Inline Functions

- To save execution time in short functions, you may elect to put the code in the function body directly inline with the code in the calling program.
- That is, each time there's a function call in the source file, the actual code from the function is inserted, instead of a jump to the function.
- Functions that are very short, say one or two statements, are candidates to be inlined.

Inline function example

```
#include <iostream>
using namespace std;
inline float lbstokg(float pounds) // lbstokg() converts pounds to kilograms
{
        return 0.453592 * pounds;
}
int main()
{
        float lbs;
        cout << "\nEnter your weight in pounds: ";
        cin >> lbs;
        cout << "Your weight in kilograms is " << lbstokg(lbs)<< endl;
        return 0;
}
```

Note: Inline keyword is actually just a *request to the compiler.* Sometimes the compiler will ignore the request and compile the function as a normal function.

Output?

```
#define MUL(a, b)  a*b
int main()
{
    printf("%d", MUL(2+3, 3+5));
    return 0;
}
```

// The macro is expended as 2 + 3 * 3 + 5, not as 5*8
// Output: 16`

type cast or Cast in c++

- Sometimes a programmer needs to convert a value from one type to another in a situation where the compiler will not do it automatically or without complaining.
- **Syntax:** aCharVar = static_cast<char>(anIntVar);

```
#include <iostream>
using namespace std;
int main()
{
int intVar = 1500000000;                //1,500,000,000
intVar = (intVar * 10) / 10;                    //result too large
cout << "intVar = " << intVar << endl;      //wrong answer
intVar = 1500000000;
intVar = (static_cast<double>(intVar) * 10) / 10;      //cast to double
cout << "intVar = " << intVar << endl;              //right answer
return 0;
}
```

