Chapter: 11

Overloading: function & Operators. Inheritance.

General structures of : if, else-if, do, while, for & switch statements

11.1 Introduction to INHERITANCE

In C++, ***inheritance*** is the mechanism by which one class can *inherit the properties of another*. Inheritance allows a hierarchy of classes to be built, moving from the most general to the most specific.

❒ ***Base class and derived class:*** When one class is inherited by another, the class that is ***inherited*** is called the ***base class***. The ***inheriting*** class is called the ***derived class***. In general, the process of inheritance begins with the definition of a base class. *The base class defines all qualities that will be common to any derived classes*.

☞ The ***base class*** represents the *most general description* of a set of traits. A ***derived class*** inherits those general traits *and adds properties that are specific to that class*.

Example of ***base*** class and ***derived*** class	
The declaration for the ***base class***	Using base class, here is a ***derived class*** that inherits it:

`/* Define base class. */`
`class Bs {  int i;`
`  public : void set_i(int n);`
`  int get_i();  };`
	`/* Define derived class . */`
	`class Drv : public Bs { int j;`
	`  public : void set_k(int n);`
	`  int mul();  };`

- ❖ Notice that after the class name ***Drv*** there is a ***colon*** followed by the keyword ***public*** and the class name ***Bs***. This tells the compiler that class ***Drv*** will inherit all components of class ***Bs***.
- ❖ They keyword ***public*** tells the compiler that ***Bs*** will be inherited such that all ***public*** elements of the base class will also be ***public*** elements of the ***derived*** class. However, all ***private*** elements of the ***base*** class remain ***private*** to it and are not directly accessible by the ***derived*** class.
- ❖ Derived class can call base class public member functions directly. For example consider the following function.

```
int :: mul(){return j*get_i(); }
```

Notice that it calls ***`get_i()`***, which is a member of the ***base*** class ***Bs***, not of ***Drv***, without ***linking*** it to any specific ***object*** (without base class it won't be possible).
However, the reason that ***`mul()`*** must call ***`get_i()`*** instead of accessing ***`i`*** directly is that the ***private*** members of a ***base*** class (in this case,***`i`***) remain ***private*** (due to ***encapsulation***) to it and not accessible by any ***derived*** class.

❒ The ***general form*** used to inherit a base class is shown here:

```
class derived_class_name : access_specifier base_class_name {
                                                . . . . . . . };
```

Here ***access-specifier*** is one of these three keywords: ***public***, ***private***, or ***protected***.

11.2 Intro to FRIEND functions

There will be times when you want a function to have *access to the private members of a class without that function actually being a member of that class*. Towards this end, C++ supports ***friend functions***.

☞ A ***friend*** is not a member of a ***class*** but still has access to its ***private*** elements.

☞ Friend functions are useful for

[1] Operator overloading
[2] Creation of certain types of I/O functions
[3] Need one function to have access to the private members of two or more different classes.

❒ ***Definition:*** A friend function is defined as a regular, ***nonmember function***. However, inside the ***class*** declaration for which it will be a friend, *its prototype is also included*, prefaced by the keyword ***friend***. To understand how this works, examine this short program:

```
class myclass { int n, d;
  public :
  myclass(int i, int j) {n = i; d = j; }

  /* declare a friend of myclass */
  friend int isfactor( myclass ob);  };

/* Here is friend function definition. It returns true if d is a factor
of n. Notice that the keyword friend is not used in the definition of
isfactor()*/
int isfactor( myclass ob) {
        if(!( ob.n % ob.d)) return 1;
        else return 0;}
```

In this example, ***`myclass`*** declares its ***constructor*** function and the ***friend `isfactor()`*** inside its ***class declaration***. Because ***`isfactor()`*** is a friend of ***`myclass`***, ***`isfactor()`*** has access to its ***private members***. This is why, within ***`isfactor()`***, it is possible to directly refer to ***`ob.n`*** and ***`ob.d`***.

❒ It is important to understand that *a friend function is not a member of the class* for which it is a ***friend***. Thus, it is not possible to call a friend function by using an object name and a class member access operator (a ***`dot "."`*** or an ***`arrow "->"`***). Instead, friends are called just like *regular functions*. For example, given the preceding example, this statement is wrong:

```
ob1.isfactor();     /* wrong ; isfactor() is not a member function */
```

☞ Although a friend function ***has knowledge of the private elements of the class*** for which it is a friend, it can only ***access them through an object of the class***. That is, unlike a member function of ***myclass***, which can refer to ***n*** or ***d*** directly, a ***friend can access these variables only in conjunction with an object*** that is declared within or passed to the friend function.

❒ Inside the friend function, it is ***meaningless*** to ***refer to a private member without reference to a specific object***. A friend function is not linked to any object. It simply is granted access to the private elements of a class.

