

Lecture 4

Not:

- The variable list may consist of one or more identifier names separated by commas. Some **valid declarations are shown here:**

Example:

```
int    i, j, k;
char   c, ch;
float  f, salary;
double d;
```

The line **int i, j, k;** both declares and defines the variables i, j and k; which instructs the compiler to create variables named i, j and k of type int.

- Variables can be initialized (assigned an initial value) in their declaration. The initializer consists of an **equal sign** followed by a **constant expression** as follows:

Example:

```
int    d = 3, f = 5;  // definition and initializing d and f.
float  A1 = 2.2;      // definition and initializes 2.
char   x = 'x';       // the variable x has the value 'x'.
```

4.1 C++ Program Structure

Let us look at a simple code that would print the words Hello World.

```
#include <iostream.h>

int main() // main() is where program execution begins.
{
    cout << "Hello World"; // prints Hello World
    return 0;
}
```

Let us look at the various parts of the above program:

1. The C++ language defines several headers, which contain information that is either necessary or useful to your program. For this program, the header **<iostream.h>** is needed for output string in the screen.
2. **int main()** : is the main function where program execution begins.
3. **//** : is a single-line comment available in C++. Single-line comments begin with **//** and stop at the end of the line.
4. **cout << " : This is my first C++ program."**; causes the message "This is my first C++ program" to be displayed on the screen.
5. **<<** : it is the send operator
6. **return 0**: terminates main() function and causes it to return the value 0 to the calling process.
7. **;** : semicolon , its used as terminator for every C++ statement.

❖ The **OUTOUT** for this program is :



Hello World

4.2 Standard Output (cout)

cout: the standard output of a program is the screen, and the C++ stream object defined to access it is cout. The **>>** **operator** is overloaded to output data items of built-in types integer, float, double, strings and pointer values.

Example:

```
cout << "Output sentence"; // prints Output sentence on screen
cout << 120;                // prints number 120 on screen
cout << x;                  // prints the content of x on screen
```

4.3 Standard input (cin)

cin: is the input stream object, its read the input value from keyboard.

>> : it is the operator use to get from operator.

endl : is used to add a new-line at the end of the line.

- You can also use cin to request more than one datum input from the user:

```
cin >> a >> b;
```

is equivalent to:

```
cin >> a;
```

```
cin >> b;
```

- In both cases the user must give two data, one for variable a and another one for variable b that may be separated by any valid blank separator: a space, a tab character or a newline.

Example:

```
#include <iostream.h>
int main()
{
    char name;
    cout << "Please enter your name: ";
    cin >> name;
    cout << "Your name is: " << name << endl;
    return 0;
}
```

Example:

```
#include <iostream.h>
int main()
{
    cout << "This is a sentence,";
    cout << "This is another sentence.";
    return 0;
}
```

- ❖ The **OUTPUT** for this program: will be shown on the screen one following the other **without any line break between them:**

This is a sentence,This is another sentence.

Example:

```
// i/o example
#include <iostream.h>

int main ()
{
    int i;
    cout << "Please enter an integer value: ";
    cin >> i;
    cout << "The value you entered is " << i;
    cout << " and its double is " << i*2 << endl;
    return 0;
}
```

- ❖ The **OUTPUT** for this program: will be shown on the screen:

Please enter an integer value: 702
The value you entered is 702 and its double is 1404.

Lecture Five

C++ Operators

An operator is a symbol that tells the compiler to perform specific mathematical or logical calculations on operands(variables).

Types of operators available in C++

- Arithmetic / Mathematical operator
- Assignment operator
- Increment Decrement operator
- Relational operator
- Logical operator
- Unary operator

Arithmetic Operator:

There are following arithmetic operators supported by C++ language:

Assume variable A holds 10 and variable B holds 20, then:

Operator	Description	Example
+	Adds two operands	A + B will give 30
-	Subtracts second operand from the first	A - B will give -10
*	Multiplies both operands	A * B will give 200
/	Divides numerator by denominator	B / A will give 2
%	Modulus Operator and remainder of after an integer division	B % A will give 0

Increment Decrement operator

Increment Decrement operators increase or decrease the operand by one value .

Example: Assume A=10, find the output result for the following expressions:

++	Increment operator, increases integer value by one	A++ will give 11
--	Decrement operator, decreases integer value by one	A-- will give 9

Assignment operator

Assignment operator is used to copy value from right to left variable.

