THE OPEN UNIVERSITY
CSE 0002

Introductory Course in PASCAL PROGRAMMING

Prepared by the Computer Literacy Project Course Team
for the Board of Study for Management, Science and
Technology of the Open University of Sri Lanka.

AN OPEN UNIVERSITY PUBLICATION

1985.

THE OPEN UNIVERSITY

OF

SRI LANKA

INTRODUCTORY COURSE IN

PASCAL PROGRAMMING

CSE 0002

01-8505055-OUP-6000

COURSE TEAM

Mr. B.I.V.Jayatilaka

Professor H.Sriyananda

Mr. Arjuna de Zoyza

Mr. L.P.Ranatunga

Mr. H.G.V.T.Vithanage

Mr. D.D.Prabath

Mr. W.M.A.Bandara

Miss. C.A.Witharanage

COVER

Mrs. Rani Ponnamperuma

DATA ENTRY

Miss. A.L.Sujatha Swarnalatha

CONTENTS

	Page
Preface	v
Introduction to the Commodore 64 Key Board (with WATCOM Editor) and peripherals.	1
CHAPTER 1 An Overview of the Main Parts of A Computer System.	4
CHAPTER 2 Fundamentals of Computer Programming.	6
CHAPTER 3 Introduction to Pascal Language, Running Pascal from A Diskette, Editing the Program, Identifiers, Assignments and Expressions, Relational Operators, A Programming Example Controlling Output.	9
CHAPTER 4 Repetition; For, Repeat and While Statements, Decision Statement, the Case Statement.	22
CHAPTER 5 Procedures and Functions, Array Variables.	30
CHAPTER 6 Variable Type, Scalars, Subranges, Sets, Structured Types, Packed arrays, Boolean arrays, Records, The With Statement.	38
CHAPTER 7 Loading, Saving and Scratching Programs, Files, Text Files, Non-Text-Files.	50

02-8505055-OUP

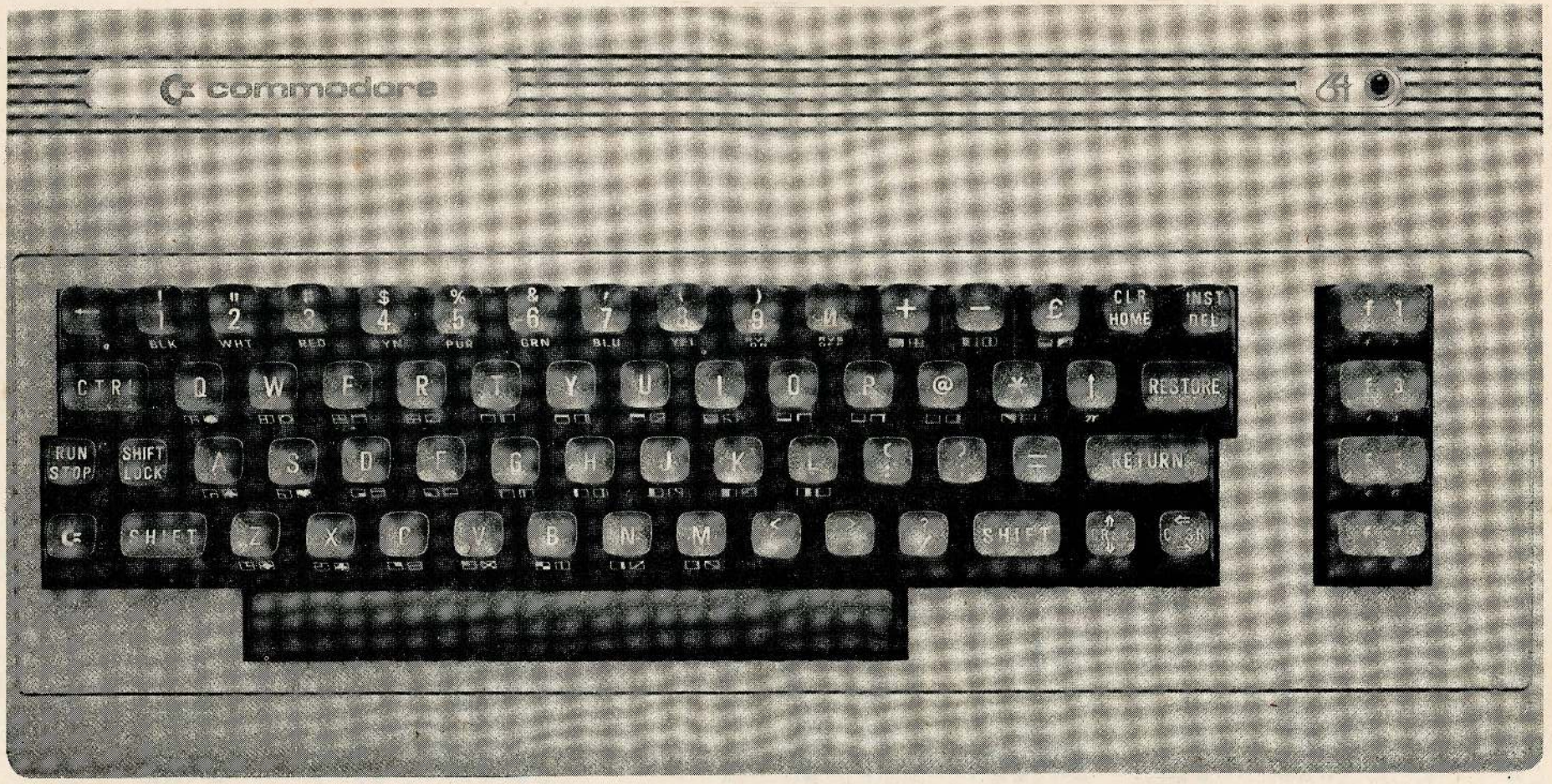
PREFACE

This book contains the lessons for the computer literacy course, being conducted at the Open University.

The course first instructs the student in the use of a Commodore - 64 Microcomputer with WATCOM Editor and peripherals and then proceeds to teach a programming language known as Pascal. The pascal language operating on the commodore is WATCOM pascal.

Further pascal start from chapter 6 and is provided with self assessment questions and explanatory answers. Although the implimentation of WATCOM Pascal do provide with extra facilities such as Direct access we restricted our course only to standard pascal.

The student at the end of this course should be able a operate the micro computer he has used and be able to write and understand simple programs in Pascal.



INTRODUCTION TO THE COMMODORE 64 KEY BOARD (WITH WATCOM EDITOR) AND PERIPHERALS

i. Introduction :

The computer (Commodore 64) you see in front of you, consists of three units;

(1) The CPU (Central Processing Unit)
This is the typewriter type keyboard you see in front of you. The casing contains the electronics which drives the computer and its memory for storing information during its operation.

(2) The Monitor (TAXAN)
This is the screen you see in front of you. Any entry on the keyboard will be displayed on the screen, and the computer will 'answer' back to you by displaying on this screen.

(3) The Floppy Disk Drive Unit
This is the rectangular grey unit which lies on your right hand side. The diskettes (Black square shaped 'disks' stored inside the paper cover), are loaded into this unit. The diskettes are used to store programs, data and compilers and will be necessary for you to use.

ii. Starting the Computer :

To start the computer, follow these instructions:

(a) Switch on the power at the plug points. The plug base should be on the table.

(b) Switch on the three units, starting with the Monitor, the Disk Drive, and then the CPU - keyboard.

After a few seconds, the following will be displayed on the screen.

< beginning of file >
< end of file >

WATCOM Editor V2.0
Copyright 1984 WATCOM Systems Inc.

* There is a small blinking square at the bottom of the screen; this square is called a CURSOR. Now type the command "INPUT" and press RETURN key. A blank line will appear after the statement < beginning of file > on the screen and the cursor will be placed at the beginning of this line. You are now ready to start.

iii. Keyboard :

Before you actually start any serious work it is necessary to be familiar with the keyboard. Observe the keyboard and note the position of alphabetic letters, numbers,numerical signs etc. There are two important keys that need to be explained;

(1) The SHIFT Key :
These are two identical keys placed on either side of the keyboard. When the SHIFT key is pressed together with another key you obtain the captal letters (upper case characters). Try pressing SHIFT and number 1 key together. Try a few others.

* Please Note that this section can be understood only by operating a Commodore 64 Computer.

(2) The RETURN Key :
Pressing the RETURN key makes the computer accept the typed command. Type in the following statement My first Pascal Program and the press RETURN key. A new blank line will be created.
Try typing anything at random and pressing RETURN. To ENTER what has been typed is the as pressing RETURN.

There are some other keys which might be of special interest to you. Try these,

(1) Press SHIFT/LOCK and then some other keys. Press SHIFT/LOCK again to get out of the upper case.

(2) Press the two keys CRSR and CRSR, alone and then together with the SHIFT keys. These two keys are used to move the flashing square (cursor) around the screen. Note:You will notice that one line of the display is highlighted; This line is referred to as the CURRENT LINE.

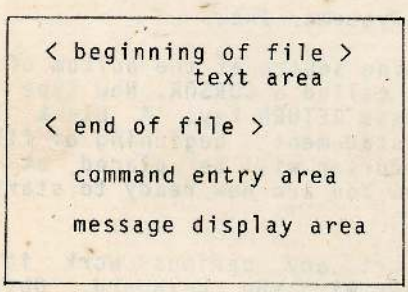
(3) Type in something and press INST/DEL.

(4) Move the CURSOR using the CRSR keys to a position in between some characters and press SHIFT together with INST/DEL. Now press an alphabetic (A---> Z) Key. You can insert characters in this manner.

The workspace between the < beginning of file > and the < end of file > is called the text area, and the cursor may be moved anywhere in this area to modify text. This mode of operation is called SCREEN MODE and we say Editor is in screen mode.

iv. Special Function Keys :

The four keys labelled f1 , f3 , f5 , and f7 are called FUNCTION KEYS. These keys are used to tell the editor to perform a particular function.
There are actually eight functions which can be used. The other four are requested by pressing these same keys while holding down the SHIFT key.



(1) The F2 key (SCREEN/COMMAND)
While holding down the shift key press the key labaled F1. Then F2 will function and If the Editor is in screen mode then the cursor will move to the bottom of the screen; and this space at the bottom of the screen is called the COMMAND AREA. This mode of operation is called COMMAND MODE Now Press F2 again the cursor will move to the beginning of the highlighted line (i.e. Editor is in screen mode)

Hence F2 switches the editor between "command" and "screen" modes.

(2) The F4 key (LINE INSERT)
Press F4 when in screen mode, This will adds a blank line after the line on which the cursor is positioned. You may insert extra lines by using the key F4. So type something on this line.

(3) The F6 key (LINE DELETE)
Press F6 and see what happen. It deletes the line on which the cursor is positioned (when in screen mode). Hence you have to use this key carefully.

(4) The F8 key (HELP)
Press F8. It displays the functions assigned to each special function key as well as a list of the available commands. Press RETURN key to return to editing mode.

(5) The F1 key (PAGE UP)
Pressing F1 causes the previous screenfull of lines in the file to be displayed on the screen.

(6) The F3 key (LINE UP)
Pressing F3 causes the line before the current line to become the current line. ("UP" means "towards the beginning of the file").

(7) The F5 key (LINE DOWN)
Pressing F5 causes the line following the current line to become the current line ("DOWN" means "towards the end of the file").

(8) The F7 key (PAGE DOWN)
Pressing F7 causes the next screenfull of lines in the file to be displayed on the screen.

(9) To delete all lines type *delete (or simply *d) in command entry area and press RETURN key.

Exercise : Type the following sentences in text area.

Micro Computer Laboratary,
Nawala, Nugegoda.

Now try to insert "The Open University" in between above two lines.

v. Some Key Words :

CPU (Central Processing Unit - Main Unit in which computations and operations are carried out.

