PASCAL

Mini text

by

Paul Stewart

Ver 3.2

January 93
reprint 2000

TABLE OF CONTENTS

LESSON 1

A BRIEF HISTORY

The computer language Pascal was named after Blaise Pascal, and was originally developed in the early 1970's by Niklaus Wirth of Zurich, Switzerland. It was derived from the language ALGOL.

Pascal is a structured programming language that can be used for both data processing and scientific problems.

Sample 1

```
        { SAMPLE 1 }
        { PROGRAM TO ADD 3 NUMBERS TOGETHER }
        PROGRAM Add3;
        VAR Sum: INTEGER;
        BEGIN { ADD3 }
                Sum := 25 + 33 + 75;
                WRITELN(Sum)
        END. { Add3 }
```

Notes on Sample 1

1) The first two lines of this program are comments. Comments are delimited parenthesis or brackets and and asterick. ie. The left delimiter is { or (* and the right delimiter is } or *) . Comments may appear anywhere on a line and may be on the same line as an executable statement.

2) The third line is the PROGRAM statement. It must be the first executable line of a Pascal program. The general form is:

PROGRAM name ;

- "name" is a variable given to the program, it may be any legal variable name.

3) The next line specifies the variable names that will be used in the program.
The general form is:

VAR var1,var2,var3, . . . :type;

```
eg.  VAR Sum, Number: INTEGER;

eg.  VAR Average: REAL;

eg.  VAR Sum, Number: INTEGER;
           Average: REAL;
```

Rules for Variable Names and identifies in Pascal

- may be any number of letters or digits
- first character must be a letter
- all identifies used in the program must be declared, such as in a VAR statement, or other
- an identifier maybe a keyword, but then that keyword has a different use. eg. You could use WRITELN as a variable name, but then it would NOT output information any more.

4) BEGIN and END are keywords that act as "markers" indicating the beginning and the end of a set of statements (a set of statements are called compound statements).

Every program must have at least one set of BEGIN & ENDs. The last END must terminate with a period.

Compound statements may be nested within each other. (see examples)

Note the comment after the BEGIN and END, the comments are not necessary but they can help the programmer to identify matching BEGINs and ENDs.

```
eg.     BEGIN
                statement 1;
                statement 2;
                        .
                statement n
        END;


eg      BEGIN
                 statement 1;
                 statement 2;
                 statement 3;
                      BEGIN
                        statement 4;
                        statement 5
                      END;
                 statement 6
        END;
```

Note: BEGIN and END lines are not considered statements and thus a semicolon is not needed. In general, statements must be separated by semicolons. The trick is to distinguish between what is a statement and what is not a statement. The best method for getting use to where to and where not to place semicolons is to look at examples of programs.

Note: Pascal statements may begin in any column. Indentation is used only for the purpose of clarity. It is suggested, however, that you do get in the habit of indenting.

5) The line Sum := 25 + 33 + 75; is an assignment statement. The symbol **:=** means "is assigned the value". Note the colon :
must be used.

The statement assigns what ever is on the right hand side of the **:=** to the variable name on the left hand side.

PASCAL arithmetic operators are

+	
-	
*	
/	operands may be integer or real result is real
DIV	operands must be integer result is integer
MOD	operands must be integer result is integer

The operator DIV is used for integer division
eg

8 DIV 2 gives a result of 4
8 DIV 3 gives a result of 2
3 DIV 8 gives a result of 0

The operator MOD is used to determine the remainder of an integer division.
eg.

8 MOD 2	gives a result of 0
8 MOD 3	gives a result of 2
3 MOD 8	gives a result of 3

Order of operations

a) parentheses

b) negation

```
*  /  DIV  MOD

+  -
```

6) The statement WRITELN is used for output. In this example the value stored in the variable Sum is outputted. If a character string is to be outputted it must be enclosed in <u>single</u> quotes (').

```
ie.    WRITELN('THE ANSWER IS ',Sum)
```

The general form is:

WRITELN(variable 1, variable 2, . . .   **);**

LESSON 2

Sample 2

```
{ THIS IS THE SECOND SAMPLE PROGRAM }
{ THIS PROGRAM HAS AN ERROR IN IT }
PROGRAM Samole;
VAR Number1, Number2:INTEGER;
        Sum:INTEGER;

BEGIN
        Number1 := 36;
        Number2 := 47;
      _ -THIS IS AN INVALID LINE-;
        WRITELN(Number1, Number2, Sum)
END.
```

```
ERROR 113: Error in Statement
```

The above is an example of a Pascal error message. The Cursor stops at the location the compiler has detected the error.

Sometimes it may detect the wrong error
eg. WRITELN(Hello There) If you forget to put quotation marks, it will assume the message Hello There is a variable name and give you an error about variables.

Sample 3

```
        PROGRAM Whldo;
        { THIS PROGRAM SHOWS THE WHILE-DO LOOP }

        VAR Counter:INTEGER;
                Sum:INTEGER;

        BEGIN
                Counter := 1;
                Sum := 0;
                WHILE Counter <= 15 DO
                        BEGIN { WHILE LOOP }
                        Sum := Sum + Counter;
                        Counter := Counter + 1
                        End;
                WRITELN('SUM =',Sum)
        END.


SUM =120                            output
-----------------------------------------------------
```

Notes on sample 3

1) The VAR statement is divided into two lines. Note though that the keyword VAR is used only once. The VAR statement could also have been written on just one line;

```
                ie.   VAR  Counter, Sum:INTEGER;
```

2) The WHILE-DO loop

The general form of the WHILE-DO is:

```
                WHILE condition DO
                        statement;


                or if there is a compound statement

                WHILE condition DO
                        BEGIN
                        statement 1;
                        statement 2;

                        statement n
                        END;
```

The "condition" is evaluated at the beginning of each loop. If the "condition" is true the loop is executed. The loop continues to be executed until the condition becomes false.

Pascal relational operators

Symbol	Meaning
=	equal to
<>	not equal to
>	greater than
<	less than
>=	greater than or equal to
<=	less than or equal to

Sample 4

```
{ THIS PROGRAM WILL READ IN TWO NUMBER,
        FIND THE SUM OF THE TWO NUMBERS
        AND PRINT THE NUMBERS AND THEIR SUM }
PROGRAM Sample;

VAR
 Number1:INTEGER; {THE FIRST NUMBER}
 Number2:INTEGER; {THE SECOND NUMBER}
 Sum:INTEGER; {THE SUM}

BEGIN
        READLN(Number1,Number2);
        Sum := Number1 + Number2;
        WRITELN('THE SUM OF',Number1,'    AND',
                Number2,'    IS',Sum)
END.
```

```
THE SUM OF19642    AND41234    IS60876
```

Notes on sample 4

Notice that in the output there is No space between the word and the number, if you wish a space you must include the space in your quote marks.

```
                                    sp                 sp
                                     |                  |
                  WRITELN('THE SUM OF ',Number1,'    AND ',
                          Number2,'    IS ',Sum)
                                         |
                                        sp
```

The READLN statement is used to read information into memory, (usually from the keyboard).

The general form is:

READLN(variable 1, variable 2, . . .**);**

Note: Each time a READLN is executed a new data line is read
Data items must be separated by 1 or more blanks.

Sample 5

```
       { THIS PROGRAM WILL READ IN 6 NUMBERS AND }
       {  ADD THEM TOGETHER }
       PROGRAM Adder;
       VAR Count, Number, Sum:INTEGER;

       BEGIN
               Count := 1;
               Sum := 0;
               WHILE Count <= 6 DO
                       BEGIN { READ AND SUM LOOP }
                       READLN(Number);
                       Sum := Sum + Number;
                       Count := Count + 1
                       END; { READ AND SUM LOOP }
               WRITELN('THE SUM IS',Sum)
       END.
 346
  35
 987
1243
```

```
            8
          456
THE SUM IS3075
```

Sample 6

```
{ THIS PROGRAM WILL READ  N  INTEGERS,
     AND ADD THEM TOGETHER }
PROGRAM Adder2;
VAR Number, Sum, Qty:INTEGER;

BEGIN
        Sum := 0;
        READLN(Qty);
        WHILE Qty > 0 DO
                BEGIN { WHILE LOOP }
                READLN(Number);
                Sum := Sum + Number;
                Qty := Qty - 1
                END; { WHILE LOOP }
        WRITELN('SUM IS ',Sum)
END.
```

```
3
45
921
35
SUM IS 1001
```

Notes on sample 5 & sample 6.