Suppose we have:

float X = 5, Y = 2;

=	Equal sign Copy value from right to left.	X = Y, Now both X and Y have 2
+=	Plus Equal operator to increase the left operand by right operand.	X+=5 → X=X+5 will give X= 10
-=	Minus Equal operator will return the subtraction of right operand from left operand.	Y-=1 → Y= Y-1 will give Y=1
*=	Multiply Equal operator will return the product of right operand and left operand.	X *= Y → X = X * Y, X = 10
/=	Division Equal operator will divide right operand by left operand and return the quotient.	X /= Y → X = X / Y, X = 2.5
%=	Modulus Equal operator will divide right operand by left operand and return the mod (Remainder).	X %= Y is similar to X = X % Y, now X is 1

Examples:

Rewrite the equivalent statements for the following expressions and find the 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 ←

Relational Operator:

Relational operators are used for checking conditions whether the given condition is true or false. If the condition is true, it will return non-zero value, if the condition is false, it will return 0.

Suppose we have,

int X = 5, Y = 2;

Operator	Name	Description	Example
>	Greater than	Check whether the left operand is greater than right operand or not.	(X > Y) will return true
<	Smaller than	Check whether the left operand is smaller than right operand or not.	(X < Y) will return false
>=	Greater than or Equal to	Check whether the left operand is greater or equal to right operand or not.	(X >= Y) will return true
<=	Smaller than or Equal to	Check whether the left operand is smaller or equal to right operand or not.	(X <= Y) will return false
==	Equal to	Check whether the both operands are equal or not.	(X == Y) will return false
!=	Not Equal to	Check whether the both operands are equal or not.	(X != Y) will return true

Logical Operators

Logical operators are used in situation when we have more than one condition in a single if statement.

Suppose we have,

int X = 5, Y = 2;

Operator	Name	Description	Example
&&	AND	Return true if all conditions are true, return false if any of the condition is false.	if(X > Y && Y < X) will return true
	OR	Return false if all conditions are false, return true if any of the condition is true.	if(X > Y X < Y) will return true
!	NOT	Return true if condition is false, return false if condition is true.	if(!(X > Y)) will return false

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: The following example to understand all the arithmetic operators available in C++.

```
#include <iostream.h>

main()
{
int a = 21;
int b = 10;
int c ;
c = a + b;
cout << "Line 1 - Value of c is :" << c << endl ;

c = a - b;
cout << "Line 2 - Value of c is :" << c << endl ;

c = a * b;
cout << "Line 3 - Value of c is :" << c << endl ;

c = a / b;
cout << "Line 4 - Value of c is :" << c << endl ;

c = a % b;
cout << "Line 5 - Value of c is :" << c << endl ;

c = a++;
cout << "Line 6 - Value of c is :" << c << endl ;

c = a--;
cout << "Line 7 - Value of c is :" << c << endl ;

return 0;
}
```

The output for the above program is:

```
Line 1 - Value of c is :31
Line 2 - Value of c is :11
Line 3 - Value of c is :210
Line 4 - Value of c is :2
Line 5 - Value of c is :1
Line 6 - Value of c is :21
Line 7 - Value of c is :22
```

Example: find the output result for the following logical operations:

Assume a=4, b=5, c=6

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: find the output result for the following logical operations:

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

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

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

$= 1 || (2 \&\& 2)$

$= T || (T \&\& T)$

$= T || T$

$= T$

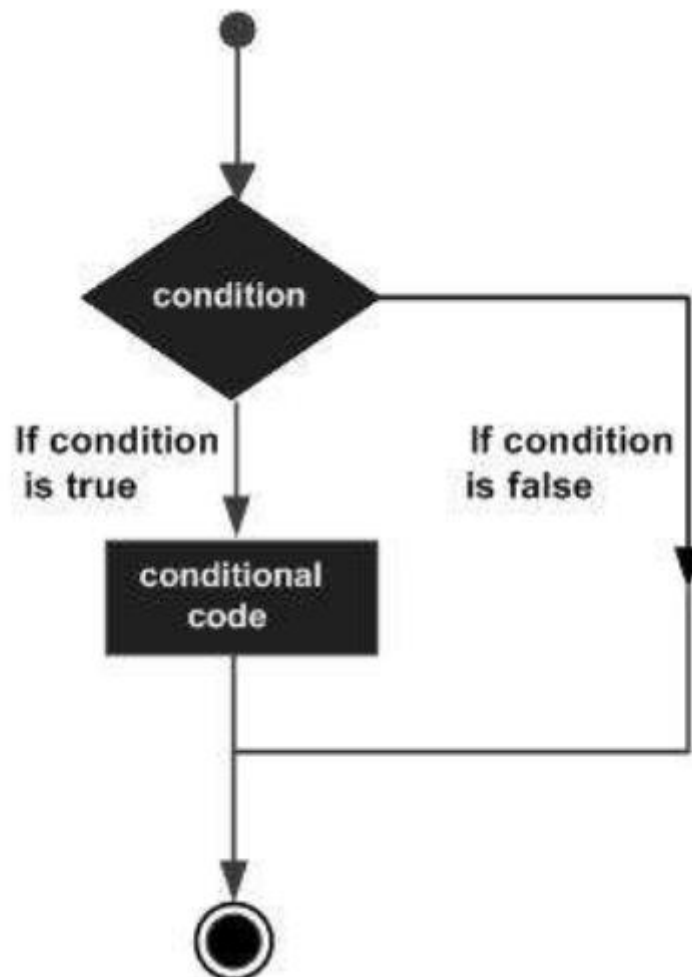
$= 1$

Lecture 6

DECISION-MAKING STATEMENTS

Decision making structures require that the programmer specify one or more conditions to be evaluated or tested by the program, along with a statement or statements to be executed if the condition is determined to be true, and optionally, other statements to be executed if the condition is determined to be false.

