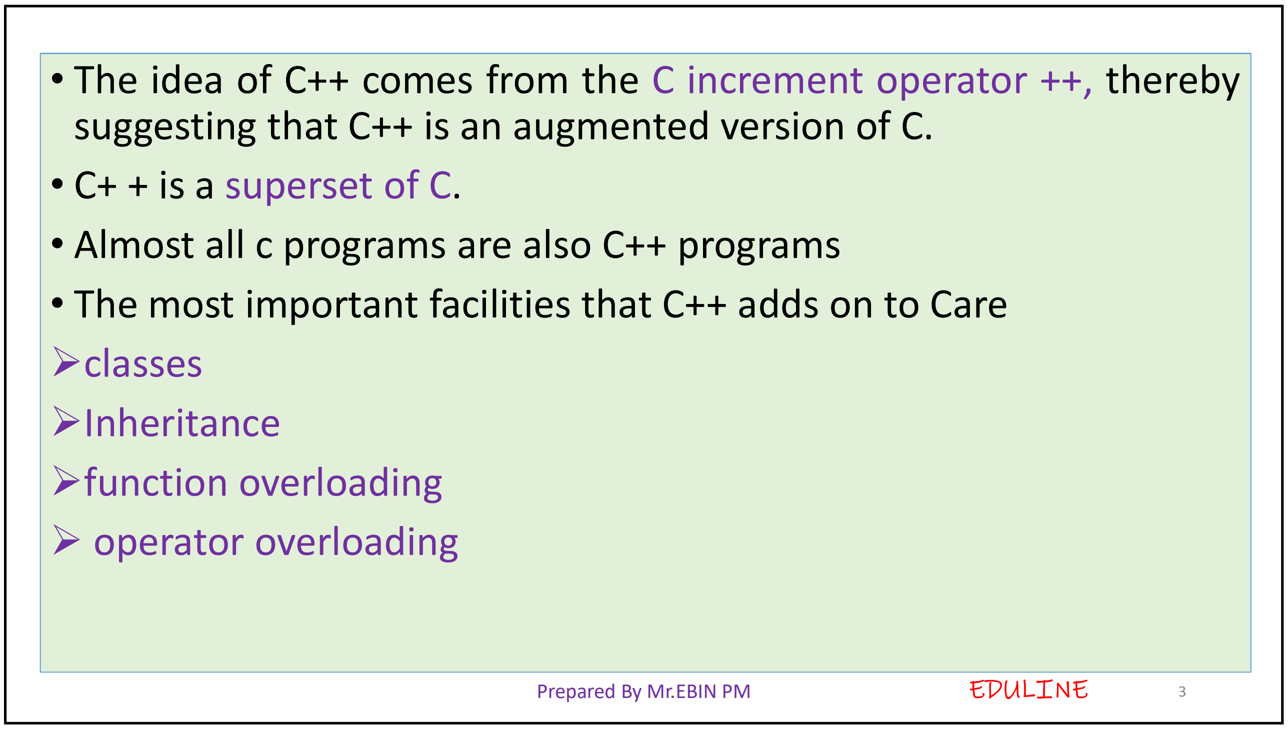
Object oriented programming with c++

EDULINE FOR CSE STUDENTS

C++

- C++ is an object-oriented programming language.
- It was developed by **Bjarne Stroustrup** at AT&T Bell Laboratories in Murray Hill, New Jersey, USA, in the early 1980's.
- C++ is an extension of C with a major addition of the class construct feature.
- Since the class was a major addition to the original C language, Stroustrup initially called the new language "**C with classes**".
- However, later in 1983, the name was changed to C++.



- The idea of C++ comes from the C increment operator ++, thereby suggesting that C++ is an augmented version of C.
- C+ + is a superset of C.
- Almost all c programs are also C++ programs
- The most important facilities that C++ adds on to Care

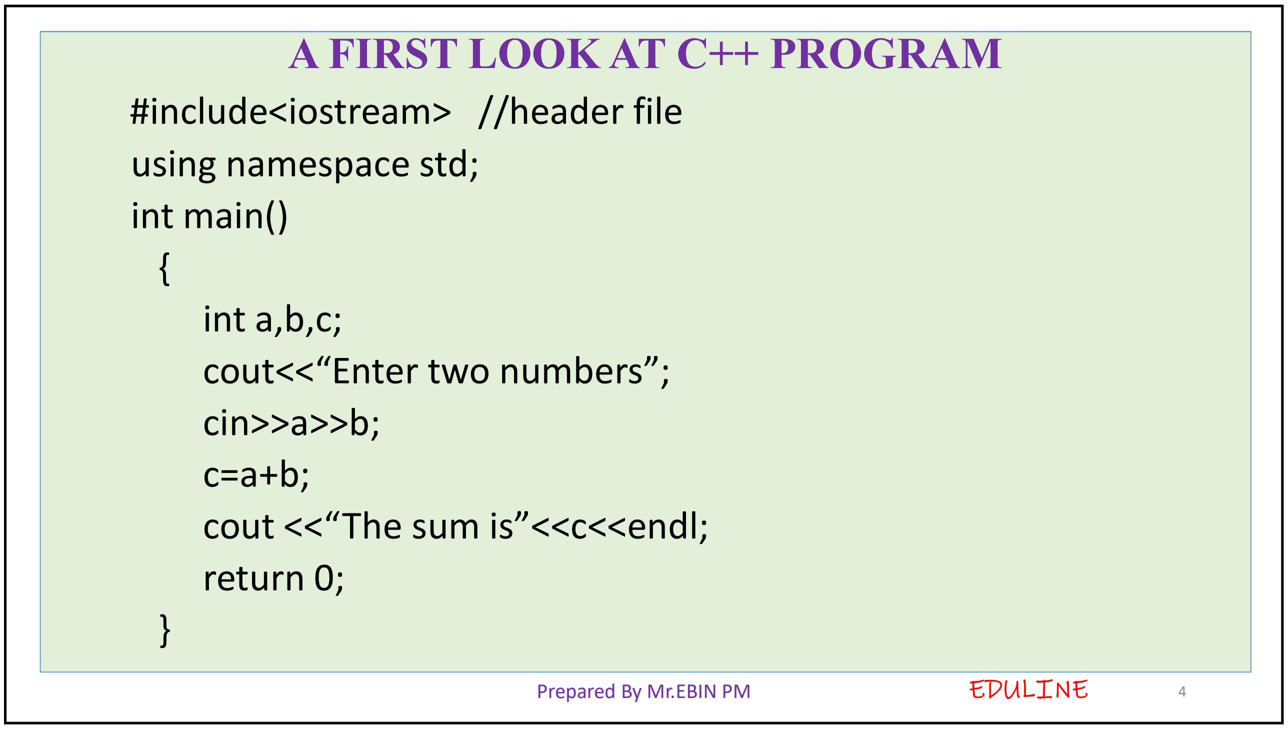
➢classes

➢Inheritance

➢function overloading

➢ operator overloading

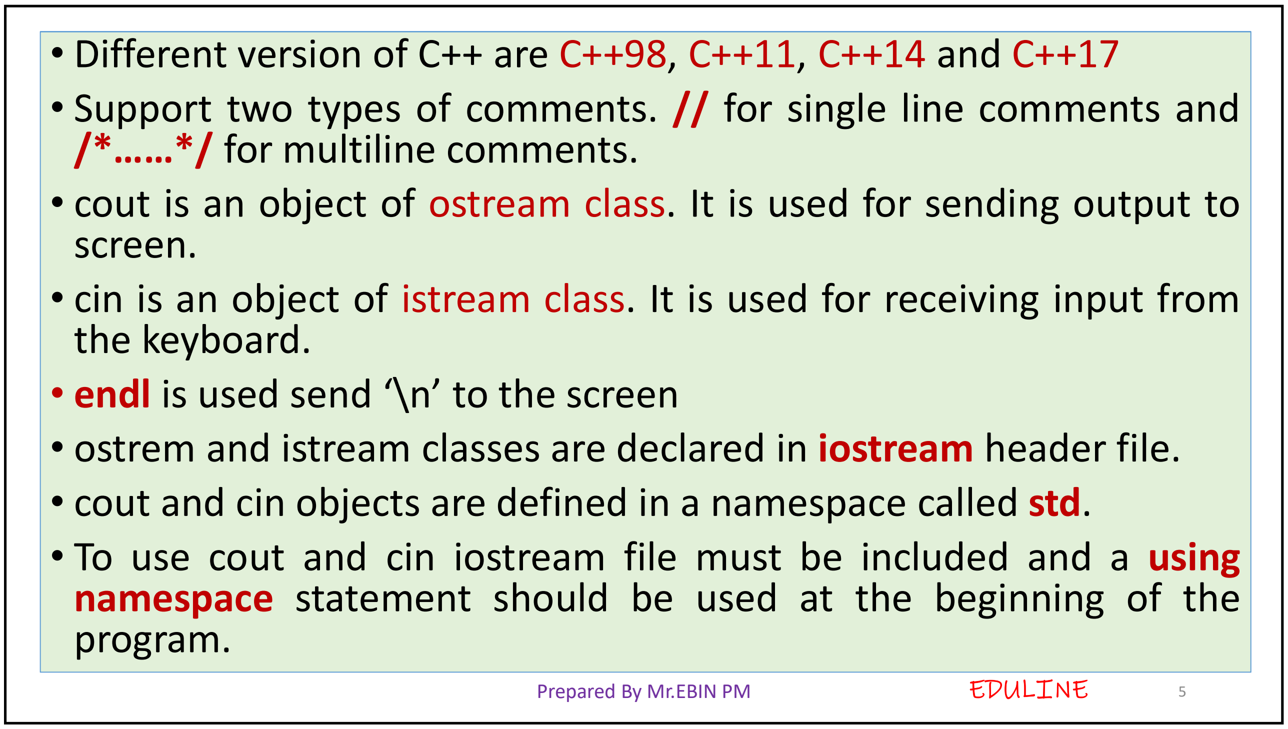
Prepared By Mr.EBIN PM EDULINE 3



A FIRST LOOK AT C++ PROGRAM

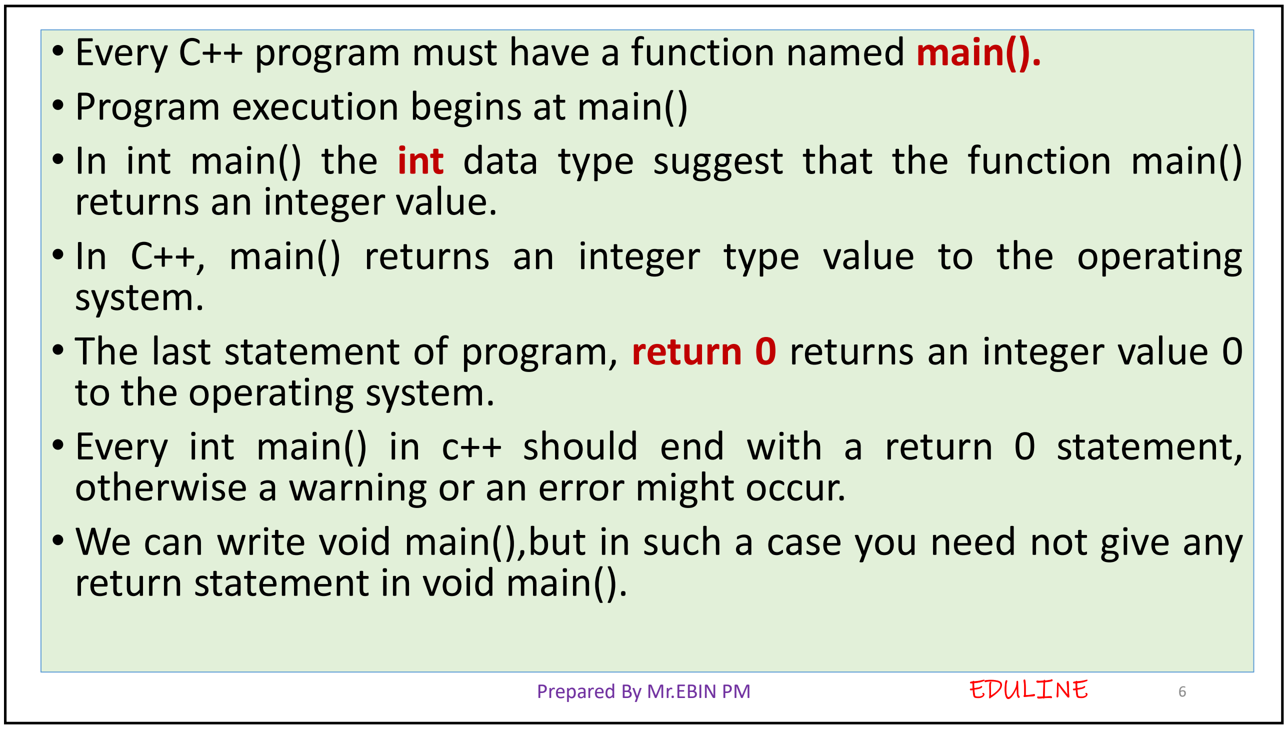
```
#include<iostream>   //header file
using namespace std;
int main()
  {
      int a,b,c;
      cout<<“Enter two numbers”;
      cin>>a>>b;
      c=a+b;
      cout <<“The sum is”<<c<<endl;
      return 0;
  }
```

Prepared By Mr.EBIN PM EDULINE 4



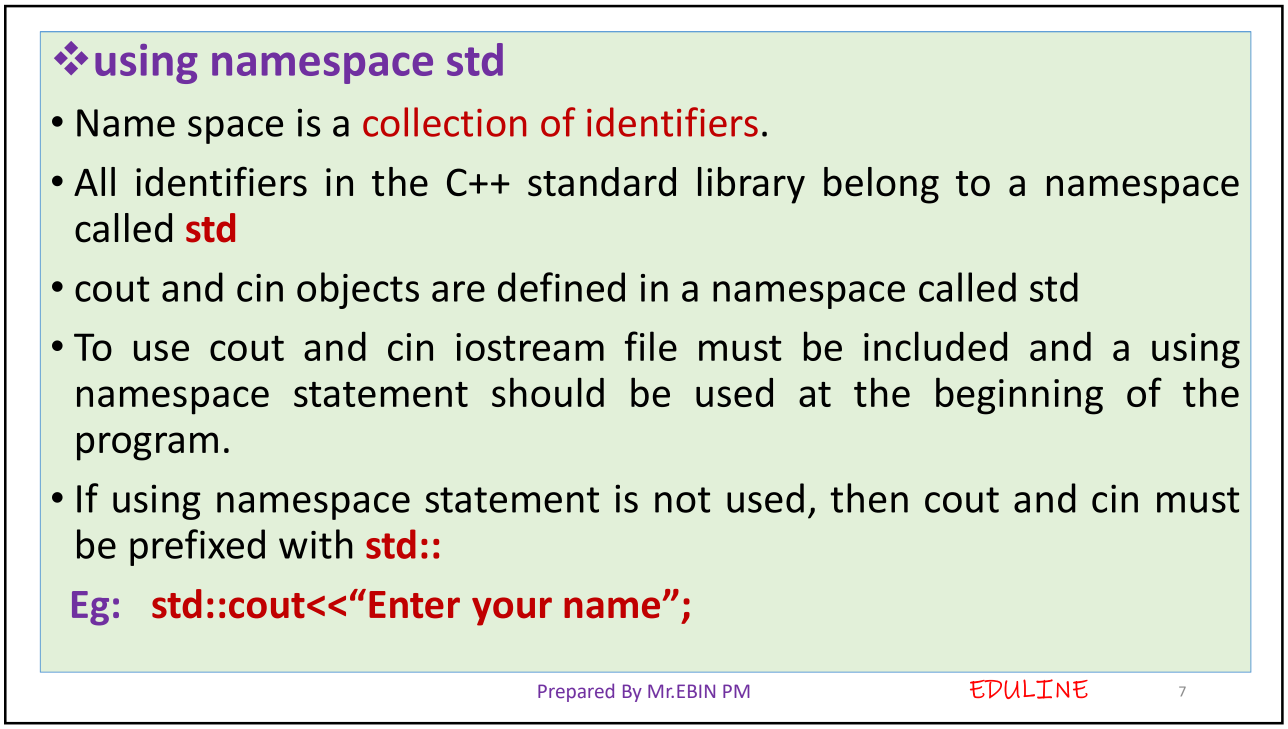
- Different version of C++ are C++98, C++11, C++14 and C++17
- Support two types of comments. **//** for single line comments and **/*……*/** for multiline comments.
- cout is an object of ostream class. It is used for sending output to screen.
- cin is an object of istream class. It is used for receiving input from the keyboard.
- **endl** is used send '\n' to the screen
- ostrem and istream classes are declared in **iostream** header file.
- cout and cin objects are defined in a namespace called **std**.
- To use cout and cin iostream file must be included and a **using namespace** statement should be used at the beginning of the program.