Both sample 5 & sample 6 perform basically the same procedure. Both read integers from data, add them together and print out their total. The difference between the two is; sample 5 is limited to exactly 6 data lines, whereas sample 6 can read n or any number of data lines.

For sample 5 there must be exactly six data values with one number per line. For sample 6 there may be any amount of data values with one number per line, In each use of sample 6 the number of values to be used is provided as the first data item.

Examples of data

For Sample 5	For Sample 6
346	3
35	45
987	921
1243	35
8	

456

over

```
        or sample 6 could use                          5
                                                       73
                                                       121
                                                       57
                                                       4632
                                                       30
```

Sample 7

The next sample program is going to determine the maximum of a list of non-negative numbers. The program uses a process that is repeated for each of the numbers. It will get the next number on the list and compare it to the greatest number found "so far". If the new number is greater than the previous maximum it will be kept as a new maximum.

The numbers at the beginning of each line of sample 7 are not part of the program but have been added only to identify each part of the program for discussion.

Sample 7

```
1               { FIND THE MAXIMUM OF NON-NEGATIVE NUMBERS }
2               PROGRAM Findmax;
3
4               VAR Number,           { THE CURRENT NUMBER }
5                   MaxNbr,           { MAXIMUM VALUE SO FAR }
6                   NbrLine:          { NUMBER OF LINES }
7                           INTEGER;
8
9               BEGIN { FIND MAX }
10                  MaxNbr := -99;  { INITIAL VALUE, LESS THAN
11                                 ALL POSSIBLE VALUES }
12                  READLN(NbrLine);
13                  WHILE NbrLine > 0 DO
14
15                        BEGIN
16                         READLN(Number);
17                         IF Number > MaxNbr THEN
18                                 MaxNbr := Number;
19                         NbrLine := NbrLine - 1
20                        END;
21
22                   WRITELN('MAXIMUM VALUE =',MaxNbr)
23              END.
```

Notes on sample 7.

The main loop that represents the heart of the program is lines 15 to 20. Line 10 initializes the value so that it works properly the first time through the loop. Lines 3, 8, 11, 14, and 21 are blank and are inserted to visually indicate the separate sections of the program. They have no effect on its execution.

If the following was used as test data the output would be:

```
data                        output
  5                      MAXIMUM VALUE =123
 45
 37
123
 59
101
```

ASSIGNMENT LESSON 2

1) Write a PASCAL program to read N data lines, find the sum and the average of these N numbers.

2) Write a PASCAL program to read N 2 digit numbers and:
   a) finds the sum
   b) finds the sum of the ones digits
   c) finds the sum of the tens digits

3) Write a PASCAL program to find both the maximum and the minimum from a list of non-negative numbers.

LESSON 3

Mixed Mode Arithmetic

Using REALS & INTEGERS

In PASCAL generally:

1) When a REAL is expected, it will generally accept an INTEGER.

2) When an INTEGER is expected, it will generally NOT accept a REAL.

3) When REAL and INTEGER values are combined in an expression the result is generally of type REAL

examples

a)	Real & REAL	Real
b)	Real & Integer	Real
c)	Integer & Integer	Integer

Examples of illegal expressions.

assume I1, I2, I3 are integer variables
R1, R2 are real variables

a)	I1 := R1	illegal
b)	I2 := I2 + R2	illegal
c)	I3 := 45.7	illegal

SAMPLE 8

```
PROGRAM MixMode;
VAR I1, I2, I3:INTEGER;
    R1, R2, R3:REAL;
BEGIN
        I1 := 4;
        I2 := 37;
        R1 := 8.4;
        R2 := 0.15;
        R3 := R1 + R2;
        I3 := I1 + I2;
        WRITELN(I1,I2,I3,R1,R2,R3);
        R2 := I1 + R1;
        WRITELN(R2)
END.
```

```
43741 8.4000000000E+00 1.5000000000E-01 8.5500000000E+00
 1.2400000000E+01
```

Notes on sample 8

Real variable names are declared in the VAR statement the same way as integer variables, except the variables are followed by the keyword REAL.

Notice that the keyword VAR is used only once.

Real variables are outputted in scientific notation.

example

15.23	becomes	1.5230000000E+01
0.1523	becomes	1.5230000000E-01
1523.0	becomes	1.5230000000E+03
0.0001523	becomes	1.5230000000E-04

NOTE: In PASCAL use R1 := 0.34

NOT

R1 := .34

NOTE: Real numbers leave a space in front of the number for the sign

Integer numbers do NOT leave the space.

IF THEN ELSE statement

SAMPLE 9

```
                PROGRAM FindLarge;

                { THIS PROGRAM READS IN TWO INTEGERS. IT
                   WILL DETERMINE WHICH IS THE LARGER OF THE TWO,
                  AND PRINT AN APPROPRIATE MESSAGE }

                VAR
                        Number1:INTEGER;{FIRST NUMBER READ}
                        Number2:INTEGER;{SECOND NUMBER READ}
                        Larger:INTEGER;{THE LARGER ONE}

                BEGIN
                        READLN(Number1,Number2);
                        IF Number1 > Number2 THEN
                                Larger := Number1
                        ELSE
                                Larger := Number2;
                        WRITELN('THE LARGER OF ',Number1,
                                ' AND ',Number2,' IS ',Larger)
                END.
```

Notes on Sample 9

Often it is necessary for a program to choose between two different statements, depending on some condition. The IF statement can be used for this. In this program, an IF statement is used to choose one of two different statements.

The general form of the IF is;

```
        IF condition THEN
                statement
        ELSE
                statement;
```

The condition (as in the WHILE DO) is evaluated to be either TRUE or FALSE. If the condition is true, then the statement following the word **THEN** is executed, otherwise it is not. This statement may, optionally, be followed by the reserved word **ELSE** and a second statement. The statement following the word ELSE will be executed only if the condition is FALSE.

SAMPLE 10

```
PROGRAM  BigAndSmall;

{ THIS PROGRAM READS IN TWO NUMBERS, DETERMINES
    WHICH IS THE LARGER AND WHICH IS THE SMALLER,
   AND THEN OUTPUTS THE RESULTS. }

VAR
        Number1,Number2:INTEGER; {THE NUMBERS}
        Larger:INTEGER; {THE LARGER ONE}
        Smaller:INTEGER; {THE SMALLER ONE}

BEGIN
        READLN(Number1,Number2);
        IF Number1 > Number2 THEN
                BEGIN
                Larger := Number1;
                Smaller := Number2
                END
        ELSE
                BEGIN
                Larger := Number2;
                Smaller := Number1
                END;
        WRITELN('THE NUMBERS ARE ',Number1,
                ' AND ',Number2);
        WRITELN('THE LARGER ONE IS ',Larger);
        WRITELN('THE SMALLER ONE IS ',Smaller)
END.
```

Notes sample 10

The above example uses compound statements in the IF. Note the use of semicolons in the IF. Only one semicolon appears inside each compound statement, to separate the two smaller statements inside. The entire IF statement is ten lines long, but is still a single statement and thus has only one semicolon at the end. It contains two compound statements as part of itself, which in turn, contain other statements. This **nesting** of statements inside other statements is common in most programming.

There is **NEVER** a semicolon BEFORE an ELSE.

SAMPLE 11

```
  { PROGRAM TO READ A NUMBER AND IF THE NUMBER
     IS POSITIVE FIND THE SQUARE ROOT }
  PROGRAM SquareRoot;

  VAR Value, SqRoot:REAL;
      Finished:CHAR;

  BEGIN
     Finished := 'N';
     WHILE Finished <> 'Y' DO
           BEGIN {WHILE LOOP}
            READLN(Value);
            IF Value >= 0 THEN
                 BEGIN
                  SqRoot := SQRT(Value);
                  WRITELN(Value,SqRoot)
                 END
            ELSE
                 WRITELN(Value,' IS A NEGATIVE NUMBER);

            WRITELN('Are you Done entering data  ');
            READLN(Finished);
     END {WHILE LOOP}
  END.
```

```
 2.0000000000E+00 1.414235623731E+00
 4.0000000000E+00 2.000000000000E+00
-3.2000000000E+00 IS A NEGATIVE NUMBER
 6.2500000000E+02 2.500000000000E+01
 0.0000000000E+00 0.000000000000E+00
 1.6000000000E+01 4.000000000000E+00
```

Notes on Sample 11

The WHILE loop uses a CHARacter variable to determine if more data is to be input. A CHARacter variable can store one single character only, not a string of characters.

SQRT is built-in function, it determines the square root of a real number.

