Programming Basics For Beginners

د. محمد إبراهيم الدسوقى

كلية هندسة و علوم الحاسب – قسم نظم المعلومات

جامعة الأمير سطام بن عبد العزيز - السعودية – محافظة الخرج

Email : mohamed_eldesouki@hotmail.com

What Is Programming and Program Development Life Cycle ?

- Programming is **a process of problem solving**

- Step 1: Analyze the problem
  - Outline the problem and its requirements
  - Design steps (algorithm) to solve the problem

- Step 2: Implement the algorithm
  - Implement the algorithm in code
  - Verify that the algorithm works

- Step 3: Maintenance
  - Use and modify the program if the problem domain changes

Algorithm:
  - Step-by-step problem-solving process

The Problem Analysis–Coding–Execution Cycle

- Understand the Overall problem
- Understand problem requirements
  - Does program require user interaction?
  - Does program manipulate data?
  - What is the output?
- If the problem is complex, divide it into subproblems
  - Analyze each subproblem as above

The Problem Analysis–Coding–Execution Cycle (cont'd.)

- Run code through compiler
- If compiler generates errors
  - Look at code and remove errors
  - Run code again through compiler
- If there are no syntax errors
  - Compiler generates equivalent machine code
- Linker links machine code with system resources

The Problem Analysis–Coding–Execution Cycle (cont'd.)

- Once compiled and linked, loader can place program into main memory for execution
- The final step is to execute the program
- Compiler guarantees that the program follows the rules of the language
  - Does not guarantee that the program will run correctly

The Language of a Computer

- Machine language: language of a computer
- Binary digit (bit):
  - The digit 0 or 1
- Binary code:
  - A sequence of 0s and 1s
- Byte:
  - A sequence of eight bits

The Evolution of Programming Languages

- Early computers were programmed in machine language
- To calculate `wages = rates * hours` in machine language:

```
100100 010001    //Load
100110 010010    //Multiply
100010 010011    //Store
```

The Evolution of Programming Languages

- Assembly language instructions are mnemonic
- Assembler: translates a program written in assembly language into machine language

TABLE 1-2 Examples of Instructions in Assembly Language and Machine Language

Assembly Language	Machine Language
LOAD	100100
STOR	100010
MULT	100110
ADD	100101
SUB	100011

The Evolution of Programming Languages

- Using assembly language instructions, `wages = rates • hours` can be written as:

```
LOAD  rate
MULT   hour
STOR   wages
```

The Evolution of Programming Languages

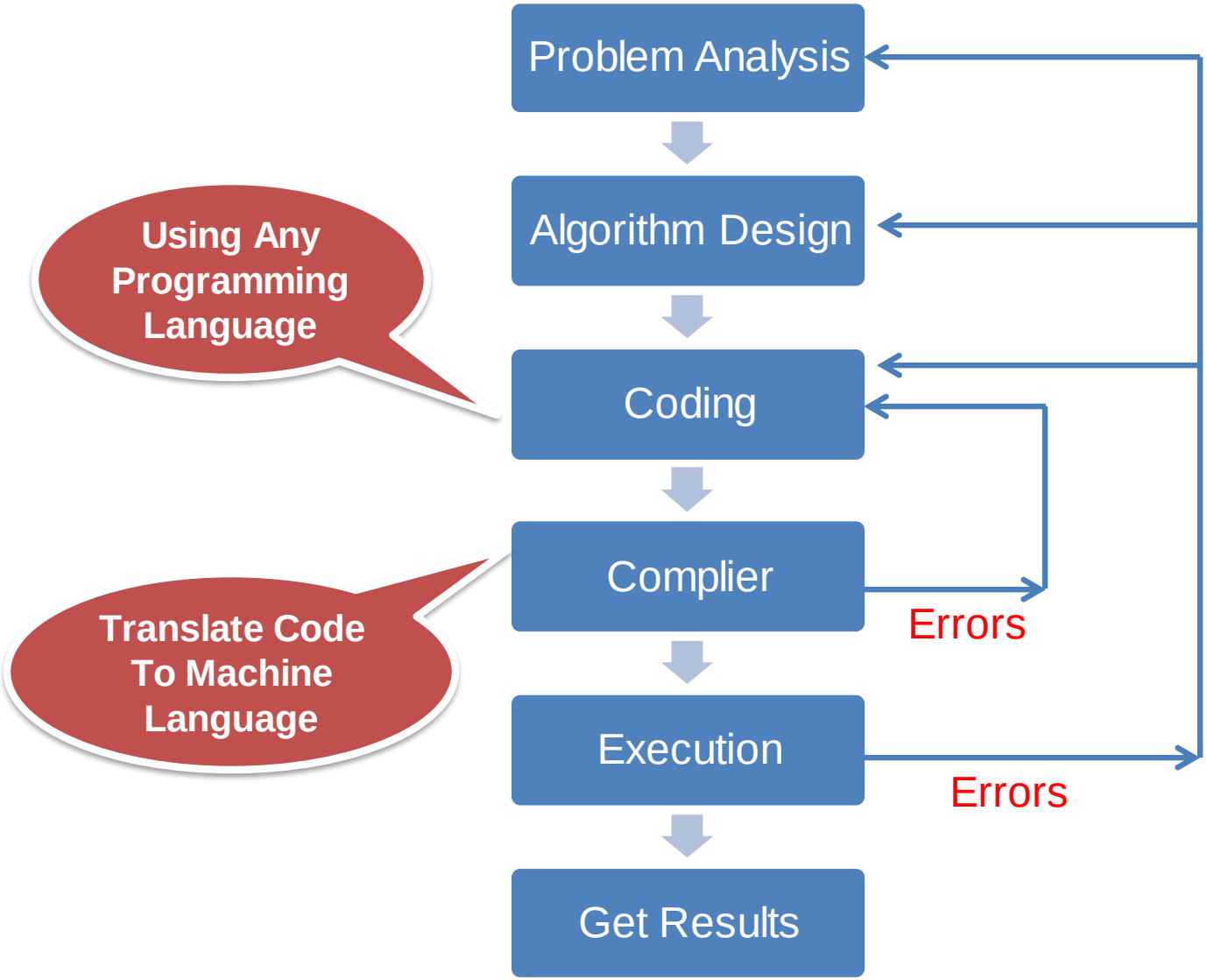
- High-level languages include Basic, FORTRAN, COBOL, Pascal, C, C++, C#, and Java

- **Compiler**: translates a program written in a high-level language machine language

- The equation `wages = rate • hours` can be written in C++ as:

```
wages = rate * hours;
```

The Problem Analysis–Coding–Execution Cycle



Think

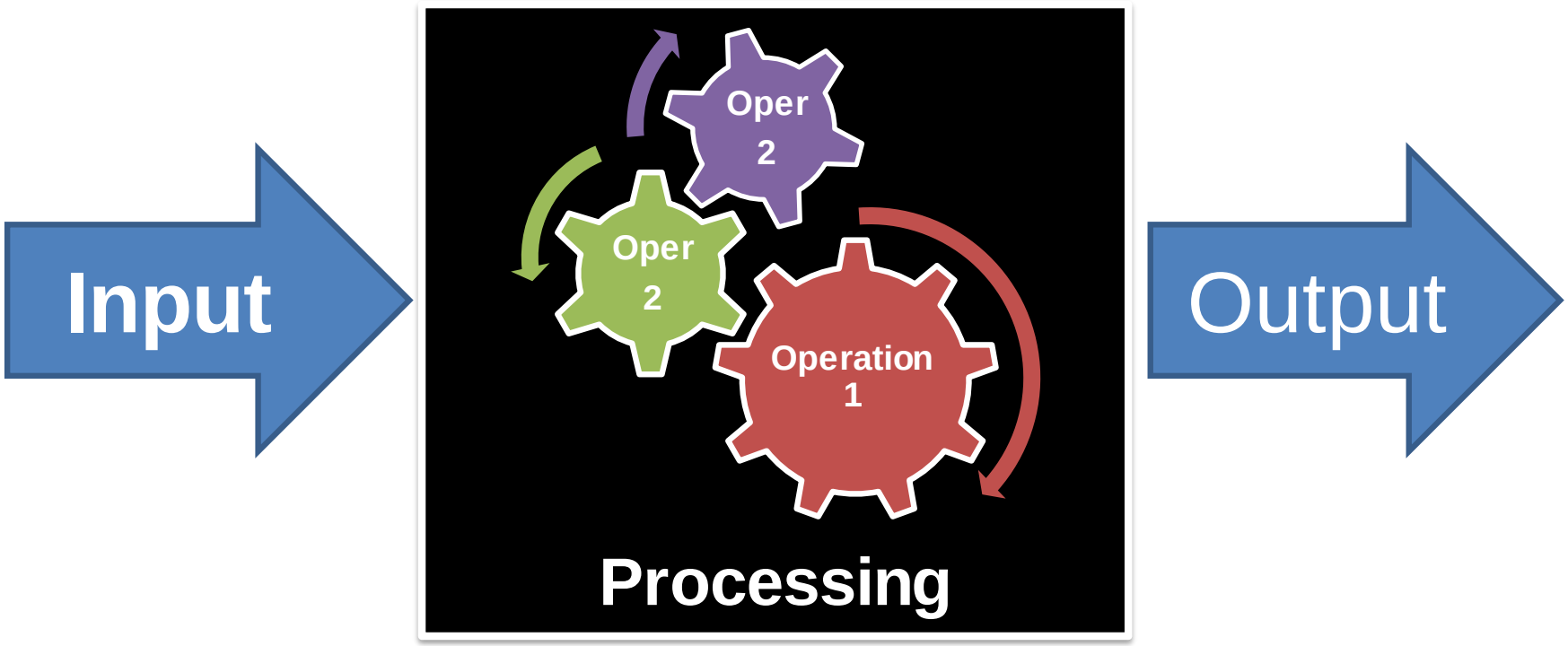
Then

Write Code

Basic Definitions

- Programming language:
  - a set of rules, symbols, and special words used to write computer programs.
- Computer program
  - Sequence of statements whose objective is to accomplish a task.
- Syntax:
  - rules that specify which statements (instructions) are legal

Computer System



Example 1

- **Write a program to find the Area of a rectangle**

The area of the Rectangle are given by the following formula:

Area = Rect Length * Rect Width.

Input :

Rectangle Length , Rectangle Width.

Processing :

Area = Rect Length * Rect Width.

Output :

Print Out The area.

Example 2

- **Write a program to find the perimeter and area of a square**

The perimeter and area of the square are given by the following formulas:

```
perimeter = Side Length * 4
area = Side Length * Side Length
```

Input:

Square Side Length

Processing:

perimeter = Side Length * 4

area = Side Length * Side Length

Output:

Print Out The Perimeter and Area.

Your First C++ Program

```
#include <iostream>
using namespace std;
int main()
{
  // This program is written by Mohamed El Desouki

   cout << "My first C++ program." << endl;

   return 0;
}
```

Sample Run:

```
My first C++ program.
```

Processing a C++ Program (cont'd.)

- To execute a C++ program:
  - Use an editor to create a source program in C++
  - Preprocessor directives begin with # and are processed by a the preprocessor
  - Use the compiler to:
    - Check that the program obeys the rules
    - Translate into machine language (object program)
  - Linker:
    - Combines object program with other programs provided by the SDK to create executable code
  - Loader:
    - Loads executable program into main memory
  - The last step is to execute the program

Processing a C++ Program (cont'd.)

