DIT 111 PRINCIPLES OF PROGRAMMING

BY

J. B. HAYFRON-ACQUAH

DEPARTMENT OF COMPUTER SCIENCE

KNUST

KUMASI

Aim of the Course

- Intended for those without any prior knowledge of programming or absolute beginners

- To help learners write set of instructions for solving problems

Course Outline

Software Development Cycle	Input Statements
Algorithms	Input and Output Statements
Properties of Algorithms	Operators – Mathematical, Relational, Logical and Unary
Classification of Algorithms	Precedence Rules of Arithmetic Operators
Types of Programming Errors	Mathematical Expressions and rules for writing expressions
Components of an Algorithms	Control Structures – Sequence, Decision making and looping
Variables and their naming	Examples of Algorithms
Variable Declarations	Flowcharts
Variable declarations	Subprograms or subroutines or functions
Variable Initialisation	Arrays

Course Outline

Program Translators :-
Interpreters
Assemblers
and Compilers

Sorting –
Bubble
Selection
Insertion
and Radix Sort

Searching –
Linear
Binary Search

Learning objectives

After reading this unit you should be able to

- Explain what programming is and what is a program
- The software development cycle
- Explain what an algorithm is
- Explain what the properties of an algorithm are
- Different types of programming errors
- Understand what is a variable and how they named
- Explain what goes into writing an algorithm

- Input and output statement
- Operators (Arithmetic, Relational and Logic)
- Expressions
- Control Structures (Sequence, Looping and Decision making)

Learning Objectives

- Writing algorithms
- Flowcharts
- Drawing flowcharts
- Subprograms
- Use of subprograms in an algorithm
- Different types of programming languages
- Programs translators and their functions
- How a program is converted to executable file

Learning Objectives

- The use of Arrays
- Understand sorting and some sorting techniques
- Understand searching and some searching techniques

Computer Programming

- Programming is the act of writing a set of instructions in solving problems. Thus, Computer programmers write instructions for a computer to execute.
- The instructions for the computer to execute are written in a programming language
- The language can be either Machine, Assembler or High level language
- The one who writes a computer program is called a programmer

Software Development Cycle

- Defines the various processes involved in software development and maintenance
- The objective is to produce software product for a customer (user)
- There are four major steps

STEP 1: SOFTWARE SPECIFICATION (PROGRAM DEFINITION)

i. **Program Objectives** – What exactly does the user want

ii. Output - What kind of results is the program expected to generate or produce

iii. **Input Data** - What inputs can generate the expected output

iii. **Processing** – What instructions can generate the expected output from the input data

iv. **Specification document** – all the above must be document for reference

STEP 2: SOFTWARE DEVELOPMENT

This involves using the software specification to develop the required software for the client

- **Program Design** : Deals with the design of the solution using any of the following:
    - Top-Down design (use of modules/subprograms)
    - Pseudocode/Algorithms (Using English and Mathematical expressions in writing the logical set of instructions)
    - Flowcharts (A graphical/pictorial representation of the logical set of instructions)

- **Program code** – Conversion of the algorithm/flowchart into a language that the computer can understand (programming languages)

- **Development documentation**

STEP 3: SOFTWARE VALIDATION

The main aim of this step is to validate the software to see if it does exactly what the customer wants. To do this basically involved two things namely:

- **Program testing** : Using sample input data to see if the output generated is what is expected. The sample data must be a data whose manual output is known.

- **Debugging**: Usually carried out if the testing produce wrong output. It involves locating the error and correcting it.

- **Validation documentation**

STEP 4: SOFTWARE EVOLUTION

- **Maintenance** : Is done to ensure that the software meets the changing need of the customers. This is because customer needs do change over time.

- **Documentation**: To provide description of the program and the procedures used.

ALGORITHMS

- Is a logical sequence of instructions required to solve a given problem or a step-by-step instructions for solving a given problem

- They are normally written using English and Mathematical Expressions

Properties of Algorithms

- **Finiteness** – Capable of producing the required output after a finite number of steps for any input or should be able to terminate after a certain number of steps with the right output.

- **Definiteness-** All instructions should be precise and not to give room for ambiguity

- **Generality** – The instructions should be capable of solving all problems of a particular type or class

Properties of Algorithms

- **Effectiveness** – All instructions should be basic and capable of being performed mechanically within a finite amount of time.

- **Having input and Output Capabilities** – Capable of accepting data as input and generate an expected output

Classification of Algorithms

Two main ways of classifying algorithms, by control shifts and by repetitive steps

- Control Shift – number of alternatives, two alternatives gives **deterministic** algorithm while more than two alternatives gives **non-deterministic**. If the shift is based on a random number then we have **Random** algorithm

- Repetitive – number of repetitive steps. If the number of repetitions is known in advance then we have **direct** otherwise **indirect**

Types of programming Errors

There are three types of error namely

- **Syntax**: violation of a programming rule or not writing an instruction in conformity with the rules of the language being used.

  Examples are: x + y = z instead of z = x + y

  INPUT ABC when A, B and C are variables

  IF A>B<C instead of IF A> B and B<C

When a syntax error occurs, it is usually flagged by the interpreter or the compiler. It can also be detected by the programmer

Types of programming Errors

- **Logical**: error as a result of one’s reasoning. It is usually detected when there is disparity between the program output and the expected output. Example, the roots of the equations $x^2 - 4 = 0$ are 2 and -2. If a program gives 3 and 6 as the roots then there is a problem with the program

- This type of error can only be detected by the programmer or the program user

Types of programming Errors

Runtime: Occurs when a program is executing. Some of the common runtime error messages are as follows:

- Subscript out of range
- Device time out
- Illegal function call
- Overflow

What goes into writing Algorithms

- Variables
- Input statements
- Output statements
- Operator and Expressions
- Looping/iterative/repetitive statements
- Decision making statements

Variables and their naming

- Names used in a computer solution to store data are known as variables
- Variable name must start with either an **English** alphabet (a-z or A-Z) or the underscore symbol (_). If more than one character is used, the rest of the characters should be English alphabets, digits (0-9) or the underscore symbol.
- Variable names can be of any reasonable length
- Variable names must be *Short* and *meaningful* to make our programs easy to read and understand.
- They should not include special characters such as @, #, $

Variables and their naming

- Variable names can be in lower, upper and/or a combination of lower and uppercase characters.
- Note that some programing languages such as C++ are case sensitive as such Pay, PAY and paY are different variable names. In this course, we will assume case sensitivity.
- Variable names must not be keywords or reserved words such as for, while, repeat, until, if, do, repeat. **For this course all words being used as reserved words will be written in UPPERCASES**.
- Whenever two words are concatenated as a variable name, we will capitalise each word such as NetPay, TotalMark, IncomeTax, etc

Variables – Try questions

- Indicate whether the following are valid variable names. Those that are invalid indicate why they are so.

i. d.o.b
ii. Interest-rate
iii. Else
iv. B
v. 7y
vi. You
vii. βeta
viii. interest_rate
ix. IV
x. 5Times_Table
xi. Bra_Kwame
xii. KNUST
xiii. base 2
xiv. why?
xv. program
xvi. ɛtesen
xvii. Office 2013
xviii. sum

Variables Declarations

- Most programming languages such as C++ require that variables be declared before their first usage. Hence we will declare all variables before their usage.
- The format for declaring a variable will be as follows:

DECLARE variable $[, variable]_0^\infty\ AS\ \ dataType$

Where ***datatype*** is one of the following:

INTEGER	for integer numbers
DOUBLE	for real numbers
BOOL	for Boolean (True/False)
CHAR	for variables that store single characters
STRING	for variables that stores more than one character.

Variables Declarations

- Variables that are of the same data type be declared using a single DECLARE statement.

