MODULE 14
INHERITANCE I

My Training Period: hours

Abilities

- Able to understand the inheritance concept.
- Able to understand and use base class (parent class).
- Able to understand and use derived class (child class).
- Able to understand the use of the preprocessor directive to avoid the multiple inclusion of the same file.
- Able to understand the class hierarchy.

10.1 Introduction

- Inheritance permits to **reuse code** from the already available classes, also gives you the flexibility to do modification if the old code doesn't exactly fit the task of the new project.
- It doesn't make sense to start every new project with new classes, from scratch since most of the code will certainly be repeated in several programs such as common tasks or routines.
- You are less likely to make an error if you leave the original alone and only add new features to it. Another reason for using inheritance is if the project requires the use of several classes which are very similar but slightly different.
- In this Module, we will concentrate on the **mechanism of inheritance** and how to build it into a program.
- C++ allows you to inherit all or part of the members and methods of classes, modify some, and add new ones that not available in the base class.

14.2 Starting Point

- Examine the file named `vehicle.h` for simple class which we will use to begin our study of inheritance. It consists of four simple methods which can be used to manipulate data pertaining to our vehicle.

```
1.  //class declaration part, vehicle.h, header file
2.  //save and include this file in your project
3.  //do not compile or run
4.
5.  #ifndef  VEHICLE_H        //preprocessor directive
6.  #define  VEHICLE_H
7.
8.  //----------class declaration part – the interface----------
9.  class Cvehicle
10. {
11.    protected: //new keyword
12.        int  wheels;
13.        int  weight;
14.        public:
15.        void  initialize(int input_wheels, float  input_weight);
16.        int   get_wheels(void);
17.        float get_weight(void);
18.        float wheel_load (void);
19. };
20. #endif
```

20 Lines of codes

Program 14.1: `vehicle.h` program, declaration of `Cvehicle` class

- We will eventually refer to this as a **base class** or **parent class**, but for the time being, we will simply use it like any other class to show that it is indeed just a normal class.
- Note that we will explain the added keyword `protected` shortly. Figure `14.1` is a graphical representation of the `Cvehicle` class. Ignore lines `5`, `6`, and `11`, they will be explained in detail later. This program cannot be compiled or executed because it is only a header file but makes sure there are no errors such as typing errors.

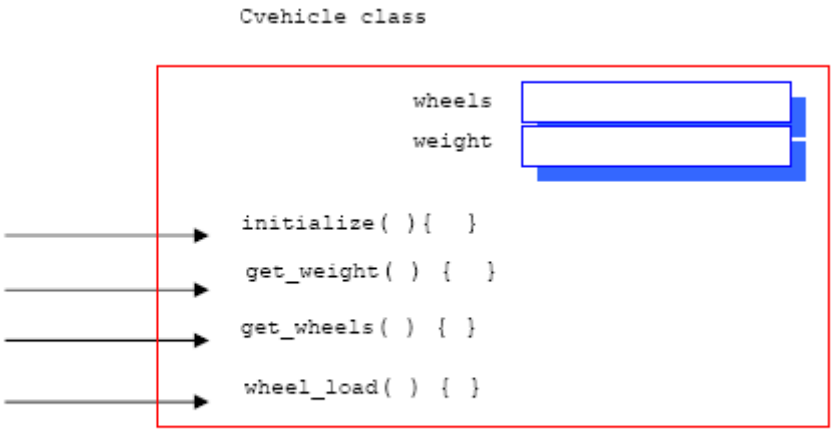


Figure 14.1: Graphical representation of the `Cvehicle` class with two member variables and four methods.

14.3 The `Cvehicle` Class Implementation – `vehicle.cpp`

- Examine the file named `vehicle.cpp`, and you will find that it is the implementation of the vehicle class. The `initialize()` method assigns the values input as parameters to the `wheels` and `weight` variables.

```
1.   //Program vehicle.cpp, implementation part,
2.   //compile without error, generating object file, do not run
3.   #include "vehicle.h"
4.
5.   //---------------class implementation part---------------------
6.   //initialize to data input by user
7.   void  Cvehicle::initialize(int input_wheels, float input_weight)
8.   {
9.      wheels = input_wheels;
10.     weight = input_weight;
11.  }
12.
13.  //get the number of wheels of this vehicle
14.  int  Cvehicle::get_wheels()
15.  {
16.     return wheels;
17.  }
18.
19.  //return the weight of this vehicle
20.  float Cvehicle::get_weight()
21.  {
22.     return  weight;
23.  }
24.
25.  //return the load on each wheel
26.  float  Cvehicle::wheel_load()
27.  {
28.     return  (weight/wheels);
29.  }
```

29 Lines of codes

Program 14.2: `vehicle.cpp`, implementation program of `Cvehicle` class

- We have methods to return the number of `wheels` and the `weight`, and finally, we have one that does a calculation to return the load on each wheel. At this point, we are more interested in learning how to implement the **interface** to the classes.
- We will use it as a **base class** later. Compile this program without error, generating object file, in preparation for the next program example, but you cannot execute it because there is no `main()` entry point.

