Friend functions

Access privileges in C++.

You have access privileges in C++ such as public, protected and private that helps in encapsulation of data at various level.

Private

If data are declared as private in a class then it is accessible by the member functions of the class where they are declared. The private member functions can be accessed only by the members of the class. By default, any member of the class is considered as private by the C++ compiler, if no specifier is declared for the member.

Public

The member functions with public access specifier can be accessed outside of the class. This kind of members is accessed by creating instance of the cass.

Protected

Protected members are accessible by the class itself and it's sub-classes. The members with protected specifier act exactly like private as long as they are referenced within the class or from the instance of the class. This specifier specially used when you need to use inheritance facility of C++. The protected members become private of a child class in case of private inheritance, public in case of public inheritance, and stay protected in case of protected inheritance.

Access specifier	Visible to own class members	Visible to objects of same/other class
`public`	Yes	Yes
`private`	Yes	No
`protected`	Yes	No

- When we want our private data to be shared by a non member function

  Then:

- Basically, we declare something as a friend, you give it access to your private data members.
- Single functions or entire classes may be declared as friends of a class.

- A Friend function is a non-member function of the class that has been granted access to all private members of the class.
- We simply declare the function within the class by a prefixing its declaration with keyword friend.
- Function definition must not use keyword friend.
- Definition of friend function is specified outside the class body and is not treated as a part of the class.
- The major difference b/w member function and friend function is that the member function is accessed through the object while friend function requires object to be passed as parameter.

Syntax:

```
class ABC
{
.............
public:
friend void xyz(object  of class);
};
```

Friend function characterstics

- It is not in scope of class.
- It cannot be called using object of that class.
- It can be invoked like a normal function.
- It should use a dot operator for accessing members.
- It can be public or private.
- It has objects as arguments.
- Perhaps the most common use of friend functions is **overloading << and >>** for I/O.

Example

```
class demo
{
int x;
public:
demo(int a)
{
x=a;
}
friend void display(demo);
};
```

```
void display(demo d1)
{
cout<<d1.x;
}
void main()
{
demo d2(5);
display(d2);
}
```

```
class sample
{
        int a;
        int b;
   public:
        void setval(){ a=25,b=40}
        friend float mean(sample s);
};
 float mean(sample s)
{       return (s.a+s.b)/2.0;
}

void main()
{       sample X;
        cout<<mean(X);
}
```

Friend class

- In previous section of class we declared only one function as a friend of another class.but it is possible that all member of the one class can be friend of another class.this is friend class

Friends (a few gory details)

- Friendship is not inherited, transitive, or reciprocal.
- Derived classes don't receive the privileges of friendship (more on this when we get to inheritance in a few classes).
- The privileges of friendship aren't transitive. If class A declares class B as a friend, and class B declares class C as a friend, class C doesn't necessarily have any special access rights to class A.
- If class A declares class B as a friend (so class B can see class A's private members), class A is not automatically a friend of class B (so class A cannot necessarily see the private data members of class B).
- Friendship is not inherited, transitive, or reciprocal.

Example

```
class demo
{
private:
int x,y;
public:
demo(int a,int b)
{
x=a;
y=b;
}
friend class frnd;
};
```

```
class frnd
{
public:
void display(demo d1)
{
cout<<“x is=”d1.x;
cout<<“y is=”d1.y;
}
};
```

```
void main()
{
demo d2(10,40);
frnd f1;
f1.display(d2);
getch();
}
```

This pointer

Every object in C++ has access to its own address through an important pointer called **this** pointer. The **this** pointer is an implicit parameter to all member functions. Therefore, inside a member function, this may be used to refer to the invoking object.

Friend functions do not have a **this** pointer, because friends are not members of a class. Only member functions have a **this** pointer.