Examples

(i) To declare A, B and C as integers, this can be written as

```
DECLARE A, B, C  AS INTEGER
```

(ii) To declare Sum as integer and Average as a real number we can have

```
DECLARE Sum AS INTEGER
DECLARE Average AS DOUBLE
```

Variables Initialisation

- In some programming languages, when a variable is declared and it is used without being assigned an initial value, it would have an unknown value that can lead to unpredictable outputs when used in an expression especially if the variable appears to the right hand side of the replacement sign.
- Initialise yourself any variable whose initial value is obtained from an input statement or an assignment statement.

Effect of using Integers and Real numbers in an expression

Operator	Integer	Real/Double
Integer	Integer	Real/Double
Real/Double	Real/Double	Real/Double

Examples

Evaluate each of the following expressions bearing in mind that table above.

i. 37/38*38+5.0
ii. 37/38.0*38.0+5.0
iii. 37.0/38*38+5.0
iv. 37.0/38.0*38+5.0
v. 37/38*38+5

Input Statements

- Input statements allow data to be entered into a variable during program execution.
- Note that an input data can be read from a file. For now, input will be entered via the keyboard.
- When we want to ask a user to enter data during program execution, we will make use of any of the following verbs and the format:

$$\left\{\begin{array}{l} \text{- INPUT} \\ \text{- READ} \\ \text{- ACCEPT} \\ \text{- GET} \end{array}\right\} \quad \text{variable } [, variable]_{0}^{\infty}$$

Eg. To receive 10, 5 and 30 into the variables A, B and C, we will write

```
INPUT A
INPUT B
INPUT C
```

Input Statements

- One input statement can be used to receive one or more data. The names of the variables should be separated by commas. The input statements

```
INPUT A
INPUT B
INPUT C
```

Can be written as

```
INPUT A, B, C
```

Output Statements

- Output statements allow information to be displayed on the screen
- Outputs statements also allow information to be written to a file. For now, we want our outputs to be displayed on the screen
- When we want information to be displayed on the screen during program execution, we will make use of any of the following verbs with the format:

$$\left\{\begin{array}{l} \text{- WRITE} \\ \text{- PRINT} \\ \text{- OOUPUT} \\ \text{- DISPLAY} \end{array}\right\} \text{item } [, item]_0^\infty$$

- One output statement can be used for one, two or more items separated by commas

Output Statements

- The output statements can be used to display different kinds of ***item***. The following are typical examples of the ***item***

- WRITE “What is your name?” will display the **string** *What is your name?*
- WRITE A, B will display the values in the **variables** A and B
- WRITE 2*pi*r*r will display the results of the **expression** *2*pi*r*r*

Output Statements

One output statement can be used to display one or more items separated by commas.

```
WRITE “The average age of the students is “, average
```

If the value stored in average is 20.5, then the output will be

The average of the numbers is 20.5

Operators

- **Operators** are symbols that tell the compiler to perform specific mathematical or logical manipulations
- There are four main types of operators namely arithmetic, relational, unary and Logical operators
- **Operators** are usually found between two operands except when they are Unary . An **operand** is any object such as constant, variable, string and function

Mathematical or Arithmetic Operators

- Generally, the following are the arithmetic operators:
  - + for addition,
  - - for subtraction,
  - * for multiplication,
  - / for division, and
  - % for the modulus operation, that returns the remainder of an integer division: e.g. 10 % 3= 1

Precedence Rules of Arithmetic Operators

- Expressions in brackets are evaluated first (starting with inner brackets if any)
- * / % are of the (same) highest order
- + - are the next in order of precedence
- = lowest precedence
- Arithmetic operators of the same order are evaluated from left to right in any given expression

Relational (comparison) Operators

- Generally, the following are the operators used in comparing two items

=	equal to
!= or <>	Not equal to
<	less than
<=	less than or equal to
>	Greater than
>=	Greater than or equal to

(Note how the less than or equal to, and greater than or equal to are written, the symbols cannot be interchanged)

Logical Operators

- These are used when relational tests are to be combined.
- Generally, the logical operators are:

&	and
|	or
^	exclusive or
!	Not

The next table explains how they work

Logical operators

P	Q	P & Q	P | Q	P^Q	!P
TRUE	TRUE	TRUE	TRUE	FALSE	FALSE
TRUE	FALSE	FALSE	TRUE	TRUE	FALSE
FALSE	TRUE	FALSE	TRUE	TRUE	TRUE
FALSE	FALSE	FALSE	FALSE	FALSE	TRUE

Unary Operators

- These operators are used to precede an operand
- The operators are as follows:

-	for negation (read as negative)
+	has no effect (read as positive)

Assignment Statement

- Written using the assignment operator (=)
- The “=“ is also known as the replacement sign
- General syntax is

variable =

Variable	sum = total
Constant	sum = 100
String	FirstName =“John”
Function	sum=sqrt(64)
Expression	sum = sum + 1

Sum = total means replace the value in **sum** with that of **total** or assign the value of **total** to **sum**

Similarly, **sum = sum+1** means add 1 to the value of **sum** and assign the result to **sum**

Mathematical Expressions

- The general syntax for writing an arithmetic expression is

  Variable =operand operator operand [operator operand]

Where

- ***Variable*** is the name of a variable
- ***Operand*** can be a variable, constant or a function
- ***Operators*** are *, /, %, + or –

The “=“ is known as the assignment or the replacement sign and not ***equal to*** even though is generally read “equal to”

Rules for writing mathematical expressions

- Expressions in brackets or parenthesis are evaluated first starting from the innermost set of parentheses and then to the outermost.
- Multiplication (*), division(/) and modulus(%) are evaluated next from left to right
- Next is addition (+) and subtraction(-) are evaluated from left to right.
- No two variables, constant or function must be adjacent to each other. Writing x=zy is wrong if x,y and z are variables, Why?
- No two operators must be adjacent to one another unless the right operator is a urinary operator. Is wrong to write x=y+/z but right to write x=y/-z. Why?
- Only one variable name can appear on the left hand side of the assignment operator, thus x+y=z is wrong, should be z=x+y

Rules for writing mathematical expressions

- There should be no operator on the left hand side of the assignment sign, thus x/y=z is wrong
- Avoid separating thousands by commas in numbers (avoid something like 1,543,200 as a number)

Indicate whether the following expressions are valid or not. Give reason(s) if invalid.

i. Y=3(x-6)
ii. Z=2xy
iii. Sum=sum+1
iv. Sum+1=sum
v. Average =“sum’/n
vi. Total=x/+8

Control Structures

These are very important in any Computer programming.

They are used for control of flow control in programs

There are basically three control structures namely

- Sequence
- Decision making
- Looping

Sequence control structure

- Is also known as Linear control structure
- Defines the order in which program statements are executed
- Generally, program execution is from top to bottom
- Where a line has more than one statement or instruction, the order of execution of the statements is from left to right.

Decision making control structure

- Is used for decision making in choosing between 2 or more alternatives
- The common forms of the decision making control structure are

1. Simple IF statement
2. IF-THEN-ELSE statement
3. IF-THEN-ELSE IF statement

Note: Some programming languages such as the C++ uses the switch statement. We will leave out this for now

The Simple IF statement

- Best used when there is only ONE course of action
- The structure is as follows

```
IF condition is true THEN
    take this action
END IF
```

The **condition is true** a relational expression. The **take this action** is executed only when the relational expression is true. No action is taken if it is false.

Example

```
IF A > 0 THEN
    A=A/2
    B=B*2
END IF
```

The Simple IF statement

The above example requires the following:

(i) A check to see if the current value of A is greater than 0

(ii) If (i) is true then the statements that follows are to be executed, that is

divide A by 2 and assign the result to A as its new value and

multiply B by 2 and assign the result to B.

(iii) If (i) is not true, then don’t execute the two statements, rather, execute the statement after the END IF, if any.

