Inheritance

Lecture 17

Casting Base-Class Pointers to Derived-Class Pointers

```
class base
{ protected: int x,y;
 public:
 base(int i=0,int j=0) { x=i; y=j;}
 void display_base()
 { cout<<"\n X:"<<x<<" Y:"<<y;}};
```

```
class derived : public base
{ private: int z;
  public:  derived(int k=0) { z=k; }
  void display_der()
  { cout<<"\n X:"<<x<<" Y:"<<y<<"
     Z:"<<z; }};
```

```
void main()
{
 base b(2,5); derived d(3);
 base *basepointer=&b;
 derived *derivedpointer=&d;
 basepointer->display_base();
 derivedpointer->display_der();
 derivedpointer=(derived
     *)basepointer;
 derivedpointer->display_der();
}
```

Using member functions

- Derived class
  - Cannot directly access private members of its base class
  - It can be modified using the base class member functions

Example

```
class base
{ int x;
  public:  int y;
  base(int i=0,int j=0) : x(i),y(j) { }
  void display() { cout<<"\n X : "<<x<<" Y : "<<y; }};
class derived : public base
{ int z;
  public:
  derived(int s) : z(s) { }
  void d() { display(); cout<<"\n Z : "<<z; } };
void main()
{ derived d(5);  d.d(); d.display(); }
```

Overriding base-class members in derived class

- To override a base-class member function
  - In derived class, supply new version of that function
    - Same function name, different definitions
  - The scope resolution operator can then be used to access the base class version from the derived class

Example

```
class base
{ protected: int x,y;
 public:
 base(int i=0,int j=0) { x=i; y=j;}
 void display()
 { cout<<"\n X:"<<x<<" Y:"<<y;}};
```

```
class derived : public base
{private: int z;
 public:
 derived(int k=0) { z=k; }
 void display()
 { cout<<"\n X:"<<x<<" Y:"<<y<<"
    Z:"<<z; }};
```

```
void main()
{
 base b(2,5); derived d(3);
 b.display();
 d.display();
}
```

Using constructor in derived class

- Base class initializer
  - Uses member-initializer syntax
  - Can be provided in the derived class constructor to call the base-class constructor explicitly
    - Otherwise base class' default constructor called implicitly
  - Base-class constructors are not inherited by derived classes
    - However, derived-class constructors and assignment operators can call still them

Using constructor in derived class

- Derived-class constructor
  - Calls the constructor for its base class first to initialize its base-class members
  - If the derived-class constructor is omitted, its default constructor calls the base-class' default constructor

Using destructor in derived class

- Destructors are called in the reverse order of constructor calls.
  - Derived-class destructor is called before its base-class destructor

Example 1

```
class base
{ protected: int x,y;
 public: base(int i=0,int j=0)
{ x=i; y=j;
cout<<"\n Base class constructor!!";}
 void display()
 { cout<<"\n X:"<<x<<" Y:"<<y;}
 ~base()
{ cout<<"\n base class destructor !!" ;}
};
```

```
class derived : public base
{  private: int z;
  public:  derived(int k=0)
{ z=k; cout<<"\n Derived class class
     constructor!!"; }
  void display()
  { cout<<"\n X:"<<x<<" Y:"<<y<<" Z:"<<z;
     }
  ~derived() { cout<<"\n derived class
     destructor !!" ;}
};
```

```
void main()
{
 derived d(3);
 d.display();
 }
```

Example 2

```
class base
{ protected: int x,y;
 public: base(int i=0,int j=0)
{ x=i; y=j;
cout<<"\n Base class constructor!!";}
 void display()
 { cout<<"\n X:"<<x<<" Y:"<<y;}
 ~base() { cout<<"\n base class destructor !!" ;}
};
```

```
class derived : public base
{  private: int z;
  public:  derived(int i,int j,int k=0) : base(I,j)
{ z=k; cout<<"\n Derived class class constructor!!"; }
  void display()
  { cout<<"\n X:"<<x<<" Y:"<<y<<" Z:"<<z; }
  ~derived() { cout<<"\n derived class destructor !!" ;}
};
```

```
void main()
{
 derived d(1,1,3);
 d.display();
 }
```

Asssignment

- Execution Of constructor and Destructor in various Types of Inheritance.