جامعة بوليتكنك فلسطين

CHAPTER SEVEN

PROBLEM SOLVING TECHNIQUES

Prepared by:
Eng. Yousef Salah

This material developed under the objectives of FESTEM project funded by the EU.
https://festem.ps/

Co-funded by the Erasmus+ Programme of the European Union

CHAPTER SEVEN
PROBLEM SOLVING TECHNIQUES

PREFACE:

This chapter begins with a short introduction about programming languages. It presents the classification of languages by generations, and points out the nature of each category. This is followed by the software development life cycle while giving thorough description about each phase in the software creation process. The chapter also teaches students how to think algorithmically and solve problems efficiently. Next, several computational problems are solved based on the appropriate control structures and logic needed to achieve the solution.

INTENDED LEARNING OUTCOMES:

After completing this chapter students will be able to:

1) Identify the classifications of programming languages.
2) Discuss programming language translators.
3) Recognize the main phases of the software development life cycle.
4) Explain what a computer algorithm is, and how it can be represented.
5) Describe program logic and control structures used to formalize the solution of computational problems.

FURTHER READING:

1) Computer Science An Overview, 13th Eition, J Glenn Brookshear; Dennis Brylow, Pearson, @2019.
2) Introduction to the Design and Analysis of Algorithms, Levitin, A. V., 2nd ed. Boston, Addison-Wesley, 2007.
3) Computer Algorithms: Introduction to Design and Analysis, Baase S., 3rd ed. Boston, Addison-Wesley, 2000.

INTRODUCTION

- Computer hardware is nothing without being instructed to do tasks.
- Therefore, our goal is to give commands to the computer in order to perform specific tasks in certain order.
- **Computer Program**: a list of instructions (i.e. commands) that directs the computer to perform a specific task. These instructions are written in a special language called ***programming language***.
- **Programmer** (or developer): a person who writes and modifies computer programs.
- **Programming Language**: set of words, abbreviations, and symbols used to create computer programs.
- Any programming language has a grammar (set of rules) that governs writing valid statements in this language.
    - These rules are also called the **Syntax** of the language.

TYPES OF PROGRAMMING LANGUAGES

- The language of a computer is called the ***machine language***, where the commands are represented as sequences (i.e. patterns) of ZEROs and ONEs called bits.

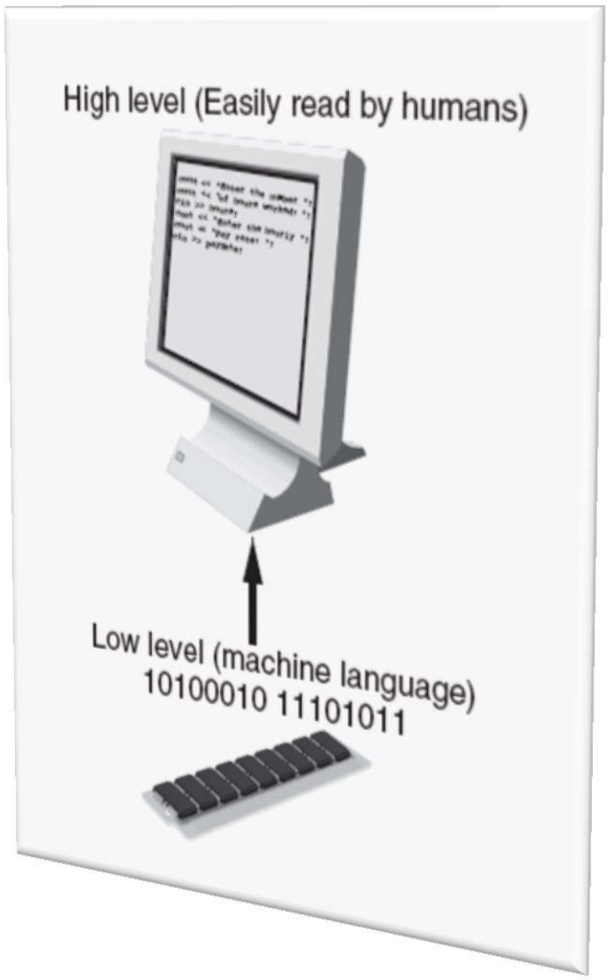
  Example of a machine code: **1011010000000101**

- However, not all machines (i.e. computers) have a global machine language.
- Each computer's designer chooses his own set of binary codes to perform the basic operations required from the computer.
- For example, IBM computer manufacturer may represent the Addition operation as **10101010**, whereas Apple computer manufacturer may choose **11110000** for the Addition operation.

- There are two types of programming languages: ***low-level*** and ***high-level*** programming languages.

- **Low-level Programming languages:**

  Their Syntax is close to the hardware. Low-level programming languages include:
  - Machine Language (First Generation)
  - Assembly Language (Second Generation)


High level (Easily read by humans)
Low level (machine language)
10100010 11101011

- **High-level Programming languages:**

  Their syntax is more like human language, and use English-like words that are understandable by people. High-level Programming languages include:
  - Third Generation Languages (3GLs).
  - Forth Generation Languages (4GLs).
  - Visual Programming Languages.
  - Web Development Languages.

PROGRAMMING LANGUAGES CLASSIFICATIONS

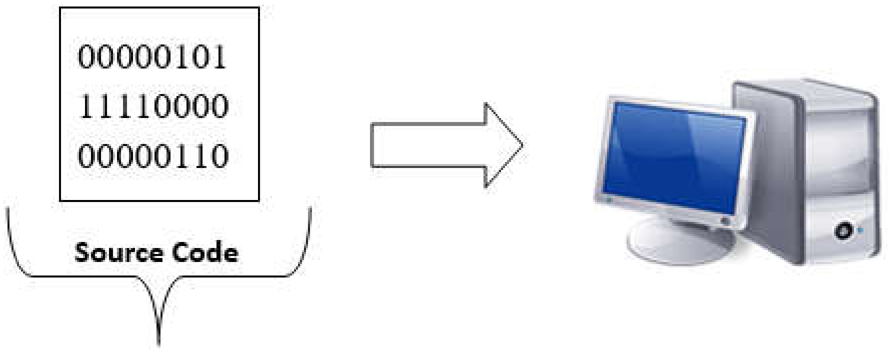
1 Machine Language (First Generation)

- Programs written in a machine language are patterns (or sequences) of 0s and 1s.
- For simplicity, assume we have a computer machine that has the following instruction set:

Binary Pattern	Operation
11110000	Add two numbers
11001100	Subtract two numbers

The following machine program (***code***) instructs the computer to add the two numbers 5 and 6.

```
00000101
11110000
00000110
```



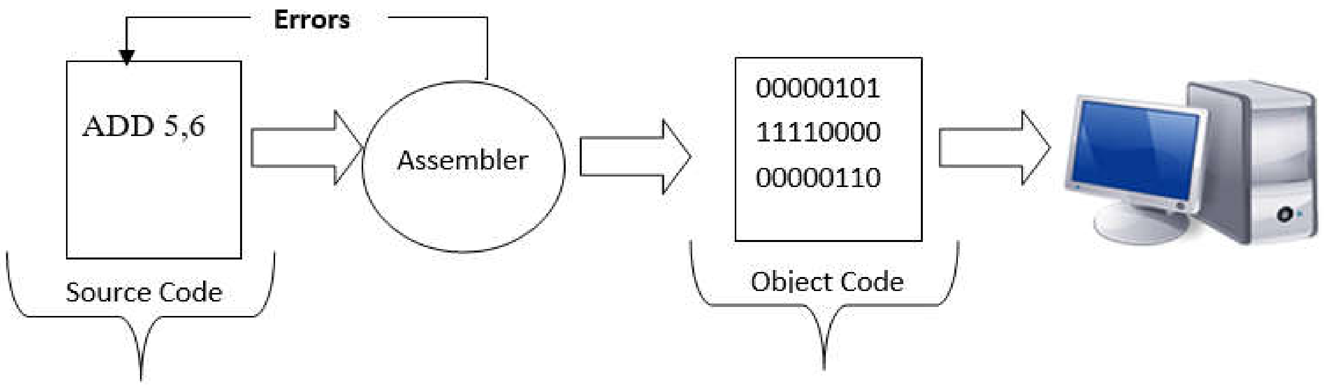
- Instructions (or statements) written in a programming language are called ***code*** or ***source code***.
- The commands are executed directly by the computer processor after loading into memory.
- The disadvantages of writing computer programs in machine languages are:
    - The programming is tedious, error prone and time-consuming.
    - Debugging (i.e. finding errors) is difficult.

2 Assembly Language (Second Generation)

- Instructions in a computer program written in an assembly language are meaningful and readable names and symbols (called mnemonics) that correspond to 0s and 1s.
- For example, for the computer machine discussed later, the following mnemonics are used:

Binary Pattern	Mnemonic	Meaning
11110000	ADD	Add two numbers
11001100	SUB	Subtract two numbers

The following assembly code instructs the computer to add the two numbers 5 and 6.



- Programs written in assembly language cannot be delivered directly to the computer.
- A system software called **Assembler** is a software that translates a program written in assembly language into its equivalent code in machine language called ***object code***.

- The disadvantages of writing computer programs in assembly languages are:
  - Debugging is still difficult.
  - The translation process requires extra time.

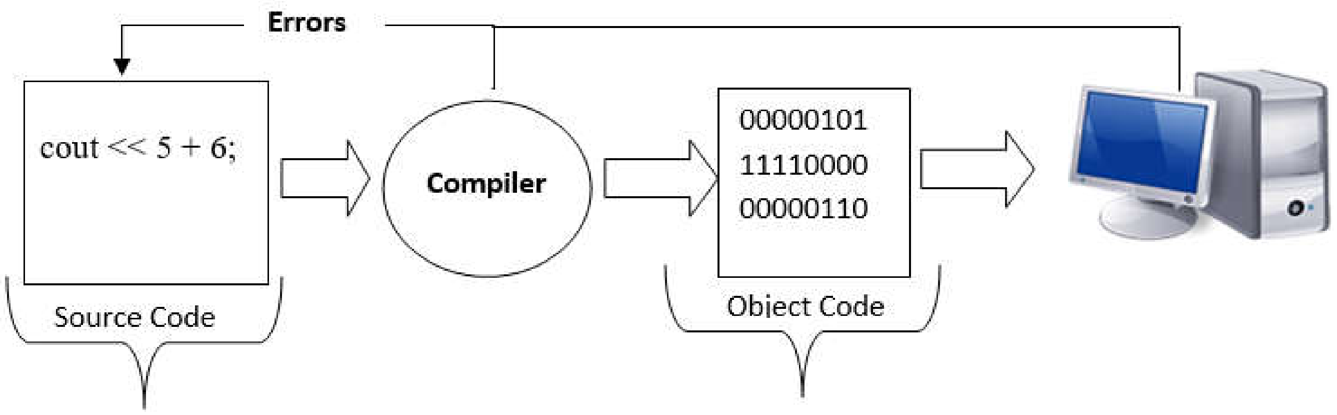
3 Third Generation Languages (3GLs)

- Third Generation Programming languages are high-level languages since the commands and instructions used are human readable, and they use English-like keywords.
- For example, keywords like: ***include***, ***return***, ***while*** are used in C++ (a high level programming language).
- 3GLs also use symbols for arithmetic and logical operations like: **+, *, > , ?, !, …**
- Examples of Third-Generation programming languages: C, C++, Pascal, FORTRAN, COBOL, Java, C#, Python ...
- A computer program written in a high level programming language should be translated into machine language before being delivered to the computer.
- There are two types of translators:
    - Compiler
    - Interpreter

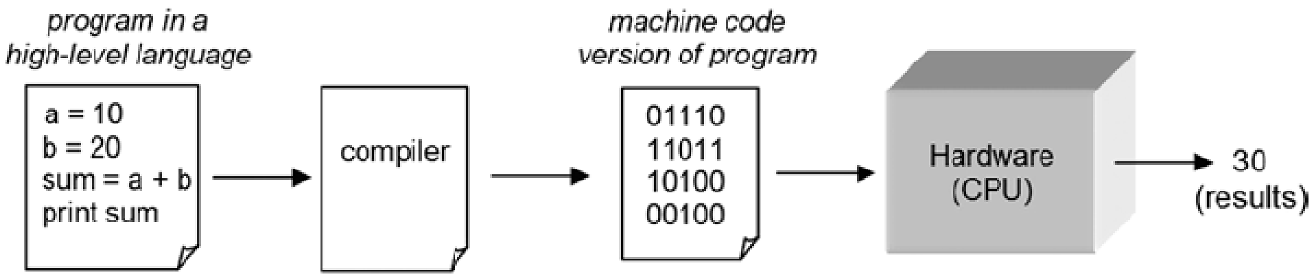
- Some programming languages have compiler, others have interpreters, and some have both.

* COMPILERS:

- A Compiler: a program that translates instructions written in a high-level language into the equivalent machine language (i.e. Object Code).