Prepared By Mr.EBIN PM EDULINE 5



- Every C++ program must have a function named **main().**
- Program execution begins at main()
- In int main() the **int** data type suggest that the function main() returns an integer value.
- In C++, main() returns an integer type value to the operating system.
- The last statement of program, **return 0** returns an integer value 0 to the operating system.
- Every int main() in c++ should end with a return 0 statement, otherwise a warning or an error might occur.
- We can write void main(),but in such a case you need not give any return statement in void main().

Prepared By Mr.EBIN PM EDULINE 6

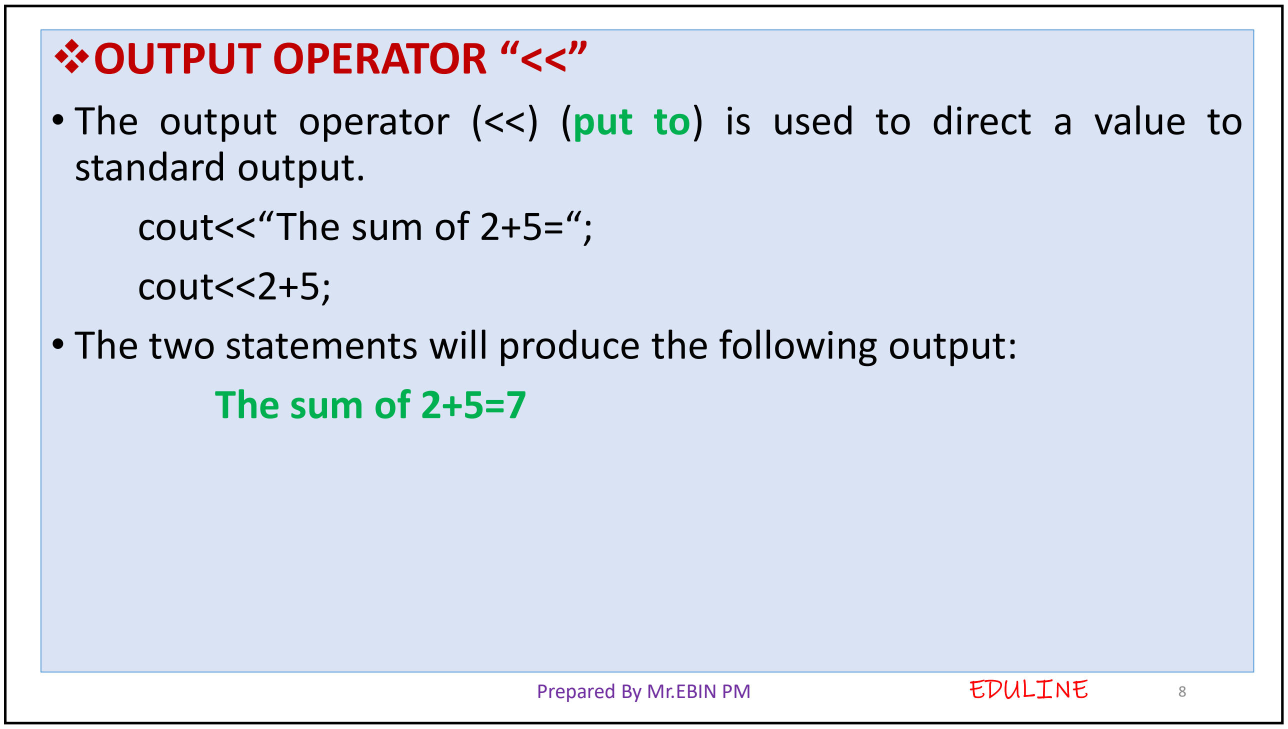


❖using namespace std

- Name space is a collection of identifiers.
- All identifiers in the C++ standard library belong to a namespace called **std**
- cout and cin objects are defined in a namespace called std
- To use cout and cin iostream file must be included and a using namespace statement should be used at the beginning of the program.
- If using namespace statement is not used, then cout and cin must be prefixed with **std::**

Eg: **std::cout<<"Enter your name";**

Prepared By Mr.EBIN PM EDULINE 7



❖OUTPUT OPERATOR "<<"

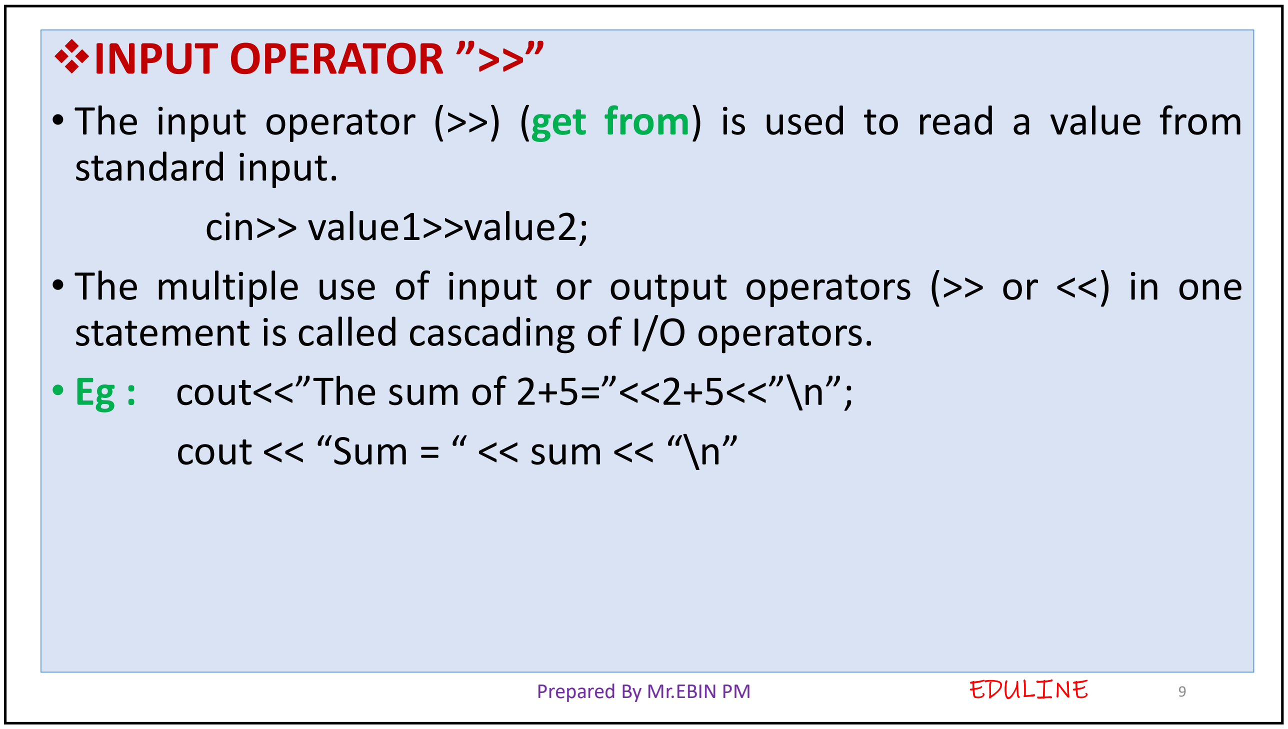
- The output operator (<<) (**put to**) is used to direct a value to standard output.

```
cout<<"The sum of 2+5=";
cout<<2+5;
```

- The two statements will produce the following output:

```
The sum of 2+5=7
```

Prepared By Mr.EBIN PM EDULINE 8



❖INPUT OPERATOR ”>>”

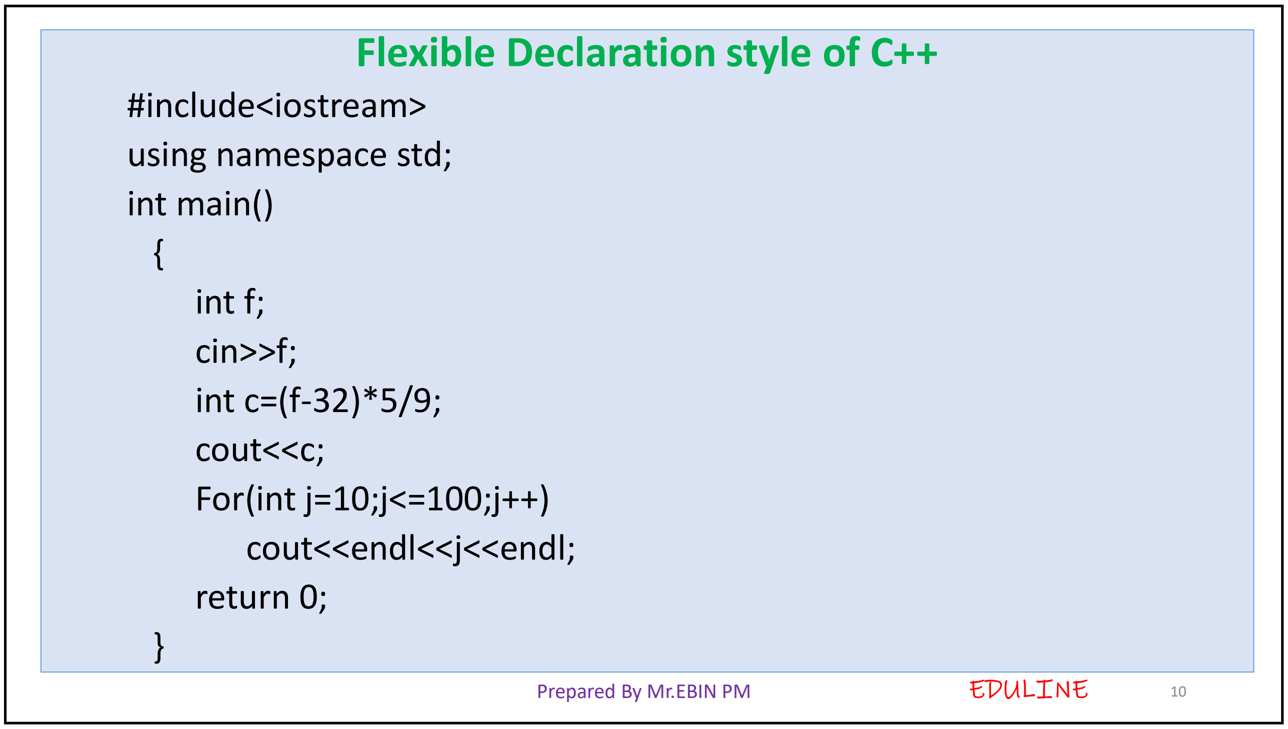
- The input operator (>>) (**get from**) is used to read a value from standard input.

  cin>> value1>>value2;

- The multiple use of input or output operators (>> or <<) in one statement is called cascading of I/O operators.
- **Eg :** cout<<”The sum of 2+5=”<<2+5<<”\n”;

  cout << “Sum = “ << sum << “\n”

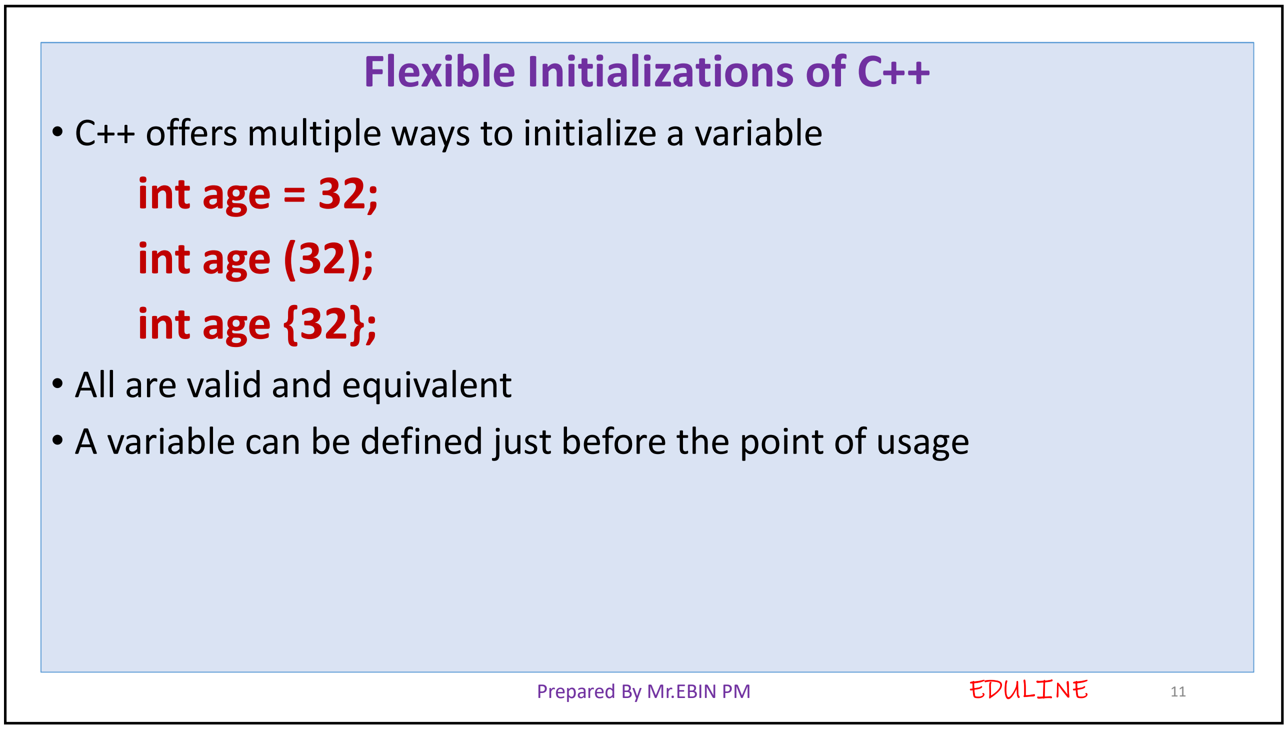
Prepared By Mr.EBIN PM EDULINE 9



Flexible Declaration style of C++

```
#include<iostream>
using namespace std;
int main()
  {
      int f;
      cin>>f;
      int c=(f-32)*5/9;
      cout<<c;
      For(int j=10;j<=100;j++)
          cout<<endl<<j<<endl;
      return 0;
  }
```

