



Cambridge Assessment  
International Education



Ghana Education  
Service (GES)

# PENFIELD HIGH SCHOOL

# INTRODUCTION TO PROGRAMMING

---

D.O AMOAH-DANQUAH |  
PENFIELD HIGH SCHOOL

# OBJECTIVES

At the end of this unit you should be able to:

---

- ✓ define programming language
- ✓ state the classification of programming languages.
- ✓ identify the features of programming language.
- ✓ explain some terminologies associated with programming.
- ✓ explain program development life cycle
- ✓ explain algorithm concepts and flowchart concepts,

# DEFINITION OF PROGRAMMING LANGUAGE

---

A programming language is a formal constructed language designed to communicate instructions to a computer.

Programming languages can be used to create programs to control the behaviour of a computer or to express algorithms.

The description of a programming language is usually split into two components of **syntax (form)** and **semantics (meaning)**.

# The Development of Programming Language

---

The earliest computers were often programmed without the help of a programming language, by writing programs in absolute machine language.

The programs, in decimal or binary form, were read in from punched cards or magnetic tape, or toggled in on switches on the front panel of the computer. Absolute machine languages were later termed ***first-generation programming***

# The Development of Programming Language

---

The next was development of so-called **second-generation programming language (2GL)** or **assembly languages**, which were still closely tied to the instruction set architecture of the specific computer.

These served to make the program much more human-readable, and relieved the programmer of tedious and error-prone address calculations.

# The Development of Programming Language

---

The *first high-level programming language*, or third-generation programming language (3GL), were written in the **1950s**.

An early high-level programming language to be designed for a computer was *Plankalkül* developed for the German 23 by Konrad Zuse between **1943 and 1945**. However, it was not implemented until **1998 and 2000**

# The Development of Programming Language

---

In 1954, language FORTRAN was invented at IBM by John Backus; it was the first widely used high level general purpose programming language to have a functional implementation.

It is still popular language for high performance computing and is used for program that benchmark and rank the world's fastest supercomputer.



# Classification of Programming Language

---

Computer programming language can be classified into two major categories:

- Low-Level programming language**
- High-level programming language**

# □ LOW-LEVEL LANGUAGES

---

The languages which use only primitive operations of the computer are known as low-level language. these languages, programs are written by means of the memory and register's available on the computer.

In other words, programs written in one low level language which it architectural cannot be ported on any other machine dependent languages.

# Low-level languages Cont.

---

A typical low-level instruction consist of two parts

- ❑ **Operational part:** specifies operation to be performed by the computer, also known as **Op-code**
- ❑ **Address part:** specifies location of the data which operation is to be performed. Examples are Machine language and Assembly language.

# □ Machine languages

---

In machine language program, the computation is based on binary numbers. All the instructions including operations, registers, data and memory locations are given in the binary equivalent.

The machine directly understands this language by virtue of its circuitry design so these programs can directly be executable on the computer without any translations. This makes the program execution very fast. Machine languages are also known as **first-generation languages**.

# Advantages of Machine languages

---

- ✓ Machine language makes most efficient use of computer system resources like storage, registers etc.
- ✓ The instruction of a machine language program is directly executable so there is no need at translators.

It can be used to process the individual bits in a computer system with high execution speed due to direct processing of memory and register

# Disadvantages of Machine languages

---

- ✓ Machine languages are machine dependent; therefore, programs are not portable from one computer to the other.
- ✓ Programming in machine language usually results in poor programmer productivity.
- ✓ Machine language requires a high level of programming skills which increases programming training cost.
- ✓ Programs written in machine language are more error prone and difficult to debug Assembly Language

# Assembly language

---

Assembly languages are also known as second-generation languages. These languages substitutes alphabetic or numeric symbols for the binary codes of machines language.

It has two parts, **macro name** and **macro body** which contains the line of instruction.

A macro can be called at any point of the program by its name to use the instructions given in the macro repetitively.

# Assembly language Cont.

---

These languages require a translator known as "Assembler" for translating the program code written in assembly language to machine language.

