Al-Ameen University

College of Engineering

Department Of

CYBERSECURITY Engineering

Lecture (   )

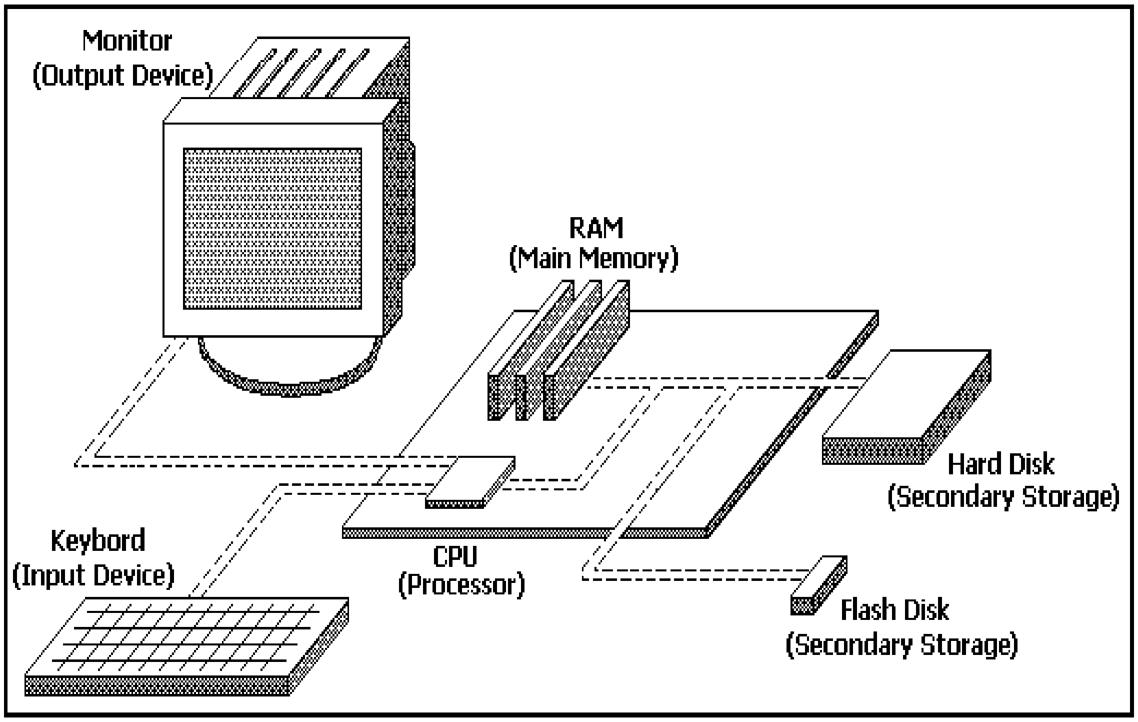
Structured Programming

2024

MEng: Sajjad Al-Mukhtar

LECTURE 1

1. Introduction:



hard ware components

Computer is a device capable of performing computations and making logical decisions at speeds millions and even billions of times faster than human beings.

Computers process data under the control of sets of instructions called **computer programs.**

Programming is the process of writing instructions for a computer in a certain order to solve a problem.

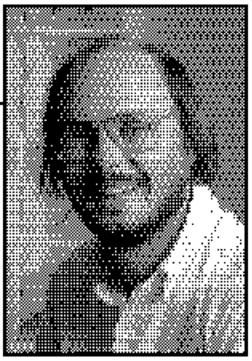
The computer programs that run on a computer are referred to as **software (SW).** While the hard component of it is called **hardware** (HW).

Developing new software requires written lists of instructions for a computer to execute. Programmers rarely write in the **langauage** directly understood by a computer.

2. Short History:

The following is a short history, just for given a general view of how languages are arrived:

- 1954: Fortran.
- 1957: Cobol.
- 1958: Algol (*Base for Simula*).
- 1958: Lisp.
- 1961: B1000.
- 1962: Sketchpad.
- 1964: Basic.
- 1967: Simula67.
- 1968: FLEX.
- 1970: Pascal (*From Algol*).
- ***1971: C*** *(From a language called B)*.
- 1972: Smalltalk72 (*Based on Simula67 and Lisp*).
- 1976: Smalltalk76.
- 1979: ADA (*From Pascal*).
- ***1980: C with classes*** *(experimental version)*.
- **1983: C++ (by Bjarne Stroustrup).**
- 1986: Objective-C (*from C and Smalltalk*).
- 1986: Eiffel (*from Simula*).
- 1991: Sather (*From Eiffel*).
- 1991: Java.
- 2000: C#.