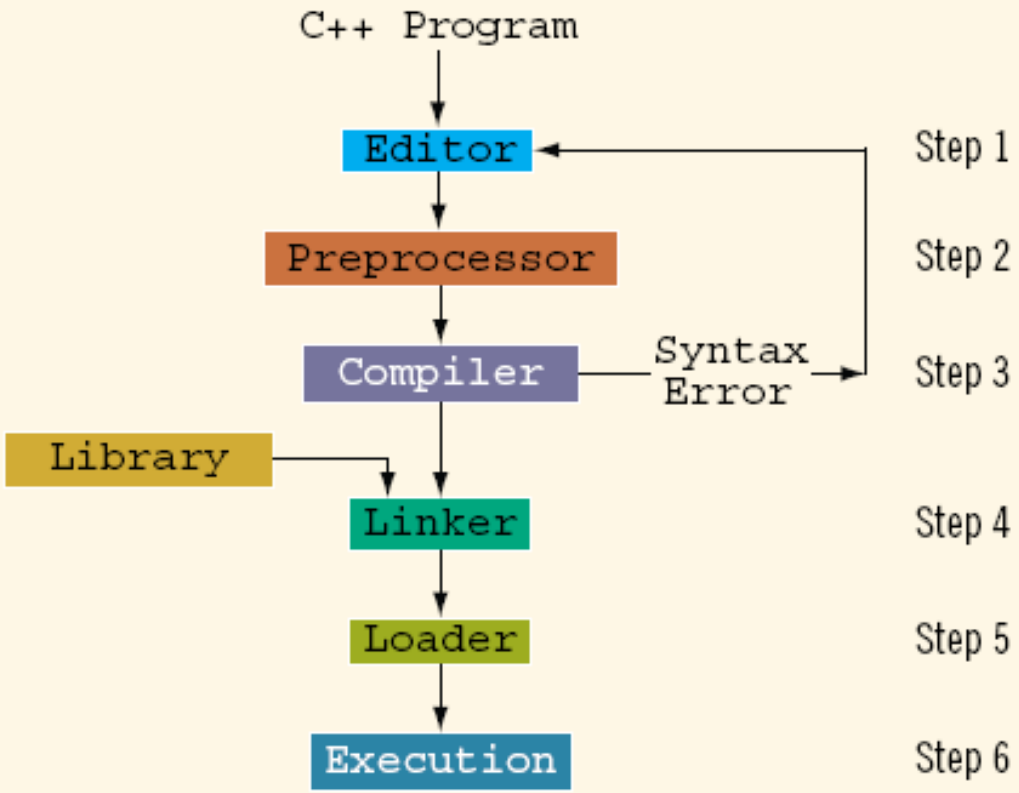


FIGURE 1-3 Processing a C++ program

Interacting With User: Displaying Messages on Screen

- In C++ , we use Cout << " Text To be Displayed on The screen " ;
- To use Cout << , we must use
  - #include <iostream> ;
  - using namespace std;

- #include <iostream> notifies the preprocessor to include the contents of the input/output stream header file <iostream> in the program
- We can use the Escape Sequence to format the Displayed Text.

Escape sequence	Description
\n	Newline. Position the screen cursor to the beginning of the next line.
\t	Horizontal tab. Move the screen cursor to the next tab stop.
\r	Carriage return. Position the screen cursor to the beginning of the current line; do not advance to the next line.
\a	Alert. Sound the system bell.
\\	Backslash. Used to print a backslash character.
\'	Single quote. Use to print a single quote character.
\"	Double quote. Used to print a double quote character.

Interacting With User: Comments

- We can put comment on our programs using
- **//** to comment a single line

```
// this progrm is written by mohamed El desouki
```

- **/***

  Mulitple Lines

- ***/** to comment multiple lines

```
/* This program is written by Mohamed El Desouki
On Monday 11/1/2012
*/
```

Example 1

- **Write a program to find the Area of a rectangle**

The area of the Rectangle are given by the following formula:

Area = Rect Length * Rect Width.

Input :

Rectangle Length , Rectangle Width.

Processing :

Area = Rect Length * Rect Width.

Output :

Print Out The area.

Central Processing Unit and Main Memory

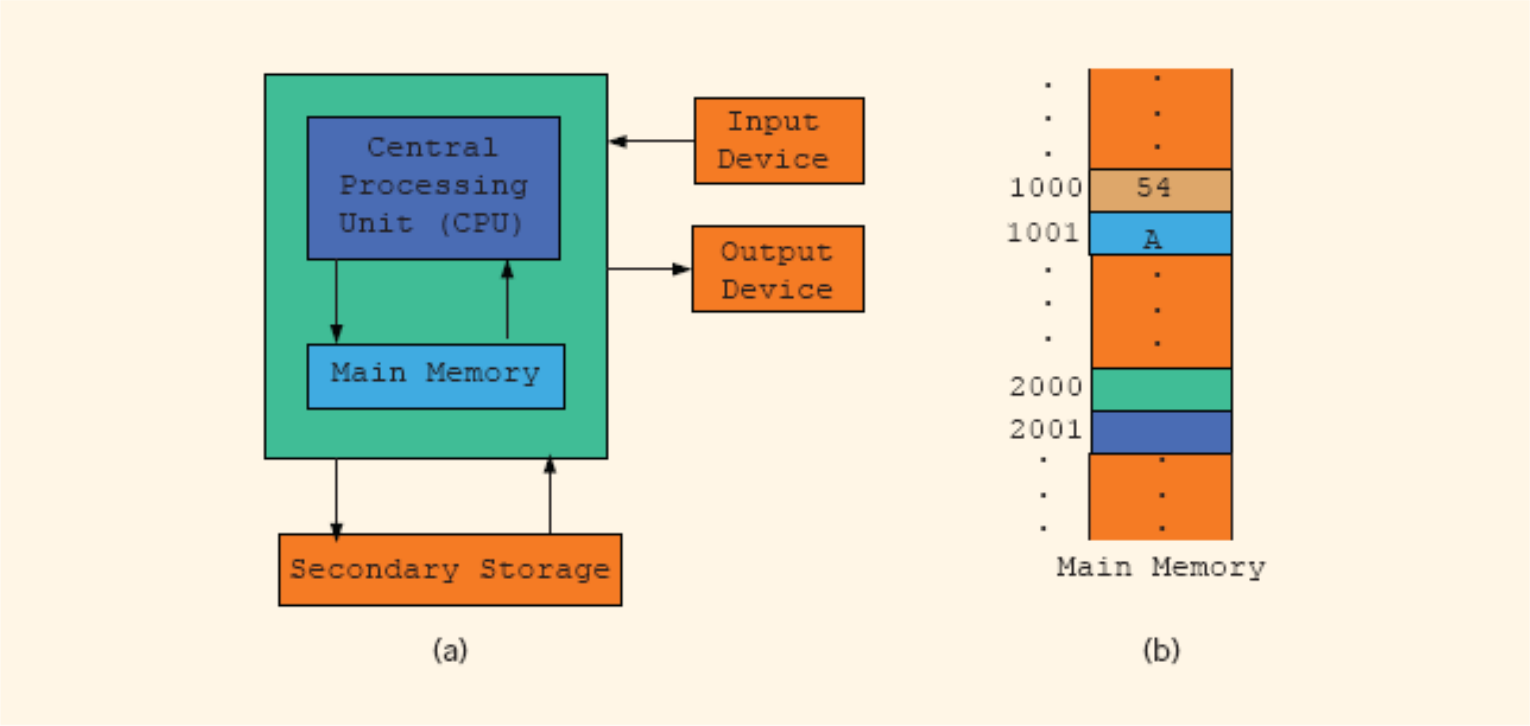


FIGURE 1-1 Hardware components of a computer and main memory

Interacting With User: Accept Input From User

- In C++ , we use Cin >> Variable; To accept an input from the user.
- To use Cin >>, we must use #include <iostream> ;
- `#include <iostream>` notifies the preprocessor to include in the program the contents of the input/output stream header file `<iostream>`.

- A variable is a location in the computer's memory where a value can be stored for use by a program.
- All variables must be **declared** with a **name** and a **data type** before they can be used in a program.

- Declaration

DataType Identifier;

Int width ;

Float salary ;

C++ Data Types

char	Character or small integer.	1byte	signed: -128 to 127 unsigned: 0 to 255
int	Integer.	4bytes	signed: -2147483648 to 2147483647 unsigned: 0 to 4294967295
short int (short)	Short Integer.	2bytes	signed: -32768 to 32767 unsigned: 0 to 65535
long int (long)	Long integer.	4bytes	signed: -2147483648 to 2147483647 unsigned: 0 to 4294967295
bool	Boolean value. It can take one of two values: true or false.	1byte	true or false
float	Floating point number.	4bytes	+/- 3.4e +/- 38 (~7 digits)
double	Double precision floating point number.	8bytes	+/- 1.7e +/- 308 (~15 digits)
long double	Long double precision floating point number.	8bytes	+/- 1.7e +/- 308 (~15 digits)

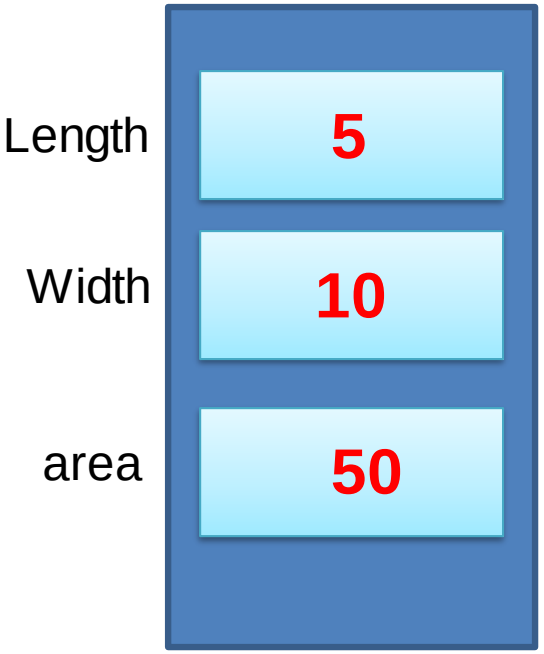
Working With Variable

```
Int  length ;
Int  width;
Int  area;

Cin>>Length;

Cin>>width;

Area =  Length * width  ;
```



Example 2

- **Write a program to find the perimeter and area of a square**

The perimeter and area of the square are given by the following formulas:

```
perimeter = Side Length * 4
area = Side Length * Side Length
```

Input:

Square Side Length

Processing:

perimeter = Side Length * 4

area = Side Length * Side Length

Output:

Print Out The Perimeter and Area.

Declaring & Initializing Variables

- Initialization means to give a variable an initial value.

- Variables can be initialized when declared:

```
int first=13, second=10;
char ch=' ';
double x=12.6;
```

- All variables must be initialized before they are used in an arithmetic operatation

  – But not necessarily during declaration

Rules on Variable Names

- **DO NOT use reserved words as variable names**
  (e.g. if, else, int, float, case, for, …).

- **The first character has to be a letter or underscore. It can not be a numeric digit.**

- **The second and the other characters of the name can be any letter, any number, or an underscore "_".**

Examples

Some valid names:

my_name, m113_1, salary, bluemoon , _at

Some invalid names:

my name, my-name , 1stmonth , salary! , guns&roses ,

Frequently used data types

Data Types	Bytes Used
int	4 Bytes
short	2 Bytes
double	8 Bytes
unsigned	4 Bytes
Float	4 Bytes
Double	8 Bytes
Char	1 Byte

• The data type **unsigned** is used to represent positive integers.

•**float** and **double** data types for storing real numbers.

The **float** data type has a precision of seven digits

-This means after the decimal point you can have seven digits

-Example: *3.14159* *534.322344* *0.1234567*

•The **double** data type has a precision of fifteen digits

Example :

-3738.7878787878 3.141592653589790

0.123456789123456

Frequently used data types

- We can use *Scientific Notation* to represent real numbers that are very large or very small in value.
- The letters e or E is used to represent times 10 to the power.

Example:

- 1.23×10^{5} *is represented as* 1.23e5 *or* 1.23e+5 *or* 1.23E5
- 1×10^{-9} *is represented as* 1e-9

Arithmetic Operations

C++ operation	C++ arithmetic operator	C++ expression
Addition	+	f + 7
Subtraction	-	p - c
Multiplication	*	b * m
Division	/	x / y
Modulus	%	r % s

Arithmetic operators.

- Parentheses are used in C++ expressions in the same manner as in algebraic expressions.

