

A SEQUENTIAL PASCAL MANUAL FOR FORTRAN PROGRAMMERS

by

JERRY DEAN RAWLINSON

B. S., University of Illinois, 1964

---

A MASTER'S REPORT

submitted in partial fulfillment of the

requirements for the degree

MASTER OF SCIENCE


Department of Computer Science

KANSAS STATE UNIVERSITY

Manhattan, Kansas

1977

Approved by:

  
William J. Hankley  
Major Professor

LD  
2668  
R4  
1977  
R39  
C.2  
Document

## CONTENTS

|     |   |      |
|-----|---|------|
| 0.  | INTRODUCTION.....                             | 0.1  |
| 1.  | SIMPLE FEATURES.....                          | 1.1  |
| 2.  | CONTROL STRUCTURES.....                       | 2.1  |
| 3.  | SIMPLE DATA TYPES.....                        | 3.1  |
| 4.  | ARRAYS.....                                   | 4.1  |
| 5.  | RECORDS.....                                  | 5.1  |
| 6.  | PROGRAM AND PROCEDURE STRUCTURE.....          | 6.1  |
| 7.  | PARAMETERS.....                               | 7.1  |
| 8.  | INPUT/OUTPUT.....                             | 8.1  |
| 9.  | STANDARD PREFIX.....                          | 9.1  |
| 10. | RUNNING THE PROGRAM.....                      | 10.1 |
| 11. | DIFFERENCES BETWEEN "PASCAL" AND "SPASCAL"... | 11.1 |

# **ILLEGIBLE DOCUMENT**

**THE FOLLOWING  
DOCUMENT(S) IS OF  
POOR LEGIBILITY IN  
THE ORIGINAL**

**THIS IS THE BEST  
COPY AVAILABLE**

**THIS BOOK  
CONTAINS  
NUMEROUS PAGES  
WITH DIAGRAMS  
THAT ARE CROOKED  
COMPARED TO THE  
REST OF THE  
INFORMATION ON  
THE PAGE.**

**THIS IS AS  
RECEIVED FROM  
CUSTOMER.**



## TABLE OF FIGURES

As the main body of this report consists of nothing but slides, the table of contents and the first slide in each slide set form a tree structure for locating figures depicting particular subjects.

### EXAMPLE

Locate a slide showing operations that can be performed on arrays:

1. The table of contents indicates the array slide set begins on page 4.1.
2. Page 4.1 indicates that slide(s) showing operations on arrays begin on page 4.9.

## INTRODUCTION

### 1. PASCAL, SEQUENTIAL PASCAL, AND CONCURRENT PASCAL.

The programming language Pascal was developed by Niklaus Wirth and was specifically designed as a general purpose language which could be used for systematic programming. As a result, Pascal is very readable and can be used in top-down program design in which the final source language program still reflects the structured design and lends itself to systematic verification.<sup>5</sup>

Sequential Pascal (SPASCAL) refers to a specific implementation of Pascal defined by Per Brinch Hansen and Alfred Hartmann of California Institute of Technology for a PDP-11/45 computer. SPASCAL differs from Wirth's definition of Pascal in several areas because of restrictions and extensions required for the implementation.

Concurrent Pascal (CPASCAL) has also been defined by Brinch Hansen and was designed for structured programming of computer operating systems. CPASCAL is an extension of SPASCAL which allows concurrent processes, monitors and classes and was used by Brinch Hansen to write a single user operating system called SOLO for the PDP-11/45 computer.

### 2. KANSAS STATE UNIVERSITY IMPLEMENTATION OF PASCAL.

Kansas State University Department of Computer Science has implemented SPASCAL, CPASCAL, and SOLO as defined by Brinch Hansen for the INTERDATA 8/32 computer. This implementation was made as part of a research project investigating computer networks. Pascal was chosen because it enforces structured programming, contains very powerful data structures, is very readable, and the SPASCAL compiler can detect many errors which would not be found at compile time using other languages.

### 3. OBJECTIVE OF THIS REPORT.

This report is designed to serve as an instructional aid for introducing persons who can at least read FORTRAN programs to the Kansas State University implementation of SPASCAL. The design concept is to expand on the student's existing knowledge of FORTRAN so that

he may begin programming in SPASCAL as quickly as possible. This report is not designed to be either a user guide or a self-contained primer for SPASCAL. Rather, the report attempts to provide a set of primer examples which may be used in conjunction with a short course or series of lectures. The format of the examples should provide the student with an overview of SPASCAL in an effective and efficient manner.

#### 4. REPORT DESIGN.

This report has been designed to provide a top-down learning process for the student. The report consists of a series of slides for which the instructor must provide his own narrative. The slides are divided into logical learning sets, and the sets are ordered in the sequence in which they should be presented. Early examples in the slide sequence intentionally conceal some details of SPASCAL so that specific teaching points may be emphasized. Later examples in the sequence enumerate upon the basic concepts introduced earlier.

Each set of slides has been designed to provide the student with a quick overview of a particular subject while requiring a minimum of reading. Examples are used to allow the student to compare FORTRAN and SPASCAL so that he may quickly grasp similarities and differences between the two languages.

#### 5. INSTRUCTOR GUIDELINES.

As stated above, the slides in this report have been organized for sequential presentation. No narrative has been provided as it is assumed each instructor will desire to develop his own narrative to fit his particular teaching style.

It is recommended that each student be provided with a copy of this report so that he may make notes on individual slides and use the report for future reference when programming.

The following references should be available for supplemental study:

Pascal User Manual and Report (3)

Sequential Pascal Report (1)

KSU Pascal Editor (PEDIT) (4)

The Solo Operating System Job Interface (2)

WARNING: Students must be made aware of the differences which exist between SPASCAL and Pascal as defined in Pascal User Manual and Report. These differences have been summarized in Slide Set 11 and are also annotated in the Cross Reference Table in this section (page 0.1.5). All "PASCAL" examples shown in this report are based on the Brinch Hansen definition of SPASCAL as implemented at Kansas State University.

All FORTRAN examples in this report are based on ASCII FORTRAN. Students should be made aware of this as some of the FORTRAN examples may differ from what can be done on the particular FORTRAN implementation they have been working with.

Italic characters in all examples indicate either an error condition or a facility that is not included in the language.

Students may notice that all PASCAL program examples have been indented to emphasize program structure while the FORTRAN program examples have not been indented. This was done based on an assumption that most persons familiar with FORTRAN are accustomed to seeing FORTRAN programs which have not been indented. It should be pointed out that while SPASCAL does not require indentation, it is a good programming practice which improves program readability.

The Cross Reference Table in this section is designed to assist instructors in narrative preparation. The table matches each slide set with the supplemental reference which best describes elements within the slide set. The table also indicates differences between the supplemental reference and SPASCAL.

Most of the SPASCAL examples in this report have been compiled on the Kansas State University implementation of SPASCAL. A card deck containing all of the examples compiled is available through the Department of Computer Science, Kansas State University.

## 6. CONCLUSION.

After compiling the examples on the INTERDATA 8/32 computer, numerous changes were made based on differences discovered between the written manuals on which the report was based and the actual implementation. All examples in the report reflect the actual implementation on the INTERDATA 8/32. Any additional discrepancies noted by users of the report should be brought to the attention of the KSU department of Computer Science.

Individuals familiar with FORTRAN were provided draft copies of the report and asked to comment on it. Those who reviewed the report felt that it met the stated objective by providing a quick introduction to SPASCAL. As expected, the report does not stand alone. Individuals who knew only FORTRAN experienced some difficulty with new concepts such as enumeration types and records when reading through the report on their own. However, these individuals were able to understand the concepts very quickly when they were talked through the examples in the same manner as an instructor would. The comments provided by these individuals were very helpfull and resulted in several changes and additions in the final report to improve the clarity of examples.

Individuals familiar with FORTRAN will find SPASCAL to be a very versatile language due to the increased number of control structures and type declaration facilities available in the language. By using these facilities, SPASCAL can provide for a more direct solution to many problems than is possible in FORTRAN.

The primary disadvantages of the current implementation of SPASCAL are it's I/O limitations and the lack of many predefined functions that FORTRAN users are accustomed to. FORTRAN users may also find it inconvenient not to be able to assign, operate, or use as parameters mixed integer and real numbers/variables. However, this is an excellent software engineering feature as many otherwise difficult to find errors are avoided by forcing the individual writing a program to declare his intention to change type.

In summary, SPASCAL is a very flexible general purpose language which is outstanding for use in structured programming. This report will provide instructors with an aid which can significantly reduce the time required to introduce SPASCAL to individuals familiar with FORTRAN and should be applicable to situations in which users merely wish to teach a new language or a more academic situation such as within a formal structured programming course.

## REFERENCES

1. Brinch Hansen, P. and Hartmann, A. Sequential Pascal Report, Information Science, Californic Institute of Technology, 1975.
2. Brinch Hansen, P. The Solo Operating System Job Interface, Information Science, California Institute of Technology, 1975.
3. Jensen, K. and Wirth, N. Pascal User Manual and Report, Springer-Verlag, 1975.
4. Neal, D. KSU Pascal Editor, Department of Computer Science, Kansas State University, 1976.
5. Wirth, N. Systematic Programming: An Introduction, Prentice-Hall, 1973.

# CROSS REFERENCE TABLE

| SLIDE<br>GROUP | REFERENCES*   | EXCEPTIONS**   |
|----------------|---|--|
| 1              | 3 CHAP 1, PP 9-11<br>3 CHAP 0, PP 3-8<br>3 CHAP 2, PP 12-15<br><br>1 CHAP 1-5, PP 1-7 | NO <u>PACKED</u> , <u>LABEL</u> , <u>FILE</u><br>NO NESTED DECLARATIONS<br>NO SIN(X), COS(X),<br>ARCTAN(X), LN(X)<br>EXP(X), SQRT(X),<br>SQR(X), ADD(X),<br>COLN(X) EOF(X), OP |
| 2              | 3 CHAP 4, PP 21-33<br><br>1 CHAP 7, PP 11-16;<br>CHAP 9, PP 26                        | NO I/O CALLS<br>(WRITELN, READ, ETC.)  |
| 3              | 3 CHAP 3, PP 16-19<br>3 CHAP 5, PP 34-35<br>1 CHAP 7, PP 9-16                         | NO LABELS  |
| 4              | 3 CHAP 6, PP 36-41<br><br>1 CHAP 7, PP 17-18  | INDEX ,(...);<br>NO <u>PACK</u> , <u>UNPACK</u>  |
| 5              | 3 CHAP 7, PP 92-99<br><br>1 CHAP 7.4, PP 19-20  | NO ENUMERATION<br>WITHIN RECORDS   |
| 6              | 3 CHAP 11, PP 67-83<br>1 CHAP 11-12,<br>PP 29-35                                      | NO NESTED PROCEDURES   |
| 7              | 1 CHAP, PP 31-32  |  |
| 8              | 2 CHAP 4, PP 5-7<br>2 CHAP 8, PP 10   |  |

# TABLE (CONT)

|    |           |
|----|-----------|
| 9  | 2 PP 1-18 |
| 10 | 4         |

- \*1 SEQUENTIAL PASCAL REPORT
- 2 THE SOLO OPERATING SYSTEM  
JOB INTERFACE
- 3 PASCAL USER MANUAL AND REPORT
- 4 KSU PASCAL EDITOR

\*\*DIFFERENCES BETWEEN REFERENCE  
AND KSU IMPLEMENTATION OF  
SPASCAL



## PRELIMINARIES

PASCAL (OR SPASCAL)

MEANS SEQUENTIAL PASCAL

AS DISTINCT FROM CONCURRENT PASCAL

KSU IMPLEMENTATION:

- PORTED FROM BRINCH HANSEN'S  
PDP-11/45 IMPLEMENTATION AT CALIFORNIA INSTITUTE OF  
TECHNOLOGY (WHICH DIFFERS SLIGHTLY FROM THE PASCAL  
REPORT.)
- SPASCAL PROGRAMS RUN AS A JOB PROCESS UNDER SOLO (SINGLE  
USER OPERATING SYSTEM WRITTEN IN CPASCAL, FROM BRINCH  
HANSEN, ET.AL.)
- SOLO RUNS AS A TASK UNDER OS-32/MT (80K PARTITION)  
ON INTERDATA 8/32.
- CURRENTLY THE PASCAL SOURCE IS INTERPRETED. THE  
INTERPRETER IS WRITTEN IN INTERDATA COMMON MODE CAL.  
(A COMPILER IS BEING DEVELOPED.)
- MAXIMUM PROGRAM SIZE IS 36K BYTES OF SOURCE TEXT  
(IT CAN BE CHANGED)

## WHY PASCAL?

### IMMEDIATE

- PASCAL ENFORCES STRUCTURE.
  - IN PROGRAM FLOW
  - IN DATA TYPES
  - IN DATA STRUCTURES
  - IN INTRA-PROCEDURE COMMUNICATION
- THE COMPILER WILL CHECK THE STRUCTURE AND DETECT ERRORS- WHICH WOULD NOT BE FOUND AT COMPILE TIME USING OTHER LANGUAGES.
- PASCAL DATA STRUCTURES ARE VERY POWERFUL (TYPES CAN BE DEFINED BY THE PROGRAMMER)
- PASCAL IS VERY READABLE.

### FUTURE

- RESEARCHERS ARE BUILDING FORMAL VERIFIERS FOR PASCAL PROGRAMS
- PASCAL HAS A FORMAL SEMANTIC DEFINITION - PROGRAMS SHOULD PRODUCE THE SAME RESULTS UNDER ALL COMPILERS.

## COMPARISONS WITH FORTRAN

FORTRAN  
VERSION

PASCAL  
VERSION

ALL PASCAL PROGRAMS HAVE BEEN INDENTED TO SHOW DESIRED PROGRAMMING STYLE. INDENTATION MUST BE DONE BY THE PROGRAMMER, IT IS NOT DONE AUTOMATICALLY.

MANY OF THE SLIDES IN THIS PRESENTATION CONTAIN COMPARISONS OF FORTRAN AND PASCAL PROGRAM ELEMENTS. IF THE COMPARISON PROGRAMS FIT ON THE SAME SLIDE, THE FORTRAN EXAMPLE WILL BE ON THE LEFT AND THE PASCAL EXAMPLE WILL BE ON THE RIGHT, THE PROGRAMS ARE NOT ALL IDENTICAL, BUT THEY WILL ILLUSTRATE RELATED FEATURES.

*ITALIC CHARACTERS SUCH AS THIS INDICATE AN ERROR CONDITION OR A FACILITY THAT DOES NOT EXIST IN THE LANGUAGE.*

## GROUPS OF SLIDES

- (A) 1 SIMPLE FEATURES
- 2 CONTROL STRUCTURES (NO GO TOS)
- 3 SIMPLE DATA TYPES (DEFINING NEW TYPES)
- 4 ARRAYS (GENERALIZE INDEXING)
- 5 RECORDS (NON-UNIFORM ARRAYS)
- 6 PROGRAM STRUCTURE
- 7 PASSING PARAMETERS (COMPILER CHECKS)
- 8 MINIMAL I/O
- 9 STANDARD PREFIX
- 10 RUNNING THE PROGRAM
- 11 DIFFERENCES FROM JENSEN/WIRTH
- (B) 12 SETS
- 13 POINTERS
- 14 FILES
- 15 OTHER THINGS TO READ

## GROUP 1: SIMPLE FEATURES

- COMMENTS (1.2)
- BLANKS (1.2)
- END-OF-LINE AND CONTINUATIONS (1.2)
- RESERVED WORDS (1.4)
- SEPARATORS (1.7)
- IDENTIFIERS AND VARIABLES (1.8)
- INTEGERS AND REALS (1.10)
- ASSIGNMENT (1.11)
- ARITHMETIC OPERATORS (1.12)
- RELATIONAL OPERATORS (1.13)
- LOGICAL OPERATORS (1.14)
- OPERATOR PRECEDENCE (1.15)

C THIS IS A FORTRAN COMMENT

C

C FORTRAN ALLOWS ONE STATEMENT PER LINE

C STATEMENT MUST BE WRITTEN IN COLUMNS 7 TO 72

C A LINE DELIMITS A STATEMENT:

A=B

C=D

E=A+B+C+D

C

C BLANKS GENERALLY HAVE NO EFFECT WITHIN A

C STATEMENT UNLESS USED FOR DATA:

C

A = B

E = A + B + C + D

D= G + H

C STATEMENTS ARE CONTINUED BY A CHARACTER IN

C COLUMN SIX:

C

A=B+C+D+

\*E+F+G

"THIS IS A PASCAL COMMENT"

"PASCAL ALLOWS PARTIAL, WHOLE, OR  
SEVERAL STATEMENTS ON EACH LINE"

"STATEMENTS ARE CODED IN COLUMNS 1 TO 80  
AND ARE SEPARATED BY A SEMICOLON (;) OR DELIMITED BY A  
RESERVED WORD"

```
A:=B;  
C:=D;E:=A+B+C+D;
```

"BLANKS AND COMMENTS HAVE NO EFFECT "

```
A := B ; "COMMENT" E := A + B + C + D ;
```

"CONTINUATION IS AUTOMATIC IF A DELIMITER OR SEPARATOR  
IS NOT INSERTED "

```
A:=B+C+D  
+ E+F+G;
```

## RESERVED WORDS

### FORTRAN

*NONE*

IF(IF.EQ.3)D0=27

(MAY NOT BE LEGAL ON  
SOME COMPILERS)

### PASCAL

31 IN KSU  
IMPLEMENTATION

*MAY NOT BE USED*

*AS IDENTIFIERS*

ARE UNDERLINED  
IN THIS PRESENTATION:

BEGIN  
END

*ARE NOT UNDERLINED*  
*IN AN ACTUAL PROGRAM*



## PASCAL RESERVED WORDS

(KSU IMPLEMENTATION)

### OPERATORS

DIV

MOD

NOT

OR

IN

### CONTROL WORDS

BEGIN

END

CASE

OF

WHILE

DO

REPEAT

UNTIL

IF

THEN

FOR

DOWNTO

TO

ELSE

WITH

FORWARD

### DECLARATIONS

CONST

TYPE

VAR

PROCEDURE

FUNCTION

PROGRAM

### TYPE IDENTIFICATION

ARRAY

RECORD

SET

UNIV

## RESERVED WORDS

CANNOT BE USED AS IDENTIFIERS--  
COMPILER WILL NOT ALLOW IT  
EG. BEGIN, VAR, TYPE

## KEY WORDS

GOOD IDEA NOT TO USE THESE AS VARIABLES--  
BUT COMPILER DOES NOT CHECK  
EG. BOOLEAN, REAL, INTEGER, LINE, PAGE, PARAM,

BASIC DATA TYPES

USED IN PREFIX  
(DEFINED IN GROUP 9)

SEPARATORS:     ;     ,     ..     :

THESE SEPARATE SYNTACTICAL UNITS

(THEY ALSO DELIMIT TOKENS DURING LEXICAL ANALYSIS)

PAIRED DELIMITERS

" --- "  
' --- '

RECORD ---- END

BEGIN --- END

( --- )

(, --- ,)

(: --- :)

CASE --- OF --- END

THESE (AND OTHERS) MARK  
THE BEGINNING AND END OF  
A SYNTACTICAL UNIT OR  
SEQUENCE OF UNITS

NON-PAIRED DELIMITERS

IF -- THEN ---

IF -- THEN --- ELSE ---

REPEAT --- UNTIL ---

VAR ---

TYPE -- = ---

ARRAY -- OF ---

THESE (AND OTHERS) DELIMIT  
THE PARTS OF A SYNTACTIC UNIT,  
BUT NOT NECESSARILY THE END  
OF THE UNIT.