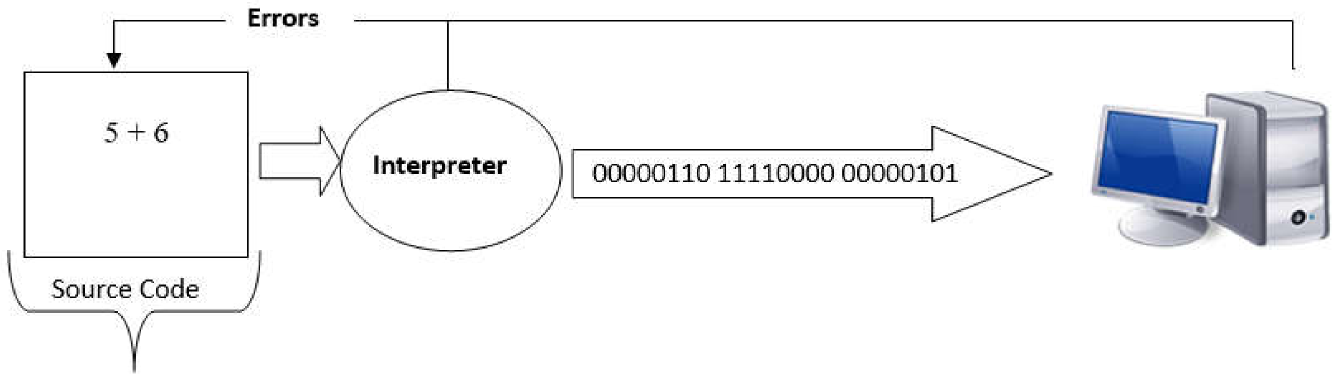
- A complier translates the entire source code into object code before the execution. So, the translation is done one time, and the object code is executed many times.



- Advantages of Compilers:
    - It is difficult to reconstruct the source code from the compiled object code.
- Disadvantages of Compilers:
    - The translation process requires extra time.
    - Finding the errors is in the source code is difficult.

* INTERPRETERS:

- An Interpreter: a program that reads the source code line by line, translates it, and then delivers it to the computer to execute it.

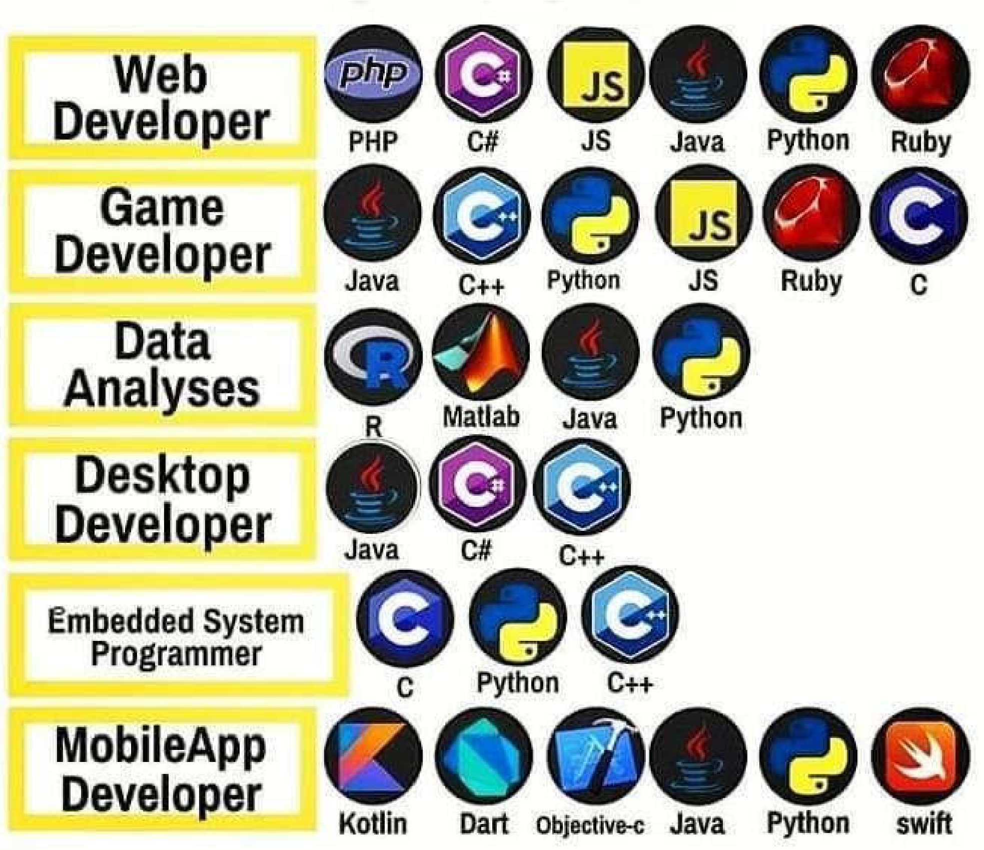


- Advantages of Interpreters: Finding errors becomes easy.
- Disadvantages of Interpreters: It needs more time to execute the program since every execution of a program requires translation.

Web Development Languages

- There are special programming languages that are used to construct web pages.
- Examples of Web Page languages include:
  - HTML: Hypertext Markup Language
  - PHP: Personal Home Page Language
  - ASP: Active Server Pages Language
  - JSP: Java Server Pages Language

Programming Languages Based on Specialty:



Activity:

- What is the meaning of open source software?

- List open source OS examples for Desktop and Mobile devices.

Activity:

Visit https://www.w3schools.com/ website, and have a look to different common programming languages.

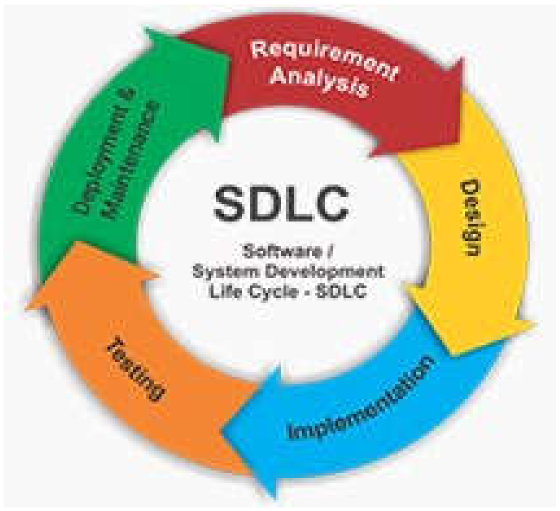
SOFTWARE DEVELOPMENT LIFE CYCLE (SDLC)

- The creation of a computer program is a developmental process.
- A ***computer program*** solves a problem in a computerized way.
- A problem here means *computational problem*.

- ***Programming*** is a process of problem solving, and a programmer is a problem solver person who finds the solution of a problem using a problem solving technique.
- ***Computer Programming*** (also called ***software development***) is the process of planning, designing, creation, and testing a computer program.
- ***Computer Programmer*** (or Software Developer) is a person who creates computer software using a programming language.

- Creating a computer program resembles building a house. You need to perform some planning, designing, and implementation in order to build a house.
- The program development cycle is a series of steps the programmers follow in order to build a computer program.

- The software development life cycle (SDLC) consists of the following main phases:
  - Problem Analysis.
  - Program Design.
  - Program Coding (or Implementation).
  - Program Testing.
  - Program Operation and Maintenance.



PHASE 1: PROBLEM ANALYSIS

- Analyzing a problem (also called problem definition) involves studying the requirements needed from the problem. Sometimes this step is called ***Requirement Analysis***.

- The programmer (or system analyst) tries to understand the user's requirements through identifying the inputs, processing, and outputs of the problem.

Software Engineering:

Software Engineering is defined as a process of analyzing user requirements and then designing, building, and testing software application which will satisfy these requirements.

- The system analyst can then document these requirements in a neat and organized format. This ***documentation*** is the road map that will lead the next steps.
- The Inputs, Processing, and Outputs also called IPO are defined clearly in this phase. An IPO chart may appear in the documentation to model the inputs, outputs, and processing:
  - **Inputs:** the information, ideas, and resources.
  - **Processing:** the operations, events, and actions taken upon/using input.
  - **Outputs:** results of the processing.

Example:

Problem Statement: Determine the <u>total cost of apples</u> given the <u>number of kilos of apples</u> purchased, and the <u>cost per kilo</u> of apples.

- ***Problem Input:***
  - Quantity of apples purchased (in kilos)
  - Cost per kilo of apples (in dinars per kilo)
- ***Processing:***

  $\text{Total cost} = \text{Number of kilos of apples} \times \text{Cost per kilo}$
- ***Problem Output:***
  - Total cost of apples (in dinars)

IPO Chart:

Input	Processing	Output
number of kilos of apples cost apples per kilo	Total cost = Number of kilos of apples × Cost per kilo	Total cost of apples

Exercise:

Problem Statement: Suppose that a store makes a discount on its items. What is the IPO chart that shows the amount of money that a customer has to pay when purchasing an item, where the following data are given: the price of the item before discount and the discount rate.

IPO Chart:

Input	Processing	Output

PHASE 2: PROGRAM DESIGN

- Designing a problem involves describing the solution of the problem. The solution lists the ***main steps*** required to be carried out in order to reach output(s) from input(s).
- The designing process involves finding an ***algorithm*** that satisfies the requirements.

- An ***algorithm*** is a step-by-step procedure that describes the solution of a problem to perform a specific task.

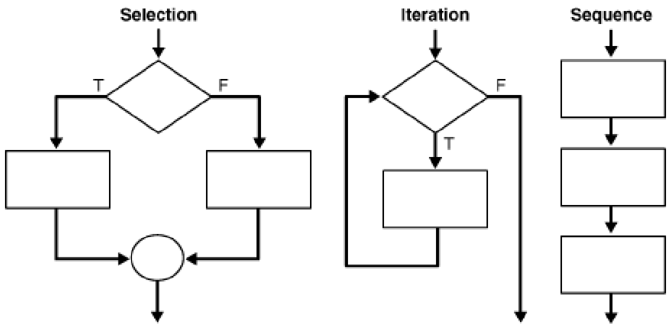
Algorithm (الخوارزمية):

The word "Algorithm" derived from the name of the Arab Muslim mathematician scientist named: "Muhammad Al-Khwarizmi.", who is the founder of "Al-jabr", in which he introduced the fundamental algebraic methods and techniques for solving equations.

- (Other Definition) An algorithm is a sequence of a finite number of steps arranged in a specific logical order which, when executed, produces the solution for a problem.

- An algorithm is an abstract and general description. This means that writing an algorithm means listing the main steps that solve the problem **without** identifying the **details** of implementation (i.e. programming language, compiler, operating system, …).

- The solution algorithm is also called the ***program logic***, since it shows the logical order of execution.
- The order in which statements (i.e. instructions) are executed is called program flow control (or flow of control).
- There are three ways (logic structures or control structures) that illustrate this order:
    - Sequence control structure
    - Selection control structure
    - Repetition control structure



- There are many ways of representing algorithms, some are textual, others are graphical.
- We'll study two design tools (methodologies) that are used to express an algorithm:
    - Pseudocode
    - Flowcharts

* PSEUDOCODE:

- English-like statements that describe the steps that solve the problem.
- Pseudocode uses keywords similar to those existed in high level languages.

Example:

Write an algorithm to determine a student's final grade and indicate whether it is passing or failing. The final grade is calculated as the average of four marks.

Solution described in Pseudocode:

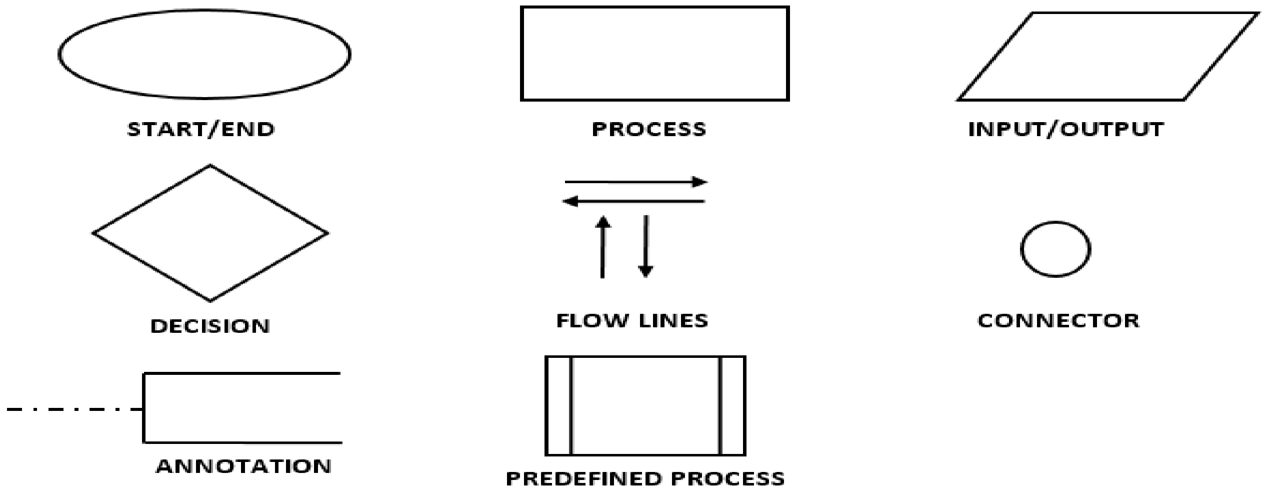
```
▪ Input a set of 4 marks
▪ Calculate their average by summing and dividing by 4
▪ if average is below 50
        Print "FAIL"
  else
        Print "PASS"
```

- No strict rules govern writing the Pseudocode. For instance, we can write the word **"otherwise"** instead the word **"else"** in the code above.
- You can write the algorithm in more detailed fashion like:

```
▪ Step 1:    READ: M1,M2,M3,M4
▪ Step 2:    grade ← (M1+M2+M3+M4)/4
▪ Step 3:    IF (grade < 50) THEN
                 PRINT "FAIL"
             ELSE
                 PRINT "PASS"
```

* FLOWCHART:

- A flowchart is a ***graphical representation*** of the sequence of operations in an algorithm.
- A flowchart shows the ***logic*** of an algorithm. This logic is the order in which operations are performed to reach the goal.
- The flowchart of an algorithm graphically shows the three logical order of the execution: Sequence, Selection, and Repetition.
- The symbols (i.e. constructs) of a flowchart are graphical symbols. These constructs collectively are called the ***notation*** of the flowchart methodology. Here is the description of these symbols:



PHASE 3: PROGRAM CODING (OR IMPLEMENTATION)

- ***Program coding*** (or ***development***) is the process of converting the designed solution algorithm into program instructions written in a particular programming language.
- Mostly, the programmer chooses a programming language to write the instructions, and saves these instructions into a file. This version of a file is what we called the ***source code***.
- Actually, the programmer needs a text editor to type the instructions. Then a language translator is used to perform the source code translation into machine language version called the ***object code***.
- We had already discussed the three common types of language translators: Assemblers, Compilers, and Interpreters.