Following is the general form of a typical decision making structure found in most of the programming languages:



C++ programming language provides following types of decision making statements.

Statement	Description
if statement	An 'if' statement consists of a boolean expression followed by one or more statements.
if...else statement	An 'if' statement can be followed by an optional 'else' statement, which executes when the boolean expression is false.
switch statement	A 'switch' statement allows a variable to be tested

If Statement

if statement consists of a boolean expression followed by one or more statements.

Syntax

The syntax of an if statement in C++ is:

```
if(boolean_expression)
{
    // statement(s) will execute if the boolean expression is true
}
```

If the boolean expression evaluates to **true**, then the block of code inside the if statement will be executed. If boolean expression evaluates to **false**, then the first set of code after the end of the if statement (after the closing curly brace) will be executed.

Example:

Write C++ program to read a given integer value from keyboard and print the value if it is positive.

```
#include <iostream.h>

int main()
{
    int a;

    cout << "Input integer value a :";

    cin >>a;

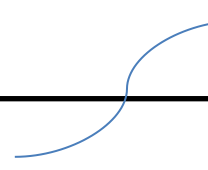
    if (a>0)
        cout<<"a is positive number" << endl;

    cout << "the value of a is :"<< a;

    return 0;
}
```

The output for the above program is :

the input value



```
Input integer value a : 10
a is positive number
the value of a is:10
```

if...else Statement

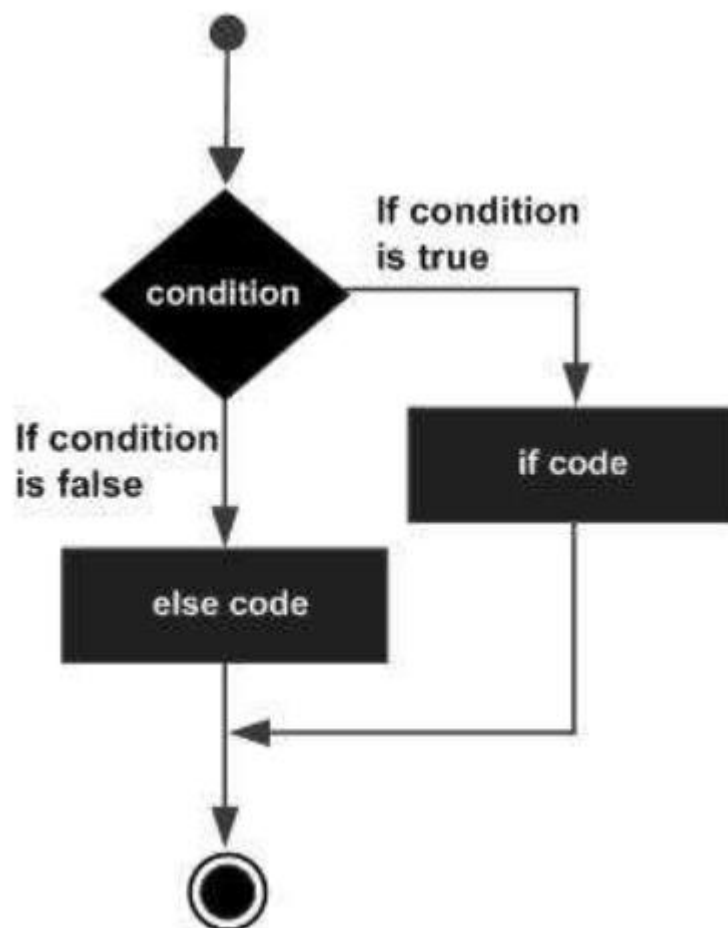
if statement can be followed by an optional **else** statement, which executes when the boolean expression is **false**.

Syntax

The syntax of an if...else statement in C++ is:

```
if(boolean_expression)
{
    // statement(s) will execute if the boolean expression is true
}
else
{
    // statement(s) will execute if the boolean expression is false
}
```

If the boolean expression evaluates to **true**, then the **if block** of code will be executed, otherwise **else block** of code will be executed.



