College of Computer Science and information Technology

Computer Programming I

Course Information	
Course Title	Computer Programming I
Credits	4 Hours
Teaching Method	3 Hour of Lecture + 2 Hours Lab

Learning Outcomes

Course Description: This course covers fundamentals of algorithms, flowcharts, problem solving, programming concept, control structures and functions.

Course Outcomes: At the end of this course, students should be able to :

- Develop algorithms to solve "computer-solvable" problems.
- Test algorithms.
- Translate algorithms to C++ programs.
- Debug, run and test C++ "procedural" programs.

Topics

- Problem solving
- Algorithms
- What is programming?
- Basic elements of C++
- General Form of a C++ Program
- Comments and Reserved Words
- Identifiers , Variables and constant
- Data Types
- Arithmetic Operators and Operator Precedence
- Expressions
- Assignment Statement
- Declaring and Initializing Variables
- Input and output
- Control Structures
- Relational Operators and precedence
- Selection: if and if...else
- Compound (Block of) Statements
- Multiple Selections: Nested if
- Selection: Switch case
- Repetition: for Looping Structure
- User-defined functions
- Function declarations and call
    - Scope rule of an Identifier

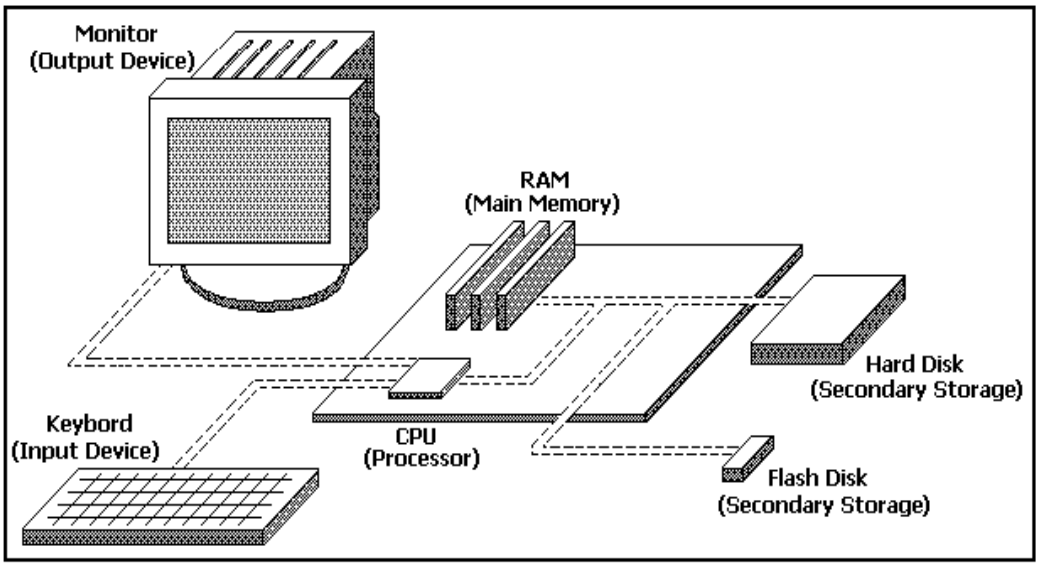
Textbook

1. Problem solving with c++ by Walter Savitch, 7th edition,2009.
2. C++: The Complete Reference by Herbert Schildt, 4th edition, 2003.

Reference

1. A first book of c++ by Gary Bronson, 4th edition, 2012 by Gary Bronson

Introduction



- **Computer:** is a device capable of performing computations and making logical decisions at speeds millions and even billions of times faster than human beings.
- **Programming** is the process of writing instructions for a computer in a certain order to solve a problem.
- The computer programs that run on a computer are referred to as **software**. While the hard component of it is called **Hardware**.

Problem solving

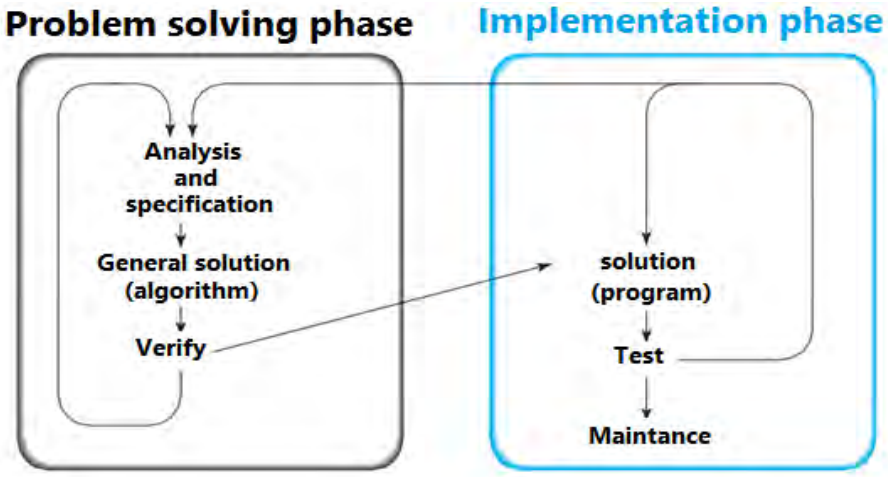
Levels of abstraction

- **Problem:** A question raised for solution.
- **Solving:** finding a solution for something.

So, the problem solving is the act of finding a solution to problem

- The result of the **problem solving** is an algorithm, expressed in English.
- To produce a program in a programming language such as C++, the algorithm is translated into the programming language.

Program Design Process



Problem Solving Phase:

- **Analysis and Specification:** Understand (define) the problem and what the solution must do.
- **General Solution (Algorithm):** Specify the required data types and the logical sequences of steps that solve the problem.
- **Verify:** Follow the steps exactly to see if the solution really does solve the problem.

Implementation Phase:

- **Solution (Program):** Translate the algorithm into a programming language.
- **Test:** Manually check the results. If you find errors, analyze the program and the algorithm to determine the source of the errors, and then make corrections.
- **Maintenance phase:** Modify the program to meet changing requirements or to correct any errors that show up while using it.

***Example*: Find value of the variable output of the equation: $Z = (x-y)^2$**

Analysis and Specification:

1- Understand the question: is the account (حساب(value of the variable Z , therefore must determine the inputs are x and y, and then finding exact value of the variable Z previous equation.

2- Analysis stage: a review of the different ways to resolve and choose the most suitable in terms of speed, ease and accuracy.

General solution:

First way: value of the variable Z is calculate equation Z=(x-y)^2

1. **set value of each variable x and y**
2. **find output x-y**
3. **finding value of variable Z by square of step 2**

Second way: value of the variable Z is calculate equation `Z = x**2 – 2 *x * y + y **2`

1. **Compensation value of each variable x and y**
2. **Find square of x (x** 2)**
3. **Find value of 2 * x * y**
4. **Find the square of y (y ** 2)**
5. **Find subtraction the result of step3 from step 2**
6. **Finding the value of Z by addition step5 value with step4 value**

Analyzed the previous two methods it is clear that the first faster, easier and more accuracy to reach the solution.