(EVERYTHING HERE WILL BE EXPLAINED LATER IN CONTEXT)

# FORMING IDENTIFIERS

|                                  | <u>FORTRAN</u>             | <u>PASCAL</u>          |
|----------------------------------|----------------------------|------------------------|
| CHARACTERS<br>IN LETTER SET      | A-Z AND \$                 | A-Z AND _ (UNDERSCORE) |
| FIRST<br>CHARACTER               | MUST BE IN LETTER SET      | MUST BE IN LETTER SET  |
| FOLLOWING<br>CHARACTERS          | LETTER OR DIGIT            | LETTER OR DIGIT        |
|                                  | <u>EXAMPLES</u>            |                        |
|                                  | TOTAL                      | TOTAL                  |
|                                  | I                          | I                      |
|                                  | A136                       | A136                   |
|                                  | \$787                      | \$787 (ERROR)          |
|                                  | _ABLE (ERROR)              | _ABLE                  |
|                                  | 2 HITS (ERROR)             | 2 HITS (ERROR)         |
|                                  | CAN,DO (ERROR)             | CAN,DO (ERROR)         |
|                                  | CAN_DO (ERROR)             | CAN_DO                 |
| SPACES WITHIN<br>IDENTIFIERS     | LEGAL IN FORTRAN           | ILLEGAL IN PASCAL      |
|                                  | <u>EXAMPLES</u>            |                        |
|                                  | HIT IT                     | HIT IT (ERROR)         |
|                                  | MISS IT                    | MISS IT (ERROR)        |
| MAXIMUM<br>SIGNIFICANT<br>LENGTH | 6 CHARACTERS               | 80 CHARACTERS          |
|                                  | BSBALL                     | BSBALL                 |
| SAME                             | { ADD COST (7 CHARACTERS)  | ADD_COST } DIFFERENT   |
|                                  | { ADD COSTE (8 CHARACTERS) | ADD_COSTE }            |

## VARIABLES

---

|                    | <u>FORTRAN</u>                                      | <u>PASCAL</u> |
|--------------------|---|---------------|
| <u>DEFAULT</u>     | I THRU N INTEGER<br>A THRU H<br>O THRU Z REAL<br>\$ |               |
| <u>DECLARATION</u> | OPTIONAL  | MANDATORY     |

### EXAMPLE

|   |  |  |
|---|--|--|
| C | PROGRAM ONE<br>IMPLICIT INTEGER (C-F)<br>INTEGER A,B,SAM<br>REAL I,J<br>LOGICAL R<br><br>:<br>:<br>:<br>:<br><br>A=2<br>I=3.0<br>K=4<br>B=K+A<br>:<br>:<br>:<br>:<br>END | <u>PROGRAM ONE</u><br>NO IMPLICIT IN PASCAL<br>VAR A,B,SAM: INTEGER;<br>I,J: REAL;<br>R: BOOLEAN;<br><br>:<br>:<br>:<br>:<br><br><u>BEGIN</u><br>A:=2;<br>I:=3.0;<br>K:=4;"ERROR AS K<br>NOT DECLARED"<br>B:=K+A"ERROR AS K<br>NOT DECLARED"<br><br>:<br>:<br>:<br><br><u>END.</u> |
|---|--|--|

# NUMERICAL TYPES

|                | <u>FORTRAN</u>   | <u>PASCAL</u>  |
|----------------|--|--|
| <u>INTEGER</u> | 7<br>350<br>0<br>-475<br>25E6<br>27E-4<br>-16E-3<br>27,<br>2,346 | 7<br>350<br>0<br>-475<br>25E6<br>27E-4<br>-16E-3<br>27,<br>2,346 |

## REAL

|   |
|---|
| <p>MUST HAVE A DIGIT<br/>BEFORE AND AFTER<br/>THE DECIMAL POINT</p> |
|---|

|  |  |
|--|--|
| 0.36<br>.25<br>27.493<br>542.<br>-2.47<br>25.0E6<br>-432.15E2<br>637.2E-4<br>-234.34E-5<br>5,2E1,3 | 0.36<br>.25<br>27.493<br>542.<br>-2.47<br>25.0E6<br>-432.15E2<br>637.2E-4<br>-234.34E-5<br>5,2E1,3 |
|--|--|

## ASSIGNMENT

---

|               | <u>FORTRAN</u>   | <u>PASCAL</u>  |
|---------------|--|--|
| SYMBOL        | =  | :=   |
| TYPE<br>RULES | INTEGER AND REAL<br>MAY BE MIXED.<br>BOOLEAN MUST BE<br>ASSIGNED TO BOOLEAN.   | IN ALL CASES<br>THE TYPE <u>MUST</u> <u>AGREE</u> .  |
|               | INTEGER = INTEGER<br>REAL = REAL<br>REAL = INTEGER<br>BOOLEAN = BOOLEAN<br>INTEGER = REAL<br>REAL = BOOLEAN (ERROR)<br>BOOLEAN = INTEGER (ERROR) | INTEGER:= INTEGER;<br>REAL:= REAL;<br>REAL:= INTEGER;(ERROR)<br>BOOLEAN:= BOOLEAN;<br>INTEGER:= REAL;(ERROR)<br>REAL:= BOOLEAN; (ERROR)<br>BOOLEAN:= INTEGER; (ERROR)<br><br>:<br>:<br>:<br><br>(THERE ARE OTHERS) |

## ARITHMETIC OPERATORS

---

### FORTRAN

+

-

\*

\*\*

..

### INTEGER DIVISION

/

### REAL DIVISION

/

### INTEGER REMAINDER

NOT AN OPERATOR USE  
FUNCTION MOD(I,J)

### PASCAL

OPERANDS MUST  
BE SAME TYPE  
OPERANDS  
INTEGER OR REAL  
RESULTS  
INTEGER OR REAL

NONE IN PASCAL

MUST WRITE PROCEDURE

I DIV J

(OPERANDS INTEGER ONLY)  
(RESULT INTEGER)

A/B

(OPERANDS REAL)  
(RESULT REAL)

I MOD J

(OPERANDS INTEGER ONLY)



## RELATIONAL OPERATORS

### FORTRAN

.EQ.

.NE.

.LT.

.GT.

.LE.

.GE.

*NONE*

### PASCAL

=

<>

<

>

<=

>=

IN (SET MEMBERSHIP)

### EXAMPLES

A.LT.B

A<B

A.GE.B

A>=B

A.EQ.B

A=B

## LOGICAL OPERATORS

---

FORTRAN

.NOT.

.AND.

.OR.

PASCAL

NOT

&

OR

# OPERATOR PRECEDENCE

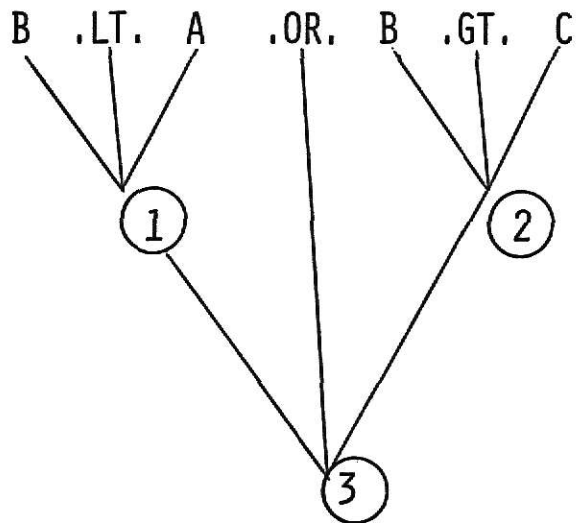
| <u>FORTRAN</u>                        | <u>PRECEDENCE</u> | <u>PASCAL</u>                             |
|---------------------------------------|-------------------|---|
| FUNCTION<br>SUBPROGRAMS               | 1      1          | <u>FUNCTION, PROCEDURE, NOT</u>           |
| **                                    | 2                 | <i>NOT IN PASCAL-<u>USE PROCEDURE</u></i> |
| *, /                                  | 3      2          | *, /, <u>DIV</u> , <u>MOD</u> , &         |
| +, -                                  | 4      3          | +, -, <u>OR</u>                           |
| .EQ., .NE., .LT.,<br>.LE., .GE., .GT. | 5      4          | =, <, >, <=, >=                           |
| .NOT.                                 | 6                 |   |
| .AND.                                 | 7                 |   |
| .OR.                                  | 8                 |   |

NOTE: MAJOR DIFFERENCE IS IN LOGICAL OPERATOR PRECEDENCE!

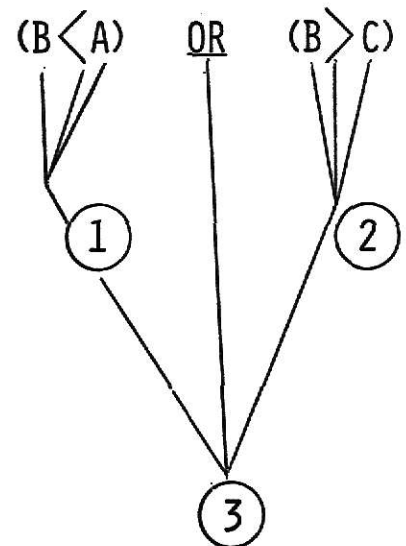
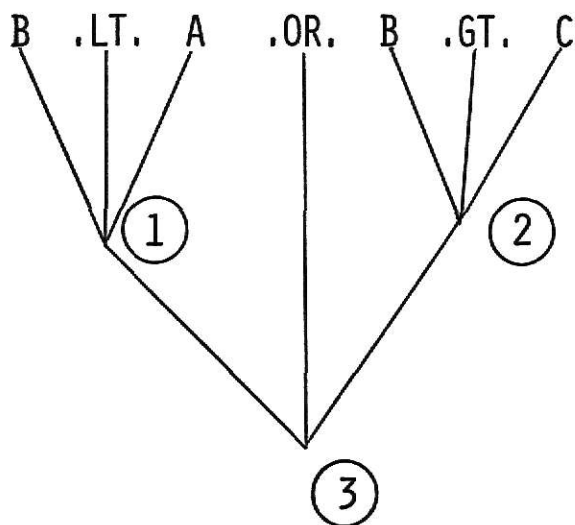
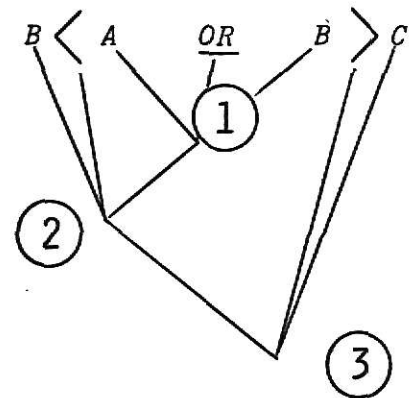
LIKE FORTRAN - EVALUATE  
EQUAL PRECEDENCE LEFT  
TO RIGHT. EXPRESSIONS  
IN ( ) ARE EVALUATED  
INDEPENDENT OF PRECEDING  
AND SUCCEEDING OPERATORS.

# EXAMPLE OF LOGICAL OPERATOR PRECEDENCE DIFFERENCE

FORTRAN



PASCAL



## GROUP 2: CONTROL STRUCTURES

IF THEN (2.2)

IF THEN ELSE (2.4)

CASE (2.5)

DO, WHILE, & REPEAT FLOW CHARTS (2.8)

LOOP EXAMPLES (2.13)

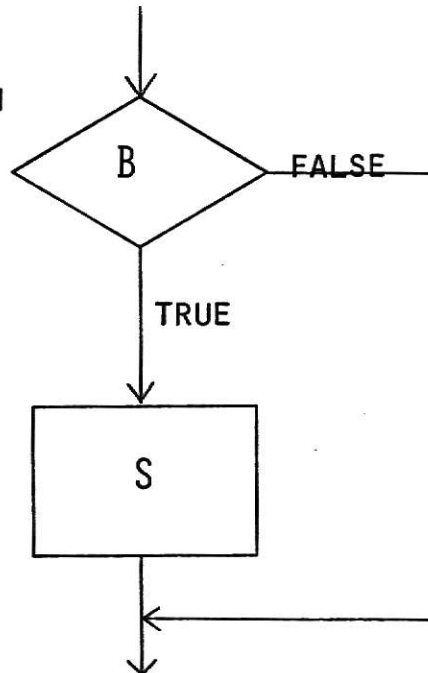
EXIT FROM THE MIDDLE OF A LOOP (2.19)

---

## IF B THEN

---

B = BOOLEAN EXPRESSION  
S = STATEMENT



---

## EXAMPLES

---

### FORTRAN

SINGLE  
STATEMENT

IF(A.EQ.B)J=1

MULTIPLE  
STATEMENT

IF(A.NE.B)GO TO 100

J=1

K=1

L=1

100 CONTINUE

### PASCAL

IF A=B THEN J:=1;

IF A=B THEN

BEGIN J:=1;

K:=1;

L:=1

END;

NOTE: NO SEMICOLON AFTER  
STATEMENT PRECEDING END

## CAUTION

---

*NEVER PUT A SEMICOLON AFTER THEN*

IF A > B THEN; J:=1;

J:=1 WILL ALWAYS EXECUTE.

IF A > B THEN;

BEGIN J:=1;

K:=-1;

L:=1

END;

THESE STATEMENTS  
WILL ALWAYS EXECUTE.

*MUST USE BEGIN END IF MORE THAN  
ONE STATEMENT IS TO BE EFFECTED.*

IF A > B THEN

J:=1;

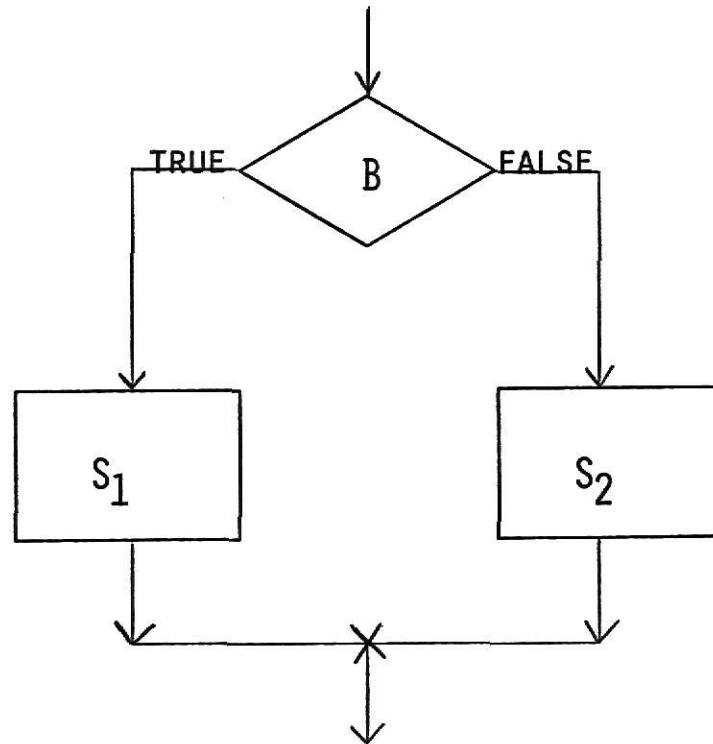
K:=1;

L:=1;

THESE STATEMENTS  
ALWAYS EXECUTE



IF B THEN S<sub>1</sub> ELSE S<sub>2</sub>



B=BOOLEAN EXPR.

S<sub>1</sub>=STATEMENT

S<sub>2</sub>=STATEMENT

FORTRAN

SINGLE  
STATEMENTS

IF(A.GT.B)J=1  
IF(A.LE.B)J=2

MULTIPLE  
STATEMENTS

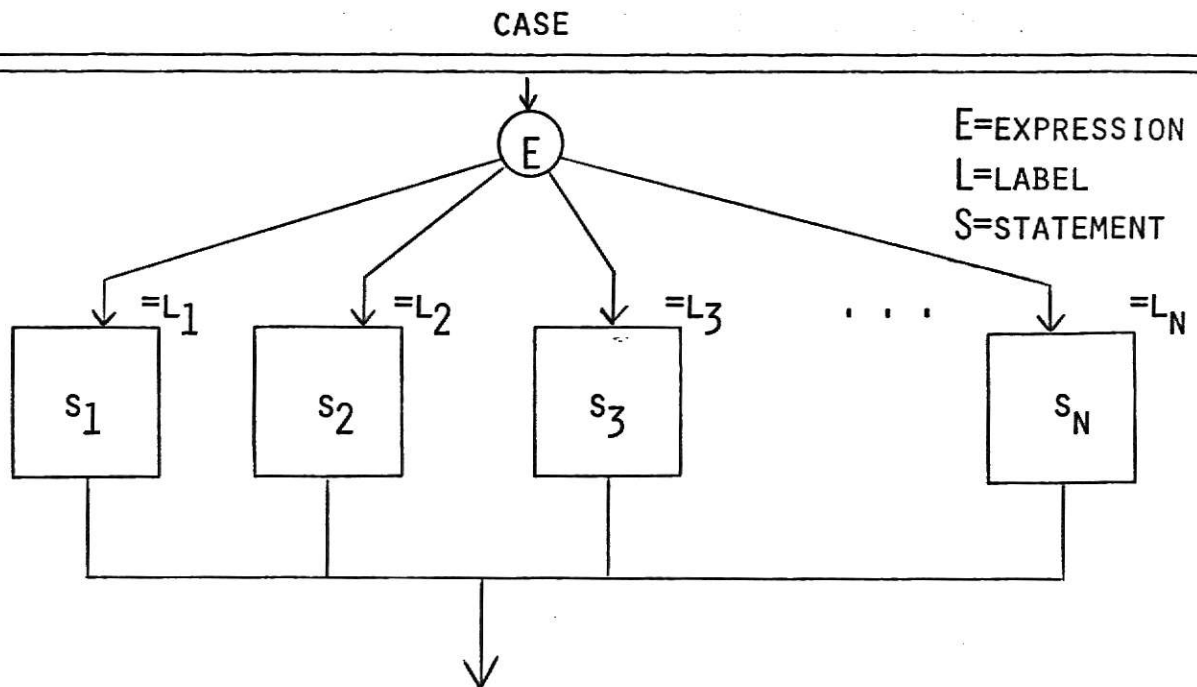
IF(A.LE.B)GO TO 100  
J=1  
K=1  
GO TO 200  
100 J=2  
K=2  
200 CONTINUE

PASCAL

IF A>B THEN J:=1  
                  ELSE J:=2;

NOTE; NEVER PLACE  
A SEMICOLON BEFORE  
OR AFTER AN ELSE.

IF A>B THEN  
    BEGIN J:=1;  
            K:=1 END  
ELSE  
    BEGIN J:=2;  
            K:=2 END;



- RULES:
1. EXPRESSION TYPE AND LABEL TYPE MUST AGREE
  2. EXPRESSION MAY NOT BE TYPE REAL
  3. LABELS MUST BE CONSTANTS

WARNING: AN INTEGER VARIABLE = INTEGER CONSTANT, THESE ARE DEFINED IN GROUP 3.

