SAN DIEGO: COMPUTER CENTER
LA JOLLA, CALIFORNIA 92037

B6700 ALGOL USERS' GUIDE

1 SEPTEMBER 1972
(REVISED)

UNIVERSITY OF CALIFORNIA – (Letterhead for interdepartmental use)

PREFACE

The intent of this manual is to describe the ALGOL language used on the Burroughs B6700 computer. The text is separated into three chapters. The first describes only a small subset of the language in very elementary terms. Although anyone with a firm knowledge of any programming language will find parts of this chapter trivial, it should present an overview of the language to beginners as well as experienced programmers.

Chapter two is as much a description of ALGOL 60 as of B6700 ALGOL, and an attempt was made to point out the differences. This chapter also deals with standard I/O - a subject intentionally deleted from the specifications for ALGOL 60.

Chapter three deals with the various facilities of B6700 ALGOL, which constitute extensions to ALGOL 60. These include list structures comparable to Fortran namelists, additional loop control mechanisms, and extensive character manipulation constructs.

It is hoped that the text is both clearly and accurately presented. Comments and criticisms may be directed to the Computer Center.

TABLE OF CONTENTS

Chapter One

0-0 Introduction . . . . Page 3
0-1 Using the Computer with ALGOL. . . . . Page 3
1-0 ALGOL Programming - The Formal Aspects . . . . Page 6
1-1 Representation of Numbers . . . . Page 6
1-2 Identifiers . . . . Page 8
1-3 Simple Variables . . . . Page 8
2-0 Replacement Statements and Simple Arithmetic Expressions . . . . Page 10
2-1 Simple Printer Output with B6700 ALGOL . . . . Page 11
2-2 Declarations . . . . Page 12
2-3 Basic Structure of ALGOL Programs . . . . Page 14
2-4 The IF Statement - Simple Boolean Expressions . . . . Page 15
2-5 Labels and the GO TO Statement . . . . Page 16
2-6 The FOR Statement . . . . Page 18
2-7 Standard Functions . . . . Page 19
2-8 Operations on Integers . . . . Page 20
2-9 Arrays and Subscripted Variables . . . . Page 21
2-10 Input and More on Output . . . . Page 23
2-11 Compound Statements . . . . Page 26
2-12 Blocks . . . . Page 27
2-13-1 Real Procedures . . . . Page 28
2-13-2 Untyped Procedures . . . . Page 30
2-14 Comments . . . . Page 31

Chapter Two

3-0 Introduction . . . . Page 32
3-1 The Metalanguage . . . . Page 32
4-0 Boolean Expressions and Boolean Algebra . . . . Page 34
4-1 Boolean Variables . . . . Page 34
4-2 Boolean Algebra . . . . Page 34
4-3 Boolean Expressions . . . . Page 35
4-4 General Boolean Expressions - The IF-Clause . . . . . Page 36
4-5-0 Parentheses in Boolean Expressions . . . . Page 37
4-5-1 A Note on Standards . . . . Page 37
5-0 General Arithmetic Expressions . . . . Page 38
5-1 Use of Parentheses . . . . Page 38
5-2 Relations . . . . Page 38
6-0 Control Statements . . . . Page 40
6-1 Labels and Switches . . . . Page 40
6-2 Designational Expressions . . . . Page 41
6-2-1 Standards Again . . . . Page 41
6-3 General Switches . . . . Page 41
7-0 FOR Statement. . . . . Page 42
7-1 Real Loop Variables . . . . Page 42
7-2 The Reserved Word WHILE . . . . Page 42
7-3 General FOR-Statement . . . . Page 43
8-0 Formatted Input/Output . . . . Page 45
8-1 The WRITE Statement . . . . Page 45
8-2 Format Specifications . . . . Page 46

8-2-1 The I and X Editing Phrases. . . . . Page 46
8-2-2 Strings . . . . Page 48
8-2-3 Slash . . . . Page 48
8-2-4 Repeat Factors . . . . Page 49
8-2-5 Unlimited Groups . . . . Page 49
8-2-6 General Guidelines . . . . Page 50
8-3-1 Decimal Points . . . . Page 50
8-3-2 E Phrase . . . . Page 51
8-3-3 R Phrase . . . . Page 51
8-3-4 Accuracy . . . . Page 51
8-3-5 Scale Factor . . . . Page 51
8-3-6 L Phrase . . . . Page 52
8-4 Variable Format . . . . Page 52
8-5 Carriage Control . . . . Page 52
8-6 Formatted Input . . . . Page 53
8-6-1 Slash and X . . . . Page 53
8-6-2 I-Phrase . . . . Page 54
8-6-3 E and F Phrases . . . . Page 54
8-6-4 R-Phrase . . . . Page 54
8-6-5 L-Phrase . . . . Page 54
9-0 Block Structure . . . . Page 55
9-1 Local and Global Identifiers . . . . Page 56
9-2 Restrictions on Use of Labels . . . . Page 56
9-3 Arrays . . . . Page 57
9-4 Own Variables . . . . Page 57
10-0 Procedures . . . . Page 58
10-1 Local and Global Variables . . . . Page 58
10-2 Call by Name and Call by Value . . . . Page 58
10-3 Recursive Invocation . . . . Page 59
10-4 Parameter Delimiters . . . . Page 60
10-5 Side Effects . . . . Page 60
11-0 Comments . . . . Page 62

Chapter Three

12-0 Introduction . . . . Page 63
13-0 List Structures . . . . Page 64
13-1 IO Switches . . . . Page 64
13-2 Control Statements . . . . Page 66
13-3 Replacement Expressions . . . . Page 67
13-4-1 Define Declarations . . . . Page 68
13-4-2 Parametric Define . . . . Page 68
14-0 Extended Arithmetic Capacity . . . . Page 69
14-1 Intrinsic Functions . . . . Page 70
15-0 Forward Declarations . . . . Page 72
16-0 The B6700 . . . . Page 73
16-1 Number Representation . . . . Page 73
17-0 Partial-Word Operations . . . . Page 75
17-1 Concatenation . . . . Page 75
17-2 Type Transfer Functions and Masking . . . . Page 77
18-0 String Manipulation . . . . Page 78
18-1 Fill Statement . . . . Page 78
18-2 Additional IO Facilities . . . . Page 79
18-2-1 A Format . . . . Page 79
18-2-2 C Format . . . . Page 80
18-2-3 O Format . . . . Page 80

18-2-4 H and K Formats . . . . Page 80
19-0 Pointers . . . . Page 81
19-1 Transfer Functions for Pointers . . . . Page 82
19-2 Pointer Levels . . . . Page 82
20-0 Scan . . . . Page 83
20-1-1 Replace . . . . Page 84
20-1-2 Pictures . . . . Page 86
20-2 Boolean Expressions . . . . Page 87
21-0 Time Functions . . . . Page 89

APPENDIX

Reserved Words . . . . Page 90
EBCDIC Codes and BCL Codes . . . . Page 91
Hexadecimal Codes . . . . Page 92
Octal Codes . . . . Page 92
Answers to Selected Exercises . . . . Page 92
Simplified Input-Output . . . . Page 99

0-0 INTRODUCTION

A digital computer is an electronic device capable of manipulating information at an extremely rapid rate. A program is the means whereby information is given to the machine - along with a set of instructions telling it how the information is to be processed.

ALGOL is a kind of language used for writing computer programs. Burroughs ALGOL is an expanded version of ALGOL 60. The term ALGOL comes from the two words ALGOrithmic Language. ALGOL 60 was formulated in Europe in 1958 as an international programming language.

0-1 USING THE COMPUTER WITH ALGOL

A person wishing to use the computer to solve a problem proceeds in two stages:

First, he writes a computer program which will cause the machine to produce the solution to the problem. Second, he "gives" the program to the computer.

The first stage may be accomplished with paper and pencil, but computers cannot read hand-writing, not yet anyway, so the second stage consists of putting the hand-written program into a form acceptable to the machine. Let us go through an example to see how this is done. Suppose the problem is to compute 355 divided by 113. A program which will do this and print the answer on paper is as follows:

```
BEGIN
FILE PRINTER (KIND=PRINTER, MAXRECSIZE=22);
REAL X;
X := 355/113;
WRITE(PRINTER,/,X)
END.
```

Do not worry yet about why or how this program does what we claim, it does. That will become clear as we explain how to write programs in ALGOL. For now we are only concerned with the question "Now that I have written it, what do I do with it."

Programs are submitted to the machine in the form of punched cards, and results are returned to the programmer by means of a printer. Punched cards are produced on keypunch machines. Figure 1 shows a sample card with all the key-board characters punched. Note that each character has its own pattern of holes punched in the column beneath it. When punching a program we generally use one card for each line in the program. If a line happens to be too long to fit into the first 72 columns of a card, however, then it must be continued on a second card. The continuation process may proceed onto a third card, etc., but in every case punching must stop on or before column 72.

In addition to the cards punched for the program, six others known as control cards are necessary. Five go in front of the program and one in back as shown in Figure 2.* This, then, is the complete deck. It may be submitted to the machine by placing it in the card reader and pressing the reset and start buttons.

The computer then begins processing the program. After the cards are read by the card-reader, the programmer may remove them. When the machine has finished processing the program, it prints a copy of the deck along with any errors that may have been made. If there are no errors, then the program itself is executed. In this instance, the result of dividing 355 by 113 is printed on the printer.

EXERCISES

1 - Punch a card like that shown in Figure 1.

2 - Punch the program shown in Figure 2 and run it.

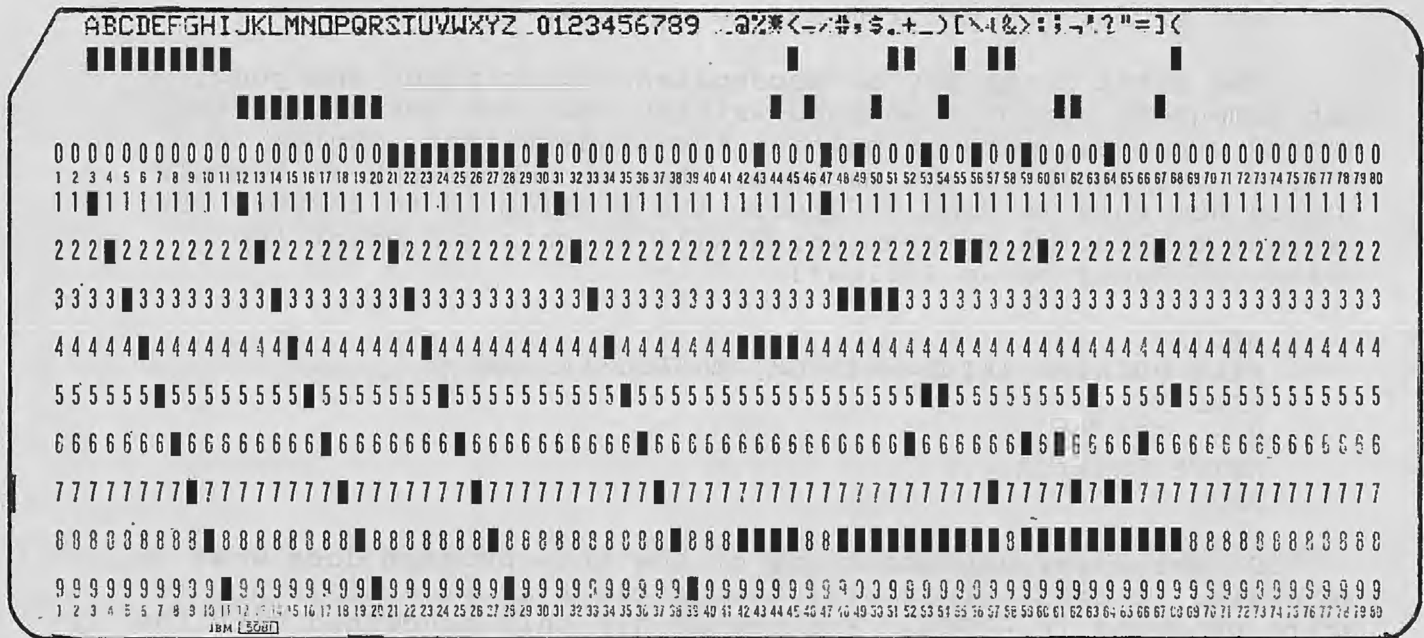

Figure 1

---

* The NAME1/NAME2 shown on the fourth card can be any pair of names. The other cards should be exactly as shown. The symbol **2** is a 1-2 multiple punch formed by simultaneously depressing the multiple punch and numeric keys on the left side of the keyboard and then punching the 1 and 2. The first card need not be punched; it is available at the Computer Center.

```
?END JOB
END.
WRITE (PRINTER,/,X);
X:=355/113;
REAL X;
FILE PRINTER(KIND=PRINTER,MAXRECSIZE=22);
BEGIN
?DATA
?COMPILE NAME1/NAME2 ALGOL GO
XXXXXXXXXXXX                              YOUR PASSWORD IN COL. 1 - 12
XX321  SMYTHE                             YOUR ACCOUNT AND NAME
?BEGIN JØB Q 2
```

Q2

NAME SMYTHE	
ACCT # XX321	PHONE EXT

☐ SEE OPERATING INSTRUCTION CARD

FIGURE 2

1-0 ALGOL PROGRAMMING - THE FORMAL ASPECTS

We are now ready to address ourselves to stage one of problem solving with a digital computer - writing programs.

As in any other language only a very special set of symbols is used. For example "H⊗ S # ←OY." is not a sentence in the English language because the symbols ⊗, #, and ← are not used in the English language. Similarly, the only allowable symbols used in ALGOL are:

the letters:	ABCDEFGHIJKLMNOPQRSTUVWXYZ
the digits:	0123456789
the blank:	
the special characters:	+ - / = [ ] ( ) < > " . , : ; @ % $ & # *

All of these characters are available on the keypunch.

1-1 REPRESENTATION OF NUMBERS

Numbers may be represented in ALGOL in several different ways, and two types of numbers are recognized, integer and real. An integer is written simply as consecutive digits. A + or - sign may precede the number if desired. The largest absolute value allowable is 549755813887 because of space limitations within the machine. Notice that no decimal point is allowed for integers. Some examples of integers are:

1
547
-10763
+999

A real number is allowed to contain a decimal point. Thus a decimal fraction is a real number. Real numbers also consist of a group of consecutive digits possibly with a decimal point. A + or - sign may precede the number. Notice that an integer is also a real but not all reals are integers. Some examples of real numbers are:

12.7
.333
-.00031
-547.
1066

Except for leading or trailing zeroes, a real number should not contain more than 12 digits. The reason for this is that space limitations within the machine determine the accuracy allowed. The fraction 1/3 cannot be represented exactly in ALGOL; 1/3 is not a number because the definition of real numbers does not provide for the /. However, .3333 and .3333333 are legitimate real numbers, and both are approximation to 1/3. The more threes we put after the decimal point, the better the approximation, but it does no good to use more than 12 threes because the machine cannot tell the difference between .3333333333330 and .3333333333339 anyway.

Scientific Notation

When the results of scientific investigations are published, measurements and other data are often written as a decimal fraction between 1 and 10, times a power of 10. Thus,

1000. is written as 1.0×10^{3}

-576.12 is written as -5.7612×10^{2}

.0003 is written as $3. \times 10^{-4}$

ALGOL provides for representation of numbers in this fashion by means of the symbol @. The usage is better illustrated than explained.

Scientific Notation	ALGOL Representation
1.0×10^{3}	1.0@+3
3.0×10^{-4}	3.0@-4
-576.12×10^{2}	-576.12@2
-3.14×10^{-10}	-3.14@-10

Plus signs are optional. Because of space limitations within the machine, the largest positive number that can be represented in this form is 4.314@+68 and the smallest positive number is 8.758@-47. As above, the decimal part should contain no more than 12 digits.

EXERCISES

1 - Which of the following are numbers in ALGOL?

A) 10,000,000
B) 23567891232
C) +-2.24
D) 757575757575
E) 2.253 21
F) +2.251
G) 3.2(5.1+2.)

1-2 IDENTIFIERS

In the English language, certain groups of letters are recognized as properly spelled words while other groups carry no meaning. Just as words may represent ideas in English, identifiers are used to designate a great many of the constructs in ALGOL. The rules for "spelling" or forming identifiers are as follows:

1 - The first character must be a letter.
2 - Other characters must be letters or digits.
3 - An identifier may be from 1 to 63 characters long.
4 - A reserved word is not an identifier.

The Appendix contains a complete list of reserved words. They will be introduced gradually in the following sections and their uses will be explained. For the time being suffice it to say that they are all recognizable English words. Thus, we can be sure that XYZ does not violate rule 4 and so is a legitimate identifier.

Examples of Identifiers:

J
Q
Q1Q
L5725

1-3 SIMPLE VARIABLES

Suppose an accountant were presented two columns of numbers and told to divide the sum of one column by the sum of the other. Chances are he would need to write down the sum of the first column to help him remember it while he was computing the sum of the other. The scratch pad he uses is a memory device.

Part of the power of the digital computer lies in its ability to "remember" or store the results of previous computations while it is performing others. The computer's memory consists of many thousands of "cells" each of which is capable of storing a single number.

A simple variable is an identifier used to specify one of these memory cells or locations. One might think of the memory as a box of tabbed index cards. For example, to say that the simple variable X1 has the value 10.3, we might say that the box of cards contains one card with the identifier X1 written on the tab and the number 10.3 written on the card itself.

EXERCISES

1 - Which of these are numbers in ALGOL?

A) 4.32@309
B) @68
C) 1342
D) 134.2
E) 134.2E-2

2 - Which of the following are identifiers. There are no reserved words in the list.

A) THISISANIDENTIFIEREVENTHOUGHITISKINDOFLONG
B) H
C) 5IVE
D) FATHER-CHRISTMAS
E) L123456789
F) ENDIT
G) FATHERCHRISTMAS

2-0 REPLACEMENT STATEMENTS AND SIMPLE ARITHMETIC EXPRESSIONS

One of the simplest and most elementary instructions that might be contained in a program is the simple replacement statement. An example of such a statement is

```
X:=1;
```

The leftmost part is an identifier. In this case the simple variable X, then the symbols := then a number, and finally a semi-colon. The example is interpreted by the machine to mean "assign the value 1 to the simple variable X." In place of a number, we may have a simple arithmetic expression between the := and the semi-colon. In this case, the machine is instructed to assign the value of the arithmetic expression to the simple variable on the left. For example, the results of executing

W2:=5+3-2; and

W2 := 6; are the same.

Simple variables may also appear within arithmetic expressions. Thus the following has the same result as the above:

```
X := 5;
Y := 3;
Z := -2;
W2 := X + Y + Z;
```

The symbol for multiplication is *
The symbol for division is /
The symbol for exponentiation is **

Thus A*B means A times B
A/B means A divided by B
A**B means raise A to the power B

When writing on paper, particularly when several divisions are involved, arithmetic expressions are usually written on several lines. However, since computer programs are ultimately punched on cards, arithmetic expressions must be written on one line. This is accomplished by means of parentheses.

