFEND;
25     46   PROCEND TEST_COMP_TIME;
25     47 MODEND COMP_TIME;
```

LAYOUT CONTROL

?? keyword : = value ??

- SOURCE LAYOUT

  LEFT (1)
  RIGHT (79)

- LISTING LAYOUT

  ~~PAGESIZE (58)~~
  EJECT
  SPACING (1)
  SKIP
  NEWTITLE : = '...'
  TITLE : = '...'
  OLDTITLE

- MAINTENANCE

  COMPILE
  NOCOMPILE

TITLES

```
    CYBIL/CC          V1.0 790712        SOURCE LISTING - LINK_LIST
      80-01-02    15:45:25   PAGE      1
  0     2 ?? TITLE := 'LINKED LIST PROCESSOR' ??
  0     3 ?? LEFT := 1, RIGHT := 79 ??
  0     4 MODULE LINK_LIST;
  0     5
  0     6   TYPE
  0     7     LINK = ^REC;
  0     8     REC = RECORD
  0     9       VALUE: INTEGER;
  0    10       PTR: LINK
  0    11     RECEND;
  0    12
  0    13   VAR
  0    14     FIRST: LINK := NIL;
  0    15
  0    16 ?? NEWTITLE := ' MAIN LOOP ' ??
  0    17 ?? EJECT ??

    CYBIL/CC          V1.0 790712        SOURCE LISTING - LINK_LIST
      80-01-02    15:45:25   PAGE      2
              LINKED LIST PROCESSOR
               MAIN LOOP
  0    18
  0    19   PROGRAM MAIN;
  0    20
  0    21
 11    22   PROCEND MAIN;
 11    23 ?? OLDTITLE ??
 11    24 ?? NEWTITLE := ' ADD ELEMENT TO LIST' ??
 11    25 ?? EJECT ??

    CYBIL/CC          V1.0 790712        SOURCE LISTING - LINK_LIST
      80-01-02    15:45:25   PAGE      3
              LINKED LIST PROCESSOR
               ADD ELEMENT TO LIST
 11    26
 11    27   PROCEDURE ADD (P: LINK);
 11    28
 11    29
 20    30   PROCEND ADD;
 20    31 MODEND;

    CYBIL/CC          V1.0 790712  COMPILATION SUMMARY - LINK_LIST
      80-01-02    15:45:25   PAGE      1
31 LINES COMPILED.
NO COMPILATION ERRORS.
```

EXAMPLES OF NEWTITLE, TITLE, OLDTITLE & EJECT

```
/COPY,LISTING
1SOURCE LISTING OF    RM0002                                    NOS       CYBIL/CC 1.0   8219
(      NOVEMBER 10, 1982      11:00 AM       PAGE 1

        0      1 ?? SET (LISTCTS := ON) ??
        0      2 MODULE RM0002;
        0      3
        0      4   PROGRAM MAIN;
        0      5 ?? TITLE := 'TITLE 1' ??
        0      6 ?? EJECT ??
1SOURCE LISTING OF    RM0002                                    NOS       CYBIL/CC 1.0   8219
(      )  NOVEMBER 10, 1982      11:00 AM       PAGE 2

 TITLE 1

        0      7
       10      8      TYPE
       10      9        INTRANGE = 0 .. 7;
       10     10
       10     11 ?? OLDTITLE ??
       10     12 ?? EJECT ??
1SOURCE LISTING OF    RM0002                                    NOS       CYBIL/CC 1.0   8219
(      NOVEMBER 10, 1982      11:00 AM       PAGE 3

       10     13
       10     14      TYPE
       10     15        A = INTEGER;
       10     16
       10     17 ?? NEWTITLE := 'NEWTITLE 1' ??
       10     18 ?? EJECT ??
1SOURCE LISTING OF    RM0002                                    NOS       CYBIL/CC 1.0   8219
(      )  NOVEMBER 10, 1982      11:00 AM       PAGE 4

 NEWTITLE 1

       10     19
       10     20      TYPE
       10     21        B = INTEGER;
       10     22
       10     23 ?? TITLE := 'TITLE 2' ??
       10     24 ?? EJECT ??
1SOURCE LISTING OF    RM0002                                    NOS       CYBIL/CC 1.0   8219
(      NOVEMBER 10, 1982      11:00 AM       PAGE 5

 TITLE 2

       10     25
       10     26      TYPE
       10     27        C = INTEGER;
       10     28
       10     29 ?? NEWTITLE := 'NEWTITLE 2' ??
       10     30 ?? EJECT ??
1SOURCE LISTING OF    RM0002                                    NOS       CYBIL/CC 1.0   8219
(      NOVEMBER 10, 1982      11:00 AM       PAGE 6

 TITLE 2
 NEWTITLE 2

       10     31
       10     32      TYPE
       10     33        D = INTEGER;
       10     34
       10     35 ?? NEWTITLE := 'NEWTITLE 3' ??
       10     36 ?? EJECT ??
1SOURCE LISTING OF    RM0002                                    NOS       CYBIL/CC 1.0   8219
(      NOVEMBER 10, 1982      11:00 AM       PAGE 7

 TITLE 2
 NEWTITLE 2
 NEWTITLE 3

       10     37
       10     38      TYPE
       10     39        E = INTEGER;
       10     40
       10     41 ?? NEWTITLE := 'NEWTITLE 4' ??
       10     42 ?? EJECT ??
```

```
1SOURCE LISTING OF    RM0002                                    NOS       CYBIL/CC 1.0  8219
  (       NOVEMBER 10, 1982      11:00 AM       PAGE 8

 TITLE 2
 NEWTITLE 2
 NEWTITLE 3
 NEWTITLE 4

        10     43
        10     44      TYPE
        10     45        F = INTEGER;
        10     46
        10     47 ?? NEWTITLE := 'NEWTITLE 5' ??
        10     48 ?? EJECT ??
1SOURCE LISTING OF    RM0002                                    NOS       CYBIL/CC 1.0  8219
  (     )  NOVEMBER 10, 1982      11:00 AM       PAGE 9

 TITLE 2
 NEWTITLE 2
 NEWTITLE 3
 NEWTITLE 4
 NEWTITLE 5

        10     49
        10     50      TYPE
        10     51        G = INTEGER;
        10     52
        10     53 ?? OLDTITLE ??
        10     54 ?? EJECT ??
1SOURCE LISTING OF    RM0002                                    NOS       CYBIL/CC 1.0  8219
  (       NOVEMBER 10, 1982      11:00 AM       PAGE 10

 TITLE 2
 NEWTITLE 2
 NEWTITLE 3
 NEWTITLE 4

        10     55
        10     56      TYPE
        10     57        H = INTEGER;
        10     58
        10     59 ?? TITLE := 'TITLE 3' ??
        10     60 ?? EJECT ??
1SOURCE LISTING OF    RM0002                                    NOS       CYBIL/CC 1.0  8219
  (       NOVEMBER 10, 1982      11:00 AM       PAGE 11

 TITLE 2
 NEWTITLE 2
 NEWTITLE 3
 TITLE 3

        10     61   PROCEND MAIN;
        10     62 ?? OLDTITLE ??
        10     63 ?? EJECT ??
1SOURCE LISTING OF    RM0002                                    NOS       CYBIL/CC 1.0  8219
  (       NOVEMBER 10, 1982      11:00 AM       PAGE 12

 TITLE 2
 NEWTITLE 2
 NEWTITLE 3

        10     64 MODEND RM0002;
0
    **** NO DIAGNOSTICS
 EOI ENCOUNTERED.
/
```

TOGGLES

?? control (toggle: = ON/OFF) ??

- TOGGLE CONTROL
  - SET
  - PUSH
  - POP
  - RESET ← no params; resets defaults

- LISTING TOGGLES

LIST	produce listing (ON)
LISTOBJ	list object code (OFF)
LISTCTS	list compile time pragmats (OFF)
LISTEXT	command control of listing; controlled by LO=X (OFF)
LISTALL	union of all listing toggles

- CHECKING TOGGLES

CHKRNG	check range (ON)
CHKSUB	check subscripts (ON)
CHKNIL	check pointer dereference (OFF)
CHKTAG	check variant record tags (ON)
CHKALL	

TOGGLES

```
?? SET (LIST := ON, LISTOBJ := ON) ??

?? PUSH (LIST := OFF) ??

?? POP ??

?? RESET, SET (CHKSUB := OFF) ??
```

TOGGLES

```
/SES.CYBIL FOR CC
*     COMPILING  FOR
*     END CYBIL    FOR -> LISTING, LGO
/SES.LINK170 CYBCLIB
*     END LINK170    LGOB
/LGOB
LGOB.
/REWIND,LISTING
$REWIND,LISTING.
/COPY,LISTING
1            CYBIL/CC          V1.0 790712        SOURCE LISTING - FOR_STATEMENT
               80-01-03     08:38:01    PAGE      1
0        0      2 MODULE FOR_STATEMENT;
        10      3 ?? SET (CHKSUB := OFF) ??
        10      4
        10      5   PROGRAM MAIN;
        10      6
        10      7     CONST
        10      8       INITIAL = 1;
        10      9       FINAL = 100;
        10     10
        10     11     VAR
        10     12       W: INTEGER;
        10     13       X: INTEGER;
        10     14       A: ARRAY [INITIAL .. 50] OF INTEGER;
        10     15
        10     16     X := INITIAL;
        15     17     FOR W := INITIAL TO FINAL DO
        16     18       X := X + 1;
        20     19       A [W] := A [W] + X;
        23     20     FOREND;
        26     21   PROCEND MAIN;
        26     22 MODEND FOR_STATEMENT;

1            CYBIL/CC          V1.0 790712  COMPILATION SUMMARY - FOR_STATEMENT
               80-01-03     08:38:01    PAGE      1
0      22 LINES COMPILED.
       NO COMPILATION ERRORS.
 EOI ENCOUNTERED.
/
```

11

INTERACTIVE DEBUGGER

CYBIL INTERACTIVE DEBUGGER

- VARIATION OF CID

- REFERENCES CYBIL
  VARIABLE NAMES
  LINE NUMBERS
  MODULES
  PROCEDURES

- SCL-LIKE COMMAND STRUCTURE

- CAPABILITIES
  BREAKPOINTS
  TRAPS
  DEBUGGER PROCEDURES
  VARIABLE CONTROL
  MEMORY AND REGISTER CONTROL
  TRACEBACK

SQA/SQB

```
 0     1
 0     2 MODULE sqa;
 0     3
 0     4   CONST
 0     5     min = 1,
 0     6     max = 30;
 0     7
 0     8   TYPE
 0     9     t_array = array [min .. max] of integer;
 0    10
 0    11 ?? SET (LIST := OFF) ??
 0    36 ?? RESET ??
 0    37
 0    38   PROCEDURE [XREF] squarer (b: t_array;
 0    39     VAR sq: t_array);
 0    40
 0    41   PROGRAM main;
 0    42
21    43     VAR
21    44       base: t_array,
21    45       square: t_array,
21    46       i: min .. max,
21    47       out: file;
21    48
21    49     lg#open (out, 'output', old#, output#, asis#);
34    50     lg#put (out, 'start of program');
41    51     for i := min to max do
45    52       base [i] := i;
47    53     forend;
52    54     squarer (base, square);
54    55     lg#put (out, 'End of Program');
61    56     lg#close (out, asis#);
63    57   procend main;
63    58 modend sqa;
```

