University of Technology
الجامعة التكنولوجية

Computer Science Department
قسم علوم الحاسوب

البرمجة المهيكلة

أ.م.د بشار سعدون ، م. ياسر منذر
م.د انمار علي ، م. رشا اسماعيل

cs.uotechnology.edu.iq

LECTURE 11

1. Functions:

A function is a set of statements designed to accomplish a particular task. Experience has shown that the best way to develop and maintain a large program is to construct it from smaller pieces or (modules). Modules in C++ are called functions.

Functions are very useful to read, write, debug and modify complex programs. They can also be easily incorporated in the main program. In C++, the main() itself is a function that means the main function is invoking the other functions to perform various tasks. The main advantages of using a function are:

- ❖ Easy to write a correct small function.
- ❖ Easy to read, write, and debug a function.
- ❖ Easier to maintain or modify such a function.
- ❖ Small functions tend to be self documenting and highly readable.
- ❖ It can be called any number of times in any place with different parameters.

2. Defining a Function

A function definition has a name, parentheses pair containing zero or more parameters and a body. For each parameter, there should be a corresponding declaration that occurs before the body. Any parameter not declared is taken to be an integer by default. The general format of the function definition is :

General Form of Function:

```
return-type function-name ( parameters-list )
    {
        (body of function)
        statement1 ;
        statement2 ;
             :
        statement-n ;
        (return something)
    }
```

The type of the function may be int, float, char, etc. It may be declared as type (void), which informs the compiler not to the calling program. For example:

Void function_name (---)

Int function_name (---)

Any variable declared in the body of a function is said to be local to that function. Other variables which are not declared either as arguments or in the function body are considered "global" to the function and must be defined externally. For example

Void square (int a, int b) → a,b are the formal arguments.

Float output (void) → function without formal arguments

Example 1:	**Example 2:**

```
void printmessage ( )
{
    cout << "University of Technology";
}

void main ( )
{
     printmessage( );
}
```
	```
	int max (int a, int b)
	{
	    int c;
	    if (a > b)  c = a;
	      else   c = b;
	    return (c);
	}

	void main ( )
	{
	     cout << max (5, 6);
	}
	```

3. Return Statement:

The keyword return is used to terminate function and return a value to its caller. The return statement may also be used to exit a function without returning a value. The return statement may or may not include an expression. Its general syntax is:

Return;

Return (expression);

The return statements terminate the execution of the function and pass the control back to the calling environment.

Example 1

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

```
#include<iostream.h>

int square ( int y )
{
        int z;
        z = y * y;
        return ( z );
}

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

Example 2

Write C++ program using function to calculate the average of two numbers entered by the user in the main program:

```
#include<iostream.h>

float aver (int x1, int x2)
{
        float z;
        z = ( x1 + x2) / 2.0;
        return ( z);
}

void main( )
{
        float x;
        int num1,num2;
        cout << "Enter 2 positive number \n";
        cin >> num1 >> num2;
        x = aver (num1, num2);
        cout << x;
}
```

Example 3

Write C++ program, using function, to find the summation of the following series: $\sum_{i=1}^{n} i^2 = 1^2 + 2^2 + 3^2 + \cdots + n^2$

```
#include<iostream.h>
int summation ( int x)
{
        int i = 1, sum = 0;
        while ( i <= x )
        {
                sum += i * i ;
                i++;
        }
        return (sum);
}
void main ( )
```

```
{
        int n ,s;
        cout << "enter positive number";
        cin >> n;
        s = summation ( n );
        cout << "sum is: " << s << endl;
}
```

Example 4

Write a function to find the largest integer among three integers entered by the user in the main function.

```
#include <iostream.h>
int max(int y1, int y2, int y3)
{
        int big;
        big=y1;
        if (y2>big) big=y2;
        if (y3>big) big=y3;
        return (big);
}
void main( )
{
        int largest,x1,x2,x3;
        cout<<"Enter 3 integer numbers:";
        cin>>x1>>x2>>x3;
        largest=max(x1,x2,x3);
        cout<<largest;
}
```

Example 5

Write program in C++, using function, presentation for logic gates (**AND, OR ,NAND, X-OR,NOT**) by in A,B enter from user.

```
#include<iostream.h>
 void ANDF(int,int);
 void ORF(int,int);
 void XORF(int,int);
 void NOTF(int);

void main()
{ char s;int a,b;
cout<<"Enter the value A,B :";
cin>>a>>b;
```

```
cout<<"Enter the select value \n";
cout<<"\ta--(AND gate)\n\to--(OR gate)\n\tx--(X-OR)\n\tn--(NOT
gate)\n\te--(<<EXIT>> :";
cin>>s;
switch(s)
        {
        case 'a':ANDF(a,b);break;
        case 'o':ORF(a,b);break;
        case 'x':XORF(a,b);break;
        case 'n':NOTF(a);cout<<" ";NOTF(b);break;
        case 'e':break;
        deafult:cout<<":bad choose";

}

}
void ANDF(int a,int b)
 {
 cout<<(a&&b);
 }
 void ORF(int a,int b)
 {
 cout<<(a||b);
 }
 void XORF(int a,int b)
 {
 cout<<(a^b);
 }
 void NOTF(int a)
 {
 cout<<(!a);
 }
```

4. Passing Parameters:

There are two main methods for passing parameters to a program:

1) **passing by value**, and 2) **passing by reference.**

A- Passing by Value:

When parameters are passed by value, a copy of the parameters value is taken from the calling function and passed to the called function. The original variables inside the calling function, regardless of changes made by the function to it are parameters will not change. All the pervious examples used this method.

B- Passing by Reference:

When parameters are passed by reference their addresses are copied to the corresponding arguments in the called function, instead of copying their values. Thus pointers are usually used in function arguments list to receive passed references.

This method is more efficient and provides higher execution speed than the call by value method, but call by value is more direct and easy to use.

Example 6:

The following program illustrates passing parameter by reference.

```
#include <iostream.h>
void swap(int *a,int *b)
{
        int t;
        t=*a;
        *a=*b;
        *b=t;
}
void main( )
{
        int x=10;
        int y=15;
        cout<<"x before swapping is:"<<x<<"\n";
        cout<<"y before swapping is:"<<y<<"\n";
        swap(&x,&y);
        cout<<"x after swapping is:"<<x<<"\n";
        cout<<"y after swapping is:"<<y<<"\n";         }
```

LECTURE 12

1. Types of Functions:

The user defined functions may be classified in the following three ways based on the formal arguments passed and the usage of the return statement, and based on that, there are three of user defined functions:

1. A function is invoked without passing any formal argument from the calling portion of a program and also the function does not return back any value to the called function.
2. A function is invoked with formal arguments from the calling portion of a program but the function does not return back any value to the calling portion.
3. A function is invoked with formal arguments from the calling portion of a program which return back a value to the calling environment.

Here are two programs that find the square of the number using and not using a return statement.

Example 1:

```
# include <iostream.h>
Void main()
{
Void square (int);
Int max;
Cout<<"Enter a value for n ?\n";
Cin>>max;
For (int i=0;i<=max-1;++i)
Square (i)
}
Void square(int n)
{
Float value;
Value=n*n;
Cout<<"i="<<n<<"square="<<value<<endl;
}
```