Prepared By Mr.EBIN PM EDULINE 10



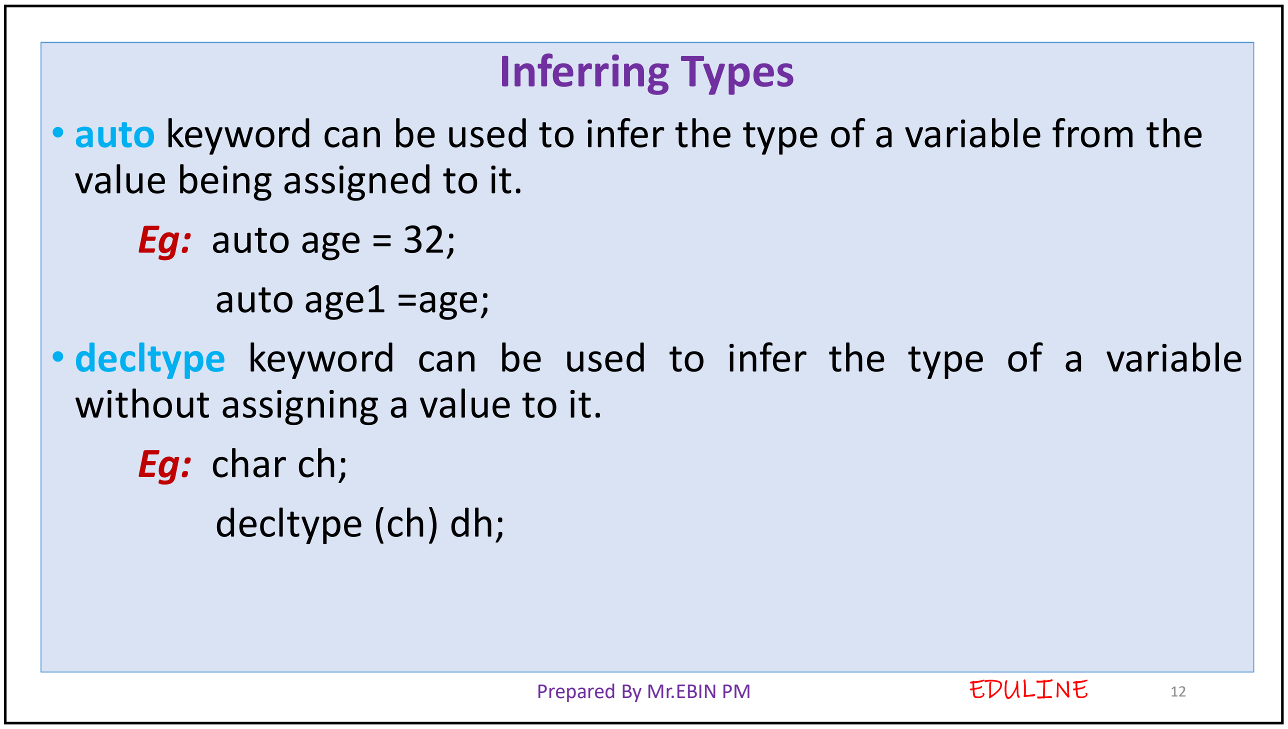
Flexible Initializations of C++

- C++ offers multiple ways to initialize a variable

```
int age = 32;
int age (32);
int age {32};
```

- All are valid and equivalent
- A variable can be defined just before the point of usage

Prepared By Mr.EBIN PM EDULINE 11



Inferring Types

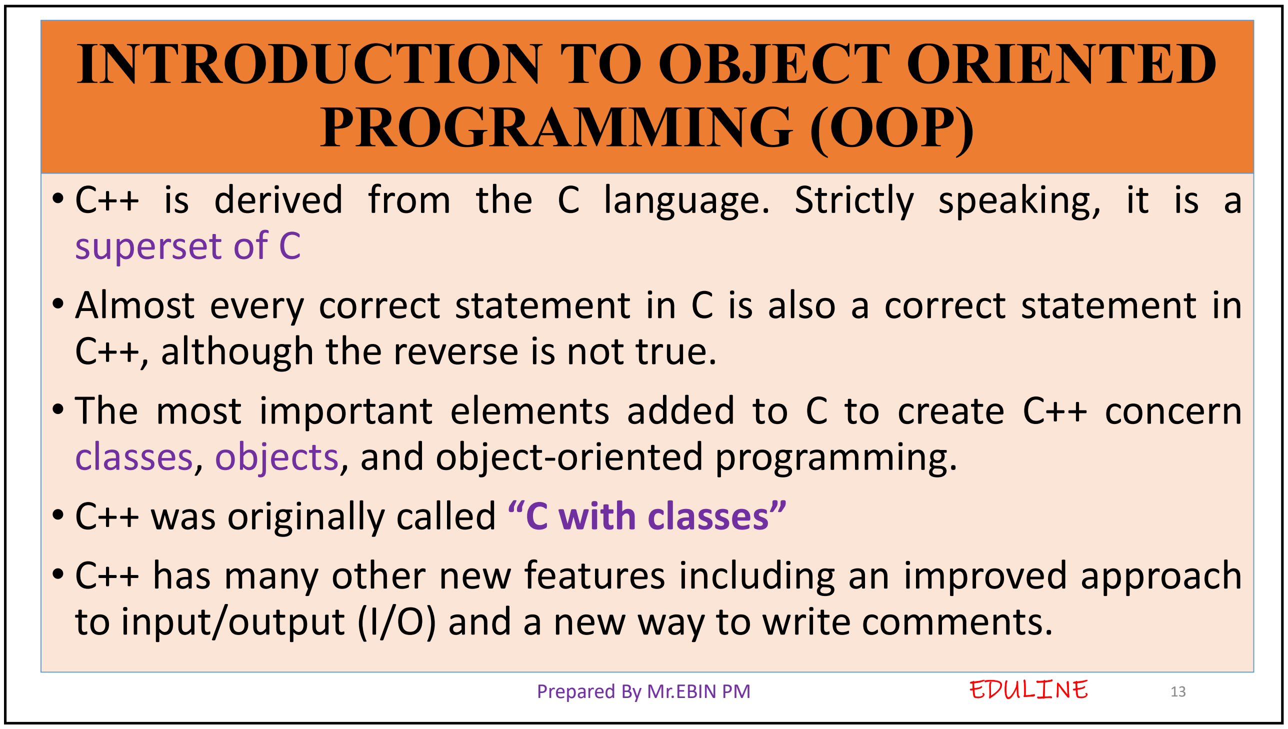
- **auto** keyword can be used to infer the type of a variable from the value being assigned to it.

Eg:
```
auto age = 32;
auto age1 =age;
```

- **decltype** keyword can be used to infer the type of a variable without assigning a value to it.

Eg:
```
char ch;
decltype (ch) dh;
```

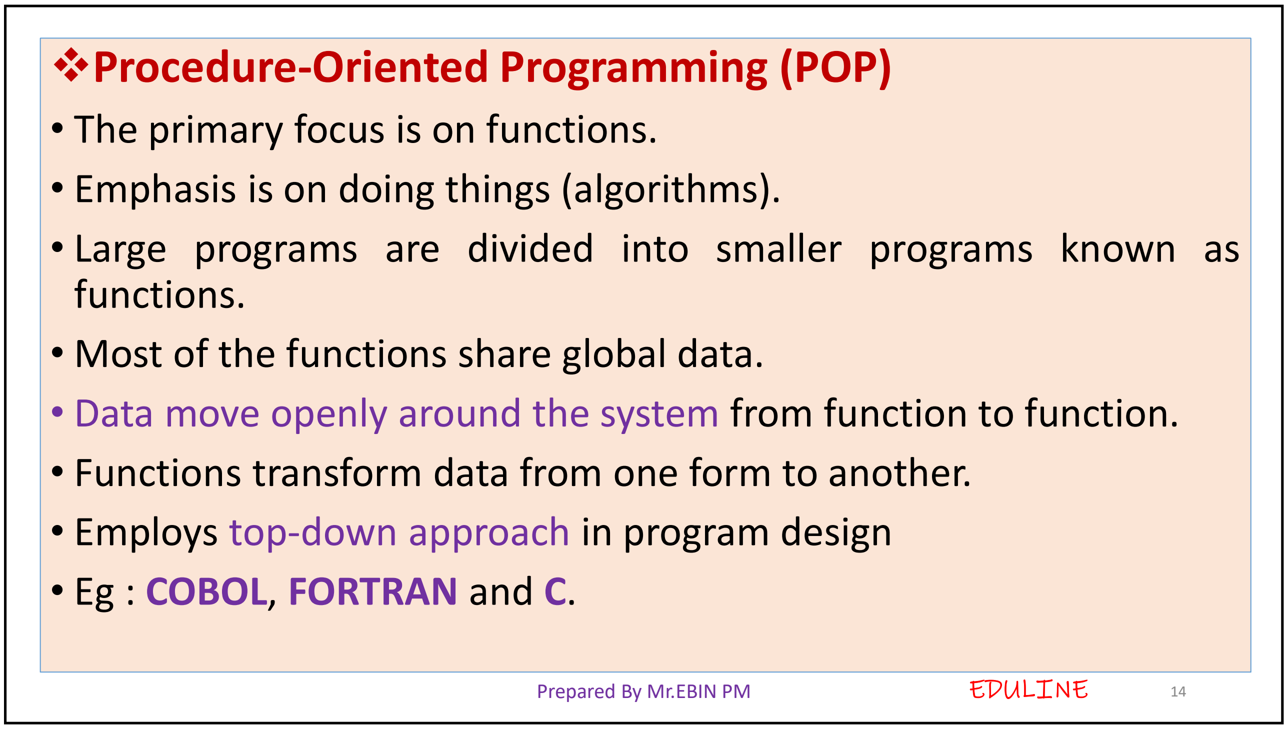
Prepared By Mr.EBIN PM EDULINE 12



INTRODUCTION TO OBJECT ORIENTED PROGRAMMING (OOP)

- C++ is derived from the C language. Strictly speaking, it is a superset of C
- Almost every correct statement in C is also a correct statement in C++, although the reverse is not true.
- The most important elements added to C to create C++ concern classes, objects, and object-oriented programming.
- C++ was originally called **“C with classes”**
- C++ has many other new features including an improved approach to input/output (I/O) and a new way to write comments.

Prepared By Mr.EBIN PM EDULINE 13

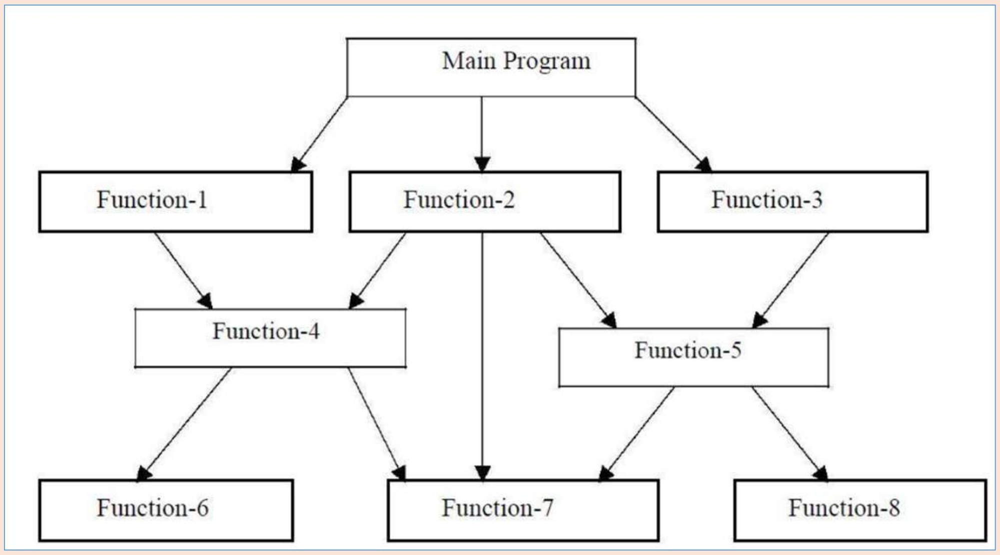


❖Procedure-Oriented Programming (POP)

- The primary focus is on functions.
- Emphasis is on doing things (algorithms).
- Large programs are divided into smaller programs known as functions.
- Most of the functions share global data.
- Data move openly around the system from function to function.
- Functions transform data from one form to another.
- Employs top-down approach in program design
- Eg : **COBOL**, **FORTRAN** and **C**.

Prepared By Mr.EBIN PM EDULINE 14

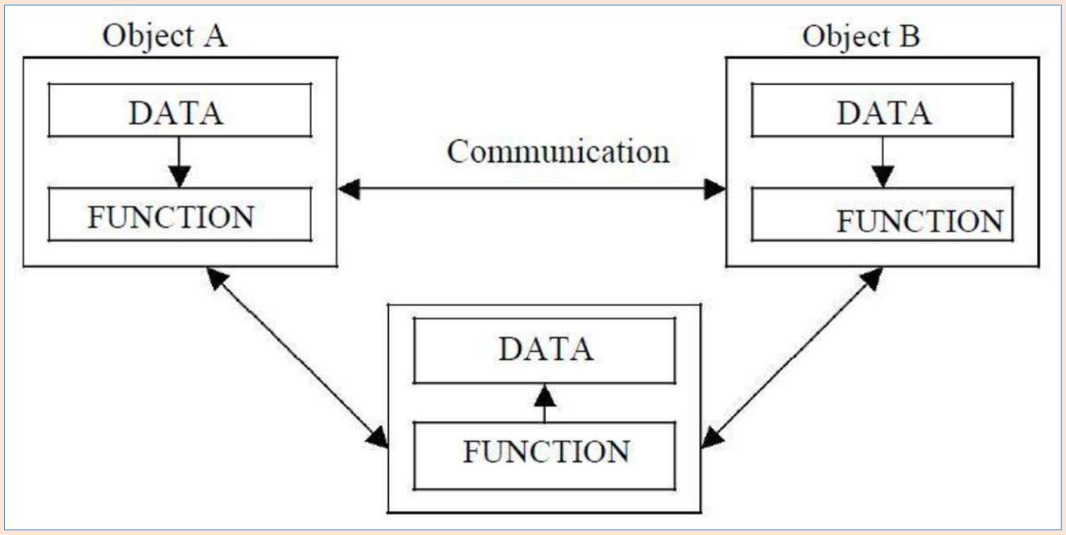
Typical structure of procedural oriented programs



❖Object Oriented Programming (OOP)

- Emphasis is on data rather than procedure.
- Programs are divided into what are known as objects.
- Data structures are designed such that they characterize the objects.
- Functions that operate on the data of an object are ties together in the data structure.
- Data is hidden and cannot be accessed by external function.
- Objects may communicate with each other through function.
- New data and functions can be easily added whenever necessary.
- Follows bottom up approach in program design.