Because computer can interpret only the machine code instruction, once the translation is completed the program can be executed.



# Advantages of Assembly languages

---

Assembly language provides optimal use of computer resources like registers and memory.

Assembly language is easier to use than machine language because there is no need to remember or calculate the binary equivalents for op-code and registers.

An Assembler is useful for detecting programming errors.

Assembly language encourages modular programming which provides the facility of reusable code, using macro.

# Disadvantages of Assembly languages

---

Assembly language programs are not directly executable due to the need of translation.

Assembly languages are machine dependent and therefore, not portable from one machine to another.

Programming in assembly language requires a high level of programming skills and knowledge of computer architecture of the particular machine

# □ High-level Languages

---

All high-level languages are procedure-oriented language and are intended to be machine independent. Programs are written in statements in English language,

These languages require translators (compilers and interpreters) for execution. The programs written in a high level language can be ported on any computer that is why they are known as machine independent.

# □ High-level Languages cont.

---

The early high-level languages come in third generation of languages, **COBOL, BASIC, API**, etc. These languages enable the programmer to write instruction using English words and familiar mathematical symbols which makes it easier than technical details of the computer

# □ High-level Languages

---

Procedures are the reusable code which can be called at any point of the program.

Each procedure is defined by a name and set of instructions accomplishing a particular task.

The procedure can be called by its name with the list of required parameters which should pass to the procedure.

# Advantages High-level Languages

---

- ✓ They are easier to learn than assembly language.
  - ✓ Less time is required to write programs.
    - ✓ They provide better documentation.
      - ✓ They are easier to maintain.
  - ✓ They have an extensive vocabulary.

# Disadvantages High-level Languages

---

- ✓ A long sequence statement is to be written for every program.
- ✓ Additional memory space is required for storing compiler or interpreter.
- ✓ Execution time is very high as the HLL programs are not directly executable.

# □ Features of Programming Language

---

## □ Date type

Is a classification of data which tells the compiler or interpreter how the programmer intends to use the data. Examples are real, integer, Boolean etc. A data type provides a set of values from which an expression may take its values.

The data type defines the operations that can be done on the data, the meaning of the data, and the way values of that can be stored.



# ❑ Features of Programming Language Cont.

---

## ❑ Variable or scalar

Is a storage location paired with an associated symbolic name (an identifier), which contains some known and unknown quantity of information.

## ❑ Constant

Is a value that cannot be change by the program during normal execution. The value used also remains constant. This is the opposite to variable, which is an identifier with a value that can be changed during normal execution

# ❑ Features of Programming Language Cont.

---

## ❑ Variable or scalar

Is a storage location paired with an associated symbolic name (an identifier), which contains some known and unknown quantity of information.

## ❑ Constant

Is a value that cannot be change by the program during normal execution. The value used also remains constant. This is the opposite to variable, which is an identifier with a value that can be changed during normal execution.

# □ Features of Programming Language Cont.

---

## □ Precedence

Operator precedence determines the order in which operators are evaluated. Operators with higher precedence are evaluated first.

A common example:  $3+4*5$  returns 23. The multiplication operator ('\*') has higher precedence than the addition operator ('+') and thus will be evaluated first.

**NB: Remember every programming language has its operator precedence.**

# □ Features of Programming Language Cont.

---

- **Input/output Statements** An input/output statement or I/O statement is a portion of a program that instructs a computer how to read and process information. It pertains to gathering information from an input device, or sending information to an output device.
- **Built-in functions** A function that is built into an application and can be accessed by end-users. For example, most Spreadsheet applications support a built-in SUM function that adds up all cells in a row or column.

# □ Features of Programming Language Cont.

---

□ **Sequential execution statement** This is when your instructions are executed in the same order that they appear in your program, without repeating or skipping any instructions from the sequence.