```
 0     1 MODULE sqb;
 0     2
 0     3   CONST
 0     4     min = 1,
 0     5     max = 30;
 0     6
 0     7   TYPE
 0     8     t_array = array [min .. max] of integer;
 0     9
 0    10   PROCEDURE [XDCL] squarer (b: t_array;
 0    11     VAR sq: t_array);
 0    12
10    13     VAR
10    14       j: min .. max;
10    15
10    16     for j := min to max do
23    17       sq [j] := b [j] * b [j];
30    18     forend;
33    19   procend squarer;
33    20 modend sqb;
```

SAMPLE—1

```
/SES.GENCOMP SF=SQ CYBCOMN
*     GENERATING COMPILE FILE  COMPILE
*     END GENCOMP    COMPILE <- BASE
/SES.CYBIL I=COMPILE CC D=ST
*     COMPILING  COMPILE
*     END CYBIL    COMPILE -> LISTING, LGO
/SES.LINK170 CYBCLIB DEBUG
*     END LINK170    LGOB,ZZZZZDT
/DEBUG;ON
$DEBUG;ON.
/LGOB
 PASCAL-X INTERACTIVE DEBUG
? SB 48
 *WARN - LINE 48 NOT EXECUTABLE - LINE 49 WILL BE USED
 OK ? YES
? SB M=SQB L=17 C; DV VAR=J; PA; CE
 IN COLLECT MODE
 END COLLECT
? DB
 *B #1 =  LINE=48,   *B #2 =  MODULE=SQB LINE=17
? SAVEB BPA
? DV BASE
 *ERROR - NO PROGRAM VARIABLE BASE
? GO
 GO
 *B #1, AT  LINE=48
? DV BASE
 BASE = 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
 0  0  0  0  0  0  0  0
? GO
 J = 1
 *B #2, AT  MODULE=SQB LINE=17
? DV BASE M=SQA
 *ERROR - NO PROGRAM VARIABLE BASE
? DV B
 DV B
 B = 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19
 20  21  22  23  24  25  26  27  28  29  30
```

SAMPLE—2

```
? GO
 J = 2
 *B #2, AT  MODULE=SQB LINE=17
? GO
 J = 3
 *B #2, AT  MODULE=SQB LINE=17
? DV SQ
 SQ = 1  4  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  1519389309
 0  0  0  0  1520961591  1520961588  -9793569970926387 1  2088  68  2088
? CV VAR=B[20] VALUE=1
? DV B
 B = 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  1
 21  22  23  24  25  26  27  28  29  30
? CB B=2
? CB
 *WARN - ALL WILL BE CLEARED
 OK ? YES
? SB 19
? GO
 *B #1, AT  LINE=19
? DV SQ
 SQ = 1  4  9  16  25  36  49  64  81  100  121  144  169  196  225  256
 289  324  361  1  441  484  529  576  625  676  729  784  841  900
? GO
START OF PROGRAM
END OF PROGRAM
 *T #17, END IN  MODULE LP#CLOS OFFSET 0(8)
? QUIT
 DEBUG TERMINATED
```

```
/SAVE;BPA
/SAVE;LGOB=SQLGO
```

SAMPLE—3

```
./SES.LINK170 SQLGO DEBUG CYBCLIB
*     END LINK170     LGOB,ZZZZZDT
/GET,BPA
/DEBUG,ON
$DEBUG,ON.
/LGOB
 PASCAL-X INTERACTIVE DEBUG
? READ F=BPA
 *CMD - ( SB LINE=48; )  *WARN - LINE 48 NOT EXECUTABLE - LINE 49 WILL
 BE USED
 OK ? YES
? DB
 *B #1 =  LINE=48,   *B #2 =  MODULE=SQB LINE=17
? CB B=1
? SB 49 C
 IN COLLECT MODE
? ST TYPE=WRITE VAR=BASE
? CE
 END COLLECT
? SB L=54 C; CT; CE
 IN COLLECT MODE
 END COLLECT
? DT
 NO TRAPS
? DB
 *B #1 =  LINE=49,   *B #2 =  MODULE=SQB LINE=17,   *B #3 =  LINE=54
? DB B=1
 *B #1 =  LINE=49
 SB LINE=49 COLLECT;
 ST TYPE= WRITE VARIABLE=BASE;
 CE;
? SP ONE C
 IN COLLECT MODE
? DV I
? DV BASE; CE
 END COLLECT
? DP
 *P #1 = ONE
? DP PR=ONE
 *P #1 = ONE
 SP PROCEDURE= ONE;
 DV VARIABLE=I;
 DV VARIABLE=BASE;
 CE;
? GO
 INTERPRET MODE TURNED ON
 *T #1, WRITE INTO BASE IN  LINE 52
? READ ONE
 I = 1
 BASE = 1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
 0  0  0  0  0  0  0  0
```

SAMPLE—4

```
? GO
 *T #1, WRITE INTO BASE IN  LINE 52
? GO
 *T #1, WRITE INTO BASE IN  LINE 52
? READ ONE
 I = 6
 BASE = 1  2  3  4  5  6  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
 0  0  0  0  0  0  0  0
? CLEAR_INTERPRET
? GO
 J = 1
 *B #2, AT  MODULE=SQB LINE=17
? DT
 NO TRAPS
? TB
 SQUARER CALLED FROM MODULE SQA LINE 54 (PROC MAIN)
 *IN MAIN PROGRAM - TRACE COMPLETED*
? CHECKPOINT F=SQCK
? DD
 DEFAULTS = MODULE SQB PROC SQUARER
? DS; DMAP
 DEFAULTS = MODULE SQB PROC SQUARER,   3 BREAKPOINTS,   NO TRAPS
 1 PROCEDURES,   VETO OFF,   INTERPRET OFF,   OUT OPTIONS = I W E D
 AUXILIARY CLEAR
 DBUG.,   SQA,   SQB,   SW=ERSC,   SW=STST,   SW=MMST,   LP#CLOS
 LG#OPEN,   LG#PUT,   SW=MAMA,   SW=STSS,   SW=STIN,   SW=MMSZ,   SW=CMAL
 CVAS612,   CVASC64,   PXIOCLS,   PXIOOPN,   PXIOCOP,   SW=ERRA
 SW=CMST,   CMMINIT,   SW=CMFR,   SW=CMFB,   SW=CMEF,   SW=MMEP
 PXIOERR,   PR#OPEN,   ZUTISFN,   SW=I2S,   SW=ERPM,   BYPASS,   SW=STRM
 SW=CMIB,   LP#WEO,   ZN7IMSG,   ZUTIABT,   ZUTICPO,   ZUTMMSG,   ZUTMW2O
 SW=STPS,   ZUTMI2S,   ZUTMS2D,   ZCLMI2S,   ZUTIDCI,   ZUTIDCP
 ZCLIIAP,   UCLOAD,   CPU.CIO,   CPU.SYS,   CPU.WTO
? QUIT
 DEBUG TERMINATED


/SAVE;SQCK
 FILE TOO LONG,  AT 000121.
/DEFINE;SQCHECK
/COPY;SQCK;SQCHECK
 EOI ENCOUNTERED.


/ATTACH;SQCHECK
/DEBUG;RESUME;SQCHECK
 PASCAL-X INTERACTIVE DEBUG RESTARTED
 DEBUG INTERNAL ERROR.
```

SET BREAKPOINT

SET_BREAKPOINT | SB

```
Address Expression
  * line | l            = line number
    entrypoint | e      = entry point name
    location | loc      = absolute address
module | m              = module name
overlay | ovl           = overlay number (n,n)
offset | o              = offset from base address
first | f               = starting iteration number
last                    = last iteration number
step | s                = skip iterations count
collect | c             start procedure
Procedure commands separated by semi-colons or EOLs
collect_end | ce        end procedure
```

EXAMPLES

SB 48

SB m=sqb l=17 C; DV var=j; PA; CE

BREAKPOINT CONTROL

DISPLAY_BREAKPOINT | DB

Address Expression	
line | l	= line_numbers
location | loc	= absolute addresses
entrypoint | e	= entry point names
b	= breakpoint_numbers
offset | o	= offset from base address
overlay | ovl	= overlay number (n,n)
module | m	= module name

SAVE_BREAKPOINT | SAVEB

file | f	= file_name
Scope Expression	
b	
offset	
overlay	
module	

CLEAR_BREAKPOINT | CB

Scope Expression
b
offset
overlay
module

EXAMPLES

DB b=2

SAVEB f=sqbp b=(1,2)

CB m=sqb

SET TRAPS

SET_TRAP | ST

trap_type | t	= OVERLAY
	= END
	= ABORT
	= INTERRUPT
	= INSTRUCTION
	= RJ
	= XJ
	= JUMP
	= READ
	= WRITE
Scope Expression	(default is entire program)
line | l	= line_numbers
location | loc	= absolute_addresses
offset | o	= relative to module
variable | var	= name
overlay | ovl	= overlay name (n,n)
module | m	= module name
proc | p	= CYBIL procedure name
collect | c	- Start Procedure
Procedure commands separated by semi-colons or EOLs	
collect_end | ce	- End Procedure

EXAMPLES

ST t=jump l=51..53

ST t=write var=b m=sqb proc=main

TRAP CONTROL

DISPLAY_TRAP | DT

type	= Same as on ST command
Scope Expression	
line | l	= line_numbers
label	= name
location | loc	= absolute addresses
entrypoint | e	= entry point name
t	= trap_numbers
overlay | ovl	= overlay number (n,n)
module | m	= name
proc | p	= CYBIL procedure name

SAVE=TRAP | SAVET

file | f	= file_name
type	
Scope Expression	
t	
overlay	
module	
proc	

CLEAR_TRAP | CT

type
Scope Expression
t
overlay
module
proc

EXAMPLES

DT type=jump

SAVET f=sqtr t=3

CT t=3

DEBUGGER PROCEDURES

SET_PROCEDURES | SP

procedure | pr = name
collect | c
Procedure commands separated by semicolons or EOLs.
collect_end | ce

DISPLAY_PROCEDURE | DP

procedure | pr = names

SAVE_PROCEDURE | SAVEP

file | f = name
procedure | pr = names

CLEAR_PROCEDURE | CP

procedure | pr

EXAMPLES

SP pr=one C; DV var=base m=sqa; proc=main; PA; CE

SP pr=two CV var=#vl value=#v1+1 m=sqb; PA; CE

SAVEP f=sqpr pr=(one,two)

CP (one,two)

VARIABLE CONTROL

DISPLAY_VARIABLE | DV

```
var                 = name
default display formats:
    integers - decimal
    ordinals, characters and strings - ascii
    booleans - true/false
    pointers and sequences - octal
    sets, arrays and records - type of components
format | f          = oct | dec | hex
module | m         = module_name
proc | p            = CYBIL procedure name
```

CHANGE_VARIABLE | CV