The IF-THEN-ELSE statement

- Best used when there are TWO alternative to choose from
- The general structure is as follows

```
IF condition is true THEN
    take this course of action
ELSE
    take this other course of action
END IF
```

The IF-THEN-ELSE statement

Example

```
IF gender="Male" THEN
   PRINT "The student is a Male"
ELSE
   PRINT "The student is a Female"
END IF
```

The above example assumes that a student can only be a Male or Female.

If the variable **gender** contains Male then the relational expression (sex="Male") is true hence the statement **The student is a Male** will be displayed.

If the variable **gender** does not contain Male, then the student cannot be Male hence **The student is Female** is displayed

The IF-THEN-ELSE IF statement

- Best used when the alternatives are more than two (2)
- Only one of the alternatives is executed at a time
- Once an alternative is executed, the next statement to be executed is that following the END IF
- Also known as Multiway or Cascaded IF statement

- The structure is as follows:

```
IF condition 1 is true THEN
        take action 1
ELSE IF condition 2 is true THEN
        take action 2
ELSE IF condition 3 is true THEN
        take action 3
      :
ELSE IF condition n is true THEN
        take action n
ELSE
        take this default action
END IF
```

The IF-THEN-ELSE IF statement

Example: Let us consider the following grading system of KNUST

Exams Score	Grade
70 and above	A
60 to 69	B
50 to 59	C
40 to 49	D
Below 40	F

Assume we are to accept a student’s exam score as input and output the corresponding grade. We will require an input statement, decision making statement and output statement in that order.

The IF-THEN-ELSE IF statement

Solution

```
INPUT ExamScore
IF ExamScore >= 70 THEN
    PRINT ExamScore, “ is a Grade A”
ELSE IF ExamScore >= 60 THEN
    PRINT ExamScore, “ is a Grade B”
ELSE IF ExamScore >= 50 THEN
    PRINT ExamScore, “ is a Grade C”
ELSE IF ExamScore >= 40 THEN
    PRINT ExamScore, “ is a Grade D”
ELSE
    PRINT ExamScore, “ is a Grade F”
END IF
```

Looping control structure

- Also known as the repetitive or iterative control structure
- Use to repeat the execution of statement(s) for specified number of times or until when a certain condition is met.
- The common looping statements are as follows:

1. FOR loop
2. WHILE loop
3. DO-WHILE loop
4. REPEAT-UNTIL loop

FOR loop Statement

- Best used when the number of times to loop is known in advance
- Usually has three commands/statements

- The first is used to set a starting value
- The second is the end condition or the last value
- The third is used to modify a value used in the block. The modifier cannot be 0.

- Also known as a pre-test loop. Thus, the loop condition is tested before the loop statement(s) is/are executed.
- The general structure is as follows:

FOR loop Statement

```
FOR control_variable = StartingValue TO EndingValue [STEP value]
    loop statement(s)
ENDFOR
```

Where

StartingValue is used to intialise the **control variable. Control_variable** is a variable used to regulate the looping or the repetition of the loop.

EndingValue specifies the end condition or the last value of the control variable

STEP value is optional. When used, it specifies how the value of the control variable should be modified for the next iteration. For all values other than 1, the STEP value must be specified. If the STEP value is not specified then it assumes the default value of 1

FOR loop Statement

Example 1: Assume we want a loop to display the numbers 1 to 100. Let i be our control variable. The loop can be written as follows:

```
FOR i=1 TO 100 STEP 1
    PRINT i
ENDFOR
```

In the above example,

i. The control variable (i) is first initiliased to 1

ii. The value of the control variable is then checked against the 100 to see if we are within the range and will it be possible to get to the 100

iii. If (ii) is true, the statement ***PRINT i*** displays the current value of i.

iv. ENDFOR causes the loop to repeat by first checking (ii)

v. The looping stops when (ii) becomes false. For example when i is 101 the control variable will be out of range and hence causes the looping to stop

FOR loop Statement

Example 2: Assume we want a loop to print all even numbers between 1 and 500.

Solution: Let our control variable be *EvenNumbers*. Note that the first even number in the range is 2 and successive even numbers differ from each other by 2. We can write our loop as follows:

```
FOR EvenNumbers = 2 TO 500 STEP 2
    PRINT EvenNumbers
ENDFOR
```

Note that the STEP value is 2 because of how successive even numbers differ from each other.

FOR loop Statement

Example 3: Indicate how many times the statement “I Love Programming” will be displayed on the screen.

```
FOR ManyTimes = 1 to 100 STEP -1
    PRINT “I Love Programming”
ENDFOR
```

Answer: 0

Reason: If we start from 1 and our step value is -1 there is no way one can ever get to 100 hence the loop is invalid and will not be executed at all.

FOR loop Statement

For each of the following loops, indicate the number of times the loop will run.

-
  ```
  FOR I=100 to 40
            :
  ENDFOR
  ```

-
  ```
  FOR I= 100 to 40 STEP 3
            :
  ENDFOR
  ```

-
  ```
  FOR I=100 to 400 STEP -1
            :
  ENDFOR
  ```

-
  ```
  FOR I=100 to 400 STEP 1
            :
  ENDFOR
  ```

WHILE loop

- Best used when the number of times to loop is not known in advance
- The loop repeats as long as a condition has been met
- Is also known as a pre-test loop, thus, the condition is tested before each execution of the loop
- The general structure is as follows:

```
WHILE (condition is true)
    statement(s)
ENDWHILE
```

Where

condition is true is a relational expression

WHILE loop

Example 1

```
i = 1
WHILE (i <= 100)
    PRINT i
    I = i+1
ENDWHILE
```

(i) In the above example, i controls the loop hence i is the control variable.

(ii) The control variable is first initialized to 1 before the start of the loop

(iii) Starting the loop, the value of i is checked to see if it is less than or equal to 100

(iv) Since (iii) is true, the loop statements are executed by displaying the value of i and then incrementing the value of i by 1

(v) The loop is repeated until the value of i is 101 which will make i<=100 false

WHILE loop

Example 2: Example of when it is best to use a WHILE loop. Assume we have a certain number of numbers to sum and that we are told the number 100 is not part of the list and that it just mark the end of the list. Here, we don’t know the total number of the numbers in the set. Our loop can be written as follows:

```
DECLARE sum, number AS INTEGER
sum=0
INPUT number
WHILE (number != 100)
    sum = sum + number
    INPUT number
ENDWHILE
```

WHILE loop

First line: Allows for the first number in the set to be entered

Second line: Checks to see if the number entered is not 100

Third line: Adds the number entered if it is not a 100

Fourth line: request for the next number in the set

Fifth line: Causes the loop to be executed again, if needed

DO-WHILE statement

- Best used when the number of times to loop is not known in advance
- The loop repeats as long as a condition has been met
- Is also known as a post-test loop, thus, the condition is tested at the end of each execution of the loop
- This loop is guaranteed to execute at least one (1) time
- The general structure is as follows:

```
DO
    statement(s)
WHILE (condition is true)
```

Where

condition is true is a relational expression

DO-WHILE statement

Example 1

```
i = 1
DO
    PRINT i
    i = i+1
WHILE(i <= 100)
```

(i) In the above example, i controls the loop hence i is the control variable.