IF statement

The IF statement may be one of the following forms:

```
a)      IF condition THEN
                statement;



b)      IF condition THEN
                statement 1
        ELSE
                statement;



c)      IF condition THEN
                BEGIN
                statement 1;
                statement 2;

                statement n
                END
        ELSE
                BEGIN
                statement 1;
                statement 2;

                statement m
                END;
```

ASSIGNMENT LESSON 3

1) Write a Pascal program to read in n integer values. It should count the number of positive integers (greater than or equal to zero) and the number of negative integers read.

2) Write a Pascal program that will print a temperature conversion table from Fahrenheit to Celsius, for all temperature from -50 to 100 degrees Fahrenheit.
   Note: the formula for conversion is:

   Celsius = (Fahrenheit - 32) * 5/9

3) Write a Pascal program to print the first 10 Fibonacci numbers. The first six Fibonacci numbers are 0, 1, 1, 2, 3, 5

4) Input consists of 2 positive integers m and n , with m <= n
   Write a program to print all Fibonacci numbers which are between (and including) m and n.
   Use the following test data:

0	0
1	1
1	3
0	13
0	14
23	100
5	2

5) Write a Pascal program which will read in the data on car salespersons. There will be input for each salesperson, containing the salesperson's number (4 digit integer), and three integers indicating the number of; compact, midsize, and gasguzzlers sold last month.
   example

```
4356  3  5  2
```

   This data indicates that salesperson number 4356, sold 3 compact cars, 5 midsize cars, and 2 gasguzzllers, last month.

   Your program should produce the following statistics. For each salesperson, print the total number of cars sold, At the end, print the total number of each size of car. Also print the salesperson number of the best salesperson (most cars sold), and of the worst salesperson (least cars sold).

LESSON 4

The   **REPEAT UNTIL** LOOP

The general form is:

```
a)      REPEAT
              statement
        UNTIL condition ;

                OR

b)      REPEAT
              statement 1;
              statement 2;

              statement n
        UNTIL condition ;
```

The REPEAT UNTIL loop is somewhat similar to the WHILE loop
Except:
The 'condition' is evaluated at the end of the loop and therefore the loop is always executed at least once.

If the 'condition' is TRUE, execution of the loop is terminated.

IF the 'condition' is FALSE, the loop continues.

This is the opposite of the WHILE loop.

There is no need for a compound statement if several statements are required inside the loop. The keywords REPEAT and UNTIL are used to delimit the sequence of statements inside the loop, making the use of BEGIN and END unnecessary.

The **FOR** **DO** loop

An example of the FOR DO loop

```
FOR Count := 1 TO 10 DO
        statement;
```

The above will cause the statement immediately following the **DO** to be executed 10 times. The first time Count will have the value 1, the second time the value 2, and the last time the value of 10. The statement

```
FOR Count := 10 DOWNTO 1 DO
        statement;
```

will also cause the statement inside the loop to be executed 10 times, but Count will start at 10 and decrease to 1.

The general form is:

```
a)     FOR  index variable := initial value  TO  final value  DO
              statement;

                                 OR

 b)    FOR  index variable := initial value TO final value  DO
              BEGIN
                 statement 1;
                 statement 2;

                 statement n
              END;
```

NOTES:

In a FOR DO loop the index variable, initial value, and final value must be integer.

Also the index variable may NOT be changed within the body of the loop.

The index variable can only be incremented or decremented by 1 each time through the loop.

SAMPLE 12

Write a Pascal program to sum the integers from 14 to 728 using
a) WHILE DO loop   c)FOR DO loop with increasing
values
b) REPEAT UNTIL loop   d)FOR DO loop with decreasing
values

```
        PROGRAM Loop;
        VAR
                I, SumWhile, SumRepeat, SumTo, SumDownTo:LONGINT;

        BEGIN
                WRITELN('THE SUM OF 14 TO 728 USING:');
                I := 14;
                SumWhile := 0;
                WHILE I <= 728 DO
                        BEGIN
                        SumWhile := SumWhile + I;
                        I := I + 1
                        END;
                WRITELN('      WHILE IS ',SumWhile);

                I := 14;
                SumRepeat := 0;
                REPEAT
                        SumRepeat := SumRepeat + I;
                        I := I + 1
                UNTIL I > 728;
                WRITELN('      REPEAT IS ',SumRepeat);

                SumTo := 0;
                FOR I := 14 TO 728 DO
                        SumTo := SumTo + I;
                WRITELN('      FOR TO IS ',SumTo);

                SumDownTo := 0;
                FOR I := 728 DownTo 14 DO
                        SumDownTo := SumDownTo + I;
                WRITELN('      FOR DOWNTO IS ',SumDownTo)
        END.
```

```
THE SUM OF 14 TO 728 USING:
        WHILE IS 265265
        REPEAT IS 265265
        FOR TO IS 265265
        FOR DOWNTO IS 265265
```

Notes On Sample 12:

Notice the variables are declared as LONGINT (long integer) not as INTEGER. This is because of memory restrictions of INTEGER values. An INTEGER may be in the range from -32768 to + 32767, the totals from sample 12 are out of this range.

LONGINTs are in the range from -2,147,483,648 to 2,147,483,647

The variable I could have been declared an INTEGER and the others as LONGINT, since the largest value stored in I is 728.

eg.

```
   VAR
           I:INTEGER:
           SumWhile, SumRepeat, SumTo, SumDownTo:LONGINT;
```

ASSIGNMENT LESSON 4

1) THE AUTOMATIC CHANGE MAKER. Write a Pascal program that will read in any amount less than $10.00 and will give the correct change using the coins; pennies, nickels, dimes, quarters, loonies.

```
   eg.     $0.67         2 pennies
                         1 nickel
                         1 dime
                         2 quarters
                         0 loonies
```

2) Write a program which will determine the unit price (cents per gram) of different boxes of laundry soap described by a line of the form:

kilograms	price
5	3.47
3.5	2.19

3) Write a program to input any positive integer number.
If the number is even divide it by 2.
If the number is odd multiply it by 3 and add 1.
Repeat the above two steps until a value of one is reached.
Count how many steps it takes for each number. Output the original number and the total number of steps. Use the following test data:

```
  32
 -49
  49
3897
1024
   7
```

LESSON 5

WRITE statement

The WRITE statement is similar to the WRITELN statement.

The general form is:

```
            WRITE( variable 1, variable 2, . . . variable n );
```

Formatting may be used in the WRITE the same as in the WRITELN.

The difference between WRITE and WRITELN is that each time a WRITELN is executed a line is output a a new line starts. Whereas in a WRITE statement a new line does NOT start.

For example:

```
    WRITELN(X, Y, Z)        is equivalent to      WRITE(X);
                                                  WRITE(Y);
                                                  WRITE(Z);
                                                  WRITELN
```

This can be very useful, since it allows a program to use separate Pascal statement to produce output items which will appear on the same output line.

A WRITELN statement with no output items should be used to terminate the current output line after a sequence of WRITE statements, as in:

```
                WRITE('HELLO THERE ');
                WRITE('HOW ARE YOU');
                WRITELN

        Output
        HELLO THERE HOW ARE YOU
```

FORMATTING OF OUTPUT

INTEGER

The general form is:

WRITELN(variable:field size**);**

where field size specifies the minimum field width the value will occupy.

for example

```
WRITELN(Num:5)
```

means to print the value of the variable Num in a field of at least 5 spaces. If the field size, in this example 5, is not large enough to print the variable, then the field is automatically increased.

EXAMPLES

If Num has a value of 9876 then:

```
WRITELN(Num:4)      gives  9876
WRITELN(Num:6)      gives  bb9876
WRITELN(Num:2)      gives  9876
WRITELN(Num)        gives  9876
```

REALS

The general form is:

WRITELN(variable: field size: decimal size**);**

Where field size is the minimum field width and decimal size is the number of digits to the right of the decimal point.

Examples

If x has the value 12.345 then:

```
WRITELN(X:5:1)      gives  b12.3
WRITELN(X:8:3)      gives  bb12.345
WRITELN(X:6:2)      gives  b12.35  note the rounding
WRITELN(X:2:2)      gives  12.35
WRITELN(X)          gives  b1.2345000000E+01
```

Note: In the last example if no field size is given the default size of 13 is assumed and it is printed in exponential notation.