- Nowadays, most of the programming languages use a single interface for all the programs and development tools required in the development process (like text editor, translator, debugger and other utilities). These programs are referred to as an **Integrated Development Environment (IDE)**, or **Software Development Kit (SDK)**.

Well-known IDEs are:
- Visual Studio (from Microsoft®).
- Android Studio.
- Visual Studio Code.
- Eclipse.

PHASE 4: PROGRAM TESTING

- After creating the source code, the programmer tests the following:
  - Make sure that the ***application*** works correctly, and it is free of errors (or bugs).
  - Make sure that all the user requirements are satisfied.
- An error can be one of the following two types:
  - **Syntax Error:** occurs when the code violates the syntax of the programming language. The translator issues an error message so that the programmer can correct it.
    - For example: entering "prnt" instead of "print".

  - **Logic Error (or Semantic Error):** occurs during the run-time of the program due to a mistake in the solution algorithm, and may cause the program to terminate, hang, or behave wrongly.
    - For Example: finding the average of three numbers n1, n2 and n3 using the equation: (n1+n2+n3)/2.0.

- **Debugging:** The process of locating and correcting syntax and logic errors in a program.

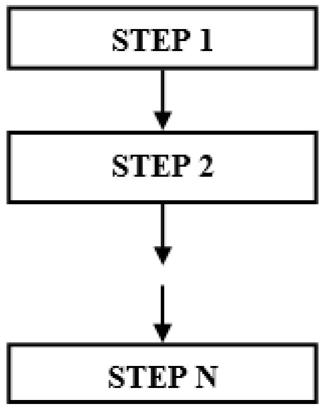
PHASE 5: PROGRAM OPERATION AND MAINTENANCE

- After testing the application software, it is released to users for operation.
- ***Program Operation*** involves installing the software, and allowing the users to use it.
- Program Maintenance involves:
  - Fixing any emerging errors or faults that did not appear during the testing process.
  - Adding new services and functionalities to the released software.

PROGRAM LOGICAL ORDER (CONTROL STRUCTURES)

* SEQUENCE CONTROL STRUCTURE:

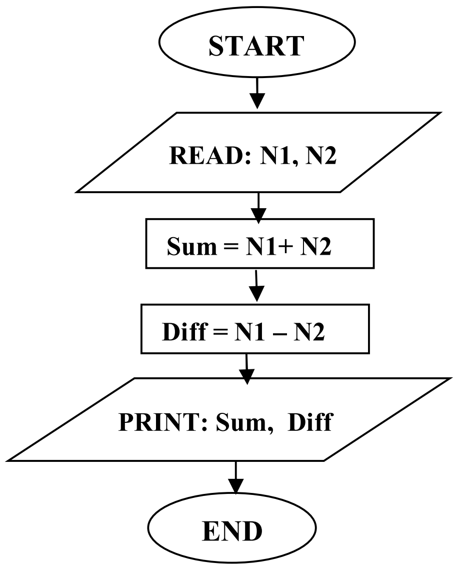
- The steps of the algorithm are carried out one after another.



Example:

Use Flowchart and Pseudocode techniques to describe the algorithm that reads two numbers, calculates and prints their sum and difference.

FLOWCHART



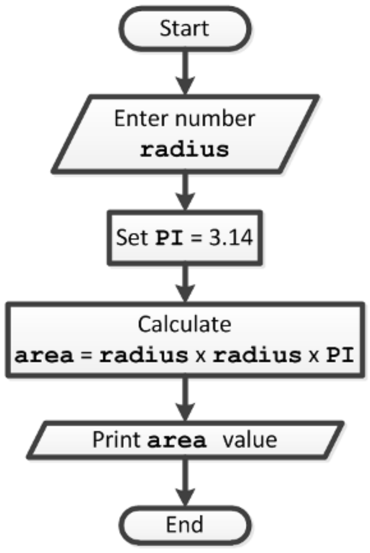
PSEUDOCODE

```
BEGIN
    ▪ INPUT:  N1,N2
    ▪ Sum ← N1+ N2
    ▪ Diff ← N1- N2
    ▪ PRINT: Sum, Diff
END
```

Example:

The following Flowchart and Pseudocode describe the algorithm that reads a radius of a circle, then finds its area ($Area = \pi \times radius^2$).

FLOWCHART



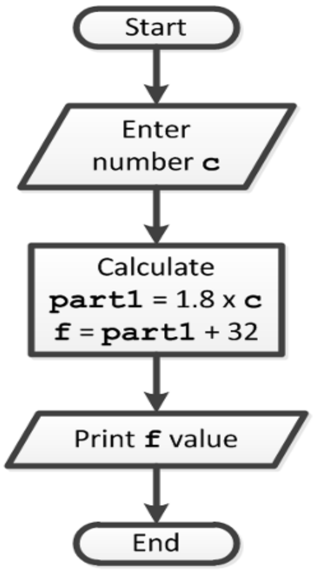
PSEUDOCODE

```
START
    Enter number radius
    Set PI = 3.14
    Calculate area = radius × radius × PI
    Print area value
END
```

Example:

The following Flowchart and Pseudocode describe the algorithm that converts the Celsius temperature to Fahrenheit temperature, using the equation: F = 1.8C + 32

FLOWCHART



PSEUDOCODE

```
START
        Enter number c
        Calculate part1 = 1.8 × c
        Calculate f = part1 + 32
        Print f value
END
```

Activity:

Use both flowchart and Pseudocode to describe the algorithm that accepts three marks for a student, and then displays their sum and average.

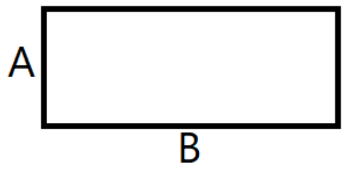
Activity:

Suppose that a store makes discount on its items. Design an algorithm (using Flowchart and Pseudocode) that calculates the amount of money that a customer has to pay when purchasing an item, adding to it the required tax, where the following data are given: the price of the item before discount, the discount rate, and the tax rate. (Assume the discount is calculated after considering the TAX.)

Exercises:

Use flowchart and Pseudocode strategies to design an algorithm that solves each of the following problems:

1) Finding the area and perimeter of a rectangle, assuming the length and width (A and B) are given as an inputs.

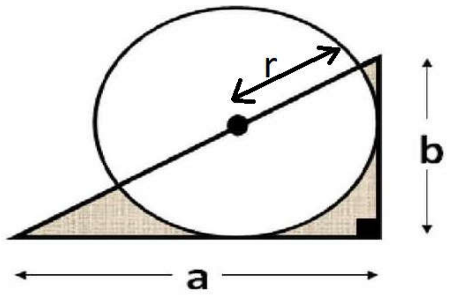


2) Calculating the percentage of increment in the salary of an employee, when inputting the salary before increment, and the value of increment.

3) Reading two numbers N1 and N2 and swapping their values.

4) Reading a number that represents an elapsed time in seconds. The algorithm displays how many hours, minutes and seconds this number contains. **For example,** if the number of input seconds is 8500, then the output is: 2 Hours, 21 Minutes, 40 Seconds.

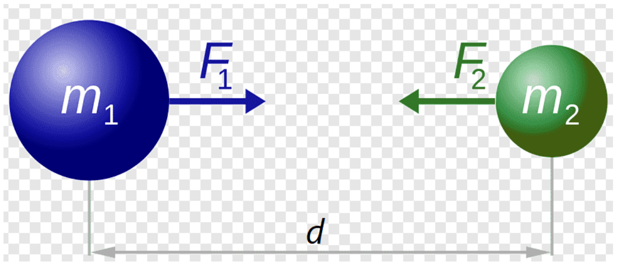
5) Given a, b and r as inputs, write the algorithm that will calculate the area of the shaded parts in the figure.



6) Reading a *three-digit* number and printing each digit separately from right to left.
If the number is 479, the algorithm will print: **9 7 4**

7) Newton's law states that the force F, between two bodies of masses M1 and M2 is given by:

$$F = k\left(\frac{M_1 M_2}{d^2}\right)$$



in which k is the gravitational constant and d is the distance between the bodies. The value of k is 6.67×10^{-8} dyn.cm²/g². Design an algorithm that prompts the user to input the masses of the bodies, and the distance between the bodies. The algorithm calculates and outputs the force between the bodies.

* SELECTION CONTROL STRUCTURE:

- A ***Selection*** (or ***Branching***) control structure shows which steps should be performed, and which should be ignored based on the evaluation of certain *condition*.
- A *condition* is a question that has an answer either **YES** (i.e. TRUE) or **NO** (i.e. FALSE).
- The condition (or ***logical condition***) is created by using logical or arithmetic operations like: $+, -, \times, \div, >, <, =, \neq, \leq, \geq$ …
- Examples of conditions:
  - $month = 2$?
  - $average \geq 90.0$?
  - $(1000 > salary)$ and $(rank =$ 'A' $)$?

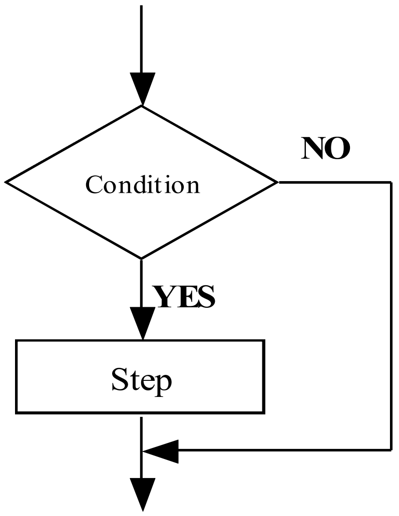
- There are three ways of using Selection Control Structure:
  - *if-then control structure*
  - *if-then-else control structure*
  - *case selection control structure*

if-then control structure:

- If the value of the condition is TRUE the following step(s) will be executed, otherwise (if it's FALSE) the step(s) will be ignored.