(ii) The control variable is first initialize to 1 before the start of the loop

(ii) The loop statements are then executed by displaying the value of i and then incrementing the value of i by 1

(iv) The value of i is checked to see if it is less than or equal to 100

(v) The loop is repeated until the value of i is 101 which will make i <=100 false

DO-WHILE statement

Using the same example 2 under the WHILE loop, the solution can be written as follows using the DO-WHILE

```
INPUT number
DO
    sum = sum + number
    INPUT number
WHILE (number != 100)
```

REPEAT-UNTIL

The REPEAT UNTIL is functions exactly the same way as the DO-WHILE. Most programming language have either the DO-WHILE or the REPEAT-UNTIL and not both

In a DO-WHILE, simply replace the DO and the WHILE with REPEAT and UNTIL respectively

Example 1

```
i = 1
REPEAT
    PRINT i
    i = i+1
UNTIL (i <= 100)
```

REPEAT-UNTIL statement

Example 2

```
INPUT number
REPEAT
    sum = sum + number
    INPUT number
UNTIL (number != 100)
```

LOOPING

- Please note that for software development purposes you are to use any of the looping statement you are comfortable with. However, for examination purposes you can be restricted to which loop to use in solving a given problem

- Having dealt with the control structures, we are now in a position to try our hands on some examples.

Problem 1: An algorithm to find average of two numbers

```
INPUT FirstNumber
INPUT SecondNumber
Sum = FirstNumber + SecondNumber
Average = Sum/2
PRINT Average
```

Problem 2: An algorithm to find average of three numbers

```
INPUT FirstNumber
INPUT SecondNumber
INPUT ThirdNumber
Sum = FirstNumber + SecondNumber+ThirdNumber
Average = Sum/3
PRINT Average
```

Problem 3: An algorithm to find average of four numbers

```
INPUT FirstNumber
INPUT SecondNumber
INPUT ThirdNumber
INPUT FourthNumber
Sum = FirstNumber + SecondNumber+ThirdNumber+FourthNumber
Average = Sum/4
PRINT Average
```

If we look at solutions 1 to 3 careful, we see that anytime the number of numbers to averaged increases there is a corresponding increase in the number of variables. The problem with the solutions are that:

- Each of them is not a generalized solution
- We can use a single variable for all inputs

This will result in the following solution:

```
INPUT Number               to get first number
Sum = Sum + Number         to add first number
INPUT Number               to get second number
Sum = Sum + Number         to add second number
INPUT Number               to get third number
Sum = Sum + Number         to add third number
INPUT Number               to get fourth number
Sum = Sum + Number         to add fourth number
Average = Sum/4
PRINT Average
```

We can see from the solution above that the statements

```
INPUT Number
Sum = Sum + Number
```

have been repeated 4 times, 4 representing the number of numbers. We can therefore use a FOR loop for the solution as follows:

```
Sum = 0
FOR i = 1 TO  4
    INPUT Number
    Sum = Sum + Number
ENDFOR
Average = Sum / 4
Print Average
```

Even though the use of the FOR loop has improved the solution, it is still not generalized. We can generalized it if we represent the number of the numbers to averaged by a variable, say NoOfNumbers. Then our solution can be modified as follows:

```
Sum = 0
INPUT NoOfNumbers
FOR i = 1 TO  NoOfNumbers
        INPUT Number
        Sum = Sum + Number
ENDFOR
Average = Sum / NoOfNumbers
Print Average
```

- By the modified solution above, the algorithm works for any number of input and hence a generalised solution

- Whenever you write an algorithm, please note the following

- Minimise the number of variables used, if possible
- Make your program as short as possible. This ensure it does not take up too much storage space
- Be conscious of the time your algorithm will take to run. Clients are not interested in solutions that takes much time to run

Problem 4: Employees of a certain firm are paid on hourly basis. If an employee works for not more than 40 hours a week, the hours worked is considered regular and Overtime for hours worked beyond 40. Regular hours are paid at 5 cedis per hour while the overtime rate is one and half times the regular rate per hour. All employees are to pay 15% of their gross pay as Income Tax, 2.5% as National Health Contribution Levy, 1% as District Tax. Employees who have more than three children are to pay 50 pesewas per child in excess of three towards Educational Fund For All. Devise a computer solution that can be used to calculate the necessary deductions as well as the Net Pay of employees. Your display for each employee each deduction, net pay and the gross pay with appropriate captions.

Let us pick the relevant information from the question and represent them by some names

ITEM DESCRIPTION	VARIABLE NAME
Number of Employees	NoOfEmployees
Number of hours worked by an employee	HoursWorked
Rate for payment of regular hours	RegularHours
Rate of payment of Overtime hours	OvertimeHours
Gross pay of an Employee	GrossPay
Income tax of an Employee	IncomeTax
National Health Insurance Levy of an employee	NHIL
District Tax payable by employee	DistrictTax
Number of children an Employee has	NoOfChildren
Payment towards Educational Fund for All	EduFund
All deductions	Deductions
Net pay of an employee	NetPay

Analysis of the problem

To solve the problem, we must do the following in the order presented:

(i) First, the number of employees should be provided as input

(ii) We then use a loop for items (iii) to (vi)

(iii) Get input on an employee, that is hours worked and the number of children

(iv) Most deductions are dependent on the grass pay hence we calculate the gross pay first

(v) We then calculate each of the deductions

(vi) We display the required output

Solution 4

```
DECLARE RegularRate, NoOfEmployees, HoursWorked,NoOfChildren, employee AS INTEGER
DECLARE GrossPay, OvertimeRate, IncomeTax,NHIL, DistrictTax,EduFund, NetPay AS DOUBLE
RegularRate = 5
OvertimeRate = 1.5*RegularRate
INPUT NoOfEmployees
FOR employee = 1 TO NoOfEmployees STEP 1
   INPUT  HoursWorked, NoOfChildren
   IF HoursWorked <= 40 THEN
       GrossPay = RegularRate * HoursWorked
    ELSE
        GrossPay = RegularRate * 40 + OvertimeRate* (HoursWorked-40)
    ENDIF
    IncomeTax = 0.15 * GrossPay
    NHIL = 0.025 * GrossPay
    DistrictTax = 0.01* GrossPay
    IF NoOfChildren > 3 THEN
        EduFund = 0.5 * (NoOfChildren - 3)
    ELSE
         EduFund = 0
    ENDIF
    NetPay = GrossPay – (IncomeTax+NHIL+DistrictTax+EduFund)
    PRINT “The Gross pay is “, GrossPay
    PRINT “The Income Tax is “, IncomeTax
    PRINT “The National Health Levy is “, NHIL
    PRINT “The District Tax is “, DistrictTax
    PRINT “The Education Fund Contribution is ‘, EduFund
    PRINT “The net pay is “, NetPay
ENDFOR
```

Problem 5

Analysis

i. Declare your variables

ii. Accept as input the first number from the set and assume it to be the largest

iii. Looping (n-1) times, receive each of the remaining numbers as input

  - Compare each number against the largest

  - If an input number is greater than the largest, then replace the largest with the current input

iv. Display the value in largest as the largest number.

Problem 5

Given a set of n integers, write al algorithm for selecting the largest number from the set.

Solution

```
DECLARE n, i, FirstNumber, NextNumber, Largest AS INTEGERS
INPUT FirstNumber
Largest = FirstNumber
FOR i=1 to n-1 STEP 1
    INPUT NextNumber
    IF NextNumber >Largest THEN
        Largest = NextNumber
    END IF
ENDFOR
WRITE “The Largest number is “, Largest
```

Problem 6

Write an for finding the roots of any given quadratic equation. A quadratic equation is of the form $\mathrm{a}x^2 + bx + c = 0$ where a, b and c are coefficients. The roots of an equation mat=y be real or imaginary. If the roots are real then the roots are given by $\frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$. If the roots exits your solution should indicate whether they are equal of not. Message should be displayed if the roots a imaginary.