#### FORMAT

CASE E OF

L<sub>1</sub>:S<sub>1</sub>;

L<sub>2</sub>:S<sub>2</sub>;

L<sub>3</sub>:S<sub>3</sub>;

⋮  
⋮  
⋮

L<sub>N</sub>:S<sub>N</sub>

END

## CASE EXAMPLE

|     | <u>FORTRAN</u>                                   | <u>PASCAL</u>                  |
|-----|--|--------------------------------|
| C   | ASSUME I,J,K ARE INTEGERS                        | "ASSUME I,J,K DECLARED INTEGER |
| C   | AND I.GE.1, I.LE.3                               | AND I>=1, I<=3"                |
|     | :  | :                              |
|     | :  | :                              |
|     | :  | :                              |
|     | GO TO(100, <sup>2</sup> 100, <sup>3</sup> 200),I | <u>CASE</u> I <u>OF</u>        |
| 100 | J=1  | 1: <u>BEGIN</u> J:=1           |
|     | K=1  | K:=1 <u>END</u> ;              |
|     | GO TO 400  |                                |
| 200 | J=2  | 2: J:=2;                       |
|     | GO TO 400  |                                |
| 300 | J=3  | 3: <u>BEGIN</u> J:=3;          |
|     | K=3  | K:=3 <u>END</u>                |
| 400 | CONTINUE   | <u>END</u> ;                   |
|     | :  | :                              |
|     | :  | :                              |
|     | :  | :                              |
|     | :  | :                              |
|     | :  | :                              |

NOTE: LABELS ARE GLOBAL TO  
THE PROGRAM MODULE

NOTE: LABELS ARE LOCAL TO  
THE CASE STATEMENT (NOT  
GLOBAL TO THE WHOLE MODULE)

## CASE EXAMPLE

---

### FORTRAN

```
C  ASSUME I,J ARE INTEGER
C      I .GE. 1, I .LE.6
```

```
      :
      :
      :
      IF(I.EQ.1.OR.I.EQ.2)J=1
      IF(I.EQ.3.OR.I.EQ.4)J=2
      IF(I.EQ.5.OR.I.EQ.6)J=3
      :
      :
```

### PASCAL

```
"ASSUME I, J ARE
INTEGER AND I >=1, I <=6"
```

```
      :
      :
      :
      CASE I OF
        1,2: J:=1;
        3,4: J:=2;
        5,6: J:=3
      END;
```

---

```
C  ASSUME I,J,K,L,FLAG
C  ARE INTEGER
```

```
      :
      :
      :
      IF(I.EQ.J)FLAG=1
      IF(I.EQ.K)FLAG=2
      IF(I.EQ.L)FLAG=3
      :
      :
```

```
"ASSUME I, FLAG
DECLARED INTEGER
J,K,L ARE
      : INTEGER CONSTANTS"
```

```
      :
      :
      CASE I OR
        J: FLAG:=1;
        K: FLAG:=2;
        L: FLAG:=3
      END;
```

```
      :
      :
      :
      WARNING: INTEGER CONSTANTS
      ARE DIFFERENT FROM INTEGER
      VARIABLES!
```

PASCAL  
CASE EXAMPLE

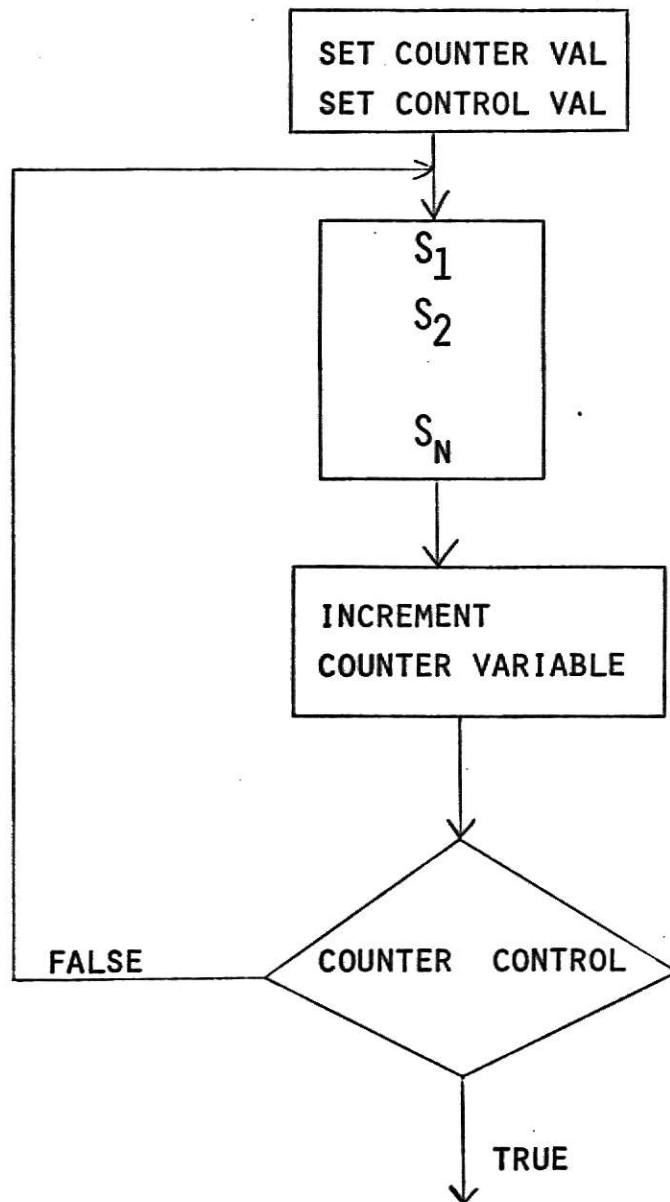
THIS CASE  
STATEMENT  
IS NOT  
EXECUTED  
(NO-OP)

"ASSUME I,J ARE  
INTEGER AND I =1, I =6"  
I:=3  
CASE I OF  
  1,2: J:=6;  
  5,6: J:=8  
END;  
.  
.  
.

"ASSUME I,J ARE  
INTEGERS AND I =1, I =6"  
.  
.  
.  
I:=7  
RUN TIME ERROR!! → CASE I OF  
  1,2: J:=6;  
  5,6: J:=8  
END;  
.  
.  
.

## THE FORTRAN DO

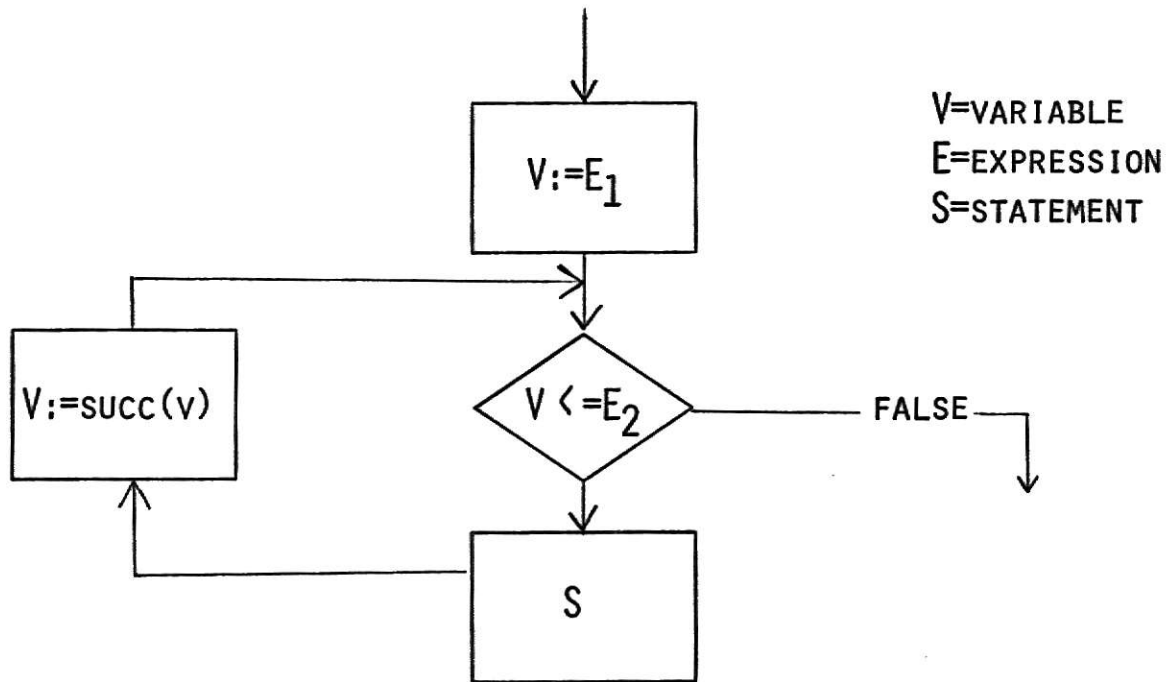
---



- 
1.  $S_1$  THRU  $S_N$  EXECUTE AT LEAST ONCE.
  2. COUNTER AND CONTROL MUST BE INTEGER AND GREATER THAN ZERO.
  3. THE COUNTER MAY ONLY BE INCREMENT BY A POSITIVE INTEGER.

## PASCAL

FOR V:=E<sub>1</sub> TO E<sub>2</sub> DO S;



1. V, E<sub>1</sub>, AND E<sub>2</sub> MUST BE OF SAME TYPE

2. TYPE MAY BE: CHARACTER  
BOOLEAN  
INTEGER

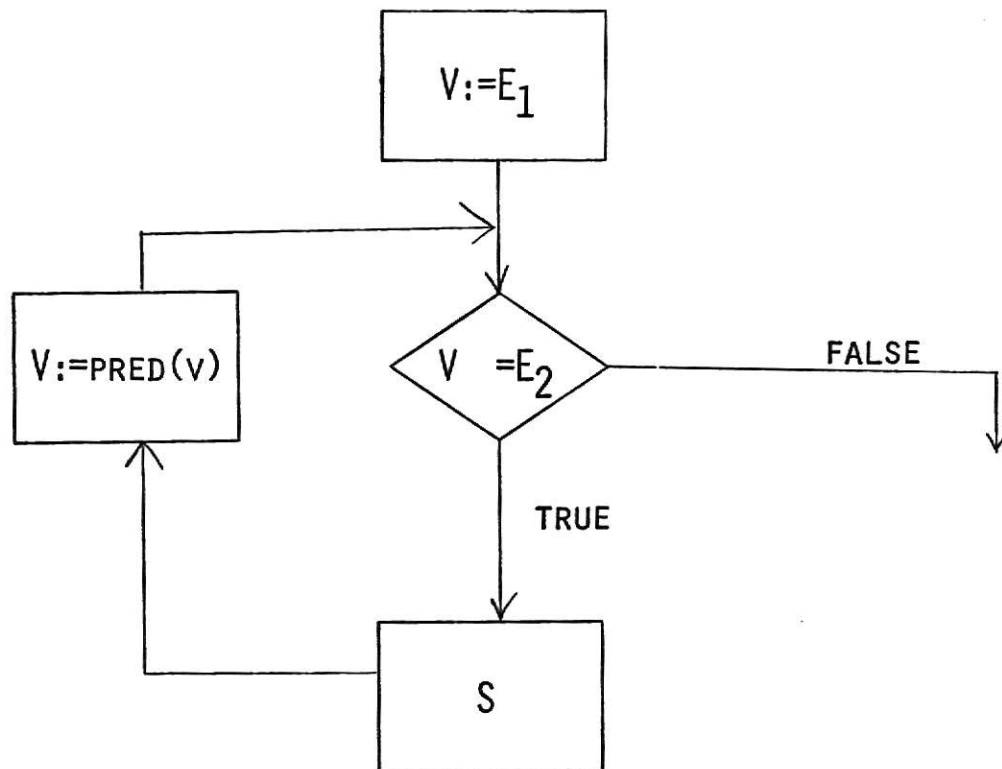
MAY NOT BE: CONSTANT  
RECORD FIELD  
FUNCTION IDENTIFIER  
ARRAY ELEMENT  
REAL

3. V MAY NOT BE CHANGED IN S.

4. V IS UNDEFINED AFTER COMPLETION OF FOR STATEMENT

5. V IS ALWAYS INCREMENTED BY ONE FOR INTEGERS.

FOR  $V := E_1$  DOWNTO  $E_2$  DO  $S$ ;

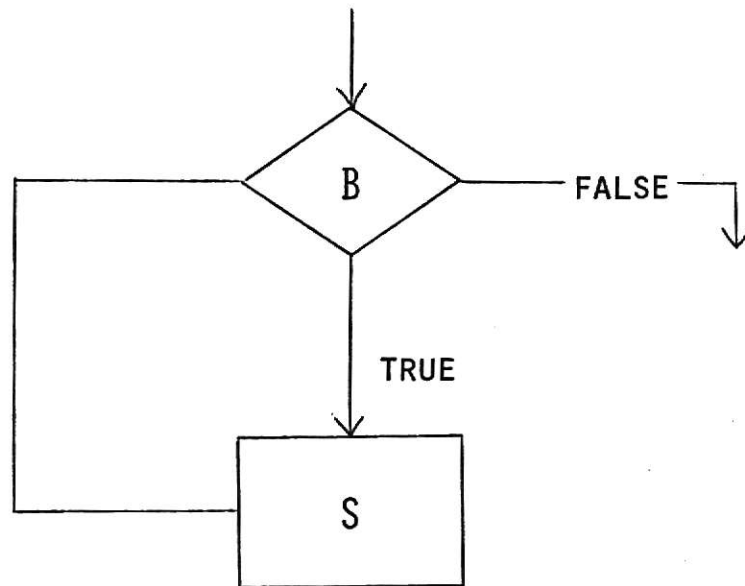


1.  $V, E_1$  AND  $E_2$  MUST BE OF COMPATABLE TYPE
2.  $V$  TYPE MAY BE: CHARACTER  
BOOLEAN  
INTEGER  
MAY NOT BE: REAL  
CONSTANT  
RECORD FIELD  
FUNCTION IDENTIFIER  
ARRAY ELEMENT
3.  $V$  MAY NOT BE CHANGED IN  $S$ .
4.  $V$  IS UNDEFINED AFTER COMPLETION OF FOR STATEMENT.
5.  $V$  IS ALWAYS INCREMENTED BY ONE FOR INTEGERS.



PASCAL  
WHILE B DO S;

---

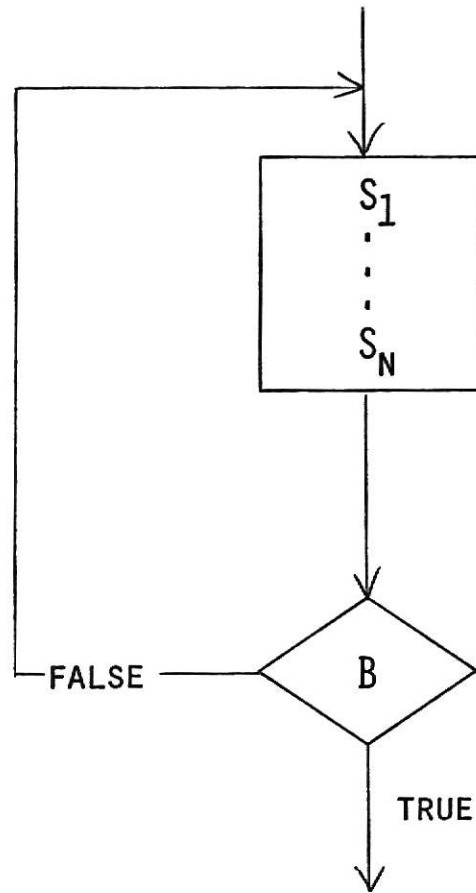


2. NORMALLY USED WHEN NUMBER OF REPETITIONS REQUIRED
3. S WILL NOT BE EXECUTED IF B IS FALSE.
4. B IS ANY LEGAL BOOLEAN EXPRESSION.

## PASCAL

REPEAT  $S_1; \dots S_N$  UNTIL B;

---



1. BODY MAY BE A SEQUENCE OF STATEMENTS. (NOTE: BEGIN, END NOT REQUIRED)
2. NORMALLY USED WHEN NUMBER OF REPETITIONS REQUIRED IS NOT KNOWN.
3.  $S_1; \dots S_N$  IS EXECUTED AT LEAST ONCE.
4. B IS ANY LEGAL BOOLEAN EXPRESSION.

---

## FOR TO DO EXAMPLES

---

### FORTRAN

```
C      I INTEGER, A,B REAL
C      BEGIN PROGRAM
      .
      B=13.0
      A =1.0
      DO 100 I=1,10
      A = A + A/B
100    CONTINUE
      .
      .
      .
      END
```

### PASCAL

```
"ASSUME I DECLARED INTEGER
  A,B DECLARED REAL"
BEGIN
  .
  B=13.0
  A:=1.0;
  FOR I:=1 TO 10 DO A:=A+A/B;
  .
  .
  .
  END.
```

## FOR TO DO EXAMPLE 2

---

### FORTRAN

```
C   ASSUME ALL VARIABLES DECLARED
C   AND INITIALIZED
C   BEGIN PROGRAM
      .
      .
      .
DO 100 I=1,10
  A=A*B
  C=C+C
100 CONTINUE
      .
      .
      .
END
```

### PASCAL

```
"ASSUME VARIABLES DECLARED
AND INITIALIZED"
BEGIN
      .
      .
      .
  FOR I:=1 TO 10 DO
    BEGIN
      A:=A*B;
      C:=C+C
    END;
      .
      .
      .
  END.
```

---

## FOR DOWNT0 DO EXAMPLE

---

### FORTRAN

```
C  ASSUME VARIABLES DECLARED
C  AND INITIALIZED
C  BEGIN PROGRAM
      .
      .
      .
      DO 100 J=1,10
      I=11-J
      B=B*B
      A(I)=B
100  CONTINUE
      .
      .
      .
      END
```

### PASCAL

```
"ASSUME VARIABLES DECLARED
AND INITIALIZED"
BEGIN
      .
      .
      .
      FOR I:=10 DOWNT0 1 DO
      BEGIN
          B=B*B
          A(I):=B
      END;
      .
      .
      .
END.
```

## WHILE DO EXAMPLES

---

|     | <u>FORTRAN</u>          | <u>PASCAL</u>                 |
|-----|-------------------------|-------------------------------|
| C   | I,K INTEGER             | "ASSUME I,K DECLARED INTEGER  |
| C   | A,B REAL                | . A,B DECLARED REAL"          |
| C   | BEGIN PROGRAM           | <u>BEGIN</u>                  |
|     | :                       | :                             |
|     | :                       | :                             |
| C   | ASSUME I AND K ASSIGNED | "ASSUME I AND K ASSIGNED SOME |
| C   | INTEGER VALUE           | INTEGER VALUE                 |
|     | :                       | :                             |
|     | B:=2.0                  | B:=2.0;                       |
|     | A=1.0                   | A:=1.0;                       |
| 100 | IF(I.LE.K)GO TO 200     | <u>WHILE</u> I > K <u>DO</u>  |
|     | A=A+A/B                 | <u>BEGIN</u> A:=A+A/B;        |
|     | I=I-1                   | I:=I-1 <u>END</u> ;           |
|     | GO TO 100               | :                             |
| 200 | CONTINUE                | :                             |
|     | :                       | <u>END.</u>                   |
|     | :                       |                               |
|     | END                     |                               |

## REPEAT UNTIL EXAMPLES

---

### INCREMENT EXAMPLE

#### FORTRAN

```
C      ASSUME J,B ARE INTEGERS
C      A IS INTEGER ARRAY INDEXED
C      FROM 1 TO 100
C      BEGIN PROGRAM
      .
      .
      .
      B=2
      DO 100 I=2,100,12
      A(I)=B*I
100    CONTINUE
      .
      .
      .
      END
```

#### PASCAL

```
"ASSUME B,I DECLARED INTEGERS
A IS INTEGER ARRAY INDEXED
FROM 1 TO 100"
BEGIN
  .
  .
  .
  B:=2;
  I:=0;
  REPEAT
    I:=I+2;
    A(I):=B*I
  UNTIL I=100;
  .
  .
  .
  END.
```

## REPEAT UNTIL

EXIT AT END EXAMPLE

### FORTRAN

```
C   ASSUME ALL VARIABLES DECLARED
C   BEGIN PROGRAM
      :
      :
      QUOTNT=0
      IF(X.GT.Y)GO TO 200
      REMAIN=Y
100  REMAIN=REMAIN-X
      QUOTNT=QUOTNT+1
      IF(X.LE.REMAIN)GO TO 100
200  CONTINUE
      :
      :
      END
```

### PASCAL

```
"ASSUME ALL VARIABLES DECLARED
  INTEGER"
  BEGIN
    :
    IF X > Y THEN QUOTIENT:=0
    ELSE BEGIN
      REMAINDER:=Y;
      QUOTIENT:=0;
      REPEAT
        REMAINDER:=REMAINDER-X;
        QUOTIENT:=QUOTIENT+1
      UNTIL X > REMAINDER
    END;
    :
    END.
```



MIDDLE EXIT

2.19

## GROUP 3 SIMPLE DATA TYPES

DECLARATIONS (3.2)

KINDS OF TYPES (3.4)

CONSTANTS (3.5)

NUMBERS (INTEGER AND REAL) (3.6)

CHARACTERS (3.8)

BOOLEANS (3.9)

ENUMERATIONS (3.10)

STRINGS (3.13)

## DECLARATIONS

### FORTRAN

```
INTEGER I,Q  
REAL R,L  
LOGICAL B  
C  DEFAULT K IS INTEGER  
C  DEFAULT X IS REAL
```

### PASCAL

```
VAR I,Q: INTEGER;  
VAR R,L: REAL;  
VAR B: BOOLEAN;  
"NO DEFAULTS"
```

### OR

```
VAR I,Q: INTEGER;  
R,L: REAL;  
B: BOOLEAN;
```

### OR

```
TYPE INT= INTEGER;  
RE= REAL;  
LOG= BOOLEAN;
```

```
VAR I,Q: INT;  
R,L: RE;  
B: LOG;
```

## NOTE

### > THE TYPE STATEMENT DEFINES

- A TEMPLATE FOR THE INTERNAL LAYOUT OF VARIABLES WHICH WILL BE DECLARED LATER; (IE, IT DECLARES TYPE IDENTIFIERS)
- IT DOES NOT DECLARE OR ALLOCATE SPACE FOR RUN TIME VARIABLES.