CHARACTERS

Formats can also be very useful with character strings. The general form is:

WRITELN('string':field size**)**

Examples

```
     WRITELN('HELLO':5)  gives     HELLO       Note: String
fields         WRITELN('HELLO':10) gives     bbbbbHELLO
are right           WRITELN('HELLO':2)  gives     HE
 justified in       WRITELN(' ':7)      gives     bbbbbbb
 Pascal
```

SAMPLE 13

```
PROGRAM ZELLAR;

{ THIS PROGRAM WILL READ IN A DATE AND DETERMINE
        THE DAY OF THE WEEK. THE DAY , MONTH , AND YEAR
        MUST BE SUPPLIED.  THE FORMULA USED IS CALLED
        "ZELLAR'S CONGRUENCE". }

VAR
        Day,Month,Year,Century:INTEGER;
        DayOfWeek:INTEGER;

BEGIN
        READLN(Month,Day,Year);
        WRITE(Month:2,'/',Day:2,'/',Year:4,'IS A');

        IF Month>2 THEN
                Month := Month - 2
        ELSE
                BEGIN
                Month := Month + 10;
                Year := Year - 1
                END;
        Century := Year DIV 100;
        Year := Year MOD 100;

        DayOfWeek := (Day+(26*Month-2)DIV 10
                + Year + Year DIV 4 + Century DIV 4
                + 5*Century) MOD 7;

        WRITELN(DayOfWeek)
END.
```

A typical line of output from this program would be:

```
11/ 1/2060 IS A 1
```

This means that November 1, 2060 will be a Monday.

THE CASE STATEMENT

If there are more then two alternatives to choose from the CASE statement is useful.

The general form is:

```
CASE expression OF
        label 1: statement 1;
        label 2: statement 2;
                .
                .
        label n: statement n
END;
```

Example- The following could be added to the program Sample 13

```
CASE DayOfWeek OF
        1 :WRITELN('MONDAY');
        2 :WRITELN('TUESDAY');
        3 :WRITELN('WEDNESDAY');
        4 :WRITELN('THURSDAY');
        5 :WRITELN('FRIDAY');
        6,7 :WRITELN('WEEKEND')
END
```

NOTES:

1) The labels in a CASE statement are 'local' to the particular CASE statement in which they appear. The same label may not appear more then once in the same CASE statement.

However the same case-label may appear in a different CASE statement.

3) The CASE statement must terminate with an END. This is the only time an END does not match with a BEGIN.

4) You may also use compound statements within a CASE statement.

eg.

```
CASE expression OF
        label 1:BEGIN
                  statement;
                  statement;
                  statement
                  END;
        label 2:BEGIN
                  statement;
                  statement
                  END;
        label 3:statement;
                .
                .
END;
```

5) Empty statements may be used in a CASE statement to allow case clauses in which no action at all is performed.

example

```
CASE Code OF
1:WRITELN('ONE');
2,3: ;
4:WRITELN('MANY')
END
```

SAMPLE 14

Sample 14 is a program which uses the CASE statement numbers from 1 to 3. Code 1 indicates that a cheque has been written. The cheque number and the amount of the cheque will appear on the next line. Code 2 indicates that a deposit has been made, and the amount of the deposit is given on the following line. Code 3 indicates that a service charge has been made. The amount of the charge will appear on the next line. A CASE statement is used to select one of the three different compound statements which handle these three cases.

```
PROGRAM Cheque;
{ SAMPLE 14}

{ THIS PROGRAM WILL PRODUCE A SIMPLE STATEMENT OF
        A CHEQUING ACCOUNT.  IT WILL HANDLE THREE POSSIBLE CODES
        THESE CODES ARE:
          STOP PROCESSING       0
          CHEQUE:               1  NUMBER  AMOUNT
          DEPOSIT:              2  AMOUNT
          SERVICE CH:           3  AMOUNT
        THE TRANSACTION WILL BE PRINTED WITH A BALANCE SHOWN
         ON EACH LINE. }

VAR
        Balance,
        Amount:REAL;
        Number, {CHEQUE NUMBER}
        Code:INTEGER;

BEGIN
        Balance := 0.0;
        WRITELN('DESCRIPTION','DEBITS':12,'CREDITS':12,
                'BALANCE':12);

        Code := 99;

        WHILE Code <> 0 DO
          BEGIN
                WRITE('ENTER THE TRANSACTION CODE ');
                READLN(Code);
                CASE Code OF

                1: BEGIN {CHEQUE}
                    WRITE('ENTER THE CHEQUE NUMBER AND AMOUNT');

                    READLN(Number,Amount);
                    Balance := Balance - Amount;
                    WRITELN('CHEQUE',Number:4,Amount:13:2,
                                ' ':12,Balance:12:2)
                   END;

                2: BEGIN {DEPOSIT}
                    WRITE('ENTER THE DEPOSIT AMOUNT ');
                    READLN(Amount);
                    Balance := Balance + Amount;
                    WRITELN('DEPOSIT',' ':16,Amount:12:2,
                                Balance:12:2)
                   END;

                3: BEGIN {SERVICE CHARGE}
                    WRITE('ENTER THE SERVICE CHARGE AMOUNT ');
                    READLN(Amount);
                    Balance := Balance - Amount;
                    WRITELN('SERVICE CHG',Amount:12:2,
                                ' ':12,Balance:12:2)
                   END
                END {CASE}
        END {WHILE}
END.
```

ASSIGNMENT LESSON 5

1) Write a Pascal program to print out the squares, and square roots of the numbers from 1 to 100. Output must be formatted with 20 records per page.

2) Write a Pascal program which will read in three integers representing a month, day, and a year. The program will determine whether or not the integers represent a valid date. For example, 02/31/1983 is not a valid date since February does not have 31 days. Your program should check the month, the day , and the year. Leap years should be handled correctly. Note: Check the library for the definition of a leap year, it is not just every four years.

LESSON 6

READ STATEMENT

The READ statement is similar to the READLN statement.

The general form is:

```
         READ( variable 1, variable 2, . . . variable n );
```

The difference between the READ and the READLN is that each time a READLN is executed a NEW line is read. Whereas with a READ the next data item is read, This may be on the same line or on another line. If the number of data items on the data line is greater than the number of variables in the READ statement, the next value on the data line will be read by the next READ statement. After a READ the "pointer" is at the next column on the current line. After a READLN the "pointer" is at the first column of the next line.

EOLN

The function **EOLN** checks for End Of Line. The value of the function is true if, while reading a line the end of the line is reached, (the ENTER key is depressed) otherwise it is false. The EOLN should be used with the READ statement, whereas the EOF should be used with the READLN statement.

SAMPLE 15

This program uses some of the built-in functions to determine whether or not an integer is a perfect square. The program will calculate the integer that is closest to the square root of the input value. If the input value is a perfect square then it must be the square of this integer.

```
{SAMPLE 15}
PROGRAM PerfectSq;

{ THIS PROGRAM WILL READ IN A POSITIVE INTEGER
        AND DETERMINE WHETHER OR NOT IT IS A
        PERFECT SQUARE. }

VAR
        Number, { THE NUMBER READ }
        NearSqrt { INTEGER CLOSEST TO ROOT }
                :INTEGER;

BEGIN
        READLN(Number);
        NearSqrt := ROUND(SQRT(Number));
        IF SQR(NearSqrt) = Number THEN
                WRITELN(Number,' IS A PERFECT SQUARE')
        ELSE
                WRITELN(Number,' IS NOT A PERFECT SQUARE')
END.
```

SAMPLE 16

This program will show one method to verify a check digit. Credit card numbers, social insurance numbers, and similar identification numbers often have one or more check digits, which make it possible to detect invalid or incorrectly entered numbers. One very simple check digit scheme involves adding together the individual digits in a number. The last digit of this sum is used as a check digit, which is added as an extra digit to the end of the number. For example, suppose the card number 971326 were to be processed by this method. The sum of the digits is 28. The last digit of this sum (8) would then be appended to the end of the number, giving 9713268. This is the number that would actually appear on the credit card. If this number is ever entered incorrectly, say as 9714268, this would easily be detected since the sum of the first 6 digits no longer agrees with the check digit. Real life situations often use more complex schemes but the principles are the same.

```
{SAMPLE 16}
PROGRAM Check;

{ THIS PROGRAM WILL READ IN A POSITIVE INTEGER
        AND DETERMINE WHETHER OR NOT IT HAS A
        VALID CHECK DIGIT.  THE CHECK DIGIT IS THE
        LAST DIGIT OF THE NUMBER. TO BE VALID, IT
        MUST EQUAL THE SUM OF THE OTHER DIGITS,
        MODULE 10. }

VAR
        CardNumber:LONGINT; {THE NUMBER READ}
        CheckDigit:INTEGER; {ITS LAST DIGIT}
        DigitsLeft:LONGINT; {THE OTHER DIGITS}
        Sum:INTEGER; {SUM OF OTHER DIGITS}

BEGIN
        WRITE('ENTER THE CARD NUMBER TO VERIFY ');
        READLN(CardNumber);
        CheckDigit := CardNumber MOD 10;
        DigitsLeft := CardNumber DIV 10;
        Sum := 0;

        WHILE DigitsLeft > 0 DO
                BEGIN
                Sum := Sum + DigitsLeft MOD 10;
                DigitsLeft := DigitsLeft DIV 10
                END;

        IF CheckDigit = Sum MOD 10 THEN
                WRITELN(CardNumber,' IS VALID')
        ELSE
                WRITELN(CardNumber,' IS INVALID')
END.
```

ASSIGNMENT LESSON 6

1) Write a Pascal program which will read in a positive integer (greater than zero), and print out this integer with the digits reversed. For example, if the integer 123456 were read in, then output 654321. Then output the total of the two results.

