

Chapter 12 - Classes

1. What is a Class?

3rd Generation Programming Languages started off with a small set of pre-defined types, each having a set of operations defined on them. All data had to be manipulated using those types. Later, 3gls such as Pascal, PL/1, and C let programmers create limited data types, but these types had no operations defined on them. Programmers had to assemble groups of free functions to manipulate the data of the created types. In addition, programmers were given direct access to the data stored in an object of the given type. A new function that was malicious or poorly written could wreak havoc on the results of a software system that had run reliably for years.

Classes are the principle means for programmers in C++ to create new types. A *class* includes *data members* that store the data, called *state information* of the *class* and can be as complex as any data structure. A *class* includes *member functions* that define the operations possible on the new type. The *interface* to the *class* is definable so that the data will only be manipulated in the manner that the creator of the *class* intended.

Classes act as in the way that the types of the original 3gls did. Like these older types, *classes* cannot be directly used in a program. Instead, *objects* of the *class* must be declared in order for any features of the *class* to be used.

2. Defining a Class

The first step in creating a *class* is to define the *access* sections, that is those *member function* headers and *data members* of the *class*. A *class* definition statement has the following form:

```
class class-name
  compound statement;
```

Inside the compound statement of the *class* definition goes the *access* mode definitions. Two possible *access* mode sections are *public* and *private*. The *public* section contains those things that a programmer using an object of the *class* will be able to access. The *private* section contains those things that a programmer using an object of the *class* will not be able to access. The form of a class definition with these sections looks like this:

```
class class-name {
  public:
  :
  :
  private:
  :
  :
};
```

Into the *public* and *private* sections go the *member function* definitions or prototypes and *variable object* declarations that are part of the class. A programmer using an object of the class cannot write new functions to manipulate the private state information of the class, but if forced to use the public functions that are available. This insures that data will only be manipulated in a proven manner.

The header of a special member function called a *constructor* is required within the *public* section of a *class*. A *constructor* performs any initializations that need to be performed when an object of the *class* is declared. There must be at least one *constructor* in a *class*, but there may be as many as needed via function overloading. *Constructors* are easily recognized in a *class* definition because they have the same name as the *class*.

The following is an example of a definition of a class that would model a simple employee information form. This class is stored in a file named *employee.h*.

```
#ifndef _EMPLOYEE
#define _EMPLOYEE
using namespace std;

class employee {
public:
    employee( );           // default constructor
    employee(char *initName,
             double initRate,
             double initHours); // preferred constructor
    char * returnName( ); // member function
    double returnPay( );  // member function
private:
    char name[25];        // state variables
    double rate;
    double hours;
};

#endif
```

The *private* section defines those things that a programmer using an object of the class cannot access directly. The variable objects *name*, *rate* and *hours* hold the state information of the class *employee*. They model an hourly wage employee, albeit in a limited fashion. To manipulate *name*, *rate* or *hours*, a programmer must use the tools provided in the *public* section.

The *public* section defines the interface to objects of class *employee*, i.e. those things that the programmer can directly access. In this example, the *constructor* with no parameters exists only because it must. The *constructor* without arguments is referred to as the *default constructor*. Every *class* requires a *default constructor* (though many compilers do not enforce this). The *constructor* with the three arguments is the *constructor* is intended to be used by the programmer using an object of class *employee*. This *constructor* allows an object of class *employee* to be created with useful values stored

as state information. The function *returnName()* returns the name of the employee. The function *returnPay()* returns the result of the rate of pay multiplied by hours worked.

The following is an example of a the use of the class *employee*. [*Warning: As the member functions have not yet been defined, compiling this program will only produce a large set of errors!*]

```
#include <iostream>
#include "employee.h"
using namespace std;

int main( ) {
    employee joe("Joe Burns", 8.90, 37);
    employee sarah("Sarah Bergeron", 9.50, 40);

    cout << joe.returnName() << " made $"
         << joe.returnPay( ) << endl;
    cout << sarah.returnName( ) << " made $"
         << sarah.returnPay( ) << endl;

    return 0;
}
```

If the *member functions* of class *employee* had been complete when this program was run, the output should look like the following:

```
Joe Burns made $329.3
Sarah Bergeron made $380
```

3. Implementing Member Functions

Member functions are usually implemented outside of the class definition.