Problem 6

Analysis

i. Declare the required variables
ii. Receive the a, b and c as inputs
iii. Calculate the discriminant (i.e $b^2 - 4ac$)
iv. If the discriminant is zero then we have two equal roots
v. Else If the discriminant is greater than 0 then we have 2 unequal roots
vi. Else the roots are imaginary

Solution

```
DECLARE a, b, c AS INTEGERS
DEACLARE Disc, Root1, Root2 AS REAL
INPUT a, b, c
Disc = b*b-4*a*c
IF Disc = 0 THEN
    Root1 = Root2=b/(2*a)
    WRITE “Roots are EQUAL and are “,Root1,” and “, Root2
ELSE IF Disc > 0 THEN
   Root1=(-b+√Disc)/(2*a)
   Root2=(-b-√Disc)/(2*a)
   WRITE “Roots are UNEQUAL and are “,Root1,” and “, Root2
ELSE
   WRITE “Roots are IMAGINARY “
ENDIF
```

Problem 7

Write an algorithm for finding the factorial of any given positive integer, n. Given a positive integer n, its factorial is calculate as n!=n(n-1)(n-2)(n-3)……4*3*2*1

Analysis

i. Declare the required variables (say n, i and factorial)

ii. To begin with, let the factorial have an initial value of 1

iii. Get as input the value of n

iv. Check if n is a positive number

v. Loop from n down to 1

  At each loop multiply the control variable and factorial value and store the result in factorial

iv. Display the value in factorial

solution

```
DECLARE n, i, Factorial AS INTEGERS
Factorial = 1
DO
    PRINT "Enter a positive integer "
    INPUT n
While (n<0)
FOR i=n to 1 STEP -1
    Factorial =Factorial * i
ENDFOR
PRINT "The factorial of ",n," is ",Factorial
```

$\}$ The DO-WHILE Loop ensures that the factorial is calculated only when a positive integer has been entered by the user.

Problem 8

Write an algorithm to display all the factors of a positive integer, n

Analysis

i. Declare your variables

ii. Receive as input the value of n

iii. Ensure that n is a positive number

iv. Loop from 1 to $\frac{1}{2}n$

    Display all values of the loop that divides n without a remainder

v. Finally, display n

solution

```
DECLARE n, i  AS INTEGERS
DO
    PRINT “Enter a positive number “
    INPUT n
While (n<0)
PRINT “The factors of n are “
FOR i=1 TO  n/2 STEP 1
  IF n % i THEN
        PRINT “ “, i
  ENDIF
ENDFOR
PRINT “ “,n
```

$\}$ The DO-WHILE Loop ensures that the input number is positive

Problem 9

Write an algorithm to determine whether a given positive integer, n is a prime number.

Analysis

i. Declare variables

ii. Set a variable MoreFactor to false

iii. Accept n as input

iv. Loop from 2 to $\frac{1}{2}n$

  If a factor is found within the range,

   - Change the value of MoreFactor to True

   - Display a message that the number is NOT a prime.

   - Exit the loop

 Use EXIT as a reserve word to exit a loop. Most programming have commands to exit a loop

v. Test if MoreFactor is True. If it is then display a message that the number is a prime.

Solution 1

```
DECLARE I, n AS INTEGER
DECLARE MoreFactor AS BOOL
MoreFactor = FALSE
INPUT n
FOR i = 2 TO n/2 STEP 1
     IF n % i = 0 THEN
           PRINT n, “ is NOT a Prime Number”
             MoreFactor = TRUE
             EXIT
      ENDIF
ENDFOR
    IF MoreFactor = TRUE THEN
          PRINT n, “ is a Prime Number”
   ENDIF
```

Alternative analysis to problem 9

Analysis

i. Declare variables

ii. Set a variable NumberOfFactors to 0

iii. Accept n as input

iv. Loop from 2 to $\frac{1}{2}n$

 Each time a factor is found within the range, increase NumberOfFactors by 1.

v. On exiting the loop, if the NumberOfFactors = 0 then the number is a prime otherwise it is not

Alternative Solution

```
DECLARE i, n, NumberOfFactors AS INTEGER
NumberOfFactors = 0
INPUT n
FOR i = 2 TO n/2 STEP 1
    IF n % i = 0 THEN
          NumberOfFactors = NumberOfFactors + 1
   ENDIF
ENDFOR
   IF NumberOfFactors = 0 THEN
        PRINT n, " is a Prime Number"
   ELSE
        PRINT n, " is NOT a Prime Number"
ENDIF
```

Brofoyedur is a small village in the Central Region of Ghana. It has been estimated that the population of this town is about 2000 people and that the population is increasing at a rate of 15% every six months. You are required to write a computer solution to determine the number of years that it would take for the population size to exceed two million for the village to become a city.

Analysis

```
i.   Declare variables
ii.  Set the population size to 2000
iii. Set also the number of years it will take to 0
iv.  While the population of the village is less than 2000000,
          - calculate an increase of population by 15 percent
          - increase the number of years by 0.5 (i.e. 6 months)
v. Display the population size and the years
```

Solution 10

```
DECLARE PopulationSize AS INTEGER
DECLARES Years AS REAL
PopulationSize = 2000
Years = 0
WHILE (PopulationSize <= 2000000)
        PopulationSize = 1.5 *PopulationSize
        Years = Years + 0.5
WHILEEND
PRINT “The population size will be “,PopulationSize,” and it will take “,
Years,” to achieve that.”
```

We can generalised the solution 10 above as follows:

```
DECLARE PopulationSize, MinimumPopulation, IncreaseRate AS INTEGER
DECLARES Years AS REAL
Years = 0
INPUT PopulationSize, MinimumSize, IncreaseRate
WHILE (PopulationSize <= MinimumSize)
        PopulationSize = (1+IncreaseRate) *PopulationSize
        Years = Years + 0.5
WHILEEND
PRINT “The population size will be “,PopulationSize,” and it will take “,
Years,” to achieve that.”
```

Note: The solution is not 100% generalised but better than initial solution

Hands-on Questions

Q1:The square root of a number N can be approximated by a repeated calculation using the following formula.

$$NewGuess = 0.5(LastGuess + N / LastGuess)$$

where *NewGuess* is the next guess and *LastGuess* the last guess. The calculation of a *NewGuess* should be terminated when the absolute value of the difference between the *NewGuess* and the *LastGuess* is about 0.0000001. Write down a computer solution for the above formula. You may use the function ABS(x) or '|x|' to obtain the absolute value of x.

Q2. A number N may be said to be either a perfect, deficient or abundant number. If the sum of divisors (excluding the number itself) equals the original number, the number is said to be perfect. If the sum of the divisors is less than the number itself then the number is said to be deficient otherwise the number is said to be abundant. For example, the number 12 has the divisors 1, 2, 3, 4 and 6. The sum of these divisors is 16. Since the sum of the divisors of 6 is greater than 6, we will say 6 is an abundant number. Write down a computer solution that can classify a given number as a perfect, abundant or a deficient number with an appropriate caption.

Q3. The RUSSIAN PEASANT method is one way of finding the product of any two given integers. Given that A and B are two positive numbers, A is divided by 2 and its decimal part truncated, while B is also multiplied by 2. This process of dividing and multiplying is repeated until A attains a value of 1. The product of the two numbers is then the sum of all B values whose corresponding A values (including the initial value of A) are odd. As a typical example, let A and B be 21 and 10 respectively then the division and the multiplication can be written down as follows:

Current value of A	Current value of B
10	
10	20
5	**40**
2	80
1	**160**

In the above, the bolded lines show when the value of A is odd. The product of the two numbers will therefore be the sum of the bolded B values, which is 210 (10+40+160). Write a computer solution that accepts as input two positive integers and then return the product of the two numbers using the above method with an appropriate caption. Hint use the modulus operator (%) where necessary.