### > THE VAR STATEMENT DECLARES

- RUN TIME VARIABLES AND SETS THE AMOUNT AND LAYOUT OF THE SPACE THAT THE VARIABLES WILL USE AT RUN TIME.
- IT DOES NOT INITIALIZE THE VALUE OF ANY VARIABLE.

### > THE CONST STATEMENT

- INITIALIZE A VALUE FOR AN IDENTIFIER, WHICH CANNOT BE CHANGED.

## KINDS OF TYPES IN PASCAL

### ORDERED SCALAR

PRIMITIVES ARE

|           |   |                  |
|-----------|---|------------------|
| INTEGER   | } | SAME AS FORTRAN  |
| BOOLEAN   |   |                  |
| REAL      |   |                  |
| CHAR..... |   | SINGLE CHARACTER |

OTHER PRIMITIVES ARE

POINTER...EFFECTIVELY, AN ADDRESS

SCALAR ENUMERATIONS ARE

USER DEFINED TO BE A LEGAL RANGE  
OF VALUES WITH A DEFINED ORDERING

COMPOSITE STRUCTURES

INCLUDE ARRAY,  
RECORD  
SET



NEW USER DEFINED TYPES

ARE COMBINATIONS OF ALL OF THE ABOVE

CONSTANTS ARE

NOT REALLY TYPES (INTEGER, REAL,  
STRING)

## CONSTANTS

AN IDENTIFIER WHICH IS

- INITIALIZED AT COMPILE TIME
- HAS INTEGER, REAL, CHAR, STRING, OR BOOLEAN VALUE
- CANNOT BE ASSIGNED TO AT RUN TIME

C MAIN PROGRAM

INTEGER A,B,D,F(2)

REAL E,G

LOGICAL C

EQUIVALENCE (A),(B)

DATA B,C,D,E,F

/'Y\_ \_ \_',.TRUE., 12,12.6,

'MABL','E\_ \_ \_'/

PROGRAM'MAIN';

CONST B = 'Y';

C = TRUE;

D = 12;

E = 12.6;

F = 'MABLE\_';

A = B;

VAR G: REAL;

THESE ARE NOT REALLY CONSTANTS!

WARNING: NOT AN :=

## PASCAL

### NUMBERS (INTEGER AND REAL)

MAXIMUM INTEGER = 32767

MAXIMUM REAL =  $10^{38}$

EPS  $\approx 10^{-6}$

ZERO  $\approx 10^{-38}$

| OPERATION  | OPERAND 1 | OPERAND 2 | RESULT |
|------------|-----------|-----------|--------|
| + - *      | I         | I         | I      |
|            | R         | R         | R      |
|            | I         | R         | ERROR  |
| /          | R         | R         | R      |
|            | I         | R         | ERROR  |
| <u>DIV</u> | I         | I         | I      |
| RELATIONAL | I         | I         | B      |
|            | R         | R         | B      |
|            | I         | R         | ERROR  |

## PASCAL

### FUNCTIONS ON INTEGERS

PRED (X).....YIELDS  $X-1$

SUCC(X).....YIELDS  $X+1$

ABS(X).....YIELDS ABSOLUTE VALUE OF  $X$   
(A NON-NEGATIVE INTEGER)

CHR(X).....YIELDS A CHARACTER WITH ORDINAL  
VALUE  $X$ , FOR  $0 \leq X \leq 127$

EG: CHR(65) IS 'A'

CONV(X).....REAL VALUE  $X$

## PASCAL

### FUNCTIONS ON REALS

ABS(X).....ABSOLUTE VALUE

TRUNC(X).....TRUNCATED INTEGER VALUE OF  $X$



## CHAR

### SINGLE CHARACTER

*ILLEGAL IN FORTRAN*

*ILLEGAL IN FORTRAN*

PROGRAM MAIN;

VAR A : CHAR;

B : BOOLEAN;

BEGIN

A: = 'Z';

B: = A > 'C'

END.

## OPERATIONS

RELATIONAL OPERATORS ARE ALL DEFINED FOR CHAR

### FUNCTIONS

ORD(X).....YIELDS INTEGER ORDINAL

EG: ORD('B') = ORD ('A') + 1

ORD('A') = 65

SUCC(X).....SUCCESSOR

SUCC('A') IS 'B'

PRED(X).....PREDECESSOR

PRED('B') IS 'A'

# ASCII CHARACTER SET

|    |     |    |    |    |   |     |     |
|----|-----|----|----|----|---|-----|-----|
| 0  | nul | 32 |    | 64 | G | 96  |     |
| 1  | soh | 33 | !  | 65 | A | 97  | a   |
| 2  | stx | 34 | "  | 66 | B | 98  | b   |
| 3  | etx | 35 | #  | 67 | C | 99  | c   |
| 4  | eot | 36 | \$ | 68 | D | 100 | d   |
| 5  | enq | 37 | %  | 69 | E | 101 | e   |
| 6  | ack | 38 | &  | 70 | F | 102 | f   |
| 7  | bel | 39 | '  | 71 | G | 103 | g   |
| 8  | bs  | 40 | (  | 72 | H | 104 | h   |
| 9  | ht  | 41 | )  | 73 | I | 105 | i   |
| 10 | lf  | 42 | *  | 74 | J | 106 | j   |
| 11 | vt  | 43 | +  | 75 | K | 107 | k   |
| 12 | ff  | 44 | ,  | 76 | L | 108 | l   |
| 13 | cr  | 45 | -  | 77 | M | 109 | m   |
| 14 | so  | 46 | .  | 78 | N | 110 | n   |
| 15 | si  | 47 | /  | 79 | O | 111 | o   |
| 16 | dle | 48 | 0  | 80 | P | 112 | p   |
| 17 | dc1 | 49 | 1  | 81 | Q | 113 | q   |
| 18 | dc2 | 50 | 2  | 82 | R | 114 | r   |
| 19 | dc3 | 51 | 3  | 83 | S | 115 | s   |
| 20 | dc4 | 52 | 4  | 84 | T | 116 | t   |
| 21 | nak | 53 | 5  | 85 | U | 117 | u   |
| 22 | syn | 54 | 6  | 86 | V | 118 | v   |
| 23 | etb | 55 | 7  | 87 | W | 119 | w   |
| 24 | can | 56 | 8  | 88 | X | 120 | x   |
| 25 | em  | 57 | 9  | 89 | Y | 121 | y   |
| 26 | sub | 58 | :  | 90 | Z | 122 | z   |
| 27 | esc | 59 | ;  | 91 | [ | 123 | {   |
| 28 | fs  | 60 | <  | 92 | \ | 124 |     |
| 29 | gs  | 61 | =  | 93 | ] | 125 | }   |
| 30 | rs  | 62 | >  | 94 | ^ | 126 | ~   |
| 31 | us  | 63 | ?  | 95 | _ | 127 | del |

## BOOLEANS

FORTRAN

, TRUE ,

, FALSE ,

PASCAL

TRUE

FALSE

## PASCAL OPERATIONS ON BOOLEANS

&, OR, NOT

YIELDS A BOOLEAN

RELATIONAL

( FALSE < TRUE )

PROGRAM "MAIN";

VAR BOY : BOOLEAN;

HT : INTEGER;

BEGIN

.  
.  
.  
.  
.  
.  
.  
.

BOY: = HT > 190 ;

END.

## ENUMERATIONS

CAN BE - AN ORDERED SEQUENCE OF INTEGERS

- ORDERED SEQUENCE FALSE, TRUE
- ORDERED SEQUENCE OF CHARACTERS
- A SEQUENCE OF IDENTIFIERS (UP TO 120)  
(THE SEQUENCE DEFINES AN ORDERING)
- A SUBSEQUENCE OF ANY OTHER ENUMERATION  
(CALLED A SUBRANGE)

|   |                             |  |
|---|-----------------------------|--|
| C | MAIN                        | <u>PROGRAM</u> MAIN ;                      |
|   | INTEGER E1, E4, E5          | TYPE PEOPLE = (BILL, RICH, RUS, RON, JAN); |
|   | LOGICAL E2                  | VAR E1: 0..10;                             |
| C | E1 IN RANGE 0 TO 10         | E2: (FALSE, TRUE);                         |
| C | BILL = 1, RICH = 2, RON = 3 | E3: 'A'..'Z';                              |
| C | RUS = 4, JAN = 5            | E5: PEOPLE;                                |
|   | E1 = 9                      | E4: BILL..RON;                             |
|   | E2 = .TRUE.                 |  |
| C | CANNOT HANDLE CHARACTERS    | <u>BEGIN</u>                               |
|   | E5 = 3                      | E1: = 9;                                   |
|   | E4 = E5                     | E2: = RUS > RICH;                          |
|   | END                         | E3: = SUCC('W');                           |
|   |                             | E5: = RON;                                 |
|   |                             | E4: = E5                                   |
|   |                             | <u>END.</u>                                |

## FUNCTIONS ON ENUMERATIONS

ALL RELATIONAL OPERATIONS ARE DEFINED

SUCC, PRED ARE DEFINED, EG.

SUCC(RON) IS RUS , BUT SUCC(JAN) IS NOT DEFINED

ENUMERATIONS ARE USED IN CASE STATEMENTS, IN VARIANT RECORDS.

EG; CONTINUING LAST EXAMPLE

|                                |                             |
|--------------------------------|-----------------------------|
| GO TO (101,101,102,101,102),E5 | <u>CASE E5 OF</u>           |
| 101 E2 = .TRUE.                | RICH, BILL, RON: E2:= TRUE; |
| GO TO 103                      | RUS, JAN : E2: = FALSE      |
| 102 E3 =,FALSE.                | <u>END;</u>                 |
| 102 CONTINUE                   |                             |

## STRINGS

STRINGS ARE ACHIEVED AS AN ARRAY OF CHARACTERS

```
C  PROGRAM MAIN
    INTEGER S1(2),S2(2)
    INTEGER NAME1(2),NAME2(2)
    LOGICAL E2
    DATA S1/'JOHN','NY_ _'/,
1      S2/'BOBB','Y_ _ _'/'
    NAME1(1) = S1(1)
    NAME1(2) = S1(2)
    NAME2(1) = S2(1)
    NAME2(2) = S2(2)
    IF (NAME1(1) - NAME2(1))100,101,102
101  IF (NAME1(2) - NAME2(2))100,101,102
102  E2 = .TRUE.
    GO TO 104
100  E2 = .FALSE.

    PROGRAM MAIN;
    TYPE STRING6 = ARRAY (.1..6.) OF CHAR;
    VAR NAME1, NAME2: STRING6;
        E2: BOOLEAN;
    BEGIN
        NAME1: = 'JOHNNY';
        NAME2: = 'BOBBY_';
        IF NAME2 < NAME1
            THEN E2: = TRUE
            ELSE E2: = FALSE;

        NAME1: = 'JOHN';
        NAME1(.3.): = 'A'
    END.
```

1. A STRING MUST CONTAIN AN EVEN NUMBER OF CHARACTERS.
2. FOR STRING ASSIGNMENT (WITH EXPLICIT INDEXING), THE WHILE STRING MUST BE ASSIGNED. EG. IN THE LAST EXAMPLE

NAME1: 'JOHN' IS ILLEGAL

### OPERATIONS ON STRINGS

RELATIONAL OPERATORS - THE STRINGS MUST BE THE SAME LENGTH.

EG: 'JOHNNY' 'BILL' IS ILLEGAL



GROUP 4      ARRAYS

ARRAYS (4.2)

EXAMPLES - ENUMERATION INDEXING (4.3)

NON-NUMERIC COMPONENTS (4.7.2)

OPERATIONS IN ARRAYS (4.9)

## THE ARRAY TYPE

### FORTRAN

FIXED NUMBER OF  
COMPONENTS ALL OF  
SAME TYPE

COMPONENT TYPES:

INTEGER

REAL

DOUBLE PRECISION

LOGICAL

COMPLEX

(LITERAL POSSIBLE)

SUBSCRIPTS:

NONZERO INTEGER

USE "(I,J)"

DECLARATION:

WITH DIMENSION, INTEGER,  
REAL, ETC., STATEMENTS

STATIC ALLOCATION OF  
VARIABLES (EXCEPT  
FOR DUMMY ARGUMENTS)

### PASCAL

FIXED NUMBER OF  
COMPONENTS ALL OF  
SAME TYPE

COMPONENT TYPES:

MAY BE ANY TYPE!

SUBSCRIPTS:

ANY ENUMERATION!

USE "(.I,J.)"

DECLARATION:

MAY BE DECLARED

IN TYPE OR VAR STATEMENT

STATIC ALLOCATION OF  
VARIABLES IN MAIN  
PROGRAM

## ARRAY EXAMPLES

---

### FORTRAN

```
C  MAIN PROGRAM
      :
      :
      DIMENSION A(10), I(10,20)
      :
      :
C  BEGIN PROGRAM
      :
      J = 2
      A(J) = 2.5
      I(J,J) = 3
      :
      :
      END
```

### PASCAL

```
PROGRAM MAIN;
VAR
  J: INTEGER;
  A: ARRAY(1..10) OF REAL;
  I: ARRAY(1..10,1..20)OF INTEGER;
  :
  :
BEGIN
  :
  J := 2;
  A(J) := 2.5;
  I(J,J) := 3;
  :
  :
END.
```

## ARRAY EXAMPLES

### FORTRAN

```
C  MAIN PROGRAM
  .
  .
  REAL B(20),C(20)
  .
  .
C  I MUST BE IN RANGE 1 TO 20
  B(I) = 10.
  .
  .
  END
```

RUN TIME ERROR



### PASCAL

```
PROGRAM MAIN ;
TYPE IROW = 1..20;
  VEC = ARRAY (.IROW.) OF REAL;
VAR INDEX:IROW;
    B,C : VEC;

BEGIN
  INDEX:= 20
  B(.INDEX.): = 10.0;
  INDEX: = 21;
  .
  .
  END.
```

```
PROGRAM "MAIN";
  VAR A: ARRAY (.'A'..'Z'.) OF REAL;
  BEGIN
    .
    .
    A(.'W'.): = 3.0;
    .
    .
  END.
```

## ARRAY EXAMPLES

---

PROGRAM MAIN;

·  
·  
·

VAR A: ARRAY (.1..10.) OF ARRAY (.11..20.) OF REAL;  
B: REAL;

·  
·  
·

BEGIN

·  
·  
·

B: = A(.4.)(.15.);

·  
·  
·

END.

## PASCAL

PROGRAM MAIN;

PROGRAM MAIN;

VAR A: ARRAY(.1..10.) OF ARRAY (.11..20.)  
OF REAL;  
B: REAL;

B: REAL;

B: REAL;

BEGIN

BEGIN

B: = A(.4.)(.15.);

B: = A(.4.)(.15.);

B: = A(.4,15.);

B: = A(.4,5.);

.

.

END.

END.

NOTE: INDEXING OF MULTIPLE DIMENSION ON  
ARRAYS CAN BE DONE IN EITHER FORM

## FORTRAN

C    MAIN

RED = 1, WHITE = 2, BLUE = 3

INTEGER FLAG (3,2), X(6)

DATA X/'BRIG', 'HT\_ \_', 'PURE', '\_ \_ \_ \_',

\*        'COOL', '\_ \_ \_ \_ '/

FLAG(1,1) = X(1)

FLAG(1,2) = X(2)

FLAG(2,1) = X(3)

FLAG(2,2) = X(4)

FLAG(3,1) = X(5)

FLAG(3,2) = X(6)

.

.

.

END

## PASCAL

### ARRAY EXAMPLE

PROGRAM "MAIN";

TYPE COLOR = (RED, WHITE, BLUE);

        STRING6 = ARRAY (.1..6.) OF CHAR;

VAR FLAG: ARRAY (.COLOR.) OF STRING6;

 :  
 :  
 :

BEGIN

 :  
 :  
 :

    FLAG(.RED.): = 'BRIGHT';

    FLAG(.WHITE.): = 'PURE\_\_';

    FLAG(.BLUE.): = 'COOL\_\_';

 :  
 :  
 :

END.



NOTE      ARRAYS REQUIRE STATIC ALLOCATION

EG.

VAR MAX: 100..500;

VAR BILL: ARRAY (1..MAX) OF INTEGER;

·  
·  
·

IS ILLEGAL FOR DECLARING A NEW ARRAY BECAUSE  
1..MAX IS AN ILLEGAL ENUMERATION SINCE MAX IS  
NOT A CONSTANT.

## PASCAL

### OPERATIONS ON WHOLE ARRAYS

|        |   |   |
|--------|---|---|
| $\neq$ | } | PROVIDED THAT THE ARRAYS<br>ARE THE SAME <u>TYPE</u> AND SIZE,<br>RESULT IS TRUE OR FALSE |
| $=$    |   |   |
| $<>$   |   |   |

REMEMBER: STRINGS ARE AN EXCEPTION:  
ALL THE RELATIONAL OPERATORS WORK FOR  
STRINGS OF THE SAME SIZE.

### OPERATIONS ON SINGLE COMPONENTS

ANY OPERATION THAT IS LEGAL FOR THAT TYPE OF  
COMPONENT.

## GROUP 5: RECORDS

GENERALIZATIONS (5.2)

EXAMPLES: MIXED COMPONENTS (5.3.1)

REFERENCING COMPONENTS ( WITH DO ) (5.3.2)

NESTED RECORDS (5.4)

ARRAYS OF RECORDS (5.5)

CONSTRAINTS (5.6)

VARIANT RECORD EXAMPLE (5.8)

## RECORD

### IS A GENERALIZATION OF ARRAY

#### ARRAYS

ALL COMPONENTS ARE SAME TYPE

ALL COMPONENTS SAME SIZE  
(IE. ARRAY IS RECTANGULAR)

TYPES OF COMPONENTS FIXED AT  
COMPILE TIME (STATIC)

COMPONENTS ARE GIVEN INDICES;

EG.

1,2,3,...  
'A','B','C',...  
RED, WHITE, BLUE,...

COMPONENTS ARE ACCESSED BY  
INDEXING; EG.

BILL(.1.)  
BOB(.'A'.)  
HUE(.RED.)

#### RECORDS

COMPONENTS CAN BE DIFFERENT TYPES

COMPONENTS CAN BE DIFFERENT SIZES

POSSIBLE TYPES OF COMPONENTS  
(CALL VARIANTS) ENUMERATED AT  
COMPILE TIME (STATIC); ACTUAL  
TYPE OF COMPONENT CAN BE  
DETERMINED AT RUN TIME (DYNAMIC).

\*WARNING: SPACE FOR THE MAXIMUM  
SIZE VARIANT IS ALLOCATED  
STATICALLY.

COMPONENTS ARE GIVEN NAMES;

EG.

FIELD1, FIELD2, FIELD3  
NAME, AGE, SSNUM

COMPONENTS ARE ACCESSED BY NAME  
QUALIFICATION, EG.

LINE.FIELD2  
PERSON.SSNUM

## FORTRAN

REAL FIELD4

INTEGER FIELD1, FIELD2, LINE(13)

LOGICAL FIELD3(10)

EQUIVALENCE (LINE(1), FIELD1), (LINE(2), FIELD2),  
(LINE(3), FIELD3(1)), (LINE(13), FIELD4)

DATA I1 = 'A \_ \_ '

FIELD1 = I1

FIELD2 = 3

FIELD3(7) = .TRUE.