The expression $\frac{A + B - C}{A - B + C}$ is written (A+B-C)/(A-B+C).

Parentheses are also required to separate two operators that would otherwise fall side by side. The first of the following two expressions is incorrect; the second is the proper method.

```
Y := X*-2;

Y := X*(-2);
```

The complexity which simple arithmetic expressions can attain is practically limitless. The rules governing arithmetic expressions are as follows:

1 - Sub-expressions inside parentheses are evaluated first.

2 - Unless parentheses over-ride, exponentiations (**) are done first then multiplications (*) and divisions (/), last additions (+) and subtractions (-)

3 - Successive multiplications and divisions are performed left to right.
Successive exponentiations are performed left to right.
Successive additions and subtractions are performed left to right.

EXERCISES

1 - Let X = 1.0, Y = 2.0, Z = 3.0

Determine the value of each expression.

```
(A)   (X+(Y*Y)/Z+Z)/Y
(B)   X+Y+Z*X/Y/Z
(C)   X+Y*Z*Y+X**Z/Y**Z+X
(D)   Z**Z**Z
(E)   Z**(Z**Z)
```

2 - Find an error in each of the following expressions.

```
(A)   3X + 4Y + Z
(B)   A(X + 5)
(C)   A + X*(B + X*(C + X*(D + X*E))))
(D)   P*[X + Y + Z]
(E)   X + Y*-X + Z**2
```

3 - Verify that C+X*(B+X*(A+X)) is equivalent to X**3+A*X**2+B*X+C

2-1 SIMPLE PRINTER OUTPUT WITH B6700 ALGOL

The statement

```
WRITE (PRINTER,/,X);
```

instructs the computer to print the value of the simple variable X on the line printer. The word WRITE is a reserved word. Each WRITE statement encountered in a program produces a new line on the printer. If it is desired to write the values of several variables on one line, we simply list all of them after the slash. For example

```
WRITE (PRINTER,/,A,B,C,D);
```

Each variable to be printed is preceded by a comma, and the entire statement is terminated with a semi-colon.

EXAMPLE

Let X = 1, Y = 2, Z = 3.3

The statement

```
WRITE (PRINTER,/,X,Y,Z);
```

produces

```
1, 2, 3.300000000002E00,
```

Numbers other than integers are written in scientific notation. The reason that fractions may not reproduce exactly will become clear later.

2-2 DECLARATIONS

Recall the analogy of the computer memory to a box of tabbed cards. At the very outset of a program, the statement

```
X:=1;
```

cannot be executed. This is because the statement means "find a card with the identifier X on the tab, and write the number 1 on the card." But in the beginning all of the tabs are blank.

Declarations are "housekeeping" instructions used to reserve memory space for use later in the program. There are many kinds of declarations, and they will be introduced gradually in the sections that follow. For now we discuss only three. The declaration

```
REAL X;
```

tells the machine "pick a card with a blank tab and write the identifier X on the tab. Allow only real numbers to be written on this card, but do not write anything yet." In other words, reserve space for the real variable X. The word REAL is a reserved word. A space must separate the word real from the identifier X to distinguish from the identifier REALX. The declaration is terminated by a semi-colon. Several variables may be declared at once by listing them separated by commas as follows:

```
REAL X,Q1Q,ALFFA;
```

The declaration

```
INTEGER I;
```

serves the same purpose as the REAL declaration except that it specifies that only integer values may be written into the memory space designated by the identifier I. In other words the simple variable I may assume only integer values. If we were to attempt to assign a real number by means of a replacement statement, the value would be rounded to the nearest whole number.

The statement I:=2.3; assigns I the value 2.

The statement I:=2.7; assigns I the value 3.

The word INTEGER is a reserved word. It must be followed by a space in order to distinguish from the identifier INTEGERI.

The declaration

```
FILE PRINTER(KIND=PRINTER, MAXRECSIZE=22);
```

associates the identifier PRINTER with a high speed line printer. It is necessary in order to give meaning to ensuing write statements just as the identifier X must be declared before it can be used for computation purposes.

The word FILE is a reserved word. Actually, any identifier can be used instead of the word PRINTER so long as the same identifier is used in the WRITE statements.

Notice throughout this section that a distinction is made between the term declaration and the term statement. Strictly speaking, a declaration is not a statement.

2-3 BASIC STRUCTURE OF ALGOL PROGRAMS

We have now reached the point at which the presentation of a few simple rules governing the structure of every program will make the sample program in the introduction clear. These rules are as follows:

1 - Every program begins with the reserved word BEGIN.

2 - Every program ends with the reserved word END followed immediately by a period.

3 - Every identifier must be declared and, with one exception to be covered later, all declarations must precede any statements.

A detailed explanation of the sample program follows:

```
BEGIN                                 required by rule 1
FILE PRINTER (KIND=PRINTER,           reserve a line printer
                    MAXRECSIZE=22);   reserve space for the
REAL X;                                 variable X
X:=355/113;                           Set X to the (approximate)
                                        value of 355/113
WRITE (PRINTER,/,X);                  print the value of X
END.                                  required by rule 2
```

In keeping with rule 3, the declarations precede the statements.

EXAMPLES

```
(A)  REAL, X,Y,Z;              no comma should appear after REAL
(B)  INTEGER I;J;              should be a comma between I and J.
(C)  REALX,Y,Z;                space needed between REAL and X
(D)  REAL INTEGER;             reserved word used as identifier
(E)  INTEGER LAP,DEG           missing semicolon
(F)  INTEGER I,J,K,L,M,N;      correct
(G)  REAL X,Y,Z;               correct

INTEGER I,J;                   no begin
REAL X,Y                       no semi-colon
X := 1,000;                    comma should not be present
Y = 49.7;                      should be :=
J := Y;                        OK value will be 50
K := 1;                        K was not declared
I := (X + Y)*J/X;              OK
END                            no period
```

2-4 THE IF STATEMENT - SIMPLE BOOLEAN EXPRESSIONS

Another aspect of the power of the digital computer is the ability to make simple comparisons and take either of two courses of action depending on the outcome. This facility is implemented in ALGOL by means of the IF statement. An example of an IF statement is

```
IF X LEQ Y THEN Z:=X ELSE Z:=Y;
```

This statement tells the machine "if the value of the variable X is less than or equal to the value of the variable Y, then set Z equal to X. Otherwise, set Z equal to Y." The words IF, THEN, and ELSE are reserved words. They must be preceded and followed by spaces so that they will not be mistaken for parts of identifiers.

The expression between the words IF and THEN is known as a simple Boolean expression. We cannot assign a numerical value to this expression as we can to an arithmetic expression, but we can tell whether the expression is true or false. One of the six relational operators usually appears within a Boolean expression. The mnemonics for the relational operators and their meanings are

LSS	less than
LEQ	less than or equal
EQL	equal
NEQ	not equal
GEQ	greater than or equal
GTR	greater than

Any statement may appear between the words THEN and ELSE. Notice that there is no semicolon preceding the word ELSE. Any statement may also appear between the word ELSE and the semi-colon. If the Boolean expression is true, then the first of these two statements is executed and the second is ignored. If the Boolean expression is false, then the first statement is ignored and the second is executed. A simple variation of the IF statement is as follows:

```
IF ALFFA NEQ O THEN B := 1/ALFFA;
```

In this case the statement B :=1/ALFFA; is executed if the Boolean expression ALFFA NEQ O is true. Otherwise it is ignored.

EXERCISES

1- write a statement that prints the maximum of X and Y.

2-5 LABELS AND THE GO TO STATEMENT

Ordinarily the successive statements in a program are executed in sequential order from the beginning. However, if the programmer would like to skip directly from one part of his program to another, he may do so by means of the GO TO statement and the label. Labels are identifiers and must therefore appear in a declaration at the beginning of the program.

```
LABEL L1,L2;
```

The word LABEL is a reserved word and so must be followed by a blank. Any number of identifiers may be designated as labels in one label declaration by separating them with commas. The declaration is terminated by a semi-colon. A statement within a program is labeled by placing a label followed by a colon in front of it.

```
L1:  IF X LSS O THEN WRITE (PRINTER,/,Y);
```

The GO TO Statement has the following forms:

```
GO TO L2;
GO L2;
```

The words GO and TO are reserved words. Notice that the word TO is optional in the GO TO statement. The meaning of the statement

```
GO TO L;
```

is "execute the statement labeled L and proceed from there."

The most devastating pitfall awaiting computer programmers is a situation known as a closed loop or infinite loop. It is often caused by improper use of the GO TO statement, and it can prove to be very costly. The following program is an example:

```
BEGIN
LABEL AGAIN;
REAL X;
AGAIN:X:=355/113;
GO TO AGAIN;
END.
```

This program will compute the value of 355/113 over and over again until the machine is stopped by some action external to the program.

Properly used, the process of repeating a group of instructions several times can be extremely useful. In order to avoid closed loops, we generally incorporate an IF statement with the GO TO statement. The following program generates a table of squares:

```
BEGIN
FILE PRINTER (KIND=PRINTER, MAXRECSIZE=22);
INTEGER N,NSQ;
LABEL MORE;
N:=1;
MORE: NSQ:=N*N;
WRITE (PRINTER,/,N,NSQ);
N:=N+1;
IF N LEQ 100 THEN GO TO MORE;
END.
```

The IF statement provides a means of escaping or exiting from the loop.

EXAMPLE

1 - Make a table of 2^N for N running from 0 to 38.

```
BEGIN
FILE PRINTER (KIND=PRINTER, MAXRECSIZE=22);
LABEL AGAIN;
INTEGER N,TWON;
N := -1;
AGAIN: N := N + 1;
TWON := 2**N;
WRITE (PRINTER,/,N,TWON);
IF N LSS 38 THEN GO TO AGAIN;
END.
```

EXERCISES

1 - Write a program to find the sum of the first 1000 integers.

2 - Write a program to compute 2*3*...*N (I.E. N factorial). Assume N is somehow previously given.

3 - Write a program to compute the binomial coefficient C(N,K). C(N,K) = (N factorial)/((K factorial)*((N-K) factorial)) Assume N and K are somehow previously given.

4 - Write a program to generate the first term of the Fibonacci sequence that is bigger than 1000000000.

5 - Make example 1 more efficient by using the fact that 2**N = 2 * 2**(N-1).

2-6 THE FOR STATEMENT

The FOR statement provides for automatic loop control. It has the following form:

```
FOR I:=3 STEP 1 UNTIL 10 DO
WRITE (PRINTER,/,I);
```

The meaning of this statement is "set I equal to 3 and print I. Then set I equal to I+1 and print I. Keep doing this until I has the value 10." The numbers 3 4 5 6 7 8 9 10 will be printed.

The words FOR, STEP, UNTIL, and DO are reserved words. Any arithmetic expression may be used in place of the numbers 3,1, and 10. Any statement (including another FOR statement) may appear in place of the WRITE statement.

EXAMPLE

1 - Compute the sum of the first 100 positive integers.

```
BEGIN
FILE PRINTER (KIND= PRINTER, MAXRECSIZE=22);
INTEGER N,SUM;
SUM := 0;
FOR N := 1 STEP 1 UNTIL 100 DO
SUM := SUM + N;
WRITE (PRINTER,/,SUM);
END.
```

2-7 STANDARD FUNCTIONS

B6700 ALGOL includes the eight standard arithmetic functions provided by ALGOL 60. They are:

ABS(X)	Absolute value of X
SIGN(X)	+1 if X is positive, -1 if X is neg. 0 if X is 0
SQRT(X)	Square root of X
SIN(X)	Sine of X
COS(X)	Cosine of X
ARCTAN(X)	Principle value of arctangent of X
LN(X)	Natural logarithm
EXP(X)	Exponential function

The arguments of these functions may be arithmetic expressions instead of simple variables as shown. The argument for SIN and COS and the result of ARCTAN are in radians. The words ABS, SIGN, SQRT, SIN, COS, ARCTAN, LN, and EXP are not reserved words in the sense that they may be used as identifiers and declared accordingly. They are reserved words in the sense that they have special meaning when used as arithmetic functions and in that case they should not appear in declarations.

EXAMPLES

1 - The following are valid statements:

```
IF SIN(X) LSS 0 THEN GO TO L1;
Y := SQRT(X*Z + W);
Z := ARCTAN(SIN(X)) + SQRT(LN(Y) + X*EXP(Z));
```

EXERCISES

1 - Write a program to print a table of square roots.

2 - Write a program to make a table of sines and cosines.

3 - Make a table of LN(EXP(X)).

4 - Make a table of C(X) = SIN(X)**2 + COS(X)** 2.

2-8 OPERATIONS ON INTEGERS

ALGOL contains an arithmetic function and two additional arithmetic operations designed especially for operations with integers.

ENTIER(X)	The largest integer less than or equal to X. For example, ENTIER(2.7) is 2 and ENTIER(-2.7) is -3.
X DIV Y	The integer part of the quotient X/Y. Thus 4 div 3 is 1.
X MOD Y	The remainder obtained when X is divided by Y. Thus 7 mod 4 is 3.

The word ENTIER is treated like the other function designators. The words DIV and MOD are reserved words and must be preceded and followed by spaces.

EXERCISES

1 - Given an amount between 1 and 99 write a program that will tell how many quarters, dimes, nickels, and pennies are needed to equal that amount. Put a loop in the program to do this for all amounts between 1 and 99 that are multiples of 7.

2 - Find all integer solutions to the system

$$X \geq 0$$
$$Y \geq 0$$
$$X - 2Y = 0$$
$$5X + 2Y \leq 1000$$

2- 9 ARRAYS AND SUBSCRIPTED VARIABLES

It is often necessary to store large quantities of information in memory. If a different identifier were required for each cell, the program would become unacceptably cumbersome. The use of arrays allows us to assign a single identifier to an entire group of memory cells. The mechanism assigns a unique integer to each memory cell in the group in order to tell one from the other. The declaration

```
ARRAY A[0:24];
```

means, "find 25 storage cells and assign the identifier A to all of them. Within the group, assign the number 0 to one of them. The number 1 to the next, etc. up to 24." The word ARRAY is a reserved word. Any integer, positive or negative, may be used in place of the numbers 0 and 24 shown, but the second must be larger than the first. The members of the ARRAY will be assumed to be real numbers. An ARRAY of integers is declared by

```
INTEGER ARRAY NUMS[1:100];
```

Single members of an array are referred to by means of the array identifier followed by an integer or arithmetic expression enclosed in square brackets:

```
NUMS[57]:=ENTIER(A[I]);
```

NUMS[57] and A[I] are examples of subscripted variables.

EXAMPLE

1 - Write a program to efficiently generate the first 1000 primes by using an array of length 1000. Method: N is prime if and only if it is not divisible by any of the primes ≤ SQRT(N).

```
BEGIN
FILE PRINTER (KIND=PRINTER, MAXRECSIZE=22);
INTEGER ARRAY PRIMES [0:999];
INTEGER N,I,ILAST;
LABEL NEXT, ITSPRIME;
PRIMES[0] :=2;
ILAST := 0;
N := 1;
NEXT: N := N + 2;
FOR I := 1 STEP 1 UNTIL ILAST DO
IF PRIMES[I] GTR SQRT(N) THEN GO TO ITSPRIME ELSE
IF N MOD PRIMES[I] = 0 THEN GO TO NEXT;
ITSPRIME: ILAST := ILAST + 1;
PRIMES[ILAST] := N;
IF ILAST LSS 999 THEN GO TO NEXT;
FOR I := 0 STEP 1 UNTIL 999 DO WRITE (PRINTER,/,
     PRIMES[I] );
END.
```

The arrays discussed above were one dimensional arrays because only one subscript was required to specify a single element. Multiple dimensioned arrays are also possible. The declaration

```
INTEGER ARRAY DECK[1:4,1:13];
```

assigns the array identifier **DECK** to a group of 52 memory locations. These cells may be thought of as being arranged in a rectangular pattern of 4 rows with each row consisting of 13 elements. An element of this array is specified by a pair of integers in brackets:

```
IF DECK[1,10] LSS DECK[2,4] THEN GO TO YOULOSE;
```

arrays are most useful in conjunction with FOR statements. For example, a simple way to search a large section of memory is as follows:

```
FOR I:=0 STEP 1 UNTIL 10 DO
FOR J:=0 STEP 1 UNTIL 1000 DO
IF A[I,J] LSS 1 THEN WRITE(PRINTER,/,A[I,J]);
```

These three lines of programming instruct the machine to search an array of over 11000 cells and list those which are less than 1.

EXERCISES

1 - Write a group of statements that will place an array of 1000 numbers in decreasing order.

2 - Make a table of 2^N for N running from 0 to 38 using an array of lengh 39.

Compare this program with those of example 2-6-1 and exercise 2-6-5.

2-10 INPUT AND MORE ON OUTPUT

So far the only means we have of assigning values to variables is a replacement statement such as

```
X:=5.32;
```

Another method essential to the writing of programs with general applicability is reading data from an external input device. The only medium we shall consider at present is the card reader. Input from the card reader is achieved by placing a special control card after the program end followed by cards containing the data to be read. The actual input is accomplished by the statement

```
READ (READER,/,X);
```

A FILE declaration is also necessary in order to reserve a card reader. It has the form

```
FILE READER(KIND=READER);
```

The file declaration is not needed if simplified input-output as described in the appendix is used.

Figure 3 shows a deck properly set up to read three numbers and set them into the locations designated by the variables X,Y,Z[3]. As in the write statement discussed earlier, several variables may be read at once. Take special note of the additional control card following the program. Each number to be read, including the last, must be followed by a comma.

Just as each print statement encountered in a program produces a new line of print, each read statement encountered begins reading a new card. Figure 4 shows the necessary deck structure for reading an array of three numbers. Since this arrangement would involve a great many data cards in order to read a large array, a more convenient form for the read statement is provided by allowing a FOR statement inside the list of variables to be read. Figure 5 shows how this is done. The result of executing this program is identical to the result of executing the program of Figure 4. Similarly, FOR statements may appear inside the list of variables in a print statement. The instruction

```
WRITE (PRINTER,/, FOR J := 1 STEP 1 UNTIL 100 DO B[J]);
```

will print the values of B[1],B[2], through B[100]. The numbers will appear strung across the page in as many lines necessary to print the 100 numbers. The READ statement will also read additional cards if all the data asked for does not appear on one card.