2) In Canada, Social Insurance Numbers contain a check digit which allows the following method to be used to determine whether or not a particular number is valid. First, separate the number into its 9 individual digits. (for example, 621578442 becomes 6, 2, 1, 5, 7, 8, 4, 4, 2.) Then double every second digit. (The example becomes 6, 4, 1, 10, 7, 16, 4, 8, 2) Then find the sum of every digit in the resulting list of numbers. For the number 16 add 1 and 6 to the sum separately, do not add the number 16 itself. (In the example, the sum is 6 + 4 + 1 + 1 + 0 + 7 + 1 + 6 + 4 + 8 + 2 = 40) The Social Insurance number is valid if the sum is a multiple of 10, and invalid otherwise. Therefore 621578442 is a valid Social Insurance Number since the sum was 40. Write a Pascal program which will read in a Social Insurance Number and determine whether or not it is valid.

3) Write a Pascal program which will read in a positive integer (greater than zero) and determine how many times that integer can be evenly divided by 2. For example, 40 may be evenly divided by 2 three times (40/2=20,20/2=10,10/2=5).

Assignment Lesson 6 cont

4) Write a Pascal program that reads in vehicle information with regards to a toll bridge. Data will consist of 1 or 2 items per record. If there is one item per record then it must be a:

```
 1 for motorcycle
or
 2 for cars
or
 3 for cars with trailers
```

If there are 2 items per record then the first number must be a 4 followed by a number representing the weight in tons.

```
4  2.5     (truck 2.5 tons)
```

the toll cost is:

motorcycle	0.25
cars	0.50
cars with trailer	0.75
trucks	0.40 per ton

Using the following data to print a report giving the total number of each vehicle type and the total amount collected in tolls for each vehicle type.

```
1
2
1
3
2
2
1
3
4 5
4 4.5
3
2
1
3
3
3
4 3
3
2
```

LESSON 7

DEFINITION OF CONSTANTS

So far we have discussed variables as being type integer, type real or type character. We store a value in the variables using the assignment statement.   ie. AREA **:=** LENGTH * WIDTH

However sometimes it is useful to define a particular value to be constant throughout the entire program. For example, suppose in a program you frequently use the value of pi, say 3.1415926536. It would become quite a bother to rewrite all those digits each time you wanted the value pi. Alternatively you could define pi as a constant.

```
                 CONST Pi = 3.1415926536;
```

With this definition you simply write the name PI instead of the numeric value. You could have also declared a variable PI and assigned it the same value:

```
                 VAR Pi: REAL;
                      . . .
                 Pi := 3.1415926536
```

In most programming language this is exactly what is done, but Pascal offers an additional feature of CONST. One of the differences is that the value of a constant name can NOT be changed. Neither an assignment nor a READ statement can accidentally change the value of a constant.

The general form is:

```
           CONST  name 1 = value 1;
                  name 2 = value 2;
                        .
                        .
```

```
        example
                CONST
                        Yes = TRUE;
                        No = FALSE;
                        Pi = 3.141592653589793;
                        MinusPi = -Pi;
                        Message = 'SOMETHING IS WRONG';
                        Blank = ' ';
```

DEFINITION OF TYPES

So far we have seen 4 types of variables; Real, Integer, Character and Constant (Pascal also has type STRING - strings of characters and type BOOLEAN - true and false). However Pascal goes beyond this and allows the user to define his/her own types.

The general form is:

TYPE type-name **=**   value1, value2, . . . ;

example

```
    TYPE    WeekDay = (MON,TUE,WED,THUR,FRI);   Note: characters
            WeekEnd = (SAT,SUN);                   are enclosed
            Dates = 1..31;                         in brackets.
                                                Numerics are not
```

In this example WeekDay may have one of 5 values, and WeekEnd may have one of 2 values. Pascal monitors the variable to ensure that only legitimate values are assigned.

User defined types are for <u>internal use only</u>.
That is the values can be assigned, used, tested within the program but those values cannot be displayed (WRITE) or assigned from external data (READ)

SAMPLE 17

```
{SAMPLE 17}
PROGRAM Days;

TYPE Day = (Sun,Mon,Tue,Wed,Thurs,Fri,Sat);

VAR DayOfWeek:Day;


BEGIN
        FOR DayOfWeek := Mon TO Fri DO
                CASE DayOfWeek OF
                        Sat,Sun: WRITELN('WEEKEND HEY');
                        Mon: WRITELN('DAY 1');
                        Tue: WRITELN('DAY 2');
                        Wed: WRITELN('DAY 3');
                        Thurs: WRITELN('DAY 4');
                        Fri: WRITELN('DAY 5')
                END{CASE}
END.
```

```
              output
              DAY 1
              DAY 2
              DAY 3
              DAY 4
              DAY 5
```

ORDER OF DECLARATION STATEMENTS

There are 4 declaration statements in Pascal. The order in which they must appear in the program is:

```
LABEL
CONST
TYPE
VAR
```

Note: The LABEL statement has not yet been used.

example

```
PROGRAM Exmpl;

CONST
        Pi = 3.1415926536;
        Name = 'JANE DOE';

TYPE
        Month = (Jan,Feb,Mar,Apr,May,Jun,
                        Jul,Aug,Sep,Oct,Nov,Dec);
        Dates =   1..31 ;

VAR
        Year,NumberOfMonth,DayOfMonth:INTEGER
        Weight:REAL

BEGIN
        .
        .
        .
```

SUBSCRIPTED VARIABLES

Pascal allows the use of arrays.

The general form is:

```
     VAR identifier: ARRAY[ lower bound . . upper bound] OF type;
         identifier: ARRAY[ lower bound . . upper bound,
                            lower bound . . upper bound] OF type;
```

examples

```
VAR     Vec1: ARRAY[1 . . 10] OF INTEGER;
        Vec2: ARRAY[-8 . . 22] OF INTEGER;
        Table: ARRAY[1 . . 5, 1 . . 10] OF REAL;
```

Vec1 is a one dimensional array of length 10.
Vec2 is a one dimensional array of length 31.
Table is a two dimensional array of size 5 by 10

Referencing of Arrays

Examples

```
Vec1[4]
Vec2[-3]
Vec1[J]
Table[3, 7]
Table[K, L]
```

Note: Neither simple variables nor arrays are automatically assigned any initial value by Pascal. They must be assigned a value by a program statement.

Note: You must use square brackets **[ ]** for arrays in Pascal

SAMPLE 18

```
{SAMPLE 18}
PROGRAM ArrayEx;

{THIS PROGRAM WILL READ IN SEVERAL INTEGER VALUES INTO AN ARRAY.
  READING WILL STOP WHEN THE ARRAY IS FULL OR END-OF-FILE IS
  REACHED.  EACH VALUE IS ROUNDED TO THE NEAREST MULTIPLE OF 10
AND
  THEN THE VALUES ARE PRINTED, 5 VALUES PER LINE. }

CONST
      Limit = 100; {MAXIMUM NUMBER OF VALUES}
TYPE
      SmallArray = ARRAY[1..Limit] OF INTEGER;
VAR
      DataList:SmallArray; {THE DATA READ}
      Index:INTEGER; {USED TO SUBSCRIP DATALIST}
      ListSize:INTEGER; {NUMBER OF ELEMENTS USED}
      YesNo:CHAR;

BEGIN
      ListSize := 0;
      YesNo := 'Y';
      WHILE (UPCASE(YesNo) = 'Y')  AND   (ListSize<Limit) DO
          BEGIN
          ListSize := ListSize + 1;
          WRITE('ENTER A NUMBER ');
          READLN(DataList[ListSize]);
          WRITE('More data y/n ');
          READLN(YesNo)
          END;

      FOR Index := 1 TO ListSize DO
          DataList[Index] := 10 * ((DataList[Index] + 5) DIV 10);

      FOR Index := 1 TO ListSize DO
          BEGIN
          WRITE(DataList[Index],'   ');
          IF (Index MOD 5) = 0 THEN
                    WRITELN
          END
END.
```

Notes:When using compound expressions
          (UPCASE(YesNo) = 'Y') AND (ListSize < Limit) you MUST
     put brackets around EACH condition.

