Advanced Programming

Inheritance (1)

Topics

- Base and Derived Classes
  - Single Inheritance
  - Declaration of derived classes
    - Order of Constructor and Destructor Execution
  - Inherited member accessibility
- Examples

Inheritance

- Inheritance is a method by which one class acquires the properties (data and operations) of another class
- <u>Base Class (or superclass</u>): the class being inherited from
- <u>Derived Class (or subclass</u>): the class that inherits

Why Inheritance?

- When a class is inherited from another class, we can:
  - Reuse the methods and data of the existing class
  - Extend the existing class by adding new data and new methods
  - Modify the existing class by overloading its methods with your own implementations

Example

- Assume a class named ***List*** is defined to store an integer list. A new class named ***Set*** is needed to store a set of integer values. Inherit class Set from class List

Class List

```
class  List
{
    int *data;
    public:
    List(int size=100);
    void Insert(int val);
    void Delete(int val);
    bool Contains(int val);
    ~List()
};
```

Class Set

```
class Set : public List
{
     int card;
     public:
         void Insert(int val);
         void Remove(int val);
         int NumMembers(){ return card;}
};
```

Inheritance and Accessibility

- A class inherits the ***behavior*** of another class and enhances it in some way

- Inheritance ***does not*** mean the inheriting class can have access to the private members of the base class

Protected Class Members

- Derived classes **cannot** access the private data of the base class
- Declaring methods and data of the base class as *protected* (instead of private) allows derived classes to access them
- Objects outside the class, however, cannot access them (same as private)

Constructors and Destructors

- We <u>cannot override a base class constructor</u> with a derived class constructor (rather, the derived class constructor calls the base class constructor first)
- If the base class constructor takes parameters, they should be passed to it.
- The destructor of the derived class is called before the destructor of the bas class

Example: Derive class 3D Point from 2D Point

```
class Point {
public:
  Point();
  Point( int xv, int yv );
  void SetX( int xv );
  void SetY( int yv );
private:
  int x;
  int y;
};
```

Example: Derive class 3D Point from 2D Point

```
class Point3D :public Point {
public:
  Point3D();
  Point3D( int xv, int yv, int zv );
  void SetZ( int zv );
private:
  int z;
};
```

```
Point3D::Point3D( int xv, int yv, int zv )
{
  SetX( xv );
  SetY( yv );
  SetZ( zv );
}
```

```
int main()
{
Point3D P;
P.SetX( 100 );
P.SetY( 200 );
P.SetZ( 300 );
return 0;
}
```

Overriding

- A function in the derived class with the same function name will override the function’s variables in the base class.
- You can still retrieve the overridden functions variables by using the scope resolution operator ”::”.

Overriding

```
#include <iostream.h>
#include <stdlib.h>
class A
{   int i;
public:
    A(){i = 5;};
    int get(){return i;};
};
class B: public A
{   int i;
public:
    B(){i = 10;};
    int get(){return i;};
};
```

```
void main()
{  B b;
   int x;
   cout << b.get()<<endl;
   cout << b.A::get()<<endl;
   cout << sizeof(x)<<endl;
   cout << sizeof(b)<<endl;
}
```

Types of Inheritance

- public
- private
- protected

Public Inheritance

- Public and protected members of the base class become respectively public and protected members of the derived class.

Private Inheritance

- Public and protected members of the base class become private members of the derived class.

Protected Inheritance

- Public and protected members of the base class become protected members of the derived class.

Why use the constructor-initializer?

- Without it, the default constructor for the base class would be called, which would then have to be followed by calls to access functions to set specific data members.

Constructors in Derived Classes

- When an object of a derived class is created, the constructor of the object must first explicitly call the constructor of the base class.
- This is the same as constructor- initializer.

```
class Base
{ int n;
  public: Base(int x);
};
class Derived : public Base
{
   int t;
   public: Derived(int y) : Base(t) {t = y;}
};
```

Destructor Function

- Destructors are called implicitly starting with the last derived class and moving in the direction of the base class.

Compatibility Between Base and Derived Classes

- An object of a derived class can be treated as an object of its base class.
- The reverse is not true.

Nested Class Scope

- A public or protected base class member that is hidden from the derived class can be accessed using the scope resolution operator ” ::”
- For example: ***base-class::member***
- The “that” of base class can not access the members of its derived classes.