- Pseudocode of the if-then structure:

```
If( Condition ) Then
        Statement
```



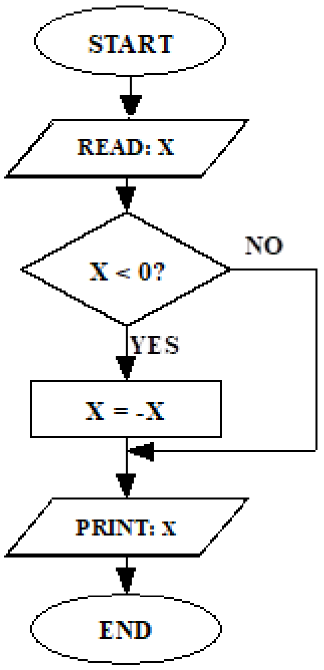
The *if-then* Control Structure

Example:

Design an algorithm that reads a number and displays its absolute value.

$$|X| = \begin{cases} +X & X \geq 0 \\ -X & X < 0 \end{cases}$$

FLOWCHART



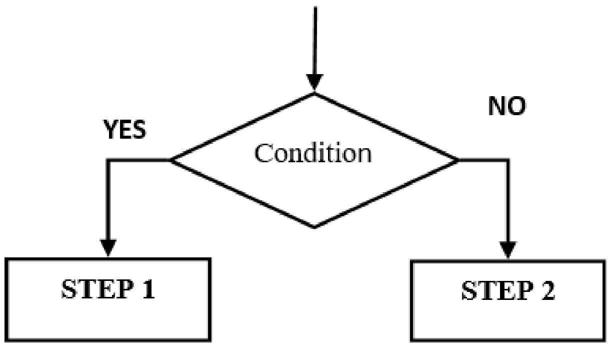
PSEUDOCODE

```
START
        READ:  X
        IF X < 0 THEN
            X = -X
        PRINT: X
END
```

if-then-else control structure:

- If the condition results YES the algorithm performs a specific action, otherwise if it results NO the algorithm performs a different action.
- Pseudocode of the if-then-else structure:

```
IF( Condition ) THEN
        Statement1
ELSE
        Statement2
```

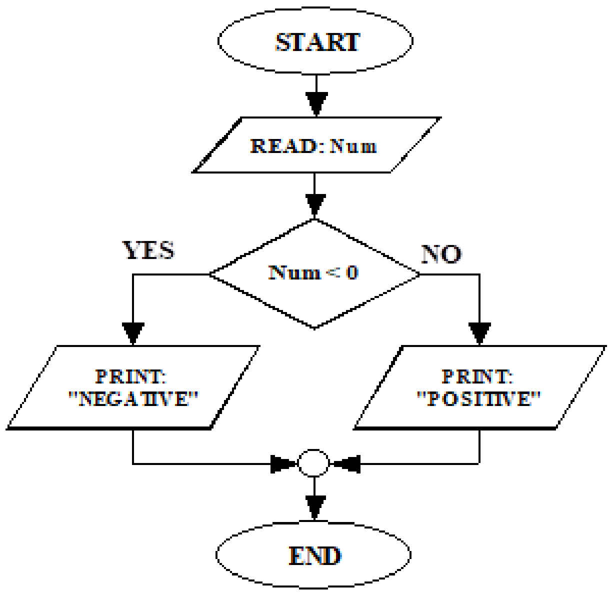


The *if-then-else* Control Structure

Example:

Design an algorithm an algorithm that reads a number and displays whether it's POSITIVE or NEGATIVE.

FLOWCHART



PSEUDOCODE

```
BEGIN
    ▪ READ: X
    ▪ IF ( X < 0 ) THEN
          PRINT "NEGATIVE"
      ELSE
          PRINT "POSITIVE"
END
```

Example:

Design an algorithm that reads a number and displays whether it is ODD or EVEN.

Example:

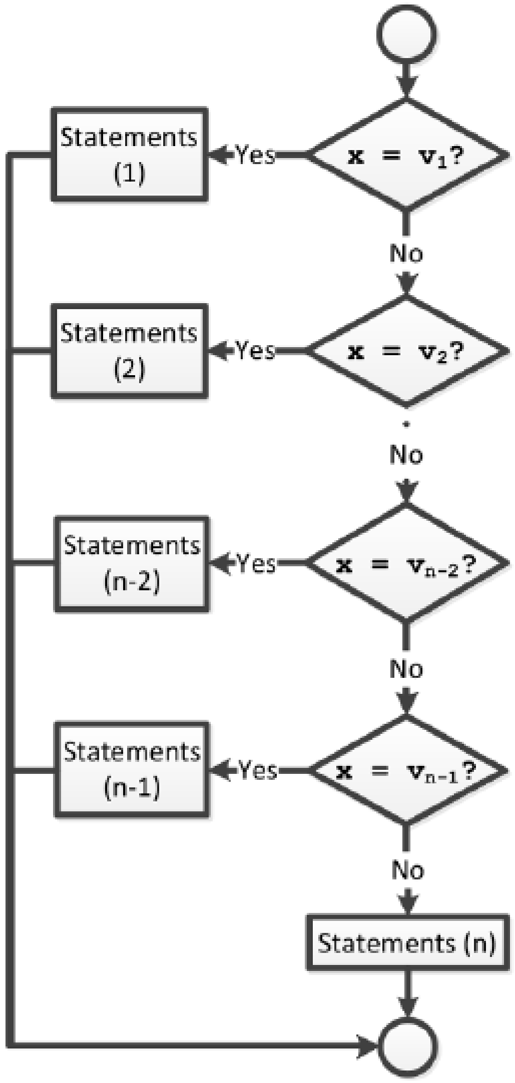
Design an algorithm that reads three numbers, then find and output the maximum entered value.

The case selection control structure:

- Multiple conditions are used to test where a single value matches some case.

- Pseudocode for the case structure:

```
IF( Condition1 ) THEN
        Statement1
ELSE IF( Condition2 ) THEN
        Statement2
ELSE IF( Condition3 ) THEN
        Statement3
            ⋮
```

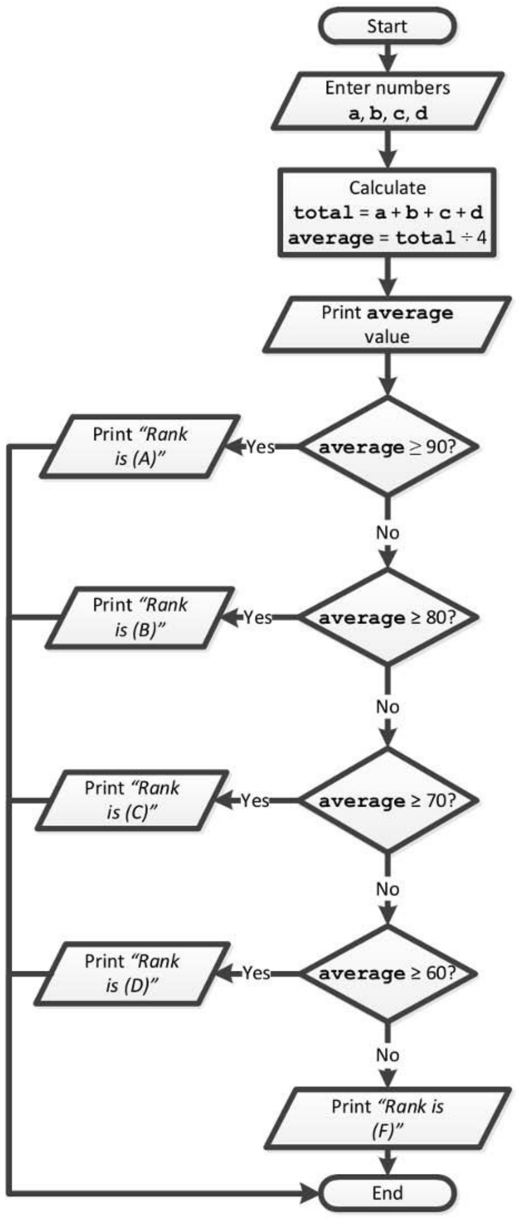


Example:

Write an algorithm, that reads four grades of a student, and output the average, then output the student rank, based on the following table:

Average Domain	Rank
90 and more	A
80 and more	B
70 and more	C
60 and more	D
Less than 60	F

FLOWCHART



PSEUDOCODE

```
Start
Enter numbers $a$, $b$, $c$, $d$
Calculate $total = a + b + c + d$
Calculate $average = total \div 4$
Print $average$ value
If ($average \geq 90$) Then
    Print "Rank is (A)"
Else If ($average \geq 80$) Then
    Print "Rank is (B)"
Else If ($average \geq 70$) Then
    Print "Rank is (C)"
Else If ($average \geq 60$) Then
    Print "Rank is (D)"
Else
    Print "Rank is (F)"
End
```

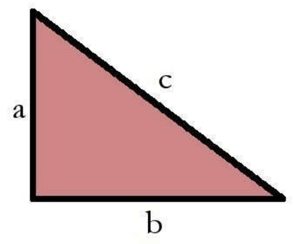
Example:

Write an algorithm, that reads a character, then determine whether it is VOWEL letter or not.
Note: Vowels letters are: A, E, I, O, U.

Exercises:

Use Flowchart and Pseudocode strategies, design an algorithm that solves each of the following problems:

1) The inputs are three numbers (a, b and c), representing the measurements of a triangle, the output should indicate if the inputs can form a right triangle (has the largest angle = 90), this can be done by using well-known equation: ($c^2 = a^2 + b^2$).



2) In some universities, the student is full-time, when he registers (12) credit hours or more, or part-time if he registers less than (12) credit hours. Use the credits as input to print the tuition fees for that student, given that full-time fees are fixed at (2000), and the part-time fees are calculated as (the number of credits × 175).

3) The input is the total money spent at the mall, and the output is the number of points earned. For all customers, the awarded points are (1 point per 1 spent dollar), and if a customer spends more than (200) dollars, then he will get (30) more points.

4) The input is the student average in the school's final year, the output should be (Pass) if the grade is greater than or equal to (50%), or (Fail) otherwise. Also, as an additional output, the word (Awarded) should be printed if the average was (95%) or more.

5) Input the birthdate of a person (***bd/bm/by***) and the current date (***cd/cm/cy***), and calculate and display his age. (Assume the month to be 30 days).

6) Prompting the user to enter three integer numbers, and then displaying the numbers in ascending order.

7) Assume four marks were entered, if all the marks in the same range (e.g. between 0 and 100), write the algorithm that will calculate the average of the top three marks.

8) Read the coefficients a, b and c of the ***quadratic equation:*** ax^2+bx+c, then calculate and display the roots of the equation.

- When the Discriminant $b^2 - 4ac$ is negative, then NO real roots.
- When the Discriminant $b^2 - 4ac$ is ZERO, then there is single root ($r = -b / 2a$).
- When the Discriminant $b^2 - 4ac$ is positive, then there are two roots $r1$ and $r2$:

$$r1, r2 = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

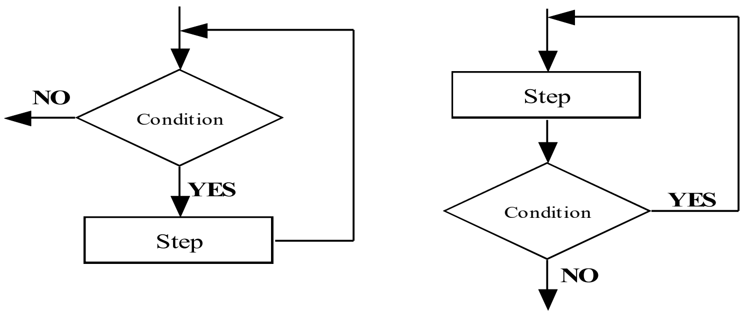
9) Calculator Algorithm: Input two numbers and the operation to be performed. The output is the result of applying the operation on the two numbers. The algorithms supports Addition ('+'), Subtraction ('-'), Multiplication ('*'), and Division ('/').

10) The input is one number representing base salary (BS), the output should be the amount of net salary (NS), which is calculated using the equation: (NS = BS + Bonus). The bonus is calculated by multiplying the base salary with its respective bonus rate. Bonus rates are listed in the following table:

Base Salary Domain	Bonus Rate
Less than (1000)	0.015
1000 - 1499	0.020
1500 - 1799	0.030
1800 - 2099	0.055
(2100) and more	0.075

* REPETITION CONTROL STRUCTURE:

- A ***Repetition*** (or ***Looping***, or ***Iteration***) control structure shows one or more steps that are being carried out repeatedly as long as some condition is being satisfied.



Possible *Looping* Control Structures

- Types of Loops:
    - Counter-Controlled: a counter controls the end of loop.
    - Event-Controlled: mostly, a special value stops the loop.

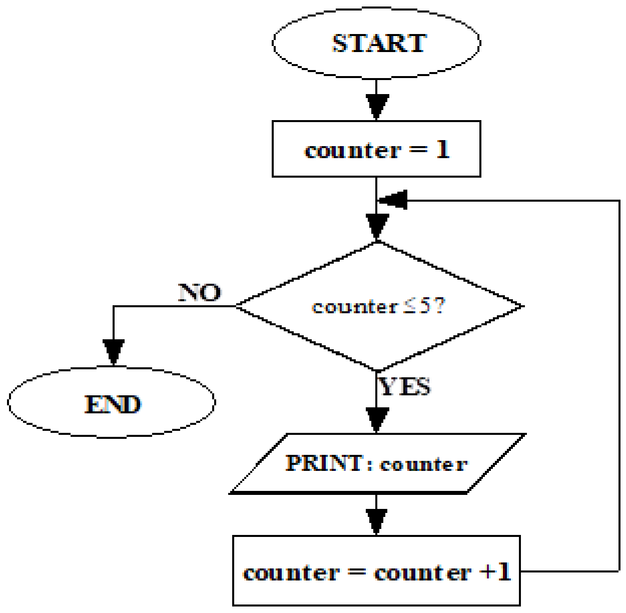
* COUNTER-CONTROLLED LOOPS:

- Iterates fixed number of times.

Example:

Write an algorithm that prints the numbers from 1 to 5 in an ascending order.