14.4 Using The Vehicle Class – The `main()` program

- The file named `transprt.cpp` uses the `Cvehicle` class in similar manner as we have illustrated in the previous Module. This should be an indication to you that the `Cvehicle` class is just a normal class as defined in C++.

```
1.   //program transprt.cpp, the main program,
2.   //compile and run
3.
4.   #include <iostream.h>
5.   #include <stdlib.h>
6.   #include "vehicle.h"
7.   //user define header file and put it in the same folder as this program
8.
9.   void main()
10.  {
11.    Cvehicle    car, motorcycle, truck, sedan_car;
12.    //4 objects instantiated
13.
14.      //data initialization
15.      car.initialize(4,3000.0);
16.      truck.initialize(20,30000.0);
17.      motorcycle.initialize(2,900.0);
18.      sedan_car.initialize(4,3000.0);
19.
20.     //Display the data
21.     cout<<"The car has "<<car.get_wheels()<< " tires.\n";
22.     cout<<"Truck has load "<<truck.wheel_load()<<" kg per tire.\n";
23.     cout<<"Motorcycle weight is  "<<motorcycle.get_weight()<<" kg.\n";
24.     cout<<"Weight of sedan car is "<<sedan_car.get_weight()<<" kg, and has
25.                                    "<<sedan_car.get_wheels()<<" tires.\n";
26.
27.     system("pause");
28.  }
```

28 Lines: Output:

Program 14.3: `transprt.cpp` the main program

- Inheritance uses an existing class and adds new functionalities to the new class, to accomplish another, possibly more complex task.
- This program declares four objects of the `Cvehicle` class as shown in the code segment below:

```
Cvehicle    car, motorcycle, truck, sedan_car;
//4 objects instantiated
```

- Initializes them as shown below:

```
//data initialization
car.initialize(4, 3000.0);
truck.initialize(20, 30000.0);
motorcycle.initialize(2,900.0);
sedan_car.initialize(4, 3000.0);
```

- And prints out a few of the data values. Compile and run this program.

14.5 Derived Class

- Examine the file named `car.h`, for our first example of using a derived class. The `Cvehicle` class is inherited due to the `":public Cvehicle"` code added to line `12` as shown below:

```
class Ccar : public Cvehicle
```

```
1.   //another class declaration car.h
2.   //save and include this file in your project
```

```
3.   //do not compile or run.
4.
5.   #ifndef   CAR_H
6.   #define   CAR_H
7.
8.   #include  "vehicle.h"
9.
10.  //-----------derived class declaration part-----------------
11.  //Ccar class derived from Cvehicle class
12.  class Ccar : public Cvehicle
13.  {
14.      int passenger_load;
15.      public:
16.      //This method will be used instead of the same
17.      //method in Cvehicle class - overriding
18.      void initialize(int input_wheels, float input_weight, int people = 4);
19.      int passengers(void);
20.  };
21.
22.  #endif
```

22 Lines of codes

Program 14.4: `car.h` program, declaration of derived class `Ccar`

- This derived class named `Ccar` is composed of all the information included in the **base class** `Cvehicle`, and all of its own additional information. Even though we did nothing to the class named `Cvehicle`, we made it available in this `car.h` program.
- In fact, it can be used as a normal class and a base class in the **same program**.
- A class that inherits another class is called a **derived class** or a **child class**.
- Likewise the terminology for the inherited class is called a **base class**, but **parent class** and **super class** are sometimes used.
- A base class is rather general class which can cover a wide range of objects attributes or behavior or properties, whereas a derived class is somewhat more specific but at the same time more useful.
- For simple example consider your family. You and your siblings inherited many characteristics of your mother and/or father such as eye and hair color. Your father and/or mother are base classes, whereas you and your siblings are derived classes.
- In this case, the `Cvehicle` base class can be used to declare objects that represent plane, trucks, cars, bicycles, or any other vehicles you can think up.
- The class named `Ccar` however can only be used to declare an object that is of type `Ccar` because we have limited kinds of data that can be intelligently used with it. The car class is therefore more restrictive and specific than the `Cvehicle` class.
- If we wish to get even more specific, we could define a derived class using `Ccar` as the base class, name it `Csports_car`, and include information such as `red_line_limit` for the tachometer or `turbo_type_engine` as new member variables.
- Then, the `Ccar` class would therefore be used as a derived class and a base class at the same time, so it should be clear that these names refer to how the class is used.

14.6 Derived Class Declaration

- A derived class is defined by including the header file of the base class as is done in line 8 as shown below:

```
#include "vehicle.h"
```

- And then the name of the base class is given following the name of the derived class separated by a colon (`:`) as is illustrated in line `12` as shown below:

```
class Ccar : public Cvehicle
```

- Ignore the keyword `public` immediately following the colon in this line. This defines public inheritance and we will discuss it later.
- All objects declared as being of class `Ccar` therefore **are composed of the two variables from the** `Cvehicle` class because they inherit those variables, and the **single** member variable declared in the `Ccar` class named `passenger_load`, and the total member variable is listed below:

  - `int wheels;`

- `int weight;`
- `int passenger_load;`

- An object of this class also will have **three of the four methods** of `Cvehicle` as listed below:

  - `get_wheels( ) { }`
  - `get_weight( ) { }`
  - `wheel_load( ) { }`

- And the **two new methods**

  - `initialize( ) { }`
  - `passengers( ) { }`

- The method named `initialize()` which is part of the `Cvehicle` class will not be available here because it is hidden by the local version of `initialize()` which is a part of the `Ccar` class. **The local method will be used if the name is repeated, allowing you to customize your new class**. Figure 14.2 is a graphical representation of an object of this class.
- Note once again that the implementation for the base class only needs to be supplied in its compiled form of the `vehicle.cpp` file. The source code for implementation can be hidden.
- The header file for the base class, `vehicle.h`, must be available as a text file since the class definitions are required in order to use the class.