Algorithms

What are algorithms

- Algorithm **,** It called that name in relation to the Muslim world **Abu Ja'far Muhammad ibn Musa al-Khwarizmi**
- In the science of algorithms no fixed rules algorithms to represent the algorithm in this way, but there are some controls (ضوابط) that must be taken into account الاعتبار بعين during the representation, and are:
- not matter(لايهم) use of any type of human languages (Arabic, English, French, ...).
- preferably used words as easy as possible and clear.
- It must be consists of only three structures : sequence, choice, repetition
- Stay away (بعد) from the use of words have meaning is limited to a specific programming language.

Algorithm Definition

An algorithm can be defined as a finite sequence of effect statements to solve a problem. An effective statement is a clear instruction that can be carried out(قابلة للتنفيذ).

Algorithm Properties:

- **Finiteness:** The algorithm must terminate a finite numbers of steps.
- **Non-ambiguity:** Each step must be clearly defined.
- **Effectiveness:** The algorithm should solve the problem in a reasonable amount of time.
- **Algorithm Language:** not matter use of any type of human languages (Arabic, English, French, ...).

Why we need the Algorithms

- **Documentation of thinking in order to solve problems code.**
- **Determine the time and storage space that the computer needs to resolve the problem.**
- **Contribute (لهس امهمة) to speed the discovery of errors before you start thinking in practical application stage.**
- **Give us the opportunity (فرصة) to solve problems in different ways.**

Algorithm Representation Ways

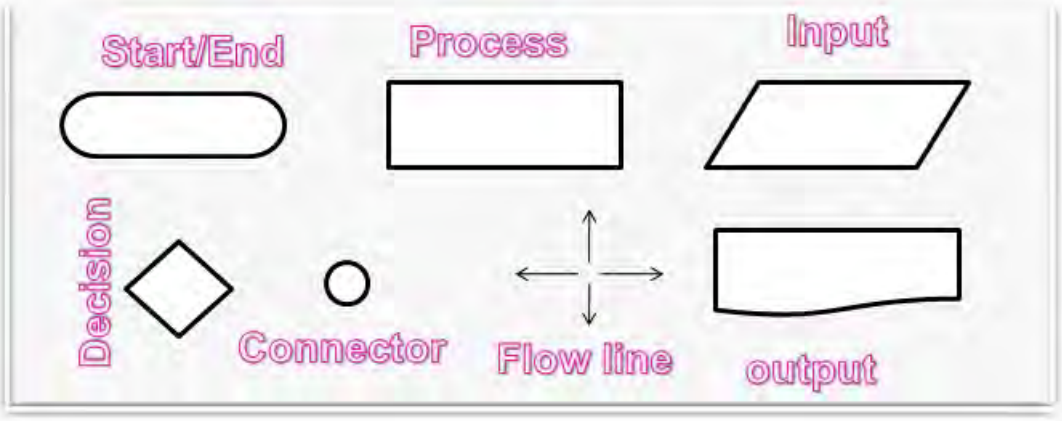
The Algorithm Representation in many ways like:

- **Natural language**
- **Pseudo code**
- **Flow Chart**

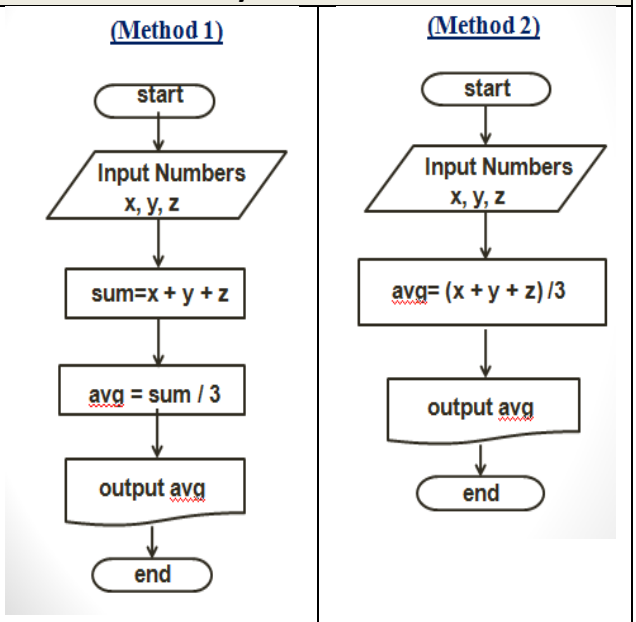
Natural language	Pseudo code	Flow Chart
1. Is way directly to express the solution by sentences and phrases natural languages: English , Arabic, .. 2. Differ from person to person	1. A clever way to represent algorithm 2. Similar to human language not considered a programming language. 3. Easily converted for different programming languages.	1. Symbolic representation to the algorithm. 2. Do not need to express your own language 3. It a lot easier

Flowcharts

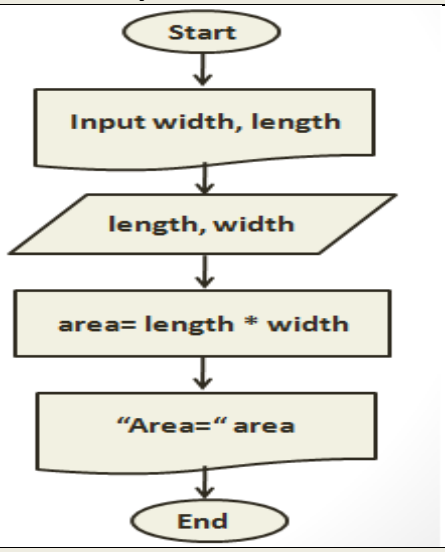
A flowchart is a graphical representation of an algorithm. Flowcharts are drawn using symbols. The main symbols used to draw a flowchart are shown in the following:



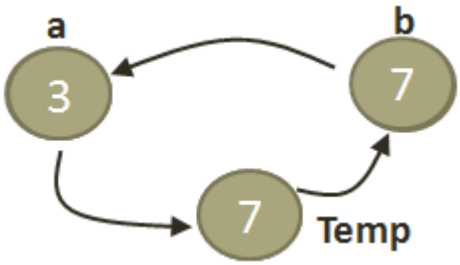
Example 1: ***Write an algorithm that Find the average of the three numbers.***

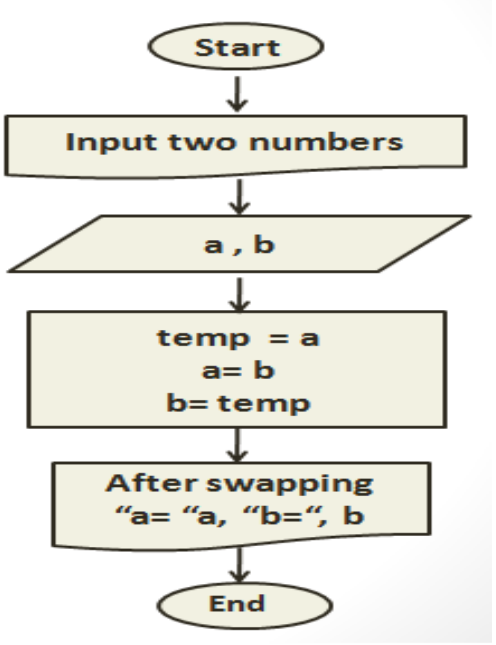
By Natural language	By Pseudo code	By Flow Chart
1. Start. **2.** Read the three numbers. **3.** Calculate the sum of the three numbers. **4.** Calculate the average by divide the sum by three. **5.** Output the average **6.** End.	**Method1** **1.** start. **2.** input x, y and z. **3.** sum= x + y + z. **4.** avg= sum /3. **5.** output avg. **6.** end. **Method2** **1.** start. **2.** input x, y and z. **3.** avg= (x + y + z) / 3. **4.** output avg. **5.** end.	