```
# include <iostream.h>
Void main()
{
float square (float);
float I,max,value;
max=1.5;
i=-1.5;
while (i<=max) {
value=square(i);
Cout<<"i="<<i<<"square="<<value<<endl;
I=i+0.5;
}
}
Float square(float n)
{
Float value;
Value=n*n;
Return (value);
}
```

Example 2:

write C++ program, using function to find the sumation of the given series: Sum=x-$(x^3)/3!+(x^5)/5!- \ldots (x^n)/n!$

```
#include <iostream.h>
Void main(void)
{
Long int fact (int);
Float power(float,int);
Float sum,temp,x,pow;
Int sign,I,n;
Longint factorial;

Cout<<"Enter a value for n?"<<endl;
Cin>>n;
Cout<<"Enter a value for x ?"<<endl;
Cin>>x;
I=3;  sum=x;   sign=1;
While (i<=n)  {
Factval=fact(i);
Pow=power(x,i);
Sign=(-1)*sign;
Temp=sign*pow/factval;
Sum=sum+temp;
I=i+2;
}
Cout<<"sum of series ="<<sum;
}


Long int fact (int max)
{
Long intvalue;
Value=1;
For(int i=1;i<=max;++i)
Value=value*I;
Return (value);
}

Float power (float x, int n)
{
Float value2;
Value2=1;
For(int j=1;j<=n;++j)
Value2=value2*x;
Return(value2);
}
```

2. Actual and Formal Arguments:

The arguments may be classified under two groups, actual and formal arguments:

(a) **Actual arguments:** An actual argument is a variable or an expression contained in a function call that replaces the formal parameter which is a part of the function declaration. Sometimes, a function may be called by a portion of a program with some parameters and these parameters are known as the actual arguments.

(b) **Formal arguments:** are the parameters present in a function definition which may also be called as dummy arguments or the parametric variables. When the function is invoked, the formal parameters are replaced by the actual parameters. Formal arguments may be declared by the same name or by different names in calling a portion of the program or in a called function but the data types should b e the same in both blocks

For example:

```
# include <iostream.h>
Void main()
{
Int x,y;
Void output (int x, int y);          // function declaration
—
—
Output (x,y);                   // x and y are actual arguments
}
Void output (int a, int b)           // forma or dummy arguments
{
                    // body of function
 }
```

3. Local and Global variables:

The variables in general bay be classified as local or global variables.

(a) **Local variables:** Identifiers, variables and functions in a block are said to belong to a particular block or function and these identifiers are known as the local parameters or variables. Local variables are defined inside a function block or a compound statement. For example,

```
Void func (int I, int j)
{
Int k,m;                // local variables
……                      // bodyofthe function
}
```

Local variables are referred only the particular part of a block or a function. Same variable name may be given to different parts of a function or a block and each variable will be treated as a different entity.

(b) Global variables: these are variables defined outside the main function block. These variables are referred by the same data type and by the same name through out the program in both the calling portion of the program and in the function block.

Example 3:

A program to find the sum of the given two numbers using the global variables.

```
#include <iostream.h>
Int x;
Int y=5;

void main( )
{
        X=10;
        Void sum(void);
        Sum();
 }
Void sum(void)
{
        Int sum;
        Sum=x+y;
        Cout<<"x="<<x<<endl;
        Cout<<"y="<<y<<endl;
        Cout<<"sum="<<sum<<endl;
}
```

4. Recursive Functions:

A function which calls itself directly or indirectly again and again is known as the recursive function. Recursive functions are very useful while constructing the data structures like linked lists, double linked lists, and trees. There is a distinct difference between normal and recursive functions. A normal function will be invoked by the main function whenever the function name is used, where as the recursive function will be invoked by itself directly or indirectly as long as the given condition is satisfied. Forexample,

```
# include <iostrem.h>
Void main(void)
{
Void func1();  //function declaration
_____
_____
Func1();  //function calling
}
Void func1()  //function definition
{
____
____
Func1();  //function calls recursively
}
```

Example 4:

💻 A program to find the sum of the given non negative integer numbers using a recursive function sum=1+2+3+4+…+n

```
#include <iostream.h>
Void main(void)
{
Int sum(int);
Int n,temp;
Cout<<"Enter any integer number"<<endl;
Cin>>n;
Temp=sum(n);
Cout<<"value="<<n<<"and its sum="<<temp;
}
Int sum(int n)  //recursive function
{
Int sum(int);   //local function declaration
Int value=0;
If (n==0)
Return (value);
Else
Value=n+sum(n-1);
Return (value);
}
```

The output of the above program:

```
Enter any integer number
Value = 11 and its sum=66
```

The following illustrations will be helpful to understand the recursive function

For value 1	For value 2	For value 3
=1+sum(1-1)	=2+sum(2-1)	=3+sum(3-1)
=1+0	=2+1+sum(1-1)	=3+sum(2-1)
=1	=3	=3+2+1+sum(1-1)
		=6

Example 5:

A program to find the factorial (n!) of the given number using the recursive function.Its the product of all integers from 1 to n (n is non negative) (so n!=1 if n=0 and n!=n(n-1) if n>0)

```
#include <iostream.h>
Void main(void)
{
Long int fact (long int);
Int x,n;
Cout<<"Enter any integer number"<<endl;
Cin>>n;
X=fact(n);
Cout<<"value="<<n<<"and its factorial=";
Cout<<x<<endl;
}
Long int fact (long int n)  //recursive function
{
Long int fact(long int);  //local function declaration
Int value =1;
If (n==1)
Return(value)
Else
{value=n*fact(n-1);
Return(value);
} }
```

The output of the above program:

```
Enter any integer number

Value = 5 and its factorial=120
```

The following illustrations will be helpful to understand the recursive function

For value 1	**For value 2**	**For value 3**
=1*fact(1-1)	**=2*fact(2-1)**	**=3*fact(3-1)**
=1	**=2*1**	**=3*2*fact(2-1)**
	=2	**3*2*1**
		=6

WORK SHEET (5)

Functions

Q1: Write a C++ program, using function, to counts uppercase letter in a 20 letters entered by the user in the main program.

Q2: Write a C++ program, using function, that reads two integers (feet and inches) representing distance, then converts this distance to meter.
Note: 1 foot = 12 inch
1 inch = 2.54 Cm
i.e.:
Input: feet: 8 or inches: 9

Q3: Write a C++ program, using function, which reads an integer value (T) representing time in seconds, and converts it to equivalent hours (hr), minutes (mn), and seconds (sec), in the following form:
hr : mn : sec

i.e.:
Input: 4000
Output: 1 : 6 : 40

Q4: Write a C++ program, using function, to see if a number is an integer (odd or even) or not an integer.

Q5: Write a C++ program, using function, to represent the permutation of n.

Q6: Write a C++ program, using function, to inputs a student's average and returns 4 if student's average is 90-100, 3 if the average is 80-89, 2 if the average is 70-79, 1 if the average is 60-69, and 0 if the average is lower than 60.

Q7: The Fibonacci Series is: 0, 1, 1, 2, 3, 5, 8, 13, 21, … It begins with the terms 0 and 1and has the property that each succeeding term is the sum of the two preceding terms. Write a C++ program, using function, to calculate the nth Fibonacci number.

Q8: Write a C++ program, using function, to calculate the factorial of an integer entered by the user at the main program.

Q9: Write a C++ program, using function, to evaluate the following equation:

$$z = \frac{x! - y!}{(x - y)!}$$

Q10: Write a C++ program, using function, to test the year if it's a leap or not.

Note: *use y % 4 == 0 && y % 100 != 0 :: y % 400 ==0*

Q11: Write a C++ program, using function, to find x^y.

Note: *use **pow** instruction with **math.h** library.*

Q12: Write C++ program, using function, to inverse an integer number:

For example: 765432 → 234567

Q13: Write C++ program, using function, to find the summation of student's marks, and it's average, assume the student have 8 marks.

Q14: Write C++ program, using function, to convert any char. From capital to small or from small to capital.

Q15: Write C++ program using recursive function to find the power of n numbers.

LECTURE 13

1. Arrays:

An array is a consecutive group of homogeneous memory locations. Each element (location) can be referred to using the array name along with an integer that denotes the relative position of that element within the array. The data items grouped in an array can be simple types like int or float, or can be user-defined types like structures and objects.

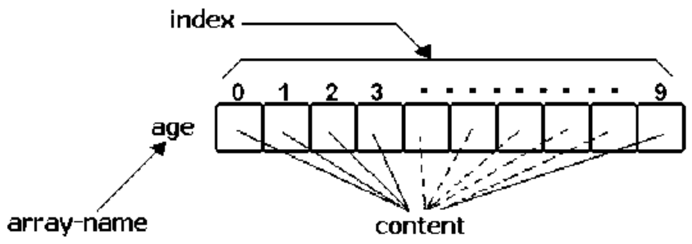
2. Array of One Dimension:

It is a single variable specifies each array element. The declaration of one dimensional arrays is:

General Form of 1D-Array:
data-type Array-name [*size*];

Examples:

```
int    age [10];
int    num [30];
float  degree[5];
char    a [15];
```



The item in an array are called elements (in contrast to the items in a structure which are called members). The elements in an array are of the same type only the values vary.

3. Initializing Array Elements:

- The first element of array age:
```
age [0] = 18;
```

- The last element of array age:
```
age [9] = 19;
```

- All elements of array age:
```
age [9] = { 18, 17, 18 ,18 ,19, 20 ,17, 18 ,19 };
```

- int x [ ] = { 12, 3, 5, 0, 11, 7, 30, 100, 22 };

- int y [10] = { 8, 10, 13, 15, 0, 1, 17, 22};

4. Accessing Array Elements:

We access each array element by written name of array, followed by brackets delimiting a variable (or constant) in the brackets which are called the array index.

- Accessing the first element of array num to variable x:
```
x = num [0];
```

- Accessing the last element of array num to variable y:
```
y = num [9];
```

- cout << num [0] + num [9];

- num [0] = num [1] + num[2];

- num [7] = num [7] + 3;   ↔ num [7] += 3;

5. Read / Write / Process Array Elements:

- cout << num [4];

- if (num [5] > 5)
```
      cout << "greater";
```

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

- for (int i=0; i<10; i++)
```
      cout << num[ i ];
```

- for (int i=9; i>=0; i++)
```
      cout << num[ i ];
```

- sum=0;
```
  for (int i=0; i<10; i++)
      sum = sum + num[ i ];
```

Example 1

💻 Write C++ program to display 2nd and 5th elements of array distance:

```
#include<iostream.h>

void main( )
{
          double  distance[ ] = { 23.14, 70.52, 104.08, 468.78, 6.28};
           cout << “2nd element is: “ << distance[1] << endl;
           cout << “5th element is: “ << distance[4];
}
```

Example 2

💻 Write C++ program to read 5 numbers and print it in reverse order:

```
#include<iostream.h>

void main( )
{
        int a [5];
        cout << "Enter 5 numbers \n";
        for ( int i =0; i <5; i++ )
          {
              cout << i << “: “;
              cin >> a [ i ];
              cout << “\n”;
          }
        cout << “The reverse order is: \n”;
        for ( i =4; i >=0; i-- )
             cout << i << “: “ << a [ i ] << endl;
}
```

Example 3

💻 Write C++ program, to find the summation of array elements:

```
#include<iostream.h>

void main (  )
{
         int const L = 10;
         int a [L];
         int sum = 0;
         cout << "enter 10 numbers \n";
         for ( int i  =0; i  <L; i++ )
            {
               cout << “enter value “ << i << “: “;
               cin >> a [ i ];
```

```
                sum += a [ i ];
            }
        cout << "sum is: " << sum << endl;
}
```

Example 4

Write C++ program, to find the minimum value in array of 8 numbers:

```
#include<iostream.h>

void main ( )
{       int n = 8;          int a [  ] = { 18, 25, 36, 44, 12, 60, 75, 89 };
        int min = a [ 0 ];
        for ( int i  = 0; i  < n; i++ )
        if ( a [ i ] < min )       min = a [ i ];
        cout << "The minimum number in array is: " << min;      }
```

Example 5

Write C++ program, to give the number of days in each month:

```
#include<iostream.h>
void main ( )
{
Int month, day, total_days;
Int days_per_month[12]={31,28,31,30,31,30,31,31,30,31,30,31}
Cout<<"\n Enter month(1 to 12):";
Cin>>month;
Cout<<"enter day(1 to 31):";
Cin>>day;
Total_days=day;
For (int j=0;j<month-1;j++)
Total_day+=day_per_month[j];
Cout<<"Total days from start of year is:"<<total_days;
}
```

Example 6

Write C++ program, using function, to find (search) X value in array, and return the index of it's location:

```
#include<iostream.h>

int search( int a[ ], int y)
{
   int  i= 0;
   while ( a [ i ] != y )
     i++;
```

```
    return ( i );
}

void main (  )
{
          int  X, f;
          int  a [ 10 ] = { 18, 25, 36, 44, 12, 60, 75, 89, 10, 50 };
          cout << “enter value to find it: “;
          cin >> X;
          f= search (a, X);
          cout << “the value “ << X << “ is found in location “<< f;
}
```

Example 7

Write C++ program, to split the odd numbers and even numbers of one array into two arrays:

a = [1, 2, 3, 4, 5, 6, 7, 8, … , 20]
aodd = [1, 3, 5, 7, … ,19]
aeven = [2, 4, 6, 8, … , 20]

```
#include<iostream.h>

void main ( )
{
        int a [ 20 ]= { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20 };
        int aodd[20], aeven [20];
        int i ,o=0, e=0;
        for ( i=0 ; i<20; i++ )
            if (a[i] % 2 !=0)
              {
                  aodd[o]=a[i];
                  o=o+1;
              }
             else
             {
                  aeven[e]=a[i];
                  e=e+1;
             }

        for ( i=0 ; i<o; i++ )
             cout<<aodd[i]<<"  ";
        cout<<endl;
        for ( i=0 ; i<e; i++ )
             cout<<aeven[i]<<"  ";
}
```

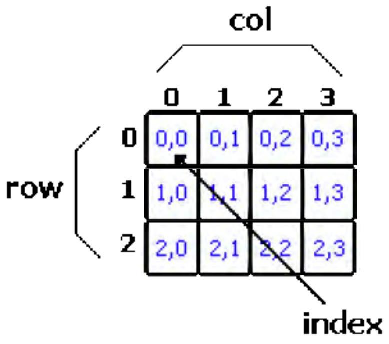
LECTURE 14

1. Array of Two Dimension:

Arrays can have higher dimension. There can be arrays of two dimension which is array of arrays. It is accessed with two index. Also there can be arrays of dimension higher than two.

General Form of 2D-Array:
data-type Array-name [*Row-size*] [*Col-size*];

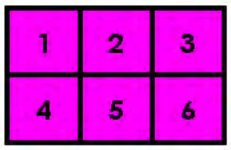
Examples:
```
int     a [10] [10];
int     num [3] [4];
```



2. Initializing 2D-Array Elements:

- The first element of array age:

```
a [2] [3] = { {1, 2, 3} , {4, 5, 6} };
```



1	2	3
4	5	6

3. Read / Write / Process Array Elements

Example 1

Write C++ program, to read 15 numbers, 5 numbers per row, the print them:

```
#include<iostream.h>
void main ( )
{
        int a [ 3 ] [ 5 ];
        int i , j;
        for ( i = 0 ; i < 3; i++ )
           for ( j = 0 ; j < 5; j++ )
              cin >> a [ i ] [ j ];

        for ( i = 0 ; i < 3; i++ )
           {
              for ( j = 0 ; j < 5; j++ )
                 cout << a [ i ] [ j ];
              cout << endl;
           }
}
```

Example 2

Write C++ program, to read 4*4 2D-array, then find the summation of the array 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;
           }
}
```

Example 3

Write C++ program, to read 3*4 2D-array, then find the summation of each row:

```
#include<iostream.h>
void main ( )
{
        int a [ 3 ] [ 4 ];
        int i , j, sum = 0;
        for ( i = 0 ; i < 3; i++ )
           for ( j = 0 ; j < 4; j++ )
              cin >> a [ i ] [ j ];

        for ( i = 0 ; i < 3; i++ )
         {
              sum = 0;
              for ( j = 0 ; j < 4; j++ )
                 sum += a [ i ] [ j ];
              cout << "summation of row " << i  << " is: " << sum << endl;
         }
}
```

Example 4

Write C++ program, to read 3*4 2D-array, then replace each value equal 5 with 0:

```
#include<iostream.h>
void main ( )
{
        int a [ 3 ] [ 4 ];
        int i , j;
        for ( i = 0 ; i < 3; i++ )
           for ( j = 0 ; j < 4; j++ )
              cin >> a [ i ] [ j ];

        for ( i = 0 ; i < 3; i++ )
           for ( j = 0 ; j < 4; j++ )
                 if  ( a [ i ] [ j ] == 5 )   a [ i ] [ j ] = 0;

        for ( i = 0 ; i < 3; i++ )
           {
              for ( j = 0 ; j < 4; j++ )
                 cout << a [ i ] [ j ];
              cout << endl;
           }
}
```

Example 5

Write C++ program, to addition two 3*4 arrays:

```
#include<iostream.h>

void main ( )
{
        int a [ 3 ] [ 4 ], b [ 3 ] [ 4 ], c [ 3 ] [ 4 ];
        int i , j;
        cout << "enter element of array A: \n";
        for ( i = 0 ; i < 3; i++ )
            for ( j = 0 ; j < 4; j++ )
                cin >> a [ i ] [ j ];
        cout << "enter element of array B: \n";
        for ( i = 0 ; i < 3; i++ )
            for ( j = 0 ; j < 4; j++ )
                cin >> b [ i ] [ j ];
        for ( i = 0 ; i < 3; i++ )
            for ( j = 0 ; j < 4; j++ )
                    c [ i ] [ j ] = a [ i ] [ j ] + b [ i ] [ j ];
        for ( i = 0 ; i < 3; i++ )
            {
                for ( j = 0 ; j < 4; j++ )
                    cout << c [ i ] [ j ];
                cout << endl;
            }
}
```

Example 6

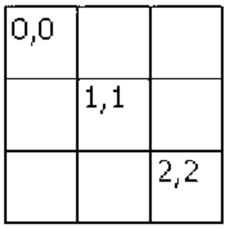
Write C++ program, to convert 2D-array into 1D-array:

```
#include<iostream.h>

void main ( )
{
        int a [ 3 ] [ 4 ];
        int b [ 12 ];
        int i , j, k = 0;

        for ( i = 0 ; i < 3; i++ )
            for ( j = 0 ; j < 4; j++ )
                cin >> a [ i ] [ j ];

        for ( i = 0 ; i < 3; i++ )
            for ( j = 0 ; j < 4; j++ )
                {
                    b [ k ] =  a [ i ] [ j ];
                k++;
            }
        for ( i = 0 ; i < k; i++ )
            cout << b [ i ];
}
```

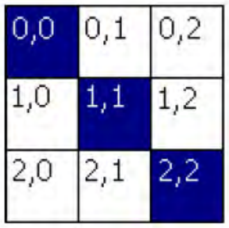
Example 7

Write C++ program, to replace each element in the main diameter (diagonal) with zero:

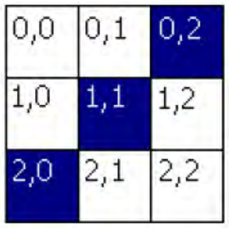
```
#include<iostream.h>
void main (  )
{
        int a [ 3 ] [ 3 ];
        int i , j;

        for ( i = 0 ; i < 3; i++ )
           for ( j = 0 ; j < 3; j++ )
              cin >> a [ i ] [ j ];

        for ( i = 0 ; i < 3; i++ )
           for ( j = 0 ; j < 3; j++ )
               if ( i == j )    a [ i ] [ j ] = 0;

        for ( i = 0 ; i < 3; i++ )
           {
              for ( j = 0 ; j < 3; j++ )
                 cout << a [ i ] [ j ];
              cout << endl;
           }
}
```



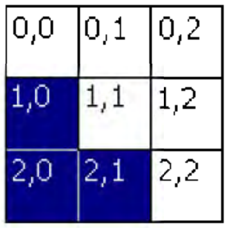
i = j



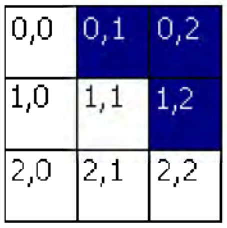
i = j



i + j = n-1



i > j



i < j

Example 8

Write C++ program, print the square root of an array:

```
#include<iostream.h>
void main ( )
{
        int a [ 3 ] [ 3 ] , b [ 3 ] [ 3 ];
        int i , j;
        for ( i = 0 ; i < 3; i++ )  {
           for ( j = 0 ; j < 3; j++ )    {
              b[ i ][ j ]= sqrt(a[ i ][ j ]);
              cout << b [ i ] [ j ];

}  }   }
```

Example 9

Write C++ program, to read 3*3 2D-array, then find the summation of the main diagonal and its secondary diagonal of the array elements, finally print these elements:

```
#include<iostream.h>
void main ( )
{
        int a [ 3 ] [ 3 ];
        int i , j, x , y;
        for ( i = 0 ; i < 3; i++ )  {
           for ( j = 0 ; j < 3; j++ )   {
              cin >> a [ i ] [ j ];

              if ( i == j )
              x=x+a[ i ][ j ];
              if ( i + j =4)
              y=y+a[ i ][ j ];
                                       }    }
        cout << "summation of diagonal is: " << x << endl;
        cout << "summation of inverse diagonal is: " << y << endl;
}
```

WORK SHEET (6)

Arrays

Q1: Write a C++ program, using function, to find if the array's elements are in order or not.

Q2: Write a C++ program, using function, to compute the number of zeros in the array.

Q3: Write a C++ program, using function, to find the value of array C from add array A and array B. `C[ i ] = A [ i ] + B [ i ];`

Q4: Write a C++ program, using function, to multiply the array elements by 2. `A[ i ] = A [ i ] * 2;`

Q5: Write a C++ program, using function, to reads temperatures over the 30 days and calculate the average of them.