Professional programmers usually create the function implementations for a class in file separate from the file that contains the definition. This allows them to compile the functions into a permanent binary form. Once the binary file is created, the class is available via the linker without repetitious recompiling. This technique is used to greatly reduce the compile time in large programming projects after each small change.

Student programs often consist of classes that are under development. For that reason student programmers usually put the function implementations in the same file as the class definition. This limits the amount of “file hopping” that occurs during debugging “learning” code. When implementing *member functions* in the same file as the *class* definition, the functions are placed after the class definition, but inside the *ifndef* and *endif* preprocessor instructions.

Since *member function* implementations are outside of the *class* definition, each *member function* must be associated with the correct *class*. This is done by placing the *scope resolution operator* (two colons, '::') between the *class* name and function name in the function header. This has the following form:

```
class-name::function-header
compound statement
```

Here is the implementation of the member function *returnName*. Note the use of the class name and the *scope resolution operator*.

```
char * employee::returnName() {
    return name;
}
```

The following is the complete declaration and implementation of the *class employee* in the file *employee.h*. Notice the *include* of *string.h* to access the function *strcpy*.

```
#ifndef _EMPLOYEE
#define _EMPLOYEE
#include <string.h>
using namespace std;

class employee {
public:
    employee();
    employee(char * initName, double initRate, double initHours);
    char * returnName();
    double returnPay();
private:
    char name[25];
    double rate;
    double hours;
};

employee::employee() {
    strcpy(name, "none");
    rate = 0.0;
    hours = 0.0;
}
```

```

employee::employee(char * initName, double initRate, double initHours) {
    strcpy(name, initName);
    rate = initRate;
    hours = initHours;
}

char * employee::returnName() {
    return name;
}

double employee::returnPay() {
    return rate * hours;
}

#endif

```

The example function main of section 2 can now be run.

Exercises

1. What is a *class*?
2. What is a *data member* of a *class*?
3. What is a *member function* of a *class*?
4. What is a *constructor*?
5. What is the purpose of the *public* area of a *class*?
6. What is the purpose of the *private* area of a *class*?

Programming Assignment 12.1

Implement the example *employee class* in a header file called *employee.h*. Include this header file in a program that uses the *class* and all of its *member functions*.

7. Car and Driver

Programmer created classes are the basis for programs in C++. By this is meant that the basic ideas that need to be expressed to model the problem are expressed as *classes*. For example, it is not hard to imagine creating *classes* called *customer*, *order*, *invoice* and *inventory* to create a computer system to model a business. In fact, a class called *business* could be created that would use objects of these *classes* as its building blocks. Building more complex *classes* out of more basic ones is typical of the programming process as expressed in an object-oriented language such as C++.

This section will develop an example of a class called *car*. *Class car* will model a race car that can run along a straight track. In the next section, *class car* will be used to build another *class* that will be used to stage races between two objects of *class car*.

The first step is to create the header file *car.h*. This file will contain the preprocessor commands, the definition of *class car*, and the implementation of the *member functions* of *class car*.

The precompiler instructions are to protect the compiler against unnecessarily compiling the program again and to include the standard library files *stdlib.h* and *time.h*, which allow the program to use random number functions to simulate the role of a pair of dice. These instructions are as follows:

```
#ifndef _CAR          // if car.h has been included, do not include
#define _CAR
#include <stdlib.h>
#include <time.h>
using namespace std;

    // ***** All other code in car.h goes here! *****

#endif
```

Next, the *class car* must be placed between the *include* instructions and the *endif*.

```
class car {
public:

private:

};
```

Since an object of *class car* need only be identifiable as it moves along a straight path, objects of *class car* need only keep track of the symbol for the car and its progress along the path. The variables *carChosen* and *location* will allow this information to be stored.

```
private:
    char carChosen;          // car symbol
    int location;           // distance car has traveled in a
```

The programmer interface (*public* area) for the *class car* needs to include a *default constructor*, a *constructor* that will allow the programmer to specify the symbol of the car, a *member function* that will return the symbol of the car, a *member function* that will move the car some random distance along the track, and a *member function* that will return the location of the car. The only function in the *public* area that will require a parameter will be the *constructor* that allows the programmer to specify a symbol for the car.