Example 2: ***Write an algorithm that outputs the rectangle area given width and length.***

By Pseudo code	By Flow Chart
1. Start **2.** output " Input width and length -> " **3.** input length ,width **4.** area = length* width **5.** output "Area = " , area **6.** end	

Example 3: *Write an algorithm that can swapping between two inputs variables.*

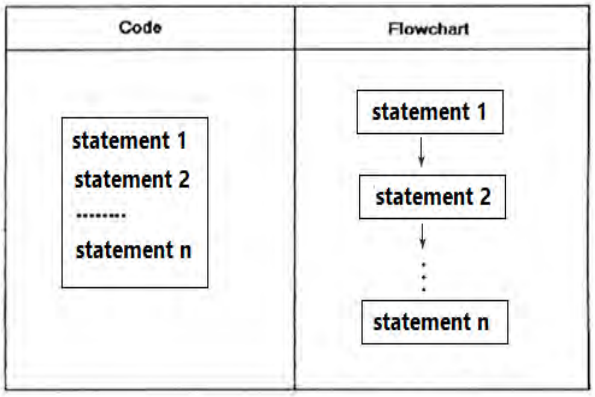


By Pseudo code	By Flow Chart
1. start **2.** output " input two numbers :- " **3.** input a , b **4.** temp = a **5.** a = b **6.** b = temp **7.** output " After swapping " **8.** output "a= ", a, "b= ", b **9.** end	

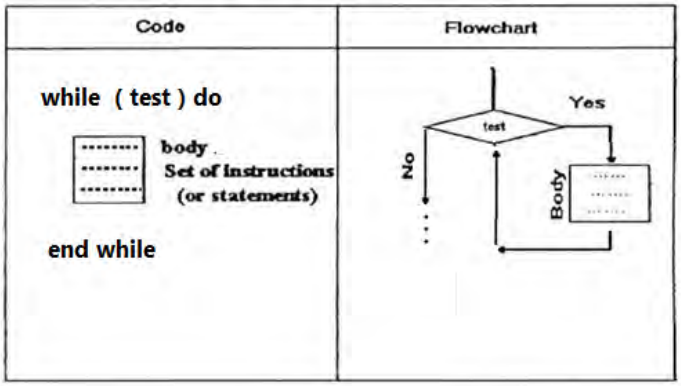
Control Structure

Any algorithm can be written using only three structures (together, individual), **Sequence, choice, repetition**, the following table describe thee representation of these structure by flowchart .

Sequence

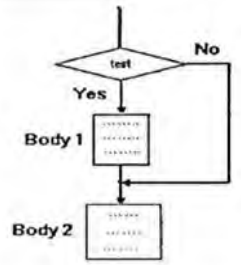


While loop structure



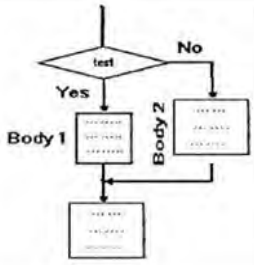
IF structure

Code	Flowchart

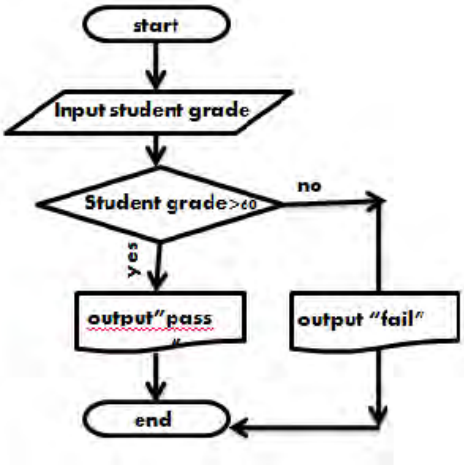
IF (test) then
body 1
Set of instructions
(or statements)
end if
body 2
	

IF - Else structure

Code	Flowchart

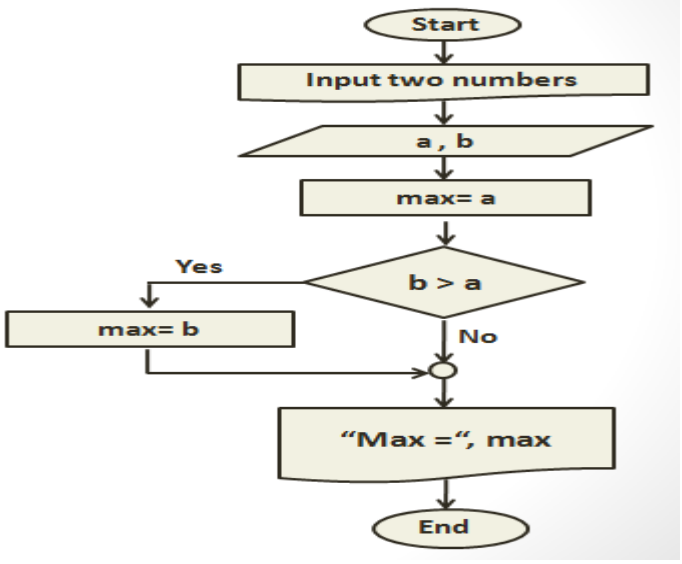
IF (test) then
body 1
Set of instructions
(or statements)
Else
body 2
Set of instructions
(or statements)
end if
	

Example 4: Write algorithm to print "pass" if student grade greater than 60 otherwise prints "fail"

By Natural language	By Pseudo code	By Flow Chart
1. start **2.** input student grade **3.** If students grade is greater than 60 Output "passed" else output "failed" **4.** end	**1.**start **2.** input grade **3.** if grade > 60 then Output "passed" Else Output "failed" **4.** end	

Example 5: *Write an algorithm that can find the large number between two inputs numbers.*

Answer:

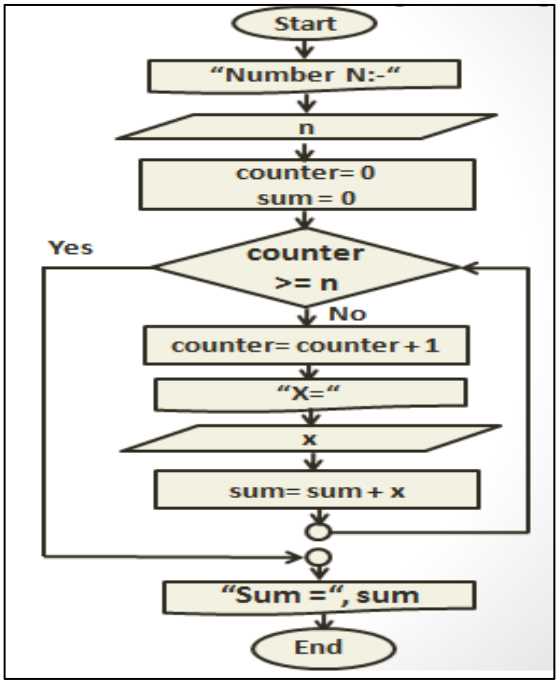
By Pseudo code	By Flow Chart
1. start **2.** output " input two numbers :- " **3.** input a , b **4.** max = a **5.** if b > a then max=b **6.** output " Max= ", max **7.** end	