intVar = 211509811
intVar = 1500000000

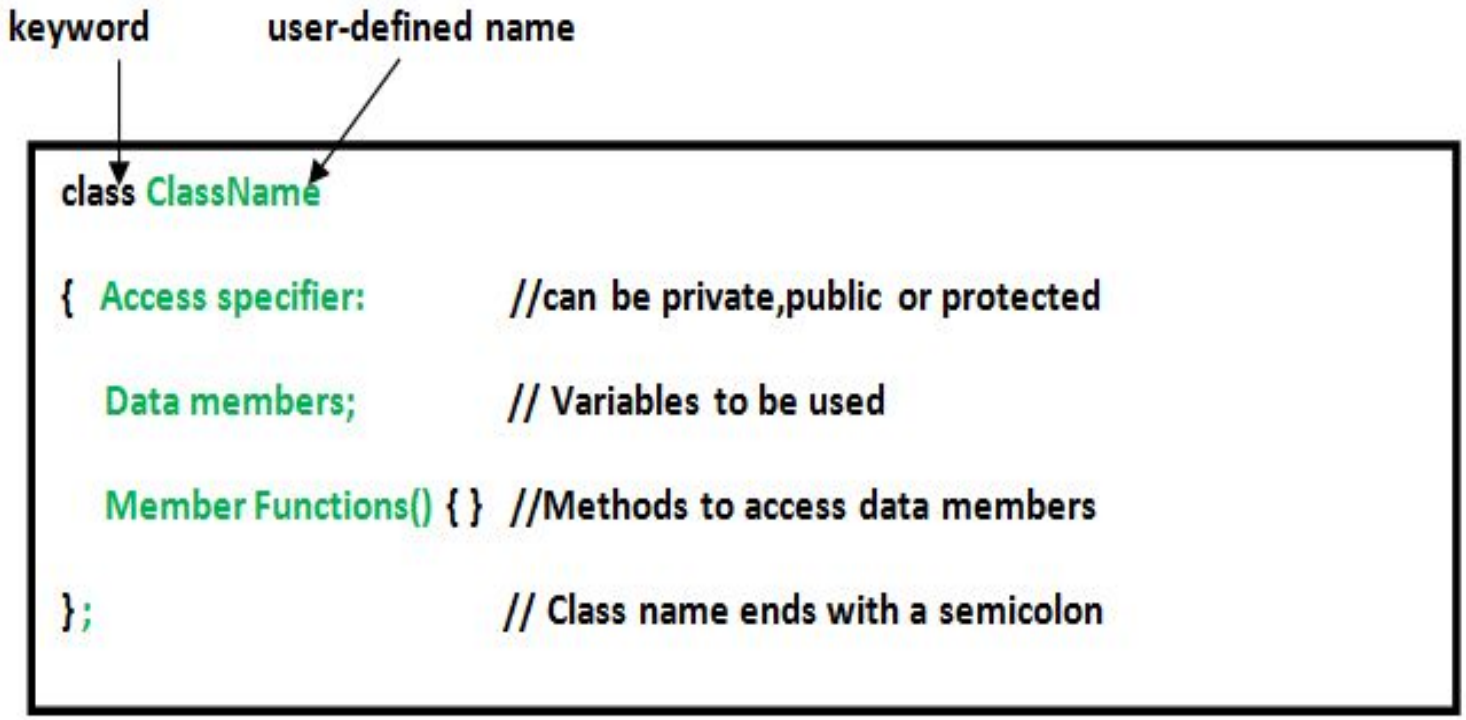
Classes and Objects C++

CLASS & OBJECT

- **Class:** It is the building block that makes C++ as Object Oriented programming.
- **It is a user defined data type, which holds its own data members and member functions, which can be accessed and used by creating an instance of that class.**
- Data members are the data variables and member functions are the functions used to manipulate these variables and together these data members and member functions defines the properties and behavior of the objects in a Class.
- In a class *Car*, the data member will be *speed limit*, *mileage* etc and member functions can be *apply brakes*, *increase speed* etc.

Note: The object occupies space in memory during runtime but the class does not.

Defining Class and Declaring Objects



- **Declaring Objects:** During class defintion, only the specification for the object is defined; no memory is allocated. To use the data and access functions defined in the class, we need to create objects.
- **Syntax: ClassName ObjectName;**