Tracing the flowchart



Example:

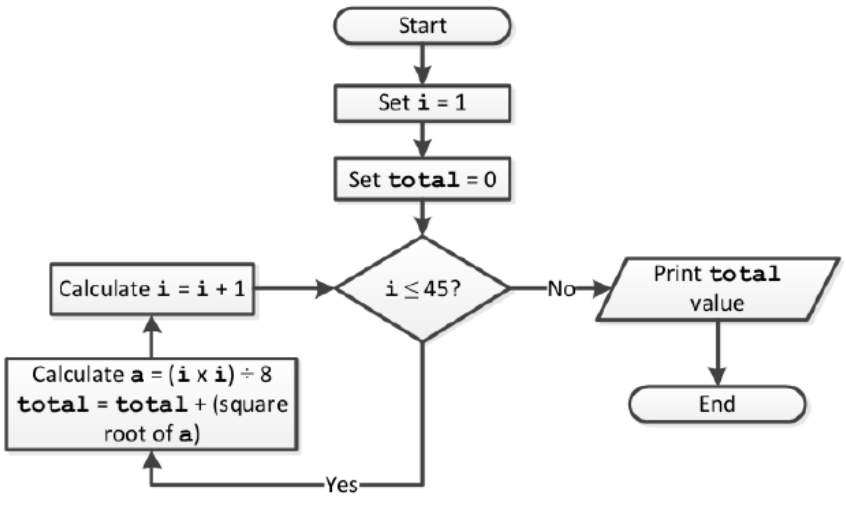
Write an algorithm that reads four numbers, calculates their sum and average, and prints output the results. (Trace the algorithm by applying random input numbers).

Example:

Write an algorithm that reads four numbers, calculates and prints the sum of EVEN entered values.

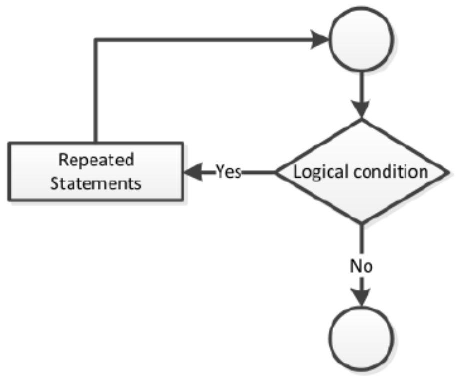
Example:

Trace the following algorithm and find the output.
What is the mathematical formula this algorithm represents?



- The **WHILE** keyword is used in the Pseudocode to indicate Looping.

```
WHILE( Condition )
    Statement
```

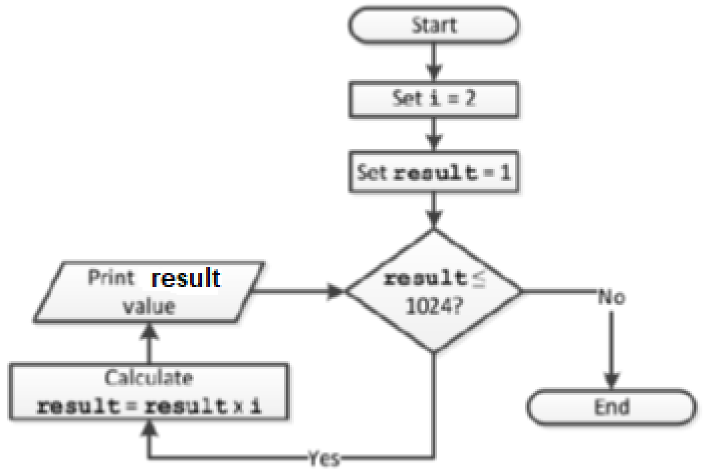


Example:

Design an algorithm that will print the following sequence:

2 4 8 16 32 64 128 256 512 1024

Flowchart:



Pseudocode:

```
Start
Set $i = 2$
Set $result = 1$
While ($result \leq 1024$)
    Calculate $result = result \times i$
    Print $result$ value
End
```

* EVENT-CONTROLLED LOOPS:

- Iterates **UNTIL** a special value is encountered.

Example:

Design an algorithm that will continuously read characters until **'#'** is entered. The output is how many uppercase letters were entered.

Example:

A customer has 500 NIS cash money and wanted to buy some items from a market. Design an algorithm that will continuously read the prices of items until their sum exceeds the 500 NIS, then output how many items can be bought with this amount of money.

Exercises:

Use Flowchart and Pseudocode strategies to design an algorithm that solves each of the following problems:

1) Read eight numbers and print how many negative numbers were entered.

2) Read a number and calculate and display the sum of its divisors.

   Example: if the user inputs (15) then the divisors will be: (1, 3, 5, 15) and the output will be their total (24).

3) Input five numbers and determine the maximum one.

4) Read two integers **X** and **n**, and prints the value of **X** raised to **n** (i.e. X^n).

   Note: $$X^n = \underbrace{X * X * X * \ldots X}_{n \text{ times}}$$

5) Input a positive number **N** and prints out its factorial (i.e. **N!**)

   Note: $$N! = \underbrace{N * (N-1) * (N-2) * \ldots * 2 * 1}_{n \text{ times}}$$

6) Compute the total of the digits, for some entered integer. An integer is a number without fraction.

7) Display a positive integer number, which is entered by the user, in reversed order.

   Example: when the input is (2753), the output will be (3572).

   Hint: use remainder and division by (10).

8) Compute the sum of all integers between two entered values (n1) and (n2), given that (n1) should be greater than (n2).

9) Find the greatest common divisor (GCD) of two positive integers. If any of these numbers is not positive, print a suitable message.

10) The user enters as much numbers as possible, this should continue if the input was not negative. After a negative number is entered, the input stops, and the output should be the count of the entered values. Do not include the last negative value into the count.
Example: if the entered values were (1, 6, 0, 4, 8, 11, 36, 2, -10), the output should be (8).

11) The user enters a positive integer (n), this is followed by entering more (n) numbers, and then output their average. Example: if the user first enters (5) to be stored as (n), then he should enter (5) other values, like: (9, 7, 3, 6, 10), the total of these values = 9 + 7 + 3 + 6 + 10 = 35, and their average is (35 ÷ 5) = 7, so the output is (7).

12) Write a C++ program that prompts the user to enter an integer number, and prints the digits of this number on separate lines: For Example: if the entered number is 65082 your program displays:

```
2
8
0
5
6
```

13) Enter an integer number and determine whether it’s a PRIME number or not. (*Note:* a PRIME number only divides on itself and 1).

CHAPTER QUESTIONS:

Q1) State whether each of the following statements is **TRUE** or **FALSE**.

Statement		TRUE / FALSE
[1]	Semantic errors are discovered during program run-time.	
[2]	Pascal is a low-level programming language.	
[3]	Instructions in assembly language are meaningful words called mnemonics.	
[4]	Pseudocode will not follow strict lexical rules when forming an algorithm.	
[5]	A language translator transforms machine code into source code.	
[6]	Program logic may combine different control structures.	
[7]	It is easy for programmers to write computer commands using machine languages.	
[8]	An interpreter will not generate an object code file for later execution.	
[9]	HTML is mainly used for game development.	
[10]	An algorithm should specify all the details of implementation.	

Q2) Choose the best answer:

1	In which phase of SDLC does the software developer analyses whether software can be prepared to fulfill all the requirements of the end user?	**A.** Design **B.** Development **C.** Testing **D.** Planning
2	A program must be converted to _________ language to be executed by a computer.	**A.** Assembly **B.** Machine **C.** High level **D.** Very high level
3	A __________ error does not prevent the program from running, but causes it to produce incorrect results.	**A.** syntax **B.** hardware **C.** logic **D.** fatal
4	A(n) __________ is a set of well-defined logical steps that must be taken to perform a task.	**A.** logarithm **B.** plan of action **C.** logic schedule **D.** algorithm
5	An informal language that has no syntax rules and is not meant to be compiled or executed is called __________.	**A.** faux code **B.** pseudocode **C.** Python **D.** a flowchart
6	A __________ structure can execute a set of statements only under certain circumstances.	**A.** sequence **B.** circumstantial **C.** decision **D.** boolean
7	A __________ -controlled loop repeats a specific number of times.	**A.** event **B.** condition **C.** decision **D.** count

جامعة بوليتكنك فلسطين

CHAPTER EIGHT

COMPUTER PROGRAMMING USING C++

Prepared by:
Eng. Yousef Salah
Dr. Mohammad Abu Taha

This material developed under the objectives of FESTEM project funded by the EU.
https://festem.ps/

Co-funded by the Erasmus+ Programme of the European Union

CHAPTER EIGHT
COMPUTER PROGRAMMING USING C++

TEXTBOOK: C++ Programming: From Problem Analysis to Program Design, 5th Edition
AUTHOR: D.S. Malik

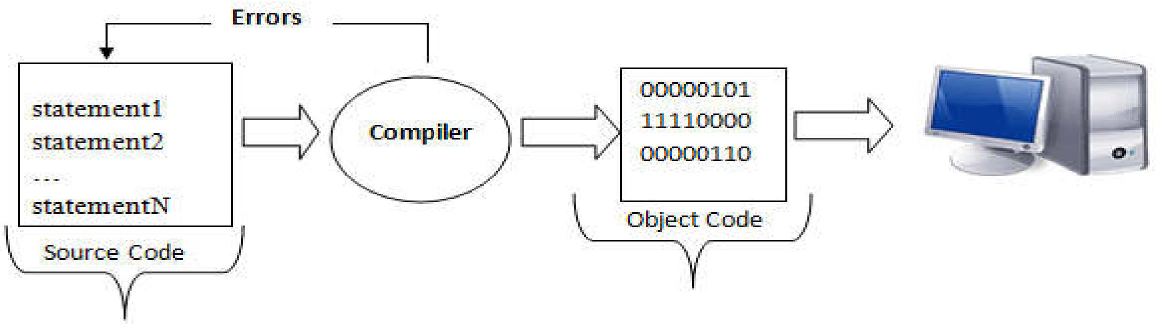
INTRODUCTION

- A Programming Language: set of words, symbols and abbreviations used to construct a computer program.
- A computer program is a collection of commands (a.k.a. instructions, or statements) that directs the computer to do a task.

- A Programming Language has:
  - Syntax: grammar rules used to construct valid (i.e. correct or legal) statements.
  - Semantics: meaning of statements.

C++ Programming Language

- C++ is a high level general-purpose programming language, which was developed by Bjarne Stroustrup at Bell Labs since 1979, as an extension of the C language.
- C++ is a compiled language, which means that it has a compiler.
- Many compilers were available by early 1990s, this let American National Standard Institution (ANSI) and International Standard Organization (ISO) to standardize the syntax of C++.
- We will study ANSI/ISO Standard C++ as it was standardized in mid-1998.
- A compiler translates source code into its corresponding machine code (or object code).



- Examples of C++ compilers, or sometimes called IDE (Integrated Development Environment): Dev C++, MS Visual C++, Eclipse, NetBeans, code::Blocks, Turbo C++, Borland C++, GCC.
- Web-based compilers for different programming languages:
  https://www.onlinegdb.com/online_c++_compiler

Rules and Syntax of C++

❖ C++ is a case-sensitive language.
This means C++ differentiates between lowercase and uppercase letters.
A ≠ a
Total ≠ total

By the way:

{ } are called Braces

( ) are called Parenthesis

< > are called Triangle Brackets

[ ] are called Brackets

❖ The basic command in C++ is called a "Statement".
Each statement in C++ ends by a semicolon **;**

❖ A C++ block is a set of statements enclosed between braces { }

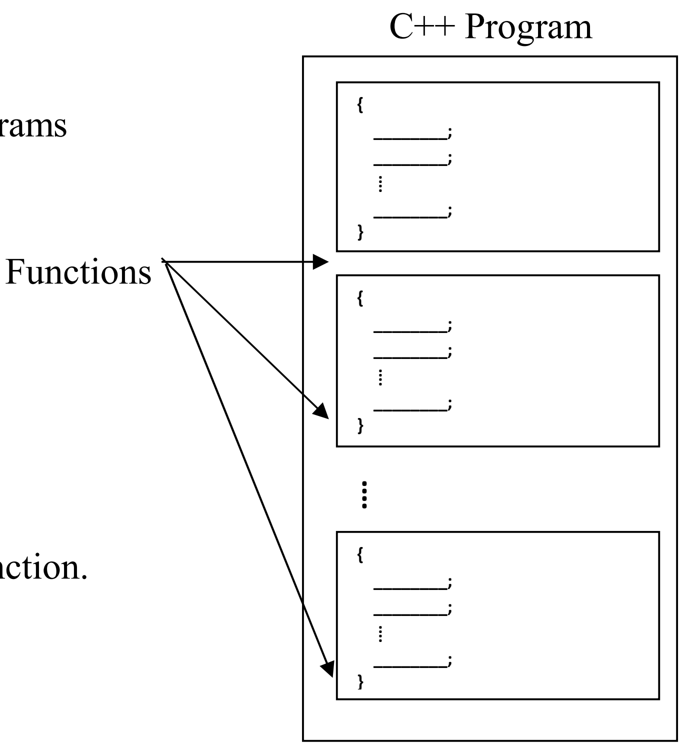
```
{
   ________;
   ________;
   ⋮
   ________;
}
```

❖ A C++ program is a collection of sub-programs called **"Functions"**.

❖ A Function is a block of code.

❖ A C++ program should have a function called **main()** function.

❖ Program execution starts by the **main()** function.