Example 6: *Write an algorithm can find summation of N numbers.*

Answer: **By Pseudo code**

1. start
2. output " Number N :- "
3. input n
4. counter = 0 , sum= 0
5. if counter >= n then goto Step 11
6. counter = counter + 1
7. output " X= “
8. input x
9. sum = sum + x
10. goto Step5
11. output “Sum= “, sum
12. end

by flowchart



Home work:

1. Write an algorithm that can calculate the number of students who succeed in computer programming I exam.
2. Write an algorithm can calculate the following question:
   z= x+y+2.
3. Write an algorithm can calculate the following question:

$$Z = \begin{cases} x + 5 & x \geq 0 \\ x - y + 2 & x < 0 \end{cases}$$

4. Write an algorithm can calculate t he following question:

$$Z = \begin{cases} x + 5 & x > 0 \\ 7 & x = 0 \\ x * y & x < 0 \end{cases}$$

5. Write an algorithm to input a number X which represents the number of seconds. Then output the number of days, hours, minutes and seconds. For example, if X = 105733 Days = 1, Hours = 5, Minutes = 22 and seconds = 13.

6. Write an algorithm to compute and outputs the value of y:

$$y = x^2 * 10 - 2$$

7. Draw flowchart to the Written Algorithm that determine if a number is positive or negative. If positive output "POSITIVE", else output "NEGATIVE".

step1: start
step2: output " input number -> "
step3: input x
step4: if (X >= 0) then
output "POSITIVE"
else
output "NEGATIVE"
step5: end

8. Draw flowchart to the Written Algorithm that determine if the number is zero then output "ZERO"

step1: start
step2: output " input numbers -> "
step3: input x
step4: if (x > 0) then
output "POSITIVE"
else if (x = 0) then
output "ZERO"
else
output "NEGATIVE"
step5: end

9. Write an algorithm that inputs a student's score outputs his grade according to the following:

Score	Grade
0 – 59	F
60 - 69	D
70 – 79	C
80 – 89	B
90 – 100	A

10. Write an algorithm that inputs a number and outputs "ODD" if the number is odd and "EVEN" if the number is even. (**Note:**: Zero is even)

11. Write an algorithm to input 2 numbers and outputs the minmum number.

12. Write an algorithm can find summation of the positive and the negative from N input numbers.

13. Draw flowchart to the written Algorithm that inputs and outputs the sum of n numbers.

step1: start
step2 : output "input n ->"
step3: input n
step4: sum = 0
step5 : for (i=1; i<= n; i=i+1)
step6 : output "input number --> "
step7: input x
step8: sum = um + x
step9: end of step 5
step10: output "sum = " , sum
step11: end

14. Draw flowchart to the written Algorithm that find the average of N input numbers.

step1: start
step2 : output "input n ->"
step3: input n
step4: sum = 0
step5 : for (i=1; i<= n; i=i+1)
step6 : output "input number --> "
step7: input x
step8: sum = um + x
step9: end of step5
step10: avg = sum / n
step11: output "avg = " , avg
step12: end

15. Draw flowchart to the written Algorithm that inputs a number N and computes and outputs the value of S :

$$s = \sum_{i=1}^{n} i$$

step1: start
step2: s = 0
step3: output " input n ->"
step4: input n
step5: for (i=1;i<=n; i=i+1)
step6: s = s + i
step7: end of step 5
step8: output "sum = ", s
step9: end

16. Write an algorithm that calculate the value of S:

$$\mathbf{S} = \sum_{\mathbf{i=1}}^{\mathbf{n}} \mathbf{x}$$

17. Write an Algorithm that inputs a number N and computes and outputs the value of s:

$$s = \sum_{i=1}^{n} x^2$$

18. Write an algorithm that inputs a number TOTAL and a sequence of numbers. The algorithm stops inputting numbers when the sum of the numbers exceeds TOTAL and then outputs how many numbers were input.

19. Write an algorithm that inputs a sequence of N numbers and outputs the following:
A) Their sum.
B) Their average.
C) The number of Negative numbers, Positive numbers and Zeros.

20. Write an algorithm that inputs a sequence of numbers ending with a negative number and outputs the following: **A)** Their sum. **B)** Their average.

21. Write an algorithm that inputs a number n and outputs n!

(Hint: N! = 1*2*3*…..*(N-1)*N).

22. Write an algorithm that inputs an even positive number N and outputs the 10 even numbers following it.

23. Write an algorithm that inputs a positive number N and outputs the 10 even numbers following it. (We don't know if N is even or odd)

24. Write an Algorithm to input a number TOTAL and compute and output the value of S. The algorithm should stop when S > TOTAL. Also output how many terms of S were used:

$$s = \sum_{i=2}^{n} 1/i$$

25. Write an Algorithm that inputs a number N and computes and outputs the value of

$$s = \sum_{i=1}^{n} x^i$$

26. 14. Write an algorithm that inputs a number X and output the value of S:

$$s = 1 - x + x^2 - x^3 + x^4 - \cdots$$

A) For N terms
B) Until S > K (K should be input as input)

27. The following series is used to compute the value of s:

$$s = 4 - \frac{4}{3} + \frac{4}{5} - \frac{4}{7} + \frac{4}{9} \dots$$

28. Write an algorithm to output the value of s by:

A) Finding the value of the series after 20 terms.

B) Computing si until value of (si – si-1)<0.0001

29. Write an Algorithm that computes and outputs the value of S:

$$s = \frac{1}{2} + \frac{2}{3} + \frac{3}{4} + \frac{4}{5} + \cdots + \frac{99}{100}$$

30. Write an Algorithm that computes and outputs the value of S:

$$s = 1 - \frac{1}{3!} + \frac{1}{5!} - \frac{1}{7!} + \frac{1}{9!} \dots$$

A) For N terms **B)** Until (Sumi – Sumi-1) < 0.0001

31. Write an algorithm that computes and outputs the value of S:

$$s = \sum_{i=1}^{n} \frac{x^i}{x!}$$

32. Write an algorithm that computes and outputs the value of S:

$$s = \sum_{i=1}^{n} \frac{x^{2i}}{(i-1)!}$$

33. Write an algorithm to sums all the even numbers between 1 and 20 and then displays the sum.

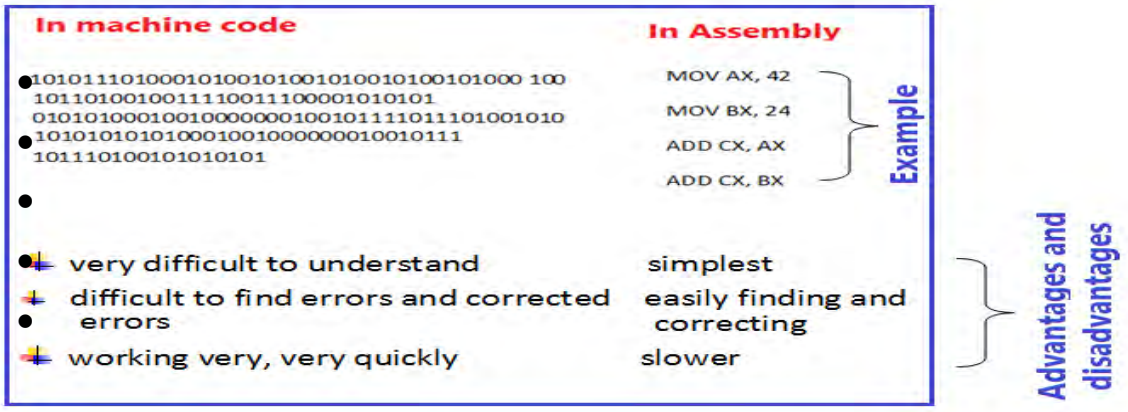
Draw flowchart for all the exercises.