Q4. Write an algorithm for evaluating the following expression for a given value of *x* and *n*. If the value of *n* is negative display the error message *value of n must be greater than or equal to 0*.

$$(1+x)^n = 1 + \frac{nx}{1!} + \frac{n(n-1)x^2}{2!} + \cdots$$

Q5. Assume you are the captain of JB and SONS Airline Ltd (JBSL) and you are approaching Britain, a country that still measures distances in miles, yards and feet. Your range finder unfortunately can read only distances in feet to the nearest whole number. Fortunately enough, your plane has a small personal computer on which the C++ compiler has been installed. It has become very necessary for you to write a simple computer solution that will later be coded in C++ language to convert a number of distances in feet to miles, yards and feet. You should accept as input a distance in feet and display the equivalent in miles, yards and feet. If a term is zero, it should not be printed.

For example

287 should be displayed as 1 mile 2 yards 1 foot

5279 should be displayed as 1759 yards 2 feet

Note: 1 mile = 1760 yards and 3 feet = 1 yard.

Q6. Write an algorithm that will allow the first and the second terms of the Fibonacci series to be entered as input and generate the series up to an nth terms. The n must also be entered as input.

Q7. A friend of yours on hearing that you have just been taught Principles of Programming at the Department of Computer Science, KNUST wants you to write an algorithm for him. He wants the algorithm to accept any date in the form day month year as three different inputs. He expects the algorithm on entering the inputs, for example, 10 12 2020 to return the date in the form 10th December 2020. You are to put in checks to ensure that only valid dates are accepted as input.

Q8. Write an algorithm to reverse the digit in a given input. Use only to Print statement to print the reverse digits.

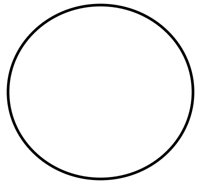
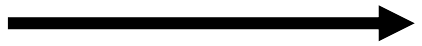
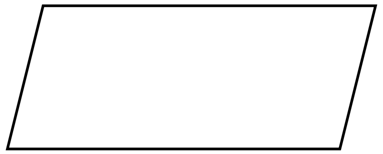
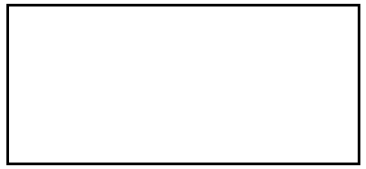
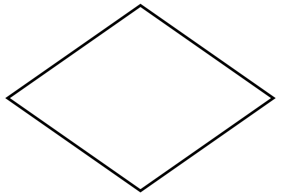
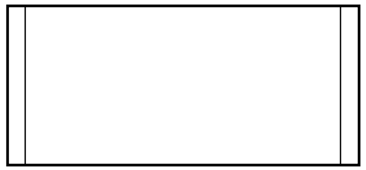
Flowcharts

- A flowchart is a graphical representation of an algorithm.
- It shows steps as boxes of various kinds in sequential order by connecting them with arrows.

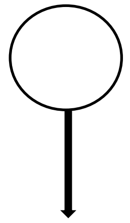
- The following are some of the commonly used symbols in flowcharting

- When a symbol is used, the instruction must be written within the symbol
- The same symbol can be used for more than one instruction of the same type
- Every flowchart must have a start (beginning) and a stop (end)
- From the start, each path that is followed must always lead to the stop

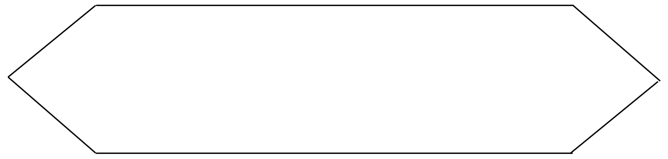
Common symbols used in flowcharts

	Start/End
	Logic flow arrow
	Input/Output
	Compute/Process
	Decision
	predefined process