- For example, to multiply **a** times the quantity **b + c**

we write **a * (b + c).**

- There is no arithmetic operator for exponentiation in C++,

so **x2** is represented as **x * x**.

Precedence of arithmetic operations

Operator(s)	Operation(s)	Order of evaluation (precedence)
( )	Parentheses	Evaluated first. If the parentheses are nested, the expression in the innermost pair is evaluated first. [*Caution:* If you have an expression such as `(a + b) * (c - d)` in which two sets of parentheses are not nested, but appear "on the same level," the C++ Standard does *not* specify the order in which these parenthesized subexpressions will be evaluated.]
*, /, %	Multiplication, Division, Modulus	Evaluated second. If there are several, they're evaluated left to right.
+ -	Addition Subtraction	Evaluated last. If there are several, they're evaluated left to right.

Precedence of arithmetic operators.

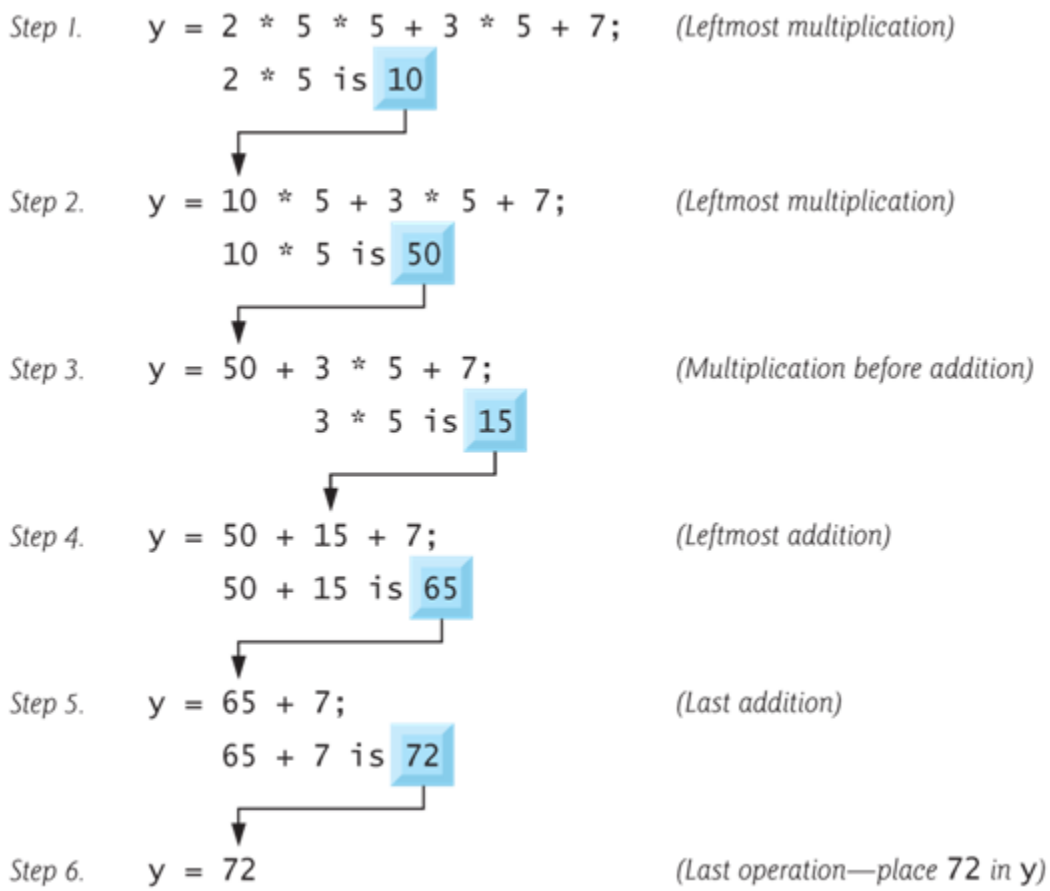
For example,

`2 + 3 * 5` and `(2 + 3) * 5`

both have different meanings

Precedence of arithmetic operations

Example :



Precedence of arithmetic operations

- ? = **1 + 2 * (3 + 4)**
  - Evaluated as **1 + (2 * (3+4))** and the result is **15**
- ? = **5 * 2 + 9 % 4**
  - Evaluated as **(5*2) + (9 % 4)** and the result is **11**
- ? = **5 * 2 % (7 – 4)**
  - Evaluated as **(5 * 2) % (7 – 4)** and the result is **1**

Data Type of an Arithmetic Expression

- Data type of an expression depends on the type of its operands
  - Data type conversion is done by the compiler

- If operators are *, /, +, or – , then the type of the result will be:
  - integer, if all operands are integer.
    - » Int A , B;
    - » A+ B → Integer.
  - float, If at least one operand is float and there is no double
    - » Int A ; Float B;
    - » A + B → Float.
  - double, if at least one operand is double
    - Int A ; Float B; Double C;
      - » A + B + C → double.

Data Type of an Arithmetic Expression

Example

int * int; → **result int**

int + float; → **result float**

Int + double / float; → **result double**

int – double; → **result double**

Data Type of an Arithmetic Expression

- The data type of the target variable is also important
- If the result is a real number and the target variable is declared as integer, only the integer part of the result will be kept, and decimal part will be lost.

Example

```
int avg;
float sum=100.0,  cnt = 6.0;
avg = sum / cnt;
```

The result is calculated as 16.66667

But avg will be 16

Data Type of an Arithmetic Expression

```
float avg;
int sum=100,  cnt = 6;
avg = sum / cnt;
```

The result of the division will be 16
avg will be 16.0

- Only the integer part of the result will be considered if two operands are integer Even when the target variable is float

Type Casting

```
int main()
{
    int  i=5, j=3;
    float div;
    div= i/j;
    cout<< div;
}
```

The div will be 1.0 and this is not write

```
int main()
{
    int  i=5, j=3;
    float div;
    div=(float) i/j;
    cout<< div;
}
```

Type cast: tells the compiler to treat **i** as a float

After type casting , The div will be 1.66667

Increment and Decrement Operators

- Increment operator: increment variable by 1
  - Pre-increment: ++variable
  - Post-increment: variable++

- Decrement operator: decrement variable by 1
  - Pre-decrement: --variable
  - Post-decrement: variable --

- Examples :

++K , K++ → k= K+1

--K , K-- → K= K-1

Increment and Decrement Operators

- If the value produced by ++ or – – is not used in an expression, it does not matter whether it is a pre or a post increment (or decrement).

- When ++ (or – –) is used before the variable name, the computer first increments (or decrements) the value of the variable and then uses its new value to evaluate the expression.

- When ++ (or – –) is used after the variable name, the computer uses the current value of the variable to evaluate the expression, and then it increments (or decrements) the value of the variable.

- **There is a difference between the following**

```
x = 5;
Cout << ++x;
```

```
x = 5;
Cout << x++;
```

special assignment statements

- C++ has special assignment statements called compound assignments

+= , -= , *= , /= , %=

- Example:

```
x +=5 ; means  x = x + 5;
x *=y;  means  x = x * y;
x /=y;  means  x = x / y;
```

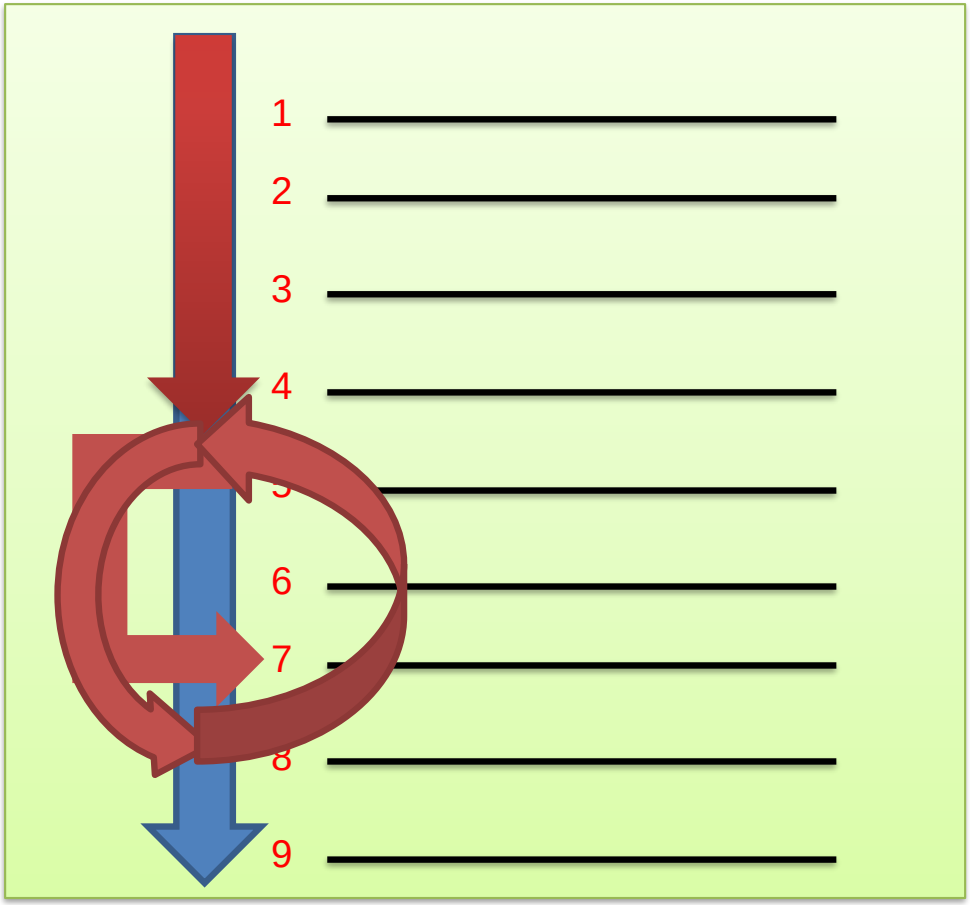
Control Statements

- Normally, statements in a program are executed one after the other in the order in which they're written.
- This is called sequential execution.

- There are control statements enable you to specify that the next statement to be executed may be other than the next one in sequence.
- This is called transfer of control.

- The control statements are categorized in almost two groups:
  - Selection control statements
  - Repetition control statements

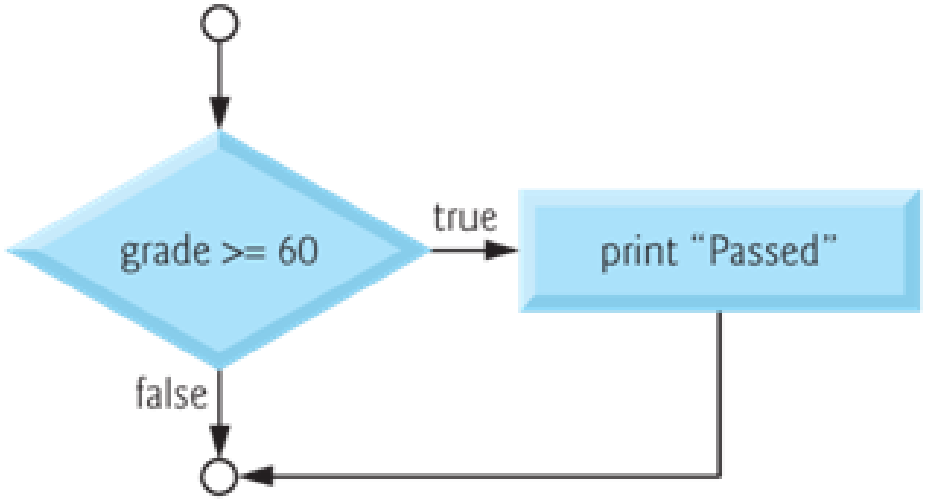
Transfer of Control



Selection Statements : **If Statement**

- Selection statements are used to choose among alternative courses of action.

- For example, suppose the passing mark on an exam is 60. The pseudocode statement
  - If student's marks is greater than or equal to 60 Then
    Print "Passed"



Flowcharting the single-selection `if` statement.