Q6: Write a C++ program, using function, to merge two arrays in one array.

Q7: Write C++ program, to read 3*4 2D-array, then find the summation of each col.

Q8: Write C++ program, to replace each element in the second diameter (diagonal) with zero.

Q9: Write C++ program, to replace the elements of the main diameter with the elements of the second diameter.

Q10: Write C++ program, to find the summation of odd numbers in 2D-array.

Q11: Write C++ program, to find (search) X value in 2D-array, and return the index of it's location.

Q12: Write C++ program, to convert 1D-array that size [16] to 2D-array that size of [4] [4].

Q13: Write C++ program, to read A[n, n] of character, then find array B and array C, such that B contain only capital letters and C contain only small letters.

Q14: Write C++ program, to read A[n, n] of numbers, then put 10 instead each even positive number.

Q15: Write C++ program, to read A[n, n] of numbers, then put 10 instead each even positive number in the first diagonal.

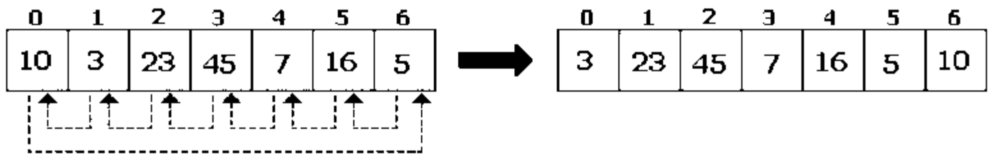
Q16: Write C++ program, to read A[n, n] of numbers, then find the minimum number in array.

Q17: Write C++ program, to exchange row1 and row3 in 4*3 array.

Q18: Write C++ program, to exchange row0 with col3 in 4*4 array.

Q19: Write C++ program, to find the greatest number in the second diagonal, in 3*3 array.

Q20: Write C++ program, to read X[n], and rotate the elements to the left by one position.
<u>i.e:</u>



Q21: Write C++ program, to read A[n] and a location Z then delete the number at location Z from the array, and print the new array after deletion.

Q22: Write C++ program to order the array in ascending and descending order.

Q23: Write C++ program to read (n) no.s and find the average of the even no. on it.

Q24: Create the array (b) from (a).

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

Q25: Create the arrays bellow.

1 1 1 1	2 1 1 1
2 2 2 2	1 2 1 1
3 3 3 3	1 1 2 1
4 4 4 4	1 1 1 2

LECTURE 15

1. String:

In C++ strings of characters are implemented as an array of characters. In addition a special null character, represented by \0, is appended to the end of string to indicate the end of the string.

General Form of String:

char String-name [*size*];

Examples: char name [10] = "Mazin Alaa";

$\hookrightarrow$ 'M' , 'a' , 'z' , 'i' , 'n' , ' ' , 'A' , 'l' , 'a' , 'a' , '\0'

char str [ ] = "ABCD";

$\hookrightarrow$ 'A' , 'B' , 'C' , 'D' , '\0'

```
str [0] : 'A'
str [1] : 'B'
str [2] : 'C'
str [3] : 'D'
str [4] : '\0'     <-> null
```

2. Read / Write / Process Array Elements:

Example 1

Write C++ program to print string, then print it character by character:

```
#include<iostream.h>
void main( )
{
        char  s [ ] = "ABCD";
        cout << "Your String is: " << s << endl;
        for ( int i =0; i < 5; i++ )
              cout << "S[" << i << "] is: " << s [ i ] << endl;
```

Output is:

```
Your String is:  ABCD
S[0] is: A
S[1] is: B
S[2] is: C
S[3] is: D
S[4] is:
```

Example 2

Write C++ program to convert each lower case letter to upper case letter:

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

void main( )
{
          char  s [  ] = "abcd";
          cout << s << endl;

          for ( int i =0; i < 4; i++ )
                s [i] = char(toupper (s[i] ));

          cout << s;
}
```

Note:

There are several ways to read and write (there are several input/output function) like:

```
cin.getline ( str, 10 );
cin.get ( ch );
cin.ignor ( 80, '\n' );
cin.putback ( ch );
cout.put ( ch );
```

Apply it ...

3. Member Function of String:

The string library has many member functions of string like:

Member Function	Functionality	Example

strlen (string)	Return the length of the string	a [ ] = "abcd";
		cout << strlen (a);

strcpy (string2, string1)	Copy the content of the 1nd string into the 2st string	char a[ ]= "abcd" , b[ ]=" ";
		strcpy (b , a);
		cout << a << b;

strcat (string1, string2)	Append the content of the 2nd string into the end of the 1st string	char a[ ]= "abcd" , b[ ]="1234";
		strcat (a , b);
		cout << a << b;
		abcd1234 1234

strcmp (string1, string2)	Return 0 if the 1st string is equal to the 2nd string. Return a Positive number if the 1st string is greater than the 2nd string. Return a Negative number if the 1st string is smaller than the 2nd string.	char a[ ]= "abcd" , b[ ]="abcd"; cout << strcmp (a , b); **0** if a == b **+** if a > b **-** if a < b

4. stdlib Library:

The stdlib library has many member functions of string like:

Member Function	Functionality	Example

A atoi (*a*)	Converts string to int type.	int i; char a [ ] = "1234";
		i = atoi (a);

A atof (*a*)	Converts string to float type.	float f; char a [ ] = "12.34";
		f = atof (a);

itoa (*i* , *a* , *10*);	Converts integer number to alphabet (char or string type).	int i = 1234; char a [ ] = "";
		cout << itoa (i , a , 10);

WORK SHEET (7)
String

Q1: Write C++ program to print a string, and then print it character by character *in reveres order.*

<u>i.e:</u>

```
abcd  → a
        b
        c
        d
```

Q2: Write C++ program to check each character in the string to convert it to lower case letter if it's an upper case letter and convert it to upper case letter if it's a lower once.

Q3: Write C++ program to read a sentence and print its words separately.

Q4: Write C++ program to apply the following instructions:

- cin.getline (str, 10);
- cin.get (ch);
- cin.ignor (80, '\n');
- cin.putback (ch);
- cout.put (ch);

Q5: Write C++ program to apply the following instructions:

- strlen (string)
- strcpy (string2, string1)
- strcat (string1, string2)
- strcmp (string1, string2)

Q5: Write C++ program to apply the following instructions:

- i atoi (a)
- f atof (a)
- itoa (i , a , 10);

LECTURE 16

1. Structures:

Structures are typically used to group several data items together to form a single entity. It is a collection of variables used to group variables into a single record. Thus a structure (the keyword **struct** is used in C++) is used. Keyword struct is a data-type, like the following C++ data-types (int, float, char, etc...). This is unlike the array, which all the variables must be the same type. The data items in a structure are called the members of the structure.

General Form of Structure:

```
struct struct-name
{
   variables ...
};
```

2. The Three Ways for Declare the Structure:

A.

```
#include <iostream.h>

struct data
{
   char *name;
   int age;
};

void main()
{
   struct data student;
```

B.

```
struct data
{
      char *name;
      int age;
   }  student;
```

C.

```
   typedef  struct
   {
      char *name;
      int age;
   }  student;
```

- The above three ways are called structure specifier (tells how the structure is organized) or it is called structure declaration.
- To access elements in a structure, use a record selector **(.)**.