❖ C++ Constructs (Vocabularies/ Alphabets):

1 CHARACTERS

A …. Z a …. z	Letters
0, 1, 2 …. 9	Digits
SPACE TAB ENTER	Whitespaces or Blanks

2 SPECIAL SYMBOLS

{ } ( ) < > [ ] + - * / % . ; , ! ? & | = # " ' …

Some symbols are not allowed, for example: ÷ @ ≤ ∑

3 RESERVED WORDS (OR KEYWORDS)

- Keywords are words with special meanings.
- C++ has more than 70 keywords like (*keywords are all in lowercase letters*):

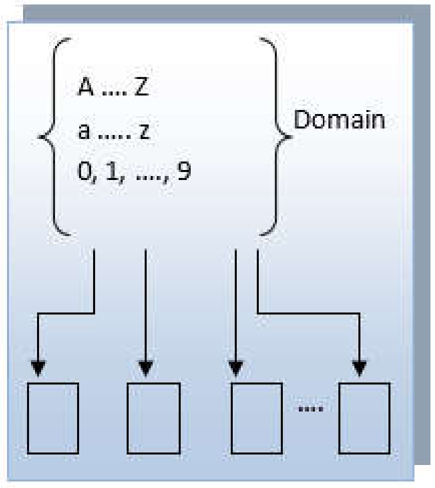
```
auto    const     double  float  int       short
struct   unsigned   break  continue  else    for
long       signed  switch   void  case   default    enum
goto    register  sizeof  typedef  volatile
char    do        extern  if      return    static
union    while
```

❖ C++ Token:

- A ***token*** is the smallest individual unit in a program.
- For Example: + is a token, ++ is a token, and == is a token.
- C++ tokens involve:
  - Reserved words.
  - Special Symbols.
  - Identifiers.

❖ Identifiers:

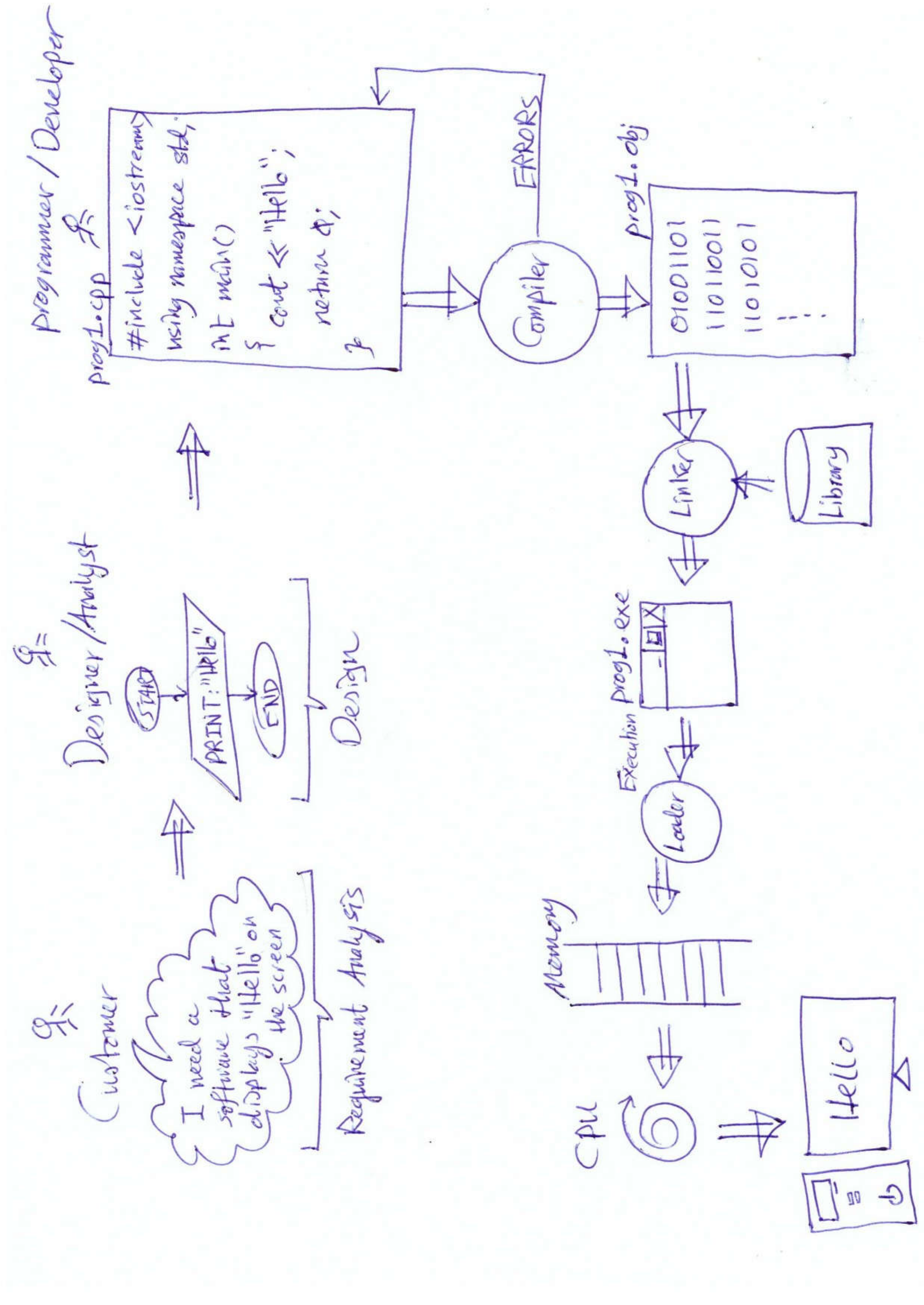
- Identifiers are rules of creating valid names in a C++ program.
- Names includes the names of variables, constants, functions, ….
- These rules are:
  - C++ identifier consists of letters, digits and the underscore character _ (cannot contain a special symbol.)
  - C++ identifier cannot begin by a digit.
  - C++ identifier cannot be a reserved word.



Exercise: ***Determine whether the following C++ identifiers are Valid or Invalid?***

Identifier		VALID/ INVALID		Identifier	VALID/ INVALID
[1]	Sum_of_Squares		[9]	Total	
[2]	G 10		[10]	40Hours	
[3]	Box_22		[11]	_Count	
[4]	alpha-12		[12]	object(1)	
[5]	else		[13]	age#1	
[6]	Return		[14]	Folder2000	
[7]	one+two		[15]	2ndValue	
[8]	my salary		[16]	var.123	

Problem Analysis-Coding-Execution Cycle



```
#include <iostream>
using namespace std;
int main()
{
cout << "Hello";
return 0;
}
```

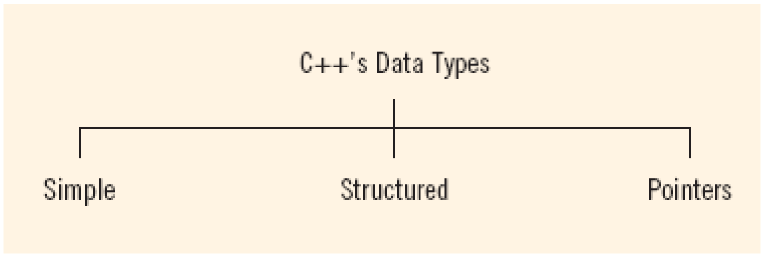
```
01001101
110110011
11010101
.....
```

My First C++ Program

❖ Any computer program can be viewed as:
- Data
- Operations (or instructions) to manipulate these data

C++ Data Types

- ❖ A Data Type: set of values together with set of operations.
- ❖ C++ Supports the following categories of data types:



- ❖ Simple data types include: (handled as one unit in memory)
  - Integral
  - Floating-Point
  - Enumeration

❖ **<u>Integral Data Type:</u>**

- Includes: **<u>int</u>**, **<u>char</u>**, **<u>bool</u>**

$\boxed{1}$ The **<u>int</u>** data type

- Used to hold whole numbers (numbers without fractional part).
- Example: 12, 0, -658, +98, 67, 35871 are all integer numbers.
- No commas are used within an integer.
  For instance: **12,638** is an invalid integer value in C++.

$\boxed{2}$ The **<u>bool</u>** data type

- Used to hold logical values.

$$\overbrace{\text{true} \qquad\qquad\qquad \text{false}}$$

- **<u>true</u>**, **<u>false</u>**, and **<u>bool</u>** are all reserved words.

$\boxed{3}$ The **char** data type

- Used to hold character data (alphanumeric values).
- Characters include:
    - Letters (A to Z and a to z)
    - Digits (0, 1, 2, …. , 9)
    - Special Symbols (#, !, @, &, %, ….)

- A character value is enclosed between two single quotes ‘ ’
  ‘A’ ‘b’ ‘5’ ‘+’ ‘>’ ‘;’ ‘ ’

- ‘abc’ ‘520’ ‘v1’ ‘Hello!’ are all invalid characters, why?

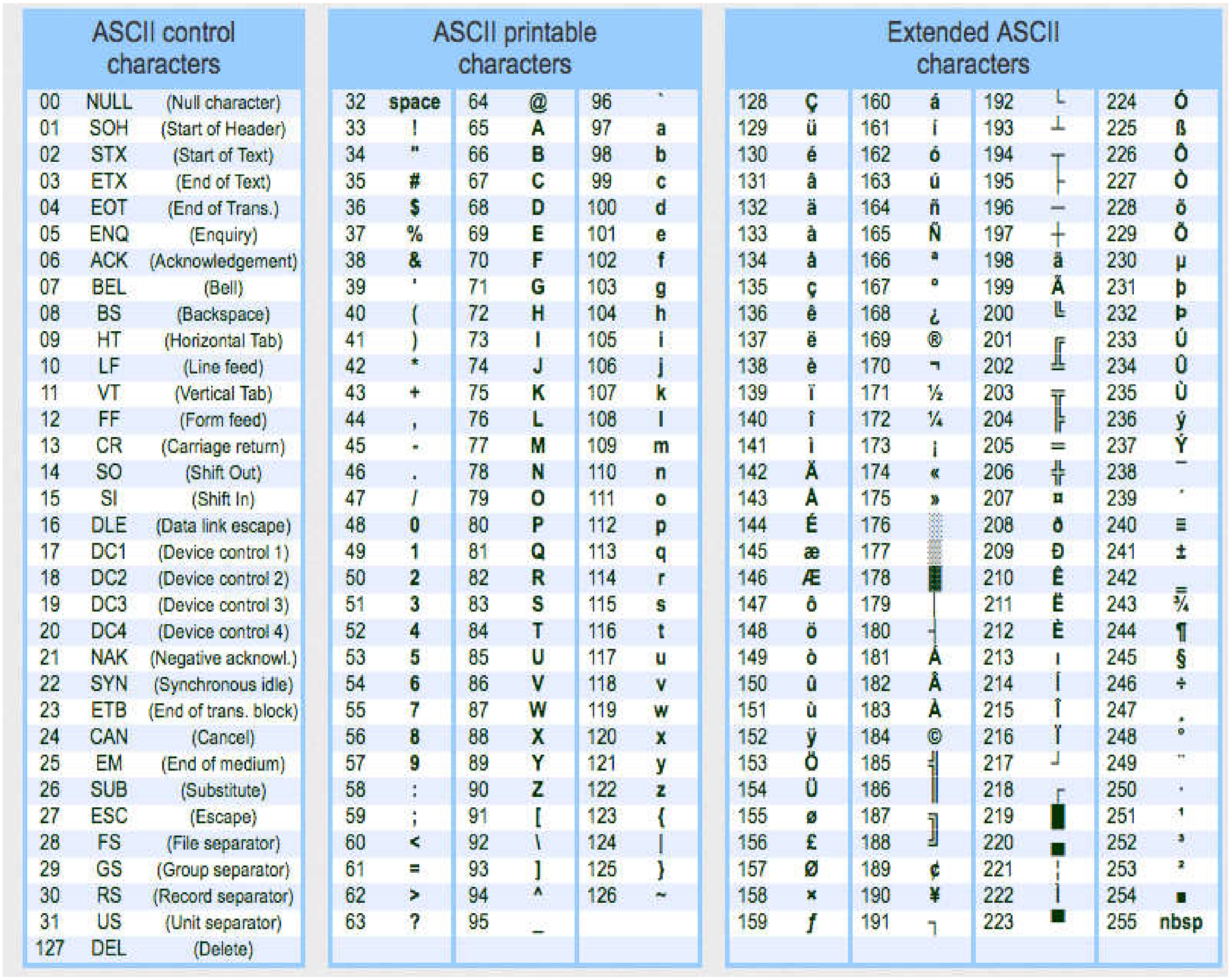
- C++ uses ASCII code to represent characters. Each character has a value.

‘0’ $\longrightarrow$ 48	‘A’ $\longrightarrow$ 65	‘a’ $\longrightarrow$ 97
‘1’ $\longrightarrow$ 49	‘B’ $\longrightarrow$ 66	‘b’ $\longrightarrow$ 98
⋮	⋮	⋮
‘9’ $\longrightarrow$ 57	‘Z’ $\longrightarrow$ 90	‘z’ $\longrightarrow$ 122

By the way:
ASCII stands for American Standard Code for Information Interchange.

- Why ASCII?
- How many bits used to encode each character in ASCII? ________
- How many characters are represented in ASCII? ________
- Is there any other character representation code?