Example:

Write C++ program to read a given integer value from keyboard and print the value if it is positive otherwise print it is negative

```
·   #include <iostream.h>
    int main()
    {
    int a;

    cout << "Input integer value a :";

    cin >>a;

    if (a>0)
        cout<<"a is positive number" << a;

    else
        cout <<"a is negative number"<< a;

    return 0;
    }
```

Example:

Write C++ program to read a given integer value from keyboard and check if the value is even or odd .

```
#include <iostream.h>

int main()
{
int a;
cout << "Input integer value a :";
cin >>a;
if (a % 2 == 0)
    cout<<"a is even number" << a;
else
    cout <<"a is odd number"<< a;
return 0;
}
```

Example

Write C++ program to calculate Z value according to the following equations:

$$Z = \begin{cases} X + 10 & \text{if } X > 0 \\ 2X + 50 & \text{if } X < 0 \end{cases}$$

```
#include <iostream.h>

int main()
{
    int X, Z;
    cout << "Input integer value X :";

    cin >>X;

    if (X > 0)
    {
        Z=X+10;
        cout<<" Z value is:" << Z;
    }
    else
    {
        Z= 2*X+50;
        cout <<"Z value is :"<< Z;
    }
    return 0;
}
```

Lecture 7

Loops

- There may be a situation, when you need to execute a block of code several number of times. In general, statements are executed sequentially: The first statement in a function is executed first, followed by the second, and so on.
- Programming languages provide various control structures that allow for more complicated execution paths.
- A loop statement allows us to execute a statement or group of statements multiple times and following is the general form of a loop statement in most of the programming languages

Loop Types:

C++ programming language provides the following type of loops to handle looping requirements.

Loop Type	Description
for loop	The <u>initialization, condition and increment/decrement all put together</u> . Initialization will be done once at the beginning of loop. Then, the condition is checked by the compiler. If the condition is false, for loop is terminated. But, if condition is true then, the statements are executed until condition is false.
while Loop	Repeats a statement or group of statements while given condition is true. It tests the condition before executing the loop body.
do...while loop	Like a 'while' statement, except that it tests the condition at the end of the loop body.
nested loops	You can use one or more loop inside any another 'while', 'for' or 'do..while' loop.

For Loop: A **for** loop is a repetition control structure that allows you to efficiently write a loop that needs to execute a specific number of times.

Syntax of For Loop

```
for ( initialization ; condition; increment )  
{  
    Statements ;  
}
```

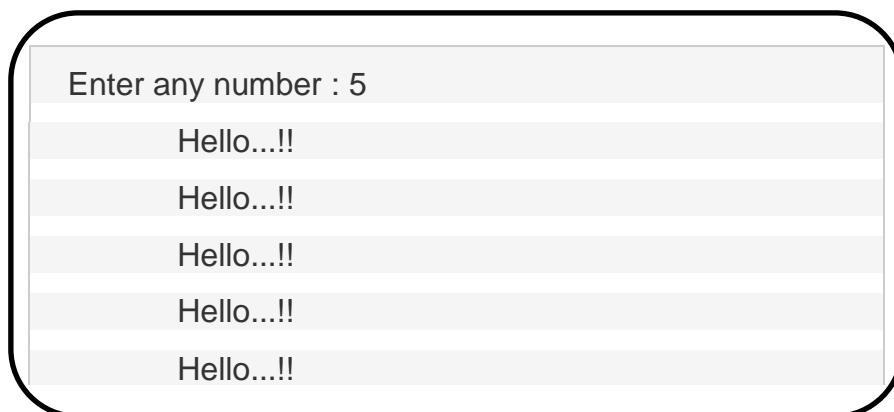
Here is the flow of control in a for loop:

- The (**initialization**) step is executed first, and only once. This step allows you to declare and initialize any loop control variables. You are not required to put a statement here, as long as a semicolon appears.
- Next, the **condition** is evaluated. If it is true, the body of the loop is executed. If it is false, the body of the loop does not execute and flow of control jumps to the next statement just after the for loop.
- After the body of the for loop executes, the flow of control jumps back up to the **increment** statement. This statement allows you to update any loop control variables. This statement can be left blank, as long as a semicolon appears after the condition.
- After the condition becomes false, the for loop terminates.

Example:

```
#include<iostream.h>  
void main()  
{  
    int a, num;  
    cout << "Enter any number : ";  
    cin >> num;  
    for (a=1;a<=num;a++)  
        cout << "\nHello...!!";    }
```

The output:



```
Enter any number : 5
Hello...!!
Hello...!!
Hello...!!
Hello...!!
Hello...!!
```

While Loop

While loop is also called entry control loop because, in while loop, compiler will 1st check the condition, whether it is true or false, if condition is true then execute the statements.