Common symbols used in flowcharts



Connector

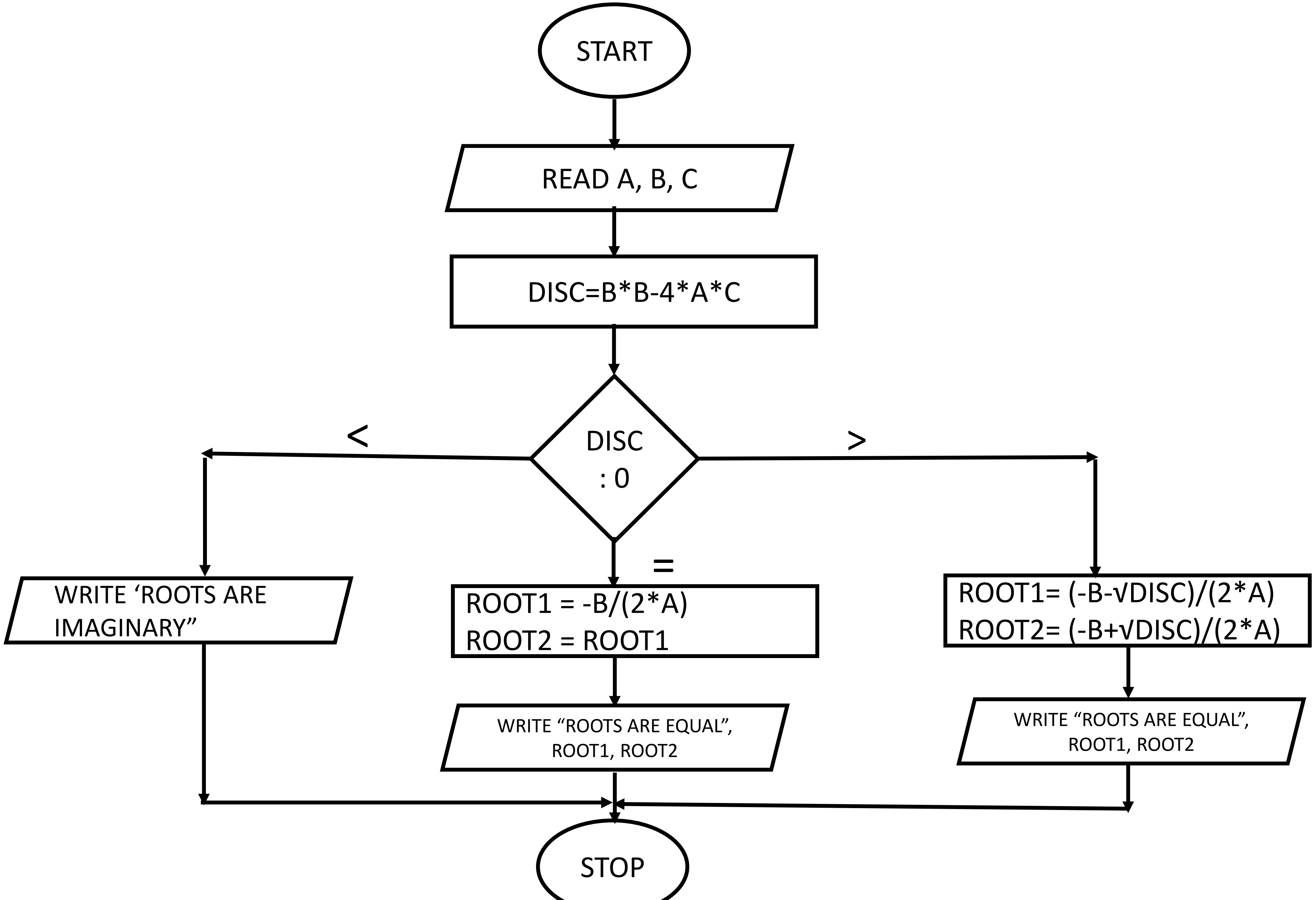


Loops

Example of a flowchart

As an example, let us convert the solution 6 below to a flowchart

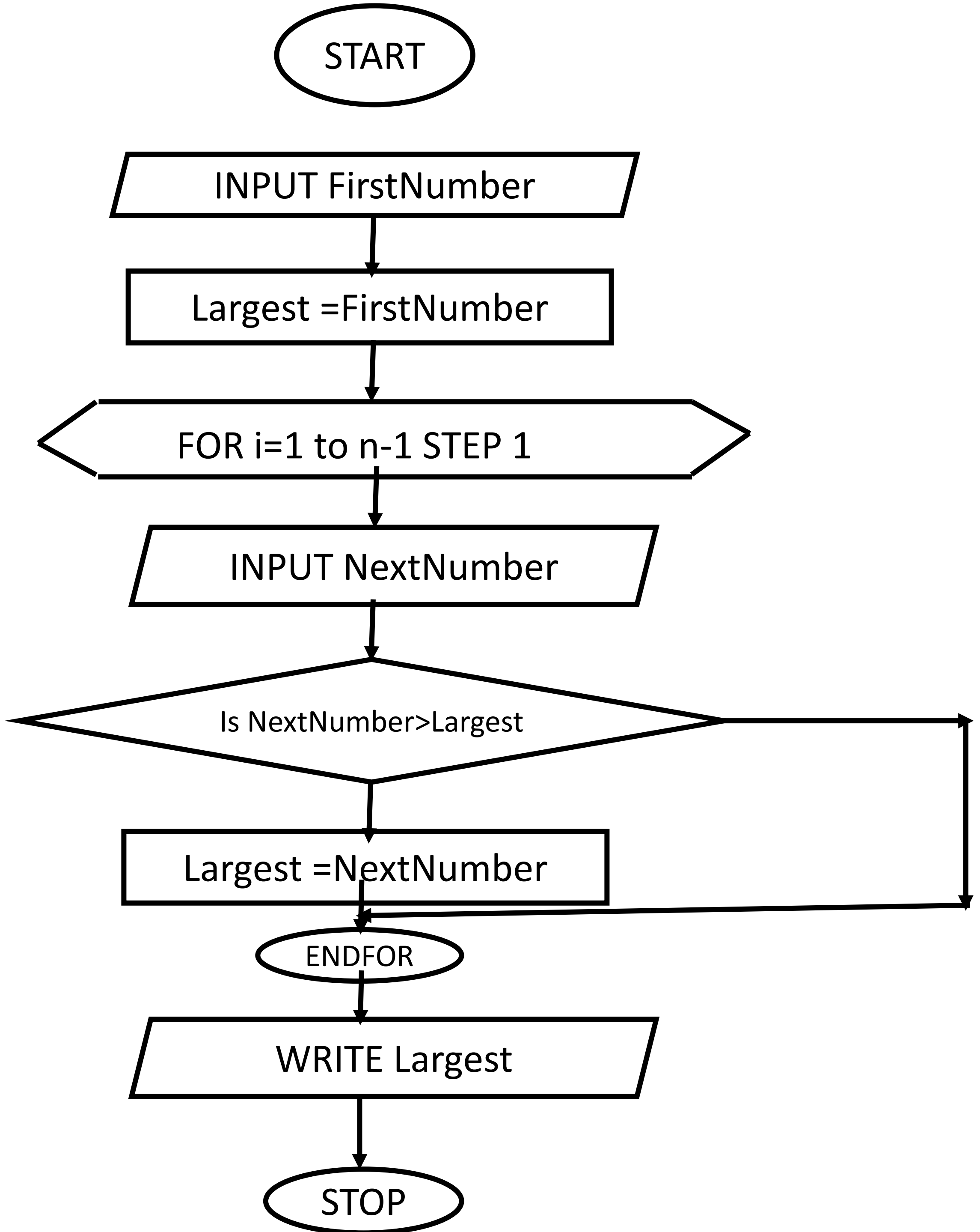
```
DECLARE a, b, c AS INTEGERS
DEACLARE Disc, Root1, Root2 AS REAL
INPUT a, b, c
Disc = b*b-4*a*c
IF Disc = 0 THEN
    Root1 = Root2=b/(2*a)
    WRITE “Roots are EQUAL and are “,Root1,” and “, Root2
ELSE IF Disc > 0 THEN
   Root1=(-b+√Disc)/(2*a)
   Root2=(-b-√Disc)/(2*a)
   WRITE “Roots are UNEQUAL and are “,Root1,” and “, Root2
ELSE
   WRITE “Roots are IMAGINARY “
ENDIF
```


START
READ A, B, C
DISC=B*B-4*A*C
DISC
: 0
<
>
=
WRITE ‘ROOTS ARE
IMAGINARY”
ROOT1 = -B/(2*A)
ROOT2 = ROOT1
ROOT1= (-B-√DISC)/(2*A)
ROOT2= (-B+√DISC)/(2*A)
WRITE “ROOTS ARE EQUAL”,
ROOT1, ROOT2
WRITE “ROOTS ARE EQUAL”,
ROOT1, ROOT2
STOP

Let us also convert solution 5 to flowchart

Solution

```
DECLARE n, i, FirstNumber, NextNumber, Largest AS INTEGERS
INPUT FirstNumber
Largest = FirstNumber
FOR i=1 to n-1 STEP 1
    INPUT NextNumber
    IF NextNumber >Largest THEN
        Largest = NextNumber
    END IF
ENDFOR
WRITE “The Largest number is “, Largest
```


START
INPUT FirstNumber
Largest =FirstNumber
FOR i=1 to n-1 STEP 1
INPUT NextNumber
Is NextNumber>Largest
Largest =NextNumber
ENDFOR
WRITE Largest
STOP

Subprograms and their importance

- A subprogram is a sequence of program instructions that performs a specific task, packaged as a unit.
- They enhance program readability and understanding
- Subprograms may be defined within programs, or separately in libraries that can be used by many programs.
- A subprogram may be called a ***routine***, ***subroutine***, ***function***, ***method***, ***procedure*** or **predefined process**

Structure of Subprograms

We shall adapt the following structure for our functions

```
DECLARE FunctionName[(Argument(s))]
    body of function
    RETURN DataToReturn
END FUNCTION
```

Where

FunctionName is the name to be given to the function when called

Argument(s) is the input to the function

body of function are the processes to obtain the **DataToReturn**

DataToReturn is what we expect the function t return after carrying out its task

Subprograms

- Each subprogram has a single point of entry
- The calling program or subprogram is suspended during the execution of the called program
- When the subprogram’s execution terminates, the control returns to the caller program or subprogram

Subprograms

- Generally, subprograms go by different names. This depends on the programming language being used. Examples are
- Subroutine – FORTRAN, Visual Basic, Perl
- Procedure – Pascal
- Function – Pascal, C, C++, Java, JavaScript, Python
- Super Expression – GML, GMPL
- Method - Java

Types of variables

- Variables in a computer program can be either Global or local
- Variables declared outside the body of any function is a Global variable while those declare in a function are called Local variables.
- Global variables can be used in any function without having to declare them. Thus, the scope of a global variable is the entire program
- Local variables can only be used in the function in which it is declared. Thus, the scope is limited to the function in which it was declared.

Example of using subprograms

Let us consider an algorithm for finding the number of possible combinations that can be obtained by taking a sample of items (r) from a larger set (n)

$$C(n,r) = {}^{n}C_{r} = {}_{n}C_{r} = \binom{n}{r} = \frac{n!}{r!(n-r)!}$$

where ***n*** is the number of items to choose from, and we are to choose r of them. There are no repetitions and the order of the items is not important.