FIELD4 = 13.31

C THESE ARE NOT REALLY THE SAME AS

C RECORD FIELD NAMES

## PASCAL

### EXAMPLE 1

```
VAR: LINE: RECORD  
      FIELD1: CHAR;  
      FIELD2: INTEGER;  
      FIELD3: ARRAY(.1..10.) OF BOOLEAN;  
      FIELD4: REAL  
      END;
```

```
BEGIN  
  LINE.FIELD1: = 'A';  
  LINE.FIELD2: = 3;  
  LINE.FIELD3(.7.): = TRUE;  
  LINE.FIELD4: = 13.31  
END.
```

OR

```
BEGIN  
  WITH LINE DO  
    BEGIN FIELD1: = 'A';  
          FIELD2: = 3;  
          FIELD3(.7.): = TRUE ;  
          FIELD4: = 13.31  
    END;  
    ⋮  
END.
```

# PASCAL


## EXAMPLE:

"RECORDS CAN BE NESTED"

TYPE STRING12 = ARRAY(.1..12.) OF CHAR;  
STRING8 = ARRAY(.1..8.) OF CHAR;

VAR MAN: RECORD  
NAME: STRING12  
AGE: INTEGER;  
WIFE: RECORD  
AGE: INTEGER  
NAME: STRING8;  
CHILDREN: BOOLEAN  
END;

THESE HAVE  
DIFFERENT  
SCOPE



·  
·  
·

## OR

"RECORDS CAN BE NESTED THIS WAY"

TYPE STRING8= ARRAY (.1..8.) OF CHAR;  
TYPE WREC = RECORD

AGE: INTEGER;  
NAME: STRING8;  
CHILDREN: BOOLEAN  
END;

VAR MAN: RECORD  
AGE: INTEGER;  
NAME: STRING8;  
WIFE: WREC  
END;

BEGIN

WITH MAN DO BEGIN

NAME:= 'BILLY\_ \_ \_';  
AGE:= 30;  
WIFE.NAME:='WALLY\_ \_ \_';  
WIFE.CHILDREN:=TRUE  
END;

·  
·  
·

# PASCAL

## EXAMPLE

"AN ARRAY OF RECORDS"

TYPE STRING8= ARRAY (.1..8.) OF CHAR;

TYPE LINE = RECORD

FIELD1: CHAR;

FIELD2: INTEGER;

FIELD3: STRING8;

FIELD4: REAL

END;

VAR TABLE: ARRAY (.1..100.) OF LINE;

BEGIN

TABLE(.13.).FIELD3:= 'IT\_WORKS';

TABLE(.13.).FIELD2:= 87;

TABLE(.13.).FIELD4:=13.31;

.

.

.

END.



VARIABLES MUST BE DECLARED BEFORE THEY  
ARE USED

TYPE STRING8= ARRAY(1..8.)OF CHAR;

TYPE MAN= RECORD

NAME : STRING8;

AGE : INTEGER;

WIFE : WOMAN ← ILLEGAL

END;

TYPE WOMAN= RECORD

NAME : STRING8;

AGE : INTEGER

END;

•  
•  
•  
•  
•

ALSO, STRUCTURES CANNOT BE RECURSIVELY DEFINED.

## CONSTRAINTS

"LEGAL"

```
VAR. LINE: RECORD  
    FIELD1: INTEGER;  
    FIELD2: 1 .. 10  
    END;  
    .  
    .  
    .
```

"LEGAL"

```
TYPE F2 = 1 .. 10;  
VAR LINE : RECORD  
    FIELD1: INTEGER;  
    FIELD2: F2  
    END;  
    .  
    .  
    .
```

ENUMERATION MAY BE DIRECTLY DEFINED WITHIN A RECORD  
DEFINITION

PASCAL  
SUPPOSE WE WANT AN ARRAY OF RECORDS

```
TYPE STRING8= ARRAY (1..8) OF CHAR;  
TYPE SEXTYP= (MALE, FEMALE);  
TYPE PERSON= RECORD  
                "SOMETHING HERE"  
            END;
```

```
VAR TABLE: ARRAY (.1..100.)OF PERSON;
```

WHERE

FOR MEN, WE WANT:

```
TYPE WINDEX= 1..100;  
TYPE PERSON= RECORD  
                NAME: STRING8;  
                SEX: SEXTYP;  
                AGE: INTEGER;  
                WIFE: WINDEX  
            END;
```

FOR WOMEN WE WANT:

```
TYPE PERSON= RECORD  
                NAME: STRING8;  
                SEX: SEXTYPE;  
                AGE: INTEGER;  
                NUMBER: INTEGER;  
                CHILDREN: ARRAY(.1..3.)  
                        OF STRING8  
            END;
```

THIS KIND OF PROGRAMMING IS VERY USEFUL; IT OCCURS COMMONLY IN  
PASCAL PROGRAMS

THAT IS:

- THE RECORD DOES NOT ALWAYS HAVE THE SAME FORM  
(IE. DIFFERENT INSTANCES OF THE RECORD TYPE DO NOT ALL HAVE THE SAME FORM)
- THE VARYING PART IS THE LAST COMPONENT OF THE RECORD
- THE TYPE OF THE VARIABLE COMPONENT IS ONE OF AT MOST 16 DIFFERENT SPECIFIED ALTERNATIVES
- FOR ANY INSTANCE OF THE RECORD,  
THE ACTUAL TYPE OF THE VARIABLE COMPONENT  
DEPENDS UPON THE VALUE OF A PRIOR COMPONENT OF  
THE RECORD WHICH IS AN ENUMERATION TYPE OF AT  
MOST 16 DIFFERENT VALUES  
(EQ. 0..15 OR ID0, ..., ID15)
- WE CAN AFFORD TO ALLOCATE SPACE WITH EVERY INSTANCE OF THE  
RECORD FOR THE LARGEST ALTERNATIVE COMPONENT.  
(HENCE THE RECORD HAS A FIXED ALLOCATION SIZE EVEN  
THOUGH ONE COMPONENT IS VARIABLE.)

## PASCAL

```
TYPE STRING8 = ARRAY (1..8) OF CHAR;  
TYPE SEXTYP = (MALE, FEMALE);  
TYPE WINDEX = 1..10;  
TYPE PERSON = RECORD  
    NAME: STRING8;  
    AGE : INTEGER;  
    CASE SEX: SEXTYP OF  
        MALE: (WIFE: WINDEX);  
        FEMALE: (NUMBER: INTEGER;  
                  CHILDREN: ARRAY (.1..3.) OF STRING8)  
    END;
```

```
VAR TABLE: ARRAY(.1..100.) OF PERSON;  
    I:1..10;  
    SEXUAL:SEXTYP;  
:  
BEGIN  
:  
    I:= 8,  
    SEXUAL:=MALE;  
    WITH TABLE(.I.) DO BEGIN  
        NAME:='JONES_ _ _';  
        AGE:=38;  
        SEX:=SEXUAL;  
        CASE SEXUAL OF  
            MALE: WIFE: = 8;  
            FEMALE: BEGIN NUMBER:=1;  
                    CHILDREN(.1.):='HAPPY_ _ _'  
                    END "BEGIN"  
            END "CASE"  
        END "WITH";  
:  
:  
END.
```

## NOTES ABOUT VARIANT RECORDS

- > USE OF END TO CLOSE BOTH THE VARIANT PART OF THE RECORD AND THE RECORD ITSELF
- > USE OF "(" ")" IN THE VARIANT SYNTAX - DIFFERENT THAN CASE\_STATEMENT SYNTAX
- > USE OF THE TAG VARIABLE (SEX) AND THE TAG VARIABLE DECLARATION (SEX:SEXTYP) AFTER CASE - DIFFERENT THAN CASE\_STATEMENT SYNTAX
- > THE TAG VARIABLE MUST BE AN ENUMERATION WITH 2 TO 16 VALUES (EG. 0..15)
- > IT IS POSSIBLE THAT FOR SOME OF THE TAG VALUES. THE VARIANT COMPONENT IS NULL (IE. NO COMPONENT)

## SET 6 PROGRAM & PROCEDURE STRUCTURE

GENERAL FEATURES (6.2)

EXAMPLES - LOCAL/GLOBAL VARIABLES (6.3)  
- DYNAMIC ALLOCATION (6.4)

CONSTRAINTS (6.5)

EXAMPLES - NESTED PROCEDURES (ILLEGAL) (6.6)  
- FORWARD REFERENCE (RESTRICTED) (6.7)  
- RECURSIVE PROCEDURE (6.8)  
- INDIRECT RECURSION (WITH FORWARD) (6.9)

## SUBPROGRAMS

### FORTRAN

- SUBPROGRAMS ARE CALLED SUBROUTINES (OR FUNCTIONS)
- SUBPROGRAMS ARE SEPARATE BLOCKS, NOT WITHIN THE MAIN PROGRAM
- MAIN AND SUBS CAN BE COMPILED AS A GROUP OR COMPILED SEPARATELY AND THEN "LINKED".
- ~~MUST~~ USE AN EXPLICIT "CALL" FOR SUBROUTINES
- EXPLICIT "RETURN"
- ALL SUBPROGRAM VARIABLES ARE LOCAL UNLESS DECLARED "COMMON"
- ALL VARIABLES ARE ALLOCATED STATICALLY

### PASCAL

- SUBPROGRAMS ARE CALLED PROCEDURES (OR FUNCTIONS)
- EACH SUBPROGRAM MUST BE A DECLARATIONS STATEMENT BEFORE THE BODY OF THE MAIN PROGRAM (EACH WHOLE PROCEDURE IS ONE DECLARATION)
- MAIN AND SUBS ARE COMPILED TOGETHER (NO "LINKING" PROCESS) HOWEVER= PROGRAMS CAN CALL OTHER PROGRAMS AND EVEN PASS SOME PARAMETERS.
- SUBPROGRAMS ARE INVOICED BY NAME.
- RETURN IS IMPLICIT
- ALL SUBPROGRAM VARIABLES (WHICH MUST BE DECLARED) ARE LOCAL. ANY VARIABLE NOT DECLARED IN THE SUBPROGRAM IS GLOBAL TO THE MAIN PROGRAM. (EXAMPLE 1)
- SUBPROGRAM VARIABLES ARE ALLOCATED DYNAMICALLY (SEE EXAMPLE 2)



## EXAMPLE 1

```
C  MAIN
    COMMON NCAT
    INTEGER NCAT
    :
    :
    NCAT = 1000
    :
    :
    CALL SUB1
```

```
C  NCAT STILL 1000
    CALL SUB2
C  NOW NCAT IS 2
    END
```

```
SUBROUTINE SUB2
COMMON NCAT
INTEGER NCAT
NCAT = 2
RETURN
END
```

```
SUBROUTINE SUB1
INTEGER NCAT
NCAT = 2
RETURN
END
```

```
PROGRAM "MAIN"
VAR NCAT: INTEGER;
```

```
PROCEDURE SUB2
    BEGIN
        NCAT:=2
    END "SUB2";
```

```
PROCEDURE SUB1
    VAR NCAT: INTEGER;
    BEGIN
        NCAT: =2
    END "SUB1";
```

```
NCAT:=1000;
SUB1; "NCAT STILL 1000"
SUB2; "NOW NCAT IS 2"
END.
```

NCAT IS GLOBAL FOR MAIN AND SUB1

NCAT IS LOCAL IN SUB2

## EXAMPLE 2

|   |   |
|---|---|
| <pre> C  MAIN COMMON/COM/NCAT, CAT(1000) NCAT = 1000 : : CALL SUB1 : : CALL SUB2 : : END  SUBROUTINE SUB1 COMMON/COM/NCAT,CAT(1000) DIMENSION CAT1(1000) : : CAT(I) = CAT1(I) : : RETURN END  SUBROUTINE SUB2 COMMON/COM/NCAT,CAR(1000) : : CAT(I) =CAT2(I) : : RETURN END </pre> | <pre> PROGRAM "MAIN"; CONST NCAT = 1000 TYPE ATYPE=ARRAY(.1..NCAT.)   OF REAL; VAR CAT:ATYP: I:INTEGER;  PROCEDURE SUB1;   VAR CAT1: ATYPE; I:1..NCAT;   BEGIN     :     CAT(.I.):=CAT1(.I.);     :   END "SUB1"; PROCEDURE SUB2;   VAR CAT2:ATYPE;     I:1..NCAT;   BEGIN     :     CAT(.I.):=CAT2(.I.);     :   END "SUB2";   BEGIN     :     SUB1;     :     SUB2;     :   END. </pre> |
|---|---|

---

STATIC ALLOCATION:  
 APPROX. 3003 CELLS  
 ALLOCATED FOR DATA

---

DYNAMIC ALLOCATION:  
 APPROX. 2002 CELLS ALLOCATED  
 FOR DATA. (CAT1 AND CAT2 BOTH  
 USE SAME CELLS ON RUN-TIME STACK,  
 BUT AT DIFFERENT TIMES)

## CONSTRAINTS

SUBROUTINES CANNOT BE NESTED

SUBPROGRAM DECLARATIONS CANNOT BE NESTED (THIS IMPLEMENTATION) (SEE EXAMPLE 3)

NO ORDERING OF SUBPROGRAMS IS REQUIRED

SUBPROGRAMS MUST BE ORDERED SO THAT THE DECLARATION PRECEEDS THE REFERENCE TO THE SUBPROGRAM. (EXAMPLE 4)

NO RECURSION ALLOWED

RECURSIVE CALLS ARE ALLOWED (EXAMPLE 5); INDIRECT RECURSION REQUIRES THE FORWARD DECLARATION SO AS TO AVOID THE REFERENCE-BEFORE-DECLARATION CONSTRAINT (EXAMPLE 6)

### EXAMPLE 3

```
PROGRAM MAIN ;  
VAR A,B: INTEGER;  
PROCEDURE SUB1;  
    VAR C,D: INTEGER;  
    {  
        PROCEDURE SUB2;  
            BEGIN  
                .  
                .  
                .  
            END "SUB2" ;  
    }  
    BEGIN  
        .  
        .  
        .  
        SUB2;  
        .  
        .  
        .  
    END "SUB1";  
BEGIN  
    .  
    .  
    .  
END.
```

*ILLEGAL*

NESTED PROCEDURES ARE ILLEGAL

## EXAMPLE 4

### ILLEGAL PASCAL

PROGRAM MAIN;

⋮

PROCEDURE A;

⋮

BEGIN

⋮

B;

⋮

END "A";

PROCEDURE B;

⋮

### LEGAL PASCAL

PROGRAM MAIN ;

⋮

PROCEDURE B;

⋮

BEGIN

⋮

END "B";

PROCEDURE A;

⋮

BEGIN

⋮

B;

⋮

END "A";

BEGIN

A;

⋮

END.

REFERENCE TO A PROCEDURE BEFORE ITS DECLARATION  
IS ILLEGAL

### EXAMPLE 5

PROGRAM MAIN  
VAR I: INTEGER;  
PROCEDURE FACTORIAL;

BEGIN

:

I:=I-1;

IF I>=2 THEN FACTORIAL;

:

:

END "FACTORIAL";

BEGIN

I:=5;

FACTORIAL;

:

:

:

END.

ILLEGAL IN  
FORTRAN

RECURSIVE PROCEDURE  
(CALLS ITSELF)

## EXAMPLE 6 INDIRECT RECURSION

ILLEGAL IN  
FORTRAN

### NOTES:

- PARAMETERS ARE LISTED WITH THE "FORWARD" DECLARATION
- PARAMETERS ARE NOT LISTED WITH THE SECOND DECLARATION
- COMPILER CANNOT DETERMINE (STATICALLY) IF THIS WILL CAUSE RUN-TIME STACK OVERFLOW. (AVOID THIS TYPE OF PROGRAMMING IF POSSIBLE!)

NOTE:  
PROCEDURE CALL!  
NOT A(I.)

```
PROGRAM MAIN ;  
VAR I: INTEGER;  
PROCEDURE B(PARAMETER LIST); FORWARD  
PROCEDURE A (PARAMETER LIST);  
    VAR J: INTEGER;  
    BEGIN  
        .  
        .  
        IF I > J THEN B(I-1);  
        .  
        .  
    END "A";  
PROCEDURE B;  
    VAR K: INTEGER;  
    BEGIN  
        .  
        .  
        IF I > K THEN A(I-1);  
        .  
        .  
    END "B";  
BEGIN  
    .  
    .  
    I:=5;  
    A(I);  
    .  
    .  
END.
```

## PASSING PARAMETERS

GENERAL (7.2)

PARAMETER LINKAGE (7.3)

VARIABLE PARAMETERS (7.4)

CONSTANT (VALUE) PARAMETERS (7.5)

PARAMETER SYNTAX (7.6)

PARAMETER SPECIFICATIONS (7.7)

TYPE CHECKING EXAMPLES (7.8)

UNIVERSAL LINKAGE (7.13)

FUNCTIONS VS PROCEDURES (7.15)

CHECK ON MOVING (7.16)



## PASSING PARAMETERS TO SUBPROGRAMS/PROCEDURES

(ALSO CALLED ARGUMENTS)

FORMAL PARAMETERS = THOSE DECLARED IN THE SUBPROGRAM BODY

ACTUAL PARAMETERS = THOSE PASSED TO (OR FROM) THE SUBPROGRAM  
BY THE CALLING PROGRAM

---

### FORTRAN

MUST AGREE IN NUMBER

ARGUMENTS AND FORMAL  
PARAMETERS MUST BE IN  
SAME ORDER.

NOT CHECKED BY THE COMPILER!

NOT CHECKED AT RUN-TIME!

NOT CHECKED BY THE COMPILER!

CAN BE DECLARED OR JUST  
USE TYPE DEFAULTS

SPECIFICATIONS MIXED IN WITH  
DECLARATIONS OF LOCAL  
VARIABLES

### PASCAL

MUST AGREE IN NUMBER

ARGUMENTS AND FORMAL  
PARAMETERS MUST BE IN  
SAME ORDER.

MUST AGREE IN TYPE!!!!

(THE COMPILER CHECK THIS!)

STRUCTURES PASSED AS PARAMETERS  
MUST ALSO AGREE IN SIZE!!!  
(PART OF TYPE CHECKING)

PARAMETERS MUST BE DECLARED  
(CALL THIS "SPECIFIED")

SPECIFICATIONS INCLUDED IN  
THE PROCEDURE LINE

## KINDS OF PARAMETER LINKAGE

### FORTRAN

#### BY LOCATION:

AN ADDRESS IS CALCULATED FOR EACH ACTUAL PARAMETER AND THESE ADDRESSES ARE LINKED INTO THE SUBPROGRAM (EXPRESSIONS ARE EVALUATED AND A LOCATION IS ASSIGNED TO STORE EACH EXPRESSION VALUE)

### PASCAL

#### OPTION:

EITHER: BY "ACCESS VALUE". AT THE TIME OF THE CALL, THE ADDRESS OF EACH LOCAL VARIABLE IS CALCULATED AND LINKED INTO THE SUBPROGRAM. HOWEVER, THE COMPILER WILL NOT ALLOW ASSIGNMENT TO THE FORMAL PARAMETERS. EXPRESSIONS IN THE ACTUAL PARAMETER LIST ARE EVALUATED AND LOCATIONS ARE ASSIGNED.

#### NOTE:

- THUS IT IS IMPOSSIBLE FOR THE PROCEDURE TO CHANGE THE VALUE OF THE ACTUAL PARAMETER.
- THESE ARE CALLED VALUE PARAMETERS

OR: BY REFERENCE (USING VAR): THE ACTUAL PARAMETER MUST BE EITHER:

- A VARIABLE
- AN ARRAY COMPONENT
- A RECORD COMPONENT

FOR EACH PARAMETER, THE VARIABLE OR COMPONENT ADDRESS IS CALCULATE AND LINKED INTO THE PROCEDURE AT THE TIME OF THE CALL.

NOTE: THESE ARE CALLED VARIABLE PARAMETERS

## KINDS OF PARAMETER LINKAGES (CONT.)

### BY FUNCTION-VALUE

THE VALUE OF A FUNCTION  
SUBPROGRAM IS RETURNED  
AS THE RESULT OF A  
FUNCTION REFERENCE.

### BY FUNCTION-VALUE

THE VALUE OF A FUNCTION  
SUBPROGRAM IS RETURNED  
AS THE RESULT OF A  
FUNCTION REFERENCE.