In C++ , The syntax for the If statement

```
if ( Expression)
   action statement ;
```

```
if ( Expression)
 {
   action statement 1 ;
   action statement 2 ;
   .
   .
   action statement n ;
}
```

```
if ( grade >= 60 )
   cout <<"Passed\n";
```

- The **Expression** can be any valid expression including a relational expression and even arithmetic expression

- In case of using arithmetic expressions , a non-zero value is considered to be true, whereas a 0 is considered to be false

Relational Expression and Relational Operators

- Relational expression is an expression which compares 2 operands and returns a TRUE or FALSE answer.

  Example : a >= b , a == c , a >= 99 , 'A' > 'a'

- Relational expressions are used to test the conditions in selection, and looping statements.

Operator	Means
==	Equal To
!=	Not Equal To
<	Less Than
<=	Less Than or Equal To
>	Greater Than
>=	Greater Than or Equal To

Selection Statements : **If Statement**

Example : write a program that accept an integer from the user and in case this integer is even print out the following message

“This number is even “ .

Selection Statements : **If .. Else Statement**

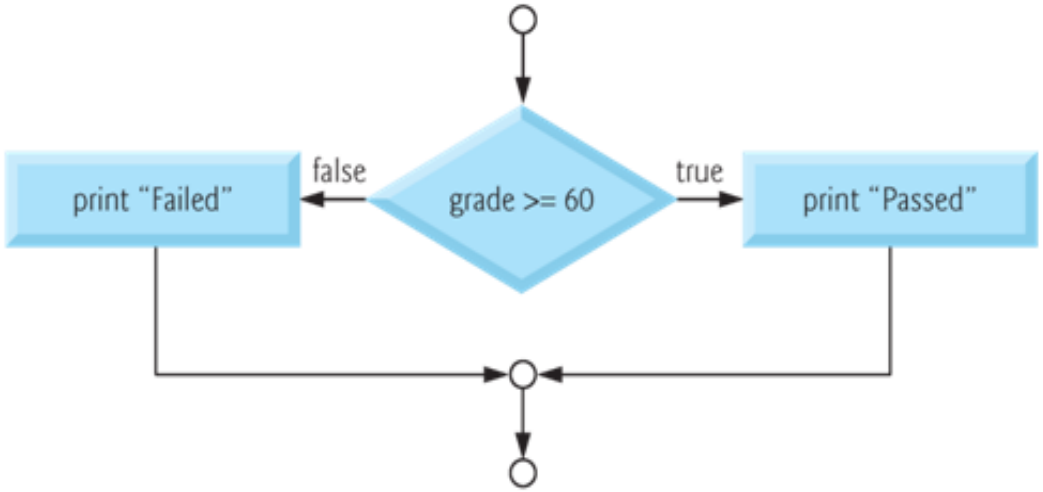
- The IF…Else selection statement allows you to specify that there is a course of actions are to be performed when the condition is true and another course of actions will be executed when the condition is false.

- For example, the pseudocode statement
  - **If** student's mark is greater than or equal to 60

    Print "Passed"

    else

    Print "Failed"

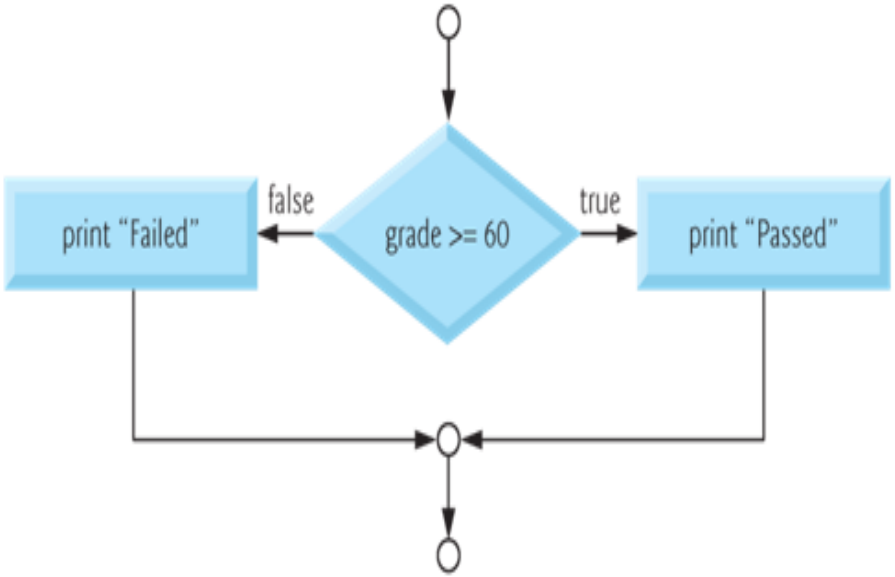


In C++ , The syntax for the If…Else statement

```
if ( Expression)
   action statement ;
Else
   action statement ;
```

```
if ( Expression)
 {
   action statements 1 ;
   .
   action statement n ;
}
Else
{
action statements 1 ;
   .
action statement n ;
}
```

```
if ( grade >= 60 )
   cout <<"Passed\n";
Else
   cout <<"Failed\n"
```



Selection Statements : **If – else Statement**

Example : write a program that accept an integer from the user and print out whether it is Positive or Negative number.

Nested If

- **Nested If : means to write an if statement within another if statement.**

Example : write a program that accept an integer number from the user , in case the number is Positive , check and print out whether it is Even or Odd number.

```
int main()
{
int number;
cout <<"Please Enter any number \n";
cin >>number;

if ( number >=0)
      if (number % 2 == 0)
            cout <<" This is an Even number \n";
      else
            cout <<"This is an Odd number \n \n";

}
```

IF – Else IF statement

- For example, write a program that ask the user to Enter 2 numbers and print out whether they are equal or there is one which is greater than the other.

```
int main()
{
        int num1, num2;
        cout <<"Enter Number 1 , Number2  \n";
        cin >>num1>>num2;

        if ( num1 == num2 )
          cout << "Both Are Equal \n";
        else if (num1 > num2 )
         cout <<"Number 1 is greater than number 2 \n";
        else
                cout <<"Number 2 is greater than number 1 \n";

}
```

IF – Else IF

- For example, the following code will print
  - A for exam grades greater than or equal to 90,
  - B for grades greater than or equal to 80,
  - C for grades greater than or equal to 70,
  - D for grades greater than or equal to 60, and
  - F for all other grades.

```
if ( grade >= 90 )
   cout << "A\n“ ;
else if ( grade >= 80 )
 cout << "B\n”;
else if ( grade >= 70 )
 cout << "C\n”;
else if ( grade >= 60 )
 cout << "D\n”;
else
 cout << "F\n“ ;
```

Combining more than one condition

- To combine more than one condition we use the logical operators.

Operator	Means	Description
&&	**And**	The Expression Value Is true If and Only IF both Conditions are true
||	**OR**	The Expression Value Is true If one Condition Is True

Example : check whether num1 is between 0 and 100

```
IF ( (num1 >= 0) && (num1 <=100) )
    Cout <<"The Number Is between 1 and 100" ;
Else
    Cout <<" The Number Is Larger Than 100";
```

Combining more than one condition

Example, print out the student grade according to the following formulas:

- A for exam marks greater than or equal 90 and less than or equal 100 ,
- B for exam marks greater than or equal 80 and less than 90 ,
- C for exam marks than or equal to 70 and less than 80 ,
- D for exam marks than or equal to 60, and less than 70 ,
- F for all other marks.

```
if ( marks >= 90      && marks <= 100)
   cout << "A\n“ ;
else if (marks >= 80 && marks <90 )
 cout << "B\n”;
else if (marks >= 70 && marks <80 )
 cout << "C\n”;
else if (marks >= 60 && marks <70 )
 cout << "D\n”;
else
 cout << "F\n“ ;
```

Example : A company insures its Employees in the following cases:

- Employee is married.
- Employee is an Single male above 30 years of age.
- Employee is an Single female above 25 years of age.

- **Conditions :**

1. Marital status = ‘M’; **OR**
2. Marital Status =‘S’ and Sex=‘M’ and Age >30 **OR**
3. Marital Status =‘S’ and Sex=‘F’ and Age >25

Selection statements: Switch Statement.

The switch control statement that allows us to make a decision from the number of choices

```
Switch (Expression )
{
case constant 1 :
        Action statement
        break ;
case constant 2 :
        Action statement
        break ;
case constant 3 :
        Action statements;
        break ;
default :
        Action statements;
}
```

Expression It could be an integer constant like

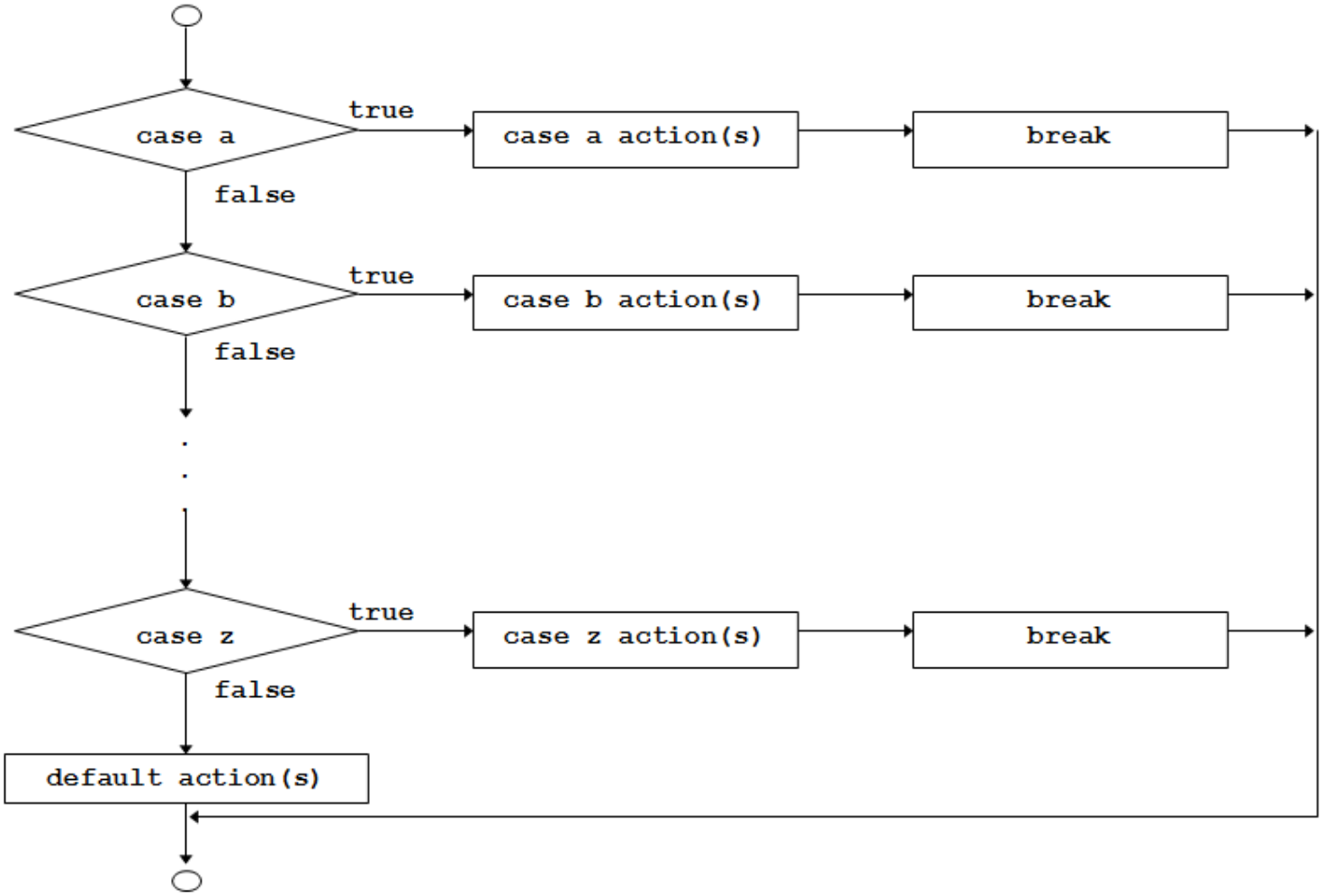
tes to an

```
Switch (10.0);

float i ;
Switch (i);

Case  >= 50;
Case   a+ b;
```

