Chapter 7

Objective

- Inheritance
- Behavior Of Private Members
- Behavior Of Protected Members
- Identical Members For Base And Derived Classes
- Single Inheritance
- Multiple Inheritance
- Constructor and Destructor Behavior with Multiple Inheritance
- Assignment Operator Behavior during Inheritance
- Virtual Class

Unit 1

Inheritance

- Inheritance saves the programmer from having "**to reinvent the wheel**"
- The primary purpose is to reuse existing source code.
- All of the properties of the base class can be inherited into the derived class.

Derived class declaration:

```
    class derived_name : access_type  base_name {
         public:
                 // members.
         private:
                 // members.
};
```

- Derived class is always at least as big as base class.
- Derived class can override a base class function by having same name and same number of arguments.
- Derived class can also have any new data members or member functions.
- Derived class itself can become a base class for future derived class.
- Every object of a derived class is also an object of that derived class's base class.
- Derived class -object-**is-a**- base class -object.

access_type

- access_type determines how members of the base class are inherited by derived class.
- access_type specifies the derivation type, either private (**default**) or public.

Unit 1

- If access_type is private then all members of the base class are inherited as private members (regardless of their access_type).

```
#include <iostream>    // Example 7-1
using namespace std;

class CVehicle {    // base class
    public:
        void SetVehicleId(int nId) { m_nVehicleId = nId; }
        void ShowVehicleId() { cout << m_nVehicleId << '\n'; }

    private:
         int m_nVehicleId;
 };

```

Unit 1

```
// Inherit base as private
class CCar : private CVehicle {    // derived class
    public:
        void SetCarId(int nId) { m_nCarId = nId; }
        void ShowCarId() { cout << m_nCarId << "\n"; }
    private:
        int m_nCarId;
};

int main()
{
    CCar Jaguar;
    //cannot access public member declared in class 'CVehicle
    Jaguar.SetVehicleId(120);
    Jaguar.SetCarId(150);

    //cannot access public member declared in class 'CVehicle
    Jaguar.ShowVehicleId();
    Jaguar.ShowCarId();
    return 0;
}
```

Unit 2

Behavior Of Private Members

- The new member functions or instances of derived class cannot access private members of the base class even if access_type is public.

```
#include <iostream>    // Example 7-2

using namespace std;

// CVehicle::id_num is not accessible in function CCar::show()
class CVehicle {    // base class
    public:
        CVehicle() { m_nId = 4; }
    private:    //protected: will fix this error
        int m_nId;
 };

```

Unit 2

```
//class CCar : public CVehicle {    // derived class
class CCar : CVehicle {    // derived class
    public:
        // Cannot access private member of base class.
        // m_nId is a private member of the CVehicle class
        //and it is not accessible from here.
        void Show () { cout << m_nId << "\n"; }
};

int main()
{
    CCar toyota;
    toyota.Show();

    return 0;
}

// OUTPUT:
// Compiler Error!!!
```

Unit 3

Behavior Of Protected Members

- To access the private members of the base class when access type is public new access category **“protected”** is used.
- By changing private to protected in the base class all member of base are accessed directly by the derived class.

```
#include <iostream>    // Example 7-3

using namespace std;

// CVehicle::id_num is accessible in function car::show()
class CVehicle {    // base class
    public:
        CVehicle() { m_nId = 4; }
        Int Add() { m_nId++; }
    protected:
         int m_nId;
 };
```

Unit 3

```
class CCar : CVehicle {    // private derivation access type
    public:
    // m_nId is a protected member of the base class
    // therefore it is accessible from here.
    void Show () { cout << m_nId << "\n"; }
};

int main()
{
    CCar toyota;
    toyota.Show();
    cout << toyota.Add();  // Error derivation access type is private
    return 0;
}
```