Bjarne Stroustrup
at: AT&T Labs

3. C++ Programming Language:

For the last couple of decades, the C programming language has been widely accepted for all applications, and is perhaps the most powerful of structured programming languages. Now, C++ has the status of a structured programming language with object oriented programming (OOP).

C++ has become quite popular due to the following reasons:

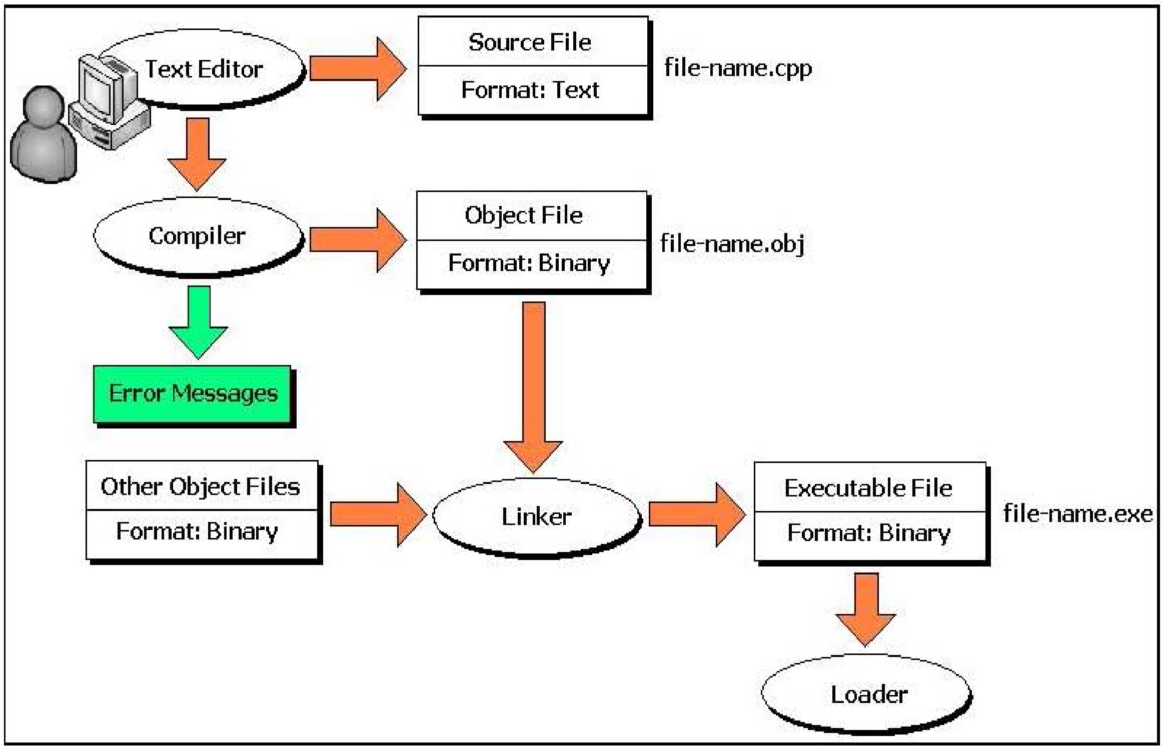
1. It supports all features of both structured programming and OOP.
2. C++ focuses on function and class templates for handling data types.

4. C++ Program Development Process (PDP):

C++ programs typically go through six phases before they can be executed. These phases are:

1. **Edit**: The programmer types a C++ source program, and makes correction, if necessary. Then file is stored in disk with extension (.cpp).
2. **Pre-Processor**: Pre-processing is accomplished by the pre-proceccor before compilation, which includes some substitution of files and other directories to be include with the source file.
3. **Compilation**: Converting the source program into object-code.
4. **Linking**: A linker combines the original code with library functions to produce an executable code.
5. **Loading**: The loader loads the program from the disk into memory.
6. **CPU**: Executes the program, residing in memory.