```
      student . name="ahmed";
      student . age=20;
         :
   }
```

Note: we can assign more than one name as a structure-name, to the one structure. For example:

```
   typedef  struct
   {
      char *name;
      int age;
   }  student , lecturer;
```

Example 1:

This example uses parts inventory to demonstrate structures.

```
#include<iostream.h>

Struct part    //   specify a structure
{
Int   model_no;
Int   part_no;
Float  cost;
}

Void main()
{
Part p1;     //  define a structure variable.
P1.model_no=6244;
P1.part_no=373;
P1.cost=217.55;
Cout<<"/n model"<<p1.model_no;
Cout<<", part"<<p1.part_no;
Cout<<", cost"<<p1.cost;
}
```

The above program has three main aspects: specifying the structure, defining a structure variable, and accessing the members of the structure.

3. A measurement example:

Let's see how a structure can be used to group a different kind of information. If you have ever looked at an architectural drawing, you know that distances are measured in feet and inches. The length of a living room, for example, might be given as 12'-8'', meaning 12 feet 8 inches. The hyphen is not a negative sign; it merely separates the feet from the inches. This is part of the English of measurement. The following program will show how two measurements of type distance can be added together.

Example 2:

Write C++ program to find the distance in English system.

```
#include<iostream.h>

struct distance
{
int feet;
float inches;
}
Void main ()
{
distance d1,d3;
distance d2={11,6.25};
cout<<"\n Enter feet:";
cin>>d1.feet;
cout<<"\n Enter inches:";
cin>>d1.inches;
d3.inches=d1.inches+d2.inches;
d3.feet=0;
If (d3.inches >=12.0)
{
```

```
d3.inches -=12.0;
d3.feet ++;
}
d3.feet +=d1.feet + d2.feet;
cout<<d1.feet<<"\'-"<<d1.inches<<"\"+";
cout<<d2.feet<<"\'-"<<d2.inches<<"\"=";
cout<<d3.feet<<"\'-"<<d1.inches<<"\"\n";
}
```

4. Structures within Structures:

You can nest structures within other structures. Here's a variation on the English system program that shows how this looks. In the bellow program we want to create a data structure that stores the dimensions of a typical room: its length and width. Since we're working with English distances, we'll use two variables of type distance as the length and width variables.

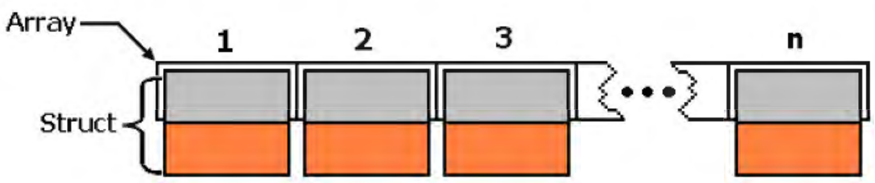
Example 3:

Write C++ program to find the area of the room in English system.

```
#include<iostream.h>

struct distance
{
int feet;
float inches;
}

struct room
{
distance length;
distance width;
};

Void main ()
{
room dining;
dining.length.feet=13;
dining.length.inches=6.5;
dining.width.feet=10;
dining.width.inches=0.0;
float L=dining.length.feet+dining.length.inches/12;
float W=dining.width.feet+dining.width.inches/12;
cout<<"\n Dining room area is"<<L*W<<"Square feet";
}
```

5. Array of Structures:

The **struct** is a data-type. So we can define an array as an array of struct, like define an array as an array of int, or of any other C++ data-types.



However, the following simple example shown how can create and use an ***array of struct***.

Example 4:

This simple example to show how can create and use an array of structure.

```
#include<iostream.h>

typedef struct
{
          char *name;
          int age;
} student;

void main ( )
{
         student   array [10];
         array [1] . name = "ahmed";
         array [1] . age = 20;
         cout  <<  array[1] . name  <<  endl;
         cout  <<  array[1] . age;
}
```

```
cin >> array [1] . name ;
cin >> array [1] . age ;
```

Example 5:

Write a C++ Program, using structure type, to read name and age for ten students.

```
#include<iostream.h>

typedef struct
{
          char *name;    //Or name[10]
          int age;
} student;

void main ( )
{
          student   array [10];

          for ( i = 0 ; i < 10 ; i++ )
          {
              cin >> array [i] . name;
              cin >> array [i] . age;
          }

          for ( i = 0 ; i < 10 ; i++ )
          {
          cout  <<  array[i] . name  <<  endl;
          cout  <<  array[i] . age;
          }

}
```

6. Functions and Structures:

A structure can be passed to a function as a single variable. The scope of a structure declaration should be an external storage class whenever a function in the main program is using a structure data types. The field or member data should be same throughout the program either in the main or in a function.

```
# include <iostream.h>
Struct sample  {
Int x;
Float y;
};
Sample first ;
Void main (void)
{
Void display (struct  sample  one);   //  function declaration
-  - -
- - -
Display (one);     // function calling
     - - -
     - - -
     }
Void display (struct  sample  out)   // function definition
{
     - - -
     - - -
Out.x=10;
Out.y=-20.20;
     - - -
     - - -
}
```

Example 6:

Write C++ program to display the contents of a structure using function definition.

```
#include<iostream.h>

struct date  {
int day;
int month;
int year;
} ;
Void main(void)
{
date today;
void display (struct date one);   // function declaration
today.day=10;
today.month=3;
today.year=2011;
display (today);
}
Void display (struct  date one)
{
cout<<"Today's date is =" << one.day << "/";
cout<< one.month;
cout<<"/" << one.year << endl;
}
```

Output

```
Todat's date is = 10/3/2011
```

WORK SHEET (8)

Structures

Q1: Write a C++ program, that declares the structure called Employee_Info, which having the following members:

1- Employee name. (must be less than 25 characters)
2- Employee age. (must be 2 digits)
3- Employee address. (must be less than 20 characters)
4- Phone number. (must be 8 or 11 digits)
5- Country name. (must be less than 29 characters)

Then read and print this information for the 100 Employees.

Q2: Show the declaration of the following:

Employees:
- **Ind- Employees:**
  - **ID.**
  - **Name.**
  - **Sex.**
  - **Rate.**
- **Home:**
  - **Street.**
  - **City.**
  - **State.**
- **BirthDate:**
  - **Month.**
  - **Day.**
  - **Year.**
- **StrartDate:**
  - **Month.**
  - **Day.**
  - **Year.**

By using your declaration, write a C++ program that reads and stores data, then print only employees whose ID number less than 100.

LECTURE 17

1. Enumerated Data Types:

The enumerated data type is a programmer-defined type that is limited to a fixed list of values. A specifier gives the type a name and specifies the permissible values definitions then create variables of this type. Internally, the compiler treats enumerated variables as integers.

Structures should not be confused with enumerated data type. Structures are a powerful and flexible way of grouping a diverse collection of data into a single entity.

General Form of EDT:

enum user_defined_name {
member_1;
member_2;
…
…
member_n;
};

Where **enum** is a keyword for defining the enumeration data type and the braces are essential. The members of the enumeration data type are the indivisual identifiers. Once the enumeration data type is defined, it can be declared in the following ways:

Storage_class enum user_defined_name var.1, var.2, …var.n

where the storage class is optional. For example:

```
1) Enum sample {
        Mon, tue, wed, thu, fri, sat, sun };
        Enum sample day1, day2, day3;
2) Enum drinks {
        Cola, tea, koffi,rani }
        Enum drinks d1, d2, d3;
3) Enum games {
        Tennis, chess, football, swimming, walking };
    Enum games  student, staff;
```

The EDT declaration can be written in a single declaration as:

```
    Enum  sample  { Mon, tue, wed, thu, fri, sat, sun  }
    Day1, day2, day3;
```

Which is exactly equivalent to:

```
    1)  Enum  sample  { Mon, tue, wed, thu, fri, sat, sun  }  day1;
        Enum  sample  Day2, day3;

    2)  Enum  sample  { Mon, tue, wed, thu, fri, sat, sun  };
        Enum  sample  Day1;
        Enum  sample  Day2;
        Enum  sample  Day3;
```

The enumeration constants can be assigned to the variable like day1=mon;. The enumeration constants are automatically assigned to integers starting from 0, 1, 2 etc. up to the last number in the enumeration.

Example 1:

Write C++ program to declare the EDT and to display the integer values on the screen.