Unit 3

- Protected and private are same i.e. direct access by the instances of classes in non-member function is denied and must use member functions.
- main() function can not access private or protected members only public members are accessible.

```
#include <iostream>    // Example 7-4

using namespace std;

// CVehicle::id_num is accessible in function car::show()
class CVehicle {    // base class
    public:
        CVehicle() { m_nId = 4; }
    protected:
        int m_nId;
 };

```

Unit 3

```
class CCar : public CVehicle {    // derived class
    public:
    // m_nId is a protected member of the base class
    //therefore it is accessible from here.
    void Show () { cout << m_nId << "\n"; }
};

// Show that protected members can not be accessed form main.
int main()
{
    CVehicle Mazada;

    //Can not access private or protected directly from here
    cout << Mazada.m_nId;

    return 0;
}

// OUTPUT:
// Compile error!!
```

Unit 4

Identical Members For Base And Derived Classes

- Derive class may override existing member from base class.
- The members from the derived classes are used.
- The base class member can only be accessed by using scope resolution operator.

```
#include <iostream>    // Example 7-5

using namespace std;

class CVehicle {    // base class
    public:
        enum eColor {BLACK,WHITE,RED,GREEN,BLUE};
        CVehicle() { m_Id = 54; }
    protected:
        int m_Id;
         eColor m_Color;

 };

```

Unit 4

```
  class CCar : public CVehicle {    // derived class
      public:
          CCar() { m_Id = 65; }
          Show();
      private:
          int m_Id;
  };

  void CCar::Show()
   {
// Explicitly access Base member by using scope resolution operator (::)
    cout << "Base Class m_Id:" << CVehicle::m_Id << "\n";
    cout << "Driverd Class m_Id:" << m_Id << "\n";
  }

  int main()
  {
      CCar toyota;
      toyota.Show();
      return 0;
  }
```

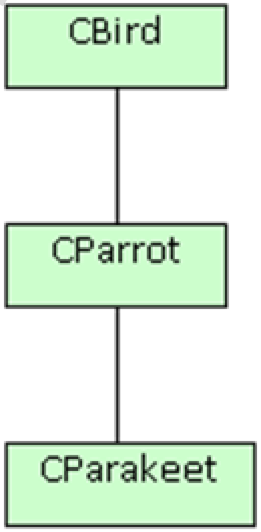
OUTPUT:

```
20    Base Class m_Id:54
21    Driverd Class m_Id:65
```

Unit 5

Single Inheritance

```
#include <iostream>    // Example 7-6

using namespace std;

class CBird {
    public:
        CBird (int x) {a = x; }
        int GetBird() { return a; }
    private:
        int a;
 };

```



Unit 5

```
class CParrot : public CBird {
    public:
        CParrot (int x, int y) : CBird (y) { b = x; }
        int GetParrot() { return b; }
    private:
        int b;
};

// Inherit a CParrot class and an indirect CBird class.
class CParakeet : public CParrot {
    public:
        CParakeet (int x, int y, int z) : CParrot(y, z) { c = x; }
        // Because CBird is inherited by CParrot as public,
       // CParakeet has access to public members of both classes
        // CBird and CParrot
        void Show();

    private:
        int c;
};
```

Unit 5

```
void CParakeet::Show()
{
   cout << GetBird() << ' ' << GetParrot() << ' ' << c << '\n';
}

  int main()
  {
      CParakeet ob(1,2,3);
      ob.Show();

      // GetBird() and GetParrot() are still public here
      cout << ob.GetBird() << ' ' << ob.GetParrot() << '\n';

      return 0;
  }
```