Organization of data and function in OOP



Prepared By Mr.EBIN PM EDULINE 17

BASIC CONCEPTS OF OBJECT ORIENTED PROGRAMMING

➢It is necessary to understand some of the concepts used extensively in object-oriented programming. These include:

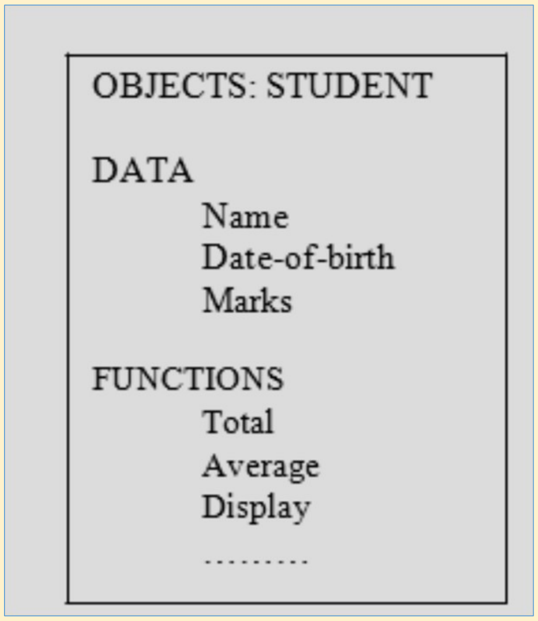
- **OBJECTS**
- **CLASSES**
- **DATA ABSTRACTION & ENCAPSULATION**
- **INHERITANCE**
- **POLYMORPHISM**
- **DYNAMIC BINDING**
- **MESSAGE PASSING**

Prepared By Mr.EBIN PM EDULINE 18

1. OBJECTS

- Objects are the variables of the type class.
- Object represents an entity that can store data and has its interface through functions.
- Object is an identifiable entity with some characteristics and behavior.
- Characteristics of an object are represented by its data, and behavior is represented by its associated functions.
- Objects are the basic run time entities in an object- oriented system.
- They may represent a person, a place, a bank account, a table of data or any item that the program has to handle

Representing an object



OBJECTS: STUDENT

DATA
- Name
- Date-of-birth
- Marks

FUNCTIONS
- Total
- Average
- Display
- ....

2. CLASSES

- A class is a way to bind the data describing an entity and its associated functions together
- In C++, class makes a data type that is used to create objects of this type.
- In fact, objects are variables of the type class.
- Once a class has been defined, we can create any number of objects belonging to that class.
- Each object is associated with the data of type class with which they are created.
- A class is thus a collection of objects similar types.

- For examples, Mango, Apple and orange members of class fruit.
- Classes are user-defined data types
- If fruit has been defines as a class, then the statement

Fruit Mango;

Will create an object mango belonging to the class fruit.

- A class is thus a description of a number of similar objects.
- An object is often called an "instance" of a class.

3. DATA ABSTRACTION AND ENCAPSULATION

- The wrapping up of data and function into a single unit (called class) is known as encapsulation.
- Data and encapsulation is the most striking feature of a class.
- The data is not accessible to the outside world, and only those functions which are wrapped in the class can access it.
- These functions provide the interface between the object‘s data and the program.
- This insulation of the data from direct access by the program is called data hiding or information hiding

- Abstraction refers to the act of representing essential features without including the background details or explanation.
- Classes use the concept of abstraction.
- They encapsulate all the essential properties of the object that are to be created.
- The attributes are sometime called data members because they hold information.
- The functions that operate on these data are sometimes called methods or member function.

4. INHERITANCE

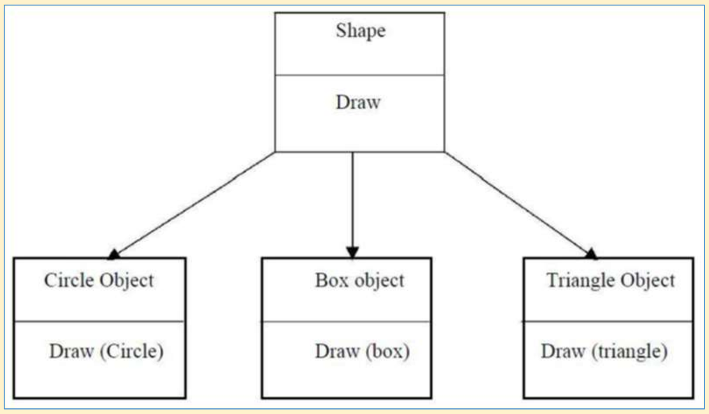
- Inheritance is the process by which objects of one class acquired the properties of objects of another classes.
- It supports the concept of hierarchical classification
- In OOP, the concept of inheritance provides the idea of reusability.
- This means that we can add additional features to an existing class without modifying it.
- This is possible by deriving a new class from the existing one.
- The new class will have the combined feature of both the classes.
- The class, whose properties are inherited, is called Base class or Super class and the class that inherits these properties, is called Derived class or Sub class.

5. POLYMORPHISM

- Polymorphism is another important OOP concept.
- Polymorphism, a Greek term, means the ability to take more than on form.
- An operation may exhibit different behavior is different instances.
- The behavior depends upon the types of data used in the operation.
- For example, consider the operation of addition. For two numbers, the operation will generate a sum. If the operands are strings, then the operation would produce a third string by concatenation.

- The process of making an operator to exhibit different behaviors in different instances is known as operator overloading.
- Using a single function name to perform different type of task is known as function overloading.

Polymorphism



6. DYNAMIC BINDING

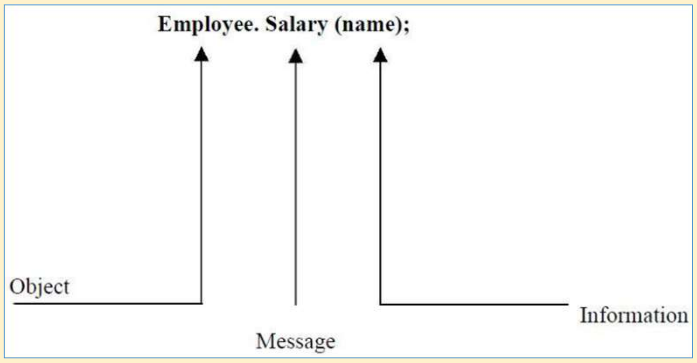
- Binding refers to the linking of a procedure call to the code to be executed in response to the call.
- Dynamic binding means that the code associated with a given procedure call is not known until the time of the call at run time.

7. MESSAGE PASSING

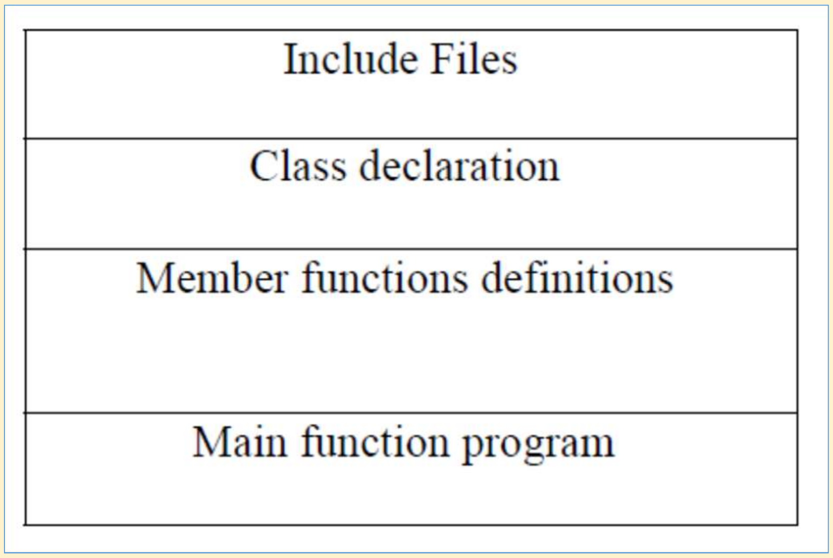
- An object-oriented program consists of a set of objects that communicate with each other
- Objects communicate with one another by sending and receiving information much the same way as people pass messages to one another.

- A Message for an object is a request for execution of a procedure, and therefore will invoke a function (procedure) in the receiving object that generates the desired results.
- Message passing involves specifying the name of object, the name of the function (message) and the information to be sent.

Example:



STRUCTURE OF C++ PROGRAM



C Structure & Its Limitations

- Structure provides a method for packing together data of different types. It is a user defined data type .

```
struct book-bank
{
    char title[20];
    char author[15];
    int pages ;
    float price;
} book1, book2, book3;
```

- do not permit data hiding
- Structure members are public members.

DEFINITION AND DECLARATION OF A CLASS

- A class in C++ combines related data and functions together. It makes a data type which is used for creating objects of this type.
- Classes represent real world entities that have both data type properties (characteristics) and associated operations (behaviour).
- A class is a way to bind the data and its associated function together.
- It allows the data to be hidden, if necessary, from external use.
- A class specification has two parts:

➢ **Class declaration**

➢ **Class function definition**

The syntax of a class definition (declaration) is shown below:

```
class class_name
        {
            Private:
              Variable declaration;  // data members
               Function declaration; // member functions (methods)
            Protected:
                 Variable declaration;
                 Function declaration;
               Public:
                 Variable declaration;
                 Function declaration;
            };
```

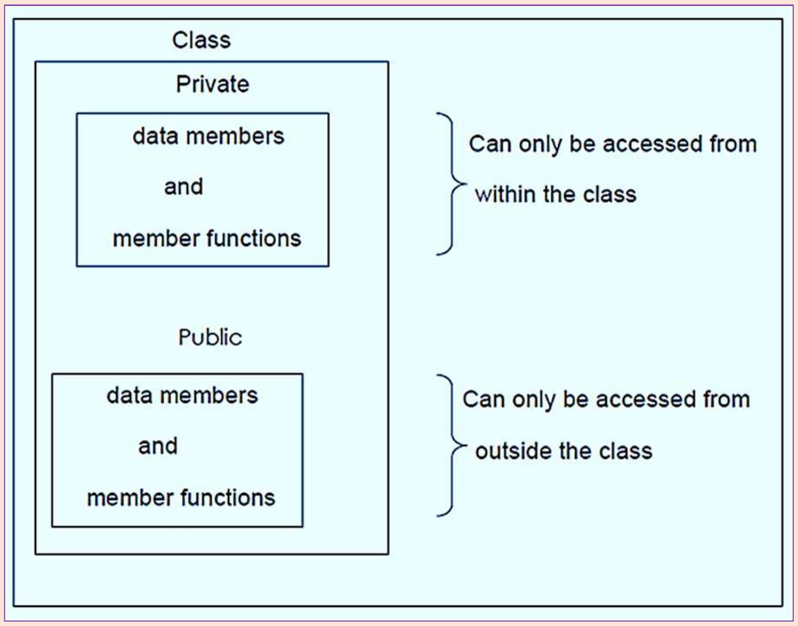
- In C++, the keywords private and public and protected are called access specifiers (access Levels or visibility Labels).
- The data hiding concept in C++ is achieved by using the keyword private.
- Private data and functions can only be accessed from within the class itself.
- Public data and functions are accessible outside the class also.
- The use of keyword private is optional.
- By default, the members of a class are private. If both the labels are missing, then by default all the members are private. Such a class is completely hidden from the outside world and does not serve any purpose.

- Variable declared inside the class are known as data members and the functions are known as member functions.
- Only the member function can have access to the private data members and private functions.
- However the public members (both functions and data) can be accessed from outside the class.
- The data declared under Private section are hidden and safe from accidental manipulation.
- Though the user can use the private data but not by accident.
- The functions that operate on the data are generally public so that they can be accessed from outside the class.

❖**Example**

```
class Abc
{
  private:
    int x,y;
  public:
    int z;
    int add (int a,int b)
      {
         int c = a+b;
         return c;
      }
     int sub (int a,int b)
      {
         int c=a-b;
         return c;
      }
};
Abc O1, O2;
```

```
class Abc
{
    int x,y;     // private by default
  public:
    int z;
    int add (int a,int b)
      {
         int c = a+b;
         return c;
      }
    int sub (int a,int b)
      {
         int c=a-b;
         return c;
      }
};
Abc O1, O2;
```

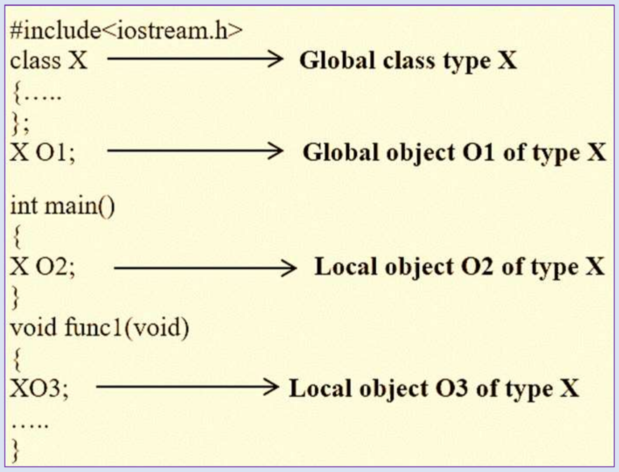

Class
Private
data members and member functions
Can only be accessed from within the class
Public
data members and member functions
Can only be accessed from outside the class

Prepared By Mr.EBIN PM EDULINE 37

THE SCOPE RULES AND CLASSES

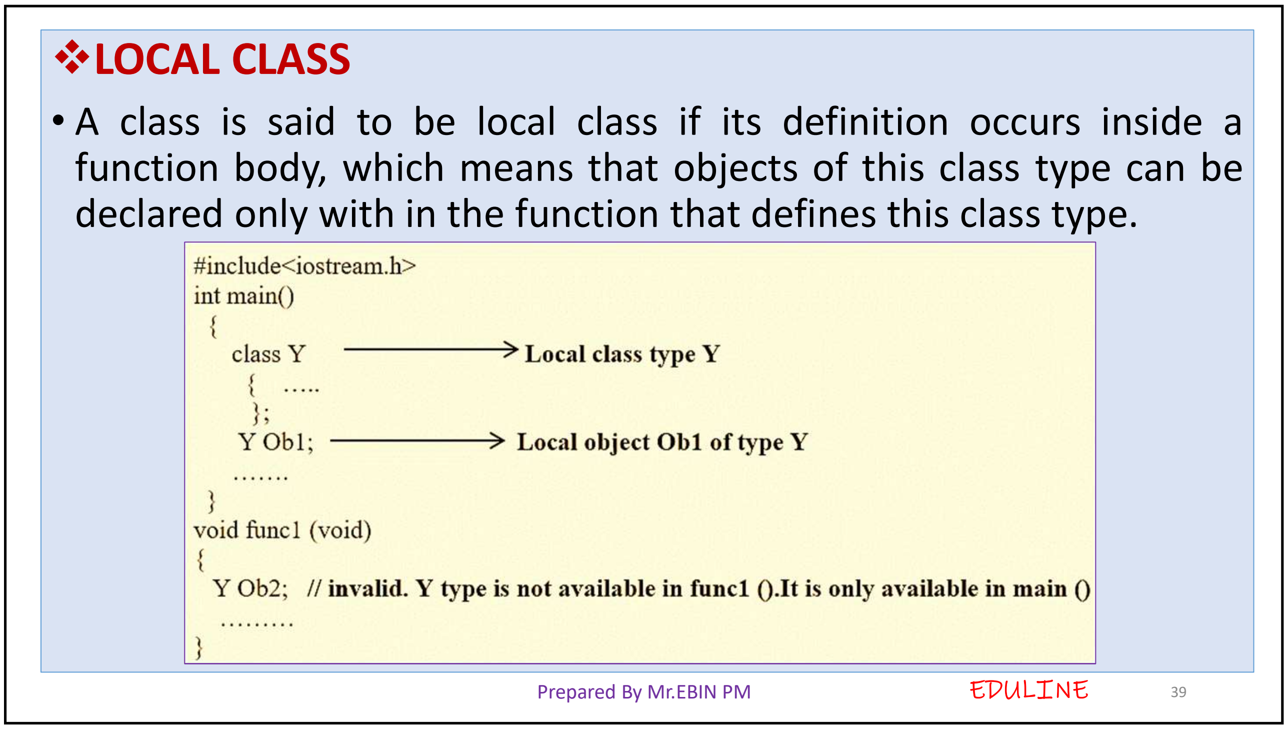
❖GLOBAL CLASS

A class is said to be global class if its definition occurs outside the bodies of all functions in a program, which means that object of this class type can be declared from anywhere in the program.



```
#include<iostream.h>
class X        ————————> Global class type X
{…..
};
X O1;          ————————> Global object O1 of type X
int main()
{
X O2;          ————————> Local object O2 of type X
}
void func1(void)
{
XO3;           ————————> Local object O3 of type X
…..
}
```