These steps are introduced in the figure below:


Text Editor
Source File
Format: Text
file-name.cpp
Compiler
Object File
Format: Binary
file-name.obj
Error Messages
Other Object Files
Format: Binary
Linker
Executable File
Format: Binary
file-name.exe
Loader

LECTURE 2

1. Algorithm:

As stated earlier an algorithm can be defined as a finite sequence of effect statements to solve a problem. An effective statement is a clear, unambiguous instruction that can be carried out .Thus an algorithm should special the action to be executed and the order in which these actions are to be executed.

Algorithm properties:

- **Finiteness:** the algorithm must terminate a finite number of steps.
- **Non-ambiguity:** each step must be precisely defined. At the completion of each step, the nest step should be uniquely determined.
- **Effectiveness:** the algorithm should solve the problem in a reasonable amount of time.

Example 1: Develop an algorithm that inputs a series of number and output their average .

A computer algorithm can only carry out simple instruction like:

- "Read a number".
- "Add a number to anther number".
- "Output a number".

Thus an algorithm is:

1. Carry out initialization required.
2. Read first number.
3. While the number of numbers is not complete do
4. begin
5. Add the number to the accumulated sum.
6. increment the count of numbers entered.

7. Read next number.
8. End
9. Evaluate the average.

<u>Example 2:</u> **Devolve an algorithm that allows the user to enter the count of numbers in a list followed by these numbers. The algorithm should find and output the minimum and the maximum numbers in the list.**

An algorithm for this might be:

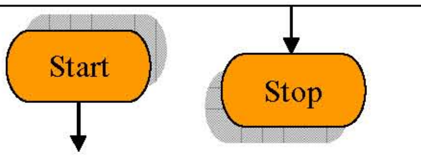
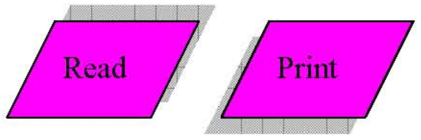
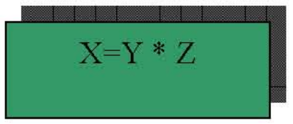
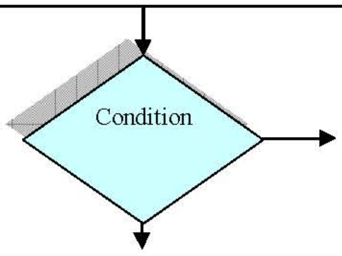
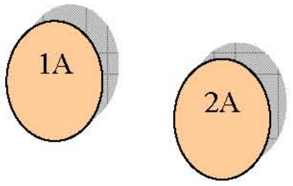
- Initialize.
- Get count of numbers.
- Enter numbers and find maximum and minimum .
- Output result.

The user might enter zero for the count. To deal with this case the above general case can be extended as follows to be an algorithm:

1. Initialize the require variables.
2. Get count of numbers.
3. If count is zero then exit.
4. Otherwise begin.
5. Enter numbers.
6. Find max and min.
7. Output result.
8. End.