Syntax of While Loop

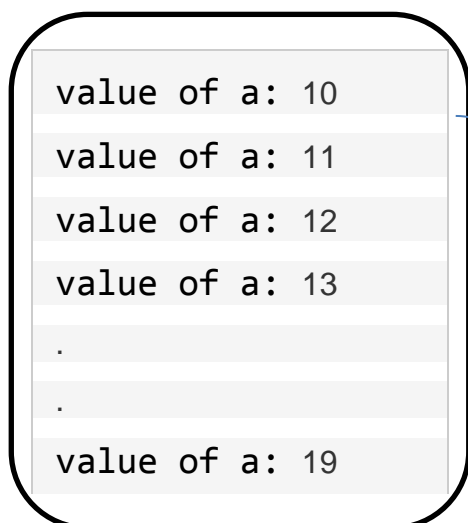
```
while ( condition )
{
    Statmentes;
    - - - - -
}
```

- Here, **statement(s)** may be a single statement or a block of statements.
- The **condition** may be any expression, and true is any non-zero value. The loop iterates while the condition is true. When the condition becomes false, program control passes to the line immediately following the loop.

Example

```
#include <iostream.h>
//using namespace std;
int main ()
{
// Local variable declaration:
int a = 10;
// while loop execution
while( a < 20 )
{
cout << "value of a: " << a << endl;
a++;
}return o;}
```

The output:



```
value of a: 10
value of a: 11
value of a: 12
value of a: 13
.
.
value of a: 19
```

يقوم هذا البرنامج بطباعة قيمة المتغير a طالما الشرط (a<20) متحقق او TRUE

Homework: trace the following c++ cods and Find the final output for these cods :

```
#include <iostream.h>

int main()
{
    int n, sum = 0;
    cout << "Enter a positive integer: ";
    cin >> n;

    for (int i = 1; i <= n; ++i) {
        sum += i;
    }

    cout << "Sum = " << sum;
}
```

```
#include <iostream.h>

int main()
{
    int n = 10 ;
    while (n > 0)
    {
        cout << n << "," ;
        -- n ;
    }
    cout << "End of program \n";
    return 0;
}
```

Exercises

1. Write C++ program to find the summation of the following series:
Sum= 1+3+5+7+ +99

(in other words: find the summation of the odd numbers, between 0 and 100)

```
#include<iostream.h>
void main( )
{
    int count = 1;
    int sum = 0;
    while ( count <= 99 )
    {
        sum = sum + count;
        count = count + 2;
    }
    cout << "sum is: " << sum << endl;
}
```

2. Write C++ program to read 10 integer numbers, and find the sum of the positive numbers only.

```
#include<iostream.h>
void main( )
{
    int num, sum = 0;
    for ( int i = 1; i <= 10; i ++ )
    {
        cout << "enter your number: ";
        cin >> num;
        if ( num > 0 )    sum = sum + num;
    }
    cout << "The sum is: " << sum;
}
```

3. Write C++ program to find the summation of the following series :

$$\sum_{i=1}^n i^2 = 1^2 + 2^2 + 3^2 + \dots + n^2$$

```
#include<iostream.h>
void main( )
{
    int i = 1, n ,sum = 0;
    cout << "enter positive number";
    cin >> n;
    while ( i <= n )
    {
        sum += i * i;
        i++;
    }
    cout << "sum is: " << sum << endl;
}
```

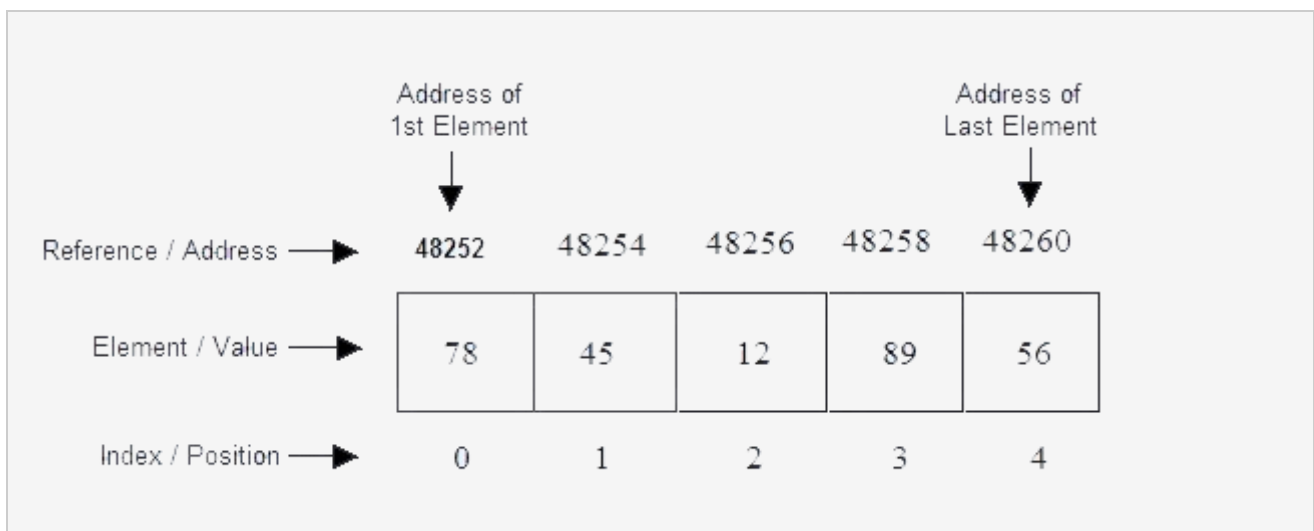
4. Write C++ program to find the summation of students marks, and its average, assume the student have 8 marks.

```
#include<iostream.h>
void main( )
{
    int mark, i, sum = 0;
    float av = 0;
    i = 1;
    while ( i <= 8 )
    {
        cout << "enter mark: ";
        cin >> mark;
        sum = sum + mark;
        i++;
    }
    cout << "sum is: " << sum << endl;
    av = sum / 8;
    cout << "average is: " << av;
}
```