Prepared By Mr.EBIN PM EDULINE 38

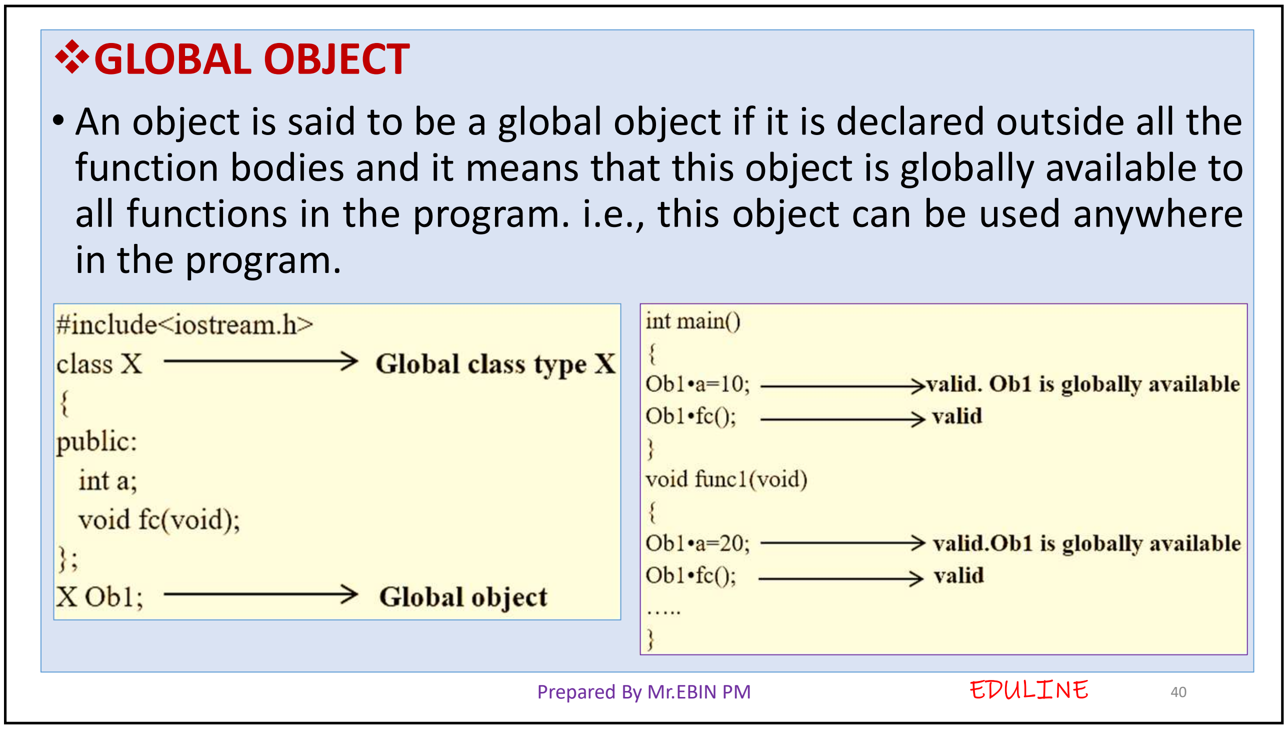


❖LOCAL CLASS

- A class is said to be local class if its definition occurs inside a function body, which means that objects of this class type can be declared only with in the function that defines this class type.

```
#include<iostream.h>
int main()
  {
    class Y          → Local class type Y
      {   .....
       };
     Y Ob1;          → Local object Ob1 of type Y
    .......
  }
void func1 (void)
{
  Y Ob2;  // invalid. Y type is not available in func1 ().It is only available in main ()
   .........
}
```

Prepared By Mr.EBIN PM EDULINE 39



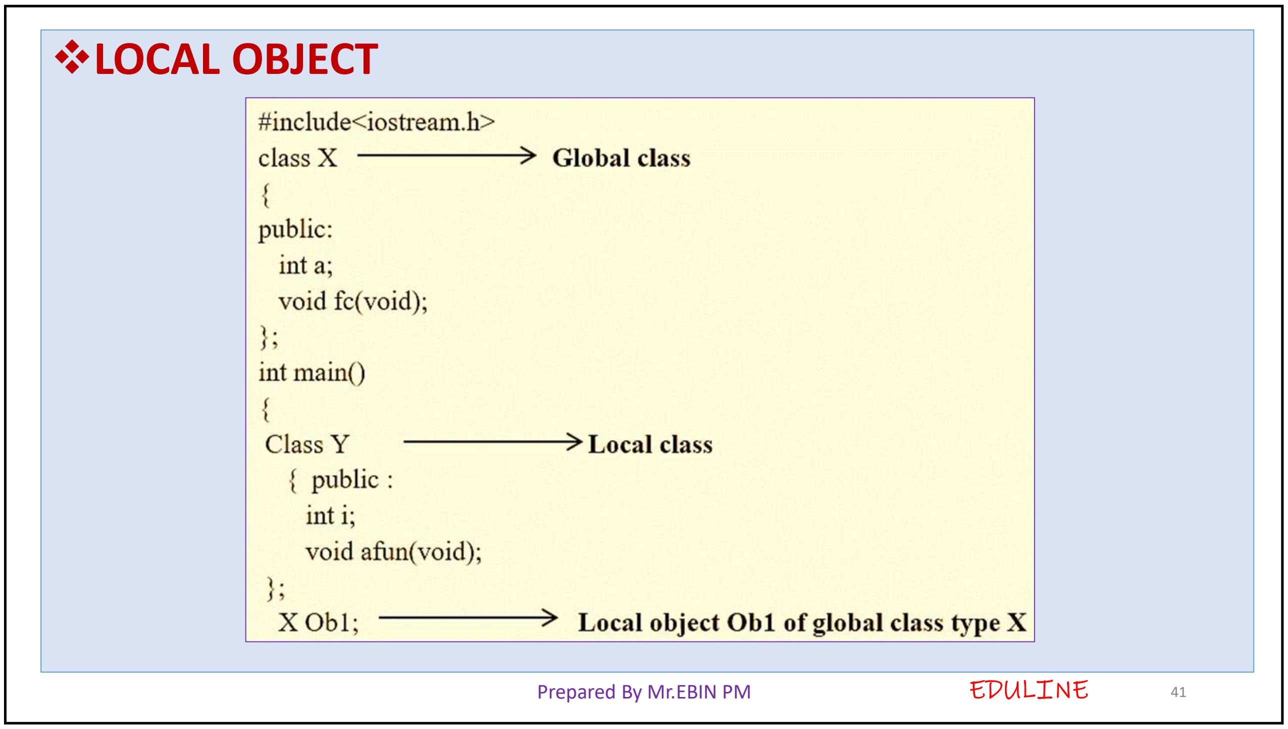
❖GLOBAL OBJECT

- An object is said to be a global object if it is declared outside all the function bodies and it means that this object is globally available to all functions in the program. i.e., this object can be used anywhere in the program.

```
#include<iostream.h>
class X          → Global class type X
{
public:
  int a;
  void fc(void);
};
X Ob1;           → Global object
```

```
int main()
{
Ob1•a=10;     →valid. Ob1 is globally available
Ob1•fc();     → valid
}
void func1(void)
{
Ob1•a=20;     → valid.Ob1 is globally available
Ob1•fc();     → valid
.....
}
```

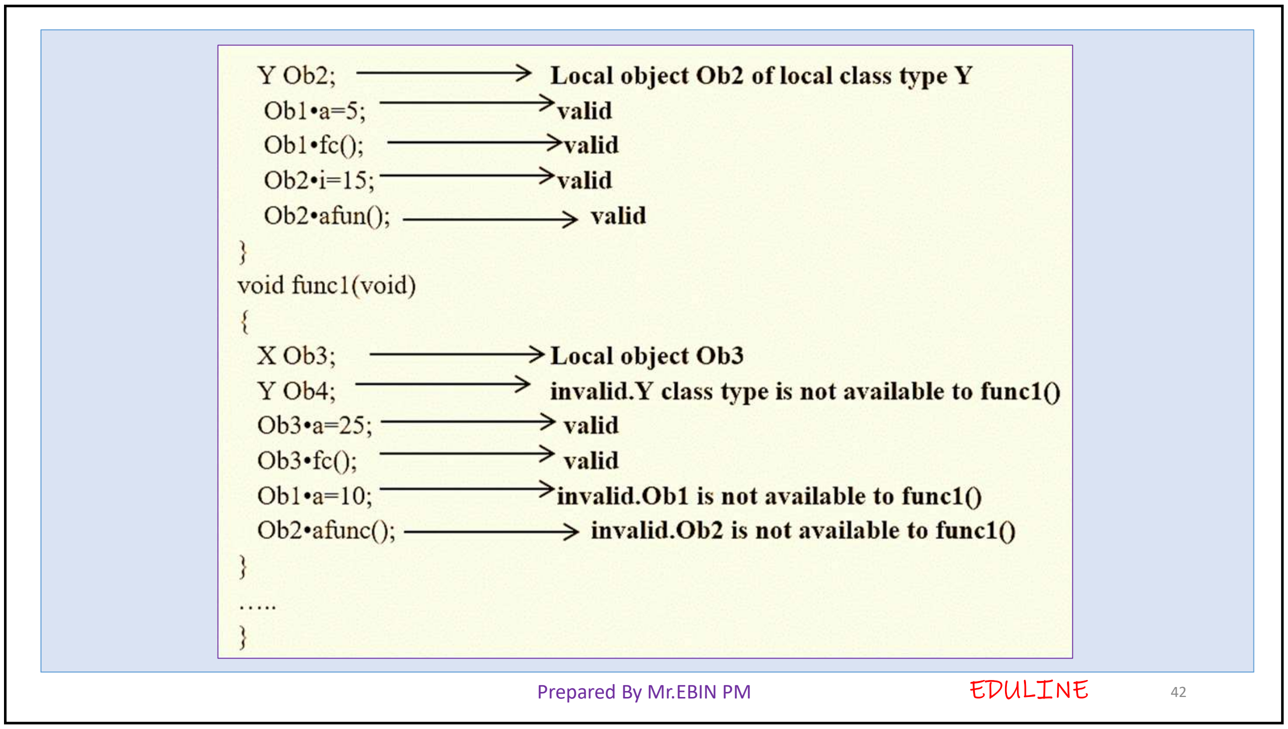
Prepared By Mr.EBIN PM EDULINE 40



❖LOCAL OBJECT

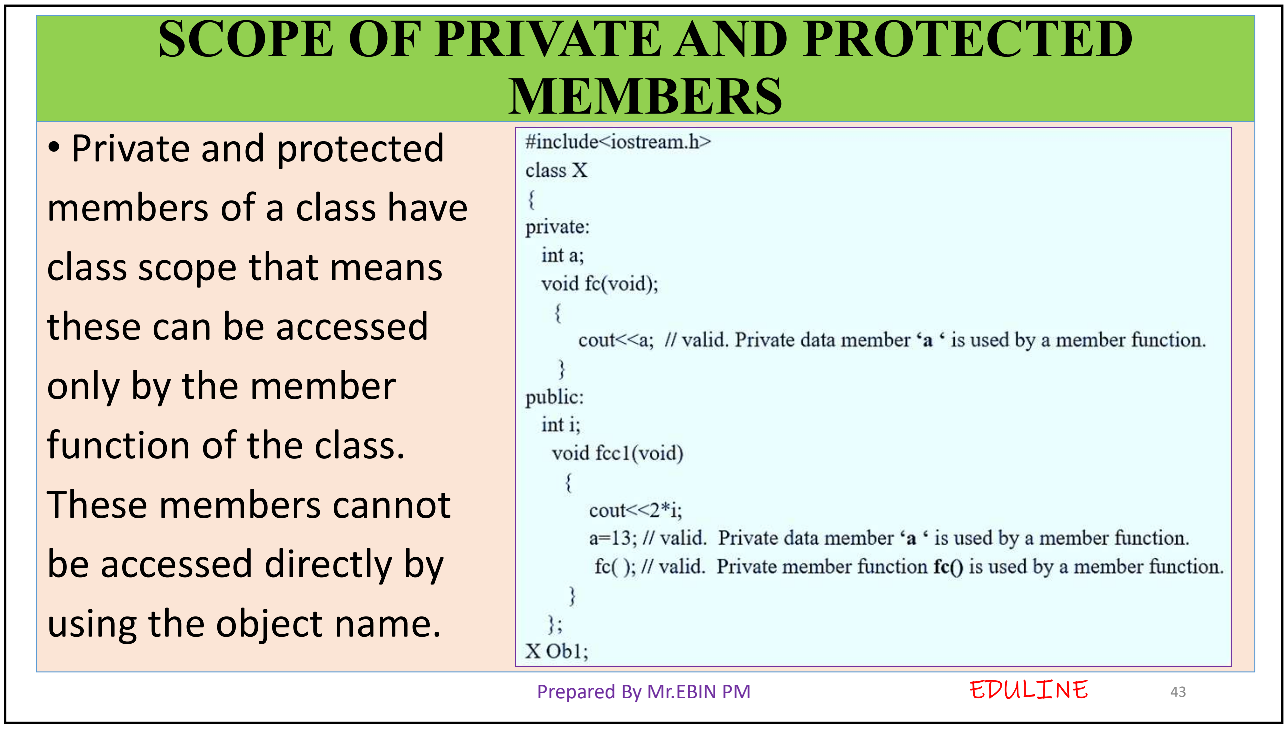
```
#include<iostream.h>
class X          ——————→ Global class
{
public:
  int a;
  void fc(void);
};
int main()
{
 Class Y         ——————→ Local class
   { public :
      int i;
      void afun(void);
 };
  X Ob1;         ——————→ Local object Ob1 of global class type X
```