Monitor - Screen display.

Floppy Disk Drive - The drive unit for the Floppy Disk.

Floppy Disk - A soft disk which stores, computer programs and Data outside the Main computer (a peripheral device).

Character - A letter, number, graphics symbol or any other symbol which can be displayed.

To Enter - Means to press RETURN and make the accept your input.

In Screen Mode - the cursor appears in the text area.

In command mode - the cursor is in command area.

1.0 An Overview of the Main Parts of A Computer System :

This chapter is intended to show you the main parts of a computer and how these parts interact with each other.

Any system can be viewed as a black box which gives a response (an output) to a given input.

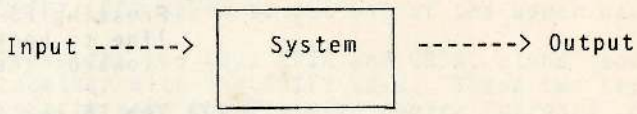


Fig. 1.1

A simple example of a system is an electric torch. When you move the switch to on position, it will give its response, light (if everything else : batteries, bulb etc. is ok).

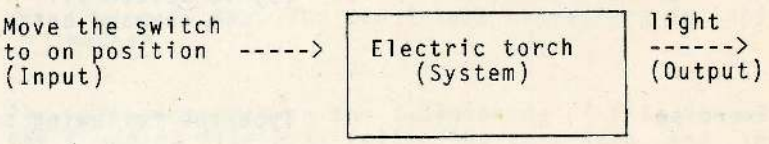


Fig. 1.2

You can look at a computer system also in a similar manner.

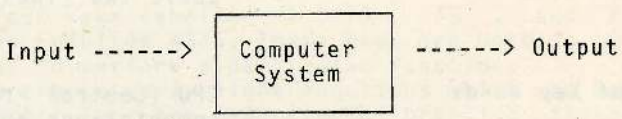


Fig. 1.3

If we look into the black box further, there are main functional parts.

A computer system has (i) a unit which does the control, arithmetic, and logical operations (processing) (ii) memory, (iii) input device through which you enter your input (iv) output device on which it displays its responses.

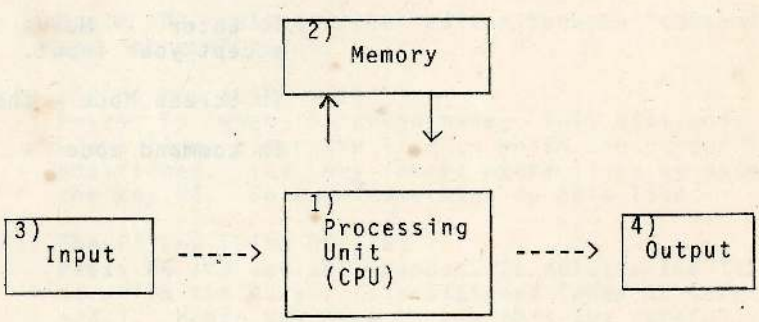


Fig. 1.4

In the computer system you are using, the major input will be from the keyboard. Main output will be the display screen { (i) CPU and (ii) memory cannot be seen from outside}.

In addition to the input/output mentioned above, there is another device which can serve as both an input and an output. That is the diskette unit.

The programme (see next chapter) which you run on the computer resides in the memory. CPU gets instructions from this program in the memory and sometimes CPU may write to the memory (that is why there are two arrows between CPU and memory in the figure 1.4).

CHAPTER 2

2.0 Fundamentals of Computer Programming:

We look at a computer system as a black box in the the previous chapter.

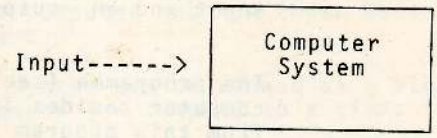
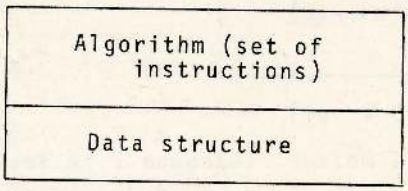


Fig. 2.1

It is possible to "make" the computer to respond in a manner you need by programming it. By running a correct computer programme on the computer it will perform operations in the manner specified by the program. The program will specify the instructions to perform operations on your data.

Let us consider a simple program to add a set of numbers. By means of the program you "tell" the computer that numbers need to be added giving their sum as the output. In this example, the numbers which you want to add can be called DATA.

A program consists of two identifiable parts.



An algorithm and a data structure can be called a program. Algorithm is the set of instructions to do the required operations. Data structure defines how data has to be stored in the memory.

When we communicate with each other we use a natural language as English language etc.. Similarly, there are programming languages that can be used to communicate with the computers. Some of these programming languages are ;

BASIC
COBOL
PASCAL
FORTRAN.

As there are a set of rules to govern the correctness of a statement in English language, there is a set of rules in each programming language. If you stick to these rules your programme will be "correct" and can be run on the computer. This does not mean that the programme will do what is really required.

If you follow few basic steps it will be possible to write programmes that will function in a required manner.

STEP 1 : Understand the problem. Read (and think about) the problem carefully and understand what needs to be done to get the desired result.

STEP 2 : Write down, in a natural language (English, Sinhalese etc.)familiar to you, the steps needed to solve the problems and to get the desired result.

STEP 3 : Translate this procedure (of steps written in STEP 2) to the statements in the programming language you want to use.

Let us look at a simple example to see how the steps 1 and 2 can be followed to write a programme.

Problem 2.1 : Find the average of a given set (collection) of numbers.

Step 1 : Read the problem carefully. It is required to find the average of given numbers. For example, if there are three numbers : 3,5,7, how would you set about finding the average?

First find the total = add all three numbers
= 3 + 5 + 7

Then divide the total by 3 (the number of terms given)

average = total -:- 3

Note:- In the above example, the numbers 3, 5, 7 can be called a set of numbers and there are three numbers in this set.

Generally, in order to find the average of a set of numbers you should know (i) the values of the numbers (in the example, 3,5,7) and (ii) how many numbers are there in the set (in the example, 3). Suppose the values and the number of values are written on a paper, then you can read the values and number of values.

Step 2 : If you want to solve the problems manually you would follow the following procedure.

Procedure 2.1 : Read how many numbers are there,
Read the numbers,
Add the numbers to get the total,
Divide the total by the number of numbers to get the average.
You can write procedure 2.1 in another way.

When reading the numbers we can read one number at a time. Suppose there are (ten) balls and an empty sack,we can find the total number of balls by adding one ball at a time to the sack.

Similarly, in the given problem if we consider memory (of the computer) as the sack we can read one number and add that number to memory, and read another number and add it to memory and so on. At the end memory will contain the total of the numbers. If we label memory as 'TOTAL' the procedure 2.1 can be re-written as the following:

Procedure 2.2 : Read how many numbers are there (suppose N).

05-8505055-OUP

Step x : Read a number (Y)
Add the number to get the total (add Y to TOTAL).
Count the number read (suppose this count is K).
If K is less than N then start doing from the step x again.
If x is equal to N then find average = TOTAL -:- N .
Write average .
Stop.

As you are aware of now, to follow the first two steps in writing a computer program, you do not need to know any programming language.

But the computer we have cannot " understand" a natural language as English. Therefore, you have to translate what is written in the step2 above to a language that can be "understood" by the computer.

CHAPTER 3

3.0 Introduction to Pascal Language

Starting from this chapter, you will find the basics of a programming language called PASCAL.

Pascal is a programming language which is easy to learn and easy to use. Furthermore, it is easy for you to understand a Pascal program written by someone else. Let us look at a simple Pascal program.

```
    PROGRAM sum (Input, Output);
    VAR
         x : real;
1        y : real;
         z : real;

    BEGIN
      read  (x) ;
      read  (y) ;
2          z  := x + y;
      write (z)

    END.
```

program 3.1

There are three words in capital letters in program 3.1. Those words should always be present in a Pascal program. The first word in program 3.1 is PROGRAM. Any Pascal program should begin with this word. In program 3.1 the word 'sum' follows the word PROGRAM. 'sum' is the name given to the programme.

Programme name is the name you want to identify the programme with. You can name your programme using any name you like to use. Best name is the one which will convey an idea of what the programme does.

The word "Input " is written to indicate that the program will read data into the program. The word "Output" indicates that the program will send data out of the program.

We can say that the general format of PROGRAM statement is the following:

PROGRAM programme name (Input,Output);

Notice the semicolon (;) at the end of the statement. Almost all PASCAL statements should end with a semicolon.

In chapter 2, we mentioned that there are two sections in a program. One is the data structure, the other is the algorithm part. In programme 3.1 the part identified by (1) defines the data structure used by the program while (2) contains the algorithm. We will be looking at data structure (part (1)) in in detail later.

In section (2) of the program 3.1 there are two words which are in capital letters. They are: BEGIN and END. BEGIN should be the word at the begining of the algorithm section ('executable' statements) and END should be at the end of the program.

Other than at the beginning of a program, BEGIN and END could be used to group statements together so

that they can be treated as one statement in some cases (see the chapter on Repetition and Decision, eg. FOR loop, WHILE loop etc).

3.1 Running Pascal from a Diskette :

Check whether the green light at the front of the disk drive unit is on and the front panel is up or else make them so. The diskette which is provided is a flat rectangular plate, black in colour, with PASCAL written on one side. Place the diskette with the written side facing upwards. Look for a notch (⌋) on one side. Gently push the diskette into the drive unit with the lettering side facing upwards and the notch on your left hand side. After inserting completely pull the front cover downwards.

You are now in a position to use the disk drive unit. Clear the workspace bt typing *delete (or *d) in command entry area, and pressing the RETURN key.

Now type

```
load pascal
```

and then press the RETURN key. The screen will go blank for about one and a half minutes while pascal is being loaded from the diskette.

DO NOT ATTEMPT TO REMOVE DISKETTE WHILE THE RED LIGHT OF THE DISK DRIVE IS ON.
Once pascal is loaded the following will appear on the screen.

```
< beginning of file >
< end of file >

WATCOM pascal V2.0
Copyright 1984 WATCOM systems Inc.
```

Pascal is now ready for use. Type the command "Input" (or i) and press RETURN key. The Editor is in screen mode and you can now enter a program and run it. Type in the earlier described program SUM. Remember to press RETURN key after typing each line.

3.2 Editing the Program :

Have you made any mistakes, if you have, you can change these by using the SPECIAL FUNCTION KEYS (and the CRSR keys) which you have learnt in the first Lesson.

We shall introduce some extra facilities here. IF YOU ARE TO USE THEM YOU SHOULD TYPE THEM IN THE COMMAND ENTRY AREA.

To DELETE lines :

Syntax:

```
< line number range > delete (or < line number range > d)
```

The delete command delets all lines in the specified line number range.

Example:

```
1,5d  delete lines 1 through 5 inclusive.

*d    delete all lines.
```

To INSERT lines:
Syntax :

```
< line number >I
Inserts after line number.
```

Example:

```
3i  Insert after the third line.
 i  Insert at the current line
```

* TO CHANGE lines:

Syntax:

```
< line number range >C/old/new
```

The computer will find all text "old" and change to 'new' between the specified line number range.

Example:

Supose you had speld real to be reel, enter

```
* 3,5C/reel/real
```

This will change reel to real wherever it occurs between lines 3 and 5.

```
* * C/reel/real
```

change all occurences of reel to real.

If you are satisfied with the program display type run in command entry area and then press the RETURN key to run the program. If you have made any mistake the screen will appear as follows:

```
Execution begins.....
*** Error :  (some error messages will be
              displayed)***

Debug?
```

The computer has entered debug mode which provides a number of facilities to help you to determine what is wrong with your program. We do not use debug mode in this lesson and so you should type q to quit. Then press the RETURN key to return to the editor. You may now correct the program as we have already discussed.