EXERCISES

1 - Write a program to read 100 numbers into an array and count how many are positive, how many negative, and how many zero.

```
2COMPILE EXAMPLE/READ ALGOL
2DATA
  BEGIN
  FILE READER(KIND=READER);
  REAL X,Y;
  ARRAY Z [0:9];
  READ(READER,/,X,Y,Z[3]);
  END.
2DATA
1.25, 3.47, 92.68,
2END
```

FIGURE 3

```
2COMPILE EXAMPLE2/READ ALGOL
2DATA
  BEGIN
  FILE READER(KIND=READER);
  INTEGER I;
  ARRAY Z[0:2];
  FOR I:=0 STEP 1 UNTIL 2 DO READ
      (READER,/,Z[I]);
  END.
2DATA
2.315,
3.71828,
.5772,
2END
```

FIGURE 4

```
2COMPILE EXAMPLE3/READ ALGOL
2DATA
   BEGIN
   FILE READER(KIND=READER);
   INTEGER I;
   ARRAY Z[0:2];
   READ (READER,/, FOR I:=0 STEP 1 UNTIL 2
      DO Z[I]);
   END.
2DATA
2.315, 3.71828, .5772,
2END
```

FIGURE 5

To print more than one value with each iteration of the FOR loop, brackets must be used. For example:

```
WRITE (PRINTER, /, FOR I := 0 STEP 1 UNTIL N DO [A[I], B[I]]);
```

Sometimes it is desirable to use a portion of the data card for comments (or just to stop the program from looking past a certain point on the card.) This can be done by placing an asterisk after the comma following the last valid number on the card. For example:

```
2.3456, 1.678E-9,*    THIS IS A COMMENT
```

2-11 COMPOUND STATEMENTS

Recall that the FOR statement allows the programmer to repeat a single statement, and the IF statement allows him to ignore a single statement. Sometimes it is convenient to apply these facilities to a group of statements taken as a whole. This construct is realized in the compound statement. A compound statement is simply a group of statements preceded by the reserved word BEGIN and followed by the reserved word END. . Compound statements are operationally the same as single statements and may be used anywhere in an ALGOL program where single statement may. Note that declarations do not appear in compound statements - only statements.

EXAMPLE

1 - Write a program to find the greatest common divisor of two numbers.

An algorithm for finding the G.C.D. of two integers M and N is: First interchange them if necessary so that $M \geq N$. (1) divide M by N to give quotient Q and remainder R. If $R \neq 0$ assign new values to M and N such that M:=N and N:=R. Repeat (1). When R:=0 the value of N is the G.C.D.

```
BEGIN
FILE READER(KIND =PRINTER);
FILE PRINTER(KIND=PRINTER, MAXRECSIZE=22);
INTEGER M,N,TEMP,R;
LABEL DONE, AGAIN;
READ (READER,/,M,N);
IF M LSS N THEN
    BEGIN
    TEMP:=M;
    M:=N;
    N:=TEMP;
    END;
WRITE (PRINTER,/,M,N);
AGAIN:R:=M MOD N;
IF R=0 THEN GO TO DONE
ELSE BEGIN
    M:=N;N:=R;
    GO TO AGAIN
    END;
DONE:
WRITE (PRINTER,/,N);
END.
```

2-12 BLOCKS

A block is a BEGIN followed by at least one declaration, then at least one statement and terminated with an END. Just like compound statements, a block is operationally the same as a single statement and may appear anywhere in a program where a single statement may appear. Notice that this allows declarations to appear in positions other than at the beginning of a program. It is the exception to rule 3 of section 2-3 that was noted then but not explained. We now amend rule 3 to read:

3 - declarations may appear only at the beginning of a block.

Although the only difference programatically between a compound statement and a block is the presence of declarations, the two are treated quite differently by the machine. While a compound statement is handled exactly like a simple statement, a block behaves more like a sub-program complete unto itself within the main program.

There are two advantages to this treatment of blocks. The first is that large, complicated programs may be broken down into blocks and in that way written a piece at a time. It is even possible to have many programmers working on the same program by assigning a different block to each one. The second advantage lies in the way ALGOL handles declarations. Any memory space reserved by a declaration at the top of a block is released upon reaching the end of that block. Going back again to the notion of memory as tabbed index cards, we might say that the identifiers written on tabs when the declarations were executed are erased and made available for use by other declarations to be executed later. The effect of this is that even though a program may define a great many variables in all, at any point during execution, space is reserved only for those variables currently in use. This facility is particularly nice in view of the current trend in the computing industry to charge a program not only for time used but for amount of memory needed as well.

EXERCISE

1 - Fill a 10 by 10 array in the following symmetric way:

```
1  1  1  1  1  1  1  1  1  1
1  2  2  2  2  2  2  2  2  1
1  2  3  3  3  3  3  3  2  1
1  2  3  4  4  4  4  3  2  1
1  2  3  4  5  5  4  3  2  1
1  2  3  4  5  5  4  3  2  1
1  2  3  4  4  4  4  3  2  1
1  2  3  3  3  3  3  3  2  1
1  2  2  2  2  2  2  2  2  1
1  1  1  1  1  1  1  1  1  1
```

2-13-1 REAL PROCEDURES

Although a variety of commonly used standard arithmetic functions such as SQRT is provided, the need for others often arises. Up to now the programmer has two alternatives. He can insert the necessary code every place he needs to compute the function, or he can write the code in a separate block and jump to that block with a GO TO statement at every place the function is needed. These alternatives are illustrated in programs 1 and 2 of figure 6. The trouble with the first method is that nearly identical code must appear in different places in the program. The trouble with the second approach is that we must keep track of where we came from in order to get back to the right place when we leave the inner block.

The real procedure is a means of implementing the second approach to the above problem while automatically keeping track of to what point in the main program to return. In effect, it enables us to define our own "standard functions." Once they have been defined, they are used just as if they were standard functions. Program 3 of figure 6 shows the same problem solved by means of the real procedure A.

Real procedures are defined by means of procedure declarations as shown in figure 6. They consist of two parts, the procedure heading and the procedure body. The procedure heading defines the name of the procedure and specifies the quantity and type of the arguments. The word PROCEDURE is a reserved word. Note the semicolon after the argument list. The next line which looks like a declaration specifies the type of the argument. It is called the specifications part of the procedure heading, and it is not really the same as a declaration because it does not cause memory space to be reserved. It does not need to reserve space because it only refers to a dummy variable that will be filled in with declared variables (for which space has already been reserved) when the procedure is actually executed in the main program. Notice in figure 7 that only the lower bounds for arrays are specified. Specifications for integer and real look exactly like declarations.

The procedure body must be a single statement in the broad sense. That is, it can be a simple statement, a compound statement or a block. It defines what computations should be performed when the procedure is called. A replacement statement with the name of the procedure on the left side must be executed somewhere within the procedure body.

```
BEGIN
FILE READER(KIND=READER);
FILE PRINTER (KIND= PRINTER, MAXRECSIZE=22);
REAL X,Y,Z,W;
READ (READER, /, X,Y,Z);
W := X*X+2*X-5  +  (Y*Y+2*Y-5)*3  -  LN(Z*Z+2*Z-5)
WRITE (PRINTER,/,X,Y,Z,W);
END.
```

Program One

```
BEGIN
FILE READER (KIND=READER);
FILE PRINTER (KIND=PRINTER, MAXRECSIZE=22);
REAL X,Y,Z,W,S;
LABEL LX,LY,LZ,ENDIT,COMP;
INTEGER L;
READ (READER,/,X,Y,Z);
L :=1;S := X; GO TO COMP;
LX: W = S;
L := 2;  S := Y;  GO TO COMP;
LY: W := W + S*3;
L := 3; S := Z; GO TO COMP;
LZ: W = W - LN(S);
WRITE (PRINTER,/,X,Y,Z,W); GO TO ENDIT;
COMP: BEGIN
      S := S*S + 2*S -5;
      IF L = 1 THEN GO TO LX ELSE IF L = 2 THEN GO TO LY
          ELSE GO TO LZ;
      END;
ENDIT: END.
```

Program Two

```
BEGIN
FILE READER(KIND=READER);
FILE PRINTER (KIND=PRINTER, MAXRECSIZE=22);
REAL X,Y,Z,W;
REAL PROCEDURE A(X);
      REAL X;
      A := X*X + 2*X - 5;
READ (READER,/,X,Y,Z);
W := A(X) + A(Y)*3 - LN( A(Z) );
WRITE(PRINTER,/,X,Y,Z,W);
END.
```

Program Three

Figure 6

```
BEGIN
FILE PRINTER (KIND=PRINTER, MAXRECSIZE=22);
ARRAY A[0:10],B[0:10]; REAL X;
PROCEDURE MV( A,B ):
     ARRAY A[0],B[0];
     BEGIN INTEGER I;
     FOR I := 0 STEP 1 UNTIL 10 DO B[I] := A[I];
     END;
PROCEDURE PRHPSPRNT( U,R );
     REAL U;  ARRAY R[0];
     IF U*R[5] GTR 0 THEN BEGIN
         INTEGER I;
         WRITE (PRINTER,/, FOR I := 0 STEP 1 UNTIL 10
             DO R[I]
         END;
  .
  .
  .
MV(A,B);
PRHPSPRNT(X,A);
  .
  .
  .
END
```

Figure 7

2-13-2 UNTYPED PROCEDURES

The result of calling a real procedure is a single real number. Similarly the result of calling an integer procedure is a single number of type integer. Untyped procedures can be used when several quantities are to be produced, when an array is to be the result, or when the task involves no numerical result at all. Figure 7 shows selective portions of a program with two procedures, the first one MV moves the values of one array into another. The second procedure, PRHPSPRNT, possibly prints the array and possibly does nothing at all. Notice that the name of the procedure does not appear in the procedure body.

EXERCISES

1 - Write a real procedure Q(X) where
Q(X) = X - X**3/(2*3) + X**5/(2*3*4*5)- X**7/(2*3*4*5*6*7).

2 - Produce a table of X,Q(X),SIN(X) , and ABS (Q(X)-SIN(X)).

3 - Write a procedure to fill an N×N array by
H[I,J] = 1/(I + J - 1) .

2-14 COMMENTS

There usually are numerous ways to attack any given problem. Some are straight-forward, some clever, and some down-right devious. In order to let one programmer know what another one was up to, ALGOL provides a way of inserting explanatory remarks into a program. The COMMENT statement is used for this purpose:

```
COMMENT GOOD PROGRAMMERS USE MANY COMMENTS;
```

The word COMMENT is a reserved word, and the statement is terminated by a semi-colon. It is a do-nothing instruction, but since it is reproduced in the program listing prior to execution, it serves the purpose intended. Special care must be taken that the semi-colon at the end is present. Without it, the next statement would be considered part of the comment and ignored. This, in turn, would probably cause other errors.

3-0 INTRODUCTION

B6700 ALGOL is actually an extension of ALGOL 60. While Chapter One presented only the simplest constructs available, the purpose of this section is to explain those aspects of B6700 ALGOL which conform to the Revised Report on the Algorithmic Language ALGOL 60 but which have not yet been covered. ALGOL 60 differs from earlier programming languages in that it is constructed recursively. In part, this means that procedures may call themselves although many other examples of recursiveness will appear. Because of this, a recursive method of describing ALGOL 60, known as the metalanguage, was developed. Although the metalanguage itself is quite simple, the meaning of the formulas used to describe various constructs is often elusive, therefore we shall avoid it entirely or else use variations whose meanings will be clear from context.

Readers interested in seeing metalinguistic descriptions of ALGOL 60 may find the revised report in the Computer Journal Vol, 5, 1963, Page 349, or in Communications of the ACM, Vol. 6 January, 1963, Page 1. The next section explains the metalanguage, but it is not essential to the rest of this text and may therefore be bypassed if desired.

EXAMPLE

The following procedure is an example of recursive calls.

```
INTEGER PROCEDURE FACTORIAL (N);
INTEGER N;
IF N EQL 0 THEN FACTORIAL:=1 ELSE FACTORIAL:=N*
  FACTORIAL (N-1);
```

3-1 THE METALANGUAGE

Consisting entirely of only five rules, the metalanguage itself is very easy to describe.

1) Variables are enclosed in broken brackets.
   Ex: <Identifier>
   <Digit>
   <Unconditional Statement>

2) The symbol ::= is used to separate a variable on the left from the definition of the variable on the right.
   Ex: <Zero> ::=0

3) The symbol / is used to separate alternative definitions of the same variable. A vertical bar is also used in the literature.
   Ex: <Digit> ::= <Zero> /1/2/3/4/5/6/7/8/9

4) English phrases whose meanings are to be taken literally are denoted as <phrase>. Braces are commonly used instead.

5) Symbols not contained in brackets represent themselves.
Ex: <List> ::= <List Element> / <List> , <List Element>

In the example of Rule 5, a <List> is an object that contains a finite number of <List Elements> s separated by commas. The comma in the definition denotes a comma. Note, too, that the thing being defined appears in one of the alternative definitions. If this seems circular, note that the variable being defined does not occur in all the alternative definitions. This "escape mechanism" is characteristic of recursion in general.

EXAMPLE

<Identifier> ::= <Letter> / <Identifier><Letter> / <Identifier><Digit>

Compare this definition with that given in section 1-2 of Chapter One.

4-0 BOOLEAN EXPRESSIONS AND BOOLEAN ALGEBRA

Recall the general form of the IF statement as IF <Boolean Expression> THEN <Statement> ELSE <Statement> ; a Boolean expression was defined simply as an expression to which the values TRUE or FALSE could be assigned. We shall now examine Boolean Expressions in detail.

4-1 BOOLEAN VARIABLES

Besides REAL and INTEGER, a variable may be declared to be BOOLEAN and assigned the value TRUE or FALSE. The following are all valid statements.

```
BOOLEAN B1, B2, B3;
BOOLEAN ARRAY B [1:10, 1:10] ;
B1:=TRUE;
B2:=FALSE;
B3:=X EQL Y;
IF B3 THEN WRITE (PRINTER, /,X) ELSE WRITE
(PRINTER,/, Y);
```

The words BOOLEAN, TRUE and FALSE are reserved words. Notice that B3 assumes the value TRUE if X equals Y and the value FALSE otherwise. This is an example of a Boolean assignment statement. It is entirely analogous to an arithmetic assignment statement.

4-2 BOOLEAN ALGEBRA

Just as it is possible to compute with real or integer variables using arithmetic operators such as + or *, it is also possible to compute with Boolean variables. The Boolean operators are

NOT
AND
OR
IMP
EQV

They are all reserved words. The following table describes their meaning. Note that NOT operates on a single value while the others operate on a pair of values. IMP stands for implication, and EQV stands for equivalence.

The Logical Operators

B1 B2	TRUE TRUE	TRUE FALSE	FALSE TRUE	FALSE FALSE
NOT B1	FALSE	FALSE	TRUE	TRUE
B1 AND B2	TRUE	FALSE	FALSE	FALSE
B1 OR B2	TRUE	TRUE	TRUE	FALSE
B1 IMP B2	TRUE	FALSE	TRUE	TRUE
B1 EQV B2	TRUE	FALSE	FALSE	TRUE

EXAMPLE

Prove that NOT(A AND B) has the same value as (NOT A) OR (NOT B) regardless of the values of A and B by considering the four possible cases.

A	B	NOT(A AND B)	(NOT A) OR (NOT B)
TRUE	TRUE	FALSE	FALSE
TRUE	FALSE	TRUE	TRUE
FALSE	TRUE	TRUE	TRUE
FALSE	FALSE	TRUE	TRUE

EXERCISES

Prove the following identities:

1- (A AND (B OR C)) EQV ((A AND B) OR (A AND C))
2- (A OR (B AND C)) EQV ((A OR B) AND (A OR C))
3- (A IMP B) EQV (NOT B IMP NOT A)

4-3 BOOLEAN EXPRESSIONS

The most elementary components of Boolean expressions are relations and Boolean variables; relations were in fact the only Boolean expressions considered in Chapter One. Simple Boolean expressions are put together with Boolean operators and parentheses much the same as simple arithmetic expressions. Two operators must never appear one after the other, the only exception being that a single NOT may be placed immediately after any of the other four operators. Evaluation proceeds according to the following rules.

1- Arithmetic expressions are evaluated first.
2- Relations are evaluated second.
3- Sub expressions enclosed in parentheses are evaluated next.
4- Unless parentheses over-ride, the operators take the following precedence:
   First NOT
   Second AND
   Third OR
   Fourth IMP
   Last EQV
5- Where none of the above rules apply, evaluation is left to right.

Note that the examples and exercises of Section 4-2 all exhibit valid Boolean expressions although most contain unnecessary parentheses.

EXAMPLES

The following are valid Boolean expressions if A, B, C are real while B1 and B2 are Boolean.

1- A NEQ 0 AND B**2 -4*A*C GTR 0
2- ABS(A+C) LSS B OR (B1 AND NOT B2)

EXERCISES

Assuming X=0, Y=1 and Z=2, evaluate the following expressions:

1- X EQL Y OR NOT Z+2*Y EQL 5 AND Z-2 NEQ 0
2- X EQL Y OR NOT (Z+2*Y EQL 5 AND Z-2 NEQ 3)
3- (X EQL Y OR NOT Z+2*Y EQL 5) AND Z-2 NEQ 0
4- X NEQ 0 IMP Y EQL 1 IMP Z EQL 2

4-4 GENERAL BOOLEAN EXPRESSIONS - THE IF-CLAUSE

The IF-Clause has the form

IF <Boolean Expression> THEN

It deserves special attention here because Boolean expressions as well as IF statements may contain them. The implication of this is that the general definition of Boolean expressions must be recursive. To understand the mechanism, suppose that BYE, BABY and BUNTING are Boolean expressions that do not contain If-clauses. Then the expression

IF BYE THEN BABY ELSE BUNTING

takes the value of BABY if BYE is true and the value of BUNTING if BYE is false, Unlike the IF-statement, the ELSE is essential to the expression. To write simply

IF BYE THEN BABY

does not provide any value in the event that BYE is false. If any of the expressions BYE, BABY or BUNTING contain <If-clause>s then we evaluate them, if necessary, just as the entire expression above. It should then become clear what is meant by the expression

IF IF B1 THEN B2 ELSE B3 THEN IF B4 THEN B5 ELSE B6 ELSE B3

We first evaluate the expression

IF B1 THEN B2 ELSE B3

If the result is false, then the expression takes on the value of B3, if the result is true then we evaluate

IF B4 THEN B5 ELSE B6

EXERCISES

Let A and B be Boolean variables. Make a table of results for each of the following Boolean expressions

1- IF A AND B THEN FALSE ELSE IF A THEN B ELSE B
2- IF IF IF A THEN B ELSE NOT B THEN A ELSE NOT A THEN TRUE ELSE FALSE
3- IF A THEN IF B THEN TRUE ELSE FALSE ELSE B

4-5-0 PARENTHESES IN BOOLEAN EXPRESSIONS

If parentheses are placed around any Boolean expression, no matter how complicated, it takes on the basic character of a Boolean variable. In particular, it can then be used as an operand with the Boolean operators. Technically, a <simple Boolean expression> is any Boolean expression that contains no unparenthesized if-clauses. Thus we have a recursive way to change linguistically complicated constructs back into elementary ones and to build up again.

EXAMPLES

The following are valid Boolean expressions

```
1- B1 AND NOT(IF B2 THEN B3 ELSE B4)
2- (IF B1 IMP B2 THEN B3 ELSE B4) OR (IF B2 IMP B1 THEN
   B5 ELSE B6)
```

4-5-1 A NOTE ON STANDARDS

The metalinguistic definition for Boolean expressions in B6700 ALGOL differs from that given in the ALGOL 60 report in one important respect. B6700 ALGOL allows the construct THEN IF while ALGOL 60 does not. The reason for this restriction is that an expression containing THEN IF is sometimes open to two different interpretations. For example, consider the statement

```
IF B1 THEN IF B2 THEN GO TO L1 ELSE GO TO L2;
```

It is not clear whether we go to L2 or forget the whole thing in the event that B1 is false.