ASCII CODE



ASCII control characters

00	NULL	(Null character)
01	SOH	(Start of Header)
02	STX	(Start of Text)
03	ETX	(End of Text)
04	EOT	(End of Trans.)
05	ENQ	(Enquiry)
06	ACK	(Acknowledgement)
07	BEL	(Bell)
08	BS	(Backspace)
09	HT	(Horizontal Tab)
10	LF	(Line feed)
11	VT	(Vertical Tab)
12	FF	(Form feed)
13	CR	(Carriage return)
14	SO	(Shift Out)
15	SI	(Shift In)
16	DLE	(Data link escape)
17	DC1	(Device control 1)
18	DC2	(Device control 2)
19	DC3	(Device control 3)
20	DC4	(Device control 4)
21	NAK	(Negative acknowl.)
22	SYN	(Synchronous idle)
23	ETB	(End of trans. block)
24	CAN	(Cancel)
25	EM	(End of medium)
26	SUB	(Substitute)
27	ESC	(Escape)
28	FS	(File separator)
29	GS	(Group separator)
30	RS	(Record separator)
31	US	(Unit separator)
127	DEL	(Delete)

ASCII printable characters

32	space	64	@	96	`
33	!	65	A	97	a
34	"	66	B	98	b
35	#	67	C	99	c
36	$	68	D	100	d
37	%	69	E	101	e
38	&	70	F	102	f
39	'	71	G	103	g
40	(	72	H	104	h
41	)	73	I	105	i
42	*	74	J	106	j
43	+	75	K	107	k
44	,	76	L	108	l
45	-	77	M	109	m
46	.	78	N	110	n
47	/	79	O	111	o
48	0	80	P	112	p
49	1	81	Q	113	q
50	2	82	R	114	r
51	3	83	S	115	s
52	4	84	T	116	t
53	5	85	U	117	u
54	6	86	V	118	v
55	7	87	W	119	w
56	8	88	X	120	x
57	9	89	Y	121	y
58	:	90	Z	122	z
59	;	91	[	123	{
60	<	92	\	124	|
61	=	93	]	125	}
62	>	94	^	126	~
63	?	95	_		

Extended ASCII characters

128	Ç	160	á	192	└	224	Ó
129	ü	161	í	193	┴	225	ß
130	é	162	ó	194	┬	226	Ô
131	â	163	ú	195	├	227	Ò
132	ä	164	ñ	196	─	228	õ
133	à	165	Ñ	197	┼	229	Õ
134	å	166	ª	198	ã	230	µ
135	ç	167	º	199	Ã	231	þ
136	ê	168	¿	200	╚	232	Þ
137	ë	169	®	201	╔	233	Ú
138	è	170	¬	202	╩	234	Û
139	ï	171	½	203	╦	235	Ù
140	î	172	¼	204	╠	236	ý
141	ì	173	¡	205	═	237	Ý
142	Ä	174	«	206	╬	238	¯
143	Å	175	»	207	¤	239	´
144	É	176	░	208	ð	240	≡
145	æ	177	▒	209	Ð	241	±
146	Æ	178	▓	210	Ê	242	‗
147	ô	179	│	211	Ë	243	¾
148	ö	180	┤	212	È	244	¶
149	ò	181	Á	213	ı	245	§
150	û	182	Â	214	Í	246	÷
151	ù	183	À	215	Î	247	¸
152	ÿ	184	©	216	Ï	248	°
153	Ö	185	╣	217	┘	249	¨
154	Ü	186	║	218	┌	250	·
155	ø	187	╗	219	█	251	¹
156	£	188	╝	220	▄	252	³
157	Ø	189	¢	221	¦	253	²
158	×	190	¥	222	Ì	254	■
159	ƒ	191	┐	223	▀	255	nbsp

❖ **Floating-Point Data Type:**

- Includes: **float**, **double**
- Used to hold real numbers (numbers with fractional part).
- Floating-point numbers can be represented in two notations:
  - Decimal : Ex: 5400.0 0.0001
  - Scientific: Ex: 5.4E+003, 1.0E-004

- Examples:

$$1.2345 = \underbrace{12345}_{\text{significand}} \times \underbrace{10}_{\text{base}}^{\overbrace{-4}^{\text{exponent}}}$$

Real Number	C++ Floating-Point Notation
75.924	7.592400E1
0.18	1.800000E-1
0.0000453	4.530000E-5
-1.482	-1.482000E0
7800.0	7.800000E3

By the way:
.25 and 25. are valid C++ Floating-Point numbers.
.25 ≡ 0.25
25. ≡ 25.0

$\boxed{1}$ **float** data type

- The data type float is used in C++ to represent any real number between -3.4E+38 and 3.4E+38.

$\boxed{2}$ **double** data type

- The data type double is used in C++ to represent any real number between -1.7E+308 and 1.7E+308.

Note: Minimum and Maximum values are system dependent (compiler dependent).

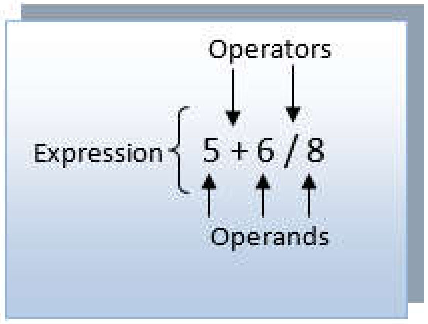
The string Data Type

- ❖ The string data type is not a built-in C++ data type.
- ❖ Requires #include <string>
- ❖ The string data type is used to store text values.
- ❖ A C++ string is enclosed between two double quotations "……………….."
- ❖ A string is sequence of zero or more characters.
  - For Example: "Hello World" "I am 18 years old!" "A" "2017"
- ❖ Zero-length string is called the NULL string "";
- ❖ C++ string cannot be separated into multiple lines.

Example: "I Live
in Palestine." } **Invalid**

C++ Arithmetic Operators

- ❖ C++ is capable of performing Addition (+), Subtraction (-), Multiplication (*), Division (/), and Modulus (or Remainder) (%).
- ❖ Arithmetic expression: numbers separated by arithmetic operators.
- ❖ Data at which the operator to be applied on are called: Operands.



- ❖ An operator can be:
  - Unary Operator: has a single operand. (Ex: +5, -num)
  - Binary Operator: has two operands. (Ex: , 15 + 6, x * y)
- ❖ + and - are unary and binary operators.
- ❖ *, /, % are binary operators. (6 * is not a valid expression)

❖ The Division Operator /

- If the two operands are integers, the result is an integer.
- Examples: 5 / 2 ➔ 2 4 / 9 ➔ 0 14 / 3 ➔ 4 7 / 2 ➔ 3
  7.0 / 2 ➔ 3.5 7 / 2.0 ➔ 3.5 7.0 / 2.0 ➔ 3.5
  5 / 0 ➔ Logical ERROR

❖ The Modulus Operator %

- % is used only with integer operands, it yields the remainder of division.
- Examples: 34 % 5 ➔ 4 15 % 4 ➔ 3 2 % 7 ➔ 2
  25.5 % 3 ➔ ERROR 39 % 0 ➔ Logical ERROR

Exercise: *What is the output of the following C++ Program?*

```
#include <iostream>
using namespace std;
void main()
{
    cout << "2 + 5 = " << 2 + 5 << endl;
    cout << "3.0 + 9.4 = " << 3.0 + 9.4 << endl;
    cout << "34 - 20 = " << 34 - 20 << endl;
    cout << "16.3 - 5.2 = " << 16.3 - 5.2 << endl;
    cout << "2 * 7 = " << 2 * 7 << endl;
    cout << "5 / 2 = " << 5 / 2 << endl;
    cout << "4.2 * 2.5 = " << 4.2 * 2.5 << endl;
    cout << "37 % 5 = " << 37 % 5 << endl;
    cout << "4 % 6 = " << 4 % 6 << endl;
}
```

❖ **Order of Precedence:**

- An expression is evaluated according to precedence rules.
- Associativity determines how operators of the same precedence are grouped in the absence of parentheses.

Precedence	Operator	Associativity
Higher	Parenthesis ( )	
↑	* / %	Left to Right
Lower	+ -	Left to Right

❖ **Examples:**

```
cout << 2 + 3 * 5;                     cout << (2 + 3) * 5;




cout << 7 / 2 * 3;                      cout << 39 % 4 * 3 / 5;




cout << 6 / 4 + 3.9;                   cout << 'A' + 2;




cout << 3 * 7 - 6 + 9 * 5 / 4 + 5;           cout << "Hello"
                                                     <<  "World";




cout << 9 + 7 * 5  % 3 - 5 * 9 % 11 - 8 << endl;
```

EXAMPLE:

```
#include <iostream>
using namespace std;
int main()
{
     cout << "3 / 2 + 5.5 = " << 3 / 2 + 5.5 << endl;
     cout << "15.6 / 2 + 5 = " << 15.6 / 2 + 5 << endl;
     cout << "4 + 5 / 2.0 = " << 4 + 5 / 2.0 << endl;
     cout << "4 * 3 + 7 / 5 - 25.5 = "
     << 4 * 3 + 7 / 5 - 25.5 << endl;
     return 0;
}
```

C++ COMMENTS

- Comments are explanatory Sentences in the source code.
- Comments are not executed, and are ignored by the compiler.
- Comments are directed to the reader of the program.
- There are two styles of writing comments in C++:

C++ Style
Single-Line Comments

```
// ……………..
// ……………..
```

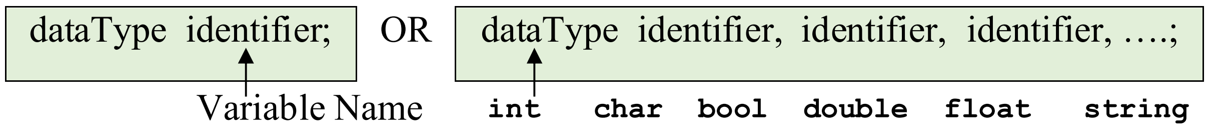
C Style
Multiple-Line
Comments

```
/*
………..
………..
*/
```

HOW C++ ALLOCATES MEMORY LOCATIONS?

Using **Declaration Statement** instructs the computer to put data into the computer's memory.

❖ **SYNTAX of Declaration Statement:**



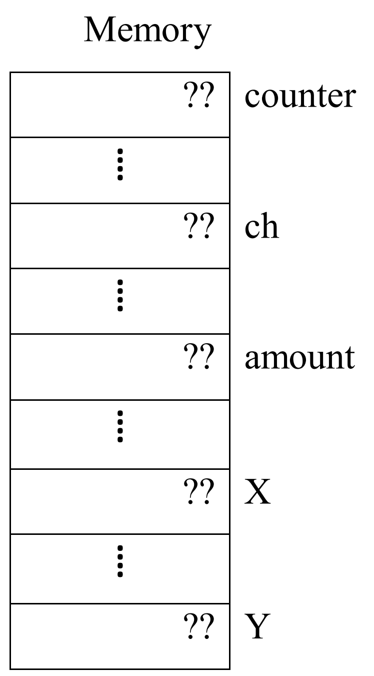
❖ A Declaration Statement reserves a memory location called ***variable***.

❖ The content (value) of a ***variable*** may change during program execution.

By the way:

- **A variable is a memory location.**
- **NO variables may have the same names (in the same block).**

❖ Examples:

int counter; /* tells the computer to allocate a memory location to store an integer value */	
char ch; // ch is a variable, it will hold a character value	

double amount;
// amount will contain a real number
cout << amount; // ?? UNKNOWN	

int X, Y; // two integer variables are created	

EXERCISE: *Which of the following C++ declaration statements are correct?*

	Declaration	VALID/ INVALID		Declaration	VALID/ INVALID
[1]	string my name;		[6]	int var@2017;	
[2]	Bool isOK,		[7]	char c1, c2, c3;	
[3]	char ch#1;		[8]	int Double;	
[4]	int I, j, k;		[9]	int 10;	
[5]	double 4sale;		[10]	int return;	

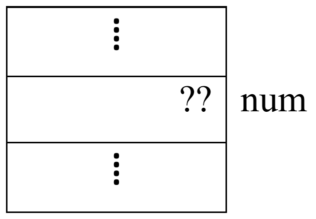
❖ **Assignment Statement:**

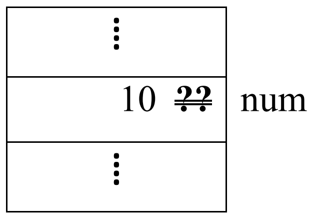
- Assigns (stores) a value into a variable. (= is called the assignment operator)
- **SYNTAX of Assignment Statement:**

$$\text{variable} \leftarrow\!\!\!\!\!\!\!\!\!\!\quad = \text{expression;}$$

```
variable = expression;
```

EXAMPLE:

int num; // declares an integer variable called ***num***
 cout << num; ??
 // UNKOWN value will be printed
	Memory
	 

num = 10; //Assignment
 cout << num; 10
	Memory
	 

num = 8 * 5 – 13;