If you have no error messages, you have successfully 'complied' your program. To 'compile' is to translate from the source language (Pascal in this case) to machine language.

Machine language is a set of instructions which the computer can operate on directly.

The computer will type Execution begins.....wait for you to enter a number , at this state it is operating on the read (x); statement. Enter a real number. It will then wait again for y. another number. The sum of the two numbers will appear underneath your entries.

You have now successfully written, edited and run a program.

Data is an important part of a programme. All the operations of a programme has to be done on data. We will see later how data can change the execution of a program.
There are two kinds of data. They are constant data and variable data. Before looking at these kinds of data lets look at another useful construct, called identifiers.

3.3 Identifiers :

So far we have seen where to use the words PROGRAM, BEGIN and END. These are called reserved words in Pascal. These words cannot be used freely.

Other than these words, you can define your own symbols. They are called identifiers. An identifier should be a word formed by a letter followed by one or more letters and/or digits.

Examples of identifiers:

```
length
area2
weight12
a562
```

The following are examples of invalid (incorrect) identifiers :

```
2force
strain.k
amp i
```

2force - this starts with a digit (2); therefore, it is not valid.(an identifier should start with a letter) Strain.k -contains a character which is neither a digit nor a letter.

amp i -contains a blank (which is neither a letter nor a digit).

A note about deciding on identifiers:- It is better to select an identifier according to its meaning. For example, if you want to use an identifiers for a force better to use force as an identifier instead of using banana.

3.4 Constant Data :

Constant data remains unchanged throughout the execution of the programme.

You can call any constant value that remains unchanged as a constant. For example, 70 is a constant. Other than the constant values which you use inside a programme you can also declare constants.

This decleration has to be done at the beginning of a program.

```
CONST
   nine = 9;
  cycle = 10 ;
```

The word CONST has to be given before the decleration of the constants.

Now in the program you can use,

```
cycle
instead of 10 , and
nine
instead of 9.
```

In order to declare constant data you can use the statements in the following format.

```
CONST
   identifier = literal
   identifier = + another constant
   identifier  (which has to be declared).
   identifier = - another constant identifier
                  (which has to be declared).
```

A literal could be a string of characters within single quotation marks(for example, 'complete result'), or a numerical value (we will be dealing with numerical values in details later, for the time being, examples are: 70 80.354 etc).

Let us declare some constant data according to the above definitions.

Example,

```
CONST
   gravity = 32 ;
   thrust  = -gravity ;
   force   = thrust ;
```

Are these declerations correct? Compare against the definitions.

(i) declarations begins with the word CONST (it is correct)

(ii) in gravity = 32
gravity is a valid identifier and 32 is a valid numerical value (therefore, this is also correct !).

(iii) thrust = - gravity ;
thrust is a valid identifier and, gravity is a constant identifier which was declared by the previous statement (therefore, this is also correct).

(iv) force = -thrust ;
force is a valid identifier and thrust is a valid constant identifier which was declared previously (therefore, this is also correct !!)

Therefore the declarations are correct

Exercise : Find out which of the following are correct and which are incorrect, identifiers.

a
1st sum
kg
size
step size
lay-by

3.5 Variable Data

The other kind of data is variable data. Varaibles can be given different values in the program.

For example, consider a variable x, then at one point in the program, x can be given the value 5 :

```
x : = 5 ;
```

At another point the same variable x can be given the value 13 :

```
x : = 13 ;
```

Variables also has to be declared at the beginning of a programme.

Types :

Before looking at how to declare variables let us learn about the different types of variables (or data) in Pascal.

In Pascal, the standard types are :

integer
real
boolean
char .

Integer values do not have a fractional part (only whole numbers).
For example,

50 -7353 8

are integer values.

Real values may contain a fractional part also.
For example,

87.8 5.0 5.3532

are real values.

Boolean type can take only two values.
They are :

true
false

True is one boolean value.
False is the other boolean value.

The values
'a' 'b' '1' '=' are of the type char. Char values has to be enclosed within (single) quotation marks.

You have to declare every variable you use in the programme.
When declaring constant data the word CONST has to be written. Similarly, when declaring variable data the word VAR has to be used.

The following is a correct declaration of variables.

```
VAR
   index, count :  integer ;
   value, total,   force   : real ;
   eof,   exist  : boolean ;
   list,  string : char ;
```

In the above example :
The variables:

index
count are of type integer.

The variables:

value
total
force are of type real.

The variables:

exist
eof are of type boolean,

and the variables,

list
string are of the type char.

3.6 Assignments and Expressions :

It was mentioned that the value of a variable may be changed in the programme.

A value of a variable can be changed by assigning a value to it.

For example,

```
VAR
   x : real ;
BEGIN
   x := 14 ;
END.
```

In the above program,the variable x is assigned the value 14.

Let us look at the general format of an assignment statement The variable to be changed is written on the left side, then the symbol := has to be written, and then the value to be assigned. (note that the symbol for assignment is not just an equal sign but a colon (:) and an equal sign (=). The assignment statement has to be of the following format.

```
variable := expression ;
```

The right side of the assignment statement shows "what is the value " and the left hand side indicates " to what this value to be given". The expression on the right side can be: another variable, a constant , a numerical value, or an arithmetic (or logical) expression. Let us look at expressions in more details to see how they are formed.

3.7 Expressions :

An expression could be one variable. It was shown earlier (section) what is a variable. Let us look at an example. If the following declarations were made at the begining.

```
VAR
    maximum : real ;
    minimum : real ;
```

Then the following is a valid assignment statement.

```
    minimum : = - maximum ;
```

You can use a single numerical value or a constant also as an expression.

For example,

```
      maximum : = 200.7 ;
```

An expression could be a valid arithmetic or logical expression. Let us see how to form a valid arithmetic expression.

We all know, that there are four standard operations. They are : addition, subtraction, multiplication, division.

In Pascal specific symbols are used to indicate these operations.

addition indicated by +
subtraction indicated by -
multiplication indicated by *

For divisions there are two different operators for the two types of data (integer and real).

Division operator when using real data is / . You can use / with integer operands (variables or constants) but the result will be real.

If the following real variables are declared :

```
VAR
   cycles    : real ;
   time      : real ;
   frequency : real ;
```

Then the following statement is correct :

```
   frequency : = cycles / time ;
```

The division operators for integers are DIV and MOD. Let us see why we need two operators to perform division on integer data. Integers are only whole numbers. If we divide one integer number by another integer number, we may get a value with a fractional part also.

(For example, 3 divided by 2 is 1.5) Suppose we have the following declaration.

```
VAR
   number : integer;
```

Then if we write the following statement :

```
   number  : =   5 DIV 2 ;
```

We will get the value of 'number'as 2. DIV will give the quotient.

If we write the following statement:

```
   number  : =   5 MOD 2 ;
```

We will get the value of 'number'as 1. MOD will give the remainder of the division.

Now we have seen how to represent the basic arithmetic operations. Let us see how we can write correct arithmetic expressions.

Suppose we have to write an arithmetic expression for the algebraic expression:

$$u + f.t$$

If we make the following declerations for the variables:

```
VAR
   u,f,t : real ;
```

Then the corresponding expression should be

```
u + f * t
```

Precedence of the Operators :

If there are several operators in an expression, how will the computer evaluate the expression ?

For example, (if we use constant values)

```
2 + 3*4
```

If there is no precedence given to the operators there are two possible values for the above expression. They are 20 or 14 (if addition is done first then 2 + 3 * 4 = 5 * 4 = 20, if multiplication is done first then 2 + 3 * 4 = 2 + 12 = 14).

Fortunately, Pascal solves this ambiguity by giving an order of precedence to the operators.
In Pascal, multiplication and division is done first then additions and subtractions.

In the above example, the value will be definitely 14 (multiplication done first 3*4 =12, then addition: 2 + 12 = 14).

Let us look at few more examples on arithmetic expressions.

The use of paranthesis expressions :

It is possible to change the order of operations by using paranthesis. In the example;

```
2+3*4
```

We can make the addition to be done first in the following way:

```
(2+3)*4
```

The result of this expression will be 20. The paranthesis has the precedence over all the other operators. The part of the expression inside paranthesis will be evaluated first. Therefore, you can use them to change the order of evalutaion (or the value) of an expression.
If we have both divisions and multiplications then the expression is evaluated from left to right.
For example,

```
10 / 2 * 5
```

will be evaluated as (10/2)*5 yielding

```
5*5 (= 25).
```

3.8 Logical Expressions :

We have seen how to write arithmetic expressions. Let us see now, how to form logical expressions.
There are three basic logical operators.

They are : AND
OR
NOT

The result of evaluating a logical expression could be either true or false.

Basic logical operations can be performed only on boolean type variables or boolean values. We have seen what are boolean type variables in a previous section (3.5).

If we have the following declerations

```
VAR
   exist, eof : boolean ;
```

We can write the following assignment statements :

```
exist := true ;
eof   := false ;
```

If we use the basic logical operators with these variables we can write few logical expressions.

```
exist AND eof
exist OR eof
NOT eof
```

The above are valid logical expressions.

Let us look at the meaning of each of the operators.

AND

As the word 'and' implies the AND operator gives the value of both the operands only.

Boolean type variables can take only two values.

They are : true
false

If we consider these boolean values the AND operator will give the following results.

```
true AND true      will be true
true AND false     will be false
false AND false    will be false
```

OR

OR operator will result in a true value if at least one of the operands (on which the operation is done) is true.

```
true OR true       will be true
true OR false      will be true
false OR false     will be false
```

NOT

NOT operator will negate a value.
For example,

```
NOT true           will be false
NOT false          will be true.
```

Let us look at the boolean variables and the logical expression we considered earlier.

```
VAR
    exist,eof : boolean;
BEGIN
    exist := true ;
    eof   := false ;
END.
```

If we have the above,then the value of exist AND eof will be false.
The value of exist OR eof will be true NOT eof will be true.

In addition to the logical operators there are some other operators that can be used in logical expressions. They are called relational (comparison) operators.

3.9 Relational Operators:

Relational operators are used to indicate relationships between variables.

The relational operators are

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

These operators will yield boolean values when used.

For example,

2 < 3	will yield the value true
2 > 3	will be false
5 <=5	will be true
6 > 10	will be false
5>= 5	will be true
5<>5	will be false

We can call the above expressions "comparative expressions". It is possible to use combinations of comparative expressions and logical operators (AND, OR, and NOT) to form logical expressions.

For example,
(2 < 3) AND (5 > 3) will be true.

This example shows how to form logical expression using both logical and relational operators.
The logical expression in the above example is :
(2 < 3) AND (5 > 3)

Brackets should be always used when comparative expressions are seperated by logical operators.

Precedence in logical operators :

NOT operator has the highest priority(it is applied first), then the AND operator has the next highest priority and the OR operator has the lowest priority.

For example,

```
VAR
   finish, exmpty,toobig : boolean ;
```

Then the expression NOT exmpty OR toobig is equivalent to (NOT empty) OR to toobig.

The expression finish AND empty OR toobig is equivalent to (finish AND empty) OR toobig.

3.10 A Programming Example :

Problem : -
Write a program to find the area of a circle when its radius is given.
Let us follow the steps given in Chapter 2.

Step 1 : Read the problem carefully and understant (what is given, what has to be done and what should be the result ?)

Step 2 : Write a procedure.
Read the radius (suppose r)
Area = constant pi x square of radius
write Area.

Step 3 : Write the Pascal programme from step 2.
Before starting to write the program let us look at how to read and write in Pascal.
(This will be described in detail later.

08-8505055-000

The following is a read statement in Pascal.

```
read (argument) :
```

If we want to read the values for three variables.

```
first
middle
last
```

We can write the following.

```
VAR
   first, middle, last : real ;
BEGIN
        read (first);
        read (middle);
        read (last );
END.
```

or, we can use one read statement, read (first, middle, last)

```
VAR
    first,middle, last : real ;
BEGIN
    read(first,middle,last);
END.
```

You can use more than one argument in read statements. Similarly, it is possible to 'write' in Pascal.