Unit 6

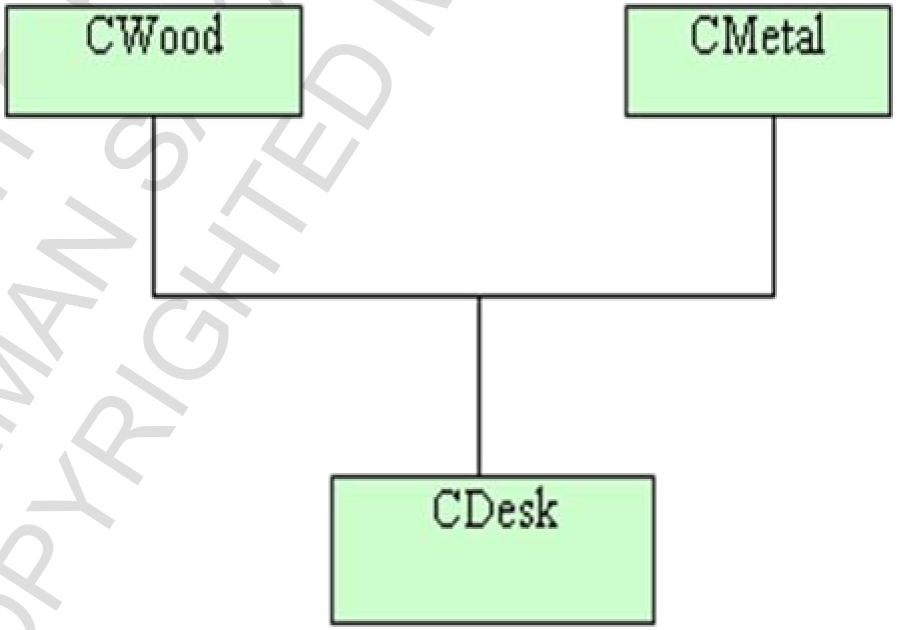
Multiple Inheritance

- Derived Class Directly Inherits Two Base Classes

```
#include <iostream>    // Example 7-7
using namespace std;
class CWood {    //Template for first base class.
    public:
        CWood (int x=3) { m_nType = x; }
        int GetTypeW() { return m_nType; }
    private:
        int m_nType;
 };

 class CMetal {    // Template for second base class.
     public:
         CMetal (int x=2) { m_nType = x; }
         int GetTypeM() { return m_nType; }
     private:
         int m_nType;
 };

```

Unit 6

```
class CDesk : public CWood, public CMetal {
    public:
        // here, y and z are passed directly to CWood and CMetal
        CDesk (int x=1, int y=2, int z=3) : CWood(y), CMetal(z)
        { m_nType = x;}

        void Show();

    private:
        int m_nType;
};

```



Unit 6

```
void CDesk::Show()
{
   // Because base classes were inherited as public, CDesk has access
   // to public member functions of both CWood and CMetal
   cout << GetTypeM () << ' ';
   cout << GetTypeW () << ' ' << m_nType << '\n';
}

int main()
{
    CDesk ob(10, 20, 30);
    ob.Show();

    return 0;
}

```

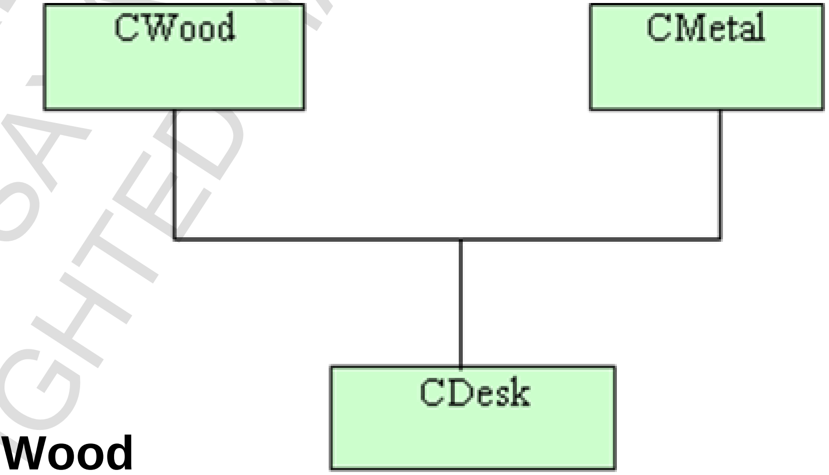
Unit 6

Constructor and Destructor Behavior with Multiple Inheritance

```
#include <iostream>    // Example 7-8

using namespace std;

class CWood {
    public:
        CWood() { cout << "Constructing CWood\n"; }
        ~CWood() { cout << "Destructing CWood\n"; }
};

class CMetal {
    public:
        CMetal() { cout << "Constructing CMetal\n"; }
        ~CMetal() { cout << "Destructing CMetal\n"; }
    private:
        int b;
};
```