Lecture 8

ARRAY

- **Array** : is a collection of similar data type. A single variable can hold only one value at a time, If we want a variable to store more than one value of same type we use array. C++ provides a data structure, **the array**, which stores a fixed-size sequential collection of elements of the same type.
- Instead of declaring individual variables, such as num0, num1, ..., and num99, you declare one array variable such as **ns** and use: ns[0], ns[1], and ..., ns[99] to represent individual variables.
- A specific element in an array is accessed by an **index**. All arrays consist of adjacent memory locations (مواقع متجاورة في الذاكرة).
- The lowest address corresponds to the first element and the highest address to the last element.



- **Address** of first element is random, address of next element depend upon the type of array. Here, the type is integer and integer takes two bytes in memory, therefore every next address will increment by two.
- **Index** of array will always starts with zero.

Declaration of Arrays: Declaration of array means creating sequential blocks of memory to hold fixed number of values.

Syntax of array declaration :

```
Data-type  Array-name [ size of Array ] ;
```

Example of array declaration :

int Array [5];	//Statement 1
float Array [10];	//Statement 2
char[50];	//Statement 3

In the above example, statement 1 will allocate memory for an integer array which will hold five values and statement 2 will allocate memory for float point array which will hold ten values. statement 3 will allocate memory for an char array which will hold fifteen values.

Initialization of Array

Initialization means assigning value to the declared Array. You can initialize C++ array elements either **one by one** or using a **single statement** as follows:

```
float A1 [5 ] = { 78.1, 45.22, 0.5, 89.14, 56.7 };
```

In the above example we are declaring and initializing an array at same time.

A1[0]	A1[1]	A1[2]	A1[3]	A1[4]
↓	↓	↓	↓	↓
78.1	45.22	0.5	89.14	56.7

If we assign another values to some elements in A1 array such that:

```
A1[2]=33.50;
```

```
A1[4] = 22.4; // assigns element number 5th in the array a value of 22.
```

Then the array A1 will be :

A1[0]	A1[1]	A1[2]	A1[3]	A1[4]
↓	↓	↓	↓	↓
78.1	45.22	33.50	89.14	22.4

Few key notes:

- Arrays have 0 as the first index not 1. In this example, `A1[0]` is the first element.
- If the size of an array is `n`, to access the last element, `(n-1)` index is used. In this example, `A1[4]` is the last element.
- Suppose the starting address of `A1[0]` is 2120. Then, the next address of `A1[1]`, will be 2124, and address of `A1[2]` will be 2128 and so on. It's because the size of a float is 4 bytes.

Example:

```
#include<iostream.h>
void main()
{
    int array [5];           // declaration of array
    int i;
    for(i=0;i<5;i++)
    {
        cout << "\nEnter any number : ";
        cin >> array [i];    //Input array from user.
    }
    for(i=0;i<5;i++)         //Output array to screen.
        cout << array [i];
}
```

Output :

Enter any number : 78

Enter any number : 45

Enter any number : 12

Enter any number : 89

Enter any number : 56

78, 45, 12, 89, 56, **Pointer and Array**

Question: //--- Print array in reverse order

Example:

```
#include<iostream.h>
void main()
{
int n[ 10 ]; // n is an array of 10 integers
int i ;
//initialize elements of array n to 0
for (i = 0; i < 10; i++)
n[ i ] = i + 100;           // set element at location i to value i + 100

for (i = 0; i < 10; i++)    // output each array element's value
    cout<<n[i]<<" ";
}
Output: ??
```

Example:

```
#include <iostream.h>
```

```
int foo [7] = {16, 2, 77, 40, 12071, 1, 33};
```

```
int n, result=0;
```

```
int main ()
```

```
{
```

```
    for ( n=0 ; n<7 ; n ++)
```

```
    {
```

```
        result += foo[n];
```

```
    }
```

```
    cout << result;
```

```
    return 0;
```

```
}
```

Output: ??

