C++ : Friend and Inheritance

Friend functions

Private and protected members of a class cannot be accessed from outside the same class in which they are declared. However, this rule does not affect *friends*.

- A function that is not a member of a class but has access to the class's private and protected members.
- They are normal external functions that are given special access privileges.
- **Friend** function is declared by the class that is granting access. The **friend** declaration can be placed anywhere in the class declaration. It is not affected by the access control keywords (public, private and protected.)

```
#include <iostream>
using namespace std;

class myclass {
        int num;
public:
         myclass(int x) {
                num = x;
        }
        friend int isneg(myclass ob);
};

int isneg(myclass ob)       //friend function
{
        return (ob.num < 0) ? 1 : 0;
}

int main()
{
        myclass a(-1), b(2);

        cout << isneg(a) << ' ' << isneg(b);
        cout << endl;

         return 0;
}
```

Friend classes

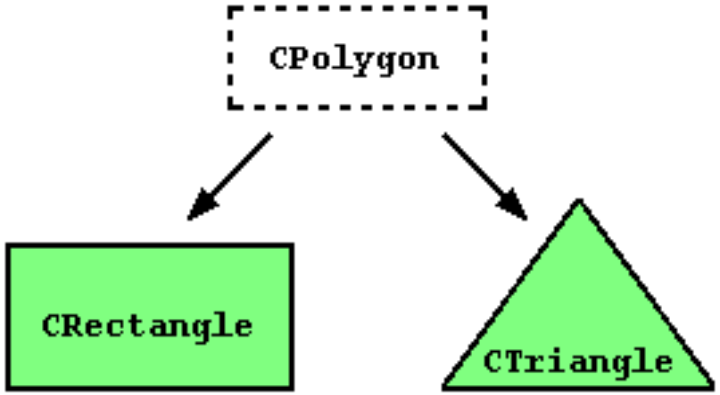
We can define a class as friend of another one, granting that second class access to the protected and private members of the first one.

```
#include <iostream>
using namespace std;
class CSquare {
  private:
       int side;
  public:
     void set_side (int a)
        {side=a;}
  friend class CRectangle;
};
class CRectangle {
    int width, height;
  public:
    int area ()
      {return (width * height);}
    void convert (CSquare a);}
```

```
void CRectangle::convert (CSquare a) {
  width = a.side;
  height = a.side;}

int main () {
  CSquare sqr;
  CRectangle rect;
  sqr.set_side(4);
  rect.convert(sqr);
  cout << rect.area();
  return 0;}
```

Inheritance between classes

- allows to create classes which are derived from other classes, so that they automatically include some of its "parent's" members, plus its own.



The class CPolygon would contain members that are common for both types of polygon. In our case: width and height.

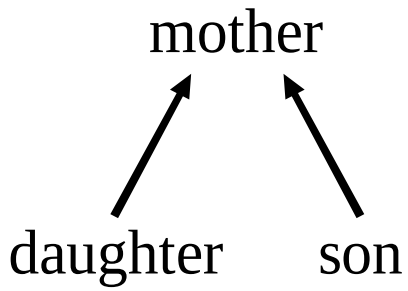
```
class A : public B
{       // Class A now inherits the members of Class B
        // with no change in the “access specifier” for
}       // the inherited members

class A : protected B
{       // Class A now inherits the members of Class B
        // with public members “promoted” to protected
}       // protected members “promoted” to private

class A : private B
{       // Class A now inherits the members of Class B
        // with public and protected members
}       // “promoted” to private
```

Type of Inheritance

Visibility modifier of members

	private	protected	public
private	private	private	private
protected	private	private	protected
public	private	protected	public

- The type of inheritance defines the minimum access level for the members of derived class that are inherited from the base class
- With public inheritance, the derived class follow the same access permission as in the base class
- With protected inheritance, only the public members inherited from the base class can be accessed in the derived class as protected members
- With private inheritance, none of the members of base class is accessible by the derived class




```
class son : protected mother{
    private:
        double b;
    public:
        void foo ( );
}
```

```
class mother{
    protected:
        int x, y;
    public:
        void set(int a, int b);
    private:
        int z;
}
```

```
void son :: foo ( ){
    x = y = 20;  // error, not a public member
    set(5, 10);
    cout<<"value of b "<<b<<endl;
    z = 100;    // error, not a public member
}
```

son can access only 1 of the 4 inherited member

What is inherited from the base class?

In principle, a derived class inherits every member of a base class except:

- its constructor and its destructor
- its friends

Although the constructors and destructors of the base class are not inherited themselves, its default constructor (i.e., its constructor with no parameters) and its destructor are always called when a new object of a derived class is created or destroyed.

If you want that an overloaded constructor is called when a new derived object is created, you can specify it in each constructor definition of the derived class:

derived_constructor_name (parameters) : base_constructor_name (parameters) {...}

```
class mother {
  public:
    mother ()
      { cout << "mother: no parameters\n"; }
    mother (int a)
      { cout << "mother: int parameter\n"; }
};
class daughter : public mother {
  public:
    daughter (int a)
      { cout << "daughter: int parameter\n\n"; }};
```

```
class son : public mother {
  public:
    son (int a) : mother (a)
      { cout << "son: int parameter\n\n"; }
};

int main () {
  daughter cynthia (0);
  son daniel(0);
  return 0;
}
```

Multiple inheritance (C++ only)

In C++ it is perfectly possible that a class inherits members from more than one class. Deriving a class from more than one direct base class is called *multiple inheritance*.

This is done by simply separating the different base classes with commas in the derived class declaration.

```
class A { /* ... */ };
class B { /* ... */ };
class C { /* ... */ };
class X : public A, public B, public C { /* ... */ };
```

```
class CPolygon {
  protected:
    int width, height;
  public:
    void set_values (int a, int b)
      { width=a; height=b;}
  };

class COutput {
  public:
    void output (int i);
};
void COutput::output (int i) {
  cout << i << endl;
}
```

```
class CRectangle: public CPolygon, public COutput {
  public:
    int area ()   { return (width * height); }
  };
class CTriangle: public CPolygon, public COutput {
  public:
    int area ()   { return (width * height / 2); }
  };
int main () {
  CRectangle rect;
  CTriangle trgl;
  rect.set_values (4,5);
  trgl.set_values (4,5);
  rect.output (rect.area());
  trgl.output (trgl.area());}
```

The main problem appears when data fields or methods with same names are present in both super classes.

```
class Truc {
public:
   int chose() {}
};

class Machin {
public:
  int chose() {}
};

class Bidule : public Truc, Machin {
};

int main() {
   Bidule x;
   x.chose();
}
```

This can not compile:
chose.cc: In function ‘int main()’:
chose.cc:16: error: request for member ‘chose’ is ambiguous
chose.cc:8: error: candidates are: int Machin::chose()
chose.cc:3: error: int Truc::chose()

```
Truc:: x.chose();     or
Machin:: x.chosen();
```