For example, this sequential execution:

```
Int a= 5;
```

```
Int b= 12;
```

```
Int c= a*a+b+7;
```

# ❑ Features of Programming Language Cont.

---

- ❑ **Conditional execution statement** is a feature of a programming language, which perform different computations or actions depending on whether a programmer-specified Boolean condition evaluates to true or false. Example is stated here: if-then-else statements.
  - ❑ **Loop** A loop is a sequence of instructions that is continually repeated until a certain condition is reached. Typically, certain process is done, such as getting an item of data and changing it, and then some condition is checked such as whether a counter has reached a prescribed number.

# ❑ TERMINOLOGIES ASSOCIATED WITH PROGRAMMING

---

- ❑ **Source code** is any collection of computer instructions, possibly with comments, written using a human-readable programming language, usually as plain text.
- ❑ **Boolean expression** is an expression in a programming language that produces a Boolean value when evaluated, that is one of True or False.
- ❑ **Class** is an extensible program-code template for creating objects, providing initial values for state (member variables) and implementations of behaviour (member functions or methods)

# ❑ TERMINOLOGIES ASSOCIATED WITH PROGRAMMING

---

❑ **Comment** is a programmer-readable explanation in the source code of a computer program. They are added with the purpose of making the source code easier for humans to understand and are generally ignored by compilers and interpreters.

❑ **Compiler** is a special program that processes statements written in a particular programming language and turns them into machine language or code that computer's processor uses.



# ❑ TERMINOLOGIES ASSOCIATED WITH PROGRAMMING

---

- ❑ **Interpreter** is a computer program that directly executes, that is, performs, instructions written in a programming or scripting language, without previously compiling them into a machine language program.
- ❑ **Debugging** is the routine process of locating and removing computer program bugs, error of abnormalities, which is methodically handled by software programmers via debugging tools.

# ❑ TERMINOLOGIES ASSOCIATED WITH PROGRAMMING

---

- ❑ **Bug** is an error, flaw, failure or fault in a computer program or system that causes it to produce an incorrect or unexpected result, or to behave in unintended ways.
- ❑ **Event procedure** The code that performs actions in response to events is written in event procedures. Each event procedure contains the statements that execute, when a particular event occurs on a particular object.

# ❑ TERMINOLOGIES ASSOCIATED WITH PROGRAMMING

---

- ❑ **Syntax** The syntax of a computer language is the set of rules that defines the combinations symbols that are considered to be a correctly structured document or fragment in that language
- ❑ **Run-time error** - An error that occurs during the execution of a program. In contrast, compile time errors occur while a program is being compiled. Runtime errors indicate bugs in the program or problems that the designer had anticipated but could do nothing about. For example running out of memory will often cause a runtime error.

# ❑ **TERMINOLOGIES ASSOCIATED WITH PROGRAMMING**

---

❑ **Coding** is a term used for both the statements written in a particular programming language, the source code, and a term for the source code after it has been processed by a compiler and made ready to run in the computer.

❑ **Object-oriented programming (OOP)** is a programming language model organized around objects rather than actions and data rather than logic. Ideally, a program has been viewed as a logical procedure that takes input data, processes it, and produces output data.

# Program Development Life Cycle

When we want to develop a program using any programming language, we follow a sequence of steps. These steps are called phases in program development. The program development life cycle is a set of steps or phases that are used to develop a program in any programming language