Example of Class

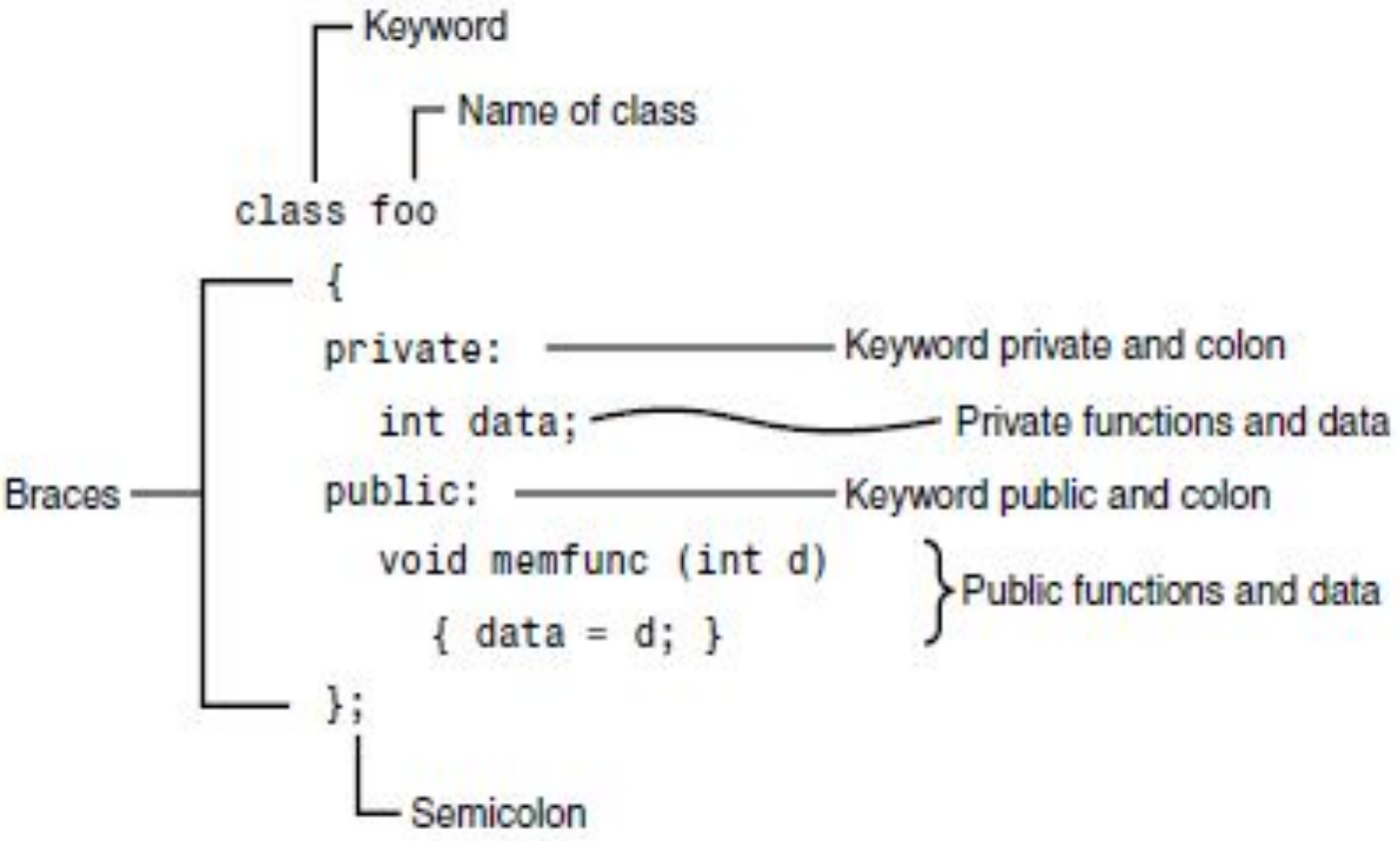


Fig:Syntax of a class definition.

Defining the Class (cont..)

Declaring the class **smallobj** which contains one data item and two member functions:

```
class smallobj                          //define a class
{
private:
        int somedata;                   //class data or data member
public:
        void setdata(int d) //member function to set data
        {
                somedata = d;
        }
        void showdata()                 //member function to display data
        {
                cout << "\nData is " << somedata;
        }
};
```

private and public (cont...)

- The data member ***somedata*** follows the keyword private, so it can be accessed from within the class, but not from outside.
- setdata() and showdata() follow the keyword public, they can be accessed from outside the class.
- **Generally Functions are public and Data is private:**
  - The data is hidden so it will be safe from **accidental manipulation**, while the functions that operate on the data are public so they can be accessed from outside the class.
  - However, there is no rule that says data must be private and functions public; in some circumstances you may find you'll need to use private functions and public data.