* k)


case a
true
case a action(s)
break
false
case b
true
case b action(s)
break
false
case z
true
case z action(s)
break
false
default action(s)

```
int i;
Cin>>I;
switch ( i )
{
case 10 :
        Cout << "I am in case 1 \n“  ;
        break ;
case 20 :
        Cout << "I am in case 2 \n“ ;
        break ;
case 30 :
        Cout << "I am in case 3 \n“ ;
        break ;
default :
       Cout << "I am in default \n“ ;
}
```

```
char ch = 'x' ;

switch ( ch )
{
case 'v' :
        Cout<< "I am in case v \n“  ;
        break ;
case 'a' :
        Cout<<  "I am in case a \n“  ;
        break ;
case 'x' :
        Cout<< "I am in case x \n” ;
        break ;
default :
         Cout<< "I am in default \n„  ;
}
```

Expression Possible Forms

switch (i + j * k)
switch (23 + 45 % 4 * k)

```
if ( marks >= 90      && marks <= 100)
      cout << "A\n“ ;
   else if (marks >= 80 && marks <90 )
    cout << "B\n”;
   else if (marks >= 70 && marks <80 )
    cout << "C\n”;
   else if (marks >= 60 && marks <70 )
    cout << "D\n”;
   else
    cout << "F\n“ ;
```

```
switch (grade)
{
case 90 :
cout <<"You Got A \n";   break;

case 80 :
cout <<"You Got B \n";   break;

case 70 :
cout <<"You Got C \n";   break;

case 60 :
cout <<"You Got D \n";   break;

default :
cout <<" Sorry , You Got F \n";
}
```

Switch Versus IF – Else IF

- There are some things that you simply cannot do with a **switch. These are:**
  - A float expression cannot be tested using a **switch**
  - Cases can never have variable expressions (for example it is wrong to say **case a +3 :)**
  - Multiple cases cannot use same expressions.

  - **switch works faster than an equivalent IF - Else ladder.**

Control Statements : Repetition statements

- Some times we need to repeat a specific course of actions either a specified number of times or until a particular condition is being satisfied.

- For example :
  - To calculate the Average grade for 10 students ,
  - To calculate the bonus for 10 employees or
  - To sum the input numbers from the user as long as he/she enters positive numbers.

- There are three methods by way of which we can repeat a part of a program. They are:
  - **(a) Using a while statement**
  - **(b) Using a do-while statement**
  - **(C) Using a for statement**

1- The While Loop

```
While ( continuation condition)
{

 Action statement 1 ;
Action statement  2 ;
.
.
Action statement  n ;
}
```

```
initialise loop counter ;
while ( test loop counter using a condition )
{
      do this ;
      and this ;
      increment loop counter ;
}
```

1- loop counter is any numeric variable (int , float ,….).

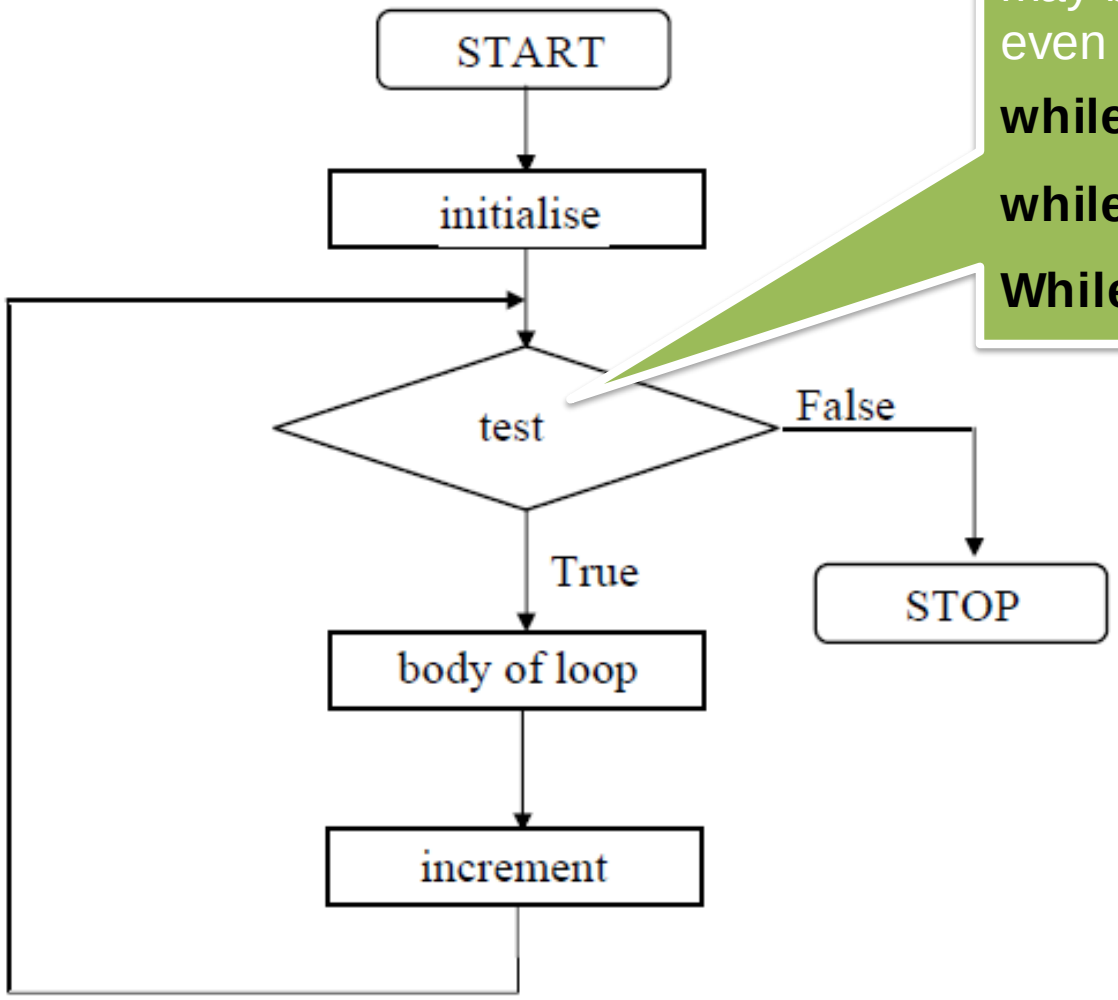
2- when using a counter , the while loop is called

counter –controlled loop.

3- The initial value of the loop counter is up to you,

but you have to increment it to avoid endless loops.

1- The While Loop



The condition being tested may be relational, logical or even numeric expression :

while (i <= 10)

while (i >= 10 && j <= 15)

While (3) , While (3 + 5)

1- The While Loop

- **Example** : Write a program that calculates and prints out the Average grade for 6 students .

```
int counter = 1;
int grade=0 , sum=0;
while (counter <=6)
{
cout <<"Enter grade for student no " << counter <<"\n";
cin >>grade;
sum += grade;
counter ++;
}
cout <<"Average Grade is " << sum/counter <<"\n";
```

1- The While Loop

- **Example** : Write a program To print out the sum of the numbers entered from the user as long as he/she enters positive numbers.

```
int number=0 , sum=0;

while (number >= 0)
{
cout <<" Enter Positive numbers to sum \n";
cin>>number;
sum += number;
}
Cout <<“ sum  = “ << sum <<“\n”;
```

1- The While Loop

- **Example** : Write a program To print out the sum of the numbers entered from the user as long as he/she enters positive numbers.

```
int number=0 , sum=0;
cout <<" Enter Positive numbers to sum \n";
cin>>number;

while (1)
{
sum += number;
cout <<" Enter Positive numbers to sum \n";
cin>>number;
If (number < 0)
 break;
}
```

1- The While Loop

- **Example** : Write a program that calculates and prints out the Average grade for 5 students or ends the program by entering -1.

```
int grade=0, counter =1, sum=0;
cout <<" Enter 5 grades or -1 To Exit \n";
while (counter <=5 && grade !=-1)
{
cin>>grade;
sum += grade;
counter ++;
}

cout <<"The sum of grades is " << sum <<"\n";
```

2- The do-while Loop

```
do
{
      this ;
      and this ;
      and this ;
      and this ;
} while ( this condition is true ) ;
```

```
while ( this condition is true )
{
      this ;
      and this ;
      and this ;
      and this ;
}
```

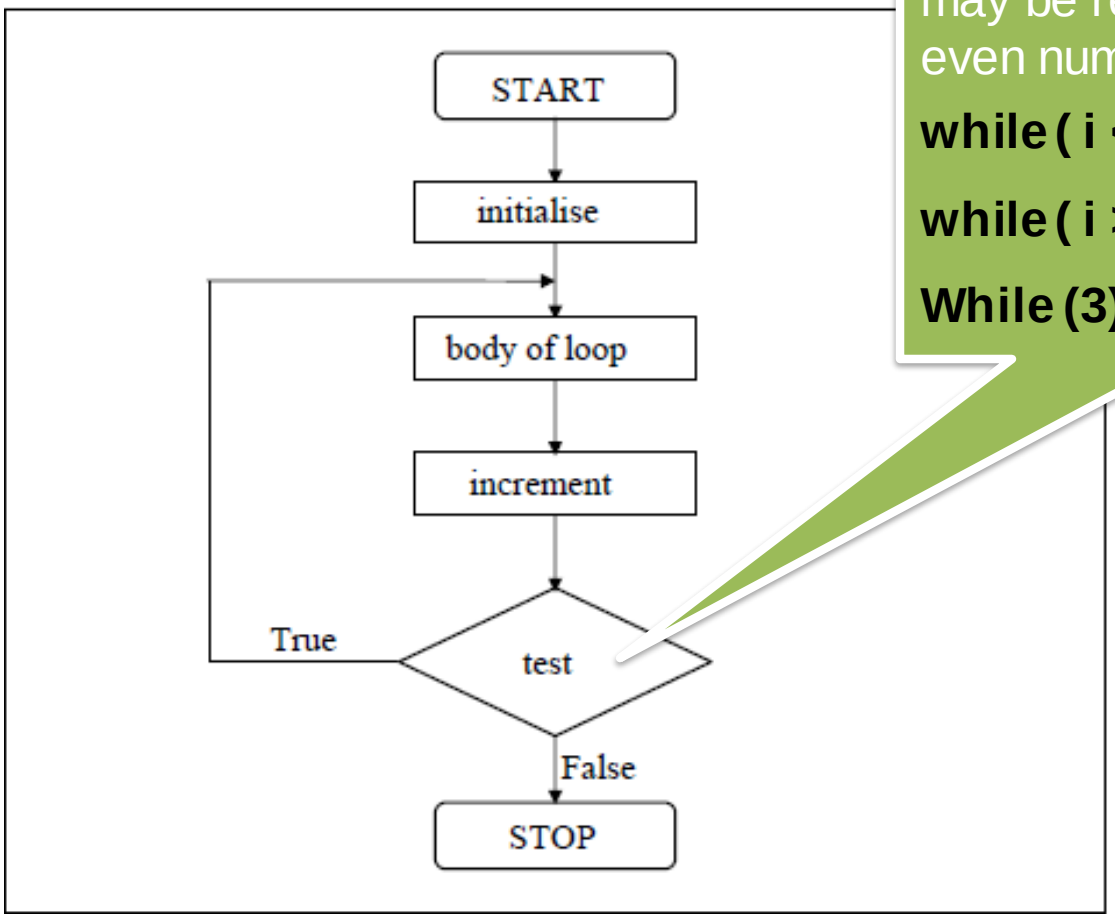
1- loop counter is any numeric variable (int , float ,….).

2- The initial value of the loop counter is up to you,

but you have to increment it to avoid endless loops.

3- The Loop Body is Executed at least one Time.