What is programming?

- Programming: is the process by which to determine how to deal with the data entered into the computer for the desired results.
- Computer Programming: is the process of providing(توفير) the commands to computer to perform a specific task in a certain way .

Programming

- **Programming Language :** it's a sequence of instructions that convert An Algorithms (**in human language)** to the A program **(computer Language**) .
- Programming language can be classified into:
  - **Low Level Language** : in this type of languages the programmer can write programs must knowing the details of how the computer work, storage locations and details of the device like machine code and Assembly language, the fallowing example of adding the number (24 , 42) by these language .



- **High Level Language:** in this type of languages the programmer can write programs without the knowing details of how the computer work, storage locations and details of the device , like C++ , Java , …… , the fallowing example show part of program in C++ language .

```
void main(){
int x, y, z;
x = 1;
y = 12;
z = x + y ;}
```

Any Program in High Level language passing this stages

Program → interpreter/compiler → machine language

Difference between Compiler and Interpreter

No	Compiler	Interpreter
1	Compiler Takes **Entire** program as input	Interpreter Takes **Single** instruction as input .
2	Intermediate Object Code is **Generated**	**No** Intermediate Object Code is **Generated**
3	Conditional Control Statements are Executes **faster**	Conditional Control Statements are Executes **slower**
4	**Memory Requirement** : **More** (Since Object Code is Generated)	**Memory Requirement** is **Less**
5	Program need not be **compiled** every time	Every time higher level program is converted into lower level program
6	**Errors** are displayed after **entire program** is checked	**Errors** are displayed for **every instruction** interpreted (if any)
7	**Example** : C Compiler	**Example** : BASIC

Basic elements of C++

C++ is a general-purpose programming language. ***C++ was derived from C, and is largely based on it.***

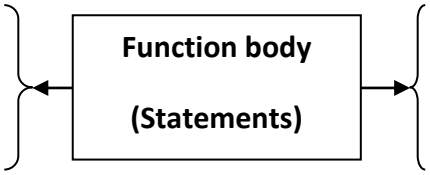
General Form of a C++ Program

Programming language is a set of rules, symbols, special words

- rules(syntax) – specifies legal instructions
- Symbols - special symbols (**+ - * !** …)
- Special Word (reserved words) (**int, float, double, char** …)
- A C++ program is a collection of one or more subprograms (functions)
- Function
  - Collection of statements
  - Statements accomplish a task
- Every C++ program must contain a function called **main**

Program structure in C++

```
int main ()
 {
   .
   .
  return 0;
 }
```




```
void  main()
  {
     .
     .
     .
  }
```

Where

- The **int** specifies that it returns an integer value
- The **void** specifies there will be no arguments

Example: Program in c++ to display "Welcome In Computer Programming Course"

```
#include <iostream.h>
 int main( )
  {
     cout   <<"   Welcome   In   Computer
                    Programming Course ";
    return 0;
  }
```

```
#include <iostream.h>
 void main( )
  {
     cout <<" Welcome In Computer
              Programming Course ";
  }
```

Program Result :

Welcome In Computer Programming Course

#include <iostream.h>	Lines beginning with a hash sign (#) are directives for the preprocessor. They are not regular code lines with expressions but indications for the compiler's preprocessor. **Preprocessor directives must be specified in their own line and do not have to end with a semicolon (;)**. In this case the directive **#include <iostream.h>** tells the preprocessor to include input-output library in C++ .
int main() - void main()	Program execution begins with the main function. The entry point of every C++ program is **main()**.
Curly brackets { }	indicate the beginning and end of a function, which can also be called the function's body. The information inside the brackets indicates what the function does when executed.
;	In C++, the semicolon is used to terminate a statement. Each statement must end with a semicolon. It indicates the end of one logical expression.
Cout	is used in combination with the insertion operator, <<, to insert the data that comes after it into the stream that comes before.
Statements	A **block** is a set of logically connected statements, surrounded by opening and closing curly braces. For example **{** **cout << "Welcome In Computer programming course ";** **return 0;** **}** You can have multiple statements on a single line, as long as you remember to end each statement with a semicolon. failing to do so will result in an error.
return 0 ;	The line **return 0;** terminates the **main()** function and causes it to return the value 0 to the calling process. A non-zero value (usually of 1) signals **abnormal** termination

Note: The program has been structured in different lines in order to be more readable, but in C++, we do not have strict rules on how to separate instructions in different lines.

Exercise: Write a program in C++ to display :

Welcome In IS Dep.
I'm a C++ course
First Stage
Group A

Comments

Comments are explanatory statements(ملاحظات توضيحية على) that you can include in the C++ code to explain what the code is doing. The compiler ignores everything that appears in the comment, so none of that information shows in the result. There are two types of comment:

Comment type	Description	Example

Single-line comment	**// line comment**	#include <iostream.h>
		
		int main( )
		{
		// print "Welcome In IS Dep. ".
		cout << "Welcome In IS Dep. ";
		return 0;
		}

Multi-Line Comments	**/* block comment */**	include <iostream.h>
		
		int main( )
		{
		/* Welcome In IS Dep */
		
		/* Example for display
		Welcome In IS Dep
		*/
		
		cout << "Welcome In IS Dep. ";
		return 0;
		}

Note: Comments can be written anywhere, and can be repeated any number of times throughout the code. Within a comment marked with /* and */, // characters have no special meaning, and vice versa. This allows you to "nest" one comment type within the other.

Reserved Words (keywords)

Reserved words have a predefined meaning in C++ and that you cannot use as names for variables or anything else.

asm	dynamic_cast	new	template
auto	else	operator	this
bool	enum	private	throw
break	extern	protected	true
case	false	public	try
catch	float	register	typedef
char	for	reinterpret_cast	typeid
class	friend	return	union
const	goto	short	unsigned
const_cast	if	signed	using
continue	inline	sizeof	virtual
default	int	static	void
delete	long	static_cast	volatile
do	mutable	struct	wchar_t
double	namespace	switch	while

Note: *Keep in mind that the case of the keywords is significant. C++ is a case-sensitive language, and it requires that all keywords be in lowercase. For example,* ***RETURN*** *will not be recognized as the keyword* ***return****.*

Identifiers

Any item might define in a program is called an ***identifier.***

Rules for identifiers

- must begin with letter or the underscore _
- followed by any combination of numerals, letters or underscore
- recommend (وصي) meaningful identifiers
- Another rule that you have to consider when inventing your own identifiers is that they cannot match any keyword of the C++ language nor your compiler's specific ones, which are ***reserved keywords (keyword)***.