```
var                 = name
value | v           = new_value (must match type of var)
module
proc
```

EXAMPLES

DV base

CV base [20] v=1

MEMORY CONTROL

DISPLAY_MEMORY | DM

Address Expression	
line | l	= n
entrypoint | e	= name
location | loc	= n
offset | o	= offset in base module
module | m	= base module name
format | f	= oct | dec | hex | adr
numlocs | n	= number of locations [1]
indirect | i	

FORWARD | FW

numlocs
format

BACKWARD | BW

numlocs
format

CHANGE_MEMORY | CM

Address Expression
module
offset
value | v
numlocs
indirect

EXAMPLES

DM offset=10,n=10

FW 10

REGISTER CONTROL

DISPLAY_REGISTERS | DR

p | fl | a | b | x = n
format | f = oct | dec | adr | hex
indirect | i

CHANGE_REGISTERS | CR

a | b | x = n
value | v = value_to _store
indirect | i

EXAMPLES

DR a

CR a=1 v=2100(8)

SEQUENCE COMMANDS

PAUSE | PA

MESSAGE | ME 'text'

GO address=n

SKIPIF | S v1 op v2

LABEL | LA n=name

GOTO label=label_name

READ | R f=proc_file_name

READ | R pr=proc_names

EXAMPLES

ST t=write var=j m=SQB c

SKIPIF j=20; GO

MESSAGE 'j is too large'

READ file=SQPR; READ pr=two; ec

INTERACTIVE COMMANDS

EXECUTE | EXEC

Address Expression
  line | l = n
  location | loc = n
offset | o = n

TRACE_BACK | TB

entrypoint | e = name

MOVE | M

source_line = n	destination_line = n	numlocs = n
source_offset = name	destination_offset = name	
source_location = addr	destination_location = addr	

HELP | H

subject
command_name

SAVE_ALL | SAVEA

file = file_name

CHECKPOINT | CK

file = file_name

QUIT

normal
abort

ENVIRONMENT COMMANDS

SET_INTERPRET | SI

CLEAR_INTERPRET | CI

SET_VETO | SVE

CLEAR_VETO | CVE

SET_OUTPUT | SO lo=options

SET_AUXILIARY | SAUX f=file_name lo=options

CLEAR_AUXILIARY | CAUX

DISPLAY_MAP | DMAP module=name ovl=(n,n)

DISPLAY_DEFAULTS | DD

DISPLAY_STATUS | DS

CHANGE_DEFAULTS | CD

module = name
proc = name
overlay = (n,n)

APPENDIX A

ASCII CHARACTER SET

ASCII CHARACTER SET

HEX — Hexadecimal (8-bit) representation of the character
M — Cursor will be moved (Yes or No)
C — Character will be displayed on screen (Yes or No)
G — Graphic representation of character displayed

NAME	VALUE	DESCRIPTION	HEX	M	C	G
NUL	CHR(0)	NULL	00	N	N	
SOH	CHR(1)	START OF HEADING	01	N	N	
STX	CHR(2)	START OF TEXT	02	N	N	
ETX	CHR(3)	END OF TEXT	03	Y	Y	
EOT	CHR(4)	END OF TRANSMISSION	04	N	N	
ENQ	CHR(5)	ENQUIRY	05	N	N	
ACK	CHR(6)	ACKNOWLEDGE	06	N	N	
BELL	CHR(7)	BELL	07	N	N	
BS	CHR(8)	BACKSPACE (CURSOR LEFT)	08	Y	N	
HT	CHR(9)	HORIZONTAL TABULATION	09	N	N	
LF	CHR(10)	LINE FEED	0A	Y	N	
VT	CHR(11)	VERTICAL TABULATION	0B	N	N	
FF	CHR(12)	FORM FEED	0C	N	N	
CR	CHR(13)	CARRIAGE RETURN	0D	Y	N	
SO	CHR(14)	SHIFT OUT (BLACK ON WHITE)	0E	Y	Y	
SI	CHR(15)	SHIFT IN (WHITE ON BLACK)	0F	Y	Y	
DLE	CHR(16)	DATA LINK ESCAPE	10	N	N	
DC1	CHR(17)	DEVICE CONTROL 1	11	N	N	
DC2	CHR(18)	DEVICE CONTROL 2	12	N	N	
DC3	CHR(19)	DEVICE CONTROL 3	13	N	N	
DC4	CHR(20)	DEVICE CONTROL 4	14	N	N	
SKIP	CHR(21)	CURSOR RIGHT (SKIP)	15	Y	N	
LCLR	CHR(22)	LINE CLEAR	16	Y	N	
ETB	CHR(23)	END OF TRANSMISSION BLOCK	17	N	N	
CLR	CHR(24)	CLEAR SCREEN	18	Y	N	
RSET	CHR(25)	RESET CURSOR (TO HOME)	19	Y	N	
CUP	CHR(26)	CURSOR UP	1A	Y	N	
ESC	CHR(27)	ESCAPE	1B	N	N	
FS	CHR(28)	FILE SEPARATOR	1C	N	N	
GS	CHR(29)	GROUP SEPARATOR	1D	N	N	

NAME	VALUE	DESCRIPTION	HEX	M	C	G
RS	CHR(30)	RECORD SEPARATOR	1E	N	N	
US	CHR(31)	UNIT SEPARATOR	1F	N	N	
SP	CHR(32)	SPACE	20	Y	N	
EX	CHR(33)	EXCLAMATION POINT	21	Y	Y	!
DQ	CHR(34)	QUOTATION MARKS	22	Y	Y	"
PS	CHR(35)	NUMBER SIGN	23	Y	Y	#
DS	CHR(36)	DOLLAR SIGN	24	Y	Y	$
PER	CHR(37)	PERCENT	25	Y	Y	%
AMP	CHR(38)	AMPERSAND	26	Y	Y	&
RAP	CHR(39)	APOSTROPHE	27	Y	Y	'
LP	CHR(40)	OPENING PARENTHESIS	28	Y	Y	(
RP	CHR(41)	CLOSING PARENTHESIS	29	Y	Y	)
AS	CHR(42)	ASTERISK	2A	Y	Y	*
PL	CHR(43)	PLUS	2B	Y	Y	+
CM	CHR(44)	COMMA	2C	Y	Y	,
MI	CHR(45)	HYPHEN (MINUS)	2D	Y	Y	-
PERD	CHR(46)	PERIOD (DECIMAL POINT)	2E	Y	Y	.
SOL	CHR(47)	SLANT	2F	Y	Y	/
ZER	CHR(48)	ZERO	30	Y	Y	0
ONE	CHR(49)	ONE	31	Y	Y	1
TWO	CHR(50)	TWO	32	Y	Y	2
THR	CHR(51)	THREE	33	Y	Y	3
FOUR	CHR(52)	FOUR	34	Y	Y	4
FIVE	CHR(53)	FIVE	35	Y	Y	5
SIX	CHR(54)	SIX	36	Y	Y	6
SEV	CHR(55)	SEVEN	37	Y	Y	7
EGH	CHR(56)	EIGHT	38	Y	Y	8
NIN	CHR(57)	NINE	39	Y	Y	9
COL	CHR(58)	COLON	3A	Y	Y	:
SC	CHR(59)	SEMICOLON	3B	Y	Y	;
LT	CHR(60)	LESS THAN	3C	Y	Y	<
EQ	CHR(61)	EQUALS	3D	Y	Y	=
GT	CHR(62)	GREATER THAN	3E	Y	Y	>
QM	CHR(63)	QUESTION MARK	3F	Y	Y	?
AT	CHR(64)	COMMERCIAL AT	40	Y	Y	@
A	CHR(65)	UPPER CASE A	41	Y	Y	A
B	CHR(66)	UPPER CASE B	42	Y	Y	B
C	CHR(67)	UPPER CASE C	43	Y	Y	C
D	CHR(68)	UPPER CASE D	44	Y	Y	D
E	CHR(69)	UPPER CASE E	45	Y	Y	E
F	CHR(70)	UPPER CASE F	46	Y	Y	F
G	CHR(71)	UPPER CASE G	47	Y	Y	G
H	CHR(72)	UPPER CASE H	48	Y	Y	H
I	CHR(73)	UPPER CASE I	49	Y	Y	I

NAME	VALUE	DESCRIPTION	HEX	M	C	G
J	CHR(74)	UPPER CASE J	4A	Y	Y	J
K	CHR(75)	UPPER CASE K	4B	Y	Y	K
L	CHR(76)	UPPER CASE L	4C	Y	Y	L
M	CHR(77)	UPPER CASE M	4D	Y	Y	M
N	CHR(78)	UPPER CASE N	4E	Y	Y	N
O	CHR(79)	UPPER CASE O	4F	Y	Y	O
P	CHR(80)	UPPER CASE P	50	Y	Y	P
Q	CHR(81)	UPPER CASE Q	51	Y	Y	Q
R	CHR(82)	UPPER CASE R	52	Y	Y	R
S	CHR(83)	UPPER CASE S	53	Y	Y	S
T	CHR(84)	UPPER CASE T	54	Y	Y	T
U	CHR(85)	UPPER CASE U	55	Y	Y	U
V	CHR(86)	UPPER CASE V	56	Y	Y	V
W	CHR(87)	UPPER CASE W	57	Y	Y	W
X	CHR(88)	UPPER CASE X	58	Y	Y	X
Y	CHR(89)	UPPER CASE Y	59	Y	Y	Y
Z	CHR(90)	UPPER CASE Z	5A	Y	Y	Z
OB	CHR(91)	OPENING BRACKET	5B	Y	Y	[
BSH	CHR(92)	REVERSE SLANT	5C	Y	Y	\
CB	CHR(93)	CLOSING BRACKET	5D	Y	Y	]
CIR	CHR(94)	CIRCUMFLEX	5E	Y	Y	^
UND	CHR(95)	UNDERLINE	5F	Y	Y	_
LSQ	CHR(96)	GRAVE ACCENT	60	Y	Y	`
LCA	CHR(97)	LOWER CASE A	61	Y	Y	a
LCB	CHR(98)	LOWER CASE B	62	Y	Y	b
LCC	CHR(99)	LOWER CASE C	63	Y	Y	c
LCD	CHR(100)	LOWER CASE D	64	Y	Y	d
LCE	CHR(101)	LOWER CASE E	65	Y	Y	e
LCF	CHR(102)	LOWER CASE F	66	Y	Y	f
LCG	CHR(103)	LOWER CASE G	67	Y	Y	g
LCH	CHR(104)	LOWER CASE H	68	Y	Y	h
LCI	CHR(105)	LOWER CASE I	69	Y	Y	i
LCJ	CHR(106)	LOWER CASE J	6A	Y	Y	j
LCK	CHR(107)	LOWER CASE K	6B	Y	Y	k
LCL	CHR(108)	LOWER CASE L	6C	Y	Y	l
LCM	CHR(109)	LOWER CASE M	6D	Y	Y	m
LCN	CHR(110)	LOWER CASE N	6E	Y	Y	n
LCO	CHR(111)	LOWER CASE O	6F	Y	Y	o
LCP	CHR(112)	LOWER CASE P	70	Y	Y	p
LCQ	CHR(113)	LOWER CASE Q	71	Y	Y	q
LCR	CHR(114)	LOWER CASE R	72	Y	Y	r
LCS	CHR(115)	LOWER CASE S	73	Y	Y	s
LCT	CHR(116)	LOWER CASE T	74	Y	Y	t
LCU	CHR(117)	LOWER CASE U	75	Y	Y	u

NAME	VALUE	DESCRIPTION	HEX	M	C	G
LCV	CHR(118)	LOWER CASE V	76	Y	Y	v
LCW	CHR(119)	LOWER CASE W	77	Y	Y	w
LCX	CHR(129)	LOWER CASE X	78	Y	Y	x
LCY	CHR(130)	LOWER CASE Y	79	Y	Y	y
LCZ	CHR(131)	LOWER CASE Z	7A	Y	Y	z
OBR	CHR(123)	OPENING BRACE	7B	Y	Y	{
SOR	CHR(124)	VERTICAL LINE	7C	Y	Y	|
CBR	CHR(125)	CLOSING BRACE	7D	Y	Y	}
TL	CHR(126)	OVERLINE (TILDE)	7E	Y	Y	~
RO	CHR(127)	RUBOUT	7F	Y	Y	

CHR(128) .. CHR(255) PRINT AS A SPACE ON THE TERMINAL AND ARE UNASSIGNED ASCII GRAPHICS

APPENDIX B

I/O

EXAMPLES

I/O EXAMPLES

```
MODULE SEE;
?? SET (LIST := OFF) ??
*CALLC PXIOTYP
*CALLC LGZOPEN
*CALLC LGZCLOS
*CALLC LGZPUT
*CALLC LGZGET
?? RESET ??