Prepared By Mr.EBIN PM EDULINE 41



```
  Y Ob2;         ——————→ Local object Ob2 of local class type Y
   Ob1•a=5;      ——————→ valid
   Ob1•fc();     ——————→ valid
   Ob2•i=15;     ——————→ valid
   Ob2•afun();   ——————→ valid
}
void func1(void)
{
  X Ob3;         ——————→ Local object Ob3
  Y Ob4;         ——————→ invalid.Y class type is not available to func1()
  Ob3•a=25;      ——————→ valid
  Ob3•fc();      ——————→ valid
  Ob1•a=10;      ——————→ invalid.Ob1 is not available to func1()
  Ob2•afunc();   ——————→ invalid.Ob2 is not available to func1()
}
…..
}
```

Prepared By Mr.EBIN PM EDULINE 42

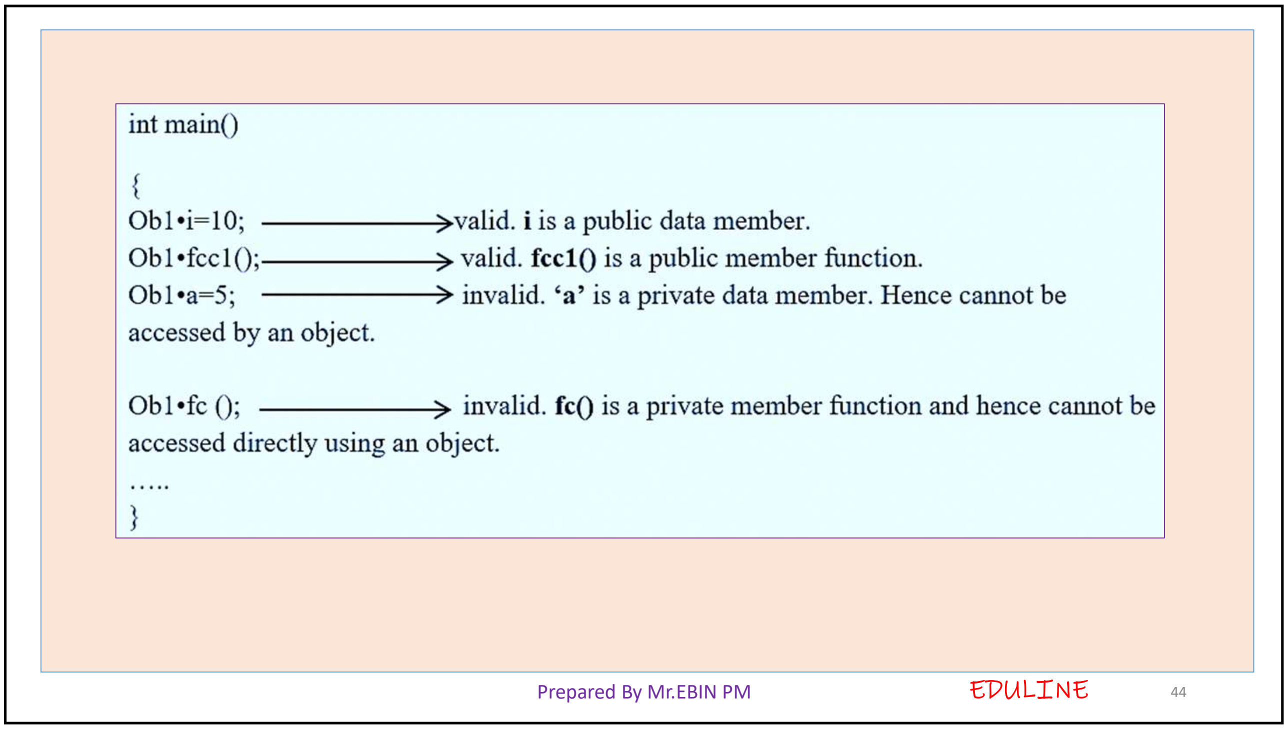


SCOPE OF PRIVATE AND PROTECTED MEMBERS

- Private and protected members of a class have class scope that means these can be accessed only by the member function of the class. These members cannot be accessed directly by using the object name.

```
#include<iostream.h>
class X
{
private:
  int a;
  void fc(void);
    {
       cout<<a;  // valid. Private data member 'a ' is used by a member function.
    }
public:
  int i;
    void fcc1(void)
     {
        cout<<2*i;
        a=13; // valid.  Private data member 'a ' is used by a member function.
         fc( ); // valid.  Private member function fc() is used by a member function.
     }
  };
X Ob1;
```

Prepared By Mr.EBIN PM EDULINE 43

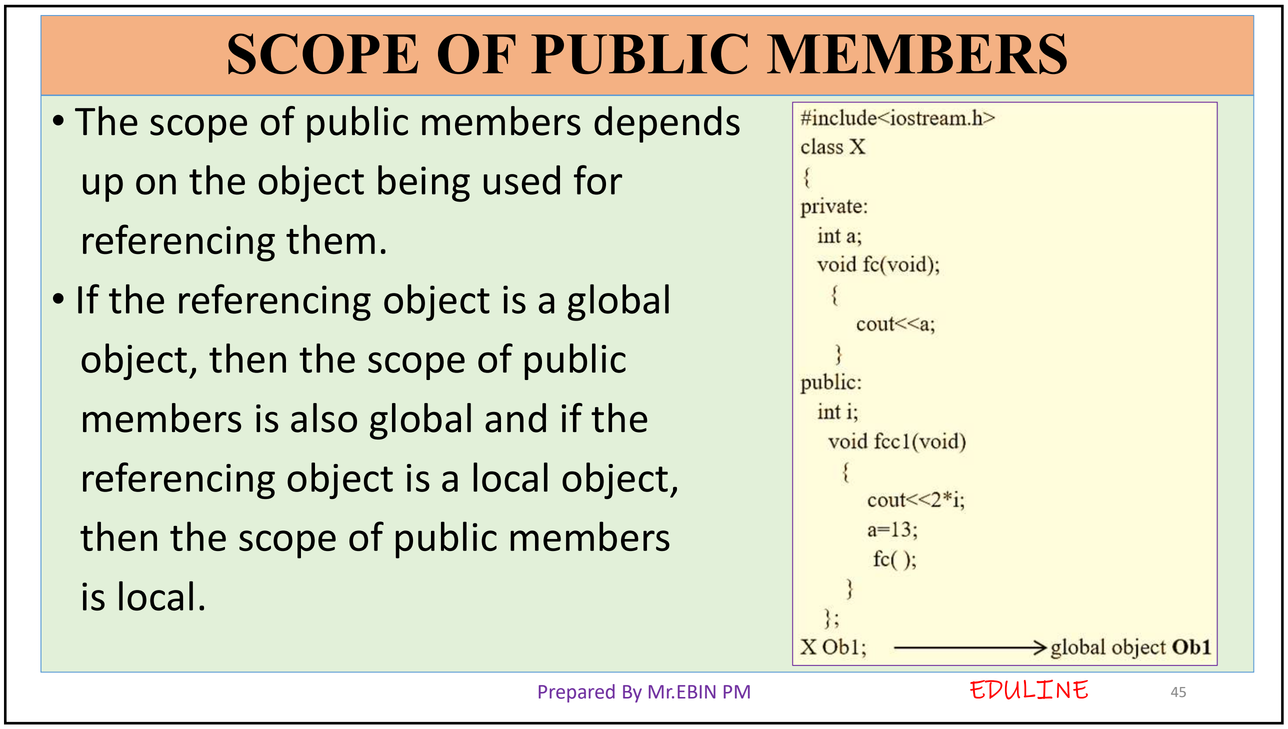


```
int main()

{
Ob1•i=10;  ————————>valid. i is a public data member.
Ob1•fcc1();————————> valid. fcc1() is a public member function.
Ob1•a=5;   ————————> invalid. 'a' is a private data member. Hence cannot be
accessed by an object.

Ob1•fc ();  ————————> invalid. fc() is a private member function and hence cannot be
accessed directly using an object.
…..
}
```

Prepared By Mr.EBIN PM EDULINE 44

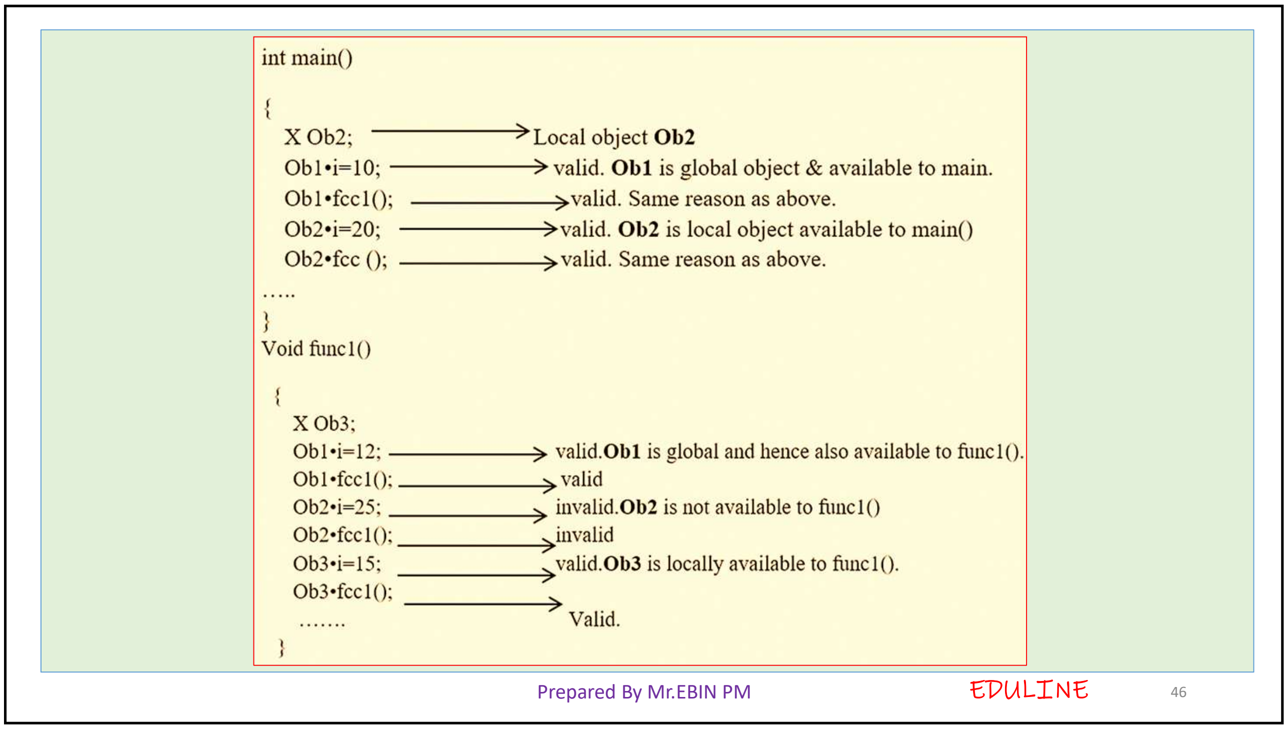


SCOPE OF PUBLIC MEMBERS

- The scope of public members depends up on the object being used for referencing them.
- If the referencing object is a global object, then the scope of public members is also global and if the referencing object is a local object, then the scope of public members is local.

```
#include<iostream.h>
class X
{
private:
  int a;
  void fc(void);
    {
        cout<<a;
    }
public:
  int i;
    void fcc1(void)
     {
         cout<<2*i;
         a=13;
          fc( );
     }
  };
X Ob1;  ——————→global object Ob1
```

Prepared By Mr.EBIN PM EDULINE 45



```
int main()

{
   X Ob2;  ——————→Local object Ob2
   Ob1•i=10;  ——————→ valid. Ob1 is global object & available to main.
   Ob1•fcc1();  ——————→valid. Same reason as above.
   Ob2•i=20;  ——————→valid. Ob2 is local object available to main()
   Ob2•fcc ();  ——————→valid. Same reason as above.
…..
}
Void func1()

 {
    X Ob3;
    Ob1•i=12;  ——————→ valid.Ob1 is global and hence also available to func1().
    Ob1•fcc1();  ——————→ valid
    Ob2•i=25;  ——————→ invalid.Ob2 is not available to func1()
    Ob2•fcc1();  ——————→invalid
    Ob3•i=15;  ——————→valid.Ob3 is locally available to func1().
    Ob3•fcc1();  ——————→
     …….                   Valid.
 }
```

Prepared By Mr.EBIN PM EDULINE 46

- In the above code fragment, Ob1 is a global object and thus the public members of Ob1 can be accessed from any of the functions in the program.
- Ob2 is a local object, local to function main (), and thus, the public members of Ob2 can be accessed only in main () and not in func1 ().
- Similarly, Ob3 being local to func1 (), its public members can only be accessed inside func1 () and not anywhere else.

MEMBER FUNCTION (METHOD) DEFINITION

In C++, the member functions can be coded in two ways:

➢**Inside class definition**

➢**Outside class definition using scope resolution operator (::)**

❖**Inside class definition**

- When the function is defined inside a class, it is treated as an **inline function.**
- Only small functions are defined inside the class definition.
- In case of inline function the compiler inserts the code of the body of the function at the place where it is invoked (called) and in doing so the program execution is faster but memory penalty is there.

- Within inline code, the compiler replaces the function call statement with the function code itself (this process is called **expansion**) and then compiles the entire code.
- Thus, with inline functions, the compiler does not have to jump to another location to execute the function, and then jump back as the code of the called function is already available to the calling program.

EXAMPLE

```
class Item
{
   int number;
   float cost;
 public:
  void getdata (int a, float b); // declaration
  void putdata(void)
       {
          cout<<number<<"\n";     // definition inside the class
          cout<<cost<<"\n";
       }
  };
```

➢**Making an outside function inline:** We can define a member function outside the class definition and still make it inline by just using the qualifier **inline** in the header line of the function definition.

Example

```
class Item
{
   .........
   .........
 public :
 void getdata (int a, float b);

};

inline void item :: getdata (int a, float b)
{
   number = a;
   cost = b;

}
```

❖Outside class definition using scope resolution operator (::)

The syntax for a member function definition outside the class definition is :

```
return_type  name_of_the_class :: function_name (argument list)
      {
            body of function
      }
```

- Here the operator **::** known as scope resolution operator helps in defining the member function outside the class. It specifies that the scope of the function is restricted to the class class_name.
- **Scope:** The part of program in which an item (variable or function) is accessible.

Prepared By Mr.EBIN PM EDULINE 51

OBJECT CREATION

- In c++, the class variables are known as objects. General method of creating object is

Class_ name object_ name;

Eg: Item x;

- X is called an object of type item. We can also declare more than one object in one statement.

Eg: Item x,y,z;

- At the time of declaration of an object, the necessary memory space is allocated to an object.
- Note that class specification does not create any memory space for the objects.

Prepared By Mr.EBIN PM EDULINE 52

REFERENCING (ACCESSING) CLASS MEMBERS

- The members of a class are referenced using objects of the class. Consider the following example

```
class Abc
{
    int x,y;    // private by default
  public:
    int z;
    int add (int a,int b)
      {
          int c = a+b;
          return c;
      }
    int sub (int a,int b)
      {
          int c=a-b;
          return c;
      }
};
Abc O1, O2;
```

- The private data of a class can be accessed only through the member functions of that class.
- The public data can be accessed by the non-member functions through the objects of that class.