For example, to write the above three variables the statement,

```
write(first,middle,last);
```

can be used or else the statements,

```
write(first);
write(middle);
write(last);
```

can be used.

You can write character strings also.
For example,

```
write("Programming is fun")
```

or

```
CONST
      string = " Programming is fun";
      .......
      .......
      write (string);
```

Now let us write the program for the procedure we wrote in step 2. Enter the program in the computer and run

```
Program area(Input,Output);
CONST
     pi = 3.1428;
VAR
     area,radius : real ;
BEGIN
     read(radius);
     area := pi*radius*radius;
     write('area of the circle',area);
END.
```

Program 3.2

First, there should be the PROGRAM statement and the program is named as 'area' using this statement.

The words Input Output should follow in the program heading for reading and writing data.

Then a constant is declared for pi. The two variables 'radius' and 'area' are declared for radius and area of the circle.

Then the keyword BEGIN before starting the algorithm section of the program. First, read the radius using read statement, then calculate the area using an assignment statement. Then write the value for area.

3.11 Controlling output :

If we run the program 3.2 on the computer, with radius = 0.1, the output on the screen will appear as follows:

3.14280000e - 02

The output has been displayed using EXPONENT or SCIENTIFIC notation, usually referred to as E-NOTATION. The number following the "e" indicates the power of ten which is used to multiply the number which precedes it. For example 2.12000000e + 02 means 2.12 multiplied by 10 to the second power and is therefore equivalent to 212.

E-notation is useful for displaying very small or very large real number.

Pascal provides a way to control the format in which numbers are displayed, as illustrated below.

Modify the write statement in program 3.2 as

Write ("area of the circle", area: 8:2);

Now run the program and see how the output appears.

In the write statement two integers, seperated by colons(:), have been added after the variable name area. The first integer specifies the total number of character positions to be used to display the number. The second integer specifies the number of digits to be displayed to the right of the decimal point.

The notation area:8:2 indicates that the contents of the variable AREA should be displayed to 2 decimal places using a total of 8 character positions. Since the decimal point takes up one of the character positions up to 7 digits can be displayed; 5 to the left of the decimal point and 2 to the right.

CHAPTER 4

4.0 Repetition :

Sometimes, it is necessary to repeat a sequence of statements.

For example, consider the problem (of finding averages) given in chapter 2. Suppose that we know how many numbers need to be added before writing the program.

Then we can write a program as the following by using only the statements we have learnt so far.

```
CONST
    n = 5 ;
VAR
    x : real ;
    total,average:real;
BEGIN
    total := 0 ;
    read (x) ;
    total := total + x ;
    read (x) ;
    total := total + x;
    read(x) ;
    total := total + x ;
    read(x);
    total := total + x;
    read(x) ;
    total := total + x;
    average := total / N;
    write ("average =", average);
END.
```

Program 4.1

If you follow the statements in this program you could see that it finds average of N numbers (where N is set equal to 5 at the beginning).

But there are problems in writing the program in this way (i) you have to know the number of values (N) before writing this program. (As you can observe the statements

```
Read(x);
total := total + x;
```

have to be repeated in order to find the total of 5 numbers) (ii) If you need to change the number of values to 10 then you have to rewrite the program.

There are several ways to repeat a group of statements in a program without physically writing them several times. One of them is the FOR statement.

FOR statement :

The general format of the FOR statement is :

```
FOR variable : =
        expression1 TO expression2 DO statement ;
```

In this,the 'variable' is first assigned the value of expression 1 and the following statement will be executed. Next, the value of 'variable' will be incremented by 1 and the statement following will be executed. This process will be repe ated until 'variable' reaches the value of expression 2.

If expression 1 > expression 2 the statement will not be executed at all.

Example,

```
VAR
    I, N : integer ;
BEGIN
    FOR I := 1 TO N DO
    total := total + 1;
END.
```

The above FOR statement asks to execute the statement,

```
total := total + 1
```

N times (while the value of I is changed from 1 to N in steps of 1).

The variable of the FOR statement is called control variable. The values of expression1 and expression2 are evaluated only once at the beginning of the FOR statement. (therefore, you cannot change the limits of the FOR loop within the loop.

Let us write the program 4.1 using the FOR loop.

```
Program average (input,output);
   VAR
      N : integer ;
      I : integer ;
      x : real ;
      total,average:real ;
   BEGIN
      read(N);
      total := 0 ;
      FOR I := 1 to N DO
   BEGIN
      read (x) ;
      total := total + x ;
   END ;
      average := total / n;
      write ("average", average);
   END.
```

Now we have replaced 10 statements by 3 statements (actually by 1 compound statement). Type the program in and run it.

Instead of writing,

```
read (x) ;
total := total + x;
```

several times, we can use the following :

```
FOR I := 1 To N DO
BEGIN
    read (x) ;
    total := total + x ;
END
```

Note the use of BEGIN and END statements to group the statements.(Such a set up is called a BEGIN END block also).

If we write,

```
FOR I := 1 TO N DO
        read (x) ;
        total := total + x;
```

only the statement immediately following the FOR statement will be executed. In the above sequence of statements (without BEGIN, END) only read (x) statement will be executed N times.

09-8505055-OUP

Let us look at the statement:

```
total := total + x .
```

Do you remember the analogy of the sack given in Chapter 2. Similarly, here we are adding the value of 'x' to 'total' and replacing 'total'with this new value. At the end, total will contain the sum of all the values of x. It is good practice to initialize 'total' to zero because some computer systems do not initialize values to zeros.

That is why we had the statement

```
total := 0;
```

outside the FOR loop.

There is another format for the FOR loop.

```
FOR variable := expression1 DOWNTO
                expression 2 DO statement ;
```

In this statement, the value of variable will be decremented instead of being incremened (in the case of the earlier form of FOR loop). Other than this difference, this form will be the same as the previous one.

The FOR loop will terminate when the control variable achieves the limit (given by the expression 2). Sometimes, it will not be possible to fix the limit (by expression 2) for the FOR loop beforehand. A sequence of statements may have to be executed till a given condition is met (or until a given condition is true). For this purpose there are two kinds of statements(one is the REPEAT statement the other is the WHILE statement).

REPEAT Statement : The format of the REPEAT statement is the following:

```
REPEAT
        statements
UNTIL   condition
```

In this case, all the statements between REPEAT and UNTIL will be executed until the condition given by 'condition' (of the UNTIL) is true.

Example : We can use the problem given in chapter 2. If we write the program for the procedure written to find the average of a set of numbers using a REPEAT statement we will get the following program.

```
Program average (input,output);

  VAR
     N : integer ;
     K : integer ;
     x : real ;
     total,average :real ;
  BEGIN
     read(N) ;
     k := 0 ;
     total := 0;
  REPEAT
     read(x) ;
     k := k + 1;
     total := total + x ;
  UNTIL (k = N) ;
     average  := total / n ;
     write("average",average);
  END.
```

Now in this program the statements:

```
read(x) ;
k := k + 1 ;
total := total + x
```

are repeatedly executed until k = N.

In procedure 2.1 it was stated "count the number read".
The statement k:= k + 1 perform this.

Initially, (outside REPEAT loop) k is set to zero. Then after reading a value for x, 1 is added to k each time.

Therefore, k will have a number which is equal to number of times read (x) is executed.

The condition in the UNTIL clause should be a logical expression (which will take only true or false values).

WHILE Statement :

WHILE statement also can be used to execute statements repeatedly.
The general format of WHILE statements is :
WHILE condition DO statement ;

In this case also the condition has to be given by a logical (boolean)expression. The statement following DO will be executed while the condition is true. Therefore, you can set a condition that is true and let the statement be repeatedly executed. When the condition becomes false the execution of the WHILE statement will be stopped.

Note that only one statement will be repeated (executed with the WHILE statement). Therfore, if more than one statement need to be repeated use a BEGIN , END statements to group the necessary statements.

Let us re-write the program to find the average using a WHILE statement (instead of the REPEAT statement).

```
Program average (input,output);

VAR
    N : integer ;
    k : integer ;
    x : real ;
    total,average : real ;
BEGIN
    read (N) ;
    k := 0 ;
    total := 0 ;
WHILE  (k < N) DO
BEGIN
    read(x);
    k := k + 1 ;
    total := total + x ;
END ;
    average := total / N ;
    write ("average=",average);
END.
```

The statements in this program before the WHILE statement are the same as that of the previous one program-using REPEAT statement).
The condition in the WHILE statement of this example is

k < N.

If we have read a value for N, since k is set to zero this condition will be true initially. Therefore the statement (in this case the group of statements with the BEGIN , END) will be executed. Within the BEGIN- END block k is incremented by one every time when a value is read. When the value of k is less than N,the condition for the WHILE statement will be true. Therefore, the statements within the BEGIN-END block will be executed until the value of k becomes equal to value of N. Once k becomes equal to N the condition k<N will not hold true. At that point the statement following DO will not be executed.(The statement following the WHILE statement will be executed-in this case the statement:

```
average := total / N )
```

4.1 Decision Statement :

In real life situations we have to select a course of action (from two courses) depending on a condition.

For example,
If it rains, then read a book, else go out side.

In programming also, we might have to execute certain statements if a condition is true and if the condition is not true we might have to execute another group of statements.

Let us look back at procedure 2.2, which we wrote to find the average of numbers. There are two statements in that procedure which use conditions to test whether a certain function need to be performed.
They are:

(i) If k is less than N then start doing from the step labelled x.
(ii) If k is equal to N then average := total divided by N.

There is a Pascal statement which allows us to perform this function.
The format of this statement is:

```
IF condition
THEN statement
ELSE statement.
```

The condition has to be a boolean expression. The statement after THEN and the one after ELSE can be any valid Pascal statement.

Let us look at some examples of using the IF statement.

Suppose we want to get a positive number always from a given value.

If we have the following decleration:

```
VAR
   x,y : real ;
```

We can use the following statement:

```
IF x < 0
THEN y = - x
ELSE y = x;
```

Another example,

```
   IF x > 99999
THEN write ("Number is too large")
ELSE x = x*2 ;
```

Sometimes, it will be necessary to have more than one statement with the IF statement. Then BEGIN- END blocks can be used for this purpose.

```
IF condition
THEN
BEGIN
     statements
END
ELSE
BEGIN
     statements
END
```

It is possible to use several IF statements to form a compound IF statement. This way of compounding statements is also called nesting.

```
IF condition1
THEN
IF condition2
THEN
IF condition3
THEN statement1
ELSE statement2
ELSE statement3
ELSE statement4
```

In a compound IF statement the ELSE clause belongs to the nearest IF for which there is no ELSE clause. In the above : the ELSE clause with the statement2 belongs to IF with condition3, the ELSE clause with statement3 belongs to IF with condition2, and the ELSE clause with statement4 belongs to IF with condition 1.

The following program illustrate an application of the IF statement.

```
Program greater (input,output);

VAR
   x,y : real;
BEGIN
   Read (x,y);
   If  x>y then writeln (x, "is
            greater than", y)
            else writeln (x, "is
            less than", y);
END.
```

Program 4.2

In the above program the expression x>y will yield a boolean value (either true or false). Hence we may assign its value to a Boolean variable. Suppose t is a Boolean variable. Then the statement,

```
t := x > y;
```

would assign t the value of the Boolean expression x > y.

Exercises :

Ex - 1
Introduce a Boolean variable t as t := x > y; where appropriate in the program 4.2 and check whether its value is true or false.

Ex - 2 : Factorial n!
Write a program to determine the factorial of an integer number n given by

n! = n (n-1) (n-2) .....3.2.1. Run this program to see if it works.

This example is very similar to that of finding the average of n numbers. Write down the solution steps,and then translate it to a computer program. The program needs to work only on integers.