PASCAL  
VARIABLE PARAMETERS

```
PROGRAM MAIN
VAR I,J: INTEGER;
:
:
:
PROCEDURE SUB1 (VAR K: INTEGER);
:
:
:
BEGIN "SUB1"
:
:
:
END; "SUB1"
BEGIN "MAIN"
:
:
:
SUB1(I);
:
:
:
SUB1(J);
:
:
:
END. "MAIN"
```

RESULTS OF PROCEDURES MUST BE VARIABLE PARAMETERS.

VARIABLE PARAMETERS IN THE PROCEDURE HEADING ARE NOT ALLOCATED STORAGE SPACE, THEY USE THE STORAGE LOCATION ALLOCATED FOR THE CALLING PARAMETER. (EX. K POINTS TO THE STORAGE LOCATION OF I OR J WHEN SUB2 IS CALLED ABOVE).

PASCAL  
CONSTANT (VALUE) PARAMETERS

```
PROGRAM MAIN;  
VAR I,J: INTEGER;  
PROCEDURE SUB1 (K: INTEGER);  
VAR L: INTEGER;  
    BEGIN "SUB1"  
        :  
        :  
        "ILLEGAL AS K IS CONSTANT"  
        K:= K + 2;  
        L:= K * 4;  
        :  
        :  
    END "SUB1";  
BEGIN "MAIN"  
    :  
    :  
    I:= 2;  
    J:= I * 8;  
    :  
    :  
    SUB1 (I);  
    :  
    :  
    SUB1 (J);  
    :  
    :  
END. "MAIN"
```

- 
1. CONSTANT PARAMETERS ARE ASSIGNED STORAGE SPACE.
  2. VALUE OF CALLING PARAMETER IS NEVER CHANGED BY PROCEDURE.

PASCAL  
SYNTAX VARIATIONS

```
PROCEDURE JIM (I: INTEGER;  
                J: INTEGER;  
                W: REAL;  
                VAR X: REAL;  
                VAR Y: REAL;  
                VAR Z: INTEGER);
```

SAME AS:

```
PROCEDURE JIM (I,J: INTEGER; W: REAL;  
                VAR X, Y: REAL;  
                VAR Z: INTEGER);
```

(OR IT COULD ALL BE WRITTEN ON ONE LINE)

NOTE: IT IS A GOOD HABIT TO GROUP ALL THE CONSTANT  
PARAMETERS BEFORE THE VARIABLE PARAMETERS

## PARAMETER SPECIFICATIONS

MUST BE TYPE IDENTIFIER

### ILLEGAL

```
PROGRAM MAIN;  
VAR X: ARRAY (.1..10.) OF INTEGER;  
PROCEDURE A (Y: ARRAY (.1..10.) OF INTEGER) ;  
:  
:  
:  
:
```

### LEGAL

```
PROGRAM MAIN;  
TYPE ARR = ARRAY (.1..10.) OF INTEGER;  
VAR X: ARR;  
PROCEDURE A(Y:ARR);  
:  
:  
:
```

## TYPE CHECKING

|   | <u>FORTRAN</u>             | <u>PASCAL</u>                  |
|---|----------------------------|--------------------------------|
| C | MAIN PROGRAM               | <u>PROGRAM</u> MAIN;           |
|   | INTEGER I,J                | <u>VAR</u> I,J: INTEGER;       |
|   | :                          | <u>PROCEDURE</u> A(X: INTEGER; |
|   | :                          | <u>VAR</u> Y: REAL);           |
|   | I = 10                     |                                |
|   | CALL A(I,J)                | <u>BEGIN</u>                   |
| C | WOW, THIS IS LEGAL         | :                              |
| C | ACTUALLY USEFULL SOMETIMES | :                              |
|   | :                          | IF X<=20 THEN Y:= 5.6;         |
|   | :                          | :                              |
|   | END                        | <u>END</u> "A";                |
|   | SUBROUTINE A(X,Y)          | <u>BEGIN</u> "MAIN"            |
|   | INTEGER X                  | :                              |
|   | REAL Y                     | I:= 10 ;                       |
|   | :                          | A(I,J);                        |
|   | :                          | "THIS IS ILLEGAL AND           |
|   | IF (X.LE.20) Y= 5.6        | WON'T COMPILE AS J AND         |
|   | :                          | Y ARE DIFFERENT TYPES"         |
|   | :                          | :                              |
|   | RETURN                     | :                              |
|   | END                        | <u>END.</u>                    |



## TYPE CHECKING

### FORTRAN

```
C   MAIN
      REAL Z,Y
      ONE= 1.0
      Z = 1.0
      Y = 1.0
      CALL A (ONE)
      CALL A(Y+Z)
C   WOW
C   ALL LEGAL
      END

      SUBROUTINE A(X)
      X= X + 1.0
      RETURN
      END
```

### PASCAL

```
PROGRAM MAIN;
  CONST ONE = 1.0;
  VAR Z,Y : REAL;
  PROCEDURE A (VAR X: REAL);
    BEGIN X:= X+1.0
    END "A";
```

BEGIN

A(ONE);

Z:=1.0;

Y:=1.0;

A(Z+Y)

"WONT COMPILE"

END.

ILLEGAL

NOTE: THESE WOULD BE LEGAL  
IF THE PARAMETER WERE  
DECLARED AS A CONSTANT  
PARAMETER (IE NO VAR  
IN THE DECLARATION)

## TYPE CHECKING

|   |                 |  |
|---|-----------------|--|
| C | MAIN            | PROGRAM "MAIN";                                      |
|   | DIMENSION Y(10) | <u>TYPE</u> B= <u>ARRAY</u> (.1..3.) <u>OF</u> REAL; |
|   | :               | <u>VAR</u> Y: <u>ARRAY</u> (.1..10.) <u>OF</u> REAL; |
|   | :               | <u>PROCEDURE</u> A( <u>VAR</u> X: B);                |
|   | CALL A(Y(7))    | <u>BEGIN</u>   |
| C | WOW, LEGAL      | X(.4.):=1.0  |
|   | :               | <u>END</u> "A";                                      |
|   | :               | <u>BEGIN</u>   |
|   | END             | :  |
|   | SUBROUTINE A(X) | :  |
|   | DIMENSION X(4)  | A (Y(.7.) )  |
|   | :               | "WONT COMPILE -                                      |
|   | :               | <u>TYPE ERROR:</u>                                   |
|   | X(4)= 1.0       | REAL $\nexists$ <u>ARRAY OF REAL</u> "               |
|   | :               | :  |
|   | :               | :  |
|   | RETURN          | :  |
|   |                 | <u>END.</u>  |
|   | END             |  |

## TYPE CHECKING

```
PROGRAM "MAIN"  
TYPE B= ARRAY (.1..20.) OF REAL;  
VAR Y: ARRAY (.1..10.) OF REAL;  
PROCEDURE A(X: B);  
    BEGIN  
        :  
        :  
    END "A";  
BEGIN  
    :  
    :  
    :  
    A (Y);  
    "WONT COMPILE"  
    :  
    :  
    :  
END.
```

### SIZE ERROR:

ARRAY (.1..10.)  $\neq$  ARRAY (.1..20.)

## AN EXCEPTION!!!

THE SIZE CHECK IS RELAXED FOR STRINGS:

THE ACTUAL STRING MAY BE LESS THAN OR EQUAL TO THE  
FORMAL STRING.

```
PROGRAM MAIN;  
TYPE STRING8= ARRAY (.1..8.) OF CHAR  
VAR X: ARRAY (.1..10.) OF CHAR;  
    Y: ARRAY (.1..6.) OF CHAR;  
PROCEDURE A(Z: STRING8);  
    VAR I: INTEGER;  
    BEGIN  
        :  
        :  
    END "A";  
BEGIN  
    :  
    :  
    A(Y); "THIS IS LEGAL!"  
    A(X); "THIS IS ILLEGAL"  
    :  
    :  
END.
```

WARNING: Z(.7.) IS SOME VALUE BEYOND Y  
WARNING: IF Z WERE A VAR PARAMETER, THEN ASSIGNMENT TO Z(.7.)  
OR Z(.8.) WOULD OVERWRITE SOMETHING IN MEMORY BEYOND  
Y!!! AWFUL!!!!

## UNIVERSAL LINKAGE

SOMETIMES IT IS NECESSARY TO CHEAT ON PARAMETER TYPE CHECKING!

(USUALLY FOR I/O PROCEDURES)

THE ADDITIONAL LINKING MECHANISM UNIVERSAL (UNIV) REMOVES THE TYPE CHECK PART OF THE LINKAGE, BUT STILL ENFORCES THE SIZE CHECK.

(DOES NOT WORK FOR POINTER VARIABLES, WHICH ARE INTRODUCED LATER)

HENCE YOU NEED TO KNOW STORAGE REQUIREMENTS OF DIFFERENT TYPES:

| <u>DATA TYPES</u>        | <u>#WORDS OF STORAGE</u> |
|--------------------------|--------------------------|
| CHAR                     | 1                        |
| BOOLEAN                  |                          |
| LOWER..UPPER IDENTIFIERS |                          |
| INTEGER                  |                          |
| REAL                     |                          |
| SET                      | 8                        |
| STRING (M CHAR)          | M/2                      |

## LEGAL

```
PROGRAM MAIN;  
TYPE STRINGX= ARRAY (.1..16.) OF CHAR;  
VAR A: ARRAY (.1..2.) OF REAL;  
PROCEDURE STORE (X: UNIV STRINGX);  
  BEGIN  
    :  
    :  
    "USEFULL BECAUSE I/O  
    PROCEDURES HANDLE ONLY  
    CHARACTERS OR ARRAYS OF  
    CHARACTERS"  
    :  
    :  
    :  
  END "STORE";  
  
BEGIN  
  :  
  :  
  :  
  STORE (A);  
  :  
  :  
  :  
END.
```

COMMON UNIVERSAL TYPES ARE "LINE" AND "PAGE"  
WHICH ARE DEFINED IN THE STANDARD PREFIX.

## PASCAL

FUNCTION PARAMETERS MUST BE: CONSTANT

PROCEDURE PARAMETERS MAY BE: CONSTANT OR VARIABLE

## PROCEDURE AND FUNCTION CALLS

### FORTRAN

SUBROUTINES ARE CALLED BY A  
STATEMENT (CALL SUB1)

FUNCTIONS ARE ALWAYS CALLED AS  
A FACTOR IN AN EXPRESSION

```
C    MAIN PROGRAM
      .
      .
      .
      CALL SUB1
      CALL SUB2(A,B)
C    SUBROUTINE CALLS
      .
      .
      .
      F= FUNC1(C)
      IF (FUNC1(B).LT.1000) I=1
C    FUNCTION CALLS
      .
      .
      .
      END
```

### PASCAL

A ROUTINE CALLED AS A  
STATEMENT MUST BE A  
PROCEDURE CALL.

A ROUTINE CALLED AS A  
FACTOR IN AN EXPRESSION  
MUST BE A FUNCTION CALL.

```
PROGRAM MAIN;
      .
      .
      .
      BEGIN "MAIN"
      .
      .
      .
      SUB1
      SUB2(A,B);
      "PROCEDURE CALLS"
      .
      .
      F:= 'FUNC1(C);
      IF FUNC1(B) > 1000 THEN I=1;
      "FUNCTION CALLS"
      .
      .
      .
      END. "MAIN"
```



## FORTRAN FUNCTIONS

1. A VALUE IS ASSIGNED AND RETURNED AS THE FUNCTION NAME
2. VALUE RETURNED MUST BE ASSIGNED TO FUNCTION IDENTIFIER WITHIN THE FUNCTION
3. THE FUNCTION AND ITS ASSIGNED VALUE MAY BE OF DIFFERENT TYPE (EX. REAL= INTEGER)
4. PARAMETERS MAY RETURN A VALUE TO THE CALLING PROGRAM!!

```
C      MULTIPLICATION OF POSITIVE INTEGERS
C      MAIN
C      INTEGER I,J,K
C      BEGIN MAIN
      .
      .
      I= 2
      J= 3
      K= MULINT (I,J)
      .
      .
      END

      FUNCTION MULINT (L,M)
      INTEGER L,M,N
      N=0
100  IF(L.LE.0) GO TO 200
      N= N + M
      L= L - 1
      GO TO 100
200  MULINT= N
      RETURN
      END
```

NOTE: I.EQ.Ø AFTER  
FUNCTION COMPLETES

## PASCAL FUNCTIONS

1. A SINGLE VALUE IS RETURNED UNDER FUNCTION NAME.
2. THE VALUE RETURNED MUST BE ASSIGNED TO THE FUNCTION IDENTIFIER WITHIN THE FUNCTION BLOCK.
3. THE FUNCTION AND ITS ASSIGNED VALUE MUST BE OF COMPATIBLE TYPE
4. PARAMETERS CANNOT RETURN A VALUE ( PASSED AS CONSTANTS)

"MULTIPLICATION OF POSITIVE INTEGERS"

PROGRAM MAIN;

VAR I,J,K: INTEGER;

FUNCTION MULINT (L,M:INTEGER): INTEGER;

"L>0, M>0"

VAR L1,N : INTEGER;

BEGIN

L1:= L;

N:= 0;

WHILE L1>0 DO

BEGIN N:= N + M;

L1:= L1-1 END;

MULINT:= N

END "FUNCTION MULINT";

BEGIN "MAIN"

I:= 2;

J:= 3;

K:= MULINT(I,J);

END. "MAIN" "NOTE I,J ARE NOT CHANGED AFTER FUNCTION COMPLETES"

RESTRICTION

KSU (CIT) IMPLEMENTATION  
WILL NOT ALLOW

CAN BE DONE  
IN FORTRAN

FUNCTION NAMES OR PROCEDURE NAMES  
TO BE PASSED AS AN ACTUAL PARAMETER  
TO A SUBPROGRAM

NOTE: THIS CAN BE DONE IN THE  
WINSEN/WIRTH PASCAL REPORT

## GROUP 8: SIMPLE INPUT/OUTPUT

I/O PRIMITIVES

LINE ORIENTED ROUTINES

FOUR CONVERSION ROUTINES

## PRIMITIVES PROVIDED IN THE STANDARD PREFIX

### FOR THE PASCAL CONSOLE

IDENTIFY (HEADER: LINE).....INITIALIZATION TO IDENTIFY  
THE CALLING PROGRAM; LINE IS  
DEFINED IN THE STANDARD PREFIX  
AS A TYPE OF ARRAY OF 132  
CHARACTERS

ACCEPT (VAR C: CHAR) } ..... READ AND PRINT, RESPECTIVELY,  
DISPLAY ( C: CHAR) } ..... A SINGLE CHARACTER TO THE  
PASCAL CONSOLE; THE ASCII LINE  
FEED CHARACTER (IE CHR (10))  
IS USED TO TERMINATE EACH LINE  
OF INPUT AND OUTPUT

### FOR CHARACTER ORIENTED DEVICES (SEE ALSO GROUP 9)

WRITEARG .....USED TO IDENTIFY THE FILE OR  
DEVICE DRIVER TO BE ACCESSED

READARG .....USED TO CHECK THE SUCCESS OR  
FAILURE OF THE FILE OR DEVICE  
ACCESS

READ (VAR C: CHAR) } ..... INPUT AND OUTPUT, RESPECTIVELY,  
WRITE ( C: CHAR) } ..... OF A SINGLE CHARACTER FROM/TO  
THE FILE OR DEVICE DRIVER  
SPECIFIED USING WRITEARG.

## EXTENDED I/O ROUTINES (NOT IN STANDARD PREFIX)

RDLINE (NAME; IDENTIFIER; VAR INLINE: LINE; VAR R: BOOLEAN)

WRTLINE (NAME:IDENTIFIER; OUTLINE: LINE; VAR R: BOOLEAN)

USED TO READ/WRITE A LINE OF CHARACTERS  
FROM/TO A FILE OR DEVICE DRIVER; WHERE

NAME-----A STRING OF LENGTH 12 WHICH  
IS THE NAME OF A FILE OR AN  
I/O DRIVER, SUCH AS LISTED  
ON PAGE 10.4

INLINE, OUTLINE----A STRING OF LENGTH LINELENGTH  
(132); TERMINATED WITH THE  
CHARACTER EM (DEFINED AS CHR  
(25) IN THE PREFIX).

R-----RESULT: FALSE INDICTS AN I/O ERROR.

CONRDLN (VAR INLINE: LINE)-----SIMILAR TO RDLINE AND WRTLN,  
CONWRTLN (OUTLINE: LINE) EXCEPT THESE READ/WRITE FROM/TO  
THE PASCAL CONSOLE, AND THE  
LINE IS TERMINATED BY A NL  
CHARACTER.

## CONVERSION ROUTINES (NOT IN STANDARD PREFIX)

```
INT_TO_STR (INT, WIDTH: INTEGER;  
            VAR INDEX: INTEGER;  
            VAR R: CONVTErr;  
            VAR OUTLINE: LINE)
```

---INTEGER\_TO\_STRING CONVERSION, WHERE

INT .....INTEGER TO BE CONVERTED

WIDTH .....WIDTH OF "WINDOW" IN OUTLINE

INDEX .....STARTING POSITION OF "WINDOW" IN  
OUTLINE; SET TO STARTING POSITION  
OF NEXT POSSIBLE WINDOW

R .....RETURN PARAMETER:

CONVTErr.....ENUMERATION TYPE OF:

OK.....CONVERSION DONE

INDEXERR..ILLEGAL WINDOW POSITION

WIDTHErr..ILLEGAL WINDOW SIZE

LPRSN

RPRSN

NOTINT

OVERFLOW

NOTFIXED

TRANS.....I/O DEVICE ERROR

} USED LATER

FIXED\_TO\_STR (NUMB: REAL;  
                WIDTH, LPRECISION, RPRECISION: INTEGER;  
                VAR INDEX: INTEGER  
                VAR R: CONV TERR;  
                VAR OUTLINE; LINE)

---REAL NUMBER TO STRING CONVERSION, USING FIXED POINT  
NOTATION; WHERE

NUMB.....REAL NUMBER TO BE CONVERTED  
LPRECISION....NUMBER OF SPACES ALLOWED TO LEFT  
                    OF DECIMAL POINT, INCLUDING THE  
                    SIGN  
RPRECISION....NUMBER OF SPACES ALLOWED TO RIGHT  
                    OF DECIMAL POINT

NOTE: INDEX  $\geq$  LPRECISION + RPRECISION + 1

CONV TERR.....ADDITION VALUES ARE:  
                    LPRSN.....ILLEGAL LPRECISION  
                    RPRSN.....ILLEGAL RPRECISION



STR\_TO\_INT (VAR INT: INTEGER;  
            WIDTH: INTEGER;  
            VAR INDEX: INTEGER;  
            VAR R: CONVERR;  
            VAR INLINE; LINE)

---STRING TO INTEGER CONVERSION; WHERE

INT.....INTEGER VALUE DETERMINED

WIDTH.....WIDTH OF WINDOW TO BE PROCESSED;

            HOWEVER, IF THE WINDOW IS SET TO ZERO,  
            THEN THE WINDOW IS EXPANDED UNTIL AN  
            INTEGER IS CONVERTED OR UNTIL AN ERROR  
            IS DETECTED.

CONVERR.....ADDITIONAL VALUES ARE:

            NOTINT.....SCANNING SYNTAX ERROR

            OVERFLOW...INTEGER TOO LARGE TO CONVERT

STR\_TO\_FIXED (VAR NUMB; REAL;  
              WIDTH: INTEGER;  
              VAR INDEX: INTEGER,  
              VAR R: CONVERR;  
              VAR INLINE: LINE)

---STRING TO REAL NUMBER CONVERSION, ASSUMING A FIXED POINT  
NOTATION, WHERE

NUMBR.....REAL VALUE TO BE DETERMINED

CONVERR.....ADDITION VALUES ARE:

NOTFIXED.....SCANNING SYNTAX ERROR

GROUP 9: STANDARD PREFIX  
PROGRAM FORMAT (9.2)  
PURPOSE OF PREFIX (9.3)  
PREFIX CONSTANTS (9.4)  
PREFIX PROCEDURES (9.5)  
FILE PROCEDURES (9.6)  
ANNOTATED PREFIX (9.7)

"HEADER COMMENTS"

(OPTION, OPTION . . . )

STANDARD\_PREFIX\_

DECLARATIONS

(WITH\_OUT COMMENTS)

(ABOUT 120 LINES)

INCLUDES PROGRAM STATEMENT

PROGRAM P(VAR PARAM:ARGLIST);

YOUR\_DECLARATIONS\_HERE

BEGIN

"BODY OF PROGRAM HERE"

END.