```
#include <iostream.h>
 Void main(void)
{
        enum sample { Mon, tue, wed, thu, fri, sat, sun  }
        day1, day2, day3, day4, day5, day6, day7;
        day1=mon;
         day2=tue;
        day3=wed;
        day4=thu;
        day5=fri;
        day6=sat;
        day7=sun;
        cout<<"Monday      = "<<day1 <<endl;
        cout<<"Tuesday     = "<<day2 <<endl;
        cout<<"Wednesday = "<<day3 <<endl;
        cout<<"Thursday    = "<<day4 <<endl;
        cout<<"Friday        = "<<day5 <<endl;
```

Output:

```
Monday      = 0
Tuesday     = 1
Wednesday = 2
Thursday    = 3
Friday        = 4
Saturday    = 5
Sunday      =6
```

```
        cout<<"Saturday     = "<<day6 <<endl;
        cout<<"Sunday       = "<<day7 <<endl;
}
```

These integers are normally chosen automatically but they can also be specified by the programmer with negative or positive numbers, for example

Enum sample {mon, tue, wed=10, thu, fri, sat=-5, sun}
day1, day2, day3, day4, day5, day6, day7;

The C++ compiler assigns the enumeration constants as

Monday	**= 0**
Tuesday	**= 1**
Wednesday	**= 10**
Thursday	**= 11**
Friday	**= 12**
Saturday	**= -5**
Sunday	**=-4**

Example 2:

Write C++ program to declare the EDT and to display the difference between days.

```
#include <iostream.h>
        Enum days_of_week { Mon, tue, wed, thu, fri, sat, sun }
 Void main(void)
{
        days_of_week  day1, day2;
        day1=mon;
        day2=thu;
        int diff=day2-day1;             // can do integer arithmatic
        cout<<"Days between="<<diff<<endl;
        if (day1 < day2)                // can do comparision
        cout<<"dau1 comes before day2 \n";
}
```

Example 3:

Write C++ program to count the places in a sentence where a string of nonspaces character changes to spaces.

```
#include <iostream.h>
#include <conio.h>
          enum boolean {false,true};      // false=0    , true=1
 Void main()
{
boolean isword=false;       //  true when word, false when whitespace
char ch='a';                // character read from keyboard
int wordcount=0;            // numbers of words reads
do
{
ch=getche();                // get character
if (ch==' ' || ch=='\r')    // if white space or enter key
{
if (isword)
{
wordcount++;          // count the word
isword=false;         // reset flag
}
}
else                  //  otherwise its normal char.
if (! isword)         //   if start of word
  isword=true;        //   set flag
}
while (ch!='\r');   // quit on enter key
cout <<"\n  --- word count is"<<wordcount<<"---\n";
}
```

WORK SHEET (9)

EDT

Q1: Write C++ program to declare the EDT and to display the difference between months.

Q2: Write C++ program to declare the EDT and to display the number of each season.

Q3: Write C++ program to read 10 employee and display the information of each one when enter its number.

Employee code
Employee name
Employee sex (EDT)
Employee craduate (B.Sc. , M.Sc., Ph. D.) (EDT)

LECTURE 18

1. Pointers:

The pointer is a powerful technique to access the data by indirect reference as it holds the address of that variable where it has been stored in the memory.

2. Pointer Declaration:

A pointer is a variable which holds the memory address of another variable. The pointer has the following advantages:

1. It allows passing variables, arrays, functions, strings and structures as function arguments.
2. A pointer allows returning structured variables from functions.
3. It provides functions which can modify their calling arguments.
4. It supports dynamic allocation and deallocation of memory segments.
5. With the help of a pointer, variables can be swapped without physically moving them.
6. It allow to establish links between data elements or objects for some complex data structures such as linked lists, stacks, queues, binary trees, tries and graphs.
7. A pointer improves the efficiency of certain routines.

In C++ pointers are distinct such as integer, float, character, etc. A pointer variable consists of two parts, namely, (i) the pointer operator and (ii) the address operator.

Pointer Operator:

A pointer operator can be represented by a combination of (*) with a variable, for example **int *ptr**; where ptr is a pointer variable which holds the address of an integer data type.

General Form of Pointer:
Data_type   *pointer_variable;

Example: int x, y; int *ptr1, *ptr2;

Address Operator:

An address operator can be represented by a combination of & with a pointer variable. For example, if a pointer variable is an integer type and also declared (&) with the pointer variable, then it means that the variable is of type "**address of**". For example **m=&ptr;**. Note that the pointer operator & is an operator that returns the address of the variable following it.

Note:

Notice the difference between the reference and dereference operators:

- & is the reference operator and can be read as "**address of**"
- * is the dereference operator and can be read as "**value pointed by**"

Thus, they have complementary (or opposite) meanings. A variable referenced with & can be dereferences with *.

Examples:

(1) **Ptr1 = &x;** The memory address of variable x is assigned to the pointer variable ptr1.
(2) **Y=*prt1;** The contents of the pointer variable ptr1 is assigned to the variable y, not
the memory address.
(3) **Ptr1=&x;** The address of the ptr1 is assigned to the pointer variable ptr2. The contents
Ptr2=ptr1; of both ptr1 and ptr2 will be the same as these two pointer variables hold
the same address.

Example:

```
andy = 25;
ted = &andy;
```

Right after these two statements, all of the following expressions would give true as result:

```
andy == 25
&andy== 1776
ted == 1776
*ted == 25
```

Examples of invalid pointer declaration:

(1) int x;
int x_pointer;
x_pointer=&x;
Error: pointer declaration must have the prefix of *.

(2) float y;
float *y_pointer;
y_pointer=y;
Error: While assigning variable to the pointer variable the address operator (&) must used along with the variable y.

(3) int x;
char *c_pointer;
c_pointer = &x;
Error: Mixed data type is not permitted.

Example 1:

This simple example to show how can create and use pointer of char.

```
#include <iostream.h>
 int main()
{
         char c='a';
          char *p_c = &c;
          cout<< *p_c;
}
```

Example 2:

This simple example to show how can create and pointer of integer.