Here are some correct and incorrect identifier names:

Correct	Incorrect	explain way incorrect
Count	1count	?
test23	hi!there	?
high_balance	high...balance	?
_name	_n ame	?

Note: The C++ language is a "**case sensitive**" language. That means that an identifier written in capital letters is not equivalent to another one with the same name but written in small letters. Thus, for example, the **RESULT** variable is not the same as the result variable or the **Result** variable.

Use Meaningful Names

Variable names and other names in a program should at least hint at the meaning or use of the thing they are naming. It is much easier to understand a program if the variables have meaningful **names. Contrast the following**: **x = y * z;**
With the more suggestive: **distance = speed * time;**

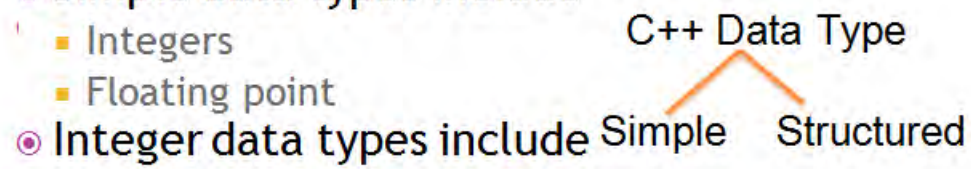
The two statements accomplish the same thing, but the second is easier to understand.

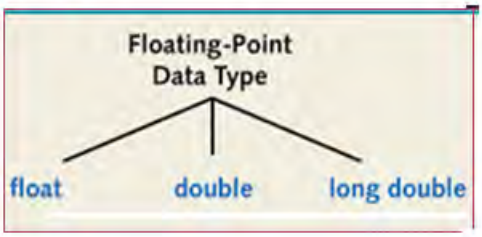
Data Types

- When programming, we store the variables in our computer's memory, but the computer has to know what kind of data we want to store in them, since it is not going to occupy the same amount of memory to store a simple number than to store a single letter or a large number, and they are not going to be interpreted the same way.
- The memory in our computers is organized in bytes. A byte is the minimum amount of memory that we can manage in C++.
- In addition, the computer can manipulate more complex data types that come from grouping several bytes, such as long numbers or non-integer numbers. In the following figure and table shown summary of the basic fundamental data types in C++, as well as the range of values that can be represented with each one:

DATA TYPES

- Simple data types include
  - Integers
  - Floating point
- Integer data types include
  - char ← Numerals, symbols
  - short, int, long ← Numbers without decimals
  - bool ← Values true and false only
- Different floating-point types
- Note that various types will
  - have different ranges of values
  - require different amounts of memory





Data type	Size(bytes)	Range
char	1	-128 to 127
unsigned char	1	0 to 255
short	2	-32,768 to 32,767
unsigned short	2	0 to 65535
int	4	-2147483648 to +2147483647
unsigned int	4	0 to 4294967295
long	4	-2147483648 to +2147483647
Unsinged long	4	0 to 4294967295
float	4	-3.4e-38 to +3.4e-38
double	8	1.7 e-308 to 1.7 e+308
long double	8	1.7 e-308 to 1.7 e+308
bool	1 bit	
void	-	-
wchar_t	2 or 4	1 wide character

Variables

- Programs manipulate data such as numbers and letters. C++ and most other programming languages use programming constructs known as ***variables*** to name and store data.
- Creating a **variable** reserves a memory location, or a space in memory for storing values. The compiler requires that you provide a **data type** for each variable you declare.
- Variables like small ***blackboards*** , can be written and then can be changed.
- The number or other type of data held in a variable is called its **value .**
- All integer, floating-point, and other values used in a program are stored in and retrieved from the computer's memory. Conceptually, locations in memory are arranged like the

rooms in a large hotel, and each memory location has a ***unique address***, like room numbers in a hotel .

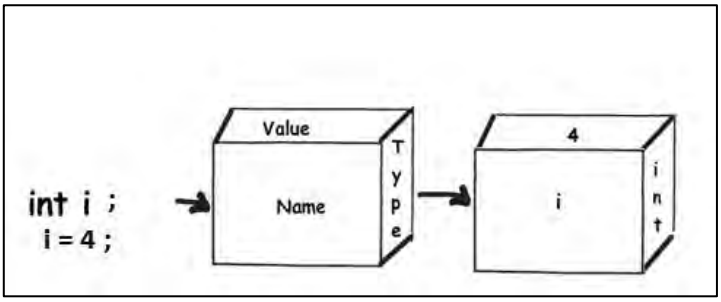
Variable Declarations

All variables must be declared before they are used. The syntax for variable declarations is as follows:

Syntax **Type_name variable_name1 , variable_name2, …… ;**

Example int count , number_of_students ;
double distance ;

Where:
type_name : must be a valid data type
Variable_Name_1, Variable_Name_2, . . . ; or (variable_list) : may consist of one or more Identifier names separated by commas (,). **Each Variable name** must follow the rules of identifier name.



Example for variable declaration using some of data type

Integer declaration : int x ; long int y ; short int z;	**Floating Point Numbers declaration:** float x1 ; double y1; long double z1;
Character declaration : char ch ;	**Boolean declaration :** bool b1 ;// true or flase

Exercise:

valid variable declaration	Not valid	Explain way not valid ?
int a, b, c;	int a; b,c ;	?
int a; int b; int c;	int a; int b , int c;	?

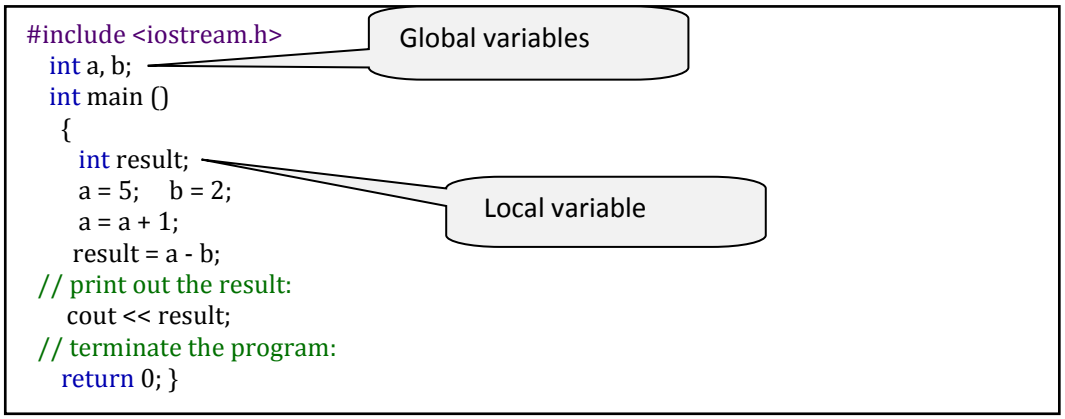
Example : To see what variable declarations look like in action within a program, we are going to see the C++ code of the example about your mental memory :

// operating with variables
#include <iostream.h>
int main ()
{
// declaring variables:
int a, b;
int result;
// process:
a = 5; b = 2;
a = a + 1;
result = a - b;
// print out the result:
cout << result;
// terminate the program:
return 0;
}
	Explain the execution steps of the program :

Scope of variables

A variable can be either of

- **Global** : A global variable is a variable declared in the main body of the source code, outside all functions.
- **Local**: while a local variable is one declared within the body of a function or a block.