  VAR
    OUT,
    INN: FILE,
    LENGTH: INTEGER,
    VALUE: INTEGER,
    MESSAGE: STRING (20);

  PROGRAM MAIN;

    LG#OPEN (OUT, 'OUTPUT', OLD#, OUTPUT#, ASIS#);
    LG#PUT (OUT, 'START OF JOB');

    VALUE := 345;
    STRINGREP (MESSAGE, LENGTH, VALUE);

    LG#PUT (OUT, MESSAGE);
    LG#PUT (OUT, 'END OF JOB');

    LG#CLOSE (OUT, ASIS#);
  PROCEND MAIN;
MODEND SEE.
/SES.GENCOMP SF=IOEX CYBCOMN
*     GENERATING COMPILE FILE  COMPILE
*     END GENCOMP     COMPILE <- BASE
/SES.CYBIL CC
*     COMPILING  COMPILE
*     END CYBIL     COMPILE -> LISTING, LGO
/SES.LINK170 CYBCLIB
*     END LINK170     LGOB
/LGOB
START OF JOB
 345
END OF JOB
LGOB.
```

I/O EXAMPLES

```
1SOURCE LISTING OF    SEE                                    NOS       CYBIL/CC
 1.0  8133 (      )            12/30/81    4:35 PM           PAGE 1

        0      1 MODULE SEE;
        0      3
        0      4 { PXIOTYP     CONTAINS CYBIL TYPE DECLARATIONS. }
        0      5
        0      6   TYPE
        0      7     FILE = ^CELL,
        0      8     FILE_STATUS = (NEW#, OLD#),
        0      9     FILE_MODE = (INPUT#, OUTPUT#, CONCURRENT#),
        0     10     FILE_ENCODING = (ASCII64#, ASCII612#, ASCII#),
        0     11     FILE_MARK = (DATA#, EOR#, EOF#, EOI#),
        0     12     FILE_POSITION = (FIRST#, ASIS#, LAST#);
        0     13
        0     14   CONST
        0     15     RETURN# = LAST#;
        0     16
        0     17   TYPE
        0     18     FILE_DISPOSITION = FIRST# .. RETURN#;
        0     19
        0     20   { I.E. (FIRST#, ASIS#, RETURN#) }
        0     21
        0     22 { LGZOPEN     OPENS LEGIBLE FILE AS LOCAL FILE. }
        0     23
        0     24   PROCEDURE [XREF] LG#OPEN (VAR LEGIBLE_FILE: FILE;
        0     25     FILE_NAME: STRING ( * );
        0     26     STATUS: FILE_STATUS;
        0     27     MODE: FILE_MODE;
        0     28     POSITION: FILE_POSITION);
        0     29
        0     30 { LGZCLOS     CLOSES LEGIBLE FILE. }
        0     31
        0     32   PROCEDURE [XREF] LG#CLOSE (LEGIBLE_FILE: FILE;
        0     33     DISPOSITION: FILE_DISPOSITION);
        0     34
        0     35 { LGZPUT     WRITES SOURCE STRING AS COMPLETE LINE TO LEG
IBLE FILE. }
        0     36
        0     37   PROCEDURE [XREF] LG#PUT (LEGIBLE_FILE: FILE;
        0     38     LINE: STRING ( * ));
        0     39
        0     40 { LGZGET     READS NEXT COMPLETE LINE FROM LEGIBLE FILE.
}
        0     41
        0     42   PROCEDURE [XREF] LG#GET (LEGIBLE_FILE: FILE;
        0     43     VAR NUMBER_OF_CHARACTERS_READ: INTEGER;
        0     44     VAR LINE: STRING ( * ));
```

```
MODULE SHOW;
?? SET (LIST := OFF) ??
*CALLC PXIOTYP
*CALLC LGZOPEN
*CALLC LGZCLOS
*CALLC LGZPUT
*CALLC LGZGET
*CALLC FZMARK
?? RESET ??

  PROGRAM MAIN;

    VAR
      I: 1 .. 80,
      NCHARS: INTEGER,
      MARK: FILE_MARK,
      STR: STRING (16),
      OUT: FILE,
      INPUT: FILE;

    LG#OPEN (INPUT, 'INPUT', OLD#, INPUT#, ASIS#);
    LG#OPEN (OUT, 'OUTPUT', OLD#, OUTPUT#, ASIS#);
    LG#GET (INPUT, NCHARS, STR);
    F#MARK (INPUT, MARK);
    WHILE (STR (1) <> ')') AND (MARK = DATA#) DO

      LG#PUT (OUT, STR(1,NCHARS));
      LG#GET (INPUT, NCHARS, STR);
      F#MARK (INPUT, MARK);
    WHILEND;

    LG#CLOSE (INPUT, ASIS#);
    LG#CLOSE (OUT, ASIS#);

  PROCEND MAIN;
MODEND SHOW.

/SES.GENCOMP SF=IOEX2 CYBCOMN
*     GENERATING COMPILE FILE  COMPILE
*     END GENCOMP    COMPILE <- BASE
/SES.CYBIL CC
*     COMPILING  COMPILE
*     END CYBIL    COMPILE -> LISTING, LGO
/SES.LINK170 CYBCLIB
*     END LINK170    LGOB
/LGOB
? STRING1
? STRING2
? STRING3
? END OF STRING
? )
STRING1
STRING2
STRING3
END OF STRING
LGOB.
```

APPENDIX C

EXERCISES

DAY 1

1. The following list contains several compilation errors. Fix them.

```
        0      1 MODULE ERROR3;
        0      2   TYPE
 *ERROR*       3     S = ARRAY [1..FINAL] OF INT;
        0      4   VAR
        0      5     I : INTEGER,
 *ERROR*       6   VAR
 *ERROR*       7     X : REAL;
        0      8
 *ERROR*       9   PROGRAM ERROR-PROC;
        0     10     VAR
 *ERROR*      11       I : 0...99999,
 *ERROR*      12       ARRAY =S;
        0     13
 *ERROR*      14     I := 3;
 *ERROR*      15     X := 10.0;
 *ERROR*      16     FOR I = 1 TO FINAL DO
 *ERROR*      17       S[I] = I + X;
        0     18     FOREND
 *ERROR*      19     VAR := S[1] + S[FINAL];
        0     20
        0     21 MODEND
```

2. Write and compile a program to initialize the following table (named INFO).

```
CONST
  LIM = 100;

TYPE
  REC = RECORD
    NAME: STRING <10>,
    AGE: 0 .. 99,
    NEXT_OF_KIN: ^REC;
  RECEND;