SAMPLE 19

The program below will print out Pascal's triangle (the programming language Pascal is named after the same person). Pascal's triangle is the mathematical table shown below.

```
                                1
                              1   1
                           1    2   1
                         1    3   3    1
                       1   4    6   4    1
```

Each number in the triangle is the sum of the two numbers immediately above in the previous row. The first and last numbers in a row are always 1.

```
{SAMPLE 19}
PROGRAM Triangle               ;
{ THIS PROGRAM WILL PRINT OUT THE FIRST FEW
         ROWS OF PASCAL'S TRIANGLE. }
CONST
        LineWidth = 80; {SIZE OF OUTPUT LINES ie. THE SCREEN
WIDTH}
        Spacing = 3; {OFFSET BETWEEN ROWS}
        Size = 12; {NUMBER OF ROWS DESIRED}
VAR
        Row:ARRAY[1..SIZE] OF INTEGER;
        LeadingBlanks, LineNumber, I:INTEGER;

BEGIN
        LeadingBlanks := (LineWidth DIV 2) - Spacing;
        {SET FIRST ROW TO 1 0 0 0 . . . 0}
        Row[1] := 1;
        FOR I := 2 TO Size DO
                Row[I] := 0;

        FOR LineNumber := 1 TO Size DO
                BEGIN {FOR LOOP #2}
                FOR I:=LineNumber DownTo 2 DO
                        Row[I]:=Row[I] + Row[I-1];

                WRITE(' ':LeadingBlanks);
                FOR I:=1 TO LineNumber DO
                        WRITE(Row[I]:2*Spacing);

                WRITELN;
                LeadingBlanks := LeadingBlanks - Spacing
                END {FOR LOOP #2}
END.
```

ASSIGNMENT LESSON 7

1) Write a Pascal program which will read in up to 100 integers values, and print the values in the opposite order. There may be any number of values on each input line. Your program should place 5 values on each output line.

Example, if the input lines were

```
1    10    9    7    6    4
15    12    3
```

then the output should be

```
3          12          15          4          6
7           9          10          1
```

2) Write a Pascal program which will read in one input line for each employee of the Canadian Widget Company containing a 4 digit employee number and a sales rating from 0 to 100. There will be at most 50 employees. The program should calculate and print the average sales rating. It should then list, the employee numbers and their sales rating for those employees whose sales rating was at or above the average. The program should then list the data for those employees whose rating was below the average. Include appropriate headings.

3) Suppose that there are four different stores which each sell the same five products. The unit price for each product varies from store to store. The table of unit prices is shown below. Write a Pascal program which will read in this table of prices and then find the cost of filling the same purchase order at each of the four stores. It should also find the store with the lowest total cost. The purchase order will consist of 5 integers which will specify the quantities desired for each item. The input format is left up to you the programmer.

		PRODUCT				
S		1	2	3	4	5
T	1	1.06	3.49	0.43	0.22	6.98
O	2	1.04	3.55	0.40	0.25	6.50
R	3	0.99	3.25	0.50	0.26	6.40
E	4	1.06	3.35	0.45	0.20	6.72

LESSON 8

FUNCTIONS

In lesson 3, the built in function SQRT was discussed. The programmer can also define his or her own functions. A function is a subprogram which accepts certain data items (or parameters), processes this data to produce a result. As an example, the built-in function SQRT accepts a REAL data value as a parameter, and returns the square root as the result.

The following program shows a user defined function called MAX. This function accepts two integer data values as parameters and returns the largest one as its result.

SAMPLE 20

```
PROGRAM Biggest;
{THIS PROGRAM INPUT ONE POSITIVE INTEGER FROM EACH
  DATA LINE AND DETERMINES THE LARGEST ONE}

VAR
   Number,Largest:INTEGER;
   StopReading:CHAR;

FUNCTION Max(A:INTEGER;B:INTEGER):INTEGER;
   BEGIN
   IF A>B THEN
      Max:=A
   ELSE
      Max:=B
   END;

BEGIN {MAINLINE}
Largest:=-1;
StopReading := '';

WHILE UPCASE(StopReading) = 'Y' DO
   BEGIN
   WRITE('Enter a number ');
   READLN(Number);
   Largest:=MAX(Number,Largest)
   WRITE('Do you wish to stop  Press Y ');
   READLN(StopReading)
   END;
WRITELN('THE BIGGEST NUMBER READ WAS',Largest)
END.
```

Notes on Sample 20

After the declaration of the variables Number and Largest, the "function declaration" for the function Max appears. This declaration specifies the name (or identifier) of the function (Max), the parameters that this function will accept (the two integers A and B), and the type of result that will be produced (integer). The processing that is to be performed to give the result of the function is given between the words BEGIN and END. The function is also known as a "subprogram". Following the subprogram is the main program or "mainline". The statements of the mainline are executed as usual. The statements of the subprogram are only executed when the subprogram is used by the main program.

The program performs in the following manner. The variable name LARGEST is initialized to -1 then the while loop is executed repeated until the user enters Y at the prompt to stop reading, any other reply including a nul means continue. The first iteration of the loop, a value will be read into the variable Number. Assume this number is 32. The next statement; Largest:=Max(Number,Largest) contains a "function call" to the function Max. The values stored in Number and Largest (32 and -1) are passed to the function. The function stores these values in its two parameters A and B. Execution of the mainline is temporarily suspended while the result of the function call Max(Number,Largest) is calculated. The statements within the function are now executed. Variable A has a value of 27 and B has a value of -1, so the value of A (32) is assigned to the "function identifier" Max. The function is now complete so the result (32) is returned to the main program. The value of Max(Number,Largest) is now known, so the main program resumes executing and the value 32 is assigned to the variable Largest.

The sequence of events which occurs when a function is called by the mainline is:

1) Each expression that appears in a function call is evaluated, before the set of values is passed to the function.

2) All the variables of the function (local variables) are created and the parameters are assigned the values passed to them.

3) The mainline is "suspended" until a result is returned from the function.

4) The statements of the function are executed. A value is assigned to the function identifier and this becomes the result of the function.

5) The result is returned to the mainline and all the local variables of the function disappear.

6) The mainline resumes execution, using the value returned from the function.

The general form of a FUNCTION is:

```
     FUNCTION function name (formal parameter list):result-type;
           local declaration section
     BEGIN
        body of function (statements)
     END;
```

SAMPLE 21

```
PROGRAM Factorial;

{THIS PROGRAM READS IN AN INTEGER N AND THEN GIVES
  THE FACTORIALS OF THE INTEGERS FROM 1 TO N}

VAR
  I,N:INTEGER;

FUNCTION Fact(N:INTEGER):INTEGER;
{FUNCTION TO CALCULATE N FACTORIAL}
  VAR
    I,Ans:INTEGER;
  BEGIN
   Ans:=1;
   FOR I:=2 TO N DO
       Ans:=Ans * I;
   Fact:=Ans
  END;{FUNCTION FACT}

BEGIN {MAIN PROGRAM}
    WRITE('What number do you wish to find the factorial of ');
    READLN(N);
    FOR I:=1 TO N DO
        WRITELN(I,'FACTORIAL IS',Fact(I))
END.
```

Notes Sample 21

The program uses the variable names I and N in both the mainline and the subprogram. These are NOT the same variables, they use different memory locations to store their values. Variable names are local to their respective program. Therefore if a variable name is used in a mainline for one purpose, the same variable name could be used in a subprogram for a completely different purpose since each subprogram is a self contained unit.