Unit 6

```
  // Inherit two base classes.
  class CDesk : public CWood, public CMetal {
      public:
          CDesk() { cout << "Constructing CDesk\n"; }
          ~CDesk() { cout << "Destructing CDesk\n"; }
  };

  int main()
{
      CDesk ob;

      return 0;
  }

```



```
OUTPUT:
Constructing CWood
Constructing CMetal
Constructing CDesk
Destructing CDesk
Destructing CMetal
Destructing CWood
```

Unit 7

Assignment Operator Behavior during Inheritance

Version I

```
// (Assignment Operator function only in the CBase class body)
// Constructor from CDerived class is called
// when string is assigned to an object.
#include <iostream>    // Example 7-9
#include <cstring>

using namespace std;

class CBase {
    public:
         CBase (const char* sName);
         const CBase &operator=(const char* sName);
         void Show() { cout << m_aName << "\n"; }
     protected:
         char m_aName[20];
 };
```

Unit 7

```
CBase::CBase(const char* sName)
{

    cout << "CBase Constructor\n";
    strcpy(m_aName, sName);
}

const CBase &CBase::operator=(const char* sName)
{
    cout << "CBase Operator function\n";
    strcpy(m_aName,sName);
    return *this;
}

```

Unit 7

```
class CDerived : public CBase {
    public:
        CDerived(const char* sName) : CBase(sName)
        {
            cout << "CDerived Constructor\n";
        }
};

int main()
{
    CDerived ObjD1("Suleman");
    ObjD1.Show();
    cout << "\n\n";
    cout << "Call to String Assignment\n";
    ObjD1 = "Saya";
    ObjD1.Show(); // implicit call to the show function
    ObjD1.CBase::Show(); // explicit call to the show function
    return 0;
}
```

Version I OUTPUT:

```
CBase Constructor
CDerived Constructor
Suleman
```

```
Call to String
Assignment
CBase Constructor
CDerived Constructor
Saya
Saya
```

Unit 7

Version II

```
// (Assignment Operator function in CBase and in CDerived class)
// User provided assignment operator function is called for Cderived
// class when string is assigned to an object.
#include <iostream>    // Example 7-10
#include <cstring>

using namespace std;

class CBase {
    public:
         CBase (const char* sName);
         const CBase &operator=(const char* sName);
         void Show() { cout << m_aName << "\n"; }
     protected:
         char m_aName[20];
 };

```

Unit 7

```
CBase::CBase(const char* sName)
{

    cout << "CBase Constructor\n";
    strcpy(m_aName, sName);

}

const CBase &CBase::operator=(const char* sName)
{
    cout << "CBase Operator function\n";
    strcpy(this->m_aName,sName);

    return *this;
}

```

Unit 7

```
class CDerived : public CBase {
    public:
        CDerived(const char* sName) : CBase(sName)
        { // called for member wise copy
            cout << "CDerived Constructor\n";
        }

        const CDerived &operator=(const char* sName);

};

const CDerived &CDerived::operator=(const char* sName)
{
    cout << "CDerived Operator function\n";
    CBase::operator=(sName);

    return *this;
}

```

Unit 7

```
int main()
{
    CDerived ObjD1("Suleman");

    ObjD1.Show();
    cout << "\n\n";

    cout << "Call to String Assignment\n";
    ObjD1 = "Saya";
    // implicit call to the show function
    ObjD1.Show();
    // explicit call to the show function
    ObjD1.CBase::Show();

    return 0;
}