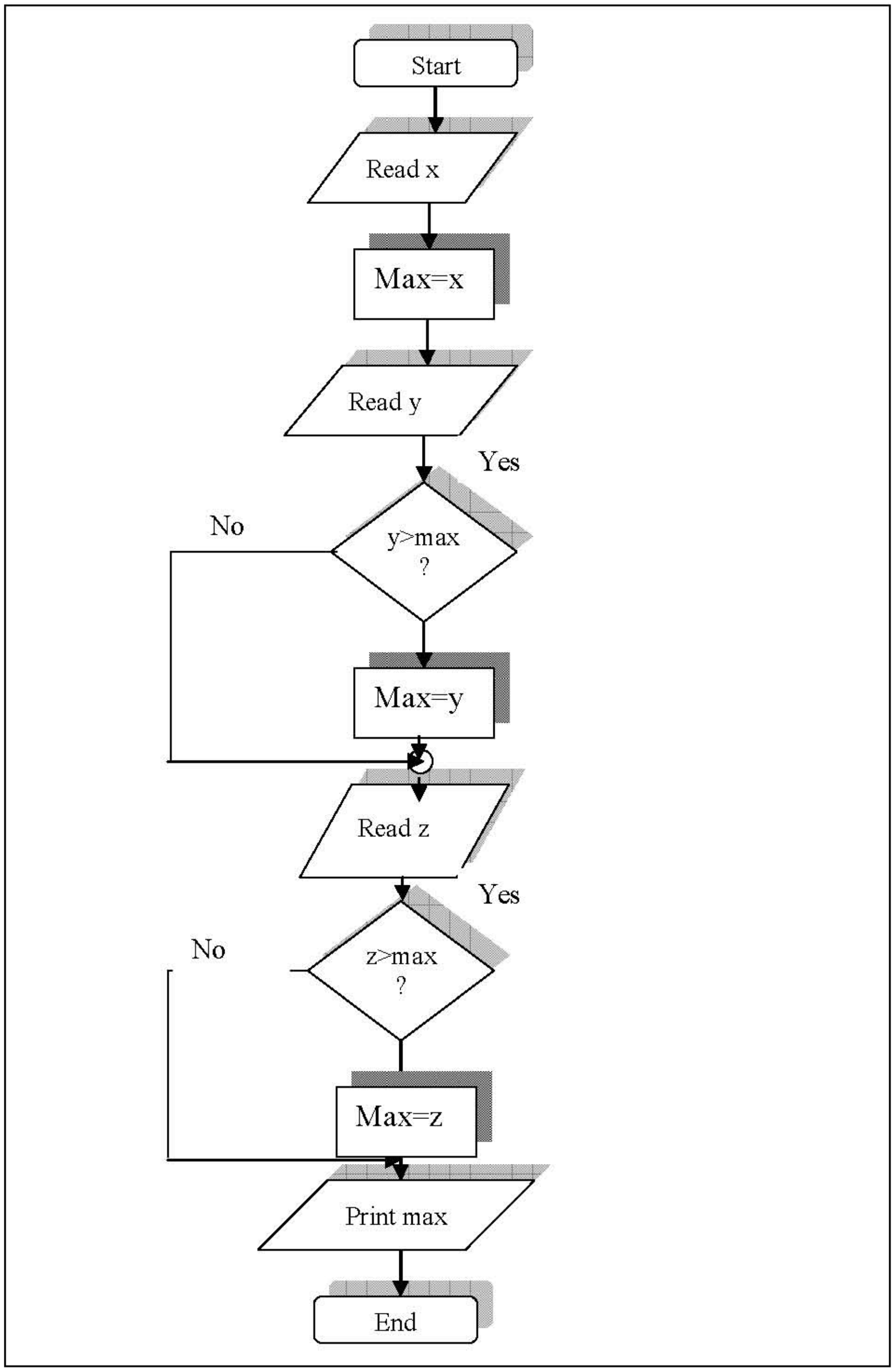
2. Flowcharts

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

	Start and Stop Symbols
	Input and Output Symbols
	Mathematical and logical processing symbol
	Decision making symbol
	Connector symbols