Lecture 9

Two Dimension Array

The simplest form of the multidimensional array is the two-dimensional array. A two-dimensional array is, in essence, a list of one-dimensional arrays. To declare a two-dimensional integer array of [x, y] you would write something as follows:

Syntax:

```
type arrayName [ x ][ y ];
```

Where **type** can be any valid C++ data type and **arrayName** will be a valid C++ identifier.

- A two-dimensional array can be think as a table, which will have x number of rows and y number of columns.
- A two-dimensional array **a**, which contains three rows and four columns can be shown as below:

	Column 0	Column 1	Column 2	Column 3
Row 0	a[0][0]	a[0][1]	a[0][2]	a[0][3]
Row 1	a[1][0]	a[1][1]	a[1][2]	a[1][3]
Row 2	a[2][0]	a[2][1]	a[2][2]	a[2][3]

- Thus, every element in array **a** is identified by an element name of the form **a[i][j]**, where **a** is the name of the array, and **i** and **j** are the subscripts that uniquely identify each element in **a**.
- Total number of elements that can be stored in a multidimensional array can be calculated by multiplying the size of all the dimensions. For example:
The array : `int x[10][20]` can store total $(10*20) = 200$ elements.

Example :

```
int    a[3][5] ; // declaration of an integer array a  with 3 rows and 5 columns
```

```
float  tab[2][6] ; // declaration of an float array a  with 2 rows and 6 columns
```

Initializing Two-Dimensional Arrays

Multi dimensioned arrays may be initialized by specifying bracketed values for each row. Following is an array with 3 rows and each row have 4 columns.

```
int a[3][4] = {
    {0, 1, 2, 3} , /* initializers for row indexed by 0 */
    {4, 5, 6, 7} , /* initializers for row indexed by 1 */
    {8, 9, 10, 11} /* initializers for row indexed by 2 */
};
```

0	1	2	3
4	5	6	7
8	9	10	11

- The nested braces, which indicate the intended row, are optional. The following initialization is equivalent to previous example :

```
int  a[3][4] = {0,1,2,3,4,5,6,7,8,9,10,11};
```

Accessing Two-Dimensional Array Elements

An element in 2-dimensional array is accessed by using **row index and column index of the array**. For example :

The element in the position a [2][2] is 10

The element in the position a [1][3] is 7

- To output all the elements of a Two-Dimensional array we can use nested for loops. We will require two for loops. One to traverse the rows and another to traverse columns.

Example: write c++ program to print the elements of the array buf [5][2], which initialized as : buf [5][2]={ 0, 0, 1, 2, 2, 4, 3, 6, 4, 8 }.

```
#include <iostream.h>

int main ()
{
    int a[5][2] = { {0,0}, {1,2}, {2,4}, {3,6},{4,8}}; // array with 5 rows and 2 col.

    for ( int i = 0; i < 5; i++ )    // output each array element's value
        for ( int j = 0; j < 2; j++ )
            cout << a[i][j]<< endl;
}
```

Example : Write c++ program to read an integer elements of array with size(4x4), then find the summation of these elements, finally print these elements.

```
#include<iostream.h>

void main ( )
{
    int a [ 4 ] [ 4 ];
    int i , j, sum = 0;
    for ( i = 0 ; i < 4; i++ )
        for ( j = 0 ; j < 4; j++ )
            cin >> a [ i ] [ j ];

    for ( i = 0 ; i < 4; i++ )
        for ( j = 0 ; j < 4; j++ )
            sum += a [ i ] [ j ];
    cout << "summation is: " << sum << endl;

    for ( i = 0 ; i < 4; i++ )
    {
        for ( j = 0 ; j < 4; j++ )
            cout << a [ i ] [ j ];
        cout << endl;
    }
}
```

Lecture 10

Functions

Function is a group of statements that together perform a task. Every C++ program has at least one function, which is **main()**, and all the most trivial programs can define additional functions.

You can divide up your code into separate functions. How you divide up your code among different functions is up to you, but logically the division usually is such that each function performs a specific task.

Function declaration tells the compiler about a function's name, return type, and parameters. A function **definition** provides the actual body of the function.

Defining a Function

The general form of a C++ function definition is as follows:

```
return_type function_name ( parameters )
{
    body of the function
}
```

- C++ function definition consists of a **function header** and a **function body**.

Here are all the parts of a function:

- **Return Type**: A function may return a value. The **return_type** is the data type of the value the function returns. Some functions perform the desired operations without returning a value. In this case, the **return_type** is the keyword **void**.
- **Function Name**: This is the actual name of the function. The function name and the parameter list together constitute the function signature.