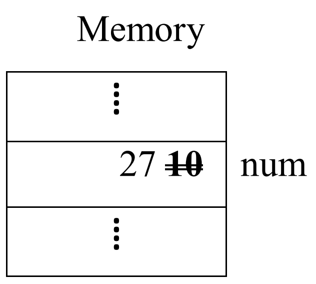
cout << num; $\boxed{27}$

2 + 3 = num; // ERROR

38 = num; // ERROR

num = num + 2; // Valid

cout << num; // Prints 29



EXAMPLE:

```
#include <iostream>
#include <string>
using namespace std;
int main()
{
     int  val;  char ch;
     float tax; string str;
```

cout << “val”; ----> $\boxed{\quad}$

cout << val; ------> $\boxed{\quad}$

val = 6 + 4 * 2;

cout << val; ------> $\boxed{\quad}$

val = 17 % 7 ;

cout << val; ------> $\boxed{\quad}$

ch = ‘A’;

cout << ch; ------> $\boxed{\quad}$

ch = A; //ERROR int quantity = 12,000; //ERROR

ch = ‘@’; // VALID ch = ‘465’; // ERROR

ch = ‘7’; // VALID val = 465; // VALID

```
    tax = 8.15;
    cout << tax; ------>

    tax = 1.2E3;   // VALID

    cout << tax; ------>

    tax = 5.0 / 2;  cout << tax;   // Prints ______

    str =  I'm Programming in C++;   // ERROR
    str =  "I'm Programming in C++";   // VALID
    cout << "str";  // Prints _______
    cout << str;     // Prints _______

    double ch;  // ERROR, NO variables may have the same name
    double Ch;  // VALID

return 0;
}
```

❖ **Initialization Statement:**

- Assigning a value into a variable ***during declaration***.
- **SYNTAX of an Assignment Statement:**

$$\boxed{\text{DataType identifier} \;=\; \text{value;}}$$

EXAMPLES:

```
int  num = 5;
char code = 'm';
string text = "return";
float alpha = .83;
int x = 10, y = 15;
double beta = ;  // ERROR
```

```
int a = 1, b, c=2;
b = c;  // VALID
cout << c; // prints ______
int temp = 12 – 4  % 3;    // VALID
int nVar = a * b – c;       // VALID
```

❖ Named Constant:

- A named constant is a named memory location at which its value (i.e. content) ***cannot be changed***.
- **SYNTAX of a Named Constant:**

```
const   DataType  identifier  =  value;
  ↑
reserved word
```

Example:

```
#include <iostream>
#include <string>
using namespace std;

int main()
{
     const float PI = 3.14;
     const double TAX = 0.17;
     const char STAR = '*';
     const int MAX = 20;
     const int MIN;    //ERROR

     PI = 3.1428;    //ERROR
     TAX = 0.15;   //ERROR

     int const;      //ERROR
     string STAR;    //ERROR
     const string PPU = "Palestine Polytechnic University";
     cout << PPU;    // Prints __________________________
      ⋮
}
```

The Input (Read) Statement

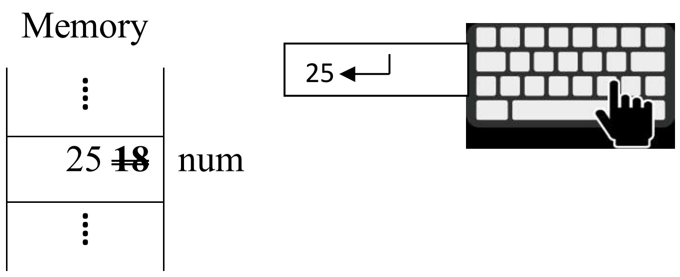
- A C++ program may need to input data ***during execution***.
- The ***cin*** and the extraction operator >> are used to prompt the user to enter a value (*in most program environments from the keyboard*), and to put it into a variable located in memory. (Interactive input)

- **SYNTAX of C++ read statement:**

cin >> variable; **OR** **cin >> variable >> variable >> ….;**

EXAMPLE:

```
int  num = 18;
cout << num;            // prints __________

cin >> num;
// The program pauses asking the user to
// enter a value from the Keyboard

cout << num;            // prints __________
cin >> num * 2;  // ERROR
cin >> 68;        // ERROR

const int AMOUNT = 100;
⋮
cin >> AMOUNT;   // ERROR, why?
```



By the way:

<< is called the **Insertion Operator.**

>> is called the **Extraction Operator.**

EXAMPLE:

```
#include <iostream>
#include <string>
using namespace std;
int main()
{
     string fstName;
     string LstName;
     int age;
     double weight;

     cout << "Enter first name, last name, age, "
     << "and weight:" << endl;

     cin >> fstName >> LstName;
     cin >> age >> weight;
     cout << "Name: " << fstName << " " << LstName << endl;
     cout << "Age: " << age << endl;
     cout << "Weight: " << weight << endl;
     return 0;
}
```

EXERCISE:

Write a C++ program that reads the radius of a circle, then finds and displays the area and circumference of this circle. (Draw the flowchart first).

Hint: $\boldsymbol{area = \pi r^2}$, $\boldsymbol{circumference = 2\pi r}$

```
// This program will find the area and circumference of a circle
#include <iostream>
using namespace std;
int main()
{




 }
```

Type Conversion (Casting and Coercion)

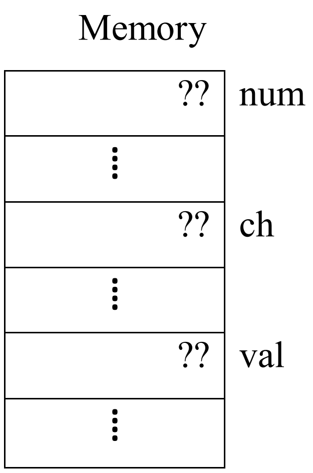
1 Coercion:

- Coercion means converting a **value** from data type to another data type **implicitly** (automatically) IF POSSIBLE.

int	$\longleftrightarrow$	**double** **float** **char**
int	$\xcancel{\longleftarrow}$	**string**

Example:

```
int  num;
char ch;
float val;

double num;  // ERROR
num = 4.9;  // Coercion
cout << num;  // Prints ___________

val = 2;  // Coercion
cout << showpoint;
cout << val;  // Prints ___________

ch = 65;  // Coercion
cout << ch;   // Prints ___________


val = "Hello";   //ERROR
num = "ABCD";  // ERROR
num = "321";  // ERROR


num = 'B';  // Coercion
cout << num;  // Prints _________
```



Note:

C++ will not print the fractional part if it equals ZERO.

```
float f = 8.0;
cout << f;
```

```
8
```

```
// showpoint forces the fraction to be
// printed even if it equals ZERO.
float f = 8.0;
cout << showpoint;
cout << f;
```

```
8.0
```

2 Casting:

- Casting means converting a **value** from data type to another data type **explicitly** IF POSSIBLE.
- C++ provides a cast operator **static_cast** to perform casting:

static_cast<DataType> (value);

- **static_cast** is a keyword.

EXAMPLES:

1	`cout << static_cast<int>(7.9) << endl;`
	
	`cout << showpoint;`
	`cout << static_cast<double>(25) ;`	

2	`cout  << static_cast<int> ('A');`
	
	`cout  << static_cast<char> (100);`	

3	`cout  << static_cast< float > (15) / 2;`	

4	`cout << showpoint;`
	`cout  << static_cast<float> (15/2);`	

5	`cout<< static_cast<int>( 7.8 + static_cast<double>(15) / 2 );`	

NOTE: Casting will not change the variable's data type.

EXAMPLE:

```
int alpha;
double beta;

beta = 2.5;
alpha = static_cast<int> (beta);  //  or alpha = beta;

cout << alpha;    // Prints _________
cout << beta;     // Prints __________
```

EXERCISE:	*What is the output of the following C++ code?*

Code Fragment	OUTPUT

int a = 5, b = 10;
char ch;
 	
cin >> a;
cin >> ch;
cin >> b;
$\left.\right\}$ cin >> a >> ch >> b;
 	
cout << a << endl << b << endl << ch;
	Assume the input is:
	27 R –18 ↵

`int firstNum, secondNum;`
`double z;`
`char ch;`
`string name;`
 	
`firstNum = 4;`
`secondNum = 2 *firstNum + 7;`
`z = (firstNum + 1) / 2.0;`
 	
`cout << z << endl;`
 	
`ch = 'A';`
`cout << secondNum << endl;`
 	
`cin >> secondNum;`
`cin >> z;`
 	
`firstNum = 2 *secondNum +`
`static_cast<int>(z);`
`cout << firstNum << endl;`
 	
`cin >> name;`
`secondNum = secondNum + 1;`
`cin >> ch;    cout << ch << endl;`
 	
`// ‘M’ in ASCII  = 77`
`firstNum = firstNum + static_cast<int>(ch);`
`z = firstNum - z;`
 	
`cout << z << endl << firstNum << endl <<`
`secondNum;`
	Assume the input is:
	8 15.3 Jenny M↵

Increment and Decrement Operators

- C++ provides the increment operator ++, which increases the value of a variable by 1, and the decrement operator, ––, which decreases the value of a variable by 1.

Pre-increment: ++variable
Post-increment: variable++
Pre-decrement: ––variable
Post-decrement: variable––

$\boxed{\text{var++ ; and ++var;} \equiv \text{var = var +1;}}$

$\boxed{\text{var-- ; and --var;} \equiv \text{var = var -1;}}$

- Example:

```
int ctr = 5;
ctr++;  // or  ++ctr;
cout << ctr; // Prints 6
(ctr+2)++; // ERROR
58++; // ERROR

double beta = 9.65;
beta --;  // or  -- beta;
cout << beta; // Prints 8.65

char ch = ‘c’;
ch++;
cout << ch; // Prints ________
```

- pre- and post- operators are different from each other when these operators are applied within an expression.
- For instance:

N = var++;	1) N = var; 2) var = var + 1;
N = ++var;	1) var = var +1; 2) N = var;

- Example:

int v = 8;
cout << v++; // Prints _______
cout << v; // Prints _______
	int v = 8;
	cout << ++v; // Prints _______
	cout << v; // Prints _______

- Example:

```
int  a = 4, b = 7, c;

c = a++ * --b;

cout <<  a;   // Prints _______
cout <<  b;  // Prints _______
cout <<  c;  // Prints _______
cout << ++c;  // Prints _______

a = b++ * 7;
cout <<  a;   // Prints _______
cout <<  b;  // Prints _______

c = 2 * a + ++b;
cout <<  c;  // Prints _______
cout <<  b;  // Prints _______
```

Escape Sequences

- Escape sequences are special characters that have special meanings inside a string or a character.
- C++ Escape Sequences: **\n \t \’ \” \\**

Consider the following table for their meanings:

	Meaning	
‘\n’	New Line	↵
‘\t’	Tab	SPACES
‘\\’	Backslash	\
‘\”’	Double Quotation	“
‘\’’	Single Quotation	‘

Remember:

char ch;
ch = 65; // Legal, coercion
ch = ’65’; // ERROR

cout << “I am Studying
a C++ Course.”; // not Legal

EXAMPLES:

1	`cout << "Hello \nWorld";`	
	`cout << "Hello" << '\n' << "World";`	
	`cout << "Hello" << endl << "World";`	

2	`cout << "I Live in "Palestine" country."; //ERROR`
	 `cout << "I Live in \"Palestine\" country.";`	

3	`cout << "\\C++\\ Course";`	
4	`cout << "Hello \t World";`	
5	`cout << "Hello \\t World";`	

6	`cout << "Hello \nthere. \nMy name is James." << endl;`
	`cout << "The newline escape sequence is \\n" << endl;`
	`cout << "The tab character is \'\\t\'" << endl;`	

7	`cout << "String \"Sunny\" contains five characters.";`	

More on Assignment Statements

- C++ provides compound assignment operators: += -= *= /= %=
- If a and b are two variables, then:

  a += b; ➔ a = a + b;

  a -= b; ➔ a = a - b;

  a *= b; ➔ a = a * b;

  a /= b; ➔ a = a / b;

  a %= b; ➔ a = a % b;

- Example:

  int x = 5, y = 7;

  x += y; // x = x + y;

  cout << x; // Prints 12

EXAMPLE 2-31

This example shows several compound assignment statements that are equivalent to simple assignment statements.

Simple Assignment Statement	Compound Assignment Statement
`i = i + 5;`	`i += 5;`
`counter = counter + 1;`	`counter += 1;`
`sum = sum + number;`	`sum += number;`
`amount = amount * (interest + 1);`	`amount *= interest + 1;`
`x = x / ( y + 5);`	`x /= y + 5;`

EXERCISES:

[1] *Use a compound assignment operator to convert the following simple assignment:*

x = x * y + z – 5;

[2] *Write the equivalent C++ code for each of the following mathematical equations:*

$$c = \frac{5}{9}(f - 32)$$

$$A = \frac{B+C}{D-E}$$

$$root = \frac{-b+(b^2-4ac)}{2a}$$

[3] *Write a C++ program that reads a 3-digit number, and prints its digits on separate lines.*

[4] *Newton's law states that the force, F, between two bodies of masses M1 and M2 is given by:*

$$F = k\left(\frac{M_1 M_2}{d^2}\right)$$

in which k is the gravitational constant and d is the distance between the bodies. The value of k is 6.67 X 10^{-8} dyn.cm^2/g^2. Write a C++ program that prompts the user to input the masses of the bodies, and the distance between the bodies. The program then outputs the force between the bodies.