The case statement

We have seen the use of IF statement when it is necessary to select one of two possible courses of action. The CASE statement allows to choose one out of several courses of action. For example,

if i = value 1 then statement 1 else if i = value 2 then statement 2 else if i = value 3 etc.,

Where 'i' is an integer variable and 'value 1', 'value 2', etc ., are constants, occur quite frequently. The CASE statement generalises the notion of conditional selection of statements to enable the above relationship to be expressed more neatly as

```
CASE i of
        value 1: statement 1;
        value 2: statement 2;
        value 3: statement 3;
          "
          "
          "
          "
          "
          "
    value N: statement N;
    Else statement ;
    END
```

When the case statement is executed, the value of the case selector ('i') matches just one of the constant (value 1... to value N) and the corresponding statement will be executed. If there is no constant matching the case selector ('i') then else part is executed, However an else part is optional and if not included then the value of the case selector ('i') must match just one of the constants or otherwise an error would occur.

The general format of the CASE statement is the following:

```
CASE expression OF

constant 1, constant 2....constant i: statement 1;
constant j, ..............constant n: statement 2;
    "                          "
    "                          "
    "                          "
    "                          "
                           constant m: statement n;

Else statement;
End
```

The word Case should be at the beginning of the CASE statement, and the expression should follow it. The statements to be executed for different values of the expression should be included. A statement should be proceeded by the values for which the statement has to be excuted. These values should be followed by a colon.

There should be the word END at the end of the CASE statement.

As an example, consider a program to calculate the number of days in a given month. We shall assume that the year is not a leap year.

Having read in the month as an integer between 1 and 12 we should employ a CASE statement to test this month number against all possibilities, as in

```
CASE month OF
     1,3,5,7,8,10,12 : days := 31;
           4,6,9,11  : days := 30;
                 2   : days := 28
END
```

Enter the following program and see if it works

```
program days (input,output);
Var month, days  :  integer;
Begin
     Write ("month ?");
     Read  (month) ;

CASE month of

     1,3,5,7,8,10,12 : days := 31;
           4,6,9,11  : days := 30;
                 2   : days := 28;
Else writeln ("Wrong Number");
END;
    writeln ("The month", month,"has", days, "days");
END.
```

Write a pascal program to get the tommorow's date as output when the today's date is given. You may omit a leap year. (Hint: You may use the above result).

5.0 Procedures and Functions :

It will be very difficult to read and understand a very long program. If you have a program that cannot fit into one page then it will be hard to follow the program properly (since you have to turn pages and look).

On the other hand, it will be necessary to repeat a set of instructions several times.

We can overcome these problems by using procedures. A procedure consists of a set of instructions that will be executed when the procedure is called at another location of a program. A procedure has to be defined (written) before it is being used.

As an example, let us replace a part of the program we wrote to find the average, with a procedure. Let us look at program 'average'.

```
Program average (input,output);
    VAR
      N : integer ;
      k : integer ;
      x : real    ;
      total,average : real;
    BEGIN
       read(N);
       k      := 0 ;
       total := 0;
    WHILE (k<N) DO
    BEGIN
       read(x)
       k      := k + 1 ;
       total := total + x;
    END;
       average := total / N ;
       write("average",average);
    END.
```

We can put the statements:

```
       read(x);
        k := k + 1;
        total := total + x ;
```

into a procedure.

If we name the procedure as findtotal, the program can be written in the following way.

```
Program average (input,output);
     VAR
       N : integer ;
       k : integer ;
       x : real ;
       total,average : integer ;

     PROCEDURE findtotal ;
     BEGIN
       read (x);
       k := k + 1;
       total := total + x
     END;
     BEGIN
       read (N);
       k := 0;
       total := 0;
     WHILE (k < N) DO
       findtotal ;
       average := total / N ;
       write("average",average)
     END.
```

Type this program in and see if it works.

This programme starts with the word PROGRAM. Then the variable declerations are given. Afterwards, the procedure is defined. Procedure starts with the PROCEDURE statement.
This is called procedure head also. In this case, the procedure head is :

```
PROCEDURE findtotal ;
```

Then there is the body of the procedure.

```
BEGIN
.....  .
.....  .
END.
```

The procedure ends with the END statement. A procedure must have BEGIN and END statements (similar to a program).

After the procedure definition the main programme starts (from the BEGIN statement). Inside the main program the procedure is invoked by using the name of the procedure. In this case, look at the WHILE statement.

```
WHILE (k< N) DO
 findtotal ;
```

When the while statement is executed, if the condition is true, the procedure findtotal will be invoked and the statements in the procedure will be executed. After completeing the execution of the procedure, the statement following the invoking statement will be executed (in this case the statement average:= total /N. Remember, here also 'findtotal' will be executed several times because it is used in a WHILE statement.)

There are two kinds of variables that can be used within a procedure. They are named local variables and global variables Let us look at the program we just wrote using a procedure. No varibales were declared within the procedure in this case. The variables defined in the main program was used within the procedure also. In this case,

```
x
k
total
```

are global variables.

Let us consider another simpler example to see the difference between local and global variables.

Consider the following program.

```
Program simple (output);
    VAR
        x : integer ;
    PROCEDURE change ;
    BEGIN
        x := 1
    END ;
    BEGIN
        x := 0 ;
        change ;
        write (x)
    END.
```

Enter this program and run it.

In this example, x is only a global variable. It is defined in the main programme (and is used both in the procedure and the main programme).

Let us change this slightly.

```
Program simple (output);
     VAR
       x : integer ;
     PROCEDURE change ;
     VAR
       x : integer ;
     BEGIN
       x := 1
     END ;
     BEGIN
       x := 0 ;
       change ;
       write (x)
     END .
```

Modify and run the new program.

In this example, there are two variables named x. One is declared in the main program and it is a global variable (similar to that of the previous example).

There is another decleration of x within the procedure change. This decleration makes the variable x in the procedure local to it only. Since now x is a local variable in the procedure, the change made inside the procedure will not be carried out the invoking program. Therefore, in this case, the value written at the end will be zero.(not as in the previous case) the x variable within the procedure will be assigned 1 but it will not be given to the global variable x, which is written).

It is possible to pass values to procedures explicitly using parameter. So far, in the examples of procedures we have seen, no values are passed to procedure as parameters.

There are two kinds of parameters. The are (i) value parameters and (ii) variable parameters.

Example of variable parameter.

```
Program simple (output);
    VAR
       x : integer ;
    PROCEDURE change (VAR y :integer);
    BEGIN
       y := 1
    END;
    BEGIN
       x := 0;
       change (x) ;
       write(x)
    END.
```

In this example ,when invoking the procedure 'change' a parameter (x) is passed. If you observe the procedure head you would see that there is a decleration of a variable (y) with the head. This makes y a variable parameter. When a variable is defined as a variable parameter, it acts as a synonym to the variable used in invoking the procedure. In the example, y acts as a synonym to x.

Whatever changes happening to y will happen to x also. The value of x at the end of the program will be 1. When change is called with parameter x the value of x is zero (since it is assigned the value 0) Therefore, y will also have the value zero. Within the procedure, the value of y is changed to 1. Since y is same as x, the value of x within the main program also will change to 1.

When a variable parameter is used in a procedure, the call to the procedure must have a variable for the parameter (It cannot have expression, constatnts etc).

The following are invalid calls to the procedure 'change'.

```
(i)    change (2*x)
       2 *x is an expression
(ii)   change(2)
       2 is a constant.
```

In the above example, y is a variable parameter, now let us see how to use a value parameter.

```
PROGRAM simple (output);
VAR
   x : integer ;
PROCEDURE change (y : integer);

BEGIN
   y := 1
END;
BEGIN
 x := 0 ;
 change (x) ;
 write (x)
END .
```

Here, the word 'VAR' is not used when declaring the parameter y in PROCEDURE change. In this case, y is a value parameter. The value of y can be changed by giving a value to the parameter while calling. But y will not return a value to the calling program. In this example, the value of x will be 0 after the call to change.

Before the call to 'change' the value 0 is assigned to x (x :=0). Then 'change' is called (change(x)). Within change value 1 is assigned to y (y := 1). Since y is declared only as a value parameter, this value will not be transmitted to x in the main programme. When value parameters are used expressions or constants can be used while calling. For example,
change (2*x) or change(2)
is valid if a value parameter is used (as y).

There are words used in identifying parameters. The parameter used in the procedure is called a formal paramtere while the parameter used when calling the procedure is called an actual parameter.

```
PROCEDURE  change (VAR y : integer)
                  formal  parameter

PROCEDURE  change (y:  integer):
                     formal  parameter
           change (x)
                  actual parameter
```

5.1 Functions :

There is a special kind of procedure which is called a function. A function will return only one value as result(a procedure can return more than one value to the calling program depending on the number of parameters used). A function definition is similar to a procedure definition except in the case of a function the type of the function also has to be given.

Let us write the program to find the average of numbers using a function.

```
PROGRAM average (input,output);
  VAR
    N: integer;
    average:real ;
FUNCTION total(N:integer):real;
VAR
    k:integer;
    sum,x : real;
BEGIN
    k := 0;
    sum := 0;
WHILE (k<N) DO
  BEGIN
    read(x);
    k := k + 1;
    sum := sum + x
  END ;
    total := sum
END ;
BEGIN
    read(N);
    average := total (N)/N;
    write("average",average)
END.
```

The PROGRAM statement is written first. Then the global variables are declared. There is the function definition next. Inside the function there are the declerations of local variable. Then the statements of the function. The program body appears next to the function is called by the statement

```
average := total(N)/N
```

The function total is called by its name (total) and by writing the arguments of the function (ie. N, the calling word is : total(N)).

The general format of a function is the following:

FUNCTION, function name (parameter1: type1, parmeter2: type2, parameter3: type3): type4;

After the function name the formal parameter of the the function are given with their types. type4 indicates the type of the resulting value of function. In the example, the function 'total' is of type real.

There are other interesting forms to the function statement. For example you can call a function from within the same function.

Try this clever program,

```
Program clever (input,output);
  VAR x, z : real;
         y : integer;
```

```
Function power(m:real;n:integer):real;
   VAR n1 : integer;
   BEGIN
       n1 :=  n - 1 ;
       IF  n = 1 THEN
       power:= m
   ELSE  power := m * power (m, n1);
   END;  (* power *)
   BEGIN (* main program *)
       write ("input number");
       read (x);
       write ("input power");
       read (y);
       z :=  Power (x, y) ;
       write ("value   of",  x,"to the power", y,
       "is", Z);
   END.
```

Input the value of 2 for x and 4 for y and carefully work through the program step by step manually and calculate the result for Z. Type the program in and run it to see if your answer is correct.

Can you see what is happening when the fuction is calling itself? This type of function is known as a 'recursive function'.

* * * * * * * * * *

Exercise :

Write a recursive function to calculate
n! = n (n - 1) (n - 2) 2.1

Write a main program to use this function and run it to see if it works

* * * * * * * * * * *

5.2 Array Variables :

So far we have met only simple variables, these variables, are so called because they can have only one value at a time. An Array variable can however have more than one value at any given time.

For example let us consider the array variable called A. Let us say that it can take six values at a time. We can declare this variable in Pascal in the following manner.

```
VAR
  A : ARRAY {1..6} of Integer;
```

It declares an array which has six elements. These elements are pascal variables which can be individually referenced as
A. 1. , A .2., ... A .6. The integer contained in square brackets is often called a SUBSCRIPT. It is also referred to as the INDEX of the array entry.

As before this declaration must be done before the Begin statement.

The use of arrays is shown best by the following example;

Consider an election contest by four candidates, let us write a program to read the ballots and count the votes cast for each candidate. Assume that the candidates are numbered 1 to 4, and that each ballot is presented a series of input containing one of these numbers. An input of 2 will mean a vote for the second candidate.