B6700 ALGOL settles the matter with the convention that an ELSE is to be matched with the nearest possible IF. Thus if B1 above is false we ignore the entire statement and continue sequential processing. Nevertheless, it is preferable to avoid such statements since they are apt to mislead other programmers who are familiar with ALGOL 60 but not with the B6700 ALGOL convention. ALGOL 60 and B6700 ALGOL both allow THEN (IF and THEN BEGIN IF. Thus it is always possible to remove ambiguities by inserting bracketing symbols (parentheses in expressions and BEGIN IF...END in a statement.)

EXAMPLE

B6700 ALGOL Allows

```
IF B1 THEN IF B2 THEN B3 ELSE B4 ELSE B5;
```

ALGOL 60 requires instead

```
IF B1 THEN (IF B2 THEN B3 ELSE B4) ELSE B5;
```

5.0 GENERAL ARITHMETIC EXPRESSIONS

The structure of arithmetic expressions is completely analogous to that of Boolean expressions. Thus if AE1 and AE2 are arithmetic expressions that do not contain if-clauses, then the expression

```
IF B THEN AE1 ELSE AE2
```

takes the value of AE1 if B is true and the value of AE2 if B is false. Note that the ELSE AE2 is essential. Lifting the restrictions on AE1 and AE2, we may build more complicated expressions.

EXERCISES

Let X=0, Y=1 and Z=2 compute the values of expressions 1, 2 and 3.

1- IF X NEQ 0 THEN Y ELSE Z
2- IF X+1 EQL Y THEN 0 ELSE IF X+1 GTR Y THEN -1 ELSE 1
3- IF Z*Z GTR 4*Y THEN IF X EQL 0 THEN Z ELSE Y ELSE X
4- Rewrite expression 3 to make it valid in ALGOL 60
5- Is expression 2 valid in ALGOL 60?

5-1 USE OF PARENTHESES

By enclosing an arithmetic expression in parentheses, we give it the character of an arithmetic variable. We may then use it as an operand with the arithmetic operators. Technically, a <simple arithmetic expression> is an arithmetic expression that contains no unparenthesized if-clauses.

EXAMPLES

The following are valid simple arithmetic expressions

```
X+(IF Y GTR 0 THEN -Y ELSE 0)*4
X*(IF B1 AND NOT B2 THEN 0 ELSE 2)
```

The difference between B6700 ALGOL and ALGOL 60 that exists in the construction of Boolean expressions carries over to arithmetic expressions. Thus B6700 ALGOL allows

```
IF X EQL Y THEN IF Z EQL 0 THEN 0 ELSE 1 ELSE 2
```

while ALGOL 60 requires

```
IF X EQL Y THEN (IF Z EQL 0 THEN 0 ELSE 1) ELSE 2
```

in order to avoid the THEN IF construct in expressions.

5-2 RELATIONS

A relation in B6700 ALGOL has the form

<Simple Arithmetic Expression> <Relational Operator>
<Arithmetic Expression>

The relational operators and their alternate forms are:

```
LSS    <
LEQ    NONE
EQL    =
NEQ    NONE
GEQ    NONE
GTR    >
```

<u>EXAMPLES</u>

```
1- (IF B1 THEN X ELSE Y)=Z
2- 0 > IF B1 THEN X ELSE Y
3- X LSS IF Y GTR Z THEN 0 ELSE 1
```

ALGOL 60 requires <simple arithmetic expression>s on both sides of the <relational operator>. Thus example 2 becomes

```
0 >(IF B1 THEN X ELSE Y)
```

6-0 CONTROL STATEMENTS

We have already seen how the GO TO statement used in conjunction with labels provides a means to jump from one place in a program to another. The designational expression and the switch are generalizations of this capability.

6-1 LABELS AND SWITCHES

The statements we have already seen of the form

GO TO <Label>;

are special cases of the general statement

GO TO <Designational Expression> ;

labels are the most elementary designational expressions. They are roughly analogous to numerical constants in arithmetic expressions in that the result of evaluating a designational expression is always a label.

In its simplest form, a switch is a one dimensional array of labels. Like an array, a switch must be declared, for example

```
SWITCH SW:=L1, L2, L3, L4, L5;
```

the word SWITCH is a reserved word. It is followed by an identifier, then the symbol := and finally a list of previously declared labels. Following the declaration above, the statement

```
GO TO SW [2] ;
```

means the same thing as

```
GO TO L2;
```

a subscript having value N refers to the label in the Nth position on the switch list. The switch designator SW [2] is another example of a designational expression. Any arithmetic expression may be used as the subscript of a switch designator. If the value of the subscript is not an integer, it is rounded. If the subscript is less than 1 or larger than the number of labels in the declaration, then the instruction is ignored.

EXAMPLE

The following block illustrates the use of a switch.

```
BEGIN
LABEL L1, L2, L3, L0; INTEGER M, Z, P, I, N; REAL X;
SWITCH SW := L1, L2, L3;
```

```
        READ(CARDS, /, N); FOR I := 1 STEP 1 UNTIL N DO BEGIN
          READ(CARDS, /, X);
        GO TO SW [SIGN(X) + 2] ;
L1:     M := M + 1; GO TO L0;
L2:     Z := Z + 1; GO TO L0;
L3:     P := P + 1;
L0:     END;
      END.
```

6-2 DESIGNATIONAL EXPRESSIONS

Labels and switch designators are termed simple designational expressions. We can make more complicated expressions by combining simple ones with if-clauses. The following is a valid GO TO statement:

```
GO TO IF 1 <J THEN SW [1] ELSE SW [J] ;
```

It has the same meaning as

```
IF 1 <J THEN GO TO SW [1] ELSE GO TO SW [J] ;
```

6-2-1 STANDARDS AGAIN

In the ALGOL 60 report a <simple designational expression> is a designational expression that contains no unparenthesized if-clauses. A <designational expression> is defined either as a <simple designational expression> or as an expression of the form

```
<If-Clause> <Simple Designational Expression> ELSE
<Designational Expression>
```

This is done to avoid the construct THEN IF by requiring the use of parentheses. In B6700 ALGOL parentheses need never appear in designational expressions.

6-3 GENERAL SWITCHES

In its most general form a switch is an array of designational expressions. The following are legal switch declarations.

```
SWITCH S1:=L1, L2, IF 1 <J THEN L3 ELSE L4;
SWITCH S2:=L3, S1 [3], S1 [IF 1 <J THEN 1 ELSE J] ,
IF J NEQ 0 THEN S1 [J] ELSE  S1 [1] ;
```

7-0 FOR STATEMENT

In Chapter One we introduced statements like

```
FOR I:= 1 STEP 1 UNTIL 10 DO
WRITE (PRINTER, /, A , B[I] , C[I,I] );
```

We shall now examine this statement in depth and also introduce two variations on the theme of loop control.

7-1 REAL LOOP VARIABLE

Typical usage of FOR statements are like the one above in that the loop variable is an integer, as are initial value and terminal value. All of these conditions, however, are special. The loop variable may be real, the initial value, terminal value and step sizes may be given by complicated arithmetic expressions, and the terminal value might not be assumed by the loop variable. The following examples are all valid statements, but it is not immediately clear how they are handled.

```
1- FOR X:=0 STEP COS(X)-X UNTIL 1.05 DO <STATEMENT> ;
2- FOR I:=2 STEP 1 UNTIL 0 DO <STATEMENT> ;
3- FOR Z:=0 STEP (-1)**Z UNTIL 2*(-1) **Z DO <STATEMENT>;
```

The rule to follow in determining the behavior of such expressions is: If E1, E2, and E3 are arithmetic expressions then the statement

```
FOR V:=E1 STEP E2 UNTIL E3 DO <STATEMENT> ;
```

is equivalent to

```
V:=E1;
L;IF (V-E3)*SIGN(E2) LEQ 0 THEN
BEGIN <STATEMENT> ;
V:=V+E2; GO TO L END;
```

Thus we see that in example 1 above the loop will be executed once; in example 2 the loop will not be executed at all; in example 3 the loop is infinite.

7-2 THE RESERVED WORD WHILE

The first variation of the FOR-Statement has the form

```
FOR V:= AE WHILE BE DO <STATEMENT> ;
```

Where AE is any arithmetic expression and BE is any Boolean expression, execution is identical to execution of

```
    V := AE;
L: IF BE THEN BEGIN
   <STATEMENT> ;  GO TO L END;
```

The word WHILE may also be used instead of UNTIL in conjection with a STEP part.

```
FOR I:=0 STEP 1 WHILE X [1] NEQ Y DO <STATEMENT> ;
```

The general form

```
FOR V:= AE1 STEP AE2 WHILE BE DO <STATEMENT> ;
```

is equivalent to

```
V:= AE1;
L; IF BE THEN BEGIN
   <STATEMENT>
   V:=V+AE2;
   GO TO L END;
```

EXAMPLES

1 - The following pair are equivalent

```
A)  FOR I:=1 STEP 1 UNTIL 10 DO <STATEMENT>
B)  FOR I:=0 WHILE I:=I+1 LEQ 10 DO < STATEMENT >
```

2 - A Loop to generate a table of squares

```
FOR N:=0 WHILE N:=N+1 LEQ 100 DO BEGIN
NSQ:=N*N; WRITE (PRINTER, /, N, NSQ) END;
```

7-3 GENERAL FOR-STATEMENT

The second variation is simply an arithmetic expression between the symbol := and the word DO.

```
FOR X:=Y+2 DO <STATEMENT> ;
```

Although it is not very useful by itself, the importance of this form lies in the fact that ALGOL allows FOR-Statements to include a list containing all three varieties, for example, the statement

```
FOR I:=10,27,30 STEP 5 UNTIL 50,64,64, WHILE I:=I+11 LEQ 100
DO S1;
```

causes the statement S1 to be executed for I assuming the values 10,27,30,35,40,45,50,64,75,86 and 97.

EXERCISES:

Assuming J=5 and K=3 list the values assumed by the loop variable N in each of the following FOR-Statements.

```
1- FOR N:=J STEP -1 UNTIL K,K-2 STEP -1 UNTIL 0 DO
2- FOR N:=J,N+K STEP K UNTIL 20 DO
3- FOR N:=1,1 WHILE N:=N*(N+1)<1000 DO
4- FOR N:=0,0 WHILE N:=N+1 LEQ K, K+1 STEP 1 UNTIL 2*K DO
5- FOR N:=0 STEP 1 UNTIL 7,9 DO
```

8-0 FORMATTED INPUT/OUTPUT

The write statement we have been using so far makes output extremely simple to handle, but the programmer pays a high price in versatility. It is impossible, for example, to print a 7 x 7 square array so that it looks like a square. However, with a little extra effort, printed output can be made to take practically any desired form. Similarly, the format of data cards to be read can be programatically prescribed, but we shall concentrate first on output.

8-1 THE WRITE STATEMENT

Figure 1 shows a program that generates and prints an eight by eight array of integers and also shows the output produced by this program. Three important constructs are involved. They are a file declaration, a format declaration and a WRITE statement. The file declaration shown associates an identifier, PT, with an output device, a 132 character printer. A detailed explanation will be presented later. The format declaration associates an identifier, F100, with a list of editing phrases enclosed in parentheses, it is this list that we are concerned with most.

```
BEGIN
FILE PT (KIND=PRINTER, MAXRECSIZE=22);
FORMAT F100( X5, 8I4/);
INTEGER ARRAY NA [1:8, 1:8] ;
INTEGER I,J;
FOR I:=1 STEP 1 UNTIL 8 DO
FOR J:=1 STEP 1 UNTIL 8 DO
NA [I, J] :=8*(I-1)+J;
FOR I:=1 STEP 1 UNTIL 8 DO
WRITE(PT, F100, FOR J:=1 STEP 1 UNTIL 8 DO NA [I,J]);
END.
```

```
     1   2   3   4   5   6   7   8

     9  10  11  12  13  14  15  16

    17  18  19  20  21  22  23  24

    25  26  27  28  29  30  31  32

    33  34  35  36  37  38  39  40

    41  42  43  44  45  46  47  48

    49  50  51  52  53  54  55  56

    57  58  59  60  61  62  63  64
```

FIGURE 1

The write statement contains a file identifier, a format identifier, and an output list. The word WRITE is a reserved word. The output list is just a list of variables to be written. List elements may be arbitrary arithmetic or Boolean expressions; the format identifier refers back to a list of editing phrases defined in a format declaration. It describes the desired appearance of the output.

The file identifier specifies which output device is to be used. Not only the printer, but also magnetic tape, disk, punched cards, and other equipment are available. Although output to any of these devices can be formatted, the remainder of this section will take the standpoint of output to a 132 character line printer.

8-2 FORMAT SPECIFICATIONS

Editing phrases may be divided into two classes. Class one consists of those which describe how a variable in the output list is to appear. For example, the number 3.141592653588 stored in the machine might be printed in any of the following ways:

```
3.1416
3.142@+00
3.14159265
3
```

Class two consists of format specifications other than those needed to output a list element. For instance, we may write labels, skip spaces between printed numbers, or skip lines.

When a WRITE statement is executed, the list of editing phrases in the format is scanned left to right until a class one phrase is found. Class two phrases encountered during the scan are executed, and the scan is continued. The first element in the output list is written in the form specified by the first class one editing phrase, and scanning resumes for an editing phase for the second number of the output list. This process continues until the output list is exhausted. Usually, the number of class one editing phrases will be the same as the number of elements in the output list although we shall see that this is not required.

8-2-1 THE I AND X EDITING PHRASES

The editing phrase I12 calls for an integer to be written right-justified in a field of 12 spaces. The field width may be any number besides 12 so long as the total width of the line is not exceeded.

The phrase X24 calls for a field of 24 blanks. The leftmost position of the field specified by the first editing phrase is the leftmost column of the printer. Thus X formats can be used to indent printed output.

EXAMPLE

Let J=1, K=23, L=456 and suppose the following delcarations are in effect:

```
    FILE PR (KIND=PRINTER, MAXRECSIZE=22);
    FORMAT F100(I3,I3,I3,I3);
    FORMAT F121(X3,I1,X3,I2,X3,I3);
```

Then the result of

```
    WRITE(PR,F100,L,J,K,L);
    WRITE(PR,F121,J,K,L); is
```

```
456  1 23456
   1   23   456
```

If the value of a list element to be written is too large for the field specified, then the field is filled with asterisks instead. This is true of all the class one editing phrases.

A minus sign preceeds negative numbers, thus the range of the I3 phrase is -99 to 999. Any other integer will appear as three asterisks. If a real number is printed under the I phrase, it is first rounded.

EXAMPLES

Let J=27.4 K=1000, L=-3.2, M=-99.6 and suppose the declarations of the above example are in effect. Then the result of

```
    WRITE(PR,F100,J,K,L,M);     is
```

```
27*** -3***
```

EXERCISES

1 - Write a program to generate and print the following:

```
1 2 3 4 5 6 7
 2 3 4 5 6 7
  3 4 5 6 7
```

2 - Write the format statement one might use to print three six-digit integers so that they are separated by three spaces and centered on a 132 character line.

8-2-2 STRINGS

In order to insert labels into printed output, we use quoted strings inside the format specifications. A string is a class two phrase.

EXAMPLE

Let I=5, A[5]=239 and F100 refer to ("A[",I1,"] =",I4) then

```
WRITE(PR,F100,I,A[I]); produces
```

```
A[5] = 239
```

Headings can be printed by means of a format specification that contains no class one editing phrases in conjunction with an empty output list.

EXAMPLE

If F100 refers to ("**"X10,"COLUMN ONE",X10,"COLUMN TWO", X5,"*****") then

```
WRITE( PR,F100 ); produces
```

```
**          COLUMN ONE          COLUMN TWO     *****
```

Along this same line, a sequence of class two phrases following the last required class one phrase will be executed.

EXAMPLE

If F223 refers to ("**",X5,I3," RESULTS"/"NEW") and K=14 then

```
WRITE( PR,F223,K ); produces
```

```
**     14  RESULTS
NEW
```

8-2-3 SLASH

A slash signals the end of a line and the start of a new one. Commas separating the slash from preceding and following phrases are optional.

EXAMPLE

If FORM refers to (I3,I3,/,I3,I3) then

```
WRITE( PR,FORM,FOR I := 0 STEP 2 UNTIL 6 DO I );   produces
```

```
  0  2
  4  6
```

Two slashes together allows us to skip a line.

EXAMPLE

If FORM refers to (I3,I3//I3,I3) then the example above produces

```
0 2

4 6
```

8-2-4 REPEAT FACTORS

An integer preceeding a class one editing phrase is a repeat factor. Thus (4I3,2I5) is equivalent to (I3,I3,I3,I3,I5,I5).

Class two phrase or entire format specifications may be repeated by inserting parentheses. For example,

```
(I3,2(I4,X5),3(/),3(I1,I3))    and
(I3,I4,X5,I4,X5 /// I1,I3,I1,I3,I1,I3 )
```

are equivalent. This construct is sometimes referred to as a repeated group.

EXERCISES

1 - Let A be a 6X10 array of integers. Write the statements necessary to print A so that it looks like a 6X10 array.

2 - Let B be an array of one-digit integers of length N. Write the statements necessary to print them all in as few lines as possible. (Recall that the printer has 132 columns.)

3 - Let A be a 10X10 square matrix. Write a program section to print the lower left triangle in triangular form.

8-2-5 UNLIMITED GROUPS

An infinite repeat factor can be specified by enclosing a list of editing phrases in parentheses with the repeat factor omitted. In this case, the right parenthesis is treated as a slash. Thus

```
(I7,(I3,I2))   is equivalent to
(I7,I3,I2/I3,I2/I3,I2/ . . .
```

Repeated groups and unlimited groups can be nested, although it makes no sense to nest one unlimited group within another.

A format is itself an unlimited group. Thus if the scan reaches the last right parenthesis before the output list is exhausted, a new line is started and scanning begins again from the beginning of the format.

EXAMPLE

If F227 refers to ("LINE"I3,3I4) then

```
WRITE(PR,F227,FOR I:=1 STEP 1 UNTIL 10 DO 2*I);
```

produces

```
LINE  2   4   6   8
LINE 10  12  14  16
LINE 18  20
```

EXERCISES

Rewrite each of the following formats with the inner parentheses removed. Show two repetitions of any unlimited groups.

```
1- (I5,2(I3,/),I5)
2- (2(I3,3(I1,X1,I2)))
3- (X15,(I5,I3),"HELLO")
4- (X5,3("GEVALT"/),I20)
5- (2(I5,X3,(I4/)))
```

8-2-6 GENERAL GUIDELINES

Three general rules to follow when writing formats are

1- The output list must be satisfied. The entire format behaves like an unlimited group if necessary.

2- Scanning proceeds as far as possible after the list is satisfied. It is then terminated by the last right parenthesis or any class one editing phrase. See the example of section 8-2-2.