The general format for calling a member function is:

Object_name • function_name (actual- arguments);

- For instance, to call add() of class Abc, we may give

O1•add(2,3); // O1 and O2 are objects of class Abc

O2•add(8,5);

To access public data member, the format used is:

Object_name • public data-member

- For instance, to reference z of Abc, we may give

  O1•z or O2•z
- But if we give

  O1•x or O2•x the compiler will report an error ,since x is a private data member of Abc
- **Observe the following statements:**

```
O1•sub(7,4); // valid
O2•add(4,5); //valid
O1•z=7;      //valid
O1•x=5;      // invalid
```

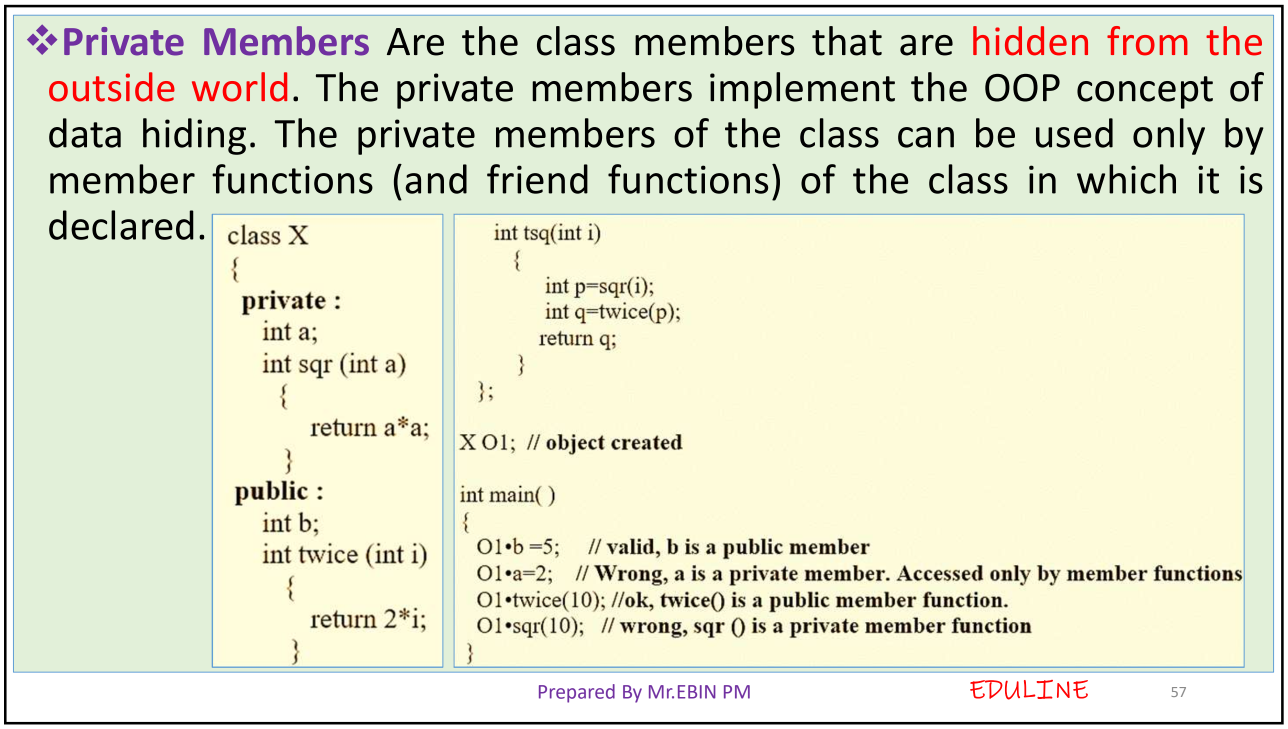
SCOPE OF CLASSES AND ITS MEMBERS

❖**Public Members** are the members (data members and member function) that can be used by any function. Consider the example:

```
class X
{
 public:
   int a;
   int sqr (int a)
     {
        return a*a;
     }
  };

X O1; // object created of type X
```

```
int main( )
{
  int b;
  O1•a =10;        // valid
   b=O1•sqr (15); // valid
 }
```



❖**Private Members** Are the class members that are hidden from the outside world. The private members implement the OOP concept of data hiding. The private members of the class can be used only by member functions (and friend functions) of the class in which it is declared.

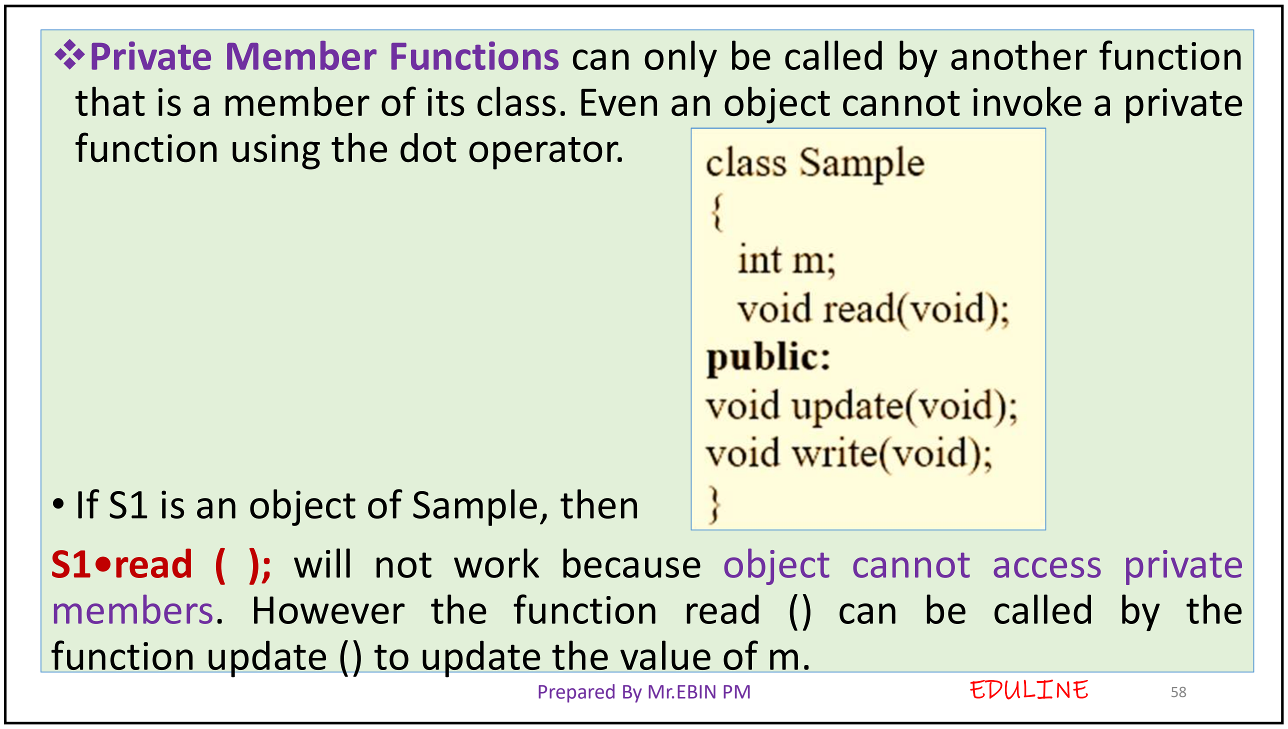
```
class X
{
 private :
   int a;
   int sqr (int a)
    {
       return a*a;
    }
 public :
   int b;
   int twice (int i)
    {
       return 2*i;
    }
```

```
  int tsq(int i)
     {
        int p=sqr(i);
        int q=twice(p);
        return q;
     }
  };

X O1;  // object created

int main( )
{
  O1•b =5;    // valid, b is a public member
  O1•a=2;    // Wrong, a is a private member. Accessed only by member functions
  O1•twice(10); //ok, twice() is a public member function.
  O1•sqr(10);   // wrong, sqr () is a private member function
}
```

Prepared By Mr.EBIN PM EDULINE 57



❖**Private Member Functions** can only be called by another function that is a member of its class. Even an object cannot invoke a private function using the dot operator.

```
class Sample
{
  int m;
  void read(void);
public:
void update(void);
void write(void);
}
```

- If S1 is an object of Sample, then

S1•read ( ); will not work because object cannot access private members. However the function read () can be called by the function update () to update the value of m.

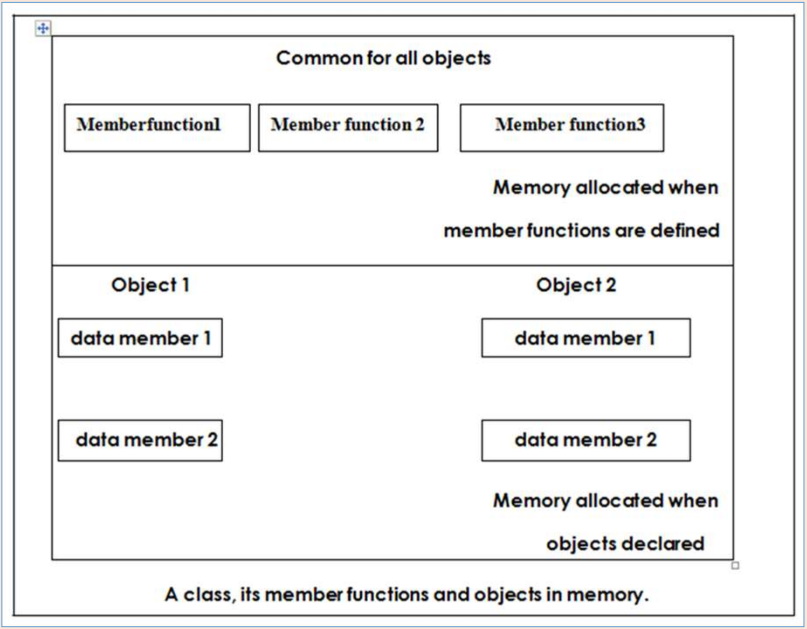
Prepared By Mr.EBIN PM EDULINE 58

```
void sample :: update (void)
  {
      read (); //simple call ; no object used.
  }
```

❖**Protected Members** are the members that can be used only by member functions and friends of the class in which it is declared. The protected members are similar to private members that they cannot be accessed by non-member functions.

MEMORY ALLOCATION OF OBJECTS

- In C++, all the member functions of a class are created and stored when the class is defined and this memory space can be accessed by all the objects related to that class.
- Memory space is allocated separately to each object for their data members.
- Member variables store different values for different objects of a class.
- The member functions are created and placed in the memory space only once when they are defined as a part of a class specification.
- Since all the objects belonging to that class use the same member functions, no separate space is allocated for member functions when the objects are created.



A class, its member functions and objects in memory.

Prepared By Mr.EBIN PM EDULINE 61

STATIC CLASS MEMBERS

- Data members and member functions of a class in C++ may be qualified as static. We can have static data members and static member function in a class.

❖**STATIC DATA MEMBER:** It is generally used to store value common to the whole class. The static data member differs from an ordinary data member in the following ways :

1) Only a single copy of the static data member is used by all the objects.
2) It can be used within the class but its lifetime is the whole program. For making a data member static, we require :

   a) **Declare it within the class** b) **Define it outside the class**

Prepared By Mr.EBIN PM EDULINE 62

For example

```
class student
  {
      static int count; //declaration within class
      -----------------
      -----------------
      -----------------
};
```

- The static data member is defined outside the class as:
  int student :: count; //definition outside class
- The definition outside the class is a must.
- We can also initialize the static data member at the time of its definition as:
  int student :: count = 0;

❖**STATIC MEMBER FUNCTION:** A static member function can access only the static members of a class. We can do so by putting the keyword static before the name of the function while declaring it. For example,

```
class student
  {
     static int count;
     -----------------
     public :
      -----------------
      -----------------
     static void showcount (void) //static member function
          {
               cout<<"count="<<count<<"\n";
      }
  };
int student ::count=0;
```

- Here we have put the keyword static before the name of the function showcount ().
- In C++, a static member function differs from the other member functions in the following ways

1) Only static members (functions or variables) of the same class can be accessed by a static member function.
2) It is called by using the name of the class rather than an object as given below:

Name_of_the_class :: function_name

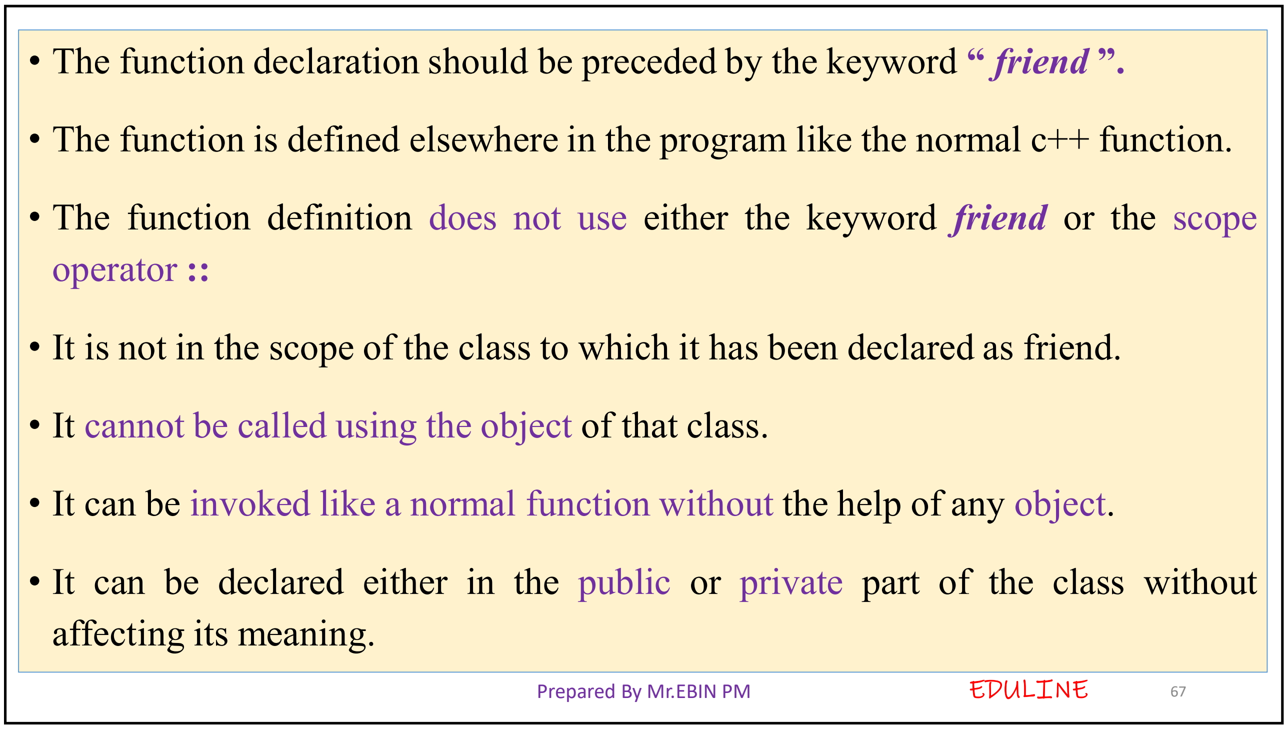
For example,

student :: showcount();

- A member function can access both ordinary members as well as static members where as a static member function can access only static members.