❒ Because friends are not members of a class, they will typically be ***passed one or more objects*** of the class for which they are friends. This is the case with ***isfactor()***. It is passed an object of ***myclass***, called ***ob***. However, because ***isfactor()*** is a friend of ***myclass***, it can access ***ob***'s private elements. Without being friend it would not be able to access ***ob.d*** or ***ob.n*** since ***n*** and ***d*** are ***private*** members of ***myclass***.

❒ ***Remember:*** A friend function is ***not inherited***. That is, when a base class includes a friend function, that ***friend function is not a friend of a derived class***.

❒ One other important point about friend functions is that ***a friend function can be friends with more than one class***.

❒ A function can be a ***member of one class*** and a ***friend of another***.

❒ ***Forward declaration:*** Sometimes, there needs to be some way to tell the compiler about ***a class name without actually declaring*** it. This is called a ***forward declaration***. In C++, to tell the compiler that an identifier is the name of a class, use a line like this : ***class class_name;*** before the class name is first used. For example, in the following program, the ***forward declaration*** is:

class truck ;

☞ One common (and good) use of a friend function occurs when ***two different types of classes have some quantity in common that needs to be compared***.
For example, consider the following program, which creates a ***class called car and a class called truck***, each containing, as a private variable, the ***speed*** of the vehicle it represents:

```
      class truck ; // a forward declaration
class car { int passengers ; int speed ;
      public :
      car (int p, int s) {
            passengers = p; speed = s; }
      friend int sp_greater (car c, truck t);
      };

class truck { int weight ; int speed ;
      public :
      truck (int w, int s) {
                  weight = w; speed = s; }
      friend int sp_greater (car c, truck t);
      };

int sp_greater(car c, truck t){
            return (c.speed - t.speed);  }
```

```
int main(){ int t;
     car c1(6, 55) , c2(2, 120) ;
     truck t1(10000 ,55) ,t2(20000 ,72);

cout << " Comparing c1 and t1 :\n";
t = sp_greater(c1 , t1);
if(t <0) cout << " Truck is faster .\n";
else if(t==0) cout <<"Speed is the same .\n";
else cout << "Car is faster .\n";

cout << " Comparing c2 and t2 :\n";
t = sp_greater(c2 , t2);
if(t <0) cout << " Truck is faster .\n";
else if(t==0) cout <<"Speed is the same .\n";
else cout << "Car is faster .\n";
return 0;}
```

This program contains the function ***sp_greater()***, which is a friend function of both the ***car and truck classes***. This function returns positive if the ***car object*** is going faster than the ***truck object***, ***0*** if their ***speeds*** are the ***same***, and negative if the truck is going faster.

In this case we need ***forward declaration*** (also called ***forward reference***). Because ***sp_greater()*** takes parameters of both the car and the truck classes, it is ***logically impossible to declare both before including sp_greater()*** in either. Now truck can be used in the friend declaration of ***sp_greater()*** without generating a ***compile-time error***.

❒ ***Use of scope resolution operator with friend:*** We have to use the ***scope resolution*** operator to declare a ***friend function to a class*** which is actually ***a member-function of another class***. For example, here is the preceding example rewritten so that ***sp_greater*** is a ***member of car*** and a ***friend of truck***:

```
   class truck ; // a forward declaration
class car { int passengers ; int speed ;
         public :
         car (int p, int s) {
         passengers = p; speed = s; }
         int sp_greater (truck t); };
```

```
class truck { int weight ; int speed ;
      public :
      truck (int w, int s) {
      weight = w; speed = s; }
/* note new use of the scope resolution operator */
friend int car :: sp_greater( truck t); };
```

```
      int car :: sp_greater(truck t){ return (speed - t.speed);  }
/*Since sp_greater() is member of car , only a truck object must be passed to it. */
```

☞ Notice the ***new use of the scope resolution operator*** as it occurs in the friend declaration within the truck class declaration. In this case, it is ***used to tell the compiler*** that the function ***sp_greater()*** is a member of the car class.

☞ However a slight change appear inside ***main()*** which need to compute ***t*** (because ***sp_greater*** is a member of car)

```
cout << "Comparing c1 and t1 :\n";      t = c1.sp_greater(t1); /* evoke as member function of car */
                                          and
cout << "Comparing c2 and t2 :\n";      t = c2.sp_greater(t2); /* evoke as member function of car */
```

☞ When referring to a member of a class, it is never wrong to fully specify its name. However, it is redundant, and seldom used. Eg:

t = c1.sp_greater(t1);

Can be written using the ***scope resolution operator*** and the class name car like this: ***t = c1.car :: sp_greater(t1);***