To solve this problem we will follow the following steps;

```
Initialize all votes -counts to 0;
  WHILE  not all ballats have been read DO
  BEGIN
    read A BALLOT:
    add 1 to the chosen candidate's vote count
  END;
     Write the vote counts
  END
```

Consider the section of the program between WHILE and END;. To hold the vote - counts, we could use four INTEGER variables count 1, count 2, count 3 and count 4. Then the necessary statements would be,

```
READLN (Ballot);
  If Ballot = 1 then
    add 1 to count 1
  Else If Ballot = 2 then
    add 1 to count 2
  Else If Ballot = 3 then
    add 1 to count 3
  Else If Ballot = 4 then
    add 1 to count 4 ;
```

Don't you think that this is clumsy. What if the number of candidates are large say 15.

Consider an alternative program,

```
Program ballot(Input,Output);
  CONST
    Ncandidates = 4 ;
  VAR
    Ballot, Nballot: Integer;
    Count : Array {1.. N candidates of integer;
    I,K,J   : Integer;
  BEGIN
    (*Initialize vote counts to 0*)
    For I := 1 to Ncandidates do
    Count {I} := 0;
    WRITE ("Total number of votes =");
    READ (Nballot);
    For K := 1 to Nballot do
  BEGIN
    READ (Ballot);
    Count{Ballot}:= Count{Ballot}+1;
  END;
    For J := 1 to Ncandidates do
    (*Writting the result*)
  WRITELN ("NO OF VOTES FOR
     CANDIDATE", J,"=", COUNT {J});
  END.
```

The nine statements which were used to count the votes cast for each candidate has been reduced to five (between READ (Nballot) and END;), furthermore if the number of candidates say were increased to 15, the number of statements in the former program would go up to 31 while all we have to do in this program is to change Ncandidates = 4 to 15. The program has become much more flexible.

Enter this program and run for a given number of votes cast, say 25. Enter any number between 1 and 4 for the value of ballot.

* * * * * * * *

Exercise : What would happen if a vote has been spoilt, i.e. someone has entered 0 or 6 for instance?

* * * * * * * *

Exercise : Can you change the program so that it will count the number of spoilt votes? Also introduce a check to see if, the total votes cast = sum of votes cast for all Candidates +spoilt votes.

6.0 Variable Type :

We have learnt about the four standard types (boolean, char, integer, real) in Pascal. Other than these stadard types, it is possible to define types within a Pascal program. The new types are declared by using the reserved word TYPE. So far, we have seen how to declare constants (using CONST) and variables (using VAR).

6.1 Scalars :

With the declaration of a scalar type a list of values that may be assumed by a variable of that type is given.
For example,

```
TYPE
length = (inches, feet, miles);
```

The above declaration states that the type length may have any of the three values indicated and no other value. Now the type length can be used in a variable declaration in the same way as one of the standard types

```
VAR
   side : length ;
```

The ordering of declarations is such that the 'type' comes immediately before 'VAR' declarations but after any 'const' declarations.

A value may not belongs to more than one type

```
TYPE
    length = (inches, feet, meters, miles)

    units = (weight, volume, inches)
```

The above declarations are in compatible because the value inches appear in both value lists.
Consider the following types.

```
TYPE
    day = (monday, tuesday, wednesday, thursday, friday,
           saturday, sunday)
           Season = (rainy, summer, spring)
           Operator = (plus, minus, multiply, divide)
```

If we declare the following variables:

```
VAR
   holiday, workday : day;
   addop, divop : operator;
   off, on : season;
```

The names of the values listed in the declaration of a scalar type are constants of that type. For example, with the above declarations we can write the following assignment statements in a program.

```
  workday :=  monday;
      off := rainy;
    addop := plus;
```

Mixed assignments are not permitted.

```
  holiday := on ;
```

is not allowed; holiday and on are of different types.

Only relational operators can be used with the scalar types, and the resulting expression will have a boolean value. The ordering of scalar types are defined by the order in which they are enumerated (listed) in the type declarations.
For example,

```
monday < friday
summer > rainy
  plus < divide
```

There are two functions
pred
succ
defined for scalar arguments.

pred
returns the predecessor of the argument, and
succ
returns the successor of the argument,

For example,
pred (tuesday)
will return monday.
and
succ (plus)
will return minus.

Scalar type variables will give no result when used with either write or read.

S.A.Q. 6.1 :

Find out whether the following declarations are correct or incorrect.

(i)
```
TYPE
  units = (meter, inches, feet);
VAR
 length : units;
```

(ii)
```
TYPE
  measure = (weight, length, volume, density);
    mass = (pounds, kilo, weight);
```

(iii)
```
TYPE
  days := (monday, tuesday, friday);
VAR
  working day := days;
```

(iv)
```
TYPE
  fruit = orange, banana, mango, plums
```

S.A.Q. 6.2 :

```
TYPE
    day = (monday, tuesday, wednesday, thursday,
           friday, saturday, sunday)
    pay = (regular, overtime, bonus, basic)
  scale = (meters, inches, feet)

VAR
 holiday, workday: day;
          amount : pay;
          length : scale;
```

If the above declarations are given which of the following assingments statements are correct and which are incorrect.

```
  (i) holiday := saturday;
 (ii) amount  := feet;
(iii) length  := meters;
 (iv) workday := bonus;
  (v) amount  := regular;
```

It is possible to declare an array using the 'type' statement.

For example,
Type count = array {1 .. 4} of Integer;
declares the type identifier 'count' and that can be used to declare variables of the new type as in

Var C : count;
This is the same as declaring the variable C by using 'VAR' as in VAR C : array {1..4} of integer;

Further, procedures or functions that have parameters of the type 'count' can be written, for example,
Procedure countballot (var C : count);

Note that it is not premissible to write 'array {1..4} of integer' in place of 'count' in the procedure countballot and therefore the 'type' statement improves the structure of a program.

Let us rewrite the program ballot by using the 'type' statement and the procedure countballot.

```
Program ballot (Input, Output);
Const
     Ncandidates = 4 ;

Type
    Count = Array {1.. Ncandidates} of Integer;
Var c : count;
    Nballot, j : Integer;

Procedure countballot (Var b : count;
                       Nballot : Integer);


Var
    I, ballot : Integer;

Begin
   For I := 1 to Ncandidates DO
            b{I} := 0;

   For I := 1 to Nballot DO
Begin
   Read (ballot);
   b{ballot} := b{ballot} + 1
   end;
end;
Begin {*Main Program*}
Write ("Total Number of votes = ");
Read (Nballot);
Countballot (C , Nballot);
For j := 1 to Ncandidates DO
Writeln ("No of votes for candidate ", j, "=", C{J})
end.
```

Program 6.1

6.2 Subranges :

A subrange type is defined using two constants.
For example,

```
TYPE
     index = 1..40;
    letter = 'a'..'Z';
     digit = '0'..'9';
```

Subrange variables are declared in the variable declaration section.

For example,

```
VAR
   limit, middle : index;
   first, last : letter ;
```

The type constants in the subrange could be integer, char or scalar type. (in the above example 1..40 - integer, and 'a'..'z' - char). Subranges of type real are not allowed.
Read and write can be used with the subranges of the type

```
char
integer
```

and have expected results.

6.3 Sets :

A set is a collection of objects of the same type. If we have a scalar type we can declare a scaler type also corresponding to that scalar type.
For exmple,

```
TYPE
    days = (monday, tuesday,wednesday, thrusday,
            friday,saturday, sunday)
    week = SET OF days;
```

now, variables of the type week can be declared in the VAR section.

```
VAR
   time : week;
```

The values of a constant or a variable of the type week are objects of the set of days.

In Pascal a set is represented by its members enclosed in square brackets, { }. If we continue with declarations given above.

```
{saturday, sunday}
{monday}
```

are two sets we can write.

If a set contains the consecutive values, only the first and the last may be sepcified.
For example,
{monday, tuesday, wednesday, thursday, friday}
can be written as,
{monday, friday}

S.A.Q. 6.3 :

Declare a set containing the following members and name the set as force.
gravity, friction, inertia and kinetic

Three operations can be performed on sets.

The union of two sets is a set containing the members of both sets.
The union operator is

```
+
```

For example,

```
{monday} + {tuesday, wednesday} = {monday, tuesday,
                                   wednesday}
```

The other is the intersection. The intersection of two sets is a set containing only the objects which are members of both sets.
The intersection operator is

```
*
```

For example,

```
{monday, tuesday} * {tuesday, wednesday,
                     friday}  = {tuesday}
```

The difference of two sets is a set containing all the members of the first set which are not members of the second set.

The difference operator is

-

Example of difference operation:

```
{monday, wednesday, friday} - {wednesday,
                               saturday} = {monday, friday}
```

The relational operators can be used to compare sets.

```
 =  denotes  set equality
< > denotes set inequality
= < denotes 'is contained in'
> = denotes 'contains'
```

For example,

```
{monday, tuesday} =  {tuesday, monday}
     {wednesday} < > {tuesday, friday}
      {wednesday} =< {tuesday, wednesday}
 {monday..friday} >= {tuesday, wednesday}
```

We shall write a simple Program which will read a sentence terminated by a fullstop and print out the number of vowels and consonants.

```
Program CVCOUNT (Input, Output);

Var ch:char;
    Vowels, consonants : integer;
Begin
    Vowels := 0 ; consonants := 0;

Repeat
    read (ch);
if ch in {'A', 'E', 'I', 'O', 'U', 'a', 'e', 'i', 'o'
          'u'}
then vowels := vowels + 1
else if ch in {'A'..'Z','a'..'z'}
then consonants := constants + 1

Until ch = '.';
write {'sentence contains', vowels: 3, 'vowels and',
consonants : 3, 'consonants'}

end.
```

The set specification using only the first and the last element (A and Z) is the most advantageous statement used in this Program since otherwise, we have to write every element in the set A to Z in order to get the required result.

S.A.Q. 6.4 :

For the declaration made in SAQ 6.3, find the values of the following expressions.

(i) gravity In {friction, kinetic}
(ii) friction = < {gravity, friction}
(iii) {kinetic, gravity, friction} > = {gravity, kinetic}
(iv) Kinetic In {friction, kinetic, gravity)
(v) {friction} < > {kinetic,gravity}

The word IN is used to test whether an object is in a set.
For example, the value of the expression monday IN {monday..friday}

is true (because the object monday is in the set of elements formed by monday, tuesday, wednesday, thursday, friday}.

The value of
sunday IN {monday..friday}

is false.

At the begining, we declared
```
VAR
   time : week;
```
now time can be treated as a set. For instance, we can assign a set to this variable.
```
time := {monday, tuesday, wednesday};
```

6.4 Structured Typed :

MORE ON ARRAYS

The base type of an array is the type of the elements of the array. For example, with regard to the following declaration:

```
column = ARRAY {1..10} OF real;
```

the base type of column is real.

In Pascal it is possible to have a base type as an array itself. For example, the following declarations are valid.
```
CONST
     size = 10;
TYPE
   subscript =  1..size
     column  = ARRAY{subscript} OF real
     matrix  = ARRAY{subscript}OF column
```

In the above example, the base type of the array matrix is another array called column.

The above declaration can be written in another form.
```
CONST
     size = 10
TYPE
     subscript = 1..size
        matrix = ARRAY{subscript,subscript} OF real;
```
This type of arrays can be called two dimensional arrays.

In mathematics, a matrix is a collection of numbers which are ordered in rows and columns.