# Program Development Life Cycle

## 1. Analysis Specification

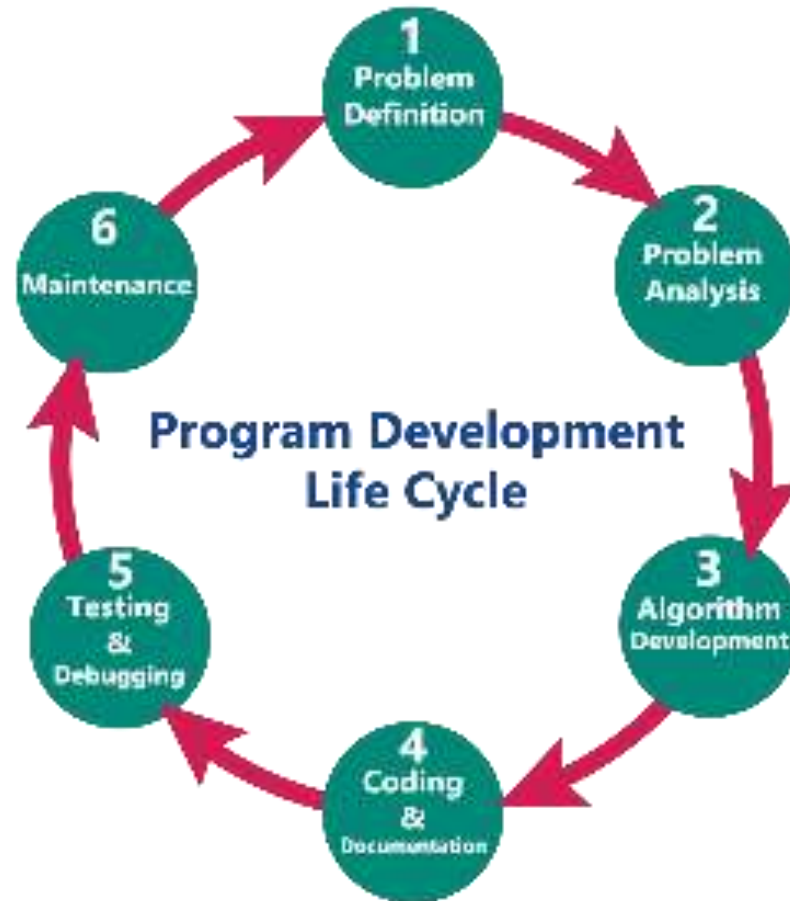
The program's objectives, outputs, inputs, and processing requirements are determined.

## 2. Program design

A solution is created using programming techniques such as top-down program design, Pseudocode, flowcharts, and logic structures.

## 3. Program code

The program is written or coded using a programming language.



## 4. Program debug and test

The program is debugging and tested by looking for syntax and logic errors..

## 5. Program Documentation

Documentation is an on-going process throughout the programming process. This phase focuses on formalizing the written description and processes used in the program.

## 6. Program maintenance

Completed programs are periodically reviewed to evaluate their accuracy, efficiency, standardization and ease of use. Changes are made to the program's code as needed.

# Program Specification

Program specification is also called program definition or program analysis. It requires programmer or the end user to specify five items:

- The program objectives,
- The desire output,
- The input data required,
- The processing requirements,
- and documentation.



## □ The program objectives

You solve all kinds of problems every day.

A problem might be decided how to commute to school. Thus every day you determine your objectives, the problem you are trying to solve, Programming is the same; you need to make a clear statement of the problem.






## □ The desire output

The best way is to specify outputs before inputs. That is, you need to list what you want to get out of the computer system. Then you should determine what will go into it. The best way to do this is to draw a picture.

## □ The input data



Once you know the output you want, you can determine the input data and the source of this data. For example, for a time and billing report, you can specify that one source of data to be processed should be time cards.



## □ The processing requirements

Here you define the processing tasks that must happen for input data to be processed into output. For advantage, one of the tasks for the program will be to add the hours worked for different jobs for different clients.

## ❑ **Documentation.**

You should record program objectives, desired outputs, needed inputs and required processing. This lead to the next step, program design

# Program Design

During this phase, the system is designed to satisfy the functional requirements identified in the previous phase. Here you plan a solution, preferably using structured programming techniques.

These techniques consist of

- Top-down programming
- Pseudocode,
- Flowchart and,
- logic structure.



## ❑ Top-down programming

First determine the outputs and inputs for the program. Then use top-down program design to identify the program's processing steps. Such steps are called **program modules**. Each module is made up of logically related program statements.

## □ Pseudocode

Pseudocode is an outline of the logic of the program you will write. It is like doing a summary of the program before it is written. Remember this expresses the logic of what you want the program to do. It is a representation of code used to demonstrate the implementation of an algorithm without actually doing so.