2- The do-while Loop



The condition being tested may be relational, logical or even numeric expression :

while (i <= 10)

while (i >= 10 && j <= 15)

While (3) , While (3 + 5)

2- The do-While Loop

- **Example** : Write a program that calculates and prints out the Average grade for 6 students .

```
int counter = 1;
int grade=0 , sum=0;
do
{
cout <<"Enter grade for student no " << counter <<"\n";
cin >>grade;
sum += grade;
counter ++;
}
while (counter <=6) ;

cout <<"Average Grade is " << sum/counter <<"\n";
```

2- The do-While Loop

Consider the following two loops:

a.
```
i = 11;
while (i <= 10)
{
    cout << i << " ";
    i = i + 5;
}
cout << endl;
```

b.
```
i = 11;
do
{
    cout << i << " ";
    i = i + 5;
}
while (i <= 10);

cout << endl;
```

In (a), the `while` loop produces nothing. In (b), the `do...while` loop outputs the number 11 and also changes the value of `i` to 16.

Control Statements : Repetition statements

- Some times we need to repeat a specific course of actions either a specified number of times or until a particular condition is being satisfied.

- For example :
  - To calculate the Average grade for 10 students ,
  - To calculate the bonus for 10 employees or
  - To sum the input numbers from the user as long as he/she enters positive numbers.

- There are three methods by way of which we can repeat a part of a program. They are:
  - **~~(a) Using a while statement~~**
  - **~~(b) Using a do-while statement~~**
  - **(C) Using a for statement**

3- The For Loop.

- For Loop is probably the most popular looping instruction.

- general form of **for statement is**

```
for ( initialise  counter ; test  counter ; increment  counter )
{
      do this ;
      and this ;
      and this ;
}
```

- The **for allows us to specify three things about** a loop in a single line:

(a) Setting a loop counter to an initial value.

(b) testing the loop counter to detect whether its value reached the number of repetitions desired.

(c) increasing the value of loop counter each time the program segment within the loop has been executed.

3- The For Loop.

- **Example** : Write a program that calculates and prints out the Average grade for 6 students .

```
int grade=0, sum=0;
for (int counter =1 ; counter <=6 ; counter ++)
{
cout <<"Enter Grade \n";
cin>>grade;
sum += grade;
}
cout <<"The Average grades is " << sum/6 <<"\n";
```

```
int counter = 1;
int grade=0 , sum=0;
while (counter <=6)
{
cout <<"Enter grade for student \n" ;
cin >>grade;
sum += grade;
counter ++;
}
cout <<"Average Grade is "
<< sum/6 <<"\n";
```

3- The For Loop.

- **Example** : Write a program that prints out numbers from 0 to 10;

```
for (int i= 0 ; i <=10 ; i++)
        cout << i << "\n“;
```

- **Example** : Write a program that prints out numbers from 0 to 10 in descending order;

```
for (int i = 10 ; i >=0 ; i-- )
        cout << i << "\n“;
```

- **Example** : Write a program that prints out the even numbers from 2 to 20;

```
for (int i = 2 ; i <=20 ; i+=2 )
        cout << i << "\n“;
```

3- The For Loop.

- **Example** : Write a program that accepts 10 numbers from the user and prints out the sum of even numbers and the sum of odd numbers.

3- The For Loop.

```
int i = 1 ;
for ( ; i <= 10 ; i + + )
cout<< i ;
```

```
int i ;
for ( i = 0 ; i++ < 10 ; )
cout<< i ;
```

```
int i = 1 ;
for ( ; i <= 10 ; )
{
cout<< i ;
i ++;
}
```

```
int i ;
for ( i = 0 ; ++ i < 10 ; )
cout<< i ;
```

3- The For Loop.

- **Example** : Write a program that calculates the Factorial for any given positive number.

  Ex : **Factorial (5) = 5 * 4 * 3 * 2 * 1**

```
int number, factorial=1;
cout <<"Enter a positive number\n";
cin >> number;
if (number < 0 )
cout <<" Enter Positive Numbers only\n";
else
for (int i= 1 ; I <=number ; i++)
     factorial = factorial * i;
cout <<" Factorila  = “ << factorial <<"\n";
```

Nested Loops

- **Example** : Write a program that calculates the Factorial for numbers from 1 to 10;

```
for ( int number=1; number<=10 ; number++)
{
      for ( int i= 1 ; i <=number ; i++)
      {
           factorial = factorial * i ;
      }
  cout <<" Factorila of " << number <<"=" << factorial <<"\n";
}
```

Nested Loops

- **Example** : Write a program that prints out the following pattern.

```
*
**
***
****
*****
******
```

```
for (int i = 1; i <= 5; i++)
{
    for (int j = 1; j <= i ; j++)
    cout << "*";
cout << endl;
}
```

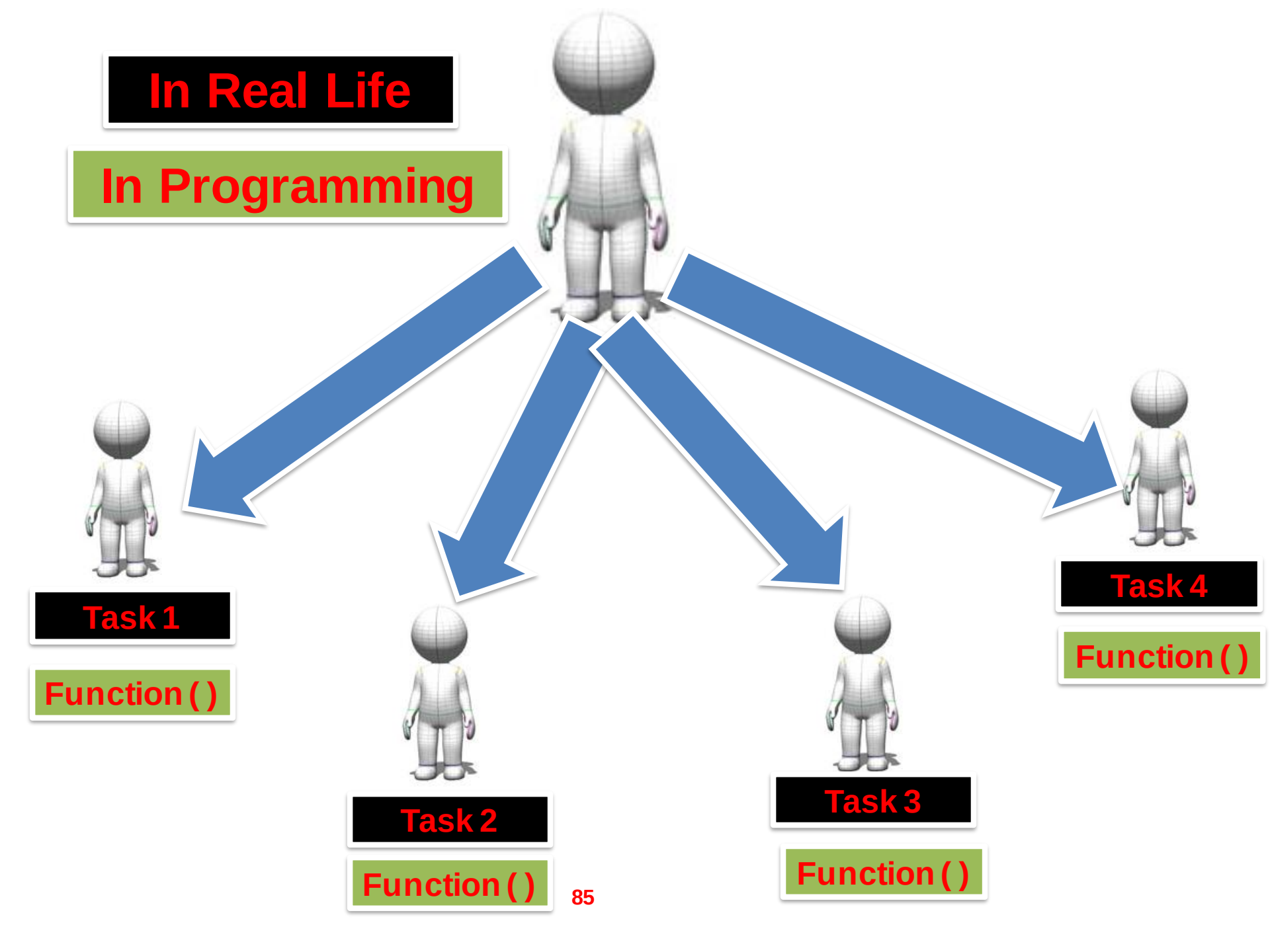
Functions

- Most computer programs that solve real-world problems include hundreds and even thousands of lines.

- Experience has shown that the best way to develop and maintain a large program is to construct it from smaller pieces or modules, each of which is more manageable than the original program.

- This technique is called divide and conquer.

- **A Function :** is a self-contained block of statements that perform a specific task.
- using a **function** is something like hiring a person to do a specific job for you.


In Real Life
In Programming
Task 1
Function ( )
Task 2
Function ( )
Task 3
Function ( )
Task 4
Function ( )

Why to use Functions ?

- **Functions** allows to write more manageable units of code.
- Writing functions avoids rewriting the same code over and over.
- Easy maintenance for programs.

Important Tips

1- Don't try to cram the entire logic in one function. It is a very bad style of programming.

2- Instead, break a program into small units and write functions for each of these isolated subdivisions.

3- Don't hesitate to write functions that are called only once.

Calculate and print out The sum and the Average of 3 student marks.

Task 1

Get_Marks( )

Task 2

Calc_sum( )

Task 3

Calc_Average ( )

Task 4

Print_out ( )

Functions

Function Definition

```
return-value-type function-name( parameter-list )
{
Lines of code to be
executed
...
...
...
(Body)
}
```

Parameter-List : Is a list of data values supplied from the calling program as input to the function. It is **Optional**

return-value-type: Is the data type for the returned value from the function to the calling program. It is **Mandatory, if no returned value expected , we use keyword Void.**

Function Types

1. Built in Functions (Ready Made).

- For Example : Math Library Functions <cmath> Like

Function	Header File	Purpose	Parameter(s) Type	Result
`floor(x)`	`<cmath>`	Returns the largest whole number that is not greater than x: `floor(45.67) = 45.00`	`double`	`double`
`pow(x, y)`	`<cmath>`	Returns x^y; if x is negative, y must be a whole number: `pow(0.16, 0.5) = 0.4`	`double`	`double`
`sqrt(x)`	`<cmath>`	Returns the nonnegative square root of x; x must be nonnegative: `sqrt(4.0) = 2.0`	`double`	`double`
`abs(x)`	`<cmath>`	Returns the absolute value of its argument: `abs(-7) = 7`	`int (double)`	`int (double)`
`ceil(x)`	`<cmath>`	Returns the smallest whole number that is not less than x: `ceil(56.34) = 57.0`	`double`	`double`

Built in Functions

To correctly use the built in functions we must know well its header (prortotype).

return-value-type function-name(*parameter-list*)

int abs (int number) ;

double pow (double base, double exponent) ;

Double floor (double number);

Double sqrt (double number);

```
#include <iostream>
#include <cmath>
using namespace std;
int main ( )
{
int i = -5 ;
double x = 5.0;
double y = 2.0;
double z = 5.21;

cout << "The absolute value of i is " << abs (i) << "\n \n";

cout << "The power of x to the power y is " << pow (x,y) << "\n \n";

cout << "The Floor for Z is " << floor (z) <<"\n \n";

cout << "The Ceil for Z is " << ceil (z) <<"\n \n";
}
```

Function Types

2. User Defined Functions (Tailored by Programmers).

Working with user defined functions is done in three steps :

1- Function declaration (**Prototype** , in C++ written before Main ( )).

2- Function Definition (**Body**, in C++ written After Main ( )).

3- Invoking Function (**Calling**, From The Main ( )).