3- Commas are optional if their absence does not introduce ambiguity. The format (I32I4) is equivalent to (I32,I4) not (I3,2I4).

8-3-1 DECIMAL POINTS

There are several ways available to output quantities of type real. The phrase F9.2 specifies a real number with two decimal places and a field of nine spaces. Since the decimal point itself requires a space, the range of this phrase is -99999.99 to 999999.99. A number out of range results in a field of asterisks. Any integers may be used instead of 9 and 2, but in the general form F<W>.<D> we must have W≥D+2 to allow for the decimal point and a possible minus sign. If the number being written does not require all the spaces in the field, it is right justified. The phrase F<W> is not valid.

EXAMPLES

If FFIELDS refers to ("**"F5.2,F5.1,F5.3,F5.0) and P1 has the value 3.14159265359 then

```
WRITE( PR,FFIELDS,P1,P1,P1,P1 );
```

produces

```
** 3.14  3.13.142   3.
```

8-3-2 E PHRASE

The E-phrase specifies a real number written in scientific notation as a power of 10 times a number between 1 and 10. The phrase E10.2 specifies a total field width of 10 spaces and 2 spaces to the right of the decimal point. Under this format, the variable P1 above would appear as 3.14@+00. Note that besides the 2 places to the right of the decimal point, we must allow 4 spaces for the exponent part, one for the decimal point, one for a digit to the left of the decimal point, and one for a possible minus sign. Thus in the general form E<W>.<D>, we must have W≥D+7. The phrase E<W> is not valid. The range of the E-phase covers the full range of the B6700 single precision.

EXAMPLE

If EFIELDS refers to ("**"E10.2,E10.0,E10.0) and P1 has the value above, then

```
     WRITE( PR,EFIELDS,P1,P1,P1 );   produces

**  3.14@+00    3.@+00   3.@+00
```

8-3-3 R PHRASE

The R editing phrase combines the E and F phrases. A quantity printed with R<W>.<D> is output using F<W>.<D> formatting if possible; otherwise, it is output under E<W>.<D>. For example under R9.2 the number 123456.789 is printed as 123456.79 while the number 1234567.8 is printed as 1.23@+06. If W<D+7 then the R phrase is identical to the F phrase since the E option cannot be taken.

8-3-4 ACCURACY

Since numbers are stored with no more than 12 significant digits, nothing is to be gained by a phase such as E30.20. In this case the variable P1 might be printed as

```
     3.14159265359417639042@+00
```

Although the string of digits looks impressive, those after the twelfth are quite meaningless.

8-3-5 SCALE FACTOR

The phrase S<W> calls for subsequent numbers printed under the R-phrase to be multiplied by 10**W.

EXAMPLE

If SF refers to ("**", S1,R10.3,S3,R10.3,S0,R10.3) then

```
     WRITE( PR,SF,P1,P1,P1 );   produces

**    31.416  3141.593    3.142
```

8-3-6 L-PHRASE

Boolean expressions may be printed under the L editing phrase.

EXAMPLE

If BB is false and LFIELDS refers to ("**"L2,L5,L10) then

```
WRITE( PR[SKIP1], LFIELDS, BB, BB, BB; produces
```

```
**FAFALSE     FALSE
```

8-4 VARIABLE FORMAT

If an asterisk is encountered outside a quoted string during the format scan then the value of the current list element replaces it.

EXAMPLE

If VF refers to ("**",X*,"HI") then

```
WRITE(PR,VF,FOR I:=1 STEP 1 UNTIL 5 DO I); produces
```

```
** HI
**  HI
**   HI
**    HI
**     HI
```

EXERCISE

1- Write a program to produce 15 lines of Pascals triangle in triangular form.

8-5 CARRIAGE CONTROL

Printer controls can be specified by a bracketed instruction immediately following the file identifier.

```
WRITE(PR[SKIP1]) ,FMT,X,Y,Z);
```

The instruction illustrated causes the printer to move to the top of the next page after each line designated by the format is output. In general, the carriage control instruction is executed after each printed line.

The other control instructions contain either SKIP or SPACE. The instruction SPACE 4 causes the printer to advance 4 lines. Although an arbitrary arithmetic expression may be used, its value should not be less than zero nor more then 120.

The instruction SKIP 4 causes the printer to execute the channel 4 command on its carriage control tape. There are eleven channels on this tape numbered from 1 to 11. They function as follows:

Channel	
1	move to top of form (line 4)
2	move to half page (lines 4, 34)
3	move to one-third page (lines 4, 24, 44)
4	move to one-quarter page (lines 4, 19,34,49)
5	move to line 65
6	unconditional single space (66 lines per page)
7	move to line 65
8	move to line 63
9	move to line 66
10	move to line 1
11	move to top of even page (up when folded)

8-6 FORMATTED INPUT

Formatted input is done by means of a statement like

```
READ(CARDFILE,FMT,V1,V2,V3);
```

Basically, the mechanism is the same as the write statement. The identifier CARDFILE must have been declared previously to be an input device. The identifier FMT refers to a format which is scanned in parallel with the input list.

The file declaration needed to define a card-reader is

```
FILE CARDFILE (KIND=READER);
```

Any identifier may be used instead of CARDFILE. Although other input devices such as magnetic tape and disk are available, we shall concern ourselves here only with punched cards. In the examples that follow, we shall assume that three data cards appear as follows

```
1234567890 XXXXX1.204@-01XX 1.204@-01
1234567890 XXXXX1.204@-01XX 1.204@-01
1234567890 XXXXX1.204@-01XX 1.204@-01
```

8-6-1 SLASH AND X

Class two editing phrases consist of slash and X. A slash calls for the start of a new card. The phrase X<W> calls for W columns to be ignored.

8-6-2 I-PHRASE

Reading under I<W> calls for an integer to appear in a field of width W. Any symbols other than plus, minus, the ten digits, and blank will cause an error. Imbedded and trailing blanks are treated as zeroes.

EXAMPLE

If FI refers to (X8,I3) then

```
READ( CARDFILE,FI,J );
```

assigns J the value 900.

8-6-3 E and F PHRASES

The phrase F<W>.<D> calls for a number in a field <W> characters wide. If the decimal point is missing, from the field, it is assumed to be immediately to the right of the field, and the number is multiplied by 10^{-D}.

The E<W>.<D> phrase calls for a number in scientific format. The exponent need not be present, in which case the phrase is equivalent to the F phrase described above. If the decimal point is missing, it is assumed to be immediately to the left of the exponent, and the number is multiplied by 10^{-D}.

EXAMPLE

If FMT refers to (F5.3, X11, E9.4, X3, F5.3, /, E6.3) and the data cards of section 8-6 are read by

```
READ (CARDFILE, FMT, X, Y, Z, A);
```

then we will have X=12.345, Y=1.204@-01, Z=1.204, and A=123.456

8-6-4 R-PHRASE

The R-editing phrase offers some degree of latitude in the form of the data. The phrase R10.3, for example, calls for a number in either the E or F form contained in a field of 10 spaces. The decimal point need not be present, but if it is, then it over-rides the format. The letter E carries the same meaning as the symbol @ and the symbol & is interpreted as +.

EXAMPLE

If FMT refers to (R6.1,X10,R9.1/R6.3) and the data cards of section 8-6 are read by

```
READ(CARDFILE,FMT,X,Y,Z);
```

then we will have X=12345.6 Y=1.204@-01 Z=123.456

8-6-5 L-PHRASE

The phrase L<W> calls for a Boolean quantity right justified in a field of width W. If W equals one, then either the letter T or the letter F is expected. If W is larger than four, then the word TRUE or the word FALSE is expected.

9-0 BLOCK STRUCTURE

So far we have concerned ourselves with the detailed structure of ALGOL statements. We now turn to the overall organization of programs. Recall that a block starts with the reserved word BEGIN followed by at least one declaration, then at least one statement, and is terminated by the reserved word END. In particular a program is a block. Furthermore, since blocks are operationally the same as single statements, they may be nested. Thus if one looks only at the words BEGIN and END, various levels of nesting can be recognized. The level decreases by one each time an END is reached. This notion of level taken together with dynamic storage allocation, gives rise to a fundamental rule of program organization.

> The meaning of an identifier appearing within a nest of blocks is determined by the highest (i.e. innermost) level at which that identifier is declared.

Dynamic storage allocation is a means of changing the amount of reserved storage needed depending on conditions during execution. It is implemented by means of the rule that any space reserved by declarations at the beginning of a block is released upon exit from that block. This means that the entity represented by an identifier declared within a block does not exist outside that block. It then follows that if the same identifier is declared in a block and again in an inner block, the entity represented by the identifier in the outer block is completely inaccessible to the inner block.

EXAMPLE 1

```
BEGIN
FILE P(KIND=PRINTER, MAXRECSIZE=22);
REAL X;
X:=1;
   BEGIN               This program results in the numbers
   REAL X;
   X:=5;                5,
   WRITE(P,/,X);        1,
   END;                being printed.
WRITE(P,/,X)
END.
```

EXAMPLE 2

```
BEGIN
FILE P(KIND=PRINTER, MAXRECSIZE=22);
FORMAT F100 (F7.4);
LABEL L1;
BOOLEAN B;
B:=FALSE;
   L1:BEGIN            This program results in the numbers
   REAL L1;
   L1:=2.7183;              2.7183
   WRITE(P,F100,L1);        2.7183  being printed.
```

```
          B:=NOT B;
          END;
     IF B THEN GO TO L1;
     END.
```

Since B is not declared in the inner block, it retains the meaning ascribed to it in the outer block.

9-1 LOCAL AND GLOBAL IDENTIFIERS

We say that an identifier is local to a block if it is declared in that block. In other words, the entity it represents will be eliminated upon exit from the block. An identifier is said to be global to a block if it is not declared in that block but is declared in an outer block. Thus a global identifier carries the same meaning both inside and immediately outside the block.

9-2 RESTRICTIONS ON USE OF LABELS

Although labels appearing within designational expressions behave just like other identifiers with respect to the fundamental rule, labels that are used in front of statements are inherently local to the block in which they appear. This means that we may exit a block from the middle by means of a GO TO statement, but a block can be entered only through its BEGIN. Compound statements, however, can be entered in the middle.

EXAMPLES

Block one is valid and it is not a closed loop. Block two is not valid because the GO TO leads into the middle of a block. Block three is all right since the GO TO leads into a compound statement

```
BEGIN                   BEGIN              BEGIN
LABEL L; REAL X;        LABEL L1;          LABEL L;
L: X := 1;              GO TO L1;          REAL X;
BEGIN                   BEGIN              X := 1;
LABEL L;                REAL X;            GO TO L;
WRITE(P,/,X);           L1: X := 1;        BEGIN
GO TO L;                END;               X := X+1;
L: END;                 END:               L: WRITE(P,/,X);
END;                                       END;
                                           END;

Block one               Block two          Block three
```

9-3 ARRAYS

Dynamic storage is especially useful when large arrays of temporary storage are required and it is not known in advance exactly how much is needed. ALGOL provides for simple and efficient handling of this situation by allowing subscript bounds in array declarations to contain arbitrary arithmetic expressions provided the variables involved are defined in an outer block.

EXAMPLE

```
BEGIN
INTEGER N;
READ(CARDS,/,N);
  BEGIN
  ARRAY A[-N:N],B[1:N,1:N];
```

If several arrays with the same dimensions are to be declared, ALGOL allows a shorthand declaration. The following two declarations have the same meaning:

```
ARRAY A,B,C[0:X*2],D,E[0:1];
ARRAY A[0:X*2],B[0:X*2],C[0:X*2],D[0:1],E[0:1];
```

9-4 OWN VARIABLES

We have already seen that when a program exits a block, storage reserved for local variables is released. Upon re-entering the block, the values of these variables are initially undefined. However, if a local variable is declared OWN, then upon re-entering the block, it will initially have the value it had at the last exit. The word OWN is a reserved word. It may appear in front of any type declarator.

```
OWN REAL ARRAY A,B,C[1:10,2:4];
OWN BOOLEAN B,BB,BBB,BBBB
OWN INTEGER X;
```

A type declarator such as INTEGER or REAL must follow the word OWN.

Dynamic own arrays are not implemented in B6500 ALGOL because of the ambiguity inherent in their definition. For example, the declaration

```
OWN REAL ARRAY Y[0:N];
```

is not valid since the value of N may change before the block is re-entered.

10-0 PROCEDURES

In Chapter one the method for declaring and invoking procedures was introduced. We shall now examine the mechanisms involved in detail. Three concepts will be covered: (1) local and global variables, (2) call by name and call by value, and (3) recursive invocation.

10-1 LOCAL AND GLOBAL VARIABLES

When a procedure is called the actual parameters are substituted into the procedure body wherever formal parameters appear, and the procedure body is then executed as if it were a new block. Thus the same rules that apply to block structure also apply to procedures. In particular, identifiers that appear in the body can be classified as either local or global to the procedure, and it is not possible to transfer control to a label inside a procedure except from a GO TO statement inside that procedure.

EXAMPLE

The following two blocks are essentially identical.

```
BEGIN
REAL X,Y;
INTEGER N;
PROCEDURE COMP(U,V);
   REAL U,V;
   BEGIN
   INTEGER I; REAL Z;
   FOR I:=1 STEP 1 UNTIL N DO
      BEGIN
      Z:=(U-V)*I;
      WRITE(P,/,Z);
      END;
   END;
READ(C,/,N,X,Y);
COMP(X,Y);
END;
```

```
BEGIN REAL X,Y;
INTEGER N;
READ(C,/,N,X,Y);
   BEGIN
   INTEGER I; REAL Z;
   FOR I := 1 STEP 1 UNTIL N DO
      BEGIN
      Z := (X-Y)*I;
      WRITE(P,/,Z);
      END;
   END;
END;
```

In the left-hand one, the identifiers I and Z are local to the procedure while the identifier N is global. Note that the declaration of N precedes the procedure declaration.

10-2 CALL BY NAME AND CALL BY VALUE

The actual parameters of a procedure call may be arbitrary arithmetic expressions. Ordinarily, the entire expression, parenthesized if necessary, is substituted for the formal parameters in the procedure body. However, we may specify that the values of the actual parameters be substituted instead by means of the value specification.

```
PROCEDURE P(X,Y);
VALUE X,Y;
REAL X,Y;
```

The word VALUE is a reserved word, and the value specifications must precede the type specifications. Only parameters which have values may be called by value. Thus arguments which are specified as labels, arrays, or untyped procedures cannot be called by value.

EXAMPLE

The following two blocks are essentially identical. Note that when parameters are called by value, the procedure body behaves like two blocks one nested within the other. The evaluation of actual parameters takes place in the outer block, and the procedure body corresponds to the inner block. Notice that the value of Y is not changed by the procedure call.

```
BEGIN
REAL X,Y;
PROCEDURE COMP(U,V)
   VALUE U,V; REAL U,V;
   BEGIN
   V:=U*U;
   WRITE(P,/,V);
   END;
Y:=0;
READ(C,/,X);
COMP(2*X-5,Y);
WRITE(P,/,Y);
END;
```

```
BEGIN
REAL X,Y;
Y := 0;
READ(C,/,X);
   BEGIN
   REAL TU,TV;
   TU := 2*X-5;  TV := Y;
      BEGIN
      TV := TU*TU;
      WRITE(P,/,TV);
      END;
   END;
WRITE(P,/,Y);
END;
```

10-3 RECURSIVE INVOCATION

ALGOL allows procedures to call themselves. Recall that the procedure identifier of a typed procedure must appear as the left part of an assignment statement contained within the procedure body. The appearance of the procedure identifier on the right side of an assignment statement invokes the procedure again.

EXAMPLE

The following procedure returns the value TRUE if the array A has an element equal to the variable X and the value FALSE otherwise.

```
BOOLEAN PROCEDURE E(X,A,N); VALUE X,N;
REAL X; INTEGER N; ARRAY A[0];
E := IF N = -1 THEN FALSE ELSE
     IF A[N] = X THEN TRUE ELSE
     E( X,A,N-1 );
```

An ancient puzzle known as the towers of Hanoi consists of three spikes upon one of which a number of disks of different diameters are placed largest on the bottom, smallest on top. The object is to move the pyramid thus formed to one of the other spikes subject to the following rules:

1- Only the top disk on any spike may be moved.
2- A disk may not be placed over another of smaller diameter.

A program to produce a list of moves which will transfer a tower of 10 rings from spike 1 to spike 2 follows:

```
BEGIN
FILE P ( KIND=PRINTER, MAXRECSIZE=22);
FORMAT F (I3 "TO" I3);
PROCEDURE HANOI (A,B,C,N); VALUE A,B,C,N;
  INTEGER A,B,C,N;
  IF N=1 THEN WRITE (P,F,A,B) ELSE
  BEGIN HANOI(A,C,B,N-1);
  WRITE(P,F,A,B);
   HANOI(C,B,A,N-1)END;
HANOI (1,2,3,10);
END.
```

10-4 PARAMETER DELIMITERS

We have been using commas to separate the arguments of a procedure. The symbols)" <String> "(behave exactly like a comma and so the following two procedure calls are equivalent.

```
IDEAL(P,T,V);
IDEAL(P)"PRESSURE"(T)"TEMPERATURE VOLUME" (V);
```

10-5 SIDE EFFECTS

The existence of global variables in a procedure opens a devastating trap for careless programmers. The following example illustrates what might happen:

```
BEGIN
INTEGER E,N;
INTEGER PROCEDURE F(X);
  INTEGER X;
  BEGIN  F := X MOD N;
  E:=E+1;END;
.
.
.
IF E <F(X) THEN ...
```

The invocation of the procedure F changes the value of E, therefore the rules given in Section 1-3 for evaluating Boolean expressions are not sufficient for determining the value of this expression.

Although it is at times necessary to introduce side effects, the practice should be avoided if at all possible and handled with extreme caution otherwise.

11-0 COMMENTS

There are three ways to insert comments into a program. The comment statement was described in Part One. If a percent sign is punched anywhere except within quotes, then the remainder of the card is printed but not considered part of the program.

```
I:=0;J:=K; %INITIALIZE COUNTERS
```

Letters or digits inserted between the reserved word END and a semicolon are also treated as comments.

END CHAPTER TWO ;

12-0 INTRODUCTION

Burroughs ALGOL includes many facilities which are not included in ALGOL 60. Among these are provisions for handling character strings, manipulating bits within words, and debugging. There are also various additional constructs used for describing iterations, input-output operations, and in-line Macro-instructions. The first sections of this chapter deal with the extensions which are machine independent. Then follows a general description of the B6700 and finally a treatment of those aspects which depend on the structure of the machine.

13-0 LIST STRUCTURES

It is possible to assign a name to the IO list of a READ or WRITE instruction by means of a declaration such as the following:

```
LIST LST1(X,Y,Z, FOR J:=1 STEP 1 UNTIL 20 DO A[J]);
```

The word LIST is a reserved word, and the members of a list must consist of previously declared identifiers. Arbitrary arithmetic or Boolean expressions as well as simple variables are allowed provided the list is not used for input. Unfortunately, the members of a list may not themselves be lists.