## □ Flowchart

A flowchart is a type of diagram that represents an algorithm, workflow or process.

These graphically present the detailed sequence of steps needed to solve a programming problem.



## ❑ Logic Structures

Logical structure refers to the way information in a document is organized; it defines the hierarchy of information and the relation between different parts of the document. Logical structure indicates how a document is built, as opposed to what a document contains

## ❑ Logic Structures

All Logic structures follow these three structures — **sequence, selection, and looping.**

**The Sequence Structure-**In the sequence structure, one program statement follows another. They logically follow each other. There is no question of "yes" or "no" of a decision suggesting other sequence.

## ❑ Logic Structures

**The Selection Structure** - The selection structure occurs when a decision must be made. The outcome of the decision determines which of two paths to follow. This structure is also as IF-THEN-ELSE structure, because that is how you can formulate the decision

## ❑ Logic Structures

**The loop structure** - The loop structure describes a process that may be repeated as long as a certain condition remains true. The structure is called a "loop" or "iteration" because the program loops around (iterates or repeats) again and again. The last thing to do before leaving the program design step is to document the logic of the design. This report typically includes Pseudocode, flowcharts, and logic structure.

# Program Code

Writing the program is called coding. Here you use the logic you developed in the program design step to actually write the program. That is, you write out using pencil and paper or typing on a computer



# Program Test

Program testing is the process of executing a program with the intent of finding errors. A good test is one that has a high probability of finding an error. Programming errors are of two types:  
**Syntax errors and Logic errors**



The arrangement of words and phrases to create well-formed sentences in a language.  
the structure of statements in a computer language.

## Syntax Errors

A syntax error is a violation of the rules of the programming language. For example in C++ each statement must end with a semicolon (:). If the semicolon is omitted, the program will not run due to syntax error.

Syntax errors are **mistakes in using the language**. Examples of syntax errors are missing a comma or a quotation mark, or misspelling a word

a system of reasoning that aims to draw valid conclusions based on given information.

## Logic Errors

Logic errors occur when there is a fault in the logic or structure of the problem. Logic errors do not usually cause a program to crash.

A logic error occurs when the programmer uses an incorrect calculation or leaves out a programming procedure.

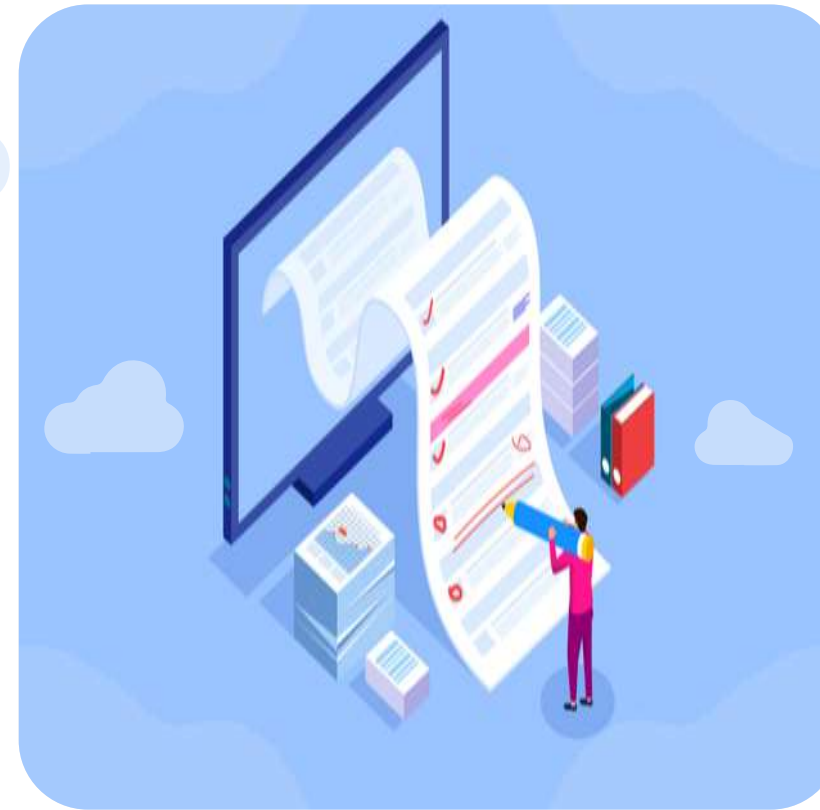
***For example, a payroll program that did not compute overtime hours would have a logic error***



# Program Documentation

Documentation consists of written descriptions and procedures about a program and how to use it. It is not something done just at the end of the programming process. Program documentation runs through all the programming steps.