VAR
  INFO: ARRAY [1 .. LIM] OF REC;
```

DAY 1
CONT'D

3. For each of the structures described below, provide the required declarations and code needed to perform the specified initialization.

   a. array to contain prices for each of 300 items stocked by a small store. Initialize all entries to zero.

   b. array of 26 integers, indexed by the characters 'A' to 'Z' and initialized to all zeros.

   c. 10 x 10 matrix, initialized to one's on the diagonal and zero elsewhere.

   d. a record containing the following information:

      Name

      Address

      — street #
      — street name
      — city
      — state
      — zip code

   Initialize it to your name and address.

DAY 2

The following problems involve some simple I/O. I/O procedures will be provided in class.

1. Write procedures to initialize the structures described in the Day 1 problem #3c and d. Print the structures.

2. A program reads a card and calls a procedure. The procedure catagorizes the characters as alphabetic, digit or special. It prints the results.

3. Write a procedure to convert string to integer. Test it.

4. Write a procedure which inspects a set of attributes. The attributes are read, write and execute. Return normal status unless the set contains write and execute.

5. Write a function to compute powers of N. When called, the function is provided with two parameters: 1) the integer base (e.g. 2) and the integer power (e.g.10). The function should compute base ** power (e.g. 2**10). Test the function. Optionally, make the code recursive.

DAY 3

1. Write a procedure that will remove an entry from a linked list. Use the example from class as a base. You will be passed a pointer to the entry to be removed. You must restore the linkage and deallocate the entry.

2. Suppose that an EST (Equipment Status Table) entry is a bound variant record with the following fields.

   a. Ordinal of the entry (1 .. 256)

   b. ON/OFF

   c. UP/DOWN

   d. Pointer to CHT (Channel Table)

   e. Device type

      0 = none

      1 = disk

      2 = tape

      3 = CR

      4 = CP

      5 = CP

      6 = Mainframe

      7 = Multiplexer

   This field controls the following variants:

   disk : number of units ($n \leq 8$)

   tape : number of 7 track drives
   number of 9 track drives

   multiplexor : number of ports ($n \leq 256$)

   Create a dummy CHT. Create the EST as an array of pointers to EST entries. Allocate space for a tape entry. Initialize the fields.

DAY 4

1. Convert the Day 2 problem #2 module so that the procedure accepts an adaptable array. The program reads several cards and quits on a blank line.

2. Establish a sequence. Store some simple records in it. Write it to some file. In another module, read the file into a sequence and print it.

APPENDIX D

PROJECT

BLACKJACK RULES

1. Dealer deals two cards to player (both face down) and two to self (one up, one down).
   Value of cards are:
   ACE: 1 or 11
   King, Queen, Jack: 10
   All others: face value

2. If player has 21 with two cards, player wins double.

3. Player can get additional cards. If total exceeds 21, the player loses and the game is over.

4. When the player is through, the dealer can take cards following these rules:
   If total is 16 or under, dealer takes a card.
   If total is 17 or over, dealer does not take a card.

5. Now, if dealer's total is greater than 21, player wins.
   If player's total is greater than dealer's, player wins.
   If totals are equal, nobody wins.
   If dealer's total is greater than player's, player loses.

BLACKJACK MODIFICATIONS

1. Print Blackjack rules on request.

2. Check all input for validity. If input is wrong, give the player some help and an opportunity to resubmit.

3. Allow multiple players.
   a. Allocate the player records so they can be referenced by pointer.
   b. Give additional player a chance to start at the beginning of each round.
   c. Give players a chance to quit after each round.

4. Allow players to split pairs.
   If the player has a pair, he or she may get additional cards for each pair. It is like playing two hands at once; they are won and lost separately. Be sure that the player has enough money left to do this.

5. Allow player to double down.
   If the player has a total of 9, 10, or 11 with two cards, he or she can double the bet and take exactly one card. Winning and losing is evaluated as usual.

PARTICIPANT INFORMATION FORM

In order for our seminars/courses to be most effective, they need to take into account the characteristics, needs and objectives of the people who attend them. The information asked for below will assist us in keeping our presentations relevant to the participants and in developing and scheduling new presentations that will meet participant needs. Please complete this form and leave it with the presenter at the next break.

Seminar/Course Title _____ **Date of Presentation** _____

Name _____ **Field or Type of Business** _____

Title _____ **Years of Experience** _____

Business Address _____ **Supervisor's Title** _____

_____ **Last professional degree** _____

_____

List your three primary objectives in attending this seminar.

1. _____
2. _____
3. _____

Will this course/seminar be credited toward certification/training requirements? _____

Rank in order of importance in your choice of this seminar session.

Instructor _____ **Date** _____ **Location** _____ **Employer's Preference** _____

Previous courses/seminars attended relating to this topic.

1. _____
2. _____
3. _____

Topics for additional courses/seminars in which you would be interested.

1. _____
2. _____
3. _____

PARTICIPANT INFORMATION FORM

What trade journals/magazines do you regularly read or subscribe to in order to keep abreast in your profession?

1. _____
2. _____
3. _____

How did you become aware of this course/seminar?

Schedule/Catalogue _____,

Direct Mail Brochure _____,

Recommendations of Supervisor _____,

Recommendation of Colleague _____,

Corporate Training Department _____,

Other _____ .

COMMENT SHEET

MANUAL TITLE: NOS CYBIL

PUBLICATION NO.: DA3800-1 **REVISION:** B

NAME: _____

COMPANY: _____

STREET ADDRESS: _____

CITY: _____ **STATE:** _____ **ZIP CODE:** _____

This form is not intended to be used as an order blank. Control Data Corporation welcomes your evaluation of this manual. Please indicate any errors, suggested additions or deletions, or general comments below (please include page number references).

CUT ALONG LINE

AA3419 REV 4/79 PRINTED IN U.S.A.

NO POSTAGE STAMP NECESSARY IF MAILED IN U.S.A.

FOLD ON DOTTED LINES AND STAPLE

STAPLE

STAPLE

FOLD

FOLD

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 8241 MINNEAPOLIS, MINN.

POSTAGE WILL BE PAID BY

CONTROL DATA CORPORATION
National Coordinator
Bloomington Facility (MNA02B)
5001 West 80th Street
Bloomington, Minnesota 55437

CUT ALONG LINE

FOLD

FOLD

NOS/CYBIL

DAY 1 NR 2.

11

```
/COPY,LISTING
1SOURCE LISTING OF   MOD1                                    NOS      CYBIL/CC 1
.0  8209  (     )      JUNE 16, 1982      2:11 PM       PAGE 1

        0      1 MODULE MOD1;
        0      2
        0      3   CONST
        0      4     LIM = 100;
        0      5
        0      6   TYPE
        0      7     REC = RECORD
        0      8       NAME: STRING (10),
        0      9       AGE: 0 .. 99,
        0     10       NEXT_OF_KIN: ^REC,
        0     11
        0     12   RECEND;
        0     13
        0     14   VAR
        0     15       INFO: ARRAY [1 .. LIM] OF REC := [REP LIM OF ['',
0, NIL]];
        0     16
        0     17 PROGRAM MAIN;
        0     18
      630     19 PROCEND MAIN;
      630     20 MODEND MOD1;
0
    **** NO DIAGNOSTICS
 EOI ENCOUNTERED.
/
```

OR AN ALTERNATE VERSION

```
/COPY,LISTING
1SOURCE LISTING OF   MOD1                                    NOS      CYBIL/CC 1
.0  8209  (     )      JUNE 16, 1982      2:44 PM       PAGE 1

        0      1 MODULE MOD1;
        0      2
        0      3   CONST
        0      4     LIM = 100;
        0      5
        0      6   TYPE
        0      7     REC = RECORD
        0      8       NAME: STRING (10),
        0      9       AGE: 0 .. 99,
        0     10       NEXT_OF_KIN: ^REC,
        0     11     RECEND;
        0     12
        0     13   VAR
        0     14   INFO: ARRAY [1 .. LIM] OF REC;
        0     15
        0     16   PROGRAM MAIN;
        0     17
      630     18     VAR
      630     19       I : INTEGER;
      630     20
      630     21     FOR I := 1 TO LIM DO
      635     22       INFO [I].NAME := '';
      646     23       INFO [I].AGE := 0;
      652     24       INFO [I].NEXT_OF_KIN := NIL;
      657     25     FOREND;
      662     26   PROCEND MAIN;
      662     27 MODEND MOD1;
0
    **** NO DIAGNOSTICS
 EOI ENCOUNTERED.
```

DAY 1: NR 3a.

```
/SES.CYBFORM THREEA
REVERT.     END CYBFORM THREEA
/SES.CYBIL I=THREEA CC
*    COMPILING  THREEA
REVERT.     END CYBIL THREEA -> LISTING, LGO
/REWIND,LISTING
REWIND,LISTING.
/COPY,LISTING
1SOURCE LISTING OF   THREE_A                          NOS       CYBIL/CC 1
.0  8209  (      )       JUNE 16, 1982       4:09 PM       PAGE 1

        0      1 MODULE THREE_A;
        0      2
        0      3   TYPE
        0      4     ITEMS = ARRAY [1 .. 300] OF REAL;
        0      5
        0      6   VAR
        0      7     PRICES: ITEMS := [REP 300 OF 0.0];
        0      8
        0      9   PROGRAM MAIN;
      464     10   PROCEND MAIN;
      464     11 MODEND THREE_A;
0
    **** NO DIAGNOSTICS
 EOI ENCOUNTERED.
/
```

AN ALTERNATE VERSION.

```
/SES.CYBFORM THREEA1
REVERT.     END CYBFORM THREEA1
/SES.CYBIL I=THREEA1 CC
*    COMPILING  THREEA1
REVERT.     END CYBIL THREEA1 -> LISTING, LGO
/REWIND,LISTING
REWIND,LISTING.
/COPY,LISTING
1SOURCE LISTING OF   THREE_A1                         NOS       CYBIL/CC 1
.0  8209  (      )       JUNE 16, 1982       4:17 PM       PAGE 1

        0      1 MODULE THREE_A1;
        0      2
        0      3   TYPE
        0      4     ITEMS = ARRAY [1 .. 300] OF REAL;
        0      5
        0      6   VAR
        0      7     PRICES: ITEMS;
        0      8
        0      9   PROGRAM MAIN;
        0     10
      465     11     VAR
      465     12       I: INTEGER;
      465     13
      465     14     FOR I := 1 TO 300 DO
      472     15       PRICES [I] := 0.0;
      477     16     FOREND;
      502     17   PROCEND MAIN;
      502     18 MODEND THREE_A1;
0
    **** NO DIAGNOSTICS
 EOI ENCOUNTERED.
/
```

DAY 1; NR 3b

```
/SES.CYBFORM THREEB
REVERT.     END CYBFORM THREEB
/SES.CYBIL I=THREEB CC
*     COMPILING  THREEB
REVERT.     END CYBIL THREEB -> LISTING, LGO
/REWIND,LISTING
REWIND,LISTING.
/COPY,LISTING
1SOURCE LISTING OF    THREE_B                          NOS       CYBIL/CC 1
.0  8209  (      )       JUNE 17, 1982       9:54 AM       PAGE 1

         0      1 MODULE THREE_B;
         0      2
         0      3   TYPE
         0      4     VECTOR = ARRAY ['A' .. 'Z'] OF INTEGER;
         0      5
         0      6   VAR
         0      7     CHAR_COUNT: VECTOR := [REP 26 OF 0];
         0      8
         0      9   PROGRAM MAIN;
        42     10   PROCEND MAIN;
        42     11 MODEND THREE_B;
0
    **** NO DIAGNOSTICS
 EOI ENCOUNTERED.
/
```

AN ALTERNATE VERSION

```
/GET,THREEB1
/SES.CYBIL I=THREEB1 CC
*     COMPILING  THREEB1
REVERT.     END CYBIL THREEB1 -> LISTING, LGO
/REWIND,LISTING
REWIND,LISTING.
/COPY,LISTING
1SOURCE LISTING OF    THREE_B1                         NOS       CYBIL/CC 1
.0  8209  (      )       JUNE 17, 1982      10:16 AM       PAGE 1

         0      1 MODULE THREE_B1;
         0      2
         0      3   TYPE
         0      4     VECTOR = ARRAY ['A' .. 'Z'] OF INTEGER;
         0      5
         0      6   VAR
         0      7     CHAR_COUNT: VECTOR;
         0      8
         0      9   PROGRAM MAIN;
         0     10
        42     11     VAR
        42     12       CH: CHAR;
        42     13
        42     14   /LABEL_1/
        46     15     FOR CH := 'A' TO 'Z' DO
        50     16       CHAR_COUNT [CH] := 0;
        54     17     FOREND /LABEL_1/;
        54     18
        57     19   PROCEND MAIN;
        57     20 MODEND THREE_B1;
0
    **** NO DIAGNOSTICS
 EOI ENCOUNTERED.
/BYE
```

DAY 2; NR 2

```
/copy,test2
MODULE day_2_2;
?? SET (LIST := OFF) ??
*callc pxiotyp
*callc lszopen
*callc lszclos
*callc lszput
*callc lszset
*callc fzmark
?? RESET ??