The following is a matrix.

```
1   5   6
7   3   2      diagonal of the matrix
2   1   7
```

A matrix is called a unit matrix when only the values in the diagonal are ones and the others are zeros. The following is a unit matrix.

```
1   0   0
0   1   0
0   0   1
```

Example,
Let us write a small program using a two dimensional array to form a unit matirx.

```
PROGRAM Unit;
CONST
   size = 10
TYPE
   subscript = 1..size
     matrix  = ARRAY {subscript, subscript} OF real;
VAR
  s, t : integer;
     a : matrix;
BEGIN
  FOR s := 1 TO size DO
  FOR t := 1 TO size DO
   IF s = t
  THEN a{s,t}:= 1
  ELSE a{s,t}:= 0
END.
```

Follow this program and see whether it forms a unit matrix (with 100 elements)-
Note that there are 10 rows and 10 columns in the array according to its declaration)

6.5 Packed Arrays :

If we declare an array then each element of it will take one word of the memory.

```
VAR
   String : ARRAY {1..1000} OF char;
   According to the above declaration,
   string
   will occupy  1000 words of memory.
   If we have a packed array:
VAR
   string : PACKED ARRAY {1..1000} OF char.
   string
   will occupy only 500 words (machine dependable)
```

The general format for declaring a packed array is the following

name : PACKED ARRAY {index} OF base type;
Packed array can be used in the same way as an unpacked array.
{limitaitions compiler dependant}

Now let us write a simple program which will read N names and print them in alphabatical order.

This type of work is usually called SORTING. Sorting is the re-arrangement of data into ascending or desceding order.
It is easy to divide this problem into three stages

(i) Read N names into an array
(ii) Sort the array
(iii) Print the array.

(i) A suitable array to hold the names may be as follows (we assume N to be less than 10).

```
TYPE
    String = Packed array{1..20}of char;

VAR
   name : Array {1..10} of string;
```

Now to read N names we can use a for loop as

```
for :=1 to N do
Readln (name {I});
```

(ii) The technique used to sort the array is based upon a simpler algorithm that moves the smallest element in the array to the first position of the sub array. This algorithm is called a selection sort.

name {1}	name {2}
Lalith	Aravinda

Now the problem becomes to find the smallest element in the array. For that we can compare two element at a time and pick the smaller one and then compare it with the third one and so on. The Pascal compiler version which you study allows you to compare two strings (words) using relational operators. we shall learn the less than operator (<).

The less-than operator for strings compares the two operand strings one character at a time: the first character in the left operand is compared to the first character in the right operand, then the second character in the left operand is compared to the second character in the right operand, and so on.

The less-than condition is true if

(a) the comparison terminated because the left operand was shorter than the right operand (but was the same up to that point), or

(b) the comparison found that the individual character in the left operand was less than the individual character in the right operand.

For example

'abc' < 'abcd' is true, since the first three characters of the string are the same, but 'abc' is shorter than 'abcd' and

'abc' < 'bc' is true since the character 'a' is less than the character 'b'.

Now if we use namelow to hold the smallest item the sorting algorithm can be written as follows.

```
For I := 1 to N-1 DO
Begin (*each sub array*)
namelow := name{I};
PosN  := I;
For j := I + 1 to N DO (*select the smallest
                         item*)
If name{j}<namelow then begin
          nemelow := name{j};
          PosN    := j
          end;
(*Exchange the smallest item with the first
item in the sub array*)

name {PosN}  := name{I};
name{I}      := namelow
End (*each sub array*)
```

Follow this algorithm and see whether it gives required result.

(iii) Finally, we can use a for loop to write out the sorted names as

```
for I := 1 to N DO
writeln (name {I});
```

S.A.Q. 6.5 : Write a program to read a list of N names (N<10) and print them in alphabatical order.

6.6 Boolean Arrays : An array with the base type boolean can be treated as a set. Each element of a boolean array correspond to a potential member of a set. A member of a set may be present, in this case the corresponding value of an element in a boolean array will be true. A member may be absent from a set. Correspondingly, value of an element in a boolean array may be false.

If we have the following declaration

```
TYPE
    index = 1..20,
VAR
    ib : index;
  best : SET of index;
  barr : ARRAY {index} OF boolean;
```

Then the operation

```
barr {ib} := false
```

is equivalent to removing the element ib from a set

```
bset := bset - {ib}
```

Similarly, the operation

```
barr {ib} := true
```

is equivalent to adding an element to a set

```
bset := bset + {ib}
```

(+ is the union operator)

```
barr {ib}
is equivalent to stating
ib IN bset.
```

Operations on a set will be faster than the operations on a boolean array. However, when more components need to be used (eg. 10,000 or 100,00) it will be easier to use a boolean array.

6.7 Records : Record is a variable where you can have several components. These components of a records may have different types.

A description of a chair might include the following information

height
weight
colour
price

We can define a record for a chair in a Pascal program. Suppose that 7 characters are used to indicate the colour. Then the declaration of the rocord will be as the following.

```
TYPE
   item =
   RECORD
    height : real;
    weight : real;
    colour : PACKED ARRAY {1..7} OF char;
    price  : real;
    END.
```

In the variable declaration section we can use this type to declare a variable.

For example,

```
VAR
    chair : item;
```

Now chair will refer to a record as declared in the TYPE declaraion. If you want to use one component of a record you have to use it along with the record name.

For example, if you want to refer to height

```
    chair . height
```

has to be used.

Each component of the record can be treated as a variable of the type as declared in the TYPE declaration seciton. If colour

```
    of the record
          chair
```

has to be assigned the value 'brown', the following statement can be used.

```
chair . colour = 'brown';
```

6.8 The WITH Statement :

It might be necessary to access different components of the same record several times in a program. At each time you can use record name component name to access it.

Then every time, you have to write the record name also. This can be avoided by using the WITH statement. If we use the above example of declarations for the RECORD chair, we can write the following:

```
 WITH chair DO
   BEGIN
        colour := 'green'
        height := 3.0
        weight := 5.0
         price := 120.00
   END.
```

In the above, since the WITH statement is used it is not necessary to have the record name with the components in the assignment statements.

The general format of a RECORD declaration is:

```
TYPE
   name  =
   RECORD
      component 1 : type 1 ;
      component 2 : type 2 ;
      component 3 : type 3 ;

      END;
```

= sign after the name of the record (which also can be called the record identifier) followed by the word RECORD. Then the components has to be declared along with their types. The word END has to be used at the end of component specifications.

The general format of the WITH statement is the following:

```
WITH record identifier DO
     statement.
```

note:- RECORD has to be declared in the TYPE declarations section of a program.

S.A.Q. 6.6 : Declare a record which contain the following components
day - type subrange of integer
month- array of char
year - integer;
give a suitable name to the record type.

S.A.Q. 6.7 : Declare the variable 'today' which is a record of type date. Assign January 1, 1986 to the variable 'today' declared according to S.A.Q. 6.6.

S.A.Q. 6.8 : Use a with statement to assign the value January 1, 1986 to the variable today.

Exercise : Write a program to output the date assign in S.A.Q. 6.8 (Hint use the variable 'today'.)

Answers to the S.A.Q's :

S.A.Q. 6.1 :

(i) Correct
(ii) incorrect - since weight in two different declarations
(iii) incorrect since := used insted of =
(iv) incorrect since no paranthesis to include the scalar values.

S.A.Q. 6.2 :

(i) Correct
(ii) incorrect because of types mixed
(iii) correct
(iv) incorrect because of types mixed
(v) correct

S.A.Q. 6.3 : First we have to declare scalar type and then the set type

```
TYPE
    kind = (gravity,friction, inertia, kinetic);
                    force = set of kind;
```

S.A.Q. 6.4 : The each expression is of type boolean and hence either true or false.

(i) false
(ii) true
(iii) true
(iv) true
(v) true.

S.A.Q. 6.5 :

```
Program sortnames (Input, Output);
Type
    string = packed array {1..20} of char;

Var
     name( : Array {1..10} of string;
  namelow : string ;
            N,I,J,PosN, : Integer;

Begin
   write ("Enter number of names to be sorted");
   Readln(N);
   For I := 1 to N DO

   Begin
   Write ("Enter name", I : 2);
   Readln (name {I})
   end;
(*Selection sort : For each successively shorter  sub
arrays of name, put its smallest element in its first
position*)

For I := 1 to N-1 DO (*select the smallest item*)
Begin
  namelow := name{I} ;
     posN := I;
   For j  := I + 1 to N DO

   If name{j} < namelow then
                begin
                nemelow := name{j};
                posN := j
                end;

(*Exchange the smallest item with  the first item  in
 the sub array*)

     name{posN} := name{I};
     name{I}    := namelow
     end;
     (*writing the sorted result*)
       For I := 1 to N DO
       writeln (name{I});
       end.
```

S.A.Q. 6.6 :

```
We shall name the record as date:
type
    date =
      record
        day   : 1..31;
        month : packed array {1..8} of char;
        year  : integer
      end;
```

S.A.Q. 6.7 :

```
Var today : date;

Begin
  today.day   := 1;
  today.month := 'January' ;
  today.year  := 1986
end;
```

S.A.Q. 6.8 :

```
With today do
Begin
  day   := 1;
  month := 'January' ;
  year  := 1986;
end;
```

7.0 Loading, Saving and Scratching Programs :

There are a number of programs and other files stored on the Pascal diskette. We may examine them by giving suitable commands. First ensure that the cursor is at the bottom of the screen; remember, if you are in screen mode, pressing F2 will return you to command mode and place the cursor at the bottom of the screen.

To examine the names of the files type

directory (or dir)

and then press the RETURN key.

A list of programs and files will appear on the screen and pressing the RETUEN key causes either rest of the list to appear or the list to disappear.

LOADING A PROGRAM (GET COMMAND)

One of the programs on the Pascal diskette can now be loaded into the computer and examined. We can use the get command for that purpose.

The syntax specification of the GET command is
<line no>Get<file name>

The Get command causes the entire contents of the specified file to be inserted following the line specified by < line no >

For example,

get sum insert the file named "sum" after the current line.

Type
get sum and press the RETURN key.

Then the program sum will appear on the screen. Suppose you need to insert the program area inside of the program sum.

Type
5 get area

and press the RETURN key.

Now the program 'area' has been added to the program sum. This is a very useful technique when dealing with large programs. Since you may divide a large program into a small parts and save them on the diskette and later by using the above technique you may concatenate them.

SAVING A PROGRAM (PUT COMMAND)

Any program we have in the computer can be saved on a diskette. Suppose we have program sum with us. To save this type

Put sum

BEFORE PRESSING THE RETURN KEY INSERT THE EXTRA DISKETTE PROVIDED INTO THE DISK DRIVE.

After a short pause, the number of lines transferred will be displayed on the screen. You may specify the line number range which you wish to transfer

For example;

```
2,6  put sum1
```

will put all lines from the second line to the sixth line into the file named sum1. The previous contents of the file named SUM are lost.

SCRATCHING A PROGRAM (SCRATCH COMMAND)

The scratch command is used to delete programs on the disk.

For example;

```
SCR   SUM1    delete the program sum1
```

Type the above statement and press the RETURN key.

7.1 Files :

It is frequently desirable to store data on an external device such as a diskette. For example, a file could be created which contains information about all the students in a class. This file would contain several LINES or records, one for each student. Files have names such as student, which are used to identify them.

```
1110 Anoma          63 76
1297 Aravinda       80 75
1317 Bandara        70 46
2568 Chinthamani    90 66
2587 Dias           40 55
2600 Herath         85 70
3027 Oliver         63 78
3055 Tajudeen       71 81
```

Figure 7.1

Consider figure 7.1 which is a listing of a file called STUDENT. This file contains 8 records. Each record contains several fields of information about a student. These fields are student registration number, name, and the marks obtained in 2 courses, namely Mathematics and English. Note that field quantities are seperated by blanks and the second field is 15 characters wide.

7.2 Types of Files :

Pascal views files in one of two categories: either as text files or as non-text files. The most common is the text file which as its name suggests consists entirely of characters. In non-text files the data are held in binary form, not as characters.

7.3 Text Files :

A textfile is represented as a sequence of variable-length records, where the records are seperated by end of line characters.