DATA\_STARTS\_HERE

(IF ANY)

#

← SKIP IF NO OPTIONS SELECTED

NOTE: NAME OF PROGRAM IS "P"  
(WHICH CAN BE CHANGED, USING  
THE EDITOR); HOWEVER, THE  
PROGRAM IS INVOKED BY FILE  
NAME, WHICH MAY NOT BE "P".

### PURPOSES FOR PREFIX

- . DEFINES COMMONLY USED CONSTANTS
- . DEFINES COMMONLY USED TYPES
- . LISTS NAMES OF PROCEDURES WHICH ARE ENTRY POINTS TO SOLO SYSTEM
- . IDENTIFIES PARAM, WHICH IS USED TO PASS PARAMETERS TO AND FROM THE PROGRAM

## PREFIX CONSTANTS

NL = "NEW\_LINE" OR "LINE\_FEED"

FF = "FORM\_FEED"

CR = "CARRIAGE RETURN"

EM = "END\_OF\_MEDIUM\_MARK" FOR STREAM I/O.

PAGELength = 512 BYTES

LINELENGTH = 132 CHARS

IDLENGTH = 12 = MAX LENGTH FOR IDENTIFIERS USED AS PARAMETERS

MAXARG = 10 = MAX NUMBER OF ARGUMENT TO/FROM THE PROGRAM

## PREFIX PROCEDURES

## HEAP PROCEDURES

MARK, RELEASE

## I/O PROCEDURES

READ, WRITE \_ \_ \_ CHARACTER I/O, READER/PRINTER

IDENTIFY, ACCEPT, DISPLAY \_ \_ \_ CHARACTER I/O, PASCAL CONSOLE

IO TRANSFER \_ \_ \_ PAGE I/O TO DEVICE

IO MOVE \_ \_ \_ DEVICE CONTROL

## RELATED TYPES

ARRAYS OF CHAR: PAGE, LINE, IDENTIFIER

IO DEVICES: TYPEDEVICE, DISKDEVICE, TAPEDEVICE,  
PRINTDEVICE, CARDDEVICE.

IO OPERATIONS: INPUT, OUTPUT, MOVE, CONTROL

IO RESULTS: COMPLETE, INTERVENTION, TRANSMISSION,  
FAILURE, ENDFILE, ENDMEDIUM, STARTMEDIUM

OTHER\_CONSTANTS: WRITEEOF, REWIND, VPSPACE,  
BACKSPACE

## FILE PROCEDURES

OPEN, CLOSE

GET, PUT

LOOKUP, LENGTH

### RELATED TYPES

FILEKINDS: EMPTY, SCRATCH, ASCII, SEQCODE, CONCODE  
OTHER\_CONSTANTS: FILE, FILE ATTR(IBUTE)

## INTRA\_PROGRAM\_COMMUNICATION

READARG, WRITEARG \_ \_ \_ EXCHANGE PARAMETERS WITH OTHER PROGRAM

TASK \_ \_ \_ CHECK TASK NAME

RUN \_ \_ \_ RUN A PROGRAM

### RELATED TYPES

TASKKINDS: INPUTTASK, JOBTASK, OUTPUTTASK  
ARGTAGS: NILTYPE, BOOLTYPE, INTTYPE, IDTYPE, PTRTYPE  
PROGRESUITS: TERMINATED, OVERFLOW, POINTERERROR,  
RANGEERROR, VARIANTERROR, HEAPLIMIT, STACKLIMIT,  
CODELIMIT, TIMELIMIT, CALLERROR



## APPENDIX 1 - ANNOTATED PREFIX

"SPER BRINCH HANSEN

INFORMATION SCIENCE  
CALIFORNIA INSTITUTE OF TECHNOLOGY

UTILITY PROGRAMS FOR  
THE SOLO SYSTEM

18 MAY 1975"

\*\*\*\*\*

\*4 ANNOTATED5 \*  
\* PREFIX \*  
\*\*\*\*\*

(CHECK, NUMBER, TEST, XREF)

"4 A PROGRAM MAY BE PRECEDED BY COMPILER OPTIONS  
ENCLOSED IN PARENTHESES. ONLY THE FIRST CHARACTER  
OF AN OPTION IS EXAMINED. THE OPTIONS HAVE THE FOLLOWING  
EFFECT:

CHECK - THE GENERATED CODE WILL NOT MAKE THE  
FOLLOWING CHECKS:

- A) CONSTANT EVALUATION RANGE CHECKS  
(CONCURRENT AND SEQUENTIAL).
- B) USE OF NIL VALUED POINTERS (SEQUENTIAL  
ONLY);
- C) ILLEGAL VARIANT FIELD CHECKS (SEQUENTIAL  
ONLY).

NUMBER - THE GENERATED CODE WILL CONTAIN LINE  
NUMBERS OF THE PROGRAM'S TEXT FOR THE  
BEGINNING OF ROUTINES ONLY (NORMAL  
PRODUCTION MODE - SHOULD NOT BE USED  
WHILE DEBUGGING A PROGRAM.)

TEST - THE COMPILER WILL PRINT THE INTERMEDIATE  
CODE FOR ALL PASSES.

XREF - THE COMPILER WILL PRODUCE A CROSS REFERENCE  
TABLE. 5"

CONST NL = '(:10:)' : FF = '(:12:)' : CR = '(:13:)' : EM = '(:25:)' :

"4 THE CONSTANT NL (LINE FEED) IS THE LOGICAL  
END OF LINE CHARACTER IN PASCAL. IT IS TRANSLATED BY THE  
KERNEL INTO A CARRIAGE RETURN (CR) FOR COMPATIBILITY WITH  
OS/32-BT DRIVERS. THE END OF MEDIA CHARACTER (EM) IS USED  
FOR TERMINATING A STREAM OF CHARACTER ORIENTED TRANSMISSIONS  
(SEE READ AND WRITE.) 5"

CONST PAGLENGTH = 512;

TYPE PAGE = ARRAY (1..PAGLENGTH) OF CHAR;

"4 A PAGE IS THE STANDARD UNIT OF DATA IN THE SOLO SYSTEM.  
DATA IS STORED ON BOTH DISK AND TAPE AS PAGES. IT MAY BE  
CHANGED TO CHARACTER BY CHARACTER TRANSFERS BETWEEN PROCESSES  
BY THE SOLO DATA BUFFER MONITORS. 5"

CONST LINELENGTH = 132;

TYPE LINE = ARRAY (1..LINELENGTH) OF CHAR;

"4 A LINE IS THE UNIT OF TRANSFER TO THE LINE  
PRINTER. IT IS ALSO COMMONLY USED AS THE PARAMETER FOR

## ROUTINES THAT DISPLAY ERROR MESSAGES TO THE CONSOLE. 5"

CONST IDLENGTH = 12;

TYPE IDENTIFIER = ARRAY (1..IDLENGTH) OF CHAR;

"4 AN IDENTIFIER IS USED FOR PASSING PARAMETERS AND FILE NAMES.  
ALL FILENAMES MAY BE A MAXIMUM OF TWELVE (12) CHARACTERS  
UNDER SOLO. 5"

TYPE FILE = 1..2;

"4 UP TO TWO (2) FILES MAY BE ACCESSED DIRECTLY BY THE PROGRAM  
RUNNING IN THE JOB PROCESS AT ANY TIME. THESE FILES ARE  
MANIPULATED BY THE INTERFACE ROUTINES GET, PUT, OPEN, CLOSE, AND  
LENGTH. EITHER OF THE IOPROCESSES MAY ACCESS ONLY ONE FILE AT A  
TIME. 5"

TYPE FILEKIND = (EMPTY, SCRATCH, ASCII, SFCODE, CONCODE);

"4 THESE ARE THE STANDARD FILE TYPES USED BY SOLO. 5"

TYPE FILEATTR = RECORD

KIND: FILEKIND;

ADDR: INTEGER;

PROTECTED: BOOLEAN;

NOTUSED: ARRAY (1..5) OF INTEGER

END;

"4 THIS TYPE DESCRIBES A DISK FILE. IT IS RETURNED BY THE  
SOLO LOOKUP INTERFACE PROCEDURE. ADDR IS THE ADDRESS OF THE  
FILE'S PAGEMAP (SEE BRINCH-HANSEN: 'DISK SCHEDULING AT COMPILE  
TIME' [X]). 5"

TYPE IODEVICE =

(TYPEDEVICE, DISKDEVICE, TAPEDEVICE, PRINTDEVICE, CARDDEVICE);

"4 THIS IS AN ENUMERATION OF THE ACTUAL KERNEL PHYSICAL  
DEVICES. ANY ENUMERATION WITH FIVE (5) ELEMENTS WILL SUFFICE  
FOR DEVICE SPECIFICATION. IT IS THE POSITION IN THE ENUMERATION  
WHICH IS IMPORTANT. 5"

TYPE IOOPERATION = (INPUT, OUTPUT, MOVE, CONTROL);

"4 THESE ARE THE VALID OPERATIONS FOR PHYSICAL DEVICES. THE  
CONTROL OPERATION DESERVES SPECIAL ATTENTION. THE EXECUTION OF  
A CONTROL OPERATION TO THE DISKDEVICE CAUSES THE SYSTEM  
TO BE REINITIALIZED USING THE ARG FIELD OF THE IOPARAM AS THE  
DISK ADDRESS AT WHICH TO FIND THE CONCURRENT CODE TO BE LOADED  
AS THE NEW SYSTEM. THIS FACILITY IS USED BY THE SOLO START  
PROGRAM, FOR EXAMPLE. THE USE OF THE CONTROL OPERATION ON  
THE CONSOLE IS TO MAKE THE PROCESS WAIT FOR THE NEXT BELL KEY  
TO BE ENTERED ON THE CONSOLE. THIS FUNCTION IS NOT VERY USEFUL UNDER  
SOLO BECAUSE SOLO INCLUDES A SPECIAL PROCESS (THE WATCHDOG PROCESS)  
WHICH USES THIS FUNCTION TO RELOAD SOLO. 5"

TYPE IOARG = (WRITEFOR, REWIND, UPSPACE, BACKSPACE);

"4 THIS ENUMERATION DESCRIBES THE FUNCTIONS THAT  
MAY BE USED WITH THE MOVE OPERATION TO THE TAPEDEVICE.  
THEY ARE SELF EXPLANATORY. 5"

TYPE IORESULT =

(COMPLETE, INTERVENTION, TRANSMISSION, FAILURE,

ENDFILE, ENDLOADING, STARTMEDIA);

"4 THESE ARE THE VALID STATUSES RETURNED BY THE KERNEL  
IN RESPONSE TO I/O REQUESTS. THE MAPPING FROM  
OS/32-BT DEVICE INDEPENDENT STATUS TO THESE STATUSES IS

AS FOLLOWS:

OS/32-RT STATUS

PASCAL STATUS

00

COMPLETE

01

INTERVENTION

02

TRANSMISSION

03 (ANY BIT)

FAILURE

04

ENDFILE

05

ENDMEDIUM

THE STARTMEDIUM STATUS IS NOT IMPLEMENTED ON ANY  
CURRENTLY SUPPORTED DEVICE. 5"

TYPE IOPARAM = RECORD

OPERATION: IOOPERATION;

STATUS: IORESULT;

ARG: IOARG

END;

"4 THIS RECORD IS THE ACTUAL PARAMETER WHICH IS PASSED  
TO THE KERNEL I/O ROUTINES TO CONTROL I/O REQUESTS.  
REQUESTS ARE SENT TO THE KERNEL VIA THE SOLO IOTRANSFER  
INTERFACE PROCEDURE. 5"

TYPE TASKIND = (INPUTTASK, JOBTASK, OUTPUTTASK);

"4 THIS ENUMERATION LISTS THE THREE SOLO PROCESS PARTITIONS  
INTO WHICH A SEQUENTIAL PROGRAM MAY BE LOADED.  
A SEQUENTIAL PROGRAM MAY DETERMINE WHICH PARTITION  
IT IS IN THROUGH THE USE OF THE TASK INTERFACE  
FUNCTION. 5"

TYPE ARGTAG =

(NILTYPE, BOOCTYPE, INTTYPE, IDTYPE, PTRTYPE);

"4 THIS ENUMERATION SPECIFIES THE TYPES THAT  
PARAMETERS AND ARGUMENTS MAY ASSUME. THEY CORRESPOND  
TO NO ARGUMENT, BOOLEAN, INTEGER, IDENTIFIER, AND  
POINTER. 5"

TYPE POINTER = %BOOLEAN;

"4 THIS IS A DUMMY DEFINITION WHICH ALLOWS THE  
PASSING OF POINTERS BETWEEN SEQUENTIAL PROGRAMS  
(SEE SECTION ON PASSING POINTERS.) 5"

TYPE ARGTYPE = RECORD

CASE TAG: ARGTAG OF

NILTYPE, BOOCTYPE: (BOOL: BOOLEAN);

INTTYPE: (INT: INTEGER);

IDTYPE: (ID: IDENTIFIER);

PTRTYPE: (PTR: POINTER)

END;

"4 THIS RECORD DESCRIBES THE ARGUMENTS (AND PARAMETERS)  
WHICH MAY BE PASSED AMONG SEQUENTIAL PROGRAMS RUNNING  
UNDER SOLO. ARGUMENTS MAY BE PASSED AMONG THE THREE  
PROCESS PARTITIONS BY MEANS OF THE READARG AND  
WRITEARG INTERFACE PROCEDURES. A PROGRAM  
MAY ALSO BE CALLED AND PASSED A LIST OF PARAMETERS  
(AN ARGLIST) AND BE EXECUTED IN THE SAME PARTITION BY  
MEANS OF THE SOLO RUN ROUTINE. 5"

CONST MAXARG = 10;

TYPE ARGLIST = ARRAY (1..MAXARG) OF ARGTYPE;

"4 AS NOTED ABOVE A SEQUENTIAL PROGRAM MAY BE CALLED  
BY ANOTHER BY MEANS OF THE RUN INTERFACE PROCEDURE.

A MAXIMUM OF TEN (10) PARAMETERS MAY BE PASSED TO THE CALLED PROGRAM IN AN ARGLIST. 5"

TYPE ARGSEQ = (INP, OUT);

"4 THIS ENUMERATION SPECIFIES THE DESTINATION OF AN ARGUMENT REQUEST (READARG AND WRITEARG). THE JOB PROCESS MAY USE EITHER ENUMERATION, THE INPUT PROCESS MAY ONLY USE THE OUT AND THE OUTPUT PROCESS MAY ONLY USE THE INP AS ARGUMENT DESTINATIONS. 5"

TYPE PROGRESULT =

(TERMINATED, OVERFLOW, POINTERERROR, RANGEERROR, VARIANTERROR, HEAPLIMIT, STACKLIMIT, CODELIMIT, TIMELIMIT, CALLERROR);

"4 THIS ENUMERATION REPRESENTS THE STANDARD COMPLETION CODES GENERATED BY THE KERNEL UPON PROGRAM EXIT OR ERROR. TERMINATED REPRESENTS THE FACT THAT THE PROGRAM CONTROLLED ITS TERMINATION. ALL OTHER INSTANCES REPRESENT ERRORS DETECTED BY THE KERNEL OR SOLO. TIMELIMIT IS NOT CURRENTLY IMPLEMENTED IN SOLO. OVERFLOW IS DETECTED ON REAL (FLOATING POINT) BUT NOT INTEGER ARITHMETIC. ALSO, THE KERNEL MAPS 8/32 ILLEGAL INSTRUCTION AND MEMORY FAULT ERRORS INTO OVERFLOW. THIS IS GENERALLY CAUSED BY EXECUTING A PROGRAM FILE WHICH IS LABELED AS SEQCODE, BUT WHICH ACTUALLY IS NOT COMPILER GENERATED (E.G. AN ASCII FILE READ FROM TAPE.) 5"

PROCEDURE READ(VAR C: CHAR);

PROCEDURE WRITE(C: CHAR);

"4 THESE ROUTINES CONTROL CHARACTER BY CHARACTER DATA TRANSFER BETWEEN THE JOB AND THE INPUT AND OUTPUT PROCESSES. THE SYNCHRONIZATION OF THIS IS HANDLED BY A SOLO PAGEBUFFER MONITOR. THE TYPICAL PROGRAMMING SEQUENCE USED FOR READING A TEXT FILE IN THE JOB PROCESS IS AS FOLLOWS:

```
WRITEARG(INP, ARG);          SPECIFY FILE
REPEAT                        ( OF TYPE ASCII OR SEQCODE)
  READ(C);
  ... PROCESS CHARACTER
UNTIL C = EM;                END OF MEDIUM
READARG(INP, ARG);
IF NOT ARG.BOOL THEN ... :   ERRORS OCCURED
                              IN TRANSMISSION.
```

THE PROGRAMMING SEQUENCE FOR WRITING A TEXT FILE CHARACTER BY CHARACTER TO THE OUTPUT PROCESS IS AS FOLLOWS:

```
WRITEARG(OUT, ARG);          SPECIFY FILE
WHILE NOT DONE_GENERATING DO ('NEXT' OR TYPE SEQCODE)
BEGIN
  ...GENERATE CHARACTER C;
  WRITE(C);
END;
C:= EM;
WRITE(C);
IF FILE_IS_NEXT THEN
BEGIN
  READARG(OUT, ARG);
  LENGTH_OF_FILL_IN_PAGES:= ARG.INT
END;
```

READARG(OUT,ARG);

IF NOT ARG.BOOL THEN ... ;

ERRORS OCCURED  
IN TRANSMISSION. 5"

PROCEDURE OPEN(F: FILE; ID: IDENTIFIER; VAR FOUND: BOOLEAN);

PROCEDURE CLOSE(F: FILE);

PROCEDURE GET(F: FILE; P: INTEGER; VAR BLOCK: UNIV PAGE);

PROCEDURE PUT(F: FILE; P: INTEGER; VAR BLOCK: UNIV PAGE);

FUNCTION LENGTH(F: FILE): INTEGER;

"4 THESE ROUTINES ALLOW THE SEQUENTIAL PROGRAM ACCESS TO THE SOLO RESIDENT FILE SYSTEM. THE PROCEDURE OPEN READIES THE FILE WITH THE GIVEN IDENTIFIER FOR DATA TRANSFER. AFTER OPEN HAS BEEN CALLED THE FILE IS REFERRED TO BY THE FILE ENUMERATION (1 OR 2). CLOSE IS THE INVERSE OF OPEN WHICH PURGES THE ACCESSABILITY OF THE GIVEN FILE ENUMERATION. GET AND PUT ALLOW FOR THE TRANSFER FROM OR TO THE FILE PAGE INDEXED BY P. LENGTH RETURNS THE LENGTH (IN PAGES) OF THE GIVEN FILE ENUMERATION (IT MUST CURRENTLY BE OPENED.) 5"

PROCEDURE MARK(VAR TOP: INTEGER);

PROCEDURE RELEASE(TOP: INTEGER);

"4 THESE TWO ROUTINES ARE USED TO CONTROL THE DYNAMIC STORAGE CAPABILITIES OF SEQUENTIAL PASCAL. THE PROCEDURE MARK RETURNS AN INTEGER CORRESPONDING TO THE CURRENT HEAP TOP. THE PROCEDURE RELEASE REVERSES THIS AND SETS THE HEAP TOP TO THE PROGRAM SUPPLIED VALUE. (SEE THE SECTION ON PASSING POINTERS). 5"

PROCEDURE IDENTIFY(HEADER: LINE);

PROCEDURE ACCEPT(VAR C: CHAR);

PROCEDURE DISPLAY(C: CHAR);