  VAR
    codes: array [1 .. 80] of string (10),
    infile,
    outfile: file,
    in_string: string (80),
    length: integer,
    i: integer;

  PROGRAM main;
    ls#open (infile, 'INPUT', old#, input#, asis#);
    ls#open (outfile, 'OUTPUT', old#, output#, asis#);
    ls#set (infile, length, in_string);
    FOR i := 1 TO length DO
      CASE in_string (i) OF
      = 'A' .. 'Z', 'a' .. 'z' =
        codes [i] := 'alphabetic';
      = '0' .. '9' =
        codes [i] := 'digit';
      = '!' .. '/', ':' .. '@', '[' .. '`', '{' .. '~' =
        codes [i] := 'special';
      CASEND;
    FOREND;
    ls#put (outfile, in_string);
    FOR i := 1 TO length DO
      ls#put (outfile, codes [i]);
    FOREND;
    ls#close (infile, asis#);
    ls#close (outfile, asis#);
  PROCEND main;
MODEND day_2_2;
 EOI ENCOUNTERED.
```

```
/ses.sencomp sf=test2 cybccmn
*    GENERATING COMPILE FILE  COMPILE
REVERT.    END GENCOMP    COMPILE <- PDGWORK
/ses.cybil cc
*    COMPILING  COMPILE
REVERT.    END CYBIL COMPILE -> LISTING, LGO
/ses.link170 cybclib
REVERT.    END LINK170    LGOB
/lgob
? abc123/$@shi
abc123/$@shi

alphabetic
alphabetic
alphabetic
digit
digit
digit
special
special
special
alphabetic
alphabetic
alphabetic
LGOB.
/
```

day 2,#2 {using NOS/VE}

```
/EIAF
WELCOME TO NOS/VE R00 1.11.17/1R.  MARCH 7, 1983.  7:29 PM.
PROCESSING SYSTEM PROLOG.
 SYSNOTE UPDATED 83/03/07 14.10 PM.
 SEE SYSNOTE UNDER UN = LIBRARY.
PROCESSING USER PROLOG.
/GETF DAY22
/COPF DAY22
MODULE DAY2_2;
{}{}
*COPY IODCL
{}{}
  PROGRAM MAIN;
  VAR
    C : CHAR,
    I : INTEGER,
    LNG : INTEGER,
    NBLANK,NLETTER,NDIGIT,NOTHER : INTEGER,
    LAST_CARD : BOOLEAN;
  FOPEN;
  OMSG := ' TYPE IN UP TO 80 CHAR WHEN PROMPTED';
  AMP$PUT_NEXT(OFID,^OMSG,#SIZE(OMSG),OFBA,STAT);
  OMSG := ' A BLANK LINE WILL END PROGRAM';
  AMP$PUT_NEXT(OFID,^OMSG,#SIZE(OMSG),OFBA,STAT);
  OMSG := ' A CR WILL DO AS WELL';
  AMP$PUT_NEXT(OFID,^OMSG,#SIZE(OMSG),OFBA,STAT);
  LAST_CARD := FALSE;
  /LOOP/
    WHILE NOT LAST_CARD DO
      AMP$GET_NEXT(IFID,^IMSG,#SIZE(IMSG),ITC,IFBA,IFPOS,STAT);
      IF ITC = 0 THEN
        EXIT /LOOP/;
      IFEND;
      STRINGREP(OMSG,LNG,' ',ITC:3,
        ' CHARACTERS READ OF WHICH ARE        ');
      AMP$PUT_NEXT(OFID,^OMSG,#SIZE(OMSG),OFBA,STAT);
      NBLANK := 0;
      NLETTER := 0;
      NDIGIT := 0;
      NOTHER := 0;
      FOR I := 1 TO ITC DO
        C := IMSG(I);
        CASE C OF
        =' '=
          NBLANK := NBLANK+1;
        ='0'..'9'=
          NDIGIT := NDIGIT+1;
        ='a'..'z'=
          NLETTER := NLETTER+1;
        ='A'..'Z'=
          NLETTER := NLETTER+1;
        ELSE
          NOTHER := NOTHER+1;
        CASEND;
      FOREND;
```

```
        STRINGREP(OMSG,LNG,'  ',NBLANK:3,' BLANKS':35);
        AMP$PUT_NEXT(OFID,^OMSG,#SIZE(OMSG),OFBA,STAT);
        STRINGREP(OMSG,LNG,'  ',NLETTER:3,' LETTERS':35);
        AMP$PUT_NEXT(OFID,^OMSG,#SIZE(OMSG),OFBA,STAT);
        STRINGREP(OMSG,LNG,'  ',NDIGIT:3,' DIGITS':35);
        AMP$PUT_NEXT(OFID,^OMSG,#SIZE(OMSG),OFBA,STAT);
        STRINGREP(OMSG,LNG,'  ',NOTHER:3,' OTHER CHARACTERS':35);
        AMP$PUT_NEXT(OFID,^OMSG,#SIZE(OMSG),OFBA,STAT);
        IF ITC = NBLANK THEN
          LAST_CARD := TRUE;
        IFEND;
      WHILEND;
    FCLOSE;
    PROCEND MAIN;
  MODEND DAY2_2;
/GETF BUILD
/COPF BUILD
PROC BUILD(
INPUT,INP,I:FILE=$REQUIRED)
IF $FILE(EXERC,ASSIGNED) THEN
  DELF EXERC
IFEND
IF NOT $FILE($VALUE(INPUT),ASSIGNED) THEN
  GETF $VALUE(INPUT)
IFEND
COPF $VALUE(INPUT) EXERC
IF $FILE(IODCL,ASSIGNED) THEN
  DELF IODCL
IFEND
IF $FILE(COMP,ASSIGNED) THEN
  DELF COMP
IFEND
IF NOT  $FILE(IODECL,ASSIGNED) THEN
  GETF IODECL
IFEND
CRESL
SCU B=RESULT R=TDCL
CRED D=IODCL S=IODECL M=IODCL
CRED D=EXERC S=EXERC M=EXERC
EXPD D=EXERC C=COMP AB=$SYSTEM.OSF$PROGRAM_INTERFACE_LIBRARY
END WL=FALSE
PROCEND BUILD
/GETF IODECL
/COPF IODECL

?? SET(LIST:=OFF) ??
*COPYC AMP$OPEN
*COPYC AMP$CLOSE
*COPYC AMP$GET_NEXT
*COPYC AMP$PUT_NEXT
?? RESET ??
```

```
{}
VAR
  ITC : AMT$TRANSFER_COUNT,
  IFPOS : AMT$FILE_POSITION,
  IFBA,OFBA : AMT$FILE_BYTE_ADDRESS,
  ILFN,OLFN : AMT$LOCAL_FILE_NAME,
  IFID,OFID : AMT$FILE_IDENTIFIER,
  IMSG : STRING(80),
  OMSG : STRING(40),
  STAT : OST$STATUS,
  IATTR_P,OATTR_P : AMT$FILE_ACCESS_SELECTIONS;
{}
PROCEDURE FOPEN;
ALLOCATE IATTR_P : [1..2];
IATTR_P^[1].KEY := AMC$ACCESS_MODE;
IATTR_P^[2].KEY := AMC$OPEN_POSITION;
ALLOCATE OATTR_P : [1..2];
OATTR_P^[1].KEY := AMC$ACCESS_MODE;
OATTR_P^[2].KEY := AMC$OPEN_POSITION;
ILFN := '$INPUT';
IATTR_P^[1].ACCESS_MODE := $PFT$USAGE_SELECTIONS [PFC$READ];
IATTR_P^[2].OPEN_POSITION := AMC$OPEN_AT_BOI;
AMP$OPEN(ILFN,AMC$RECORD,IATTR_P,IFID,STAT);
OLFN := '$OUTPUT';
OATTR_P^[1].ACCESS_MODE := $PFT$USAGE_SELECTIONS [PFC$APPEND];
OATTR_P^[2].OPEN_POSITION := AMC$OPEN_AT_EOI;
AMP$OPEN(OLFN,AMC$RECORD,OATTR_P,OFID,STAT);
OMSG := ' START OF PROGRAM';
AMP$PUT_NEXT(OFID,^OMSG,#SIZE(OMSG),OFBA,STAT);
PROCEND FOPEN;
{}
PROCEDURE FCLOSE;
OMSG := ' END OF PROGRAM';
AMP$PUT_NEXT(OFID,^OMSG,#SIZE(OMSG),OFBA,STAT);
AMP$CLOSE(IFID,STAT);
AMP$CLOSE(OFID,STAT);
PROCEND FCLOSE;
{}

/GETF GO
/COPF GO
PROC GO
WHEN PROGRAM_FAULT DO
  DISV OSV$STATUS
  COPF L
  CANCEL PROGRAM_FAULT
  EXIT_PROC
WHENEND
IF $FILE(B,ASSIGNED) THEN
  DELF B
IFEND
IF $FILE(L,ASSIGNED) THEN
  DELF L
IFEND
CYBIL I=COMP L=L B=B
B
DELF COMP
CANCEL PROGRAM_FAULT
PROCEND GO
```

```
/BUILD DAY22
--WARNING SC 620314--  LIBRARY FORMAT VERSION MISMATCH ON FILE :$SYSTEM.$SYSTEM.
OSF$PROGRAM_INTERFACE_LIBRARY.1.
/GO
START OF PROGRAM
TYPE IN UP TO 80 CHAR WHEN PROMPTED
A BLANK LINE WILL END PROGRAM
A CR WILL DO AS WELL
? A A B;◆+<MN65432#$%&'FADSTGEFDKKJ09LLLLL HAHA
 45 CHARACTERS READ OF WHICH ARE
   3 BLANKS
  26 LETTERS
   7 DIGITS
   9 OTHER CHARACTERS
? A ,1
  4 CHARACTERS READ OF WHICH ARE
   1 BLANKS
   1 LETTERS
   1 DIGITS
   1 OTHER CHARACTERS
?
  1 CHARACTERS READ OF WHICH ARE
   1 BLANKS
   0 LETTERS
   0 DIGITS
   0 OTHER CHARACTERS
END OF PROGRAM
/LOGOUT
PROCESSING USER EPILOG.
PROCESSING SYSTEM EPILOG.VEIAF   CONNECT TIME 00.05.03.
ILLEGAL APPLICATION, TRY AGAIN.
T13A05  - APPLICATION: BYE
LOGGED OUT.

HOST DISCONNECTED   CONTROL CHARACTER=(ESC)
ENTER INPUT TO CONNECT TO HOST
```

DAY 2 #4

```
REWIND,LISTING.
/copy,listing
1SOURCE LISTING OF   day_2_4                          NOS      CYBIL/CC 1.0  8213
  (      )      June  9, 1982      4:15 PM      PAGE 1

        0     1 MODULE day_2_4;
        0    42
        0    43   PROGRAM main;
        0    44
       30    45     TYPE
       30    46       status_record = record
       30    47         case normal: boolean of
       30    48         = TRUE =
       30    49           ,
       30    50         = FALSE =
       30    51           message: string (80),
       30    52         casend,
       30    53       recend,
       30    54       modes = (read_access, write_access, execute_access),
       30    55       mode_set = set of modes;
       30    56
       30    57     VAR
       30    58       set_modes: mode_set,
       30    59       out: file,
       30    60       status: status_record;
       30    61
       30    62     PROCEDURE inspect (current_mode_set: mode_set;
       30    63       VAR status: status_record);
       30    64       IF current_mode_set = $mode_set [write_access, execute_acc
ess] THEN
       44    65         status.normal := FALSE;
       45    66         status.message := '**ERROR** Modes set to write and exec
ute';
       52    67         RETURN;
       53    68       ELSE
       54    69         status.normal := TRUE;
       55    70       IFEND;
       55    71     PROCEND inspect;
       57    72     set_modes := $mode_set [write_access, execute_access];
       65    73     lg#open (out, 'output', old#, output#, asis#);
       73    74     inspect (set_modes, status);
       76    75     IF NOT status.normal THEN
       77    76       lg#put (out, status.message);
      104    77     ELSE
      105    78       lg#put (out, 'Modes set properly');
      112    79     IFEND;
      112    80     lg#close (out, asis#);
      114    81   PROCEND main;
      114    82 MODEND day_2_4;
0
    **** NO DIAGNOSTICS
 EOI ENCOUNTERED.
```

```
ses.gencomp sf=test3 cybccmn
    GENERATING COMPILE FILE  COMPILE
EVERT.    END GENCOMP    COMPILE <- PDGWORK
ses.cybil cc
    COMPILING  COMPILE
EVERT.    END CYBIL COMPILE -> LISTING, LGO
ses.link170 cybclib
EVERT.    END LINK170    LGOB
lgob
*ERROR** Modes set to write and execute
```

```
REWIND,*
   5 FILES PROCESSED.
/COPY,AYD2EX5
MODULE D2EX5;
*CALLC PXIOTYP
*CALLC LGZOPEN
*CALLC LGZCLOS
*CALLC LGZPUT
*CALLC LGZGET
*CALLC FZSABF
  ?? RESET ??