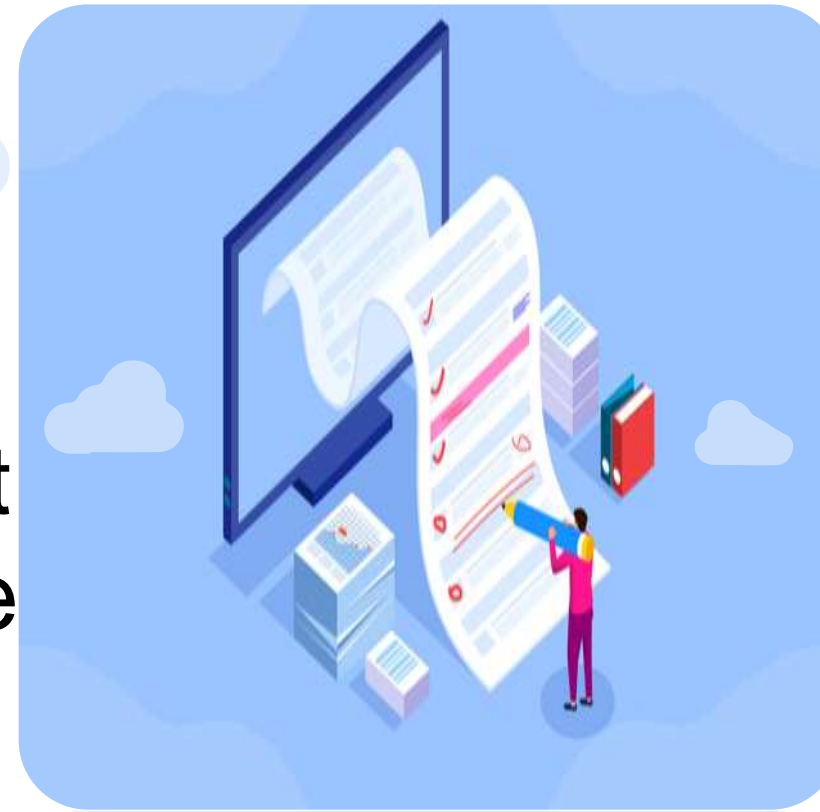
*It is the information, available in writing about a program*



# Program Documentation

This documentation is typically within the program itself and in printed documents. Documentation is important for people who may be involved with the program in the future. These people may include the following:

- User**
- Operators**
- Programmers**



## □ User

Users need to know how to use the software. Some organisations may offer training courses to guide users through the program. However, other organisations may users to learn a package just from the written documentation. Two examples of the documentation are the manuals that accompany the software and the help option within most computer applications.

## ❑ Operators

Operators documentation must be provided for computer operators. for instance, If the program sends them an error messages, they need to know what to do about them.

## ❑ The Programmers

As time passes, even the creator of the original program may not remember much about it. Other programmers wishing to update and modify it, that is, perform program maintenance may find themselves frustrated without adequate documentation therefore the need for this kind of documentation. This kind of documentation should include **text and program flowcharts, program listings, and sample outputs.**

# Program Maintenance

**Program maintenance** is the process of changing, modifying, and updating software to keep up with customer needs.

The purpose of program maintenance is to ensure that current programs are operating error free, efficiently, and effectively. Activities in this area fall into two Categories:  
**operations and changing needs.**



# Operations Activities

Operations activities deals with locating and correcting operational errors, making programs easier to use, and standardizing software using structured programming techniques.