Here is the *public* area of *class car*:

```
public:
    car();                // default constructor
    car(char initCar);   // preferred constructor,
                        // allows car symbol to be specified
    void moveCar();      // move card random distance
    char returnCar();    // returns the car symbol
    int  returnLocation(); // returns distance car has traveled
                        // in a straight line
```

The complete declaration of *class car* is as follows:

```
class car {
public:
    car();                // default constructor
    car(char initCar);   // preferred constructor,
                        // allows car symbol to be specified
    void moveCar();      // move card random distance
    char returnCar();    // returns the car symbol
    int  returnLocation(); // returns distance car has traveled
                        // in a straight line

private:
    char carChosen;      // car symbol
    int  location;       // distance car has traveled in a
                        // straight line
};
```

The default *constructor* of *class car* will set the location to 0 and assign a default symbol character to the *car* object. In addition, the *constructor* will need to set the seed value for the random number function to be used in the function *moveCar*. Here is the implementation of the default *constructor*:

```
car::car() {
    srand( (unsigned)time( NULL ) ); // sets seed for random number
    location = 0;                    // set the location of car to start
                                    // position
    carChosen = 'X';                 // set the car symbol to X
}
```

The preferred *constructor* of *class car* will set the location to 0 and assign a programmer specified character as a symbol for the *car* object. In addition, the constructor will also need to set the seed value for the random function that is used in the function *moveCar*. Here is the implementation of the preferred constructor:

```
car::car(char initCar) {
    srand( (unsigned)time( NULL ) );    // set seed value for random number
    location = 0;                       // set location of car to start posit
    carChosen = initCar;                // set car symbol to passed value
}
```

The function *moveCar* should increment the location value of the car by a random but limited amount each time it is called. Here is the *moveCar* function.

```
void car::moveCar() {
    location += rand() % 3 + rand() % 3 + 1;    // move 1 to 5 places
}
```

In the function *moveCar*, each call to *rand* generates a number from 0 to 32,727. By using the remainder (modulus) operator after each call to the *rand* function, the return of each call is limited to 0, 1, or 2. The full expression most often has the effect of adding a 3 to *location* and least often of adding a 1 or a 5 to *location*.

The function *returnLocation* simply returns the *location* value of the *car* object. Here is the function *returnLocation*:

```
int car::returnLocation() {
    return location;
}
```

The function *returnCar* simply returns the symbol used to identify the *car* object. Here is function *returnCar*:

```
char car::returnCar() {
    return carChosen;
}
```

What follows on the next page is the complete listing of *class car*.

```

#ifndef _CAR           // if car.h has been included, do not include again
#define _CARD

#include <stdlib.h>     // for rand() and srand()
#include <time.h>      // for time() - seed for srand
using namespace std;

class car {
public:
    car( );           // default constructor
    car(char initCar); // preferred constructor,
                    // allows car symbol to be specified
    void moveCar( ); // move card random distance
    char returnCar( ); // returns the car symbol
    int returnLocation( ); // returns distance car has traveled
                    // in a straight line

private:
    char carChosen; // car symbol
    int location; // distance car has traveled in a
                // straight line
};

car::car( ) {
    srand( (unsigned)time( NULL ) ); // sets seed for random number
    location = 0; // set the location of car to start position
    carChosen = 'X'; // set the car symbol to X
}

car::car(char initCar) {
    srand( (unsigned)time( NULL ) ); // set seed value for random number
    location = 0; // set location of car to start posit
    carChosen = initCar; // set car symbol to passed value
}

void car::moveCar( ) {
    location += rand() % 3 + rand() % 3 + 1; // move 1 to 5 places
}

int car::returnLocation( ) { return location; }

char car::returnCar( ) { return carChosen; }

#endif

```

Programming Assignment 12.2

Implement the class *car* in a header file called *car.h*. Test it using the following program.

```
#include <iostream>
#include "car.h" // the full path to car.h may be required
using namespace std;

void