Building Functions

1- Function Declaration (Written before The main( ))

```
return-value-type function-name( Formal parameter-list )
```

2- Function D

```
ret                                    ameter-list )
{
Line
exec
}
```

Formal Parameter-L
Is a list of input param
with their data types .
Int abs (int x)

Actual Parameter-List : Is a list of input parameters without their data types .
Cout << abs (i);

3- Invoking Function (From Th ain( ))

```
Function name ( Actual Parameter- List );
```

User-Defined Functions

- Value-returning functions: have a return type
  - Return a value of a specific data type using the `return` statement
  - the returned value can be used in one of the following scenarios:
    - **Save the value for further calculation**

      Int result;

      Result = sum (x , y);

    - **Use the value in some calculation**

      Result = sum(x , y) /2;

    - **Print the value**

      Cout << sum(x , y);

- Void functions: do not have a return type
  - *Do not* use a `return` statement to return a value

Flow of Execution

- Execution always begins at the first statement in the function `main`
- Other functions are executed only when they are called
- Function prototypes appear before any function definition
  - The compiler translates these first
- The compiler can then correctly translate a function call

Flow of Execution (cont'd.)

- A function call results in transfer of control to the first statement in the body of the called function
- After the last statement of a function is executed, control is passed back to the point immediately following the function call
- A value-returning function returns a value
  - After executing the function the returned value replaces the function call statement

- **For Example** : write a program that ask the user to Enter 2 integer numbers and print out the larger of them.

```
int larger (int num1 , int num2);
int main ()
{
int n1 , n2 , result;
cout <<"Please Enter 2 integer numbers \n";
cin >>n1 >> n2;
result = larger(n1,n2);
cout <<" The Larger number is " << result<<"\n";
}

int larger (int num1 , int num2)
{
 int max;
if (num1 >= num2)
max = num1;
else
max = num2;
return max;
}
```

For Example: Write a program to calculate the Area and volume for a sphere.

- The area of sphere = 4 * PI * Radius * Radius .
- The Volume of sphere = 4/3 * PI * Radius * Radius * Radius .
- ***Note : PI = 3.14***

For Example : write a program that ask the user to Enter 3 integer numbers and print out their sum and Average.

```
int sum (int num1 , int num2, int num3);
float average (int num1,int num2, int num3 );
int main ()
{
int n1 , n2 , n3 ;
cout <<"Please Enter 3 integer numbers \n";
cin >>n1 >> n2 >> n3;
cout <<" The sum of the 3 number is " << sum (n1, n2,n3) <<"\n";
cout <<" The average of the 3 number is " << average (n1, n2,n3) <<"\n";
}
int sum (int num1 , int num2, int num3)
{
return num1+num2+num3;
}

float average (int num1, int num2, int num3 )
{
return sum (num1 , num2 , num3)/3;
}
```

Function Parameter's Default Value

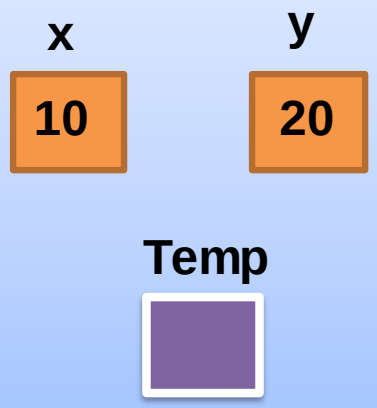
```
int sum (int num1 , int num2, int num3 = 90);

int main( )
{
int n1 , n2  ;
cout <<"Please Enter 3 integer numbers \n";
cin >>n1 >> n2 ;
cout <<" The sum of the 3 number is " << sum (n1, n2) <<"\n";
}
int sum (int num1 , int num2, int num3 = 90)
    {
        return num1+ num2+ num3;
    }
```

Function Parameters Types

- **Value parameter**: **Reference parameter**:
  - The value parameter has its own copy of the data
  - During program execution, The value parameter manipulates the data stored in its own memory space. **So** , the actual parameter not Affected with any change.

```
void swap( int  x,  int  y )
{
int temp;
temp= x;
 x = y;
 x = temp;
}
```



Function Parameters Types

- **Reference parameter**:
  - A reference parameter stores the address of the corresponding actual parameter
  - During program execution to manipulate data, The address stored in the reference parameter directs it to the memory space of the corresponding actual parameter
- Reference parameters are useful in three situations:
  - Returning more than one value
  - Changing the actual parameter
  - When passing the address would save memory space and time

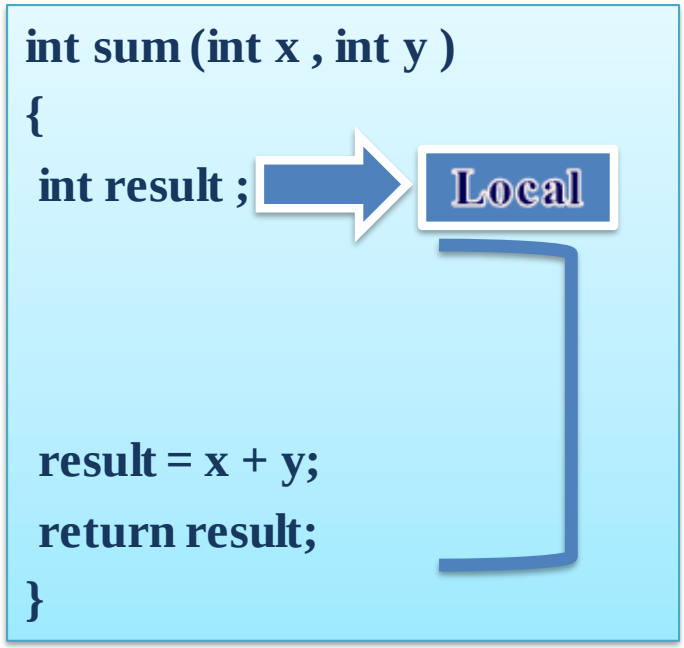
Function Parameters Types

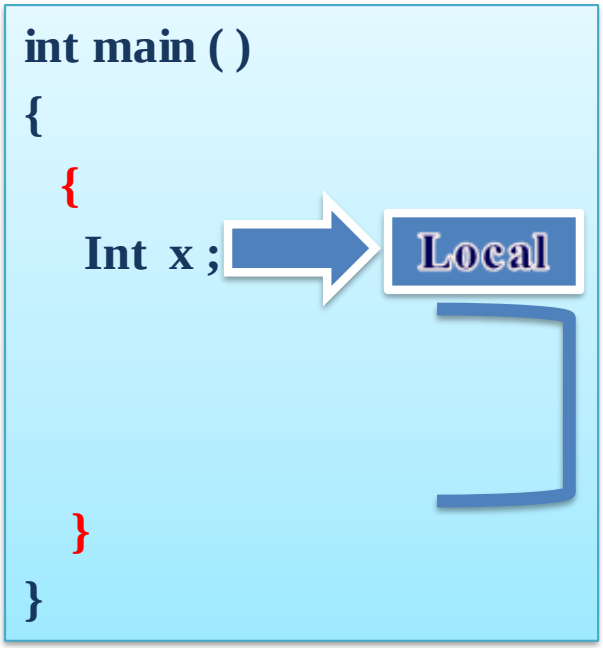
```
void swap( int  &x,  int  &y )
{
int temp;
temp= x;
 x = y;
 x = temp;
}
```

Scope of a variable

scope is the context within a program in which a variable is valid and can be used.

- ***Local variable***: declared within a function (or block)
- can be accessible only within the function or from declaration to the end of the block.
- Within nested blocks if no variable with same name exists.

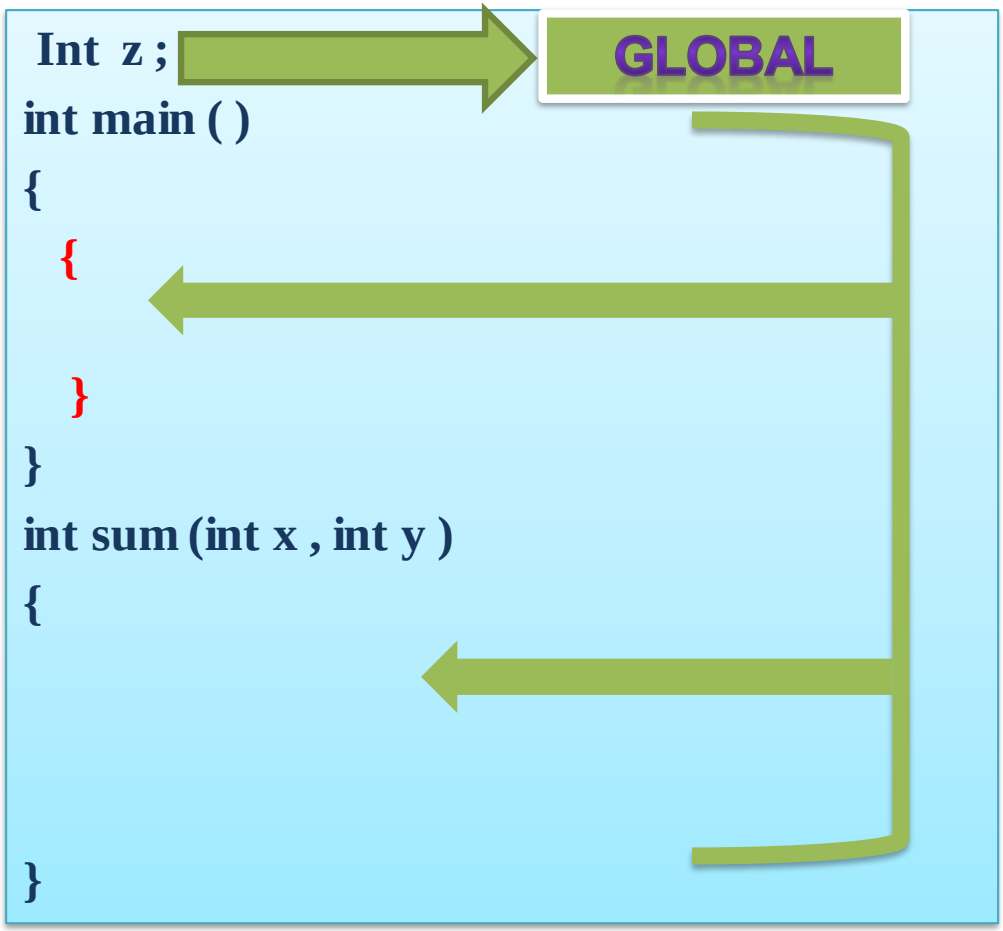




Scope of a variable

Global variable: declared outside of every function definition.

– Can be accessed from any function that has no local variables with the same name. In case the function has a local variable with the same name as the global variable ,



Scope of a variable

```
int x = 100 ;
int main ( )
{
int x= 10;
int y = 50;
cout << " Global X" << ::x  << "   Main X" << x << "     " << y;
      {
            int  z , x ;
            z=100;
            y=100;
            x= 250;
            cout <<" inner block " << x ;
      }
}
```

GLOBAL

Local

Local

Scope of a variable

- Using global variables causes side effects
  - A function that uses global variables is not independent
  - If more than one function uses the same global variable and something goes wrong ,
  - It is difficult to find what went wrong and where Problems caused in one area of the program may appear to be from another area.
  - To prevent the global variable to be modified use the **Const** Keyword.

```
Const  float pi = 3.14;
```

Static and Automatic Variables

- Automatic variable:
  - memory is allocated at block entry and de-allocated at block exit
  - By default, variables declared within a block are automatic variables

```
int  issue_ticket (int x , int y )
{
int ticket_no = 0;

 cout <<"Flight No : " <<flight_no <<"\n";
 cout <<"Ticket no : " << ++ ticket_no <<"\n";
 cout <<"Issued For: " << pname <<"\n \n \n";
}
```

Static and Automatic Variables

- Automatic variable:
  - memory is allocated at block entry and de-allocated at block exit
  - By default, variables declared within a block are automatic variables
- Static variable:
  - memory remains allocated as long as the program executes
  - Variables declared outside of any block are static variables
  - Declare a static variable within a block by using the reserved word `static .`

```
Static int x;
```

Arrays

- **Array** : is a collection of fixed number of elements, wherein all of elements have same data type.