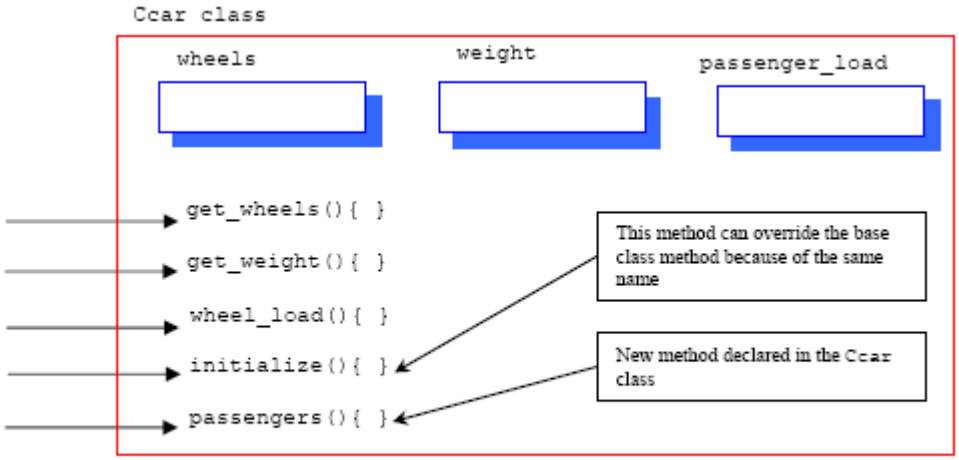
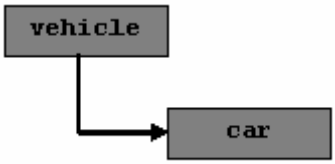


Figure 14.2: Graphical representation of an object `Ccar` derived class.

- The following figure is an illustration of our simple class hierarchy. Here, `car` inherits some of the common `vehicle`'s properties.



14.7 The `Ccar` Class Implementation – `car.cpp`

- Examine the program named `car.cpp` which is the implementation file for the `Ccar` class.
- The first thing you should notice, this file has no indication of the fact that it is a derived class of any other class, and can only be determined by inspecting the preprocessor directive of the header file `#include "car.h"`.

```
1.  //Implementation file car.cpp for derived Ccar class
2.  //compile without error and include in your project,
3.  //Do not run
4.
```

```
5.   #include  "car.h"
6.
7.   //-------implementation part of the derived class car.h----------
8.   void Ccar::initialize(int input_wheels, float input_weight, int people)
9.   {
10.    passenger_load = people;
11.    wheels = input_wheels;
12.    weight = input_weight;
13.  }
14.
15.  int Ccar::passengers(void)
16.  {
17.     return    passenger_load;
18.  }
```

18 Lines of codes

Program 14.5: `car.cpp,` implementation of the `Ccar` derived class.

- The implementations for the two new methods are similar as in other class, nothing new. Compile this program without error, this will generate object file for later use.

14.8 Another Derived Class – `Ctruck`

- Examine the program named `truck.h`, an example of another derived class that uses the `Cvehicle` class as its based class. This class will specialize in those things that pertain to trucks.

```
1.    //Another class declaration, truck.h,
2.    //Save this file in your project
3.    //Do not compile or run
4.
5.    #ifndef  TRUCK_H
6.    #define  TRUCK_H
7.
8.    #include  "vehicle.h"
9.
10.  //----------class declaration part of derived class truck------------
11.  class Ctruck : public Cvehicle
12.  {
13.      int     passenger_load;
14.      float   payload;
15.     public:
16.     void  init_truck(int how_many = 2, float max_load = 24000.0);
17.     float efficiency(void);
18.     int   passengers(void);
19.  };
20.  #endif
```

20 Lines of codes

Program 14.6: `truck.h`, derived class declaration of `Ctruck` derived class

- This derived class **adds two more variables and three more methods** compared to the base class, `Cvehicle`, is specific information for truck vehicle. Figure `14.3` is the graphical representation of the `Ctruck` class.