The solution without using subprogram(s)

```
DECLARE n, r, nFactorial, rFactorial, nrFactorial,i AS INTEGER
DECLARE nCr AS REAL
nFactorial=rFactorial=nrFactorial=1
DO
        INPUT n, r
WHILE (r>n || r<0)
FOR i=n TO 1 STEP -1
   nFactorial=nFactorial*I
ENDFOR
FOR i=r TO 1 STEP -1
   rFactorial=rFactorial*I
ENDFOR
FOR i=n-r TO 1 STEP -1
   nrFactorial=nrFactorial*I
ENDFOR
nCr=nFactorial/(rFactorial*nrFactorial)
WRITE “The Combination (“,n,”,”,r,”) is “,nCr
```

Solution using a subprogram

- Looking at the solution above, the FOR loop is being repeated 3 times. Each of the loop just calculates the factorial of a number
- If we had a built-in or predefined function or a subprogram that can return the factorial of a number we could have avoided the repletion of the loop
- Since we have no function in most programming languages that return the factorial of number we can write our own for use.

```
DECLARE n, r, Facto AS INTEGER
DECLARE FACTORIAL (m)
    Facto=1
    FOR i=m TO 1 STEP -1
        Facto=Facto*i
    ENDFOR
    RETURN Facto
END FUNCTION
DO                                  Function calls
        INPUT n, r
WHILE (r>n || r<0)
nCr=FACTORIAL(n)/(FACTORIAL(r)/(FACTORIAL(n-r))
WRITE "The Combination (",n,",",r,") is ",nCr
```

Try question using subprogram

Write an algorithm to evaluate the following given that your inputs are n and x.

$$(1+x)^n = 1 + \frac{nx}{1!} + \frac{n(n-1)x^2}{2!} + \cdots$$

Note that n≥0

Arrays

So far we have been using variables that can hold only a single value at any given time in our algorithms. It is possible for a variable to hold more that one value at a time. Such a variable is known as Array.

The characteristics of an array are as follows:

i. An array is a collection of items stored at contiguous memory locations.

ii. The items of an array must be of the same data type

iii. Each element or item can be uniquely identified by its index in the array

iv. An array is indexed by a subscript. The index of the first element can be 0 or 1 depending on which programming language is being used. For this course, we will the index of the first element to be 1.

Arrays

- An array may be a single or multi-dimensional but the elements are stored in contiguous memory locations be it for a single or multi-dimentional
- An array must have a name
- Arrays allow random access to elements.
- Very useful in implementing other data structures such as Lists, Stacks, Queues, Heaps, trees and Hash tables
- Once an array is declared you can’t change its size because of static memory allocated to it. However, some programming languages allows for dynamic arrays.
- Insertion and deletion are difficult as the elements are stored in consecutive memory locations. Inserting and deletion of an element may require other elements to be shifted and the shifting operation is costly too.

Arrays

The following shows a typical array

Program Translators

- A program translator may be defined as any computer package capable of translating a source program written in either a high level language or an assembly language into a machine language.

- There are three types of program translators namely ***Compilers***, ***Interpreters***, and ***Assemblers***.

Interpreter

- Each line of program instruction is translated on each occasion the program is run.
- If any statement violates the syntax of a language, an error message is displayed on the screen.
- Until the error is corrected, program execution cannot proceed; this slows down the running of the program, while errors are more quickly corrected as compared to compilers.

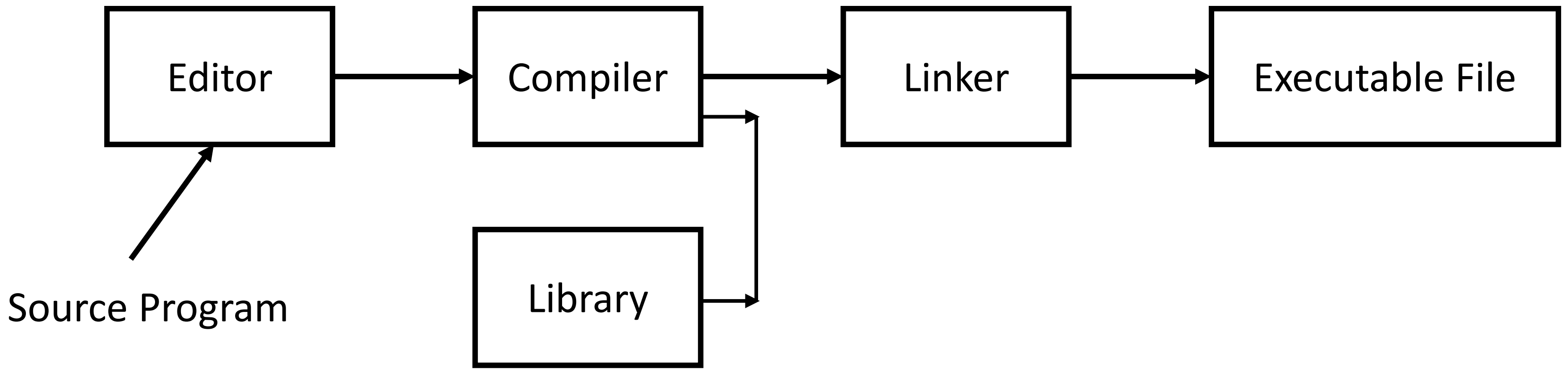
Assembler

- Assemblers translate and assemble a program written in assembly code into machine (object) code.
- Symbolic function codes (example: MOV) are translated into the equivalent machine operation code.
- After assembling, the object program is retained on a storage device in machine code. Error are diagnosed during the assembly process.
- This process occurs once, and does not have to be performed each time a program is run as with an interpreter.

Compiler

- A compiler translates high level language instruction into machine code. Each source program instruction generates a number of machine code instruction.
- A compiler is usually used together with a program called linker, capable of linking together the different translated modules.
- A list of errors which have to be corrected as a whole are printed by the compiler provided they are found in the program.

Converting a Source program into executable file



Sorting

- Sorting is the process of arranging data into numerical or alphabetical order.
- The order of the arrangement may be ***ascending*** or ***descending***.

- With respect to records in a file or database, sorting is based on one or more record fields usually referred to as ***record key(s)*** or ***key field(s)***.

- Sorting techniques to be considered are Bubble, Selection, Insertion, and Radix sorts.

Bubble Sort

- This method of sorting works by comparing adjacent items.
- If they are found to be out of order, they are interchanged.
- The first pass begins with N-1 comparisons; N=number of items to be sorted.
- At the end of each pass, the number of comparisons is reduced further by 1.

Selection Sort

- Selection sort involves selecting either the largest or the smallest entry in a list and interchanging it with the last or the first position items (depending on the programmer).

- The next largest or smallest entry is selected and interchanged with the element at the second or the last but one position, etc.

Insertion Sort

- This method of sorting is similar to the selection sort.
- The number of passes = N; N = number of items to be sorted.
- From right to left, the number of items in the sorted segment = the passth entry. For pass 1, the first item is assumed to be sorted and hence in the sorted segment. For pass 2, the first and second items are assumed to be in the sorted segment and sorted in the required order, etc., through to pass N.

Radix Sort

- Radix sort is a non-comparative sorting algorithm that sorts a list of integers based on their individual digits. It is also called ***bucket sort***.

- Sorting is performed from the least significant digit to the most significant digit.

- Number of passes = number of digits in the greatest integer among the list of integers.

- Integers are grouped in buckets/radixes from 0 – 9 based on their end-digits, and then sorted in the required order after the final pass.

Searching

- Searching is a method of locating a specific item in a larger collection of items.

- A written program may have to locate specific items within an array in addition to storing and manipulating them.

- Two searching algorithms are considered: ***Linear*** and ***Binary searches****.*

Sequential or Linear Search

- Linear search is carried out by beginning the search with the first item of the collections and then through to the last item.
- The collection of items may or may not necessarily be sorted.
- If the items are already sorted, then the search can be stopped as the item is found, or an item greater or lesser than the item being searched for is encountered depending on the arrangement order.

Binary Search

- Binary Search is more efficient than Linear search.
- For Binary Search, the collection of items to be searched must be sorted.
- Anytime a comparison is made and the item being searched for is not the desired item, the Binary Search eliminates half of the remaining items to be searched.