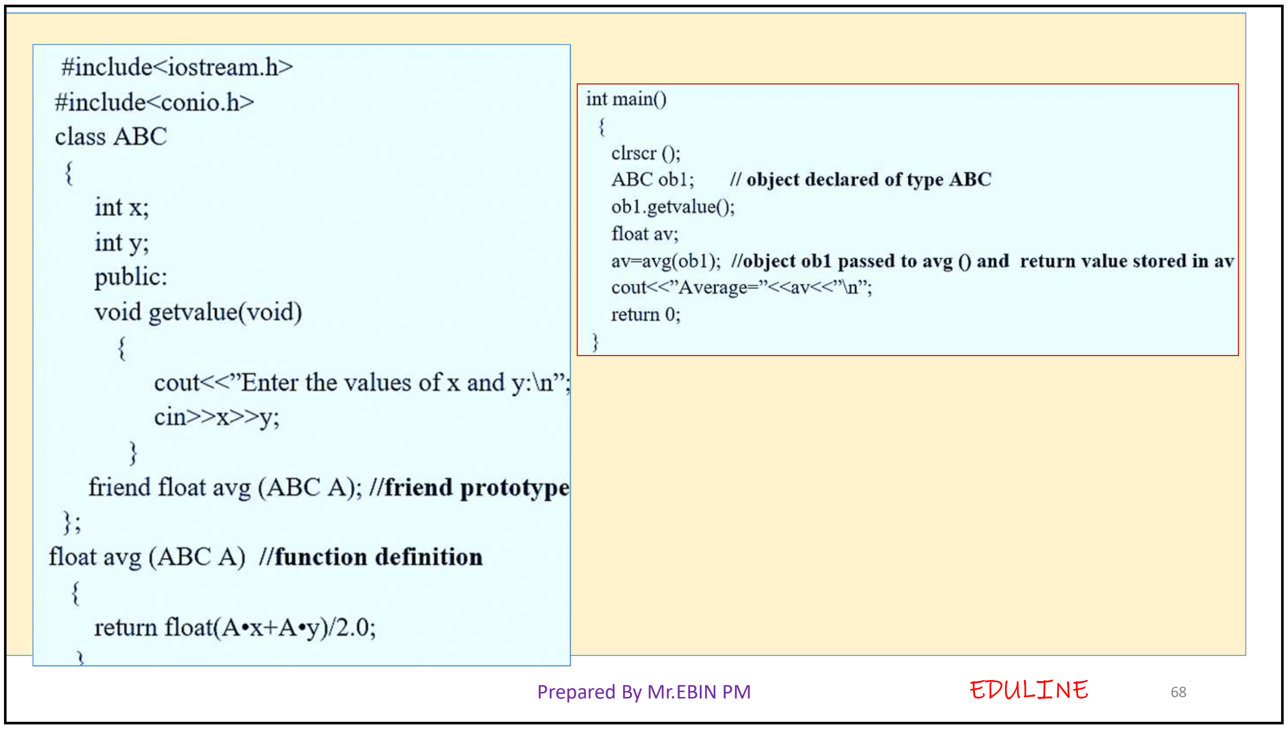
FRIEND FUNCTION

- A friend function is a non-member function, but which is given a special permission to access private and protected members of the class.
- A friend class is a class whose member functions can access another class's private and protected members.
- In some situations, the private and protected data of a class might need to be shared with some non-member of the class. In such situation, that non-member may be declared a friend of the class, and thus get the privilege of accessing private and protected members of the class.



- The function declaration should be preceded by the keyword “ ***friend*** **”.**
- The function is defined elsewhere in the program like the normal c++ function.
- The function definition does not use either the keyword ***friend*** or the scope operator **::**
- It is not in the scope of the class to which it has been declared as friend.
- It cannot be called using the object of that class.
- It can be invoked like a normal function without the help of any object.
- It can be declared either in the public or private part of the class without affecting its meaning.

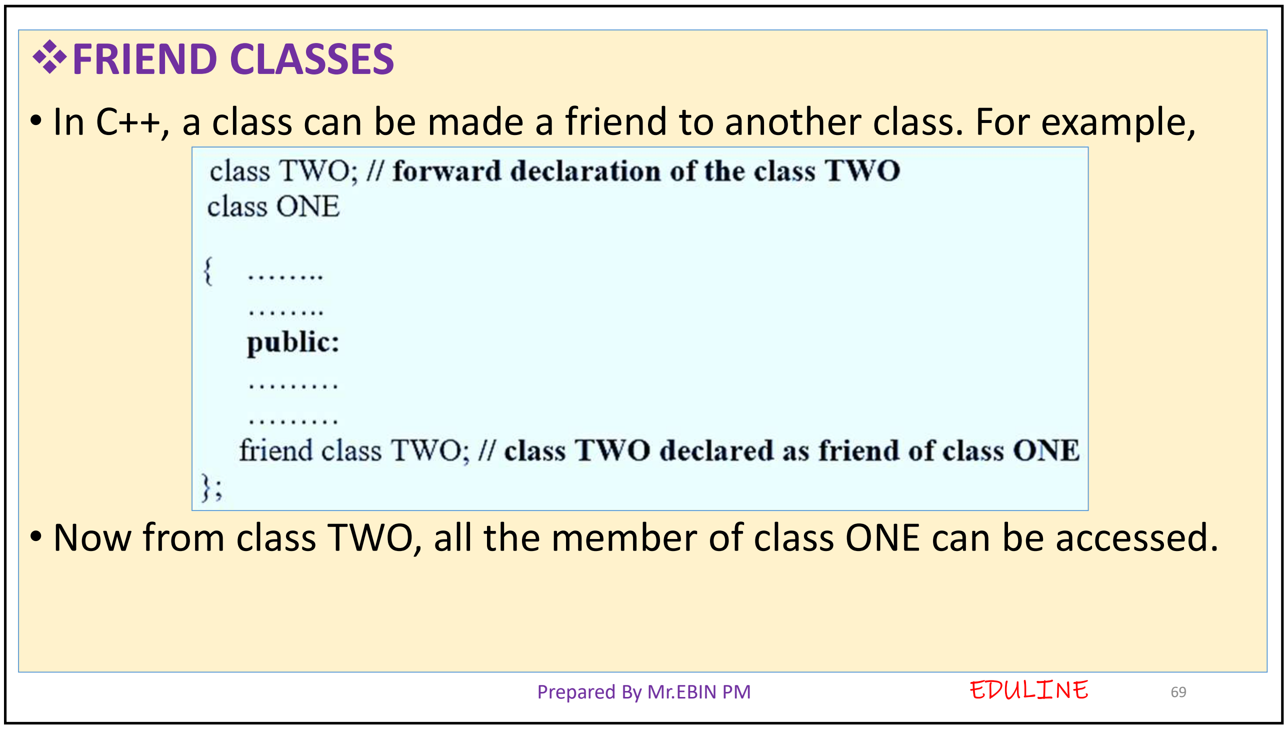
Prepared By Mr.EBIN PM EDULINE 67



```
 #include<iostream.h>
#include<conio.h>
class ABC
 {
    int x;
    int y;
    public:
    void getvalue(void)
      {
         cout<<”Enter the values of x and y:\n”;
         cin>>x>>y;
      }
    friend float avg (ABC A); //friend prototype
 };
float avg (ABC A)  //function definition
 {
    return float(A•x+A•y)/2.0;
 }
```

```
int main()
 {
   clrscr ();
   ABC ob1;     // object declared of type ABC
   ob1.getvalue();
   float av;
   av=avg(ob1);  //object ob1 passed to avg () and  return value stored in av
   cout<<”Average=”<<av<<”\n”;
   return 0;
 }
```

Prepared By Mr.EBIN PM EDULINE 68



❖FRIEND CLASSES

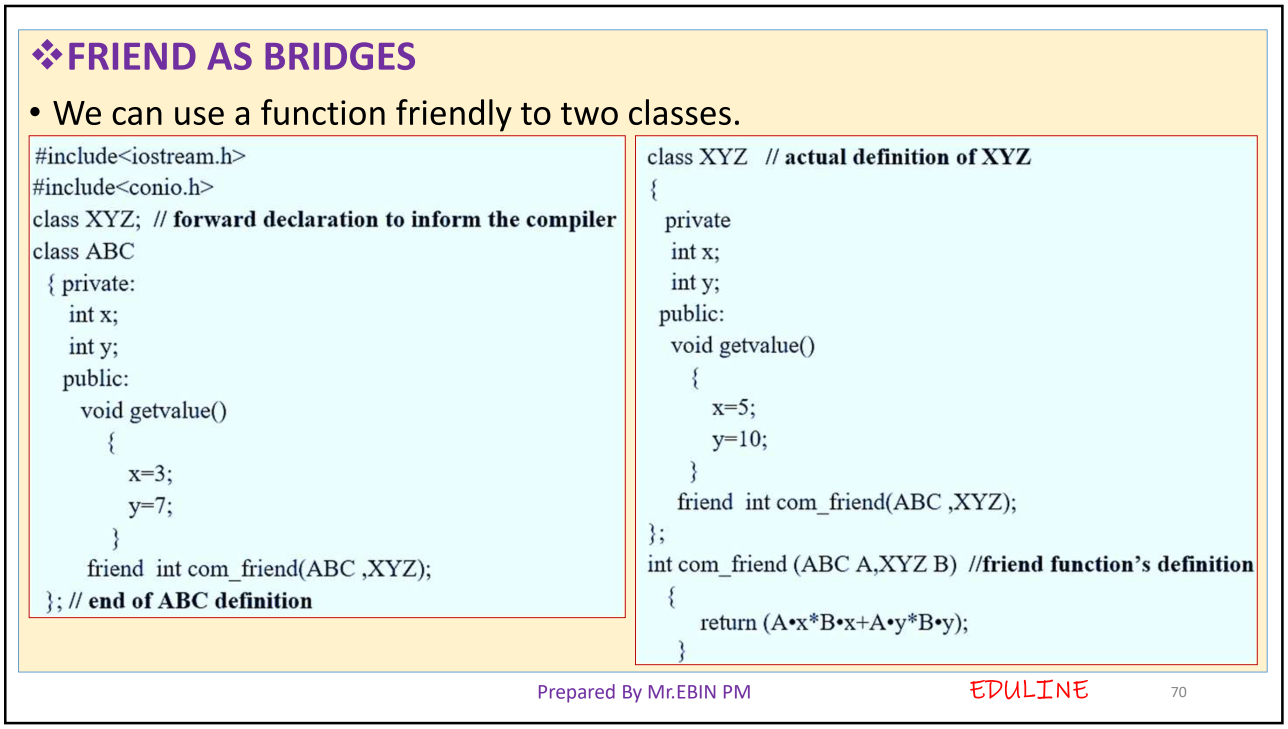
• In C++, a class can be made a friend to another class. For example,

```
class TWO; // forward declaration of the class TWO
class ONE

{   ........
    ........
    public:
    .........
    .........
   friend class TWO; // class TWO declared as friend of class ONE
};
```

• Now from class TWO, all the member of class ONE can be accessed.

Prepared By Mr.EBIN PM EDULINE 69



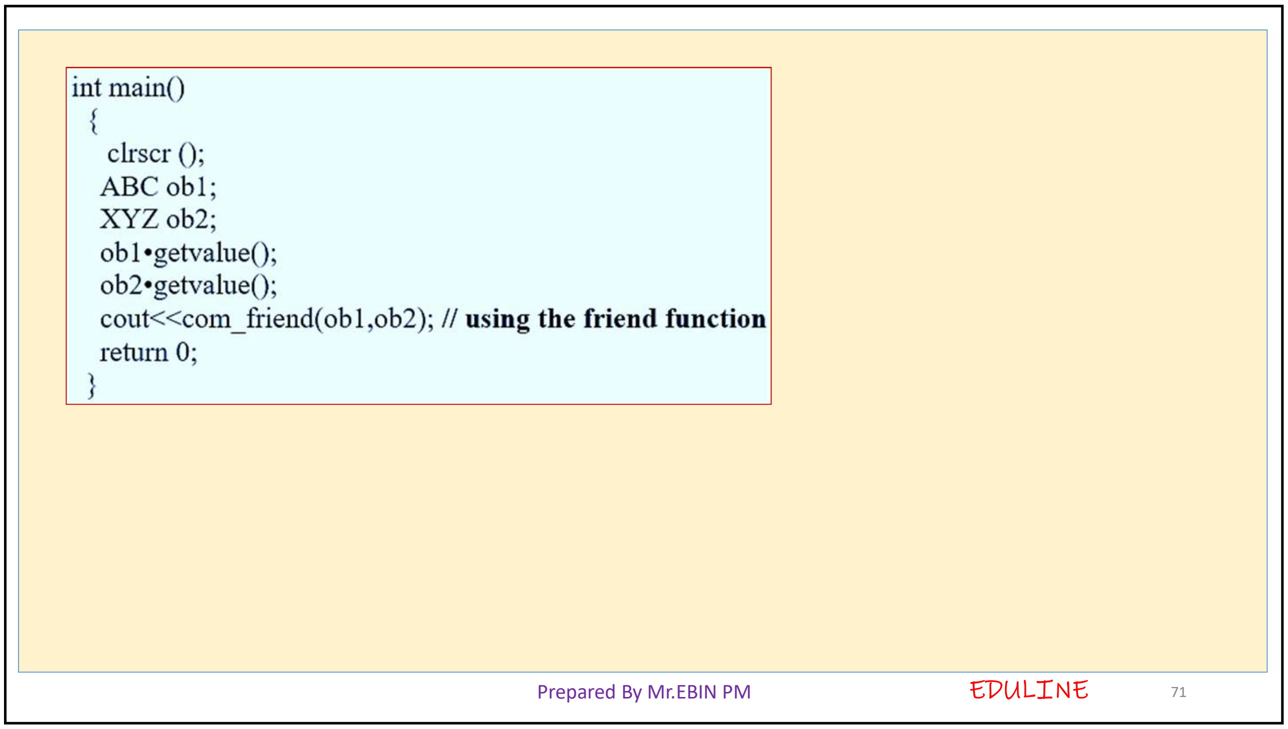
❖FRIEND AS BRIDGES

• We can use a function friendly to two classes.

```
#include<iostream.h>
#include<conio.h>
class XYZ;  // forward declaration to inform the compiler
class ABC
 { private:
    int x;
    int y;
   public:
     void getvalue()
       {
         x=3;
         y=7;
        }
      friend  int com_friend(ABC ,XYZ);
 }; // end of ABC definition
```

```
class XYZ   // actual definition of XYZ
{
   private
    int x;
    int y;
  public:
    void getvalue()
      {
        x=5;
        y=10;
      }
     friend  int com_friend(ABC ,XYZ);
};
int com_friend (ABC A,XYZ B)  //friend function's definition
   {
      return (A•x*B•x+A•y*B•y);
    }
```

Prepared By Mr.EBIN PM EDULINE 70



```
int main()
  {
    clrscr ();
   ABC ob1;
   XYZ ob2;
   ob1•getvalue();
   ob2•getvalue();
   cout<<com_friend(ob1,ob2); // using the friend function
   return 0;
  }
```

Prepared By Mr.EBIN PM EDULINE 71