EXAMPLE

Given the declaration above, the following two statements are equivalent.

```
READ (C,/,LST1);
READ (C,/,X,Y,Z, FOR J:=1 STEP 1 UNTIL 20 DO A[J]);
```

Only one list may be used with a READ statement.

```
READ (C,/, LST1, LST2);
```

is not valid.

13-1 IO SWITCHES

The concept of switching has been extended to files, formats, and lists. A switch list may be declared by

```
SWITCH LIST SWL:=LSTI,LSTJ,LSTK;
```

Where LSTI, LSTJ, and LSTK are previously declared lists, the list LSTI is associated with the number 0, LSTJ with 1 and LSTK with 2. Thus if IND equals 1 then the following two statements are equivalent:

```
WRITE(PR,F100,SWL[IND]);
WRITE(PR,F100,LSTJ);
```

The switch list may have any number of elements, and any arithmetic expression may be used for the index IND. If the index is not an integer, it is first rounded. If the result is less than zero or greater than or equal to the number of elements in the switch list, then an error occurs.

Switch files are handled in the same manner as switch lists. For example, if PRT and PCH have previously appeared in file declarations, then the declaration

```
SWITCH FILE SWF:=PRT,PCH;
```

establishes a switch file. The following two statements are then equivalent

```
WRITE (SWF[0],F100,X,Y,Z);
WRITE(PRT,F100,X,Y,Z);
```

The same rules apply to switch files and switch lists. In the example above, a reference to SWF[I] produces an error unless I is either 0 or 1.

The declaration of a switch format is somewhat different in that the formats themselves may be used instead of identifiers assigned in previous declarations.

```
SWITCH FORMAT SWFMT := (X10,I5),(X20,I5),(X30,I5),FMT1,FMT2;
```

For example, SWFMT[0] now refers to (X10,I5).

<u>EXAMPLES</u>

```
BEGIN
FILE PNT (KIND=PRINTER, MAXRECSIZE=22);
FILE PCH (KIND=PUNCH, MAXRECSIZE=14);
REAL X,Y,Z,W;
INTEGER I,J;
LIST LST0(Y,Z,W),LST1(X,Z,W),LST2(X,Y,W),LST3(X,Y,Z);
SWITCH FILE SP:=PCH,PNT;
SWITCH LIST SL:=LST0,LST1,LST2,LST3,
SWITCH FORMAT SF;=(X12,3R12.2),(R12.2,X12,2R12.2).
                  (2R12.2,X12,R12.2),(3R12.2,X12);
X:=11.2; Y:=22.3; Z:=33.4; W:=44.5;
FOR I:=0,1 DO
FOR J:=0 STEP 1 UNTIL 3 DO
WRITE(SP[I],SF[J],SL[J]);
END.
```

The following is both printed (PNT) and punched (PCH).

```
        22.3    33.4    44.5
11.2            33.4    44.5
11.2    22.3            44.5
11.2    22.3    33.4
```

13-2 CONTROL STATEMENTS

In all of this section the notation <statement>denotes any statement in the broad sense, i.e. simple statement, compound statement or block. The notation <BE> refers to any Boolean expression and <AE> to any arithmetic expression.

The statement

```
DO <statement> UNTIL <BE>;
```

is equivalent to

```
L1: <statement>;
IF NOT <BE> THEN GO TO L1;
```

The statement

```
WHILE <BE> DO <statement>;
```

is equivalent to

```
L2:  IF NOT <BE> THEN GO TO L1;
<statement>;  GO TO L2;
L1:
```

The statement

```
THRU <AE> DO <statement>;
```

causes the arithmetic expression to be evaluated and rounded to the nearest integer. The <statement> is then executed that many times.

The construct

```
CASE <AE> OF BEGIN
<statement 0>;
<statement 1>;
  .
  .
<statement N>END;
```

calls for the following action: First the arithmetic expression is evaluated and rounded to the nearest integer if necessary. This value is then used to determine which statement within the compound statement is to be executed. The statements are indexed from 0. If the result of evaluating the arithmetic expression is less than zero or greater than or equal to the number of statements, then an error occurs. Note that only one of the N+1 statements is executed. If the selected statement does not transfer control elsewhere, control then passes to the statement following the END.

Expressions can also be handled in this way.

```
X:= CASE I OF (2,4,6, Y+Z, A[J]/3,0);
GO TO CASE IND OF (L1, SW[J], IF B THEN L2 ELSE L3);
```

13-3 REPLACEMENT EXPRESSIONS

An expression which contains the replacement operator := can be contained within an arithmetic or Boolean expression. Its value is simply the value assigned to its left hand side, for example,

```
C := X + Y := Z + W ;
```

is equivalent to

```
Y := Z + W;
C := X + Y;
```

The replacement operator holds a special place in relation to the hierarchy of operator precedence. Note that in the example above, the two additions are performed as if the replacement expression on the right were parenthesized. However, the statement

```
C := Y + Y := Z + W;
```

is equivalent to

```
C := Y + Z + W;
Y := Z + W;
```

This may violate one's intuition about how replacement expressions behave, but there is no cause for confusion so long as side effects are avoided. A complete treatment of these expressions will be included later with a discussion of the compilation algorithm for arithmetic expressions.

Program readability is sometimes impaired by complicated replacement expressions, but their use results in more efficient execution. Hence, they should be used wherever possible in programs for which execution time is a primary concern.

EXERCISES

Assume all variables have been declared integer and determine in each case what is printed.

```
1 - K:=0; N:=1;  DO  K:=K+N  UNTIL  N:=N+2 GTR 9; WRITE(P,/,I);

2 - N:=91; I:=2;
    WHILE (I:=I+1)**2 LEQ N DO
    IF N MOD I = 0 THEN WRITE (P,/,I);
```

```
3 - I := 10;  THRU I DO I := I+1;
    WRITE (P,/,I);

4 - N := 0;
    FOR I := 0 STEP 1 UNTIL 3 DO
    CASE I OF BEGIN
    N:=N+2; N := N+I; N:=N+I*I;
     WRITE (P,/,N) END:
```

13-4-1 DEFINE DECLARATIONS

The DEFINE declaration allows an arbitrary string of program text to be associated with an identifier.

```
DEFINE FRI=FOR I:=1 STEP 1 UNTIL #,
       FRJ=FOR J:=1 STEP 1 UNTIL #;
```

The word DEFINE is a reserved word. Following the declaration, any appearance of the identifier to the left of the equal sign has precisely the same meaning as the appearance of the string delimited by = on the left and # on the right. Undeclared identifiers may appear provided they have been declared before the defined identifier is used.

EXAMPLES

```
WRITE (FIL,FORMT,FRI 10 DO FRJ 10 DO A[I,J]);
```

Is equivalent to

```
WRITE (FIL,FORMT,FOR I := 1 STEP 1 UNTIL 10 DO
                 FOR J := 1 STEP 1 UNTIL 10 DO A[I,J] );
```

13-4-2 PARAMETRIC DEFINE

Macro-instructions can be implemented by means of a parametric define.

```
DEFINE WD(WD1,WD2)=WD1:=WD1+WD2*WD2#;
```

Up to 9 parameters are allowed. The defined Macro is invoked using actual parameters in exactly the same way that a procedure is invoked. However, parametric defines cause code to be inserted in-line at every point of invocation. For this reason a procedure is preferred if the procedure body is long and the procedure is called from several places within the program.

EXAMPLE

```
DEFINE CUBERT(X) = SIGN(X)*ABS(X)**.333333333333#;
```

Conveniently provides a cube root function.

14-0 EXTENDED ARITHMETIC CAPACITY

The B6700 has hardware for double precision arithmetic and the ALGOL compiler includes many intrinsic functions which are not part of the specifications for ALGOL 60. Double precision variables are declared using the reserved word DOUBLE.

```
DOUBLE U,V,W;
```

They provide about 22 decimal digits and a number range of 1.56×10^{29604} to 2.42×10^{-29578}. If double precision variables are mixed with single precision, then the single precision quantities are extended with zeroes, and the result is double. The only exception is the case <real>** <double> or <integer>**<double> for which the result is single.

A special operator MUX multiplies two single precision quantities and produces a double precision result.

```
I:= B+C MUX(M-N);
```

Double precision numbers can be input or output by means of the D format specification.

```
FORMAT F (2D40.20);
READ (C,F,X,Y);
```

On input, the D format is rigid like the E format for single precision. A decimal point and the correct number of digits must be present. The letter D signifies the exponent which consists of a sign and 5 digits.

EXAMPLE

4.35718218218218218218D+00001

Conversion between double and single is handled through the following intrinsic functions:

REAL (D)	returns D normalized and rounded to single
SINGLE (D)	returns D normalized and truncated to single
FIRSTWORD (D)	returns the first word of D unchanged
SECONDWORD (D)	returns the second word of D unchanged
DOUBLE (R)	converts R to double
DOUBLE (R1,R2)	forms a double precision operand whose first word is R1 and whose second word is R2.

The internal representation of double precision quantities is shown below

The first word looks exactly like a single precision number. From the left the bit fields are:

1 bit	not used
1 bit	sign of number
1 bit	sign of mantissa exponent
6 bits	low-order bits of exponent
39 bits	high order bits of mantissa

The second word has two fields.

9 bits	high order bits of exponent
39 bits	low order bits of mantissa

The exponent is a power of 8, and the mantissa is best thought of asan octal number rather than a binary number. The "octal point" lies between the two words so that a 78 bit integer is represented as a floating point number with an exponent of 15(8).

14-1 INTRINSIC FUNCTIONS

The intrinsic functions of B6700 ALGOL which are not included in ALGOL 60 are:

DABS(D)	double precision absolute value
NABS(X)	negative of absolute value
DNABS(X)	double precision NABS
FIRSTONE(X)	one plus bit number of left-most non-zero bit of X
DARCTAN(D)	double precision ARCTAN
ARCCOS(X)	cosine inverse
ARCSIN(K)	sine inverse
TAN(X)	tangent
COTAN (X)	cotangent
DSIN(D)	double SIN
DCOS(D)	double COS
COSH(X)	hyperbolic cosine

SINH(X)	hyperbolic sine
TANH(X)	hyperbolic tangent
LOG(X)	log to base 10
DLOG(D)	double LOG
DLN(D)	double LN
DEXP(D)	double EXP
ERF(X)	error function
GAMMA(X)	gamma function
LNGAMMA(X)	natural log of gamma
DSQRT(D)	double SQRT
ARCTAN2(X,Y)	arctangent of X/Y
DARCTAN(D1,D2)	double ARCTAN2
MAX(A1,...,An)	maximum
MIN(A1,...,An)	minimum
DMAX(D1,...,Dn)	double MAX
DMIN(D1...,Dn)	double MIN
INTEGERT(X)	integerize X by truncation
DINTEGER(D)	same as ENTIER(D+.5)

15-0 FORWARD DECLARATIONS

The program used to translate Burroughs ALGOL programs into machine code which can be executed by the B6700 is a one-pass compiler. This means that the definition of each identifier must precede any other occurrence of that identifier. Therefore, it becomes necessary to provide for cases such as the following recursive procedure declarations.

```
PROCEDURE C(N); INTEGER N;
IF N=0 THEN B(N) ELSE <statement>;
PROCEDURE B(K); INTEGER K;
IF K NEQ 0 THEN C(K) ELSE <statement1>;
```

The point is that each procedure declaration refers to the other, so neither may come first without causing an error. To get out of the bind we precede the declaration of procedure C above with:

```
PROCEDURE B(K); INTEGER K;
FORWARD;
```

The procedure heading appears in full, but the body is replaced by the reserved word FORWARD. No action is taken, but now a reference to the procedure B may occur before B is actually declared. Notice that no FORWARD declaration is required for procedure C since the first reference to it is its declaration.

The same problem arises when two switches refer to each other.

```
SWITCH S1:= L1,L2,L3,S2[I];
SWITCH S2:=S1[J],L4,L5,L6;
```

In order to avoid using S2 before it is declared, we precede both declarations with

```
SWITCH S2 FORWARD;
```

This informs the compiler that S2 is to be used as a switch before it is actually defined.

EXERCISE

Determine the value of A(2), A(3), and A(4)

```
INTEGER PROCEDURE B(N); INTEGER N; FORWARD;
INTEGER PROCEDURE A(N); INTEGER N;
     A := IF N < 0 THEN 0 ELSE B(N)-N ;
INTEGER PROCEDURE B(N); INTEGER N;
     B := IF N < A(N-1) THEN 1 ELSE 0;
```

16-0 THE B6700

Several of the Burroughs extensions are intimately related to the machine itself. Therefore, we must begin with a description of the B6700. Basically it consists of 2 processors, core memory, disk memory, and various I/O devices. The core memory consists of 262,144 48-bit words, and the disk contains an additional 20 million words. Disk memory is used by the system as the primary storage area for programs and data. Core memory is used only for pieces of programs and data which are currently active. Thus, for example, it is permissable to declare an array whose total size far exceeds the capacity of core-memory. The declaration

```
ARRAY A[0:9,0:9,0:999]
```

reserves 100,000 words of disk memory structured as 100 rows of 1000 words each. The 1000 words of each row are stored consecutively on the disk, but the rows themselves are not necessarily consecutive. Additional words of core memory contain the disk addresses of the array rows. If we reference A[I,J,K] then the following actions take place: If the row referred to by I and J is not currently in core-memory, then it is read in from the disk. This process usually means that some other data read in previously must first be written from core to disk to make room. Finally, the element in position K of row (I,J) is accessed.

Just as arrays are divided into pieces called rows, programs are divided into pieces called segments. Only the segment containing code currently being executed needs to be in core memory. The rest of the program is stored on the disk. Thus programs containing more instructions than core memory holds can neverless be accommodated.

16-1 NUMBER REPRESENTATION

Numbers are represented in binary or base 2. Only the digits 0 and 1 are used, and place holders represent powers of 2 instead of powers of 10 as in the familiar decimal notation. For example, in binary the number 8 is written 100, the number 13 is wirtten 1101, and the number 16.25 is written 10000.01. The number 1/10 does not have an exact representation in binary just as the number 1/3 has no exact representation in decimal.

The 48 bits of each memory word are referenced by a number. The right-most bit is bit 0 and the left-most bit is bit 47.

Floating point numbers have the following format:

BIT	USE
47	Not used
46	Sign of number 0=+,1=-
45	Sign of exponent 0=+,1=-
44-39	Exponent
38-0	Mantissa

The binary point is assumed to be to the right of bit 0, thus an integer is merely a floating point number with 0 exponent, and the largest possible positive integer is $2^{39}-1$. The exponent part is interpreted as a power of 8. A floating point number is said to be normalized if the high three bits of the mantissa are not all zero. This means that a normalized floating point number represents a real number to the maximum precision possible in one machine word. The largest exponent possible is 63, so the range of the positive normalized floating point numbers is $(2^{39}-1) \times 8^{63}$ to $2^{36} \times 8^{-63}$.

EXERCISES

1 - Verify that the number range cited above is the same as that given in Chapter One.

2 - Octal notation is base 8. Verify that we can convert a binary integer to octal by simply grouping the digits by threes starting from the right.

3 - Write each of the following binary integers in octal and decimal.

A 1101010	B 10101010101	C 100001
D 1000101	E 111111	F 1000001

4 - Write each of the following octal integers in binary and decimal.

A 152	B 2525	C 41
D 105	E 77	F 101

5 - We can convert integers in base R (like 10) to integers in base S (like 8) by successive division by S using base R arithmetic. The remainders are the digits in base S.

```
123 (10) = 173 (8)
123 = 8x15  +3
 15 = 8x1   +7
  1 = 8x0   +1
```

Write the following base 10 integers in base 8.

A 10	B 100	C 1000
D 16	E 128	F 1024
G 1023	H 999	

17-0 PARTIAL-WORD OPERATIONS

Groups of consecutive bits (sometimes called fields) within a word may be referenced as follows:

```
Z:= X.[38:39];
```

The number 38 refers to the bit number of the left-most bit of the field and the number 39 is the number of bits. Thus, if X is a floating point number, this statement assigns the mantissa portion to the variable Z. The numbers within the brackets may be arbitrary arithmetic expressions. The left side of the colon must not exceed 47. If the right side exceeds the left side plus one, then the field is completed from bit 47 as if the word were circular. Neither field may assume a negative value.

Since the result of evaluating an arithmetic expression is contained in a single computer word, it makes sense to reference a field within an arithmetic expression. For example, the value of

```
(IF 0<1 THEN 7 ELSE 5).[2:2]
```

is 3.

Partial word designations may also appear on the left-hand side of replacement operators. In this case, only the bits referenced are affected. If the bit field on the left is shorter than the bit field on the right, then high order bits are discarded. The "wrap-around effect" also holds for this situation, thus X[1:4] consists of bits 1,0,47, and 46 of the variable X.

EXERCISES

Let X=32767, Y=2, and Z=4 all represented as integers. Compute the value of each of the following expressions.

1 - X.[5:6]
2 - IF Z.[3:2]=Y THEN 47 ELSE 23
3 - ((X+1)*Y*Z).[23:6]

17-1 CONCATENATION

A second way of manipulating partial words is by means of the concatenation operator. The value of the expression

```
X & Y[LBT:LBF:N]
```

is computed by moving an N-bit field from Y into the value of X. The left-most bit of the field taken from Y is the bit numbered LBF. The left-most bit of the field to receive these bits is numbered LBT. Like the partial word designator, the bit numbers may be arithmetic expressions. Again the values of LBT and LBF must not exceed 47 and "wrap-around" at bit 47 occurs if N is larger than LBT - or LBF plus one. LBT is referred to as the "left-bit-to", LBF as the "left-bit from" and N as the "number-of-bits". Note that concatenation, like the other arithmetic operators, does not change the values of the operands. It merely uses these values to build a new quantity.

An alternate form of the concatenation operator is

```
X & Y[J:K]
```

In this case, J is the left-bit-to, K is the number-of-bits, and K-1 is the left-bit-from, i.e. the low-order K bits of Y are moved into the bit-field specified by [J:K].

The two operands involved in a concatenation may be variables or arithmetic or Boolean expressions. If the left hand side of the ampersand is an expression, then it must be parenthesized. See the examples below. The result of a concatenation is Boolean if the operands are Boolean. It has type real if the operands are real or integer. Successive concatenations are performed from left to right.

EXAMPLES

Z:=0 & Y[38:38:39]; has the same effect as
Z:=Y.[38:39];

The value of 2**8 & 15[2:3:3] is 2**15

The value of (2**8) & 15[2:3:3] is 2**8 + 7

EXERCISES

Let X=32767 Y=1024 and Z=1
Determine the value of each of the following expressions.

```
1 - Y & (2*Z)[11:1:2]
2 - Y & Z[9:0:1] & X[3:13:4]
3 - Z & Y[40:10:2] & Z[45:0:1]
4 - X & Z[47:0:1]
5 - Y &(2*Z)[11:1:2] +5
6 - Y & (4*Z+1)[9:6:7] & X[14:14:15]
```

17-2 TYPE TRANSFER FUNCTIONS AND MASKING

The value of Boolean variables is determined by bit zero only. If this bit is one, the value is true; if it is zero, the value is false. Boolean operations, however, are performed on full words in a bit by bit manner according to the table below.