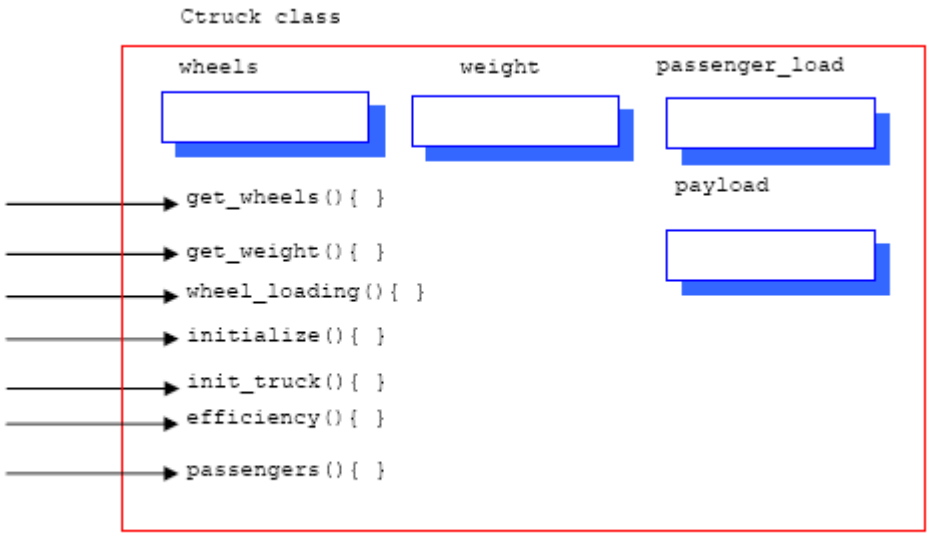


Figure 14.3: Graphically representation of the new derived class, `Ctruck`

- `Ccar` and `Ctruck` classes have absolutely nothing to do with each other, they only happen to be derived classes from the same base class.
- Note that both the `Ccar` and `Ctruck` classes have methods named `passengers()` but this causes no problems and is perfectly valid. This issue will be discussed later.
- If classes are related in some way, and they certainly are if they are both derived classes of a common base class, you would expect them to be doing somewhat similar things. In this situation there is a good possibility that a method name would be repeated in both child classes.
- No need to compile and run this program just makes sure there are no other errors such as typing error.

14.9 The `Ctruck` Class Implementation – `truck.cpp`

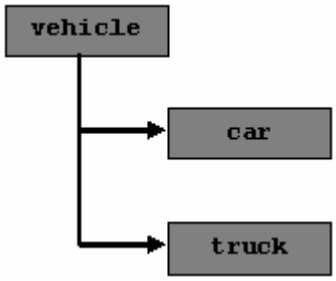
- Examine the program named `truck.cpp` for the implementation of the `Ctruck` class.
- Compile this program without error, generating object file, in preparation for our program example that uses all the three classes `Cvehicle`, `Ccar`, `Ctruck`, defined in this Module.

```
1.    //Derive class truck.cpp implementation
2.    //Compile and include in your project
3.    //Do not run
4.
5.    #include  "truck.h"
6.
7.    //--------implementation part of the Ctruck derived class------
8.    void Ctruck::init_truck(int how_many, float max_load)
9.    {
10.      passenger_load = how_many;
11.      payload = max_load;
12.   }
13.
14.   float Ctruck::efficiency(void)
15.   {
16.      return  payload/(payload + weight);
17.   }
18.
19.   int Ctruck::passengers(void)
20.   {
21.      return passenger_load;
22.   }
```

22 Lines of codes

Program 4.7: `truck.cpp` implementation of `Ctruck` derived class

- Now, our simple class hierarchy is illustrated below. Both `car` and `truck` inherit some of the common `vehicle`'s properties.

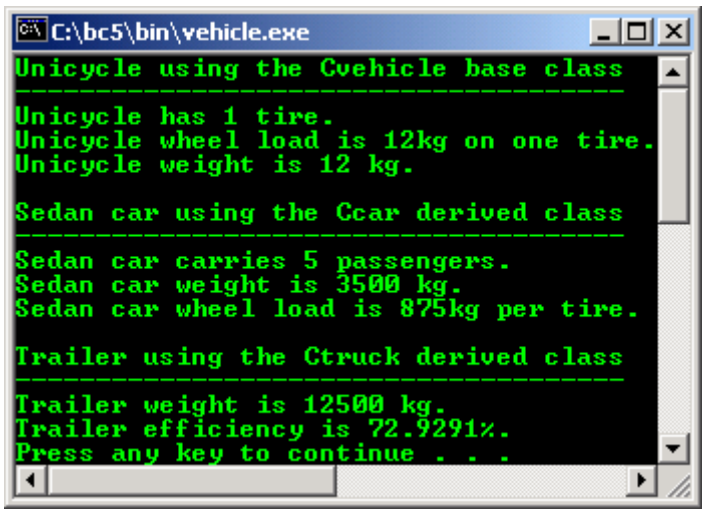


14.10 Using All The Classes

- Examine the program named `allvehicle.cpp` carefully. This main program uses all three of the classes we have been discussing in this Module.
- Before compiling and running this program, make sure these files: `vehicle.h, vehicle.cpp, car.h, car.cpp, truck.h, truck.cpp and allvehicle.cpp` are in one project. Also, you have to compile without error the `vehicle.cpp, car.cpp` and `truck.cpp` files, generating object files. Compile and execute the `allvehicle.cpp` main program.
- Also make sure the first `main()` program `transprt.cpp` not in this project. We only need one `main()` program.

```
1.  //The new main program for inheritance, allvehicle.cpp
2.  //Compile and run this program
3.
4.  #include  <iostream.h>
5.  #include  <stdlib.h>
6.  #include  "vehicle.h"  //the interface of the Cvehicle class
7.  #include  "car.h"      //the interface of the Ccar class
8.  #include  "truck.h"    //the interface of the Ctruck class
9.
10. void main()
11. {
12.    Cvehicle   unicycle;      //base class
13.
14.    cout<<"Unicycle using the Cvehicle base class\n";
15.    cout<<"--------------------------------------\n";
16.
17.     unicycle.initialize(1,12.5);
18.
19.    cout<<"Unicycle has "<<unicycle.get_wheels()<<" tire.\n";
20.    cout<<"Unicycle wheel load is "<<unicycle.wheel_load()<<"kg on one
21.                                                          tire.\n";
22.    cout<<"Unicycle weight is "<<unicycle.get_weight()<<" kg.\n\n";
23.
24.     Ccar  sedan;                  //derived class
25.
26.    cout<<"Sedan car using the Ccar derived class\n";
27.    cout<<"--------------------------------------\n";
28.
29.          sedan.initialize(4,3500.0,5);
30.
31.    cout<<"Sedan car carries "<<sedan.passengers()<<" passengers.\n";
32.    cout<<"Sedan car weight is "<<sedan.get_weight()<<" kg.\n";
33.    cout<<"Sedan car wheel load is "<<sedan.wheel_load()<<"kg per tire.\n\n";
34.
35.          Ctruck  trailer;                //derived class
36.
37.    cout<<"Trailer using the Ctruck derived class\n";
38.    cout<<"--------------------------------------\n";
39.
40.       trailer.initialize(18,12500.0);
41.       trailer.init_truck(1,33675.0);
42.
43.    cout<<"Trailer weight is "<<trailer.get_weight()<<" kg.\n";
44.    cout<<"Trailer efficiency is "<<100.00 * trailer.efficiency()<<"%.\n";
45.
46.    system("pause");
47.  }
```