## Changing Needs

The category of changing needs is unavoidable. All organisations change over time, and their programs must change with them. Programs need to be adjusted for a variety of reasons, including new tax laws, new information needs, and new company policies.

Significant revisions may require that the entire programming process begin again with program specification.








# Algorithms Concept



# Algorithms Concept



Algorithm is a step-by-step procedure to solve a given problem. They are a set of instructions (method) which if faithfully followed will produce a solution to a given problem.



**It is a procedure or formula used for solving a problem.**



**Example 1** - Design an algorithm for adding the test scores as given: **26, 49, 98, 87, 62, 75**

## The Algorithm

1. Start
2. Sum=0
3. Get the first test score
4. Add first test score to sum
5. Get the second test score
6. Add to sum
7. Get the third test score
8. Add to sum
9. Get the Forth test score
10. Add to sum
11. Get the fifth test score
12. Add to sum
13. Get the first test score
14. Add to sum
15. Output the sum
16. Stop

# Techniques for Representing Algorithms

We can express an algorithm many ways, including natural language, flow charts, pseudocode, and of course, actual programming languages.

- Pseudo code
- Flowcharts
- Actual code

# Pseudo code

**Pseudo code** is a term which is often used in programming and algorithm based fields. Pseudo code, as the name suggests, is **a false code or a representation of code which** can be understood by even a layman.

It is a readable description of what a computer program or algorithm must do, expressed in a formally-styled natural language rather than in a programming language.

# Algorithm Building Blocks

All problems can be solved by employing any one of the following building blocks or their combinations.

## □ Sequences

A sequence of instructions that are executed in the precise order they are written in:

- statement block 1
- statement block 2
- statement block 3

# Algorithm Building Blocks

Example : Suppose you are required to design an algorithm for finding the average of six numbers and the sum of the numbers is given.

The pseudo code will be as follows

Start

Get the sum  $\text{Average} = \text{sum} / 6$

Output the average

Stop

# Algorithm Building Blocks

Example 2: A pseudo-code required to input three numbers from the keyboard and output the result.

Start

Use variables: sum, number 1, number2, number3 of type integer

Accept number1, number2, number3

Sum=number1 + number2+number3

Printsum

Stop



# Algorithm Building Blocks

Example 2: A pseudo-code required to input three numbers from the keyboard and output the result.

Start

Use variables: sum, number 1, number2, number3 of type integer

Accept number1, number2, number3

Sum=number1 + number2+number3

Printsum

Stop

**Example 4** – A pseudo-code describes an algorithm which will accept two numbers from the keyboard and calculate the sum and product displaying the answer on the monitor screen.

Start

Use variables sum, product, number 1, number2 of type real

display "Input two numbers"

accept number 1, number2

sum = number1 + number2

print "The sum is", sum

Product = number1\*number2

print "The Product is", product

Stop

# Algorithm Building Blocks



## □ Conditionals

Conditionals are a fundamental programming element that allows a computer to make decisions depending on specific criteria.

Select between alternate courses of action depending upon the evaluation of a condition



# Algorithm Building Blocks

## □ Conditionals

If (condition=true)

statement block 1

Else

statement block 2

End if

**Example 5** – The following shows how the selection control structure is used in a program where a teacher chooses the options for multiplying the numbers or adding them or subtracting.

Start

Use variables: choice, of the type character,  
ans, number 1, number2, of type integer  
display "choose one of the following"  
display "m for multiply" display "a for add"  
display "s for subtract"  
accept choice  
display "input two numbers you want to use"  
accept number1, number2

```
if choice = m
    then ans=number1*number2
if choice = a
    then ans number1+number2
if choice=s
then ansnumber1-number2
display ans
Stop
```

# Class Work

The program to input an examination mark and test it for the award of a grade. The mark is a whole number between 1 and 100.

Grades are awarded according to the following criteria

$\geq 80$ -Distinction

$\geq 60$ -Merit

$\geq 40$ -Pass

$< 40$ -fail