X	Y	NOT X	X AND Y	X OR Y	X IMP Y	X EQV Y
0	0	1	0	0	1	1
0	1	1	0	1	1	0
1	0	0	0	1	0	0
1	1	0	1	1	1	1

In order to make use of these properties, a method of permitting Boolean operations on real or integer variables is required. The type transfer functions provide this facility. If X is declared real and B is declared Boolean, we might have the following statements.

```
B:= BOOLEAN(X);
X:= REAL(B);
```

These functions do not change their arguments at all. They simply allow a type change. There is also a function for converting real to integer.

INTEGER(R) is equivalent to ENTIER(R+.5)

EXAMPLES

REAL(NOT FALSE) = 7777777777777777 (octal)

REAL (TRUE) = 0000000000000001 (octal)

EXERCISES

Find the values of the following expressions

1 - REAL(BOOLEAN(5) OR BOOLEAN(2))
2 - REAL(BOOLEAN(REAL(NOT FALSE) & 0 [47:47:9]) AND BOOLEAN(X))= X. [38:39]
3 - REAL(BOOLEAN(31) OR BOOLEAN(32760))
4 - If the expression

REAL(BOOLEAN(X) EQV BOOLEAN(Y)) = REAL(NOT FALSE)

is true, what can be said about X and Y?
How does this expression differ from X = Y?

A seventh relational operater IS does a bit by bit comparison of two words. The expression

X IS Y

is true if X and Y have exactly the same bit pattern.

18-0 STRING MANIPULATION

The B6700 has circuits designed specifically for handling character strings, and ALGOL contains many constructs for making use of this hardware.

Each of the 48-bit words of the machine may be thought of as consisting of six 8-bit characters. Since there are 256 possible patterns of 8 bits, there are 256 possible characters. Table 1 in the appendix shows which character is represented by each bit pattern. Only 64 of these characters are available on the printer. All the others print as question marks.

This set of 8-bit characters, known as the EBCDIC set, is the standard internal representation of characters used by the B6700. However, there are facilities for handling 6-bit (BCL) strings, 4-bit (HEXADECIMAL) strings, 3-bit (OCTAL) strings, 2-bit (QUATERNARY) strings, and 1-bit (BINARY) strings. Tables 1, 2, and 3 show the character sets and the representative bit patterns for BCL, HEX, and OCTAL characters.

Strings are stored in memory as array rows. For example if following the declaration

```
ARRAY A[0:3,0:4];
```

we were to fill row 2 (denoted A[2,*]) with

```
"BURROUGHS ARE VEGETARIANS"
```

then, referring to Tables 1 and 2, we see that the contents of this row, written in binary notation, would be:

```
A[2,0]=  11000010  11100100  11011001  11011001  11010110  11100100
A[2,1]=  11000111  11001000  11100010  01000000  11000001  11011001
A[2,2]=  11000101  01000000  11100101  11000101  11000111  11000101
A[2,3]=  11100011  11000001  11011001  11001001  11000001  11010101
A[2,4]=  11100010  01000000  01000000  01000000  01000000  01000000
```

18-1 FILL STATEMENT

Data may be entered into an array row by means of a statement like

```
FILL B[*] WITH 14.2,.175@-1,27,3"77777","HARVEY",
   6"BCLSTR", 101(0);
```

The words FILL and WITH are reserved words. One element of the specified row is assigned a value for each member of the list. Type is not considered. Each list element is simply converted to binary form and stored. If a list element is

a string containing fewer than 48 bits then the corresponding word is completed with high-order zeroes. The number of bits per character is specified explicitly in front of strings other than EBCDIC strings. Any list entry or group of entries may be enclosed in parenthesis and preceded by a repeat factor.

Character strings need not be broken into six character pieces.

```
FILL A[2,*] WITH "BURROUGHS ARE VEGETARIANS";
```

18-2 ADDITIONAL IO FACILITIES

Character strings can be read into array rows by means of the statement

```
READ(CARDS,14,C[*]);
```

Any arithmetic expression may appear in place of the format identifier. Its value is rounded to the nearest integer, and the result is the number of words to be read into the specified array row. In this example, if the identifier CARDS refers to a card reader, then 14 words exhausts the 80 columns of a singe data card. If the number of words read exceeds the length of the array row, then the excess data is lost.

Strings can be output similarly as follows:

```
WRITE(PRINT,22,OUTBUF[*]);
```

Additional format specifications are also available for character handling. Note that formats can be placed directly inside the WRITE statement, in which case no declaration is necessary. This holds for the format specifications discussed in Chapter two as well as those included below.

18-2-1 A FORMAT

The Aω format treats a word as ω 8-bit characters right justified. Since 6 characters fill a word, the value of ω should not exceed 6. The specification A10 is equivalent to X4,A6 for both READ and WRITE statements. If ω is smaller than 6 then the high order bits of the word are set to zero.

EXAMPLE

If a data card has the letters A through Z punched in columns 1 through 26 then

```
READ(C,<A6,A10,A2>,X,Y,Z);
WRITE(P,<A6,A10,I15>,X,Y,Z);   produces
```

```
ABCDEF    KLMNOP          55513
```

18-2-2 C FORMAT

The $C\omega$ format treats a word as ω 8-bit characters left justified. The value of ω should not exceed six, and if it is less than 6 on input, then the low-order characters are filled with blanks.

EXAMPLE

If a data card has the letters A through Z punched in columns 1 through 26 then

```
     READ(C,<C10,C4,C6>,X,Y,Z);
     WRITE(P,<3A6>,X,Y,Z);
     WRITE(P,<C2,C3,C4>,X,Y,Z);  produces
EFGHIJKLMN  OPQRST
EFKLMOPQR
```

18-2-3 O FORMAT

The O specification is equivalent to either A6 or C6.

18-2-4 H AND K FORMATS

The $H\omega$ format treats the contents of a word as ω hexadecimal characters right justified. Since hexadecimal characters require 4 bits, the value of ω need not exceed 12, and the specification H20 is equivalent to X8,H12.

The $K\omega$ format treats the contents of a word as ω octal characters right justified. This time 16 characters are contained in one word.

EXAMPLE

If X has the value 32767 then

```
          WRITE (P,<H2,H8,K2,K7>,X,X,X,X);  produces

FF00007FFF770077777
```

19-0 POINTERS

Individual characters within a string are referenced by means of pointers. Like variables, pointers must be declared.

```
POINTER P,Q,R;
```

The word POINTER is a reserved word. The "values" that pointers assume are character positions within an array row, and these values may be assigned or changed by pointer assignment statements and expressions.

```
P:= POINTER(A);
```

Makes P refer to the first character in the array row A[*].

```
Q:=POINTER(A[1]);
```

Makes Q refer to the left-most character in the word A[1]. If the declared lower bound of the array A is 1, then P and Q have the same value. Offsets are specified by adding or subtracting the absolute value of a variable,constant, or parenthesized arithmetic expression.

```
R:=POINTER(A)+12;
S := R -(3-5);
```

causes R to refer to the 13th character of the string and S to the 11th character. Note that the left-most operator determines the direction of the offset. The pointer is advanced if it is a plus and moved backwards if it is a minus regardless of the sign of the arithmetic expression that follows.

Ordinarily, strings are assumed to consist of 8-bit characters. If 4 or 6 bit character size is required, then the size must be given explicitly.

```
P:=POINTER(A,6);
Q:=POINTER(B [2],4);
```

P now refers to the left most character of the array row A[*], and A[*]is given as containing 6-bit characters. Q points to the left most character of B[2], and the array row B[*]will be treated as a string of hexadecimal digits.

EXAMPLES

```
P:=IF N<80 THEN POINTER(A)+N ELSE P+1;
Q:=P+5;
R:=IF K GEQ 119 THEN P ELSE Q;
S:=CASE I OF (P,Q,R+2,POINTER(AA,6));
```

19-1 TRANSFER FUNCTIONS FOR POINTERS

To recover the actual character referenced by a pointer, the type transfer functions REAL(P,N) and INTEGER(P,N) are available. The quantity returned by the function REAL(P,N) is a variable of type REAL whose value is just the N characters referred to by the pointer P right justified with high order zeroes.

The function INTEGER(P,N) returns the integer value represented by the first N characters referenced by the pointer P. If these characters are not decimal digits, the result bears a tenuous relation to the original character string since the conversion relies on the fact that the numeric values of the four low order bits of the bit patterns used to represent the decimal digits coincide with the digits themselves. For example, the character "8" is represented by the binary sequence 11111000.

The function DELTA(P,Q) yields an integer result whose value is the distance between the two pointers P and Q. That is, P+DELTA(P,Q) and Q have the same value so long as P and Q refer to the same array row and Q points to the right of P. If the pointers do not refer to the same array row, then the value of the function is zero. If Q precedes P the value of DELTA(P,Q) is negative.

EXAMPLES

Let P refer to the first character of "01234ABC"
Then REAL(P,4) is "0123"
REAL(P,6) is "01234A"
INTEGER(P,4) is 123

19-2 POINTER LEVELS

Consider the following program section:

```
POINTER P;
<statements>
     BEGIN
     ARRAY W[0:2,0:16];
     P:=POINTER(W[0,*]);
     <statements>
     END;
```

Upon exit from the block, the array W disappears, and the pointer, being global to the block, simply loses its "raison d'etre." In order to avoid this situation, a lex level is associated with every array and every pointer depending on the nesting level of the block in which these entities are declared. The outer block is arbitrarily assigned level 2. Each time the BEGIN at the head of a block is encountered, the lex level is increased by one. It is decreased by one when the corresponding END is reached. The level of the right hand side of any pointer assignment should never exceed the level of the left hand side.

20-0 SCAN

The SCAN statement is used to examine strings for the presence or absence of various conditions.

```
SCAN P:Q FOR K:80 UNTIL NEQ " ";
```

The word SCAN is a reserved word, and the following action is taken: The pointer P is set equal to the pointer Q and the variable K is set to 80. Then the character referenced by P is examined. If it is not a blank, then action terminates. If the character is a blank, then P is incremented by 1, K is decremented by 1 and the examination step is repeated. If the condition is not met after 80 characters have been examined, then action terminates regardless. If the condition was never met, then a reserved Boolean variable called TOGGLE is set to TRUE. Otherwise it is set to FALSE.

Considerable generalization of the above example is possible. The maximum count can be any arithmetic expression and the initial pointer Q can be any pointer expression. Neither P: nor K: are necessary although no information about the string is gained if both are omitted except through TOGGLE. The condition can take a variety of forms. Any character may be used instead of a blank. An arithmetic expression can also be used, in which case the lower eight bits of the result are taken as the character. Any of the six relational operators is permissable. The order of the characters for this purpose is given by the numeric values of the bit patterns representing the characters. The word WHILE may replace UNTIL with the obvious meaning. The condition may also take either of the forms

```
WHILE IN ALPHA
UNTIL IN ALPHA
```

A character satisfies the condition IN ALPHA if it is a letter or a digit. The words IN and ALPHA are reserved words. The maximum count may also be omitted entirely in which case it is assumed to be 1048575.

Note that in the example above, 80-K is the number of characters scanned before the condition was met, and the pointer P refers to the character that met the condition. If the condition were not met K would have the value 0 and P would be pointing to the 81st character in the string. The pointer P is called an update pointer, and the variable K is called an update count.

Although a pointer is allowed to point off the end of an array row, the entity (if any) that it points to may not be referenced. For example, if the row associated with P has only 80 characters, no error is produced by the SCAN statement above, but another SCAN statement requiring that the character now referenced by P be examined would produce an error. This error will occur even if the maximum count specified is zero.

The condition IN ALPHA can be generalized to any predetermined subset of characters by means of a table membership construct. The statement

```
SCAN P:P FOR K:K UNTIL IN T[J];
```

is an example of a SCAN statement with table membership condition. The table consists of the 32 low-order bits of the eight words T[J] through T[J+7]. Membership in the table is determined as follows: The three high-order bits of the character are added to the index J to specify one of the eight words. The five low-order bits of the character are used to index one of the 32 low-order bits of the table word. If this bit is a one, then the character is part of the subset; otherwise, it is not. The low-order 32 bits are indexed for this purpose from left to right. Bit 31 has index 0, bit 30 has index 1, etc.

EXAMPLE

If T[8]=0000017400000000(8) and the table begins at T[2], then referring to table 1 in the appendix, the characters A,B,C,D,E are in the subset, but F,G,H,I,J (and others) are not in the subset.

20-1-1 REPLACE

The REPLACE statement allows us to move a character string from one array row into another. One form works exactly like the SCAN statement except that the characters scanned are also copied into a second array row.

```
REPLACE P:P1 BY Q:Q1 FOR M:N UNTIL LEQ "0";
```

The words REPLACE and BY are reserved words. The array row referenced by Q1 is the source array and the row referred to by P1 is the destination. For example, let P1 refer to the first character of

ABCDEFGHIJKLMNOPQRSTUVWXYZ

Let Q1 refer to the first character of

1234567890 ABCDEFGHIJKLMNOPQRSTUVWXYZ

and let N have the value 15. At the termination of the statement above, the destination array is

123456789JKLMNOPQRSTUVWXYZ

and the source array is unchanged. P refers to the character J, Q refers to the character O and M has the value 6. P1, Q1, and N are, of course, unchanged.

Any condition part valid for the SCAN statement can be used in a REPLACE statement. The TOGGLE variable also applies.

Other forms not possible in the SCAN statement can also be used.

(1) No condition part is required at all.

```
REPLACE P1 by Q1 FOR 80;
```

simply moves 80 characters from the array pointed to by Q1 into the array pointed to by P1.

(2) The source array may be a string explicitly contained within the REPLACE statement.

```
REPLACE P1 BY "SIXTY FIVE HUNDRED";
```

moves the 18 characters of the string into the array referenced by P1.

(3) A count may be appended to an explicit string.

```
REPLACE PNTR BY "MELVIN COWZNUFSKI" FOR N ;
```

If N is larger than the number of characters in the string, an error results unless the string is less than six characters long. Otherwise, the first N characters of the string are moved.

Strings of less than six characters receive special treatment. In this case the string is cyclically extended to six characters and the result is repeated until N characters have been moved. For example,

```
REPLACE PR BY ".A.." FOR 15;
```

The 15 characters of the array pointed to by PR now contain

```
.A...A.A...A.A.
```

(4) An arithmetic expression may be used instead of the string. In this case the value is treated as if it were a six character string.

(5) An arithmetic expression can be converted to a character string by

```
REPLACE P BY N FOR K DIGITS;
```

If more than K characters are required to represent the number N, then the right-most K digits are used.

(6) The number of characters to be moved can be specified in 48-bit words instead of 8-bit charcters.

```
REPLACE P1 BY Q1 FOR N WORDS;
```

This statement is equivalent to

```
REPLACE P1 by Q1 FOR 8*N;
```

except that the pointers are first advanced to the beginning of the next word if they are not already pointing to the first character of a word.

(7) Several different sources, separated by commas, can be used for a single destination and each may assume any of the forms discussed above.

```
REPLACE P BY "DEAR", NAMEPOINTER FOR N,":"," " FOR 84-N;
```

As with the SCAN statement, an error occurs if either the destination or source pointer is off the end of its respective row even though the REPLACE statement has a maximum count of zero.

20-1-2 PICTURES

It is possible to associate a format with a REPLACE statement by use of a PICTURE declaration. However, there is no connection between the editing specifications used with pictures and those used for formatted READ or WRITE. The declaration

```
PICTURE PIC (A(10));
```

associates a picture format with the identifier PIC. The statement

```
REPLACE P BY Q WITH PIC;
```

is then equivalent to

```
REPLACE P BY Q FOR 10;
```

since the picture specification A(10) means copy 10 characters.

Some of the other picture format specifications are:

"<string>"	The string is moved into the destination array
X(w)	The destination pointer is moved forward w.
>(w)	The source pointer is moved forward w.
<(w)	The source pointer is moved backward w.
I(w)	Move w periods into the destination array.
NA	Change the character referred to by the I format from period to the character A. Any character can be used instead of period or A.

EXAMPLE

Let P refer to the beginning of

ABCDEFGHIJKLMNOPQRSTUVWXYZ

and Q to an array of blanks. Then following the declaration

```
PICTURE PF("FIRST COMES"X(1)A(1)>(10)X(1)"LATER IS"
   X(1)A(1)I(3)>(11)A(3));
```

the statement

```
REPLACE Q BY P WITH PF
```

fills the blank array with

```
FIRST COMES A LATER IS L...XYZ
```

20-2 BOOLEAN EXPRESSIONS

There are three forms of Boolean expressions for use with character manipulation. The expression

<AE> IN ALPHA

is true if the low-order 8 bits of the arithmetic expression represent a letter or a digit. The more general table membership described in section 20-0 can also be used here.

Two strings can be compared in any of three ways.

P:P1 = Q:Q1 FOR K

is true if each of the first K characters in the row referenced by P1 matches the corresponding characters in the row referenced by Q1. The update pointers P and Q are optional. Any of the six relational operators may be used, the order relation being that induced by the numeric values of the characters.

The expression

P = "<string>"

is true if the characters of the array row pointed to by P are those of the string.

The expression

P= "<string>" FOR K

is handled in the same way as the corresponding phrase in the REPLACE statement. If the string has six or more characters, then the expression is true if the first K characters of the array row referenced by P match the first K characters of the string. In this case the value of K must not exceed the length of the string.

If the string has less than six characters, then it is concatenated with itself until it is six characters long and the resulting character string is concatenated with itself until it is at least K long. This string is then compared with the first K characters of the row referenced by the pointer P.

EXERCISES

Suppose an array A contains the string

ABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890

and a pointer Q refers to the first character. Determine what value K has and what character P referes to as a result of each of the following statements. Use table 1 if necessary.

```
1 - SCAN P:Q FOR K:36 UNTIL = "A";
2 - SCAN P:Q FOR K:36 WHILE = "A";
3 - SCAN Q FOR K:36 WHILE LEQ "Z";
4 - SCAN P:Q FOR 36 UNTIL > 216;
5 - SCAN P:Q FOR K:11 WHILE IN ALPHA;
```

Let P refer to the beginning of an eight word array and determine the contents of that array after each of the following statements has been executed successively.

```
 6 - REPLACE P BY "#" FOR 8 WORDS;
 7 - REPLACE P+18 BY "01234567890";
 8 - REPLACE P BY INTEGER(P+20,4) FOR 4 DIGITS;
 9 - REPLACE P + 4 BY " " FOR 4,"ABC","A" FOR 6 WORDS;
10 - WRITE A PROCEDURE THAT COUNTS THE NUMBER OF TIMES
     THE WORD "THE" APPEARS IN A STRING OF 8184 CHARACTERS.
```

21-0 TIME FUNCTION

TIME(N) is a function which returns the following results :

TIME(0) returns the current date in bcl characters (in the format 6"YYDDD" where YY is the year and DDD is the day of the year), e.g. 29 May 43 is 6"43149000".