Look back to ***data hiding***

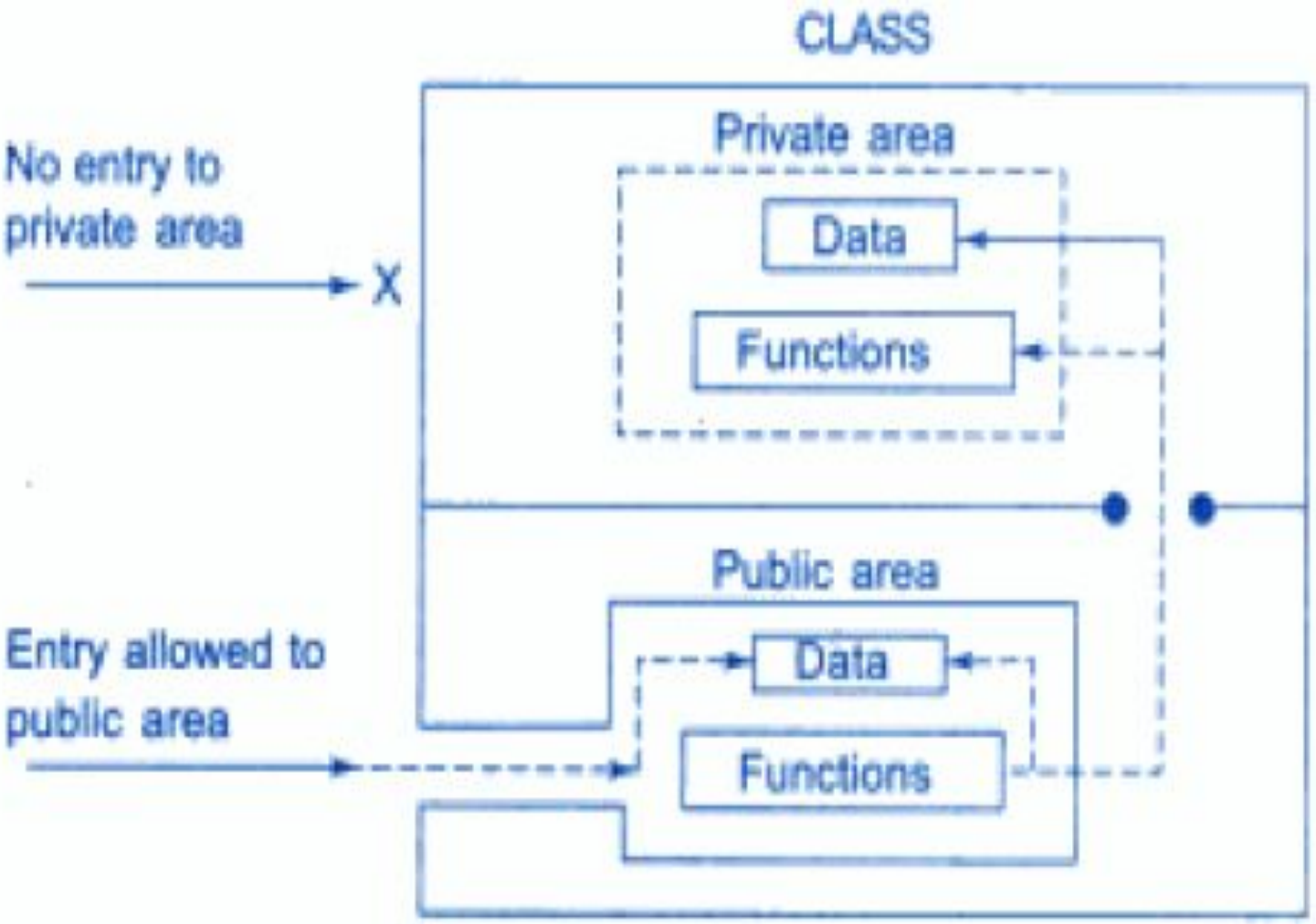
- *data hiding* means that data is concealed within a class so that it cannot be accessed mistakenly by functions outside the class.
- **The primary mechanism for hiding data is to put it in a class and make it private.**
- Private data or functions can only be accessed from within the class. Public data or functions, on the other hand, are accessible from outside the class**.**

Hidden from Whom?

- *Data hiding, on the other hand, means hiding data from parts of the program that don't need to access it. More specifically, one class's data is hidden from other classes.*

- Programmers who really want to manipulate, can figure out a way to access private data, but they will find it hard to do so by accident.

private and public access specifier



Accessing members of class

- The data members and member functions of class can be accessed using the dot('.') operator with the object. For example if the name of object is *obj* and you want to access the member function with the name *printName()* then you will have to write *obj.printName()* .
- **Accessing Data Members**

  The public data members are also accessed in the same way given however the private data members are not allowed to be accessed directly by the object.
  - Accessing a data member depends solely on the access control of that data member.
- This access control is given by Access modifiers in C++. There are three access modifiers : **public, private and protected**.

Accessing members of class

- General syntax:

  Object_name.member_function_name(arguments);

Examples:

Obj1.getData();

Obj2.showData();

Obj3.publicData=200;//public data member can be accessed by object

Following are invalid:

getData();//syntax error

Obj.private_Data;//syntax error

Using the Class

```
#include <iostream>
using namespace std;
class smallobj //define a class
{
private:
        int somedata; //class data
public:
        void setdata(int d) //member function to set data
        { somedata = d; }
        void showdata() //member function to display data
        { cout << “Data is “ << somedata << endl; }
};
**********
int main()
{
        smallobj s1, s2;                //define two objects of class smallobj
        s1.setdata(1066);     //call member function to set data
        s2.setdata(1776);
        s1.showdata();                  //call member function to display data
        s2.showdata();
        return 0;
}
```

Defining Member Functions

- Member functions can be defined in two ways:

1. **Inside the class definition (Already discussed)**
2. **Outside the class definition:**

General syntax

```
Return_type   class_name ::
  func_name(arguments)
{
//Function body
}
```

:: operator is scope resolution operator which tells to compiler that func_name function belong to class_name .