"4 THESE ROUTINES ALLOW THE SEQUENTIAL PROGRAM ACCESS TO THE PASCAL CONSOLE. THE PROCEDURE IDENTIFY SHOULD BE EXECUTED IN THE INITIALIZATION AND AFTER CALLING ANOTHER PROGRAM (USING RUN.) THE SOLO TYPERESOURCE MONITOR AND TERMINAL CLASS KEEP TRACK OF PROGRAM IDENTITIES AND DISPLAY THEM ON THE CONSOLE WHEN NECESSARY. THE HEADER IS A PROGRAMMER DEFINED STRING (E.G. THE PROGRAM NAME.) ACCEPT IS USED TO RECEIVE A CHARACTER FROM THE CONSOLE AND DISPLAY IS USED TO SEND A CHARACTER. THE ASCII LINE FEED CHARACTER (DECIMAL CODE 10) IS USED TO TERMINATE A LINE OF TRANSMISSION TO THE CONSOLE, BOTH ON INPUT AND OUTPUT. 5"

PROCEDURE READPAGE(VAR BLOCK: UNIV PAGE; VAR EOF: BOOLEAN);

PROCEDURE WRITEPAGE(BLOCK: UNIV PAGE; EOF: BOOLEAN);

"4 THESE TWO ROUTINES ALLOW FOR TRANSFER OF DATA BETWEEN THE INPUT, JOB, AND OUTPUT PROCESSES. LIKE READ AND WRITE THE SYNCHRONIZATION IS HANDLED BY THE SOLO PAGEBUFFER MONITOR. HERE, HOWEVER, THE END OF MEDIUM (EOM) CHARACTER DOES NOT SIGNIFY THE END OF TRANSMISSION. INSTEAD, WITH EACH CALL A BOOLEAN EOF IS INCLUDED. IF THIS BOOLEAN IS TRUE, IT SIGNIFIES THAT THIS PAGE



IS EMPTY AND THAT THE LAST PAGE TRANSFERRED IS THE LAST. AS IN USING READ AND WRITE, THE JOB PROCESS MUST FIRST INSTRUCT THE IO PROGRAM TO LOAD THE CORRECT PROGRAMS INTO THE IO PROCESSES BY USING THE WRITEARG ROUTINE AND FIND THE STATUS OF THE COMPLETED TRANSFER AFTER EOF USING THE READARG ROUTINE. 5"

PROCEDURE READLINE(VAR TEXT: UNIV LINE);

PROCEDURE WRITELINE(TEXT: UNIV LINE);

"4 THESE ROUTINES ARE NO-OPS TO THE JOB PROCESS. IN THE INPUT PROCESS, READLINE MAY BE USED TO REQUEST A CARD IMAGE FROM THE SOLO CARDPROCESS PROCESS. SIMILARLY THE OUTPUT PROCESS MAY USE THE WRITELINE PROCEDURE TO SEND A LINE TO THE PRINTERPROCESS PROCESS. 5"

PROCEDURE READARG(S: ARGSEQ; VAR ARG: ARGTYPE);

PROCEDURE WRITEARG(S: ARGSEQ; ARG: ARGTYPE);

"4 THESE TWO PROCESSES HAVE BEEN REFERED TO SEVERAL TIMES ABOVE. THEY ARE THE PRIMARY SOURCE OF PROCESS SYNCHRONIZATION IN SEQUENTIAL PASCAL UNDER SOLO. INITIALLY SOLO LOADS THE IO PROGRAM INTO THE JOB PROCESS PARTITION AND THE IO PROGRAM INTO BOTH THE INPUT AND OUTPUT PROCESS PARTITIONS. SO ACCEPTS COMMANDS FROM THE CONSOLE AND LOADS THE APPROPRIATE PROGRAMS USING THE RUN ROUTINE. IT IS THEN THE RESPONSIBILITY OF THE SEQUENTIAL PASCAL PROGRAM TO NOTIFY THE IO PROGRAM OF WHICH DRIVER SHOULD BE LOADED IN THE INPUT OR OUTPUT PROCESS PARTITIONS. THIS IS DONE USING THE WRITEARG ROUTINE WITH THE NAME OF THE (SEQCODE) DRIVER TO BE LOADED, OR THE (ASCII) FILE TO BE TRANSFERED FROM DISK (USING THE DISK DRIVER). WHEN TRANSFER IS COMPLETED, THE JOB PROCESS DETERMINES THE STATUS OF THE COMPLETE TRANSFER FROM THE IO PROGRAM BY USE OF THE READARG ROUTINE AS SHOWN IN READ AND WRITE ABOVE. PROGRAMS WRITTEN TO EXECUTE IN THE INPUT AND OUTPUT PROCESS PARTITIONS NEED NOT USE THESE ROUTINES AS ALL COORDINATION IS PERFORMED BY THE IO PROGRAM. 5"

PROCEDURE LOOKUP(ID: IDENTIFIER; VAR ATTR: FILEATTR; VAR FOUND: BOOLEAN);

"4 THIS ROUTINE ALLOWS THE SEQUENTIAL PASCAL PROGRAM TO DETERMINE WHETHER A GIVEN FILE EXISTS AND IF SO WHAT ITS ATTRIBUTES ARE. IT USES THE RESIDENT SOLO FILE ROUTINE TO SEARCH THE DISK CATALOG. 5"

PROCEDURE IOTRANSFER

(DEVICE: IODEVICE; VAR PARAM: IOPARAM; VAR BLOCK: UNIV PAGE);

"4 THIS ROUTINE ALLOWS THE SEQUENTIAL PASCAL PROGRAM TO ACCESS A PHYSICAL DEVICE (E.G. THE TAPE DRIVE) FOR DATA TRANSFER. THE PARAMETERS ARE DESCRIBED IN THEIR TYPE DEFINITION ABOVE. 5"

PROCEDURE IOMOVE(DEVICE: IODEVICE; VAR PARAM: IOPARAM);

"4 THIS ROUTINE ALLOWS FOR THE CONTROL OF POSITIONS OF PHYSICAL DEVICES. IT IS CURRENTLY ONLY IMPLEMENTED FOR THE TAPE (TAPEDEVICE). 5"

FUNCTION TASK: TASKKIND:

"4 THIS FUNCTION ALLOWS THE SEQUENTIAL PASCAL PROGRAM TO DETERMINE INTO WHICH PROCESS PARTITION IT HAS BEEN LOADED. THIS IS NECESSARY TO DETERMINE WHAT FACILITIES ARE AVAILABLE THROUGH THESE INTERFACE ROUTINES. 5"

PROCEDURE RUN(ID: IDENTIFIER: VAR PARAM: ARGLIST:

VAR LINE: INTEGER: VAR RESULT: PROGRESULT):

"4 AS NOTED ABOVE, THE RUN PROCEDURE ALLOWS A SEQUENTIAL PASCAL PROGRAM TO CALL ANOTHER PROGRAM. THE IDENTIFIER IS THE NAME OF THE SEPCODE DISK FILE CONTAINING THE DESIRED PROGRAM. THE LINE WILL IDENTIFY THE LINE NUMBER IN ERROR IN THE EVENT OF ABNORMAL TERMINATION. THE PARAMETERS NEEDED BY THE PROGRAM SHOULD BE SUPPLIED IN THE PARAM ARGLIST. THE FIRST OF THESE (PARAM(.1.)) IS STANDARDLY A BOOLEAN WHICH WILL BE RETURNED BY THE CALLED PROGRAM TO NOTIFY WHETHER IT COMPLETED PROCESSING NORMALLY. 5"

PROGRAM P(VAR PARAM: ARGLIST):

"4 THIS IS THE PROGRAM STATEMENT WHICH SIGNIFIES THE END OF THE PREFIX. IT ALSO CONTAINS THE PARAMETERS WHICH HAVE BEEN PASSED TO THIS PROGRAM. THEY MAY BE ACCESSED AS PARAM(.I.). "

"INSERT YOUR PROGRAM HERE AND YOU'RE READY TO GO. GOOD LUCK!"

## GROUP 10: RUNNING THE PROGRAM

LOADING SOLO (10.2)

CONSOLE COMMANDS (10.3)

SAMPLE RUN (10.5)

SAMPLE PROGRAM (10.6)

COMPILER OPTIONS (10.7)

COMPILER ERROR MESSAGES (10.8)

RUN TIME ERROR MESSAGES (10.12)



## LOADING SOLO/PASCAL AS AN OS-32M/T TASK

(Assume all necessary files are initialized on disk., see  
Pascal Packaging Manual.)

\$COPY;      OS-32M/T

"AT THE OPERATORS CONSOLE"

PASCAL CAR, SYS2;PASCAL.VD

"BRING UP PASCAL UNDER SOLO AT CARROSEL"

"VIRTUAL DISK = SYS2:PASCAL.VD"

"\$COPY IS ON"

"SYSTEM CUES USER WITH 'DO:' AT PASCAL TERMINAL"

## GETTING HELP WITH PASCAL OPERATOR COMMANDS

### type

HELP . . . . . cues user to type LIST

command\_name\_w/o\_parameters. . . . . explains the parameters.

e.g:



command\_name\_with\_parameters . . . . . executes command

object\_program\_name\_with\_any\_parameters . . . . . executes program

DO:

COPY

COPY:

TRY AGAIN

COPY(SOURCE, DESTINATION: IDENTIFIER)

DO:

CONCAT

CONCAT:

TRY AGAIN

CONCAT(SOURCE(.1.), ..., SOURCE(.N.), DESTINATION: IDENTIFIER)

DO:

## SELECTED COMMANDS AT THE PASCAL OPERATORS CONSOLE

### FILE OPERATIONS

COPY . . . . . copies a file  
DO . . . . . executes a command or a file of commands  
CONCAT . . . . . concatenate files  
FILE . . . . . stores a file on disk  
LIST . . . . . list all files of selected types

these use "devices":

#### INPUT

CARD = card reader  
APPEND = card + append standard prefix  
CONSOLE= PASCAL console  
DISK = virtual disk

#### OUTPUT

PRINTER = virtual printer  
REMOVE = remove prefix PRINTER

### EDITING

EDIT . . . . . simple editor from C.I.T.  
PEDIT. . . . . similar to OS-32M/T editor

### RUN

RUN . . . . . compile and run  
SPASCAL . . . . . compiles a program  
name. . . . . executes named PASCAL object program

## SAMPLE RUN

```
"AT THE PASCAL CONSOLE"
KSU PASCAL INTERPRETER R00-00
DO:
"READ CARDS"
COPY(CARDS,MYFILE)
DO:
"EDIT SAMPLE PROGRAM"
PEDIT(MYFILE)
PEDIT:
RE
  38      END;
EOF
CH;/./
  38      END.
EN
DO:
CONCAT(PREFIX,MYFILE,MYFILE)
DO:
"COMPILE"
SPASCAL(MYFILE,PRINTER,OBFFILE)
DO:
"EXECUTE WITH I/O TO CONSOLE"
OBFFILE
THIS IS A STRING
THIS IS A STRING
COMPARE
OBFFILE
THIS IS A STRING
THIS IS
NO COMPARE
OBFFILE
THIS IS A STRING
THIX IS A STRING
NO COMPARE
```

SAMPLE PROGRAM (PREFIX OMITTED)

```
VAR STRING1,STRING2:LINE;  
    I:INTEGER; OK:BOULCAN;
```

```
PROCEDURE READTEXT(VAR TEXT:LINE);  
VAR I:INTEGER; C:CHAR;  
BEGIN
```

```
    I:=0;  
    REPEAT  
        I:=I+1;  
        ACCEPT(C);  
        TEXT(.I.):=C;  
    UNTIL C=NL;
```

```
END;
```

```
PROCEDURE WRITETEXT(TEXT:LINE);
```

```
VAR I:INTEGER; C:CHAR;  
BEGIN
```

```
    I:=0;  
    REPEAT  
        I:=I+1;  
        C:=TEXT(.I.);  
        DISPLAY(C);  
    UNTIL C=NL;
```

```
END;
```

```
BEGIN
```

```
    READTEXT(STRING1);
```

```
READTEXT(STRING2);
```

```
    I:=0;
```

```
    OK:=TRUE;
```

```
REPEAT
```

```
    I:=I+1;
```

```
    IF STRING1(.I.)<>STRING2(.I.) THEN OK:=FALSE;
```

```
UNTIL (STRING1(.I.)=NL) OR (STRING2(.I.)=NL) OR (OK=FALSE);
```

```
    IF OK
```

```
        THEN WRITETEXT('COMPARE(:10:))')
```

```
        ELSE WRITETEXT('NO COMPARE(:10:))')
```

```
END.
```

## COMPILER OPTIONS

NUMBER . . . . only procedures declared will be numbered  
(reduces amount of code X 25%)

CHECK . . . . (really no check) eliminates run time  
checks for legal enumeration\_values, non\_NIL  
printer\_values.

TEST . . . . . for debugging the compiler.

XRED . . . . . produce a cross reference table.

Default: none of the above.

## COMPILATION ERROR MESSAGES

Example:

|                         |                     |                                 |
|-------------------------|---------------------|---------------------------------|
| ***** PASS 2            | LINE 91             | CONSTANT DEF SYNTAX             |
| ***** PASS 3            | LINE 210            | INVALID NAME USAGE              |
| <del>***** PASS 3</del> | <del>LINE 215</del> | <del>INVALID SUBSCRIPTING</del> |

### PASS1 LEXICAL ANALYSIS

- ENDLESS COMMENT. •
- INVALID NUMBER. •
- TABLE OVERFLOW. •
- INVALID STRING. •
- BAD CHARACTER. •

'SQL PROGRAM.    ')  
'DECLARATION.    ')  
'CONSTANT DFN.    ')  
'TYPE DFN.        ')  
'TYPE.            ')  
'ENUMERATION TYP.  ')  
'SUBRANGE TYPE.    ')  
'SET TYPE.        ')  
'ARRAY TYPE.       ')  
'RECORD TYPE.      ')  
'STACK LENGTH.     ')  
'VAR DECLARATION.  ')  
'ROUTINE.          ')  
'PROCEDURE.        ')  
'FUNCTION.          ')  
'WITH STMT.        ')  
'PARAMETER.        ')  
'BODY.             ')  
'STMT LIST.        ')  
'STATEMENT.        ')  
'ID STMT.           ')  
'ARGUMENT.          ')  
'COMPOUND STMT.     ')  
'IF STMT.           ')  
'CASE STMT.        ')

'LABEL LIST.        ')  
'WHILE STMT.        ')  
'REPEAT STMT.       ')  
'FOR STMT.           ')  
'CYCLE STMT.        ')  
'EXPRESSION.        ')  
'VARIABLE.           ')  
'CONSTANT.           ')  
'INIT STMT.          ')  
'TERMINATION.        ')  
'PREFIX.             ')  
'INTERFACE.          ')  
'POINTER TYPE.      ')

'SYNTAX.'



PASS 4      SEMATIC CHECKS

```
( 'UNRESOLVED ROUTINE.      '
'AMBIGUOUS IDENTIFIER.    '
'COMPILER ABORT.          '
'INVALID CONSTANT.        '
'INVALID SUBRANGE.        '
'MISSING ARGUMENT.        '
'NOT A ROUTINE.           '
'TOO MANY ARGUMENTS.      '
'LABEL VALUE TOO LARGE.   '
'INVALID LABEL.           '
'AMBIGUOUS LABEL.         '
'INVALID WITH VARIABLE.   '
'INVALID INITIALIZATION.  '
'NOT A FUNCTION.          '
'INVALID NAME USAGE.      '
'INVALID SELECTION.       '
'INVALID SUBSCRIPTING.    '
'INVALID INTERFACE.      '
'INVALID CALL.            '
'INVALID POINTING.        '
'INVALID RESOLUTION.      '
'
```

PASS 4      MORE SEMATIC CHECKS

```
'INVALID NESTING.          '
'ADDRESS OVERFLOW.        '
'ACTIVE VARIABLE.         '
'QUEUE VARIABLE.          '
'NESTED PROCESS.          '
'INVALID ENTRY VARIABLE.  '
'INVALID FUNCTION TYPE.   '
'
'RECORD ENUMERATION.      '
'LONG ENUMERATION.        '
'INVALID INDEX TYPE.      '
'INVALID MEMBER TYPE.     '
'PROCESS STACK USAGE.    '
'
'INVALID PARAMETER.       '
'COMPILER ABORT.          '
'ODD LENGTH STRING TYPE.  '
'INVALID RESOLUTION.      '
'INVALID TAG TYPE.        '
'RECORD POINTER TYPE.     '
'
```

PASS 5

MORE SEMANTIC ANALYSIS

'COMPILER ABORT. '  
'OPERAND TYPE. '  
'NOT A VARIABLE. '  
'NOT ASSIGNABLE. '  
'INVALID INITIALIZATION. '

PASS 6

CODE GENERATION AND FLOW ANALYSIS

'TOO MUCH STACK. '  
'TOO MUCH CODE. '

## RUN TIME ERROR MESSAGES

### FORM1

TASK NAME: SYSTEM LINE "LINE NUMBER"

TASK NAME: "ERROR MESSAGE"

TASK NAME: TASK PAUSED

### FORM2

TASK NAME: JOB LINE "LINE NUMBER"

TASK NAME: "ERROR MESSAGE"

TASK NAME: TASK PAUSED

### ERROR MESSAGES

- 1) OVERFLOWERROR . . . . . arithmetic error
- 2) POINTERERROR . . . . . illegal pointer
- 3) RANGEERROR . . . . . illegal value for an enumeration
- 4) VARIANTERROR. . . . . illegal value for an enumeration in a record
- 5) HEAPLIMIT . . . . . exceed size of HEAP
- 6) STACKLIMIT. . . . . exceed size of execution stack

### Examples

\*PASCAL CAR, USR6:A/PASCAL.VD2

\*11:28:17 CAR:SYSTEM LINE 280

\*11:28:17 CAR:RANGE ERROR

\*11:38:17 CAR:TASK PAUSED

\*T CAR

\*CAN

\*11:28:17 CAR:END OF TASK 255

GROUP 11: DIFFERENCES BETWEEN PASCAL  
REPORT AND SPASCAL (11.2)

## DIFFERENCES

### USER MANUAL AND REPORT

{COMMENT}

begin (UPPER AND LOWER CASE)

ARRAY [1..10] OF INTEGER

↑ (FOR POINTERS)

PACKED

AND

FILE

FORWARD (NOT RESERVED)

GO TO

LABEL

NIL

NOT HERE

PROGRAM

### KSU IMPLEMENTATION

"COMMENT"

BEGIN (UPPER CASE ONLY)

ARRAY (.1..10.) OF INTEGER

@ (FOR POINTERS)

NOT IN KSU IMPLEMENTATION

&

NOT IN KSU IMPLEMENTATION

FORWARD (RESERVED WORD)

NOT IN KSU IMPLEMENTATION

NOT IN KSU IMPLEMENTATION

NOT IN KSU IMPLEMENTATION

UNIV

PREFIX → PREFIX ROUTINES.  
PROGRAM

**A SEQUENTIAL PASCAL MANUAL FOR FORTRAN PROGRAMMERS**

**by**

**JERRY DEAN RAWLINSON**

**B. S., University of Illinois, 1964**

---

**AN ABSTRACT OF A MASTER'S REPORT**

**submitted in partial fulfillment of the  
requirements for the degree**

**MASTER OF SCIENCE**

**Department of Computer Science**

**KANSAS STATE UNIVERSITY  
Manhattan, Kansas**

**1977**

This report is designed to serve as an instructional aid in the introduction of FORTRAN programmers to the Kansas State University (KSU) implementation of the programming language Sequential Pascal.

The report has been structured to take maximum advantage of the FORTRAN programmer's previously acquired knowledge of the FORTRAN language. Basic language terms are not defined as the programmer is assumed to be familiar with them. Based on this assumption, FORTRAN and Sequential Pascal examples are presented in a sequence designed to allow the programmer to quickly grasp the similarities and differences between the two languages. Typical FORTRAN programming problems are presented along with the corresponding Sequential Pascal solution. This approach allows the user to make direct comparisons between the two languages and provides for quick reference when the user wishes to find a Sequential Pascal solution to a typical FORTRAN programming problem.

Significant differences between the two languages are emphasized through the use of illustrations and warning statements. Programming examples are also used to introduce the user to Sequential Pascal capabilities which can not be duplicated in FORTRAN.

The use of this report as an instructional aid should significantly reduce the time required to train a FORTRAN programmer to write Sequential Pascal programs.