  FUNCTION POWER (X,
        Y: INTEGER): INTEGER;
    IF X = 1 THEN
      POWER := 1;
    ELSEIF Y = 1 THEN
      POWER := X;
    ELSE
      POWER := X * POWER (X, Y - 1);
    IFEND;
  FUNCEND POWER;

  PROGRAM D3EX5;

    VAR
      ST,
      STR: STRING (12),
      CAR: CHAR,
      NUM,
      K,
      I,
      J,
      L,
      N: INTEGER,
      MARK: FILE_MARK,
      INPUT,
      OUT: FILE;

    LG#OPEN (OUT, 'OUTPUT', OLD#, OUTPUT#, ASIS#);
    F#SABF (OUT);
    LG#PUT (OUT, 'ENTER INTEGER BASE ');
    LG#OPEN (INPUT, 'INPUT', OLD#, INPUT#, ASIS#);
    LG#GET (INPUT, NUM, STR);
    I := 0;
    FOR J := 1 TO NUM DO
      I := 10 * I + ($INTEGER (STR (J)) - $INTEGER ('0'));
    FOREND;
    LG#PUT (OUT, 'ENTER INTEGER POWER');
    LG#GET (INPUT, NUM, STR);
    K := 0;
    FOR J := 1 TO NUM DO
      K := 10 * K + ($INTEGER (STR (J)) - $INTEGER ('0'));
    FOREND;
    N := POWER (I, K);
    STRINGREP (ST, L, N);
    LG#PUT (OUT, ST);
    LG#CLOSE (OUT, ASIS#);
    LG#CLOSE (INPUT, ASIS#);
  PROCEND D3EX5;
MODEND D2EX5;
 EOI ENCOUNTERED.
/
```

DAY 2; NR. 5.

DAY 3 #1

Solution 3

#1

```
MODULE LINK_LIST;
 TYPE
   LINK = ^REC,
   REC  = RECORD
          VALUE : INTEGER,
          PTR   : LINK
   RECEND;
 VAR
   FIRST: LINK:= NIL;

 PROGRAM MAIN;
   VAR
   ELEMENT: ARRAY [1..10] OF REC,
     I: 1..10,
   POINTER_TO_GO,
     POINTER: LINK;

   FOR I:= 1 TO 10 DO
     ELEMENT [I] . VALUE := I;
   POINTER := #LOC(ELEMENT[I]);
   ADD(POINTER);
     IF I = 3 THEN POINTER_TO_GO := POINTER
     IFEND;
   FOREND;
   REMOVE(POINTER_TO_GO);
   TEST;
 PROCEND MAIN;

 PROCEDURE ADD (P: LINK);
   VAR
     TEMP: LINK;
   BEGIN
     TEMP:= FIRST;
     FIRST := P;
     P^.PTR := TEMP
   END;
 PROCEND ADD;

 PROCEDURE TEST;
   TYPE AS= STRING (*);
   PROCEDURE [XREF] OPEN;
   PROCEDURE [XREF] CLOSE;
   PROCEDURE [XREF] INTOUT (S: AS, I: INTEGER);
   VAR
     P: LINK,
     V: INTEGER;
   OPEN;
   P := FIRST;
   WHILE P <> NIL  DO
     V := P^.VALUE;
     INTOUT (' VALUE IS ',V);
     P := P^.PTR;
   WHILEND;
   CLOSE;
 PROCEND TEST;

 PROCEDURE REMOVE (P:LINK);
     PROCEDURE [XREF] ERROR;
   VAR
   R: LINK;
   R := FIRST;
 /L10/

   WHILE R <> NIL DO
     IF R^.PTR = P THEN EXIT/L10/;
       ELSE R := R^.PTR;
   IFEND;
   WHILEND;
   IF R = NIL THEN ERROR;
     ELSE R^.PTR := P^.PTR;
   IFEND;
PROCEND REMOVE;

MODEND;
```

```
VALUE IS        10
VALUE IS         9
VALUE IS         8
VALUE IS         7
VALUE IS         6
VALUE IS         5
VALUE IS         4
VALUE IS         2
VALUE IS         1
```

A START TO THE BLACKJACK PROBLEM

```
1          CYBIL/CC          V1.0 790712        SOURCE LISTING - BLACKJACK
           79-10-01     11:47:15    PAGE      1
0      0      1
       0      2 MODULE BLACKJACK;
       0      3 ?? SET (LIST := OFF) ??
       0     38 ?? RESET ??
       0     39
       0     40   CONST
       0     41     C#MAX_AMOUNT = 10000;
       0     42
       0     43   TYPE
       0     44     T#PLAYER = RECORD
       0     45       NAME: STRING (10),
       0     46       AMOUNT: 0 .. C#MAX_AMOUNT,
       0     47       BET: 0 .. C#MAX_AMOUNT,
       0     48       TOTAL: 0 .. 30,
       0     49       ALT_TOTAL: 0 .. 41,
       0     50       BLACK_JACK: BOOLEAN,
       0     51       LAST_CARD: STRING (5),
       0     52     RECEND,
       0     53
       0     54     T#DEALER = RECORD
       0     55       TOTAL: 0 .. 26,
       0     56       ALT_TOTAL: 0 .. 36,
       0     57       LAST_CARD: STRING (5),
       0     58     RECEND,
       0     59
       0     60     T#WHO = (ALL, TO_PLAYER, TO_DEALER),
      0     61
       0     62     T#MESSAGE = (M_NAME, M_BET, M_CONTINUE, M_HIT, M_DEALER_CARD,

       0     63       M_PLAYER_CARD, M_DEALER_HIDDEN_CARD, M_SIGNOFF, M_SHUFFLE,
~~INT);~~              m_amount);
       0     64
       0     65   VAR
       0     66     DEALER: T#DEALER := [0, 0, ' '],
       0     67     PLAYER: T#PLAYER := [' ', 1000, 0, 0, 0, FALSE, ' '],
      0     68     INP: FILE,
       0     69     OUT: FILE;
       0     70
```

```
     0    71   PROGRAM MAIN;
     0    72
   366    73     VAR
   366    74       FIRST_PASS: [STATIC] BOOLEAN := TRUE,
   366    75       NAME: STRING (80),
   366    76       CONTINUE: STRING (80),
   366    77       SAMOUNT: STRING (80),
   366    78       DECISION: STRING (80);
   366    79
   366    80     OPEN_FILES;
   374    81     WRITES (M_NAME);
   376    82     READS (NAME);
   402    83     PLAYER.NAME := NAME (1, 10);
   402    84
   404    85   /PLAY/
   404    86     REPEAT
   405    87       WRITES (M_AMOUNT);
   407    88       IF NOT FIRST_PASS THEN
   410    89         WRITES (M_CONTINUE);
   412    90         READS (CONTINUE);
0  416    91         IF CONTINUE (1) <> 'Y' THEN
   416    92           EXIT /PLAY/
   421    93         IFEND;
   422    94       ELSE
   423    95         FIRST_PASS := FALSE;
   423    96       IFEND;
   424    97       WRITES (M_BET);
   426    98       READS (SAMOUNT);
   432    99       CONVERTSI (SAMOUNT, PLAYER.BET);
   437   100       IF ((PLAYER.BET > PLAYER.AMOUNT) OR (PLAYER.BET < 1)) THEN

   437   101         EXIT /PLAY/
   442   102       IFEND;
   442   103
   443   104       DEAL (ALL);
   445   105       IF (PLAYER.ALT_TOTAL = 21) THEN
   446   106         PLAYER.BLACK_JACK := TRUE;
   447   107         RESOLVE;
   451   108         CYCLE /PLAY/;
   452   109       IFEND;
   452   110
   452   111     /DECIDE/
   452   112       WHILE (PLAYER.TOTAL <= 21) DO
   453   113         WRITES (M_HIT);
   456   114         READS (DECISION);
   462   115         IF DECISION (1) = 'Y' THEN
   465   116           DEAL (TO_PLAYER);
   467   117         ELSE
   470   118           EXIT /DECIDE/;
   471   119         IFEND;
   471   120       WHILEND /DECIDE/;
   472   121       RESOLVE;
   472   122       (PLAY)
   473   123     UNTIL (PLAYER.AMOUNT <= 0);
   475   124     WRITES (M_SIGNOFF);
   477   125     CLOSE_FILES;
   500   126   PROCEND MAIN;
   500   127 ?? EJECT ??
```

```
500   128
500   129   PROCEDURE OPEN_FILES;
502   130     LG#OPEN (INP, 'INPUT', OLD#, INPUT#, ASIS#);
514   131     LG#OPEN (OUT, 'OUTPUT', OLD#, OUTPUT#, ASIS#);
522   132     F#SABF (OUT);
524   133   PROCEND OPEN_FILES;
524   134
524   135   PROCEDURE CLOSE_FILES;
526   136     LG#CLOSE (INP, ASIS#);
534   137     LG#CLOSE (OUT, ASIS#);
536   138   PROCEND CLOSE_FILES;
536   139
536   140   PROCEDURE READS (VAR S: STRING ( * ));
536   141
540   142     VAR
540   143       LENGTH: INTEGER;
540   144
540   145     S (1, 10) := '          ';
556   146     LG#GET (INP, LENGTH, S);
565   147   PROCEND READS;
565   148
565   149   PROCEDURE WRITES (MSG_TYPE: T#MESSAGE);
565   150
567   151     VAR
567   152       S: STRING (40),
567   153       L: 1 .. 80;
567   154
567   155     S := '                                        ';
567   156
600   157     CASE MSG_TYPE OF
600   158     = M_NAME =
617   159       LG#PUT (OUT, 'WHAT IS YOUR NAME?');
624   160     = M_AMOUNT =
625   161       LG#PUTPART (OUT, FALSE, 'YOU HAVE $');
632   162       STRINGREP (S, L, PLAYER.AMOUNT);
643   163       LG#PUTPART (OUT, TRUE, S);
650   164     = M_BET =
651   165       LG#PUT (OUT, 'HOW MUCH DO YOU WANT TO BET?');
656   166     = M_CONTINUE =
657   167       LG#PUT (OUT, 'DO YOU WANT TO CONTINUE?');
664   168     = M_HIT =
665   169       LG#PUT (OUT, 'DO YOU WANT ANOTHER CARD?');
672   170     = M_SIGNOFF =
673   171       LG#PUTPART (OUT, FALSE, 'THANKS FOR PLAYING BLACKJACK, ');