```
#include <iostream.h>
 main()
{
          int myval=10;
          int *p_myval;
          p_myval = &myval;
          cout<<*p_myval;
}
```

Example 3:

```
#include <iostream.h>
 main()
{
          int myval = 7;
          int *p_myval = &myval;
          *p_myval = 6;
          cout<<*p_myval<<"\n";
          cout<<myval;
}
```

Example 4:

```
#include <iostream.h>
  main()
{
          int myval=5;
          int myval2 = 7;
          int *p_primate;
          p_primate = &myval;
```

```
            *p_primate = 9;
            p_primate = &myval2;
            *p_primate = 10;
            cout<<myval<<" "<<myval2;
}
```

Example 5:

```
#include <iostream.h>
  Void main(void)
{
Float value;
Float *ptr;
Value = 120.00;
Ptr = &value;
Cout<< "Memory address ="<<ptr<<endl;
Ptr --;
Cout<<"Memory address after decrementer =";
Cout<<ptr<<endl;
}
```

3. Pointers and Functions:

Pointers are very much used in a function declaration. Sometimes only with a pointer a complex function can be easily represented and accessed. The use of the pointers in a function definition may be classified into two groups; they are call by value and call by reference.

Call by value:

Whenever a portion of the program invokes a function with formal arguments, control will be transferred from the main to the calling function and the value of the actual argument is copied to the function. Within the function, the actual value copied from the calling portion of the program may be altered or changed. When the control is transferred back from the function to the calling portion of the program, the altered values are not transferred back. This type of passing formal arguments to a function is technically known as call by value.

Example 6:

A program to exchange the contents of two variables using a call by value.

```
#include <iostream.h>
  Void main(void)
{
Int x,y;
void swap (int,int);
x=100;
y=20;
cout<<"values before swap"<<endl;
cout<<"x="<<x<<"and y="<<y<<endl;
swap(x,y);   //call by value
cout<<"values after swap"<<endl;
cout<<"x="<<x<<"and y="<<y<<endl;
}

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

O/P:

```
values before swap
x=100 and y=20
values after swap
x=100 and y=20
```

Call by reference:

When a function is called by a portion of a program, the address of the actual arguments is copied onto the formal arguments, though they may be referred by different variable names. The content of the variables that are altered within the function block are returned to the calling portion of a program in the altered form itself, as the formal and the actual arguments are referencing the same memory location or address. This is technically known as call by reference or call by address or call by location.

Example 7:

A program to exchange the contents of two variables using a call by reference

```
#include <iostream.h>
 Void main(void)
{
Int x,y;
void swap (int *x, int *y);
x=100;
y=20;
cout<<"values before swap"<<endl;
cout<<"x="<<x<<"and y="<<y<<endl;
swap(&x, &y);   //call by reference
cout<<"values after swap"<<endl;
cout<<"x="<<x<<"and y="<<y<<endl;
}

int func (int *x, int *y)
{
int temp;
temp=*x;
*x = *y;
*y=temp;
}
```

O/P:

```
values before swap
x=100 and y=20
values after swap
x=100 and y=100
```

LECTURE 19

1. Pointers and Arrays:

In C++, there is a close correspondence between array data type and pointers. An array name in C++ is very much like a pointer but there is a difference between them. The pointer is a variable that can appear on the left side of an assignment operator. The array name is a constant and cannot appear on the left side of an assignment operator. In all other respects, both the pointer and the array are the same.

Valid :

int value[20];

int *ptr;

ptr=&value[0]; //the address of the zeroth element is assigned to a pointer variable ptr.

ptr++ == value[1];

ptr+6 == &value[6];

*ptr == &value[0]

*ptr == &value[ ]

*(ptr+6) == value[6]

ptr ++ == &value[1]

Example:

int s[200];

int *sptr;

sptr=s; → sptr = &s[0];

a[i] → *((a) + (i)) → *(&(a)[0] + (i))

Example 1:

A program to display the content of an array using a pointer arithmetic

```
#include <iostream.h>
 void main(void)
{
          static int a[4]= {11, 12, 13, 14 };
          int i, n, temp;
          n=4;
          cout<<"Contents of the array" << endl;
          for (i=0; i<=n-1; ++i)   {
          temp = *((a) + (i)  );      //     or temp= *(&(a)[0] + (i));
          cout<<"value ="<<temp <<endl;
}
}
```

```
Contents of the array
Value =11
Value =12
Value =13
Value =14
```

2. Arrays of Pointers:

The pointers may be arrayed like any other data type. The declaration for an integer pointer array of size 10 is **int *ptr[10];** makes **ptr[0], ptr[1], ptr[2], … , ptr[10];**

Where ptr is an array of pointers that can be used to point the first elements of the arrays a, b, c. The following are valid assignment statements in C++:

Ptr[0] = &a[0];

Ptr[10] = &b[0];

Ptr[20] = &c[0];

Example 2:

A program to display the content of pointers using an array of pointers

```
#include <iostream.h>
 void main(void)
{
Char *ptr[3];
Ptr[0]="Ahmed";
Ptr[1]="Reem";
Ptr[2]="Ali";
Cout<<"contents of pointer 1 ="<<ptr[0]<<endl;
Cout<<"contents of pointer 2 ="<<ptr[1]<<endl;
Cout<<"contents of pointer 3 ="<<ptr[2]<<endl;  }
```

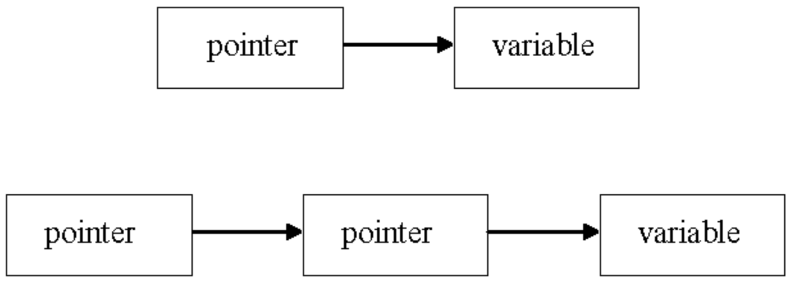
```
contents of pointer 1 = Ahmed
contents of pointer 2 = Reem
contents of pointer 3 = Ali
```

3. Pointers to Pointers:

An array of pointers is the same as pointers to pointers. As an array of pointers is easy to index because the indices themselves convey the meaning of a class of pointers. However, pointers to pointers can be confusing. The pointer to a pointer is a form of multiple of indirections or a class of pointers. In the case of a pointer to a pointer, the first pointer contains the address of the second pointer, which points to the variable that contain the values desired. Multiple indirections can be carried on to whatever extent desired, but there are a few cases where more pointer to a pointer is needed or written.

Int **ptr2;

Where ptr2 is a pointer which holds the address of the another pointer.



Example 3:

A program to declare the pointer to pointer variable and to display the contents of thesepointers

```
#include <iostream.h>
 void main(void)
{
int value;
int *ptr1;
int **ptr2;
value = 120;
cout<< “value =”<<value<<endl;
ptr1=&value;
ptr2=&ptr1;
cout<<”pointer 1=”<<*ptr1<<endl;
cout<<”pointer 2=”<<**ptr2<<endl;
}
```

```
Value = 120
Pointer 1 = 120
Pointer 2 =120
```

WORK SHEET (9)

Pointer

Q1: Write a C++ program, to applied the mathematic operation by pointer.

Q2: Find the output:

```
#include<iostream.h>

int main ()

{

        int *x;

         int *p,*q;

         int c=100,a;

         x=&c;

         p=x+2;

         q=x-2;

         a=p-q;

         cout << "The address of x : " << x << endl;

         cout << "The address of p after incrementing x by 2 : " << p << endl;

         cout << "The address of q after derementing  x by 2 : " << q << endl;

         cout << " The no of elements between p and q :" << a << endl;

         return(0);

}
```