TIME(1) Returns as an integer value the time of day, in sixtieths of a second, e.g. 1:25 PM is 2898000.

TIME(2) Returns as an integer value the elapsed processor time of the job, in sixtieths of a second.

TIME(3) Returns as an integer value the elapsed I/O time of the job, in sixtieths of a second.

TIME(4) Returns as an integer value the contents of a six-bit machine clock which increments every sixtieth of a second, e.g. varying between 0 and 63.

TIME(5) Returns month, day, year as six BCL characters right justified in the format 6"00MMDDYY".

TIME(10) Same as TIME(0), except time is expressed in EBCDIC characters in the format 8"YYDDD".

TIME(11) Same as TIME(1), except time is expressed in multiples 2.4 microseconds.

TIME(12) Same as TIME(2), except time is expressed in multiples of 2.4 microseconds.

TIME(13) Same as TIME(3), except time is expressed in multiples of 2.4 microseconds.

TIME(14) Returns as an integer value the elapsed time since the last halt/load in multiples of 2.4 microseconds.

TIME(15) Current date in the format 8"MMDDYY".

COMPILETIME(AE) Makes various system time values available at compile-time (as opposed to run-time). The form of the value returned by the compiletime intrinsic is the same as the time intrinsic for the same argument. The returned value is computed by the compiler using the time intrinsic at compile-time. The argument must be a constant.

APPENDIX

The reserved words

ALPHA	AND	ARRAY
BEGIN	BOOLEAN	CLOSE
COMMENT		DEFINE
DIV	DO	DOUBLE
EQV	ELSE	END
FILL	FALSE	FILE
FORWARD	FOR	FORMAT
IMP	GO	IF
LABEL	IN	INTEGER
MOD	LIST	
OR	MONITOR	NOT
REAL	OUT	OWN
READ	PROCEDURE	
REWIND	SAVE	RELEASE
STEP	TO	SPACE
THEN	SWITCH	
VALUE	TRUE	UNTIL
WRITE	WHILE	WITH
POINTER		

TABLE 1 EBCDIC codes and BCL codes

Graphic	EBCDIC code (hex)	BCL code (octal)	Graphic	EBCDIC code (hex)	BCL code (octal)
A	C1	21	blank	40	60
B	C2	22	[	4A	33
C	C3	23	.	4B	32
D	C4	24	<	4C	36
E	C5	25	(	4D	35
F	C6	26	+	4E	20
G	C7	27	|	4F	none
H	C8	30	←	none	37
I	C9	31	&	50	34
J	D1	41	]	5A	76
K	D2	42	$	5B	52
L	D3	43	*	5C	53
M	D4	44	)	5D	55
N	D5	45	;	5E	56
O	D6	46	¬	5F	none
P	D7	47	≤	none	57
Q	D8	50	-	60	54
R	D9	51	/	61	61
S	E2	62	,	6B	72
T	E3	63	%	6C	73
U	E4	64	—	6D	none
V	E5	65	≠	none	74
W	E6	66	>	6E	16
X	E7	67	?	6F	14
Y	E8	70	:	7A	15
Z	E9	71	#	7B	12
0	F0	00	@	7C	13
1	F1	01	'	7D	none
2	F2	02	≥	none	17
3	F3	03	=	7E	75
4	F4	04	"	7F	77
5	F5	05	⊗	none	40
6	F6	06	!	D0	none
7	F7	07			
8	F8	10			
9	F9	11			

TABLE 2 HEXADECIMAL CODES

HEX digit	binary equivalent
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
A	1010
B	1011
C	1100
D	1101
E	1110
F	1111

TABLE 3 OCTAL CODES

OCTAL digit	binary equivalent
0	000
1	001
2	010
3	011
4	100
5	101
6	110
7	111

ANSWERS TO SELECTED EXERCISES

1-1-1
(A) No commas allowed
(B) OK
(C) Cannot have two signs
(D) Too large
(E) This is two numbers
(F) OK
(G) No parentheses allowed

1-3-1
(A) Too large
(B) OK
(C) OK
(D) OK
(E) E should be @

1-3-2
(A) OK
(B) OK
(C) Must start with a letter
(D) Only letters and digits allowed
(E) OK
(F) OK
(G) OK

2-0-1
(A) 2.66666666666
(B) 3.5
(C) 14.125
(D) 19683
(E) 7625597484987
or 7.62559748499 @12

2-0-2

(A) Missing *
(B) Missing *
(C) Extra)
(D) [not allowed
(E) *- not correct, should be *(-

2-0-3

With parentheses 2 multiplications and 3 adds
Without parentheses 5 multiplications and 3 adds
(X**3 counts as two multiplications)

2-4-1

```
If X GTR Y THEN WRITE(PRINTER,/,X) ELSE WRITE(PRINTER,/,Y)
```

2-5-2

```
BEGIN
FILE PRINTER(KIND=PRINTER, MAXRECSIZE=22);
INTEGER NF,I;  LABEL B1;
NF :=1;  I := 1;
B1: NF := NF * I;
I := I + 1;  IF I LEQ N THEN GO TO B1;
WRITE(PRINTER,/,NF);
END.
```

(Notice that more than one statement may appear on one card.)

2-5-3

```
BEGIN
FILE PRINTER(KIND=PRINTER, MAXRECSIZE=22);
REAL C;  INTEGER I;
C := 1 ;
FOR I := 1 STEP 1 UNTIL K DO
C :=C * (N-I+1)/I ;
WRITE(PRINTER,/,C);END.
```

(Why is this program better than a more straight forward one?)

2-5-4

The Fibonacci sequence is 1,1,2,3,5,8,13,21,...
Each term is the sum of the two preceeding terms.(except 1,1)
The program is left to the reader.

2-7-1

```
BEGIN
FILE PRINTER(KIND= PRINTER, MAXRECSIZE=22);
REAL X,Y;
LABEL MORE,ENUF;
X := 1;
MORE:  IF X GTR 100 THEN GO TO ENUF;
Y := SQRT(X);  WRITE(PRINTER,/,X,Y);
X := X + 1; GO TO MORE;
ENUF: END.
```

```
2-8-1
BEGIN
FILE PRINTER(KIND=PRINTER, MAXRECSIZE=22);
INTEGER M,N,PENNIES,NICKELS,DIMES,QUARTERS;
LABEL FINISH,MORE;
N := 0;
MORE: N := N + 7;  IF N GTR 99 THEN GO TO FINISH;
QUARTERS := N DIV 25;
M :=N MOD 25;
DIMES := M DIV 10;
M := M MOD 10;
NICKELS := M DIV 5;
PENNIES := M MOD 5;
WRITE(PRINTER,/,N,QUARTERS,DIMES,NICKELS,PENNIES);
GO TO MORE;
FINISH: END.
```

```
2-9-1
SET N EQUAL TO 1000
FOR I := 2 STEP 1 UNTIL N IF A[I] > A[I-1] THEN EXCHANGE THEM
IF THERE WERE NO EXCHANGES, WE ARE DONE
OTHERWISE, SET N = N-1 AND REPEAT
THIS IS CALLED A BUBBLE SORT.
THE PROGRAM ITSELF IS LEFT TO THE READER
```

```
2-9-2
BEGIN
FILE PRINTER(KIND=PRINTER, MAXRECSIZE=22);
INTEGER N; INTEGER ARRAY POWER[0:38];
POWER[0]:=1;
FOR N:=1 STEP 1 UNTIL 38 DO
POWER[N]:=2*  POWER[N-1];
FOR N:= 0 STEP 1 UNTIL 38 DO
WRITE(PRINTER,/,N, POWER[N]);
END.
```

```
2-12-1
BEGIN
FILE PRINTER (KIND=PRINTER, MAXRECSIZE=22);
INTEGER K;
INTEGER ARRAY M[1:10,1:10];
FOR K:=1 STEP 1 UNTIL 5 DO
BEGIN
INTEGER N,J;
N:=11-K;
FOR J:=K STEP 1 UNTIL N DO
    BEGIN
        M[K,J]:=K;
        M[J,K]:=K;
        M[N,J]:=K;
        M[J,N]:=K;
END;END;END.
```

2-13-1

```
REAL PROCEDURE Q(X);
REAL X;
Q := X-X*X*X/2/3+X**5/2/3/4/5-X**7/2/3/4/5/6/7;
```

2-13-2

```
BEGIN
FILE PRINTER(KIND=PRINTER, MAXRECSIZE=22);
REAL PROCEDURE Q(X);
REAL X;
Q := X-X*X*X/2/3+X**5/2/3/4/5-X**7/2/3/4/5/6/7;
REAL X,Y,Z,S;
FOR X := -1.508 STEP .02 UNTIL 1.508 DO BEGIN
Y := SIN(X): Z := Q(X); S := ABS( Z-Y );
WRITE(PRINTER,/,X,Z,Y,S) END;
END.
```

4-3

1 - FALSE
2 - TRUE
3 - FALSE
4 - TRUE

4-4

A	T	T	F	F
B	T	F	T	F
1	F	F	T	F
2	T	F	T	F
3	T	F	T	F

5-0

1 - 2
2 - 0
3 - 0
4 - IF Z*Z GTR 4*Y THEN(IF X EQL O THEN Z ELSE Y) ELSE X
5 - YES. The construct ELSE IF is all right since it does not entail the ambiguities possible with THEN IF.

7-3

1 - 5,4,3,1,0
2 - 5,8,11,14,17,20
3 - 1,2,6,42
4 - 0,1,2,3,4,5,6
5 - 0,1,2,3,4,5,6,7,9

8-2-1

```
1 - BEGIN
    FILE PRR (KIND=PRINTER, MAXRECSIZE=22);
    FORMAT F101 (I2,I2,I2,I2,I2,I2,I2);
    FORMAT F102 (I3,I2,I2,I2,I2,I2);
    FORMAT F103 (I4,I2,I2,I2,I2);
    INTEGER J;
    WRITE (PRR,F101,FOR J:=1 STEP 1 UNTIL 7 DO J);
    WRITE (PRR,F102,FOR J:=2 STEP 1 UNTIL 7 DO J);
    WRITE (PRR,F103,FOR J:=3 STEP 1 UNTIL 7 DO J);
    END.

2 - FORMAT F100 (X54,I6,X3,I6,X3,I6);
    FORMAT F100 (X54,I6,I9,I9);
```

8-2-4

```
1 - FILE PR (KIND=PRINTER, MAXRECSIZE=22);
    FORMAT F (10I13);
    FOR I=1 STEP 1 UNTIL 6 DO
    WRITE (PR,F,FOR J:=1 STEP 1 UNTIL 10 DO A[I,J]);

2 - FILE PL (KIND=PRINTER, MAXRECSIZE=22);
    FORMAT SARDINES (132I1);
    WRITE(PL,SARDINES,FOR J:=1 STEP 1 UNTIL N DO B[J]);

3 - FILE PT (KIND=PRINTER, MAXRECSIZE=22);
    FORMAT FT(10I13)
    FOR I:=1 STEP 1 UNTIL 10 DO
    WRITE (PT,FT,FOR J:=1 STEP 1 UNTIL I DO A[I,J]);
```

8-2-5

```
1 - (I5,I3,/,I3,/,I5)
2 - (I3,I1,X1,I2,I1,X1,I2,I1,X1,I2,I3,I1,X1,I2,I1,X1,I2,I1,
     X1,I2)
3 - (X15,I5,I3/I5,I3/I5,I3/ . . .
4 - (X5,"GEVALT"/"GEVALT"/"GEVALT"/I20)
5 - (I5,X3,I4//I4//I4// . .
```

8-4

```
1 - BEGIN
    FILE PR (KIND=PRINTER, MAXRECSIZE=22);
    FORMAT FV (X*,*I8);
    INTEGER ARRAY CN[0:20];
    INTEGER N,K;
    CN[0]:=1;CN[1]:=1;
    FOR N:=2 STEP 1 UNTIL 16 DO
       BEGIN
       CN[N]:=1;
       FOR K := N-1 STEP -1 UNTIL 1 DO
       CN[K]:=CN[K]+CN[K-1];
       WRITE (PR,FV,68-4*(N+1),N+1,FOR K:=0 STEP 1 UNTIL N
           DO CN[K]);
       END;
END.
```

13-3
1- 25
2- 7
3- 20
4- 7

15-0
A(0) = 0 A(1) = -1 A(2) = -2 A(3) = -3 A(4) = -4

16-1
3,4 The values represented in both problems are the same. The decimal equivalents are

A 106	B 1365	C 33
D 69	E 63	F 65

5-

A 12	B 144	C 1750
D 20	E 200	F 2000
G 1777	H 1747	

17-0
1- 63
2- 23
3- 1

17-1
1- 2048
2- 1551
3- Floating point 1/64
4- 32767
5- 2053
6- 32767

17-2
1- 7
2- True
3- 32767
4- It is possible for X=Y to be true and the more complicated expression false if either X or Y is not normalized.

20-2
1- P points to "A" K = 36
2- P points to "B" K = 35
3- P not used K = 10
4- P points to "R" K = 19
5- P points to "L" K = 0

```
 6- ################################################
 7- ##################01234567890###################
 8- 2345##############01234567890###################
 9- 2345    ABC#AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

10- INTEGER PROCEDURE CTTHES(A);  ARRAY A[0];
    BEGIN INTEGER K,N;
    LABEL OUT,LOOP; POINTER P;
    K := 8184;  P := POINTER( A[0]);  N := 0;
    LOOP:  SCAN P:P FOR K:K UNTIL = " ";
    IF K LEQ 4 THEN GO TO OUT;
    IF P = " THE " THEN N := N +1;
    K := K -1;  P := P + 1;  GO TO LOOP;
    OUT:  CTTHES := N
    END;
```

SIMPLIFIED INPUT-OUTPUT

UCSD has provided a simplified version of ALGOL Read and Write statements. These are designed to make straight forward reading of cards and printing.

The simplified read and write statements are:

```
READ, format, list;

PRINT, format, list;
```

The following sample program will read one card and print it:

```
2COMPILE AP10/SIMPLEJOB ALGOL
2DATA
BEGIN
ARRAY BUFFER[0:13];
FILE READER(KIND=READER), PRINTER(KIND=PRINTER, MAXRECSIZE=22);
COMMENT READ AND PRINT A CARD;
READ,13 BUFFER[*];
PRINT,13,BUFFER[*];
END.
2DATA
THIS IS A SAMPLE DATA CARD  ]234567890 []*/+-
2END
```

A file called READER must be declared for the READ statement.

A file called PRINTER must be declared for the PRINT statement.

Formats, an asterisk, or a word count may appear after the first comma, and any list may appear after the second comma. If the format contains only quoted strings and spacing phrases, the list may be omitted. In such a case, the second comma should be replaced by a semicolon (;). i.e.

```
FORMAT GREETING ("HI THERE");
PRINT, GREETING;
```

Formats need not be declared. They can, instead by specifiec in a WRITE or READ statement. The above example could then become:

```
PRINT, <"HI THERE">;
```

Carriage control, free format, and action labels are similar to those used in the more complex READ and WRITE statements.

A sample program to print, double-spaced, the radii and circumferences of several circles might be:

```
2COMPILE AP10/RADIUS ALGOL
2DATA
BEGIN
FILE READER(KIND=READER), PRINTER(KIND=PRINTER, MAXRECSIZE=22);
REAL CIRC, RAD;
LABEL MORE, NOMORE, PAR, DERR  EOJ;
```

```
MORE:
  READ,/,RAD[NOMORE:PAR:DERR];
  CIRC:=RAD*2*3.1415;
  PRINT,,"CIRCUMFERENCE=",CIRC,"RADIUS=",RAD;
  GO TO MORE;
DERR:
  PRINT,/,"*** A DATA ERROR HAS OCCURRED ***";
  GO EOJ;
PAR:
  PRINT,<"HAD A PARITY ERROR">;
  WRITE(PRINTER,/,"SYSTEM ERROR");
  GO EOJ;
NOMORE:
  PRINT,/,"RAN OUT OF DATA CARDS";
EOJ:
  PRINT,<"JOB FINISHED AT",I2,":",I2>, TIME(1)DIV 216000,
    TIME(1) MOD 216000 DIV 3600;
END.
?DATA



?END JOB
```

ACCURACY 51
ARITHMETIC CAPACITY 69
ARRAYS 21.22.57
BASIC STRUCTURE 14
BLOCKS 27.55
BOOLEAN ALGEBRA 34
BOOLEAN EXPRESSIONS 15.35.36.87
CALL 58
CARRIAGE CONTROL 52.99
CODES 91.92
COMPOUND STATEMENTS 26
CONCATENATION 75
CONTROL STATEMENTS 40.66
DECLARATIONS 12.27
DEFINE DECLARATIONS 68
DESIGNATIONAL EXPRESSIONS 41
EDITING PHRASES 46.49
FILL STATEMENT 78
FOR STATEMENTS 18.42.43.22
FORMAT A 79
FORMAT C 80
FORMAT H 80
FORMAT K 80
FORMAT O 80.81
FORMAT SPECIFICATIONS 46
FORMAT X 53
FORWORD DECLARATIONS 72
GLOBAL VARIABLES 58
IDENTIFIERS 8.10.11.19.56
IF STATEMENTS 15.36.37
INPUT 23.45.53. 99, 100
INTEGERS.OPERATIONS ON 20
INTRINSIC FUNCTIONS 70
IO FACILITIES 79
IO SWITCHES 64
LABELS 16.40.56
LIST STRUCTURE 64
LOCAL VARIABLES 58
MASKING 77
METALANUAGE 32
MNEMONICS 15
NUMBERS, REPRESENTATION OF 6.73.20.10
OUTPUT 11.23.45. 99, 100
OWN VARIABLES 57
PARAMETER DELIMITERS 60
PARAMETRIC DEFINE 68
PARTIAL-WORD OPERATIONS 75
PHRASE E 51.54
PHRASE F 54
PHRASE I 54.46
PHRASE L 52.54
PHRASE R 51.54
PICTURES 86
POINTERS 81.82
READ STATEMENTS 99 .23.53
REAL LOOP VARIABLES 42
REAL PROCEDURES 28
RECURSIVE INVOCATION 59
REPEAT FACTORS 49
REPLACE 84

REPLACEMENT EXPRESSIONS 67
RESERVED WORDS 90,14,10,16,18,13,31,34,40,45,42,64,68
SCALE FACTOR 51
SCAN 83
SCIENTIFIC NOTATION 7
SIDE EFFECTS 60
SIMPLE VARIABLES 8
SLASH 48,53
STANDARD FUNCTIONS 19
STANDARDS 37,41
STRING MANIPULATION 78
STRINGS 48
SUBSCRIPTED VARIABLES 21
SWITCHES 40,41,64
TIME FUNCTIONS 89
TYPE TRANSFER FUNCTIONS 77
UDECIMAL POINTS 50
UNLIMITED GROUPS 49
UNTYPED PROCEDURES 30
WHILE,THE RESERVED WORD 42
WRITE STATEMENTS 45,23,13,11,100