Important point

- Note that all the member functions defined inside the class definition are by default **inline**,
- We can also make any non-class function inline by using keyword **inline** with them.
- Inline functions are actual functions, which are copied everywhere during compilation, like pre-processor macro, so the overhead of function calling is reduced.

Nesting of member functions

- A member function of a class can be called only by an object of that class using a dot operator.
- However, there is an exception to this. A member function can be called by using its name inside another member function of the same class. This is known as **nesting of member functions**.
- Example:

```
void abc ::func()
{
Display();
}
```

Private member function

- Generally we keep all the data members in private section and function in public section, some time we may need to hide member function from outside calls. Like function Delete_Account, Update_Salary etc.
- A private member function can not be called by the object of the same class; **then how to call private function**?

Private member functions can be called by the other member function of its class.

- Sqare is a class whose data member is ‘side’ and member functions are ‘get_side()’, ‘calculate_area()’ and ‘display_area()’. display_area() is a private member function. Write a program to calculate the area of a square.

Memory allocation for Objects

- Member functions get memory only once when they are defined.
- Space for all non-staticdata members are allocated for each object separately.
- Example:

```
class area{
      int l,b,area;
      public:
      void input();
      void output(int,int);
}Obj;
```

sizeof(Obj) will be 12 Bytes

Memory allocation for Objects …

- Byte padding is followed while allocating memory to objects.
- Example:

```
class abc{
    char b;
     int a;
       public:
       void input();
       void output(int,int);
}Obj;
```

sizeof(Obj) will be 8 Bytes

0 1 2 3 4 5 6 7

b - - - a a a a

Memory allocation for Objects …

- Byte padding is followed while allocating memory to objects.
- Example:

```
class abc{
    char b;
    char c;
     int a;
       public:
       void input();
       void output(int,int);
}Obj;
```

sizeof(Obj) will be Bytes

0 1 2 3 4 5 6 7

b c - - a a a a

Memory allocation for Objects …

- Byte padding is followed while allocating memory to objects.
- Example:

```
class abc{
    char b;
    Char c;
     int a;
     Char f;
       public:
       void input();
       void output(int,int);
}Obj;
```

sizeof(Obj) will be Bytes

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

b c - - a a a a f - - -

Memory allocation for Objects …

```
class abc{
    char b;
    short c;
   char f;
     int a;
     public:
       void input();
       void output(int,int);
}Obj;
```

sizeof(Obj) will be Bytes

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

b - c c f - - - a a a a

Memory allocation for Objects …

```
class abc{
    char b;
    short c;
   char f;
     int a;
     public:
       void input();
       void output(int,int);
}Obj;
```

sizeof(Obj) will be Bytes

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

b - c c f - - - a a a a

Memory allocation for Objects …

```
class abc{
    char b;
    short c;
   char f;
     int a;
     public:
       void input();
       void output(int,int);
}Obj;
```

sizeof(Obj) will be Bytes

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

b - c c f - - - a a a a

Memory allocation for Objects …

- 1 word is 4 byte in 32 bit processor
- 1 word is 8 byte in 64 bit processor
- 1 byte can be stored at multiple of 1 memory slot.
- 2 byte can be stored at multiple of 2 memory slot.
- 4 byte can be stored at multiple of 4 memory slot.
- 8 byte can be stored at multiple of 8 memory slot.
- So, storing of a char will start from 0^{th} byte or 1^{st} or 2^{nd} or 3^{rd} or 4^{th} byte or so on in the main memory.
- Storing of an integer will start from 0^{th} byte or 4^{th} or 8^{th} byte or so on in the main memory.
- So, storing of a double will start from 0^{th} byte or 8^{th} or 16^{th} byte or so on in the main memory.

Memory allocation for Objects …

```
class abc{
    Int c;
    int a;
    public:
     void input();
     void output(int,int);
};
Class def {
abc p;
Char m;
}obj;
```

sizeof(Obj) will be Bytes

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

pp p p p p p p m - - -

Class and Structures

```
Struct complex
{ double re, im;};
Main()
{
Complex n={1.2,4.5};
}
```

```
class complex
{public:
double re, im;};
Main()
{
Complex n={1.2,4.5};
}
```