7.3.1 The Text File Declaration :

To enable convenient manipulation of character text PASCAL provides a standard file type text. It is an enhenced version of the type.

```
Type
    text = file of char;
```

Files declared to be of the standard type TEXT are known as text files. A text file is structured as a sequence of lines each line containing values of type char-separated by special line control characters (the character caused by pressing the RETURN key). The identifiers Input and Output which we listed in the program heading are infact pre-declared text file identifiers.

i.e. Input, Output: text;

We may declare the file identifier student to be of the standard type text as follows.

```
Var
   student : text ;
```

7.3.2 File Initialization:

Any file declared has to be initialize for reading or writing (but not both) by use of the standard procedure RESET or REWRITE. Consider first the construction, or writing, of a file. To prepare a file f for writing, the standard procedure statement.

```
rewrite (f)
```

is used. Its action is effectively to erase the contents of the file f (i.e. f becomes an empty file containing zero components).

Now consider the operations involved in inspecting the contents of a PREVIOUSLY CONSTRUCTED file. To prepare a file f for reading the standard procedure statement.

```
reset (f)
```
is used.

7.3.3 Writing Data to A Text File :

After intializing a file f for writing (by using rewrite (f)), the write statement can be used to write data items to the text file f. The write statement used for this purpose is

```
Write (f,x)   or
Writeln (f,x)
```

The effect is to write to f sequence of characters which denotes the value specified by the parameter x. In general, the statement Write(f, x1, x2, ...,xn) can be used to write data items to the file f.

```
The form
Write(f, x1 , x2,...,xn) is equivalent to
begin Write(f , x1) ; Write(f ,  x2 );
......; Write(f , xn) end
and the form
Writeln(f, x1, x2,.... xn);
is equivalent to
begin
Write(f, x1, x2, ... xn); writeln(f)
End.
```

The effect of the form Writeln(f) is to append a line control character to the file f. If the file is printed, the characters written after the line control character will appear on a line following those Written before.

7.3.4 Construction of A Text File :

Let us construct the file student discussed in subsection 7.1. Consider the following program.

```
Program constructstudent  (Input, Output);
Type
Data  =
     Record
     RegNO : Integer;
     name: packed array {1..15}  of
     char ;
     Mam, Engm:  1..100;
     end;

Var
     student : text;
     std : Data;
     blanks, I : Integer;
Begin
    rewrite (student); (* Prepare file for writing*)
    For  I := 1 to 8 DO
    with std do
     Begin
     Write ("Enter Reg.No");
     readln(Regno);
     Write("Enter name");
     readln(name);
     Write("Enter marks for maths");
     readln(mam);
     Write("Enter marks for English");
     readln(Engm);

     blanks := 15 - strlen(name);
     Writeln(student, Regno,'  ', name, '    ': blanks,
             '  ', mam, '  ', engm)
    end;
end.
```

program 7.1

In type declaration section Data is declared to be of type record. A Data contains four fields called Regno, Name Mam and Engm which stands for Registration number, name of the student, marks obtained for Mathematics and English respectively.

Then a file variable 'student' is declared to be of type TEXT and the variable std of type record. The variable 'blanks' is declared for a special purpose. When writing data items to the file 'student' the field quantities have to be seperated by blank spaces or otherwise, when retrieving data, from the disk, we may not be able to read them seperately. Special type of problem arise when we are to seperate the field quantity 'name' since the name is "padded" with blanks on the right so that there is always a 15 character name field. To seperate it from the third field quantity (i.e. Marks for Mathematics) we have to calculate the number of blank spaces between the second and the third field quantities. Since we are using an array of 15 characters wide to hold a name, the number of blank spaces can be calculated by substracting the length of a given name from 15.

i.e. blanks := 15 - strlen(name);

The function strlen(name) gives the length of the string specified by the string variable 'name'. For example

strlen('Anoma') gives 5.

After the declaration section, the statement rewrite(student) is used to prepare the file 'student' for writing. Then to read data from the keyboard (the 'input' file) and to write data to the student file a for loop is used with the WITH statement.

Note that the other field quantities in the file 'student' are seperated by only one blank space.

Load the program 'constructstudent' from the pascal diskette provided and run it. You may use the data items in figure 7.1 as inputs.

7.3.5 Reading A Text File :

Before we proceed to write a pascal program to read a text file it should be noted that we can look at the file using the 'get' command. For example, to look at the file 'student',
First type

*delete press the RETURN key and then type
get student and press the RETURN key.

Type accordingly and see what happens.

To read a file (say f) we have to prepare it for reading. As we said earlier, the statement reset (f) will do that job. Then we can use the READ statement to read the file. The READ statement used for this purpose is

```
Read (f,x)      or
Readln(f,x)
```

For example to read the first field quantity in the 'student' file we write
Read(student, std.regno) where std. regno is defined as a component of the record DATA. and to access a complete record we may write

Read(student, std.regno, std.name, std.Mam, std.Engm)
But note that to access the complete file we have to use readln statement instead of Read. This is because we wrote the file line by line.

Try the following program to read the file 'student'.

```
Program Readstudent(output);

Type
DATA  =
     Record
     Regno : Integer ;
     Name : Packed array (1..15) of char;
     Mam, Engm : 1..100 ;
     end;

Var
     student : text
         std : DATA ;
           I : Integer;

Begin
     reset(student)  ( *prepare file for reading*)
     For I := 1 to 8 DO
     with std  do
Begin
     Readln(student, Regno, name, Mam, Engm);
     Writeln(regno:5, name:15, Mam:3, Engm);
     end;
end.
```

Program 7.2

7.3.6 The Function EOF (End Of File) :

When reading information from a file a program needs to be able to test whether there is any more data left. To enable this pascal provides a standard function EOF. For any file name 'f'.

EOF(f) returns TRUE if the file f is positioned at the end-of-file (past the last element) and FALSE otherwise. A file f may be read only when eof (f) is FALSE. An attempt to read further when EOF is true causes an execution error. With EOF files of unknown length can be handled. For example suppose we didn't know the length of the file student (we know that its length is 8 lines). Then to read the file we write,

```
While  not EOF(student) DO
With std DO
Readln(student, Regno, Name, Mam, Engm);
```

Remember the statements within the object of the WHILE are repeated as long as the expression 'not EOF(student)' is true. Initially, the EOF(student) is false and therefore 'not EOF(student)' is true. When the file 'student' is positioned at the end of file EOF(student) becomes true and this courses the repetetion to terminate.

Exercise :

Modify the program 7.2 using the atatement WHILE and and the function EOF.

7.3.7 The Function EOLN (End Of Line) :

The function Eoln can be used to test whether the end of a line has been reached. This applies only for text files. If f is a text file then

Eoln(f) returns TRUE if the text file f is positioned at the end of current line, and returns FALSE otherwise.

Note: If f is ommited from eoln(f) then the standard file 'input' is assumed.

S.A.Q. 7.1 :

Write a pascal program to read a line of text (in lower case characters) and respond with number of vowels and consonants used in the line, with no duplicates. For example the result for the line
pascal programming
is
vowels = 3
consonents = 8

7.4 Non-Text Files :

A non-textfile is represented as a sequence of fixed length records. The record length is selected automatically by pascal. No record separators are required for non-textfiles, since the beginning of each record is a fixed distance from the proceeding record.

Before we proceed further it is worth learning two major categories of file, namely

(1) sequential files
(2) direct-access files.

A sequential file may be read (or write, but not both) serially from beginning to end, one item at a time. It is not possible to start processing a sequential file from anywhere except the beginning.

There is no way, for example, to read up to a particular point and then start writing from there. The physical medium on which the concept of the sequential file is modelled is the magnetic tape, where the tape winds forward from one reel onto another part a read (write head that scans one datum at a time).
The text file 'student' is a sequential file.

A direct access file is divided up into records (except that pascal uses the word 'record' in another context), each of which may be reached individually by its numeric address. This feature is characteristic of magnetic disc rather than tape.

Sequential access is the only type of file access permitted in standard pascal. Therefore we shall restrict our discussion and deal only with sequential files.

7.4.1 File Type Declarations :

A file type may be defined simply by specifying the keywords File of, followed by any type sepcification.

For example,

```
Type
    stream   = file of char ;
    data     = record
               name : packed array
               (1..20) of char ;
               age : integer ;
               sex : (male , female) ;
                end;

      datafile = file of data ;
      number   = file of real ;
```

Any one of the above declarations gives a sequence of elements, all of one particular type (called the constituent type). For example

The file stream is a sequence of characters and the datafile is a sequence of records.

READ AND WRITE
Read and write statements can be used with non-text files (as with text files) Note that Readln and writeln are not used with non-text files (i.e, they have no meaning)

Let us write a program to construct a nontext file for the data in fig.7.1. The data need not to be entered again we may read the information from the student file and write them in the 'nstudent' file file (the file which is to constructed)

```
Program (ntextstudent);
Type
Data =
     Record
     Regno : Integer;
     name  : Packed array (1..15) of char;
     Mam,  Engm : 1 .. 100 ;
     end ;

Var
       student : text
     Ntstudent : file of data ;
          std  : Data ;
```

```
Begin
     reset (student) ;
     rewrite (Ntstudent) ;
     while not eof (student) do

Begin
     with std do (* reading data from student file*)
     Readln (student, Regno, name, Mam, Engm);
     (*writing data to the ntstudent file*)
       Write (ntstudent, std) ;
       end :
end.
```

After the variable declaration section, the 'student' file is prepared for reading by the procedure 'Reset' and the file 'Ntstudent' is prepared for writing by the procedure 'rewrite'.

When reading information from the text file 'student' we have to access the record component-wise and therefore a WITH statement is used with the procedure readln. But when writing data to the 'ntstudent' file it is not necessary to specify the components seperately. we can write the complete record (std) at once. This is because the file 'ntstudent' is a sequence of Records of type Data.

Enter this program and run it.

Now if you look at the directory of files you will notice a file with name 'ntstudent' will appear in the listing.

But if you try to access it by 'get' command you will get a strange result. This is because it is a non text file.

S.A.Q. 7.2 : Write a program to read the file 'ntstudent' and see see whether it works.

Answers to the S.A.Q.'s :

S.A.Q. 7.1 :

```
Program CVcount (Input, Output);
Var
   Charset : set of char ;
   Consonents, vowels : Integer;

   Procedure processline ;

Var
   Ch : Char;

Begin
   Charset := [ ] ;
   While not eoln do
   begin
   read(ch) ;
   charset := charset + [ch] ;
   end ;
end ;
   procedure printCVS ;
Var
   curr : char;

begin
    vowels := 0 ; consonents := 0 ;
    for curr  :=  'a' to 'z' do
```

```
        if curr in charset
        then if curr in
        ['a', 'e', 'i', 'o', 'u']
        then vowels := vowels + 1

    else consonents:= consonents + 1 ;
    Writeln('vowels:=', vowels, 'consonents =',
                        consonents) ;

    end;

    begin
        processline ;
        printcvs ;
    end.
```

A set of char is used to accumulate the characters found in each line. The set 'charset' is initilized to empty set. As each character is read, it is added to the set. It should be noted that adding a character which is already in the set has no effect on the contents of the set. At the end of the processing, the number of vowels and consonents are displayed. This is accomplished by a for loop which repeats once for each possible element in the set (the lower case letters). The presence of each letter in the set 'charset' and the vowel set is detected using the IN operator.

S.A.Q. 7.2 :

```
Program Rntextstudent (output);

Type
    Data =
        Record
        Regno : Integer ;
        Name : packed array [1..15] of char
        Mam, Engm : 1.. 100 ;
        end;

    Var
        ntstudent : file of data ;
              std : Data ;

    Begin
        reset (ntstudent) ;
        while not eof (ntstudent) do
        Begin
        Read (ntstudent, std) ;
        with std do
        Writeln(Regno, name, Mam, Engm)
        end;
    end.
```