Note: *The* ***scope*** *of* ***local variables*** *is* ***limited to the block*** *enclosed in braces (****{}****) where they are declared. T*his means that if another function existed in addition to main, the local variables declared in main could not be accessed from the other function and vice versa.

Initialization of variables

When declaring local variable, its value is by default undetermined. But you may want a variable to store a value at the same moment that it is declared. In order to do that, you can initialize the variable.

The syntax for initialization variables is as follows:

type identifier = intial_value;

vaild examples	**Not valid examples**	**Way ???**

int a= 5 ;
int a=b=c= 0 ;
int a=5 , d, f=8;
	int a=5 ; b= 5;
	int a= b=0, int c= 0;
	int a=5 , d; f=8;	

// initialization of variables
#include <iostream.h>
int main ()
{
int a=5; // initial value = 5
int b=2; // initial value = 2
int result; // initial value
a = a + 3; result = a - b;
cout << result;
return 0;
}
	Result:???

Constants

Constants refer to fixed values that the program cannot alter. Constants can be of any of the basic data types. The way each constant is represented depends upon its ***type***. Constants are also called ***literals***.

Type	examples
Integer Numerals	1776 707 -273
Floating-Point Numerals	3.14159 6.02e23 // 6.02 x 10^23 1.6e-19 // 1.6 x 10^-19 3.0
Characters	'z' 'p'
Strings	"Hello world" "How do you do?"
Bool	There are only two valid Boolean values: **true** and **false**.

Declared constants (const)

With the const prefix you can declare constants with a specific type in the same way as you would do with a variable *except that their values cannot be modified after their definition* :

```
const int pathwidth = 100;
const float pi=3.14 ;
```

Defined constants (#define)

You can define your own names for constants that you use by using the #define Preprocessor directive. Its Format is:

 # define identifer value

Example :

```
// defined constants: calculate circumference

#include <iostream.h>

#define PI 3.14159
#define NEWLINE  '\n'

int main ()
{
   double r=5.0;             // radius
   double circle;

  circle = 2 * PI * r;
  cout << circle;
  cout << NEWLINE;

 return 0;
}
```

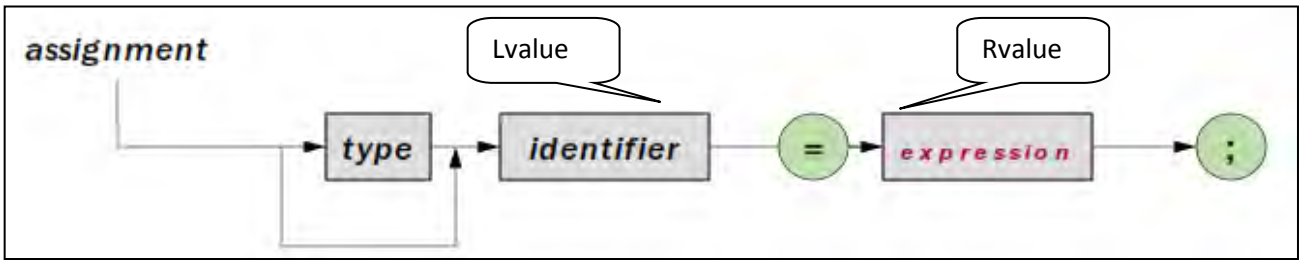
result :

```
31.4159
```

Operators

Once we know of the existence of variables and constants, we can begin to operate with them. For that purpose, C++ integrates operators.

Assignment (=) **:** The assignment operator (assignment statement) assigns a value to a variable. Its general syntax as follow :



- The **lvalue** has to be **a variable** whereas the **rvalue** can be either **a constant**, **a variable**, the result of an operation or any combination of these (expression) .

- The most important rule when assigning is the ***right-to-left* rule:** The assignment operation always takes place from **right to left,** and never the other way.

- Arithmetic expression is **any combination** of ***simple value***, **function call**, **binary expression**, **and unary expression.**

Where :

- ***Simple value :*** *constant number , string constant , character constant , identifier.*
- **Unary operator (العواهل الاحادية) are (+, - ,-- , ++).**

Example1 **(arithmetic expression)**

1. **double a= 10 + y / 5 – m1;**
2. **double a= (x + 2) / y * 5 * 9;**
3. **int a= y*10+(2-1);**
4. **int z ; z= sin(45) * 34 ;**
5. **float m*= m + x ++ ;**

Example2

// assignment operator	Result: ? ?
#include <iostream.h> **int main ( )** **{** **int a, b;** **a = 10;** **b = 4;** **a = b;** **b = 7;** **cout << "a:";** **cout << a;** **cout << " b:";** **cout << b;** **return 0;** **}**	

- the assignment operation can be used as the **rvalue** (or part of an rvalue) for another assignment operation. For example: **a = 2 + (b = 5);** is equivalent to:

```
b = 5;
a = 2 + b;
```

- The following expression is also valid in C++: **a = b = c = 5;** It assigns 5 to the all the three variables: a, b and c.

Arithmetic operators (+, -, *, /, %)

The five arithmetical operations supported by the C++ language are:

+	**Addition**
-	**Subtraction**
*****	**Multiplication**
/	**Division**
%	**Modulo** **(remainder of an integer division)**

- Arithmetic Operators require two variables to be evaluated.
- Modulo is the operation that gives the remainder of a division of two values.
- **For example**, if we write: **a = 11 % 3; // a=2**

Compound assignment (+=, -=, *=, /=, %=)

When we want to modify the value of a variable by performing an operation on the value currently stored in that variable we can use compound assignment operators:

Expression	Is equivalent to
x += y ;	x = x + y;
a-= 5;	a = a - 5 ;
price *=units + 1 ;	price =price * (units + 1) ;
a /= b;	a = a / b;
c %= 2;	C = c % 2;

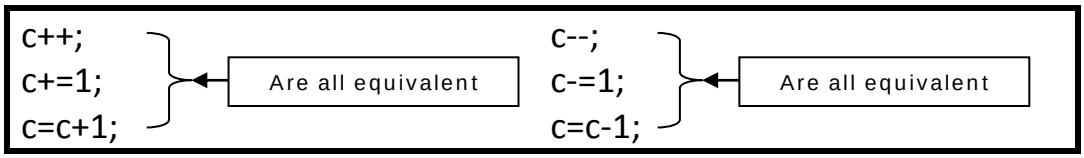
Example :

```
#include <iostream.h>
int main ()
{
    int a, b=3;
    a = b;
    a+=2;
   cout << a;
   return 0;
 }
```

Result: ??