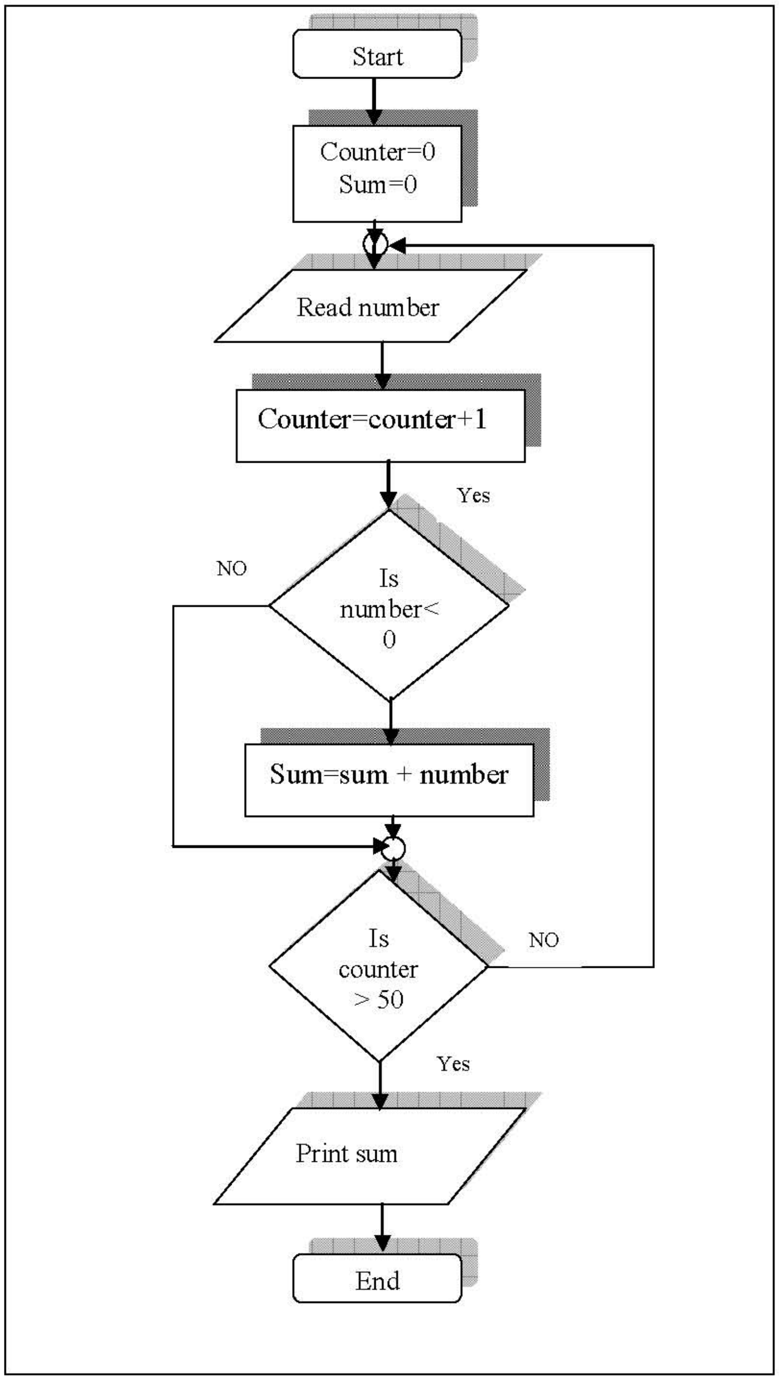
Example 1:

Draw a flowchart to read 3 numbers: x , y and z and print the largest number of them.



Example 2:

Draw the flowchart required to find the sum of negative numbers among 50 numbers entered by the user.



WORK SHEET (1)

AN INTRODUCTION

Q1: What do you means by program?

Q2: Why C++ language becomes quite popular?

Q3: Talk briefly about C++ program development process ?

Q4: Write an algorithm and flowcharts for the following:

a. Sum the even numbers for n numbers.
b. Display numbersfrom 0 to 10.
c. The multiplication of 10 numbers.

Lecture 3

1 Character set:

C++ has the letters and digits, as show below:

Uppercase:	A, B, C, . . ., Z
Lowercase:	a, b, c, . . ., z
Digits:	0, 1, 2, . . .,9

Special Characters: All characters other than listed treated as special characters for example:

+	-	*	/	^
(	[	{	}	]
)	<	=	>	, (Comma)
" (Double Conations)	. (Dot)	: (Colon)	; (Semicolon)	␣ (Blank Space)

In C++ language, upper case and lower case letters are distinct and hence there are 52 letters in all. For example **bag** is different from **Bag** which is different from **BAG**.

2 Identifiers:

An **identifier** is a name given to some program entity, such as variable, constant, array, function, structure, or class. An identifier is a sequence of alphanumeric (alphabetic and numeric) characters, the first of which must be a letter, and can't contain spaces. The length of an identifier is machine dependent. C++ allows identifiers of up to **127 characters**.

A **variable** should not begin with a digit. C++ does not set a maximum length for an identifier. Some examples of valid identifiers are as follows:

My_name	(7 char.)
i	(1 char.)
B	(1 char.)

Examples of invalid identifiers are:

3ab a()test ros sal

3 Keywords:

The keywords are also identifiers but cannot be user defined, since they are reserved words. All the keywords should be in lower case letters. Reserved words cannot be used as variable names or constant. The following words are reserved for use as keywords:

Some of C++ Language Reserved Words:				
break	case	char	cin	cout
delete	double	else	enum	false
float	for	goto	if	int
long	main	private	public	short
sizeof	switch	true	union	void

4 Constants:

There are three types of constants: **string constants**, **numeric constants**, and **character constants**.