SAMPLE 22

The function below will calculate powers (X^y) . This is not one of the built-in functions supplied with Pascal.

```
FUNCTION Exp(X:REAL,Y:REAL):REAL;
  VAR
    I:INTEGER;
  BEGIN
   Exp := 1;
   FOR I := 1 TO Y
       Exp:=Exp * X
  END;
```

SAMPLE 23

The following function accepts a character as a parameter and returns a BOOLEAN result (true or false) which indicates whether or not that character was a letter.

```
FUNCTION Letter(Ch:CHAR):BOOLEAN;
  BEGIN
   Letter:=(Ch>='A') AND (Ch<='Z')
  END;
```

Since the above function returns a BOOLEAN result, it can be used in statements such as:

```
          IF Letter(Character) THEN ...
```

FUNCTION LIMITATIONS

Functions are limited in that they can only return a single computed value. Often we need to have subprograms that can return more than one result, or even subprograms that do not return any results such as just printing. Procedures can be used for these purposes.

PROCEDURES

Procedures are very similar to functions. However, a procedure identifier is not assigned a value and therefore does not have a type. Also a procedure can NOT be referenced in an expression. Instead a separate "procedure statement" is used to call it.

SAMPLE 24

```
PROGRAM BarGraph;

{THIS IS A PROGRAM TO READ IN INTEGER VALUES, ONE PER LINE, AND
PRINT A HORIZONTAL BAR GRAPH OF STARS}

VAR
  Number:INTEGER;
  More:CHAR;


PROCEDURE PrintGraph (Size:INTEGER);
  {PROCEDURE TO OUTPUT A LINE OF 'SIZE' ASTERISKS}
  VAR
    I:INTEGER;
  BEGIN
    FOR I:=1 TO Size DO
        WRITE('*');
    WRITELN
  END;


BEGIN {MAINLINE}
More := 'Y';
WHILE UPCASE(More) = 'Y' DO
  BEGIN
  WRITE('Enter the number to graph ');
  READLN(Number);
  WRITE(Number:5,' ':3);
  PrintGraph(Number);
  WRITE('More data y/n ');
  READLN(More)
  END {WHILE}
END.
```

The general form of a procedure is:

```
     PROCEDURE procedure name (formal parameter list);
        local declaration section
      BEGIN
        procedure body
      END;
```

Note: If there are no parameters, the formal parameter list and the parentheses should be omitted.

SAMPLE 25

The following is a series of programs and subprograms which call each other. The procedure PRINTFANCY accepts a positive integer value and outputs it as:

```
            ******
            *1234*
            ******
```

The box of asterisks surrounding the number are automatically adjusted to the correct size. THe mainline inputs the number and calls the procedure PRINTFANCY. Printfancy calls the function NumDIGITS to determine the number of digits in the number. NumDIGITS in turn calls the function LOG, which calls the built-in function LN. The procedure PRINTFANCY also use the procedure PRINTGRAPH to produce the two lines of asterisks.

```
{SAMPLE 25}
PROGRAM FancyStuff;

VAR  Number:INTEGER;

FUNCTION Log(X:REAL):REAL;
  BEGIN
  Log := LN(X)/LN(10.0)
  END;
```

con't next page

```
FUNCTION NumDigits(N:INTEGER):INTEGER;
  BEGIN
    IF N = 0 THEN
       NumDigits := 1
    ELSE
       NumDigits := 1 + TRUNC(LOG(ABS(N)))
  END;

PROCEDURE PrintGraph(Size:INTEGER);
  VAR
    I:INTEGER;
  BEGIN
    FOR I:=1 TO Size DO
        WRITE('*');
    WRITELN
  END;

PROCEDURE PrintFancy(N:INTEGER);
  VAR
    Size:INTEGER;
  BEGIN
    Size:=NumDigits(N);
    PrintGraph(Size+2);
    WRITELN('*',N:Size,'*');
    PrintGraph(Size+2)
  END;

BEGIN
WRITE('Enter the number ');
READLN(Number);
PrintFancy(Number)
END.
```

NOTE: It is important that the subprograms be declared in the correct order. A subprogram can ONLY call other subprograms that have previously been declared.

Procedures and functions are two very similar types of subprograms. Functions are used whenever a single result must be returned, and the function call is used only in an expression. Procedure calls may return many or no results, and procedure calls appear as a separate line. The READ, READLN, WRITE, WRITELN statements are actually built-in procedures, in the same way that SQRT, ABS, ROUND are built-in functions. These built in procedures have some special characteristics which user defined procedures do not have (such as variable number of parameters, field width specifications, etc). The next lesson expands the use of procedures with the use of "variable parameters".

RULES FOR PARAMETER LIST

1) There must be the same number of actual parameters (calling program as there are formal parameters (subprogram).

2) The data type of each actual parameter and its corresponding formal parameter must be the same.

3) For variable parameters; an actual parameter corresponding to a variable parameter must be a variable.

ASSIGNMENT LESSON 8

1) Write a program that will read in a positive real number and then determine and output the number of digits it has to the left of the decimal point. (Hint: One method is to repeatedly divide the number by 10 until it becomes less than 1). Test the program with the following data:

4702.456
0.35
12.34
5432
500000
0.01

2) Write a program that uses procedures to input data and then determines the range of the data (smallest value & largest value) and also determines the average.

3) Write a program (use subprograms) that will input a positive integer and will determine all the divisors of the number. If the number has no divisors output a message that it is a prime number.

LESSON 9

VARIABLE PARAMETER SUBPROGRAMS

A method is needed that will allow PROCEDURES to return data to the "calling program" (mainline or subprogram which referenced the procedure). The methods so far discuss, functions returning only a single result in the function name, and procedures not returning any result. Variable parameters allow multiple results returned.

If a parameter group in a parameter list is preceded by the reserved word VAR, each parameter in the group becomes a "variable parameter".
When the word VAR is not used it is called a "value parameter".

All the parameters used so far have be value parameters. For example:

```
         (A:REAL;VAR B,C:CHAR;D:INTEGER)
```

The parameters A and D are value parameters and the parameters B and C are variable parameters.

The Difference Between Variable Parameter and Value Parameter

Remember that a variable identifier (name) simply refers to a memory location where data can be stored. When we use a variable identifier we refer to the data stored in that memory location. When a variable parameter is used, the corresponding actual parameter (calling program) or argument must be a variable identifier. It can not be an expression. Instead of passing the value stored in the corresponding argument to the subprogram, the memory location is passed to the subprogram. The subprogram then accesses that memory location directly to change the data in the variable.

SAMPLE 26

```
PROGRAM Swap;
{SAMPLE 26}
{EXAMPLE OF THE USE OF VARIABLE PARAMETERS}

VAR
  Num1,Num2,Num3:INTEGER;

PROCEDURE Switch(VAR A,B:INTEGER);
   VAR
     Temp:INTEGER;
   BEGIN
   Temp:=A;
   A:=B;
   B:=Temp
   END;

BEGIN {MAINLINE}
WRITELN('Enter three numbers');
READLN(Num1,Num2,Num3);
Switch(Num1,Num2);
Switch(Num2,Num3);
WRITELN(Num1:5,Num2:5,Num3:5)
END.
```

Assume data
```
3  5  7
```

Notes on sample 26:

The procedure Switch has two variable parameters A and B. The statements of this procedure exchange the values stored in A and B.
The first time Switch is called the statement Switch(Num1,Num2) is used. If A and B were value parameters, then the values stored in Num1 and Num2 would be passed to the procedure. However since A AND B are variable parameters, the memory locations of Num1 and Num2 are passed to the procedure. Whenever the variables A and B are used they refer to the same memory locations as Num1 and Num2. Therefore the values in these memory location are exchanged. Num1 will contain 5 and Num2 will contain 3.

The second time Switch is called the memory locations of Num2 and Num3 are passed to the procedure. When the procedure returns to the mainline the values in Num2 and Num3 are 7 and 3. The output of this program would be:

```
  5    7    3
```

The procedure Switch can be used to exchange any two integer values stored in memory. Therefore the statement

```
        Switch(Vector[J],Vector[J+1])
```

is valid.

However, the statement

```
        Switch(Num1 + 4, 5*3)
```

is not valid.

NOTE

1) Value parameters are passed a value from the calling program. The corresponding actual parameter (argument) may be an expression or variable.

2) Variable parameters are passed the location of a variable in memory. The parameter refers directly to that variable. The corresponding actual parameter (argument) must be a variable.

If the SWITCH procedure was used with value parameters instead of variable parameters, (the keyword VAR in the PROCEDURE SWITCH was removed) it would not work the same.

```
      {SAMPLE 27}
      PROGRAM Swap2;
      {THIS PROGRAM SHOWS VALUE PARAMETERS}

      VAR
        Num1,Num2:INTEGER;

      PROCEDURE Switch2(A,B:INTEGER);
         VAR
           Temp:INTEGER;
         BEGIN
         Temp:=A;
         A:=B;
         B:=Temp
         END;

     BEGIN {MAINLINE}
     WRITELN('Enter three numbers');
     READLN(Num1,Num2,Num3);
     Switch(Num1,Num2);
     Switch(Num2,Num3);
     WRITELN(Num1:5,Num2:5,Num3:5)
     END.
```

Assume the same data

```
   3    5    7
```

The output would be

```
   3    5    7
```

Since value parameters are used instead of variable parameters the values stored in Num1, Num2, Num3 always stay the same.