Class and Structures

```
Struct complex
{ double re, im;};
Void print(struct complex k)
{ cout<<k.re<<k.im;}
Main()
{
Complex n={1.2,4.5};
print(n);
}
```

```
class complex
{public:
double re, im;};
Void print(class complex k)
{ cout<<k.re<<k.im;}
Main()
{
Complex n={1.2,4.5};
print(k);
}
```

Class and structures

```
class complex
{public:
double re;
double im;
Void print() //not possible in structure
{ cout<<re<<im;
}
};
Main()
{
Complex n={1.2,4.5};
n.print();  //not possible in structure
}
```

Static Data Members

- Data member of class can be static.
- **Static data member has following properties:**

1. It is initialized to zero when the first object of its class is created.
2. Only one copy of static data member is created for the entire class and is shared by all the objects of that class.
3. It is visible only within the class but lifetime is the entire program.
4. Static variable are normally used to maintain values common to entire class

- For making a data member static, we require :
- (a) Declare it within the class.
- (b) Define it outside the class.

Static Data Members; 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 = 10;

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;

Static Member Function

- In C++, a static member function differs from the other member functions in the following ways:

(i) Only static members (functions or variables) of the same class can be accessed by a static member function.

(ii) It can be called using the **name of the class rather than an object as given below:**

Name_of_the_class :: function_name

For example,

student::showcount();

- What is default visibility mode for members of classes in C++ ?

A. Private
B. Public
C. Protected
D. Depends

Friend Function

- The concepts of encapsulation and data hiding dictate that nonmember functions should not be able to access an object's private or protected data.
- However, there are situations where such rigid discrimination leads to considerable inconvenience.
- So, there is mechanism built in C++ programming to access private or protected data from non-member function which is **friend function** and **friend class**.

Declaration of Friend Function

- The declaration of friend function should be made inside the body of class.

```
class class_name
    {
    ..... .... ........
    friend return_type function_name(argument/s); //Declaration
    ..... .... ........
    };
return_type function_name(argument/s)
{
//friend function defintion
}
```

Characteristics of friend function

1. The declaration of friend function should be made inside the body of class
2. Friend function declaration can be put anywhere inside class either in **private or public** section without affecting its meaning
3. No keywords is used in function definition of friend function, friend keyword is used only in declaration.
4. friend function can access the private and protected data of the class where it has been declared.
5. A friend function is not in the scope of the class, in which it has been declared as friend because friend function is not member function of any class.

Characteristics of friend function(Cont..)

6. It cannot be called using the object of that class, it can be invoked like a normal function without any object

7. Unlike member functions, it cannot use the member names directly.

8. Usually, it has objects as arguments. Although it is possible to write friend function without arguments.

Where friend function are used

1. Friend function is especially useful for operator overloading.
2. Friend function allow to operate on objects of two different classes.

- Example:

```
#include <iostream>
using namespace std;
class beta;       //needed for frifunc declaration
class alpha
{
private:
int data;
public:
ini_alpha() { data=3 }
friend int frifunc(alpha, beta); //friend function
};
```

Example (cont..)

```
class beta
{
private:
int data;
public:
ini_beta() { data=7 }
friend int frifunc(alpha, beta); //friend function
};
int frifunc(alpha a, beta b) //function definition
{
return( a.data + b.data );
}
```

```
int main()
{
alpha aa;
aa.ini_alpha();
beta bb;
bb.ini_beta();
cout << frifunc(aa, bb) << endl; //call the function
return 0;
}
```

Cont...

- A minor point: Remember that a class can't be referred to until it has been declared. Class beta is referred to in the declaration of the function frifunc() in class alpha, so beta must be declared before alpha. Hence the declaration

class beta; at the beginning of the program.

- Note the programmer who does not have access to the source code for the class cannot make a function into a friend. In this respect, the integrity of the class is still protected.

Friend Function without argument

```
#include <iostream>
class sample {
 private:  int x;
 public:  friend void fun();
};
 void fun()
{ sample s;
    s.x = 50000;
   cout << s.x;
}
int main
fun();
return 0; }
```

Friend class

```
class TWO; // forward declaration of the class TWO
class ONE
{
……….
……..
public:
……..
……..
friend class TWO; // class TWO declared as friend of class ONE
};
```

Friend class example

```
class Child {

friend class Mother; //Mother class members
  can access the                private parts of class
  Child.
public: void  Show_name( void );
            void setName( char * newName );

};
```