1. String Constants: A string constants are a sequence of alphanumeric characters enclosed in double quotation marks whose maximum length is 255 characters. In the following are examples of valid string constants: ("The result=", "RS 2000.00", "This is test program"). The invalid string constants are like: (Race, "My name, 'this').

2. Numeric Constants: Numeric constants are positive or negative numbers. There are four types of numeric constants: integer, floating point, hexadecimal, and octal.

Integer	Integer Short integer (short) Long integer (long)
Float	Single precision (float) Double precision (double) Long double
Hexa	Short hexadecimal Long hexadecimal
Unsigned	Unsigned char Unsigned integer Unsigned short integer Unsigned long integer
Octal	Short octal Long octal

(a) **Integer constants:** Do not contain decimal points: int x,y; shortint x,y; longint x,y;

- Integer data: size (16 or 32) fill in -2^{15} to 2^{15}-1 for 16 bit and -2^{31} to 2^{31}-1 for 32 bit.
- Short integer: fill in -2^{15} to 2^{15}-1.
- Long integer: fill in -2^{31} to 2^{31}-1.
- Unsigned: fill in (0 to 65635) for 16 bit and (0 to 4,294, 967, 295) for 32 bit.

(b) **Floating point constants:** Positive or negative numbers are represented in exponential form. The floating point constant consists of an optionally (signed) integer or fixed point number (the mantissa) followed by the letter E and e and an optionally signed integer (the exponent). Ex. (9010e10, 77.11E-11).

- Float 4 bytes.
- Double 8 bytes.
- Long double 12 or 16.

(c) **Hexadecimal constants:** Hexadecimal numbers are integer numbers of base 16 and their digits are 0 to 9 and A to F.

(d) <u>Octal constants:</u> Octal numbers are numbers of base 8 and their digits are 0 to 7.

3. Character Constants: A character represented within single quotes denotes a character constant, for example 'A', 'a', ':', '?', etc…

Its maximum size is 8 bit long, signed, and unsigned char are three distinct types.

```
Char x;          char x,y,z;
```

The backslash (\) is used to denote non graphic characters and other special characters for a specific operations such as:

Special Escape Code:

Escape Code	**Description**
\n	New line. Position the screen cursor to the beginning of the next line.
\t	Horizontal TAB (six spaces). Move the screen cursor to the next tab stop.
\r	Carriage return. Position the cursor to the beginning of the current line, do not advance to the next line.
\a	Alert. Produces the sound of the system bell.
\b	Back space
\	Backslash. Prints a backslash character.
\f	Form feed
\v	Vertical tab
\"	Double quote. Prints a (") character.
\o	Null character
\?	question mark
\ooo	Octal value
\xhhh	Hexadecimal value

5. C++ operators:

C++ operators	Arithmetic operators	
	Assignment operators	
	Comparison and logical operators	Relational,equality,logical
	Bit wise logical operators	
	Special operators	Unary, ternary, comma Scope, new&delete, other

1. **Arithmetic operators:** These operators require two variables to be evaluated:

   + addition **-** subtraction ***** multiplication

   / division **%** modula (remainder of an integer division)

The division result are:

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

Arithmetic operators as per precedence:

( ) for grouping the variables.

- Unary for negative number.

* / multiplication & division.

+ - addition and subtraction.

Example: X+y*X-Z, where X=5, Y=6, and Z=8.

5 + (6*5)-8 → (5+30)-8 → 35-8 → 27

2. **<u>Assignment Operators:</u>** The operatonal assignment operator has the form:

   Variable = variable operator expression;

   Ex: x=x+5; y=y*10;

   The operational assignment operator can be written in the following form:

   Variable operator = expression

   Ex: x+=5; y*=10;

   It is used to assign back to a variable, a modified value of the present holding:

=	Assign right hand side (RHS) value to the left hand side (LHS).
+=	Value of LHS var. will be added to the value of RHS and assign it back to the var. in LHS.
-=	Value of RHS var. will be subtracted to the value of LHS and assign it back to the var. in LHS.
***=**	Value of LHS var. will be multiplied to the value of RHS and assign it back to the var. in LHS.
/=	Value of LHS var. will be divided to the value of RHS and assign it back to the var. in LHS.
%=	The remainder will be stored back to the LHS after integer division is carried out between the LHS var. and the RHS var.
>>=	Right shift and assign to the LHS.
<<=	Left shift and assign to the LHS.
&=	Bitwise AND operation and assign to LHS
|=	Bitwise OR operation and assign to LHS
~=	Bitwise complement operation and assign to LHS

This is a valid statements:

A=b=c+4;

C=3*(d=12.0/x);

<u>Exercise:</u>

Rewrite the equivalent statements for the following examples, and find it results. Assume: X=2 , Y=3 , Z=4 , V=12 , C=8.

Example	Equivalent Statement	Result
X += 5	X = X + 5	X ← 7
Y -= 8	Y = Y - 8	Y ← -5
Z *= 5	Z = Z * 5	Z ←
V /= 4		V ←
C %= 3		C ←

3. <u>Comparision and logical operators:</u> It has three types relational operators, equality operators, and logical operators.

(a) Relational operators: < less than, **>** greater than, **<=** less than or equal, **>=** greater than or equal, an expression that use relational operators return the value of one if the relational is TRUE ZERO otherwise.

Ex: 3 > 4 → **false,** 6 <=2 →**false,** 10>-32 → **true,** (23*7)>=(-67+89) → **true**

(b) Equality operators: == equal to , **!=** not equal to

Ex: a=4, b=6, c=8. A==b→**false,** (a*b)!=c→**true,**'s'=='y' →**false.**

(c) Logical operators: The logical expression is constructed from relational expressions by the use of the logical operators **not**(!), **and**(&&), **or**(| |).

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

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

NOT (!) Table:	
A	!A
T	F
F	T

NOT (!) Table:	
A	!A
1	0
0	1

Examples:

Example 1:

a=4, b=5, c=6

(a<b)&&(b<c)	(a<b)||(b>c)	!(a<b)||(c>b)	(a<b)||(b>c)&&(a>b)||(a>c)
T && T	T | | T	!(T) | | T	T | | F && F | | F
T	T	F | | T	T | | F | | F
		T	T | | F
			T

Example 2:

Assume: X=0, Y=1, Z=1. Find the following expression:

M = ++X | | ++Y && ++Z

M = ++X || ++Y && ++Z

= 1 | | (2 && 2)

= T || (T && T)

= T || T

= T

= 1

(d) Bitwise logical operator:

& bitwise AND, **^** bitwise exclusive OR(XOR), **|** bitwise inclusive OR,

>> bitwise left shift, **<<** bitwise right shift, **~** bitwise complement.

Ex: x=23 (0001 0111) **~x**=132 (1110 1000)

X=33 (0010 0001)

X << 3

0 01000010

0 10000100

1 00001000 the resultant bit pattern will be (0000 1000)

X=5, y=2 → **x&y** (0000) , **x|y** (0111) , **x^y** (0111)

(e) special operators:

1. **Unary operator:**

*	Contents of the storage field to which a pointer is pointing.
&	Address of a variable.
-	Negative value (minus sign).
!	Negative (0, if value ≠ 0, 1 if value =0).
~	Bitwise complement.
++	Increment.
--	Decrement.
Type	Forced type of conversion
Size of	Size of the subsequent data type or type in byte.

2. **Ternary operator:** It is called conditional operator, it is like if else construction:

   Expression 1 ? expression 2 : expression 3

$$\left.\begin{array}{l}\text{If (v\%2 == 0)}\\ \text{e = true}\\ \text{Else}\\ \text{e=false}\end{array}\right\}\quad \text{E=(v\%2 ==0)? True : false}$$

3. **Comma operator: (,)**

   Int a,b,c; or it is used in control statements

4. **Scope operator: (::)** It is used in a class member function definition.
5. **New and delete operators:** it is a method for carrying out memory allocations and deallocations.

6. **Other operators:** parentheses for grouping expressions, membership operators.

6. Type Conversion:

Some variables are declared as integers but sometimes it may be required to bet the result as floating point numbers. It is carried out in two ways:

(A) Converting by assignment:
int x; float y; x=y;
	(B) Cast operator:
	Result =(int) (19.2/4); or
	Result = int (19.2/4);