- **Array Basics:**
  - Consecutive group of memory locations that all have the same type.
  - The collection of data is indexed, or numbered, and at starts at 0 and The highest element index is one less than the total number of elements in the array.

- **One-dimensional array**:
  - elements are arranged in list form.
- **Multi-dimensional array:**
  - elements are arranged in tabular form.

Arrays

- Syntax for declaring a one-dimensional array:

```
Datatype ArrayName [ArraySize] ;
```

ArraySize: any positive integer or constant variable.

- Example:

```
int num[5];
```

- Example: const int *size* = 5 ;

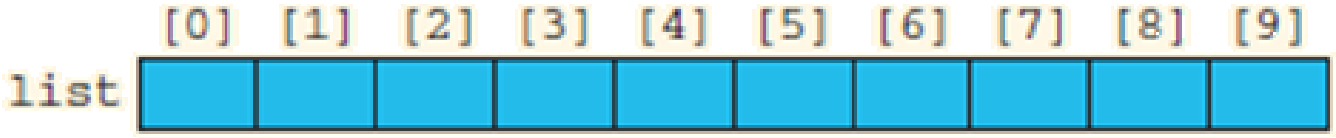
```
int num[size];
```

Arrays

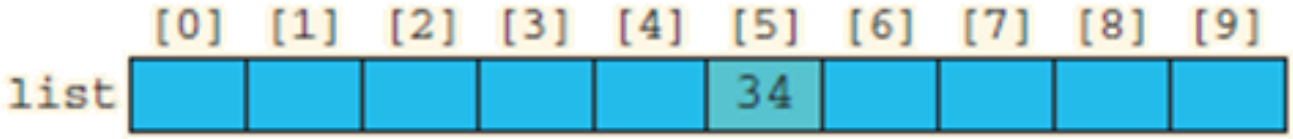
- Accessing array Elements

Arrayname [element index].

- Example: `int list[10];`



`list[5] = 34;`



`Cout << list [5];`

Array Initialization

- Consider the declaration

```
int list[10];  //array of size 10
```

- After declaring the array you can use the For .. Loop to initialize it with values submmited by the user.

- Using **for** loops to access array elements:

```
for (int i = 0; i < 10; i++)
     //process list[i]
```

- Example:

```
for (int i = 0; i < 10; i++)
     cin >> list[i];
```

- **Example** : **Write a program that ask the user to enter 10 Employee salaries and store them , then add a bonus of 10 % for each employee and print out the average salary value.**

Array Initialization

- Arrays can be initialized during declaration
  - In this case, it is not necessary to specify the size of the array
    - Size determined by the number of initial values in the braces

- Example 1: `int Items[] = {12, 32, 16, 23, 45};`

- Example 2: `int items[10]= {0};`

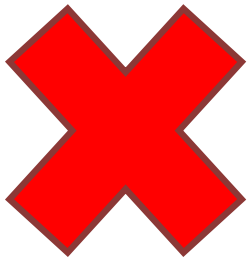
- Example 3: `int items[10]= {5, 7, 10};`

Array Initialization

```
int Arr1[5];

  int Arr2 [5];

    Arr1 = Arr2 ;
```



Array as a Parameter to Function

- Arrays are passed by reference only
- The symbol & is *not* used when declaring an array as a formal parameter
- The size of the array is usually passed as a parameter to the function.

```
Float calc_Average ( Float marks [ ] , int size )
{
float sum =0 ;

for (int I =0 ; I <size ; I ++)
    Sum += marks [ I ]  ;

 return sum / size;
}
```

- We may add keyword **const** before array name to Prevent the function from modifying the array elements.

```
Float calc_Average ( const Float marks [ ] , int size )
```

- **Example** : **Write a program that uses a function to search for an item within an array.**

```
bool find_Item( int list[ ] , int searcheditem , int size)
{
int  indx;
bool  found = false;

for ( indx = 0 ; indx < size ; indx++)
    if (list[indx] == searcheditem)
    {
       found =true ;
       break ;
    }
        return found;
}
```

Two Dimensional Array

- Used when data is provided in a table form.
- For Example , to store 4 Marks for 6 students.

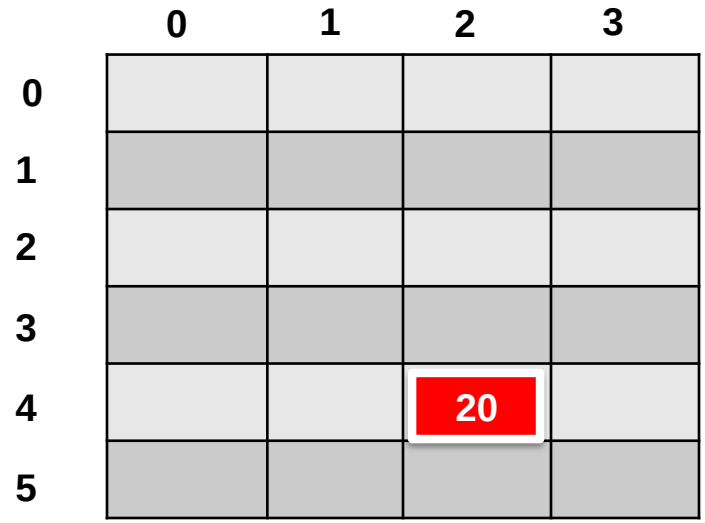
	M 1	M2	M3	M4
Student 1				
Student 2				
Student 3				
Student 4				
Student 5				
Student 6				

- **Two dimensional Array declaration**

```
Datatype ArrayName [ Rows] [Columns] ;
```

Example :

```
Float  marks [6] [4] ;
marks [4][2]= 20 ;
```



Two dimensional Array Initialization

- Consider the declaration

```
float marks[6][4];
```

- After declaring the array you can use the For .. Loop to initialize it with values submmited by the user.

- Using 2 nested **for** loops to access array elements:

```
for (int row = 0; row < 6; row++)
    for (int col = 0; col < 4; col++)
     cin >> marks[ row ][col];
```

Two dimensional Array Initialization

- Two dimensional Arrays can be initialized during declaration

- Example 1:

```
float marks[4][3] = {{20,30,35},
                     {40,45,65},
                     {60,65,75},
                     {80,65,45}};
```

- **Example** : **Write a program that build a matrix of 5 rows and 3 columns . As the use to enter the values for all the matrix items , print out the sum of all matrix items and print out the sum of the diagonal items.**

Two dimensional Array as a Parameter to Function

when declaring a two-dimensional array as a formal parameter, you can omit the size of the first dimension, but not the second;

that is, you must specify the number of columns.

```
void printMatrix( int  matrix[  ][4],  int  Rows)
  {
     int row, col;
     for (row = 0; row < Rows ; row++)
      {
      for (col = 0; col < 4; col++)
       cout << setw(5) << matrix [row] [col] ;
      }
     cout <<"\n" ;
  }
```

Struct

- **Struct**: collection of a fixed number of components (members), accessed by name
  - Members may be of different types.
  - **For Example :** to store a student record which includes different data items like (student_no , Fname, Lname , Total_Marks , GPA , …….).
  - Syntax for Defining Struct is :

```
Struct struct_name
{
        Datatype  identifier 1;
        Datatype  identifier 2;
        .
        .
        .
        Datatype  identifier n;
};
```

Struct

- **For Example :** To store an Employee data like (emp_no , fname , lname , salary , bonus , net_salary);

```
Struct   Employee
{
int        emp_no ;
string    fname ;
string    lname;
float     salary;
float     bonus ;
float    net_salary;
} ;
```

Struct

- Once , a new struct is defined , we can use it as any other data type.

```
struct  Employee
{
int        emp_no ;
string    fname ;
string    lname;
float     salary;
float     bonus ;
float     net_salary;
} ;

int  main ()
{

Employee  emp1 , emp 2;

}
```

Emp 1
Emp_no :
Fname :
Lname :
Salary :
Bonus :
Net_salary :

Emp 2
Emp_no :
Fname :
Lname :
Salary :
Bonus :
Net_salary :

To Access the Memebers of the struct , use the . Operator

```
struct  Employee
{
int        emp_no ;
string    fname ;
string    lname;
float     salary;
float     bonus ;
float     net_salary;
} ;

int  main ()
{
Employee emp1;
emp1.emp_no = 12;
emp1.fname="Ahmed";
emp1.lname="Ali";
emp1.salary=3000;
emp1.bonus=500;
emp1.net_salary= emp1.salary + emp1.bonus;
}
```

Emp 1	
Emp_no :	12
Fname :	Ahmed
Lname :	Ali
Salary :	3000
Bonus :	500
Net_salary :	3500

Assignment and Comparison

Value of one `struct` variable can be assigned to another `struct` variable of the same type using an assignment statement

Example :

```
Employee emp1 , emp2 ;
emp2 = emp1;
```

copies the contents of **emp1** into **emp2** ;

- Compare `struct` variables member-wise
  - No aggregate relational operations allowed
- To compare the values of **emp1** and **emp2**

```
if ( emp1.emp_no == emp2.emp_no &&
emp1.fname  == emp2.fname && emp1.salary == emp2.salary )
{

}
```

struct Variables and Functions

- A **`struct`** variable can be passed as a parameter by value or by reference.
- A function can return a value of type **`struct`**

```
struct distance_type
{
Int  feet ;
float  inches;
};

distance_type  Add_distances (distance_type d1 , distance_type d2 );

int main ()
{
}
distance_type  Add_distances (distance_type d1 , distance_type d2 )
{
distance_type  result;
result.feet   = d1.feet  +  d2.feet;
result.inches  = d2.inches + d2.inches ;
return result;
}
```

Arrays in structs

```
struct  Student
{
int        student_no ;
string   sname ;
float    GPA;
float    marks[3];
} ;

int  main ()
{
 Student   s1;
 s1.student_no = 120;
 s1.sname="Ahmed Ibrahim";
 s1.GPA= 3.56;
 s1.marks[0] = 80;
 s1.marks[1] = 70;
 s1.marks[2] = 90;
}
```

structs in Arrays

```
struct  Employee
{
int        emp_no ;
string   fname ;
string   lname;
float    salary;
float    bonus ;
float    net_salary;
} ;

int  main ()
{
Employee  arr[5];
 arr[0].emp_no = 12;
arr[0].fname="Ahmed";
arr[0].lname="Ali";
arr[0].salary=3000;
arr[0].bonus=500;
arr[0].net_salary= arr[0].salary  + arr[0].bonus;
}
```

structs within a struct

```
struct  employeeType
{
    string firstname;
    string middlename
    string lastname;
    string empID;
    string address1;
    string address2;
    string city;
    string state;
    string zip;
    int hiremonth;
    int hireday;
    int hireyear;
    int quitmonth;
    int quitday;
    int quityear;
    string phone;
    string cellphone;
    string fax;
    string pager;
    string email;
    string deptID;
    double salary;
};
```

→→→

```
struct nameType
{
    string first;
    string middle;
    string last;
};

struct addressType
{
    string address1;
    string address2;
    string city;
    string state;
    string zip;
};

struct dateType
{
    int month;
    int day;
    int year;
};

struct contactType
{
    string phone;
    string cellphone;
    string fax;
    string pager;
    string email;
};
```

→→→

```
struct employeeType
{
    nameType name;
    string empID;
    addressType address;
    dateType hireDate;
    dateType quitDate;
    contactType contact;
    string deptID;
    double salary;
};
```

Revision

- **Write a program that used to manage The HR data of a department that has a team of 5 employees** . The employee data like (eno , ename , job ,salary , bonus). The program should have

1. **Function to accept the data of employees and automatically set the salary according to the following formulas In case the**

   job = 'Manager' → salary = 5000

   job = 'Engineer' → salary = 3000

   job = 'Clerck' → salary = 2000

   Otherwise → salary = 1000

2. **Function to set the bonus value for a specific employee according to specific percent.**

3. **Function to print out the data of all employees.**