```

Version II OUTPUT:

```
CBase Constructor
CDerived Constructor
Suleman


Call to String Assignment
CDerived Operator function
CBase Operator function
Saya
Saya
```

Unit 8

Virtual Class

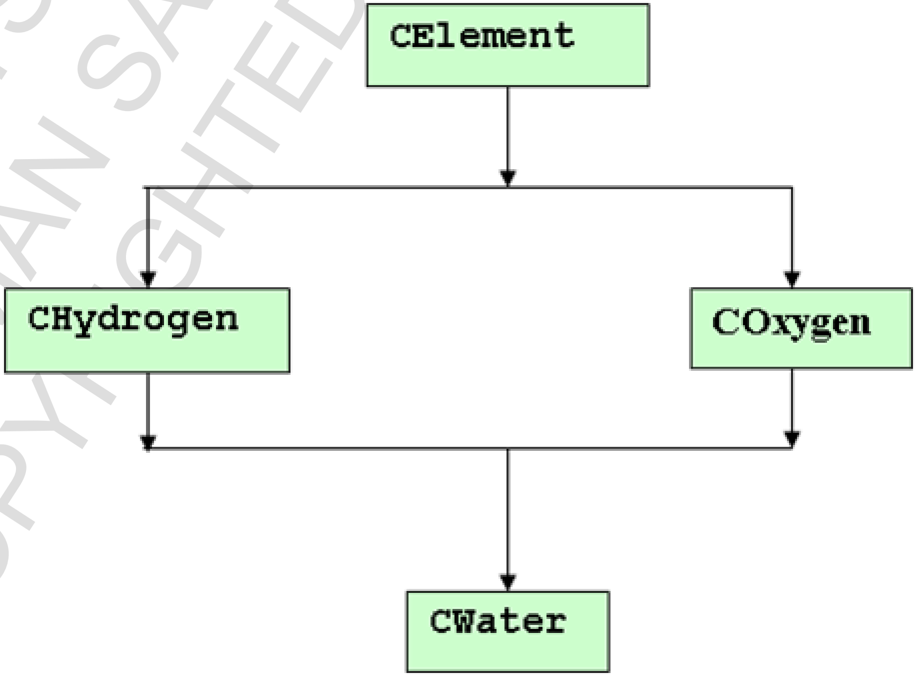
```
#include <iostream>    // Example 7-11

using namespace std;

// Template for the base class.
class CElement {
    public:
        CElement(int x) { m_nTotal = x; }
        int GetTypeE() { return m_nTotal; }

     private:
         int m_nTotal;
 };

```

Unit 8

```
// Template for first sub class.
class CHydrogen : public virtual CElement {
    public:
        CHydrogen (int x) : CElement(x), m_nType(x) { }
        int GetTypeH() { return m_nType; }

    private:
        int m_nType;
};

// Template for second sub class.
class COxygen : public virtual CElement  {
    public:
        COxygen (int x) : CElement(x) { m_nType = x; }
        int GetTypeO() { return m_nType; }

    private:
        int m_nType;
};
```

Unit 8

```
// Directly inherit two base classes.
class CWater : public CHydrogen, public COxygen {
    public:
  CWater (int h, int o, int e, int w) : CHydrogen(h), COxygen(o),
                                        CElement(e), m_nCups(w) { }
    // Because sub classes are inherited as public, CWater has access
    // to public elements of both CHydrogen and COxygen
    void show();

    private:
        int m_nCups;
};

```



Unit 8

```
 void CWater::show()
 {
 //    cout << GetTypeE () << ' ' << GetTypeO ();
 //    cout << ' ' << m_nCups << '\n';
     cout << "Water cups = " << m_nCups << '\n';
     cout << "Hydrogen Elements = " << GetTypeH() << '\n';
     cout << "Oxygen Elements = " << GetTypeO() << '\n';
     cout << "Total Elements = " << GetTypeE() << '\n';
 }

int main()
 {
     CWater ob(2, 1, 2, 5);
     ob.show();

     return 0;
 }
```