700   172       LG#PUTPART (OUT, TRUE, PLAYER.NAME);
700   173
705   174     = M_PLAYER_CARD =
706   175       LG#PUTPART (OUT, FALSE, 'YOUR CARD ----- ');
713   176       LG#PUTPART (OUT, TRUE, PLAYER.LAST_CARD);
720   177     = M_DEALER_HIDDEN_CARD =
721   178       LG#PUT (OUT, 'DEALER CARD ----- HIDDEN');
726   179     = M_DEALER_CARD =
727   180       LG#PUTPART (OUT, FALSE, 'DEALER CARD ----- ');
734   181       LG#PUTPART (OUT, TRUE, DEALER.LAST_CARD);
741   182     = M_SHUFFLE =
742   183       LG#PUT (OUT, 'SHUFFLING CARDS');
```



```
747   184     CASEEND;
```

```
0     750   187
      750   188   PROCEDURE RESOLVE;
      752   189     IF PLAYER.BLACK_JACK THEN
      757   190       PLAYER.AMOUNT := PLAYER.AMOUNT + 2 * PLAYER.BET;
      763   191       PLAYER.BLACK_JACK := FALSE;
      764   192       RETURN;
      765   193     IFEND;
      765   194     IF PLAYER.ALT_TOTAL <= 21 THEN
      766   195       PLAYER.TOTAL := PLAYER.ALT_TOTAL;
      772   196     IFEND;
      773   197     IF PLAYER.TOTAL > 21 THEN
      774   198       PLAYER.AMOUNT := PLAYER.AMOUNT - PLAYER.BET;
     1001   199       RETURN;
     1002   200     IFEND;
     1002   201   /DEALING/
     1002   202     WHILE DEALER.TOTAL < 17 DO
     1003   203       CASE DEALER.ALT_TOTAL OF
     1003   204       = 0 .. 16 =
     1012   205         DEAL (TO_DEALER);
     1014   206       = 17 .. 21 =
     1015   207         DEALER.TOTAL := DEALER.ALT_TOTAL;
     1020   208         EXIT /DEALING/;
     1022   209       = 22 .. 31 =
     1023   210       CASEND;
     1024   211     WHILEND /DEALING/;
     1025   212     IF DEALER.TOTAL > 21 THEN
     1026   213       PLAYER.AMOUNT := PLAYER.AMOUNT + PLAYER.BET;
     1033   214     ELSE
     1033   215       ;
     1034   216       IF DEALER.TOTAL > PLAYER.TOTAL THEN
     1035   217         PLAYER.AMOUNT := PLAYER.AMOUNT - PLAYER.BET;
     1042   218       IFEND;
     1043   219       IF DEALER.TOTAL < PLAYER.TOTAL THEN
     1044   220         PLAYER.AMOUNT := PLAYER.AMOUNT + PLAYER.BET;
     1051   221       IFEND;
     1052   222     IFEND;
     1052   223   PROCEND RESOLVE;
     1052   224 ?? EJECT ??
```

```
0    1052   225
     1052   226   PROCEDURE CONVERTSI (S: STRING ( * ));
    1052   227     VAR A: 0 .. C#MAX_AMOUNT);
     1052   228
     1054   229     VAR
     1054   230       P,
     1054   231       I: 0 .. 80,
     1054   232       TENS: INTEGER,
     1054   233       DIGITSET: [STATIC] SET OF '0' .. '9' := ['0', '1', '2', '3
, '5',
     1054   234         '6', '7', '8', '9'];
     1054   235
     1054   236     P := 0;
     1075   237     WHILE S (P + 1) IN DIGITSET DO
     1116   238       P := P + 1;
     1122   239     WHILEND;
     1123   240     TENS := 1;
     1123   241     A := 0;
     1124   242     FOR I := P DOWNTO 1 DO
     1131   243       A := A + TENS * (ORD (S (I)) - ORD ('0'));
    1154   244       TENS := TENS * 10;
     1157   245     FOREND;
     1161   246   PROCEND CONVERTSI;
     1161   247 ?? EJECT ??
```

```
 1161   248
 1161   249   PROCEDURE DEAL (W: T#WHO);
 1161   250
 1163   251     VAR
 1163   252       VALUE: 1 .. 10;
 1163   253
 1163   254     CASE W OF
 1163   255     = ALL =
 1177   256       GETCARD (VALUE, PLAYER.LAST_CARD);
 1202   257       WRITES (M_PLAYER_CARD);
 1204   258       PLAYER.TOTAL := VALUE;
 1205   259       IF VALUE = 1 THEN
 1207   260         PLAYER.ALT_TOTAL := 11;
 1210   261       ELSE
 1211   262         PLAYER.ALT_TOTAL := VALUE;
 1212   263       IFEND;
 1213   264       GETCARD (VALUE, DEALER.LAST_CARD);
 1216   265       WRITES (M_DEALER_CARD);
 1220   266       DEALER.TOTAL := VALUE;
 1221   267       IF VALUE = 1 THEN
 1223   268         DEALER.ALT_TOTAL := 11;
 1224   269       ELSE
 1225   270         DEALER.ALT_TOTAL := VALUE;
 1226   271       IFEND;
 1227   272       DEAL (TO_PLAYER);
 1231   273       DEAL (TO_DEALER);
 1233   274     = TO_PLAYER =
 1234   275       GETCARD (VALUE, PLAYER.LAST_CARD);
 1237   276       WRITES (M_PLAYER_CARD);
 1241   277       PLAYER.TOTAL := PLAYER.TOTAL + VALUE;
 1245   278       PLAYER.ALT_TOTAL := PLAYER.ALT_TOTAL + VALUE;
 1251   279       IF (VALUE = 1) AND (PLAYER.ALT_TOTAL <= 11) THEN
 1257   280         PLAYER.ALT_TOTAL := PLAYER.ALT_TOTAL + 10;
 1264   281       IFEND;
 1265   282     = TO_DEALER =
 1266   283       GETCARD (VALUE, DEALER.LAST_CARD);
 1271   284       WRITES (M_DEALER_CARD);
 1273   285       DEALER.TOTAL := DEALER.TOTAL + VALUE;
 1277   286       DEALER.ALT_TOTAL := DEALER.ALT_TOTAL + VALUE;
 1303   287       IF (VALUE = 1) AND (DEALER.ALT_TOTAL <= 11) THEN
 1311   288         DEALER.ALT_TOTAL := DEALER.ALT_TOTAL + 10;
 1316   289       IFEND;
 1316   290
 1317   291     CASEND;
 1320   292   PROCEND DEAL;
 1320   293 ?? EJECT ??
```

```
0   1320   294
    1320   295   PROCEDURE GETCARD (VAR VALUE: 1 .. 10;
    1320   296      NAM: STRING (5));
    1320   297 ?? FMT (FORMAT := OFF) ??
    1320   298
    1322   299     VAR
    1322   300       DECK: [STATIC] ARRAY [1..52] OF RECORD
    1322   301         NAME: STRING(5);
    1322   302         VAL: 1..10;
    1322   303         USED: BOOLEAN;
    1322   304       RECEND :=
    1322   305       [['ACE  ',1,FALSE],['TWO  ',2,FALSE],['THREE',3,FALSE],
    1322   306       ['FOUR ',4,FALSE],['FIVE ',5,FALSE],['SIX  ',6,FALSE],
    1322   307       ['SEVEN',7,FALSE],['EIGHT',8,FALSE],['NINE ',9,FALSE],
    1322   308       ['TEN  ',10,FALSE],['JACK ',10,FALSE],
    1322   309       ['QUEEN',10,FALSE],['KING ',10,FALSE],
    1322   310       ['ACE  ',1,FALSE],['TWO  ',2,FALSE],['THREE',3,FALSE],
    1322   311       ['FOUR ',4,FALSE],['FIVE ',5,FALSE],['SIX  ',6,FALSE],
    1322   312       ['SEVEN',7,FALSE],['EIGHT',8,FALSE],['NINE ',9,FALSE],
    1322   313       ['TEN  ',10,FALSE],['JACK ',10,FALSE],
    1322   314       ['QUEEN',10,FALSE],['KING ',10,FALSE],
    1322   315       ['ACE  ',1,FALSE],['TWO  ',2,FALSE],['THREE',3,FALSE],
    1322   316       ['FOUR ',4,FALSE],['FIVE ',5,FALSE],['SIX  ',6,FALSE],
    1322   317       ['SEVEN',7,FALSE],['EIGHT',8,FALSE],['NINE ',9,FALSE],
    1322   318       ['TEN  ',10,FALSE],['JACK ',10,FALSE],
    1322   319       ['QUEEN',10,FALSE],['KING ',10,FALSE],
    1322   320       ['ACE  ',1,FALSE],['TWO  ',2,FALSE],['THREE',3,FALSE],
    1322   321       ['FOUR ',4,FALSE],['FIVE ',5,FALSE],['SIX  ',6,FALSE],
    1322   322       ['SEVEN',7,FALSE],['EIGHT',8,FALSE],['NINE ',9,FALSE],
    1322   323       ['TEN  ',10,FALSE],['JACK ',10,FALSE],
    1322   324       ['QUEEN',10,FALSE],['KING ',10,FALSE]];
    1322   325 ??FMT ( FORMAT := ON )??
    1322   326 ?? EJECT ??
```

```
 1322   327
 1322   328      VAR
 1322   329        I: 1 .. 53;
 1322   330        X: [STATIC] 1 .. 2000000 := 1;
 1322   331        Y: [STATIC] 1 .. 2000000 := 1;
 1322   332        S: 0 .. 2000000;
 1322   333        COUNT: [STATIC] 0 .. 52 := 52;
 1322   334        X_STARTER: 0 .. 0FF(16);
 1322   335
 1322   336      COUNT := COUNT - 1;
 1334   337      IF COUNT < 10 THEN
 1336   338        WRITES (M_SHUFFLE);
 1340   339        COUNT := 51;
 1341   340        FOR I := 1 TO 52 DO
 1345   341          DECK[I].USED := FALSE;
 1352   342        FOREND;
 1355   343      IFEND;
 1355   344      X := X + X;
 1361   345      IF X > 999997 THEN
 1363   346        X := X - 999997
 1363   347      IFEND;
 1371   348      Y := Y + Y;
 1375   349      IF Y > 999971 THEN
 1377   350        Y := Y - 999971
 1377   351      IFEND;
 1405   352      S := X + Y;
 1411   353      S := S DIV 10;
 1417   354      I := (S MOD 52) + 1;
 1426   355      WHILE DECK[I].USED DO
 1434   356        I := I + 1;
 1437   357        IF I > 52 THEN
 1441   358          I := 1;
 1442   359        IFEND;
 1443   360      WHILEND;
 1444   361      DECK[I].USED := TRUE;
 1450   362      VALUE := DECK[I].VAL;
 1456   363      NAM := DECK[I].NAME;
 1471   364    PROCEND GETCARD;
 1471   365 MODEND BLACKJACK;
```



```
 365 LINES COMPILED.
 NO COMPILATION ERRORS.
EOI ENCOUNTERED.
```