Increase and decrease (++, --)

Shortening even more some expressions, the increase operator (**++**) and the decrease operator (**--**) increase or reduce by one the value stored in a variable (its unary operation) . They are equivalent to +=1 and to -=1, respectively. Thus:



Note:

- A characteristic of this operator is that it can be used both as a **prefix** and as a **suffix**. That means that it can be written either before the variable identifier **(++a**) or after it (**a++).**

- Although in simple expressions like a++ or ++a both have exactly the same meaning, in other expressions in which the result of the increase or decrease operation is evaluated as a value in an outer expression they may have an important difference in their meaning:
  - **In the case** that the increase operator is used as **a prefix (++a)** the value is increased before the result of the expression is evaluated and therefore the increased value is considered in the outer expression.
  - **in case** that it is used as a **suffix (a++)** the value stored in a is increased after being evaluated and therefore the value stored before the increase operation is evaluated in the outer expression. Notice the difference:

Example 1	Example 2
B=3; A=++B; // A contains 4, B contains 4	B=3; A=B++; // A contains 3, B contains 4

Note::

In Example 1, B is increased before its value is copied to A.

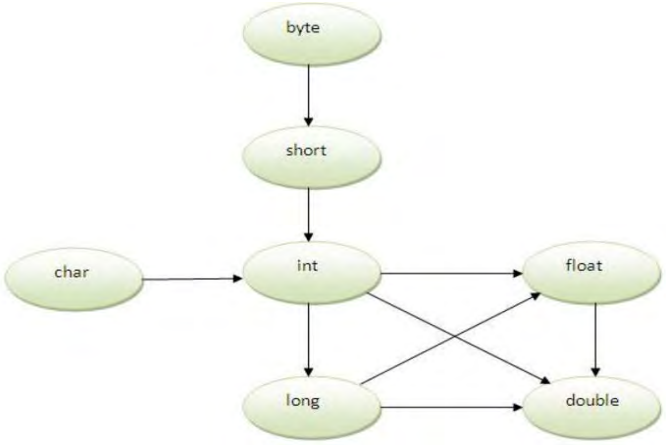
In Example 2, the value of B is copied to A and then B is increased.

Other Examples

```
a = 1;          // a = 1
b = ++a;        // a = 2, b = 2
c = a++;        // a = 3, c = 2

a = 5; b = 3;
n = ++a + b--; // a = 6, b = 2, n = 9

a = 5; b = 3;
n = ++a * ++b; // a = 6, b = 4, n = 24
n = a++ * b++; // a = 6, b = 4, n = 15

a = 5; b = 3;
n = a++ * --b; // a = 6, b = 2, n = 10
```

Automatic type conversion

If an expression contains operands of different types, an (to the type which is highest in the following hierarchy) is performed. ***Automatic type conversion***



Some of binary operation yield implicit type conversions

Integer / integer	= integer	► 39/7=5
Integer / float	= float	► 39/7.0 =5.57
float / integer	= float	► 39.0/7 =5.57
float / float	= float	► 39.0/7.0=5.57

while 39%5=7, since 39=7*5+4

Relational and equality operators (==, !=, >, <, >=, <=)

In the term *relational operator*, relational refers to the relationships that values can have with one another. In the term ***logical operator***, logical refers to the ways these relationships can be connected. We can use the relational and equality operators. The result of a relational operation is a Boolean value that can only be true or false, according to its Boolean result.

Note: C++ fully supports the zero/non-zero concept of **true** and **false**.

However, it also defines the **bool** data type and the Boolean constants **true** and **false**.

==	Equal to
!=	Not equal to
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to

Here there are some examples:	Of course, instead of using only numeric constants, we can use any valid expression, including variables. **Suppose that a=2, b=3 and c=6,**

(7 == 5) // evaluates to false.
(5 > 4) // evaluates to true.
(3 != 2) // evaluates to true.
(6 >= 6) // evaluates to true.
(5 < 5) // evaluates to false.
	(a == 5) // evaluates to false since a is not equal to 5.
	(a*b >= c) // evaluates to true since (2*3 >= 6) is true.
	(b+4 > a*c) // evaluates to false since (3+4 > 2*6) is false.
	((b=2) == a) // evaluates to true.

Logical operators (! , && , ||)

The Operator ! : is the C++ operator to perform the Boolean operation NOT, it has only one operand, located at its right, and the only thing that it does is to inverse the value of it, producing false if its operand is true and **true** if its operand is **false**. ***Basically, it returns the opposite Boolean value of evaluating its operand.***

NOT (!) Table:

A	!A
T	F
F	T

NOT (!) Table:

A	!A
1	0
0	1

Example

```
!(5 == 5)   // evaluates to false because the expression at its right (5 == 5) is true.
!(6 <= 4)   // evaluates to true because (6 <= 4) would be false.
!true       // evaluates to false
!false      // evaluates to true.
```

The logical operators && and || : are used when evaluating two expressions to obtain a single relational result. The operator **&&** corresponds with Boolean logical operation **AND**. This operation results true if both its two operands are **true**, and **false** otherwise.

AND (&&) Table:

A	B	A && B
T	T	T
T	F	F
F	T	F
F	F	F

AND (&&) Table:

A	B	A && B
1	1	1
1	0	0
0	1	0
0	0	0

The operator ||: corresponds with Boolean logical operation **OR**. This operation results true if either one of its two operands is **true**, thus being false only when both operands are false themselves.

OR (| |) Table:		
A	B	A | | B
T	T	T
T	F	T
F	T	T
F	F	F

OR (| |) Table:		
A	B	A | | B
1	1	1
1	0	1
0	1	1
0	0	0

Example:

```
( (5 == 5) && (3 > 6) )  // evaluates to false ( true && false ).
( (5 == 5) || (3 > 6) )  // evaluates to true ( true || false ).
```

Conditional operator (?)

The conditional operator evaluates an expression returning a value if that expression is true and a different one if the expression is evaluated as false.
Conditional operator which can be used to replace **if...else** statement.
Its format is:

condition? result1 : result2

If condition is true the expression will return result1, if it is not it will return result2.

Example:

```
(7==5 )? 4 : 3     // returns 3, since 7 is not equal to 5.
7==5+2 ? 4 : 3   // returns 4, since 7 is equal to 5+2.
5>3 ? a : b      // returns the value of a, since 5 is greater than 3.
a>b ? a : b      // returns whichever is greater, a or b.
```

```
// conditional operator

#include <iostream>

int main ()
 {
   int a,b;
   a=10;
    b= (a == 1) ? 20: 30;

    cout << b;
  return 0;
 }
```

Result ?

Precedence of operators

When writing complex expressions with several operands, we may have some doubts about which operand is evaluated first and which later. For example, in this expression:

a = 5 + 7 % 2

We may doubt (أشك) if it really means:

a = 5 + (7 % 2) // with a result of 6, or
a = (5 + 7) % 2 // with a result of 0

The correct answer is the first of the two expressions, with a result of 6. There is an established order with the priority of each operator, and not only the arithmetic ones (those whose preference come from mathematics) but for all the operators which can appear in C++. From greatest to lowest priority, the priority order is as follows:

Operator	Description
* / %	Multiplication/division/modulus
+ -	Addition/subtraction

High ⇩ Low

Operator	Description
!	Logical not
&&	Logical AND
||	Logical OR

All these precedence levels for operators can be manipulated or become more legible by removing possible ambiguities using parentheses signs (and), as in this example:

a = 5 + 7 % 2; *can be written either as* a = 5 + (7 % 2); *or* a = (5 + 7) % 2;

Examples

1. **x = 3 + 4 + 5;**
2. **z *= ++y + 5;**
3. **a || b && c || d;**