- **Parameters:** When a function is invoked, you pass a value to the parameter. This value is referred to as actual parameter or argument. The parameter list refers to the type, order, and number of the parameters of a function. **Parameters are optional; that is, a function may contain no parameters.**
- **Function Body:** The function body contains a collection of statements that define what the function does.

Example:

Following is the source code for a function called **max()**. This function takes two parameters num1 and num2 and returns the maximum between the two:

(1) **Defining a Function**

```
// This function returning the max between two numbers
int max(int num1, int num2)
{
int result;    // local variable declaration

if (num1 > num2)
result = num1;
else
result = num2;
return result;

}
```

(2) Function Declaration

- A function declaration has the following parts:

return_type **function_name** (**parameter list**);

- For the above defined function **max()**, following is the function declaration:

int max (int num1, int num2) ;

- Parameter names are not important in function declaration only their type is required, so following is also valid declaration:

int max (int, int) ;

- you should declare the function at the top of the file calling the function.

(3) Calling a Function

- To use a function, you will have to call or invoke that function.
- When a program calls a function, program control is transferred to the called function.
- A called function performs defined task and when it's return statement is executed or when its function-ending closing brace is reached, it returns program control back to the main program.
- To call a function, you simply need to pass the required parameters along with function name, and if function returns a value, then you can store returned value.

For example:

```

#include <iostream.h>

int max(int num1, int num2);    // function declaration

int main ()
{
    int a = 100, b = 200 , ret; // local variable
    declaration:

    ret = max(a, b);            // calling a function max

    cout << "Max value is : " << ret << endl;

    return 0;
}

// This function returning the max between two numbers

int    max(int num1, int num2)

{
    int result;    // local variable declaration

    if (num1 > num2)
        result = num1;
    else
        result = num2;

    return result;
}

```

Example : Write C++ program to calculate the **squared value** of a given integer number passed from the main function. Use this function in the program to calculate the squares of numbers from 1 to 10.

```
#include <iostream.h>

int square(int y);    // function declaration

void main( )
{
    int x;
    for ( x=1; x <= 10; x++ )
        cout << square ( x ) << endl;
}

// This function to calculate the square value of number
int square ( int y )
{
    int z;
    z = y * y;
    return ( z );
}
```

Example: Write a function to find the largest integer number among three integer numbers entered by the user in the main function.

```

#include <iostream.h>
int max(int , int, int) ; // function declaration

void main( )
{
    int largest,x1,x2,x3;
    cout<<"Enter 3 integer numbers:";
    cin>>x1>>x2>>x3;
    largest=max(x1,x2,x3);
    cout<<largest;
}

int max(int y1, int y2, int y3)
{
    int big;
    big=y1;
    if (y2>big) big=y2;
    if (y3>big) big=y3;
    return (big);
}

```

Example: Write C++ program to create a simple calculator to (add, subtract, multiply and divide) using function.

```

# include <iostream.h>

float calc( float x, float y, char op) ; // function declaration


int main()
{
    char op;
    float num1, num2, result;
    cout << "Enter operator either + or - or * or /: ";
    cin >> op;
    cout << "Enter two numbers: ";
    cin >> num1 >> num2;
    result= calc(num1, num2, op); // calling function
    cout<<"the output is :"<< result;
}


float calc( float x, float y, char op) // function definition
{
    float z;
    if (op == '+' ) {
        z= x + y;
        return z;    }
    if (op == '-' ) {
        z= x - y;
        return z;    }
    if (op == '*' ) {
        z= x * y;
        return z;    }
    if (op == '/' ) {
        z= x / y;
        return z;    }
    else {
        cout << "Error! operator is not correct";
        return 0;
    } }

```

Scope of variables

Scope is a region of a program. **Variable Scope** is a region in a program where a variable is declared and used. Variables are thus of two types depending on the region where these are declared and used.

1. Local Variables :

Variables that are declared inside a function or block are local variables. They can be used only by statements that are inside that function or block of code. Local variables are not known to functions outside their own. Following is the example using local variables .

```
#include <iostream>
using namespace std;
int main () {
    int a, b; // Local variable declaration:
    int c;
    a = 10;
    b = 20;
    c = a + b;
    cout << c;
    return 0;
}
```

2. Global Variables

Global variables are defined outside of all the functions, usually on top of the program. The global variables will hold their value throughout the life-time of your program.

A global variable can be accessed by any function. That is, a global variable is available for use throughout your entire program after its declaration. Following is the example using global and local variables.

```
#include <iostream.h>

void func1(void); // declaration function

int g =10;        // Global variable declaration:

int main(){

    func1();    // call function

    g = 30;

    cout << g << endl;

    return 0;

}

void func1()    // function definition
{

    g = 20;

    cout << g << endl;

}
```