47 Lines:Output:



Program 14.8: `allvehicle.cpp`, the main program

- It uses the base class `Cvehicle` and also two derived classes to declare objects. This was done to illustrate that all three classes can be used in a single program.
- All three of the header files for the classes are included in lines `6` through `8`, so the program can use the components of the classes as shown below:

```
#include  "vehicle.h"  // the interface of the Cvehicle class
#include  "car.h"      // the interface of the Ccar class
#include  "truck.h"    // the interface of the Ctruck class
```

- Notice the implementations of the three classes (`vehicle.cpp`, `car.cpp` and `truck.cpp`) are not in view here and do not need to be in view. We only need these files in compiled form.
- This allows the code to be used without access to the source code for the actual implementation of the class. However, it should be clear that the header file definition must be available to act as an interface.
- In this program example, only one object of each class is declared and used as an example, but as many as desired could be declared and used in order to accomplish your programming task.
- You will notice how clean and uncluttered the source code is for this program. The classes were developed, debugged, and stored away and the interfaces were kept very simple.

14.11 The `#ifndef ... #endif`

- Look at the directive in lines `5`, `6` and `20` in the `vehicle.h` file as shown below. These directives have been discussed in Module related to preprocessor directive.

```
#ifndef  VEHICLE_H        // preprocessor directive
#define  VEHICLE_H
...
...
...
#endif
```

- When we define the derived class `Ccar`, we are required to supply it with the full definition of the interface to the `Cvehicle` class since `Ccar` is derived class of `Cvehicle` and must know all about its base class.
- We do that by including the vehicle class (`vehicle.h`) into the `Ccar` class, and the `Ccar` class can be compiled. The `Cvehicle` class must also be included in the header file of the `Ctruck` class for the same reason.
- When we get to the `allvehicle.cpp` program, we must inform it the details of all three classes, so all three header files must be included as is done in lines `6` through `8` of `allvehicle.cpp`, but this leads to a problem.
- When the preprocessor gets to the `Ccar` class, it includes the `Cvehicle` class because it is listed in the `Ccar` class header file, but since the `Cvehicle` class already included in line `6` of `allvehicle.cpp`, it is **included twice** and we attempt to re-declare the class `Cvehicle`.

- Of course it is the same declaration, but the compiler **simply doesn't allow re-declaration of a class**. We allow the double inclusion of the file (`vehicle.h` and `car.h`) and at the same time prevent the double inclusion of the class (`Cvehicle`) by building a bridge around it using the word `VEHICLE_H`.
- If the word is already defined, the declaration is skipped, but if the word is not defined, the declaration is included and `vehicle.h` is defined at that time. The end result is the **actual inclusion of the class only once**, even though the file is included more than once.
- Even though `ANSI-C` allows multiple definitions of entities, provided the definitions are identical, `C++` does not permit this. The primary reason is because the compiler would have great difficulty in knowing if it has already made a constructor call for the redefined entity, if any and this also create un optimize codes.
- A multiple constructor call for a single object could cause great havoc, so `C++` was defined to prevent any multiple constructor calls by making it illegal to redefine any entity.
- The name `VEHICLE_H` was chosen as the word because it is the name of the file, with the period replaced by the underline. If the name of the file is used systematically in all of your class definitions, you cannot have a name clash because the filename of every class must be unique.
- Figure `14.4` and `14.5` has shown that the inheritance concepts will create class hierarchy. From the top, a base class with general description of the object, traversing down of the derived class levels with more details description of the objects.
- This class hierarchy provides very powerful 'tool' for programming. We can reuse readily available classes which has been developed and tested. At the same time we can combine with our own new defined classes.
- This will shorten the program development process and cycle, enhances the software quality and increase the productivity of the programmers.
- For example, Microsoft Foundation Classes (MFC) have hundreds readily available classes which arranged in hierarchical manner and you can imagine that there should be thousands of member variables and methods defined, readily available for us to use.