NESTED SUBPROGRAMS

SAMPLE 28

```
PROGRAM Printing;
{SAMPLE 28}
{ THIS PROGRAM WILL INPUT A SET OF STRINGS OF VARYING LENGTH
   AND WILL OUTPUT EACH STRING CENTERED ON THE PAPER.}

CONST
     StringWidth = 80;
     MaxList = 50;
TYPE
     MyString = ARRAY[1..StringWidgth OF CHAR;
     List = ARRAY[1..MaxList] OF MyString;
     SizeString = 0..StringWidth;
     SizeList = 0..MaxList;
VAR
     ReadStrings:List;  {THE STRINGS TO BE INPUT}
     NumStrings:SizeList;    {NUMBER OF STRINGS TO INPUT}
     C:SizeList;            {A LOOP COUNTER}

PROCEDURE WriteString(VAR Line:MyString);
  {PROCEDURE TO OUTPUT A STRING CENTERED ON THE PAGE}
     VAR
          Size:SizeString; {SIZE OF THE STRING}
          LBlank:SizeString;  {LEADING BLANKS }
          C:SizeString;    {LOOP COUNTER}

  FUNCTION Length(VAR Line:MyString):SizeString;
    {CALCULATE SIZE OF THE STRING}
     VAR
          C:SizeString;  {LOOP COUNTER}
          Found:BOOLEAN; {FOUND FLAG}
  BEGIN
     C := StringWidth;
     Found := FALSE;
     WHILE (C>0) AND NOT Found DO
          IF Line[C] = ' ' THEN
               C := C - 1
          ELSE
               Found := TRUE;
     Length := C
  END; {FUNCTION LENGTH}

  BEGIN {PROCEDURE WRITESTRING}
     Size := Length(Line);
     LBlank := (StringWidth - Size) DIV 2;
     WRITE(' ':LBlank);
     FOR C := 1 TO Size DO
```

```
          WRITE(Line[C]);
  END; {PROCEDURE WRITESTRING}

PROCEDURE ReadString(VAR Line:MyString);
  {PROCEDURE TO READ ONE STRING OF INPUT
    AND PAD IT WITH BLANKS TO FILL THE ARRAY}
     VAR
          C:SizeString;      {LOOP COUNTER}
          Length:SizeString; {NUMBERS OF CHARACTERS INPUT}
  BEGIN
     Length := 0;
     WHILE (Length<StringWidth) AND NOT EOLN DO
          BEGIN
          Length := Length + 1;
          READ(Line[Length])
          END;
     READLN;
     FOR C := Length + 1 TO StringWidth DO
          Line[C] := ' '
  END; {PROCEDURE READSTRING}

BEGIN {MAINSTRING}
NumStrings := 0;
WHILE (NumStrings < MaxList) AND NOT EOF DO
     BEGIN
     NumStrings := NumStrings + 1;
     ReadString(ReadStrings[NumStrings])
     END;
FOR C := 1 TO NumStrings DO
     WriteString(ReadStrings[C])
END. {MAINSTRING}
```

Recall that a subprogram is a complete and independent Pascal program. Since a program can contain subprograms, it is possible for one Pascal subprogram to contain smaller subprograms. This is called nesting of subprograms.

A Pascal program consists of a program header and a block. This block is known as the "global" block. Identifiers declared at the beginning of this block are known as the global identifiers. The global block may contain other smaller blocks as part of procedure and function declaration. The identifiers in these smaller blocks are called "local" identifiers. It is possible for the same identifier name to be used in several different blocks in a program. The "scope" of an identifier is the set of blocks where that identifier can be used.

The Rules Governing Scope

1) When an identifier is declared in a block, or in the header that precedes the block, it may be used anywhere within that block (nested blocks included), and only within that block.

2) If an identifier which was declared outside a block is re-declared inside the block, the new meaning of that identifier applies throughout the block.

Sample 28 contains four different blocks. The global block contains declaration for eleven different global identifiers. The CONSTant identifiers StringWidth and MaxList, the TYPE identifiers List, SizeString, and SizeList, and the VARiable identifiers ReadStrings, NumStrings, and C, and the PROCEDURE identifiers WriteString and ReadString are all global identifiers. Since these are global they may be used anywhere within the block, including the nested procedures WriteString and ReadString.

The procedure WriteString has five local identifiers declared; the parameter Line, the variable identifiers LBlank, Size, and C, and the function identifier Length. Note that the global identifiers MyString and SizeString have been used. Also note that the identifier C has been redefined, This is the local identifier C. Outside this block the global variable C may still be used.

The function Length contains its own set of local identifiers as well. These identifiers are Line, C, and Found. Also the global identifers MyString and SizeString are used. Identifier C has been redefined for a second time.

A second procedure, ReadString is declared within the global block. Its local identifiers are Line, C and Length. The identifier C has been redclared. The identifiers Length and Line are used as well. The identifier Length was also used as a function identifier in the procedure WriteString. Since that function was declared inside the block WriteString, it may not be used from outsie that block. The function Length may therfore only be called from within WriteString. This means the procedure ReadString can use the identifier Length for something completely different. In this case ReadString uses Length as a variable identifier.

The following table lists the identifiers and their scope

Identifier	class	defined in	scope
StringWidth	constant	global block	entire program
MazList	constant	global block	entire program
MyString	type	global block	entire program
List	type	global block	entire program
SizeString	type	global block	entire program
SizeList	type	global block	entire program
ReadStrings	variable	global block	entire program
NumStrings	variable	global block	entire program
C	variable	global block	main line only
WriteString	procedure	global block	entire program
ReadString	procedure	global block	entire program
Line	parameter	WriteString	WriteString,Length
C	variable	WriteString	WriteString only
Size	variable	WriteString	WriteString,Length
LBlank	variable	WriteString	WriteString,Length
Length	function	WriteString	WriteString,Length
Line	parameter	Length	Length only
Found	variable	Length	Length only
C	variable	Length	Length only
Line	parameter	ReadString	READString only
C	variable	ReadString	READString only
Length	variable	ReadString	ReadString only

ASSIGNMENTS LESSON 9

1) Write a program that will convert Roman numerials to Arabic form (integers). The program should input a string (max 10 characters) in Roman numeral form and convert it to Arabic. Output both the Roman and Arabic numbers. The character values for Roman numerials are:

M	1000
D	500
C	100
L	50
X	10
V	5
I	1

Use the following test data:

LXXXVII	87
CCXIX	219
MCCCLIV	1354
MMDCLXXIII	2673
MCDLXXVI	?

2) Write a program that will convert Arabic numbers to Roman numerals. The opposite of the above.

3) Write a program to solve the following problem:
Compute the monthly payment and total payment for a bank loan, given:
1) the amount of the loan
2) the duration of the loan in months
3) the interest rate for the loan

Your program should input one record at a time (each record contains loan value, months value, rate value), perform the computations and output the values of the loan, months, rate and monthly payment and total payment.

1. The formula for computing monthly payments is:

$$\text{monthly} = \frac{\frac{\text{rate}}{1200.} \times \left(1. + \frac{\text{rate}}{1200}\right)^{\text{months}} \times \text{loan}}{\left(1. + \frac{\text{rate}}{1200.}\right)^{\text{months}} - 1.}$$

Question 3) cont

2. The formula for computing the total payment is

total = monthly x months

Hints: You may find it useful to introduce additonal variables to simplify the calculations of the monthly payments. You should output the values of RATEM and EXPM to check if your computations are correct.

```
ratem = rate/1200.
expm = (1. + ratem)
```

You will need a loop to multiply expm by itself months times.

4) Write a program which will scan a line of text and will replace all multiple occurances of blanks with a single blank.

5) Write a program that will input a sentence and output each word in the sentence and the number of letters in that word.

6) Write a program that will input a string of words and output a table giving the frequency of word length ie.

```
LENGTH OF WORD         NUMBER OF WORDS
   1  LETTER                3
   2  LETTERS               2
   3  LETTERS               6
   .    .                   .
   .    .                   .
```

Notes re: corrections errors

INDEX