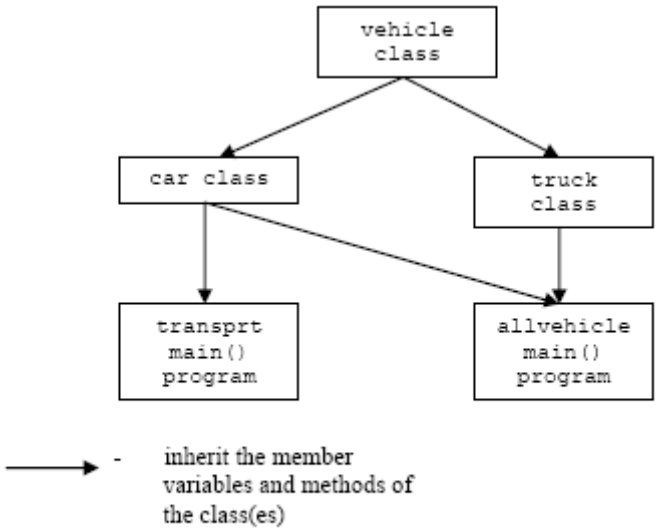


Figure 14.4: Inheritance among base and derived classes

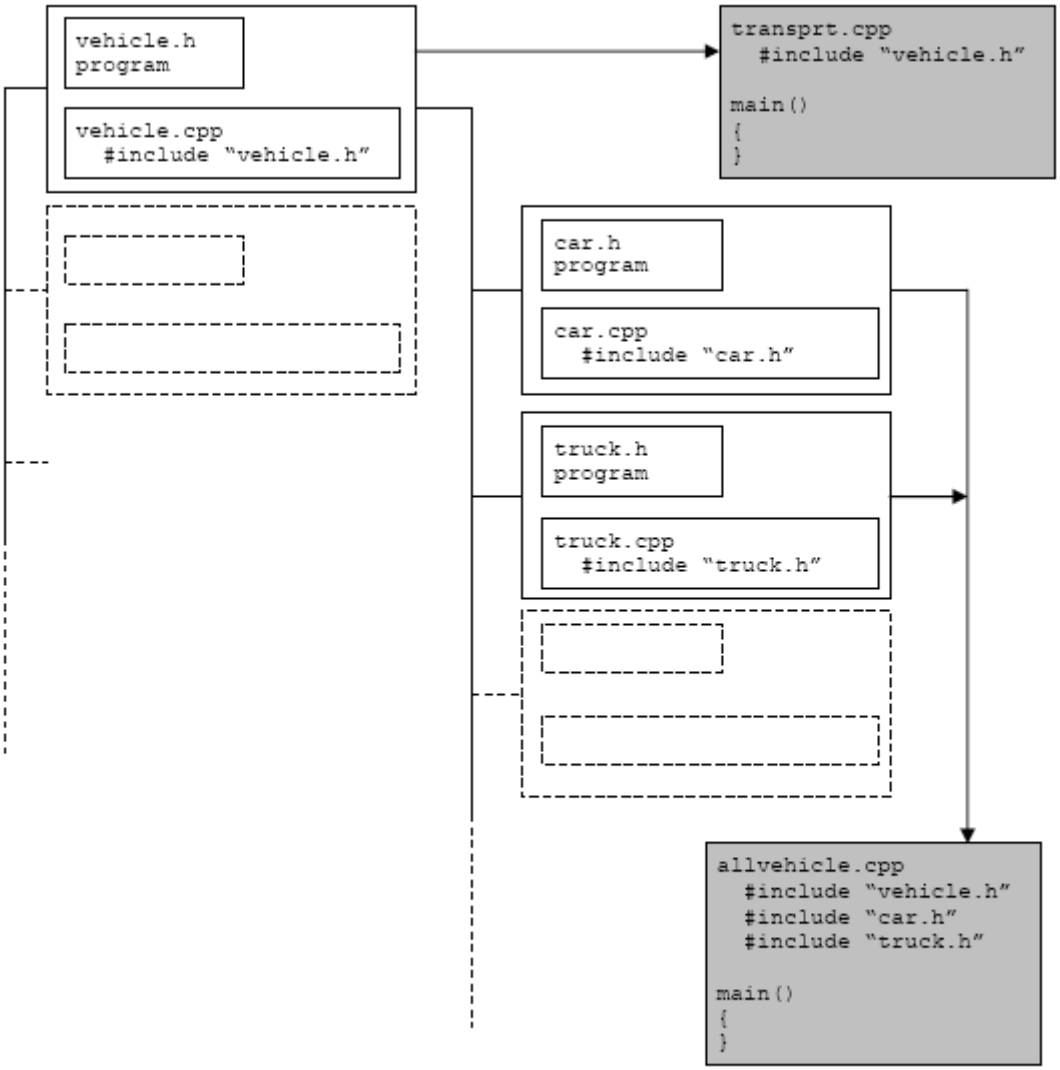


Figure 14.5: Graphically representation of the simple hierarchy of classes.

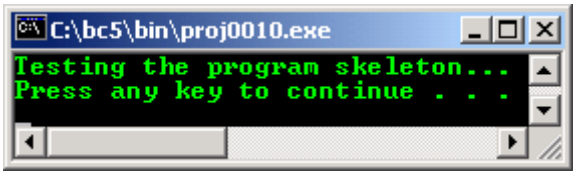
Program Examples and Experiments

- Let start with a simple program skeleton.

```
//inheritance
#include <iostream.h>
#include <stdlib.h>

//base class
class Base
{
	//member variables and member
	//functions...
	public:
	Base(){}
	~Base(){}
	protected:
	private:
};

//derived class...
class Derived:public Base
{
	//same as normal class actually...
	//member variables and member function...
	public:
	Derived(){}
```

```
        ~Derived(){}
        private:
        protected:
};

void main()
{
        cout<<"Testing the program skeleton..."<<endl;
        system("pause");
}
```

Output:

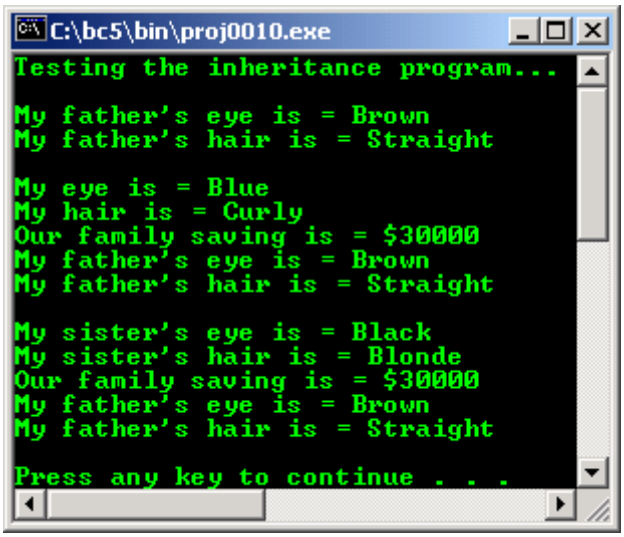


- Then, let try the real objects.

```
//inheritance
#include <iostream.h>
#include <stdlib.h>

//---class declaration and implementation------
//base class...
class MyFather
{
        //member variables and member
        //functions...
        private:
        char* EyeColor;
        char* HairType;
        double FamSaving;

        protected:
        public:
        MyFather(){}
        ~MyFather(){}
        char* GetEye()
        { return EyeColor = "Brown";}
        char* GetHair()
        { return HairType = "Straight";}
        double GetSaving()
        {return FamSaving = 30000;}
};

//derived class...
class MySelf:public MyFather
{
        //same as normal class actually...
        private:
        char* MyEye;
        char* MyHair;

        public:
        MySelf(){}
        ~MySelf(){}
        char* GetMyEye()
        { return MyEye = "Blue";}
        char* GetMyHair()
        {return MyHair = "Curly";}
        protected:
};

//another derived class...
class MySister:public MyFather
{
        private:
        char* SisEye;
        char* SisHair;

        public:
        MySister(){}
        ~MySister(){}
```

```
        char* GetSisEye()
        {return SisEye = "Black";}
        char* GetSisHair()
        { return SisHair = "Blonde";}
};

//-----------main program--------------
int main()
{
  //base class object...
  MyFather Test1;

  cout<<"Testing the inheritance program...\n"<<endl;
  cout<<"My father's eye is = "<<Test1.GetEye()<<endl;
  cout<<"My father's hair is = "<<Test1.GetHair()<<endl;

  //derived class object...
  MySelf Test2;
  cout<<"\nMy eye is = "<<Test2.GetMyEye()<<endl;
  cout<<"My hair is = "<<Test2.GetMyHair()<<endl;
  //the following are inherited from MyFather class...
  cout<<"Our family saving is = $"<<Test2.GetSaving()<<endl;
  cout<<"My father's eye is = "<<Test2.GetEye()<<endl;
  cout<<"My father's hair is = "<<Test2.GetHair()<<endl;

  //another derived class object...
  MySister Test3;
  cout<<"\nMy sister's eye is = "<<Test3.GetSisEye()<<endl;
  cout<<"My sister's hair is = "<<Test3.GetSisHair()<<endl;
  //the following are inherited from MyFather class...
  cout<<"Our family saving is = $"<<Test3.GetSaving()<<endl;
  cout<<"My father's eye is = "<<Test3.GetEye()<<endl;
  cout<<"My father's hair is = "<<Test3.GetHair()<<"\n\n";
  system("pause");
  return 0;
}
```

Output:

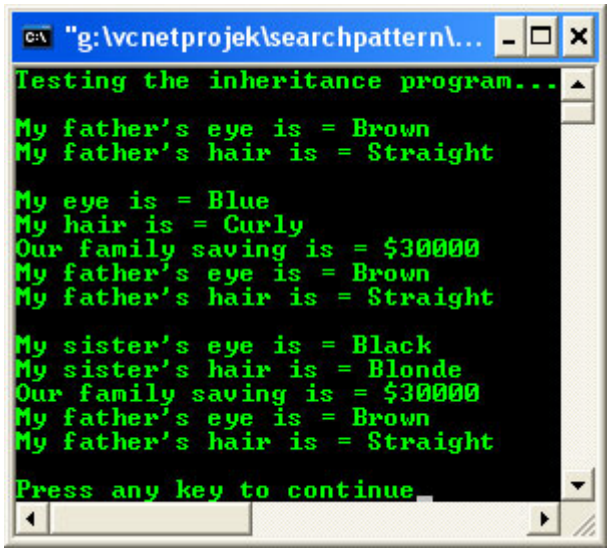


- If you have noticed, the program examples in this Module become smaller, simpler and manageable compared to what is in the previous Modules.
- Let try compiling in VC++/VC++ .Net.

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

//---class declaration and implementation------
//base class...
class MyFather
{
     //member variables and member
     //functions...
     private:
     char* EyeColor;
```

```
        char* HairType;
        double FamSaving;

        protected:
        //protected members here…

        public:
        MyFather(){}
        ~MyFather(){}
        char* GetEye()
        { return EyeColor = "Brown";}
        char* GetHair()
        { return HairType = "Straight";}
        double GetSaving()
        {return FamSaving = 30000;}
};

//derived class...
class MySelf:public MyFather
{
        //same as normal class actually...
        private:
        char* MyEye;
        char* MyHair;

        public:
        MySelf(){}
        ~MySelf(){}
        char* GetMyEye()
        { return MyEye = "Blue";}
        char* GetMyHair()
        {return MyHair = "Curly";}
        protected:
};

//another derived class...
class MySister:public MyFather
{
        private:
        char* SisEye;
        char* SisHair;

        public:
        MySister(){}
        ~MySister(){}
        char* GetSisEye()
        {return SisEye = "Black";}
        char* GetSisHair()
        { return SisHair = "Blonde";}
};

//-----------main program---------------
int main()
{
  //base class object...
  MyFather Test1;

  cout<<"Testing the inheritance program...\n"<<endl;
  cout<<"My father's eye is = "<<Test1.GetEye()<<endl;
  cout<<"My father's hair is = "<<Test1.GetHair()<<endl;

  //derived class object...
  MySelf Test2;
  cout<<"\nMy eye is = "<<Test2.GetMyEye()<<endl;
  cout<<"My hair is = "<<Test2.GetMyHair()<<endl;
  //the following are inherited from MyFather class...
  cout<<"Our family saving is = $"<<Test2.GetSaving()<<endl;
  cout<<"My father's eye is = "<<Test2.GetEye()<<endl;
  cout<<"My father's hair is = "<<Test2.GetHair()<<endl;

  //another derived class object...
  MySister Test3;
  cout<<"\nMy sister's eye is = "<<Test3.GetSisEye()<<endl;
  cout<<"My sister's hair is = "<<Test3.GetSisHair()<<endl;
  //the following are inherited from MyFather class...
  cout<<"Our family saving is = $"<<Test3.GetSaving()<<endl;
  cout<<"My father's eye is = "<<Test3.GetEye()<<endl;
  cout<<"My father's hair is = "<<Test3.GetHair()<<"\n\n";
  return 0;
}
```

Output:



- The following is a previous example compiled using **g++**. To link more than one object files, it is better to create `makefile`.

```
//**************herit2.cpp**************
//**************FEDORA 3, g++ x.x.x****
//inheritance
#include <iostream>
using namespace std;

//---class declaration and implementation------
//base class...
class MyFather
{
        //member variables and member
        //functions...
        private:
        char* EyeColor;
        char* HairType;
        double FamSaving;

        protected:
        public:
        MyFather(){}
        ~MyFather(){}
        char* GetEye()
        {return EyeColor = "Brown";}
        char* GetHair()
        {return HairType = "Straight";}
        double GetSaving()
        {return FamSaving = 30000;}
};

//derived class...
class MySelf:public MyFather
{
        //same as normal class actually...
        private:
        char* MyEye;
        char* MyHair;

        public:
        MySelf(){}
        ~MySelf(){}
        char* GetMyEye()
        {return MyEye = "Blue";}
        char* GetMyHair()
        {return MyHair = "Curly";}
        protected:
             //...
};
```

```
//another derived class...
class MySister:public MyFather
{
        private:
        char* SisEye;
        char* SisHair;

        public:
        MySister(){}
        ~MySister(){}
        char* GetSisEye()
        {return SisEye = "Black";}
        char* GetSisHair()
        {return SisHair = "Blonde";}
};

//-----------main program--------------
int main()
{
  //base classes object...
  MyFather Test1;

  cout<<"Testing the inheritance program...\n"<<endl;
  cout<<"My father's eye is = "<<Test1.GetEye()<<endl;
  cout<<"My father's hair is = "<<Test1.GetHair()<<endl;

  //derived class object...
  MySelf Test2;
  cout<<"\nMy eye is = "<<Test2.GetMyEye()<<endl;
  cout<<"My hair is = "<<Test2.GetMyHair()<<endl;
  //the following are inherited from MyFather class...
  cout<<"Our family saving is = $"<<Test2.GetSaving()<<endl;
  cout<<"My father's eye is = "<<Test2.GetEye()<<endl;
  cout<<"My father's hair is = "<<Test2.GetHair()<<endl;

  //another derived class object...
  MySister Test3;
  cout<<"\nMy sister's eye is = "<<Test3.GetSisEye()<<endl;
  cout<<"My sister's hair is = "<<Test3.GetSisHair()<<endl;
  //the following are inherited from MyFather class...
  cout<<"Our family saving is = $"<<Test3.GetSaving()<<endl;
  cout<<"My father's eye is = "<<Test3.GetEye()<<endl;
  cout<<"My father's hair is = "<<Test3.GetHair()<<"\n\n";
  return 0;
}
```

[bodo@bakawali ~]$ g++ herit2.cpp -o herit2
[bodo@bakawali ~]$ ./herit2

```
Testing the inheritance program...

My father's eye is = Brown
My father's hair is = Straight

My eye is = Blue
My hair is = Curly
Our family saving is = $30000
My father's eye is = Brown
My father's hair is = Straight

My sister's eye is = Black
My sister's hair is = Blonde
Our family saving is = $30000
My father's eye is = Brown
My father's hair is = Straight
```

----------o0o----------

Further reading and digging:

1. Check the best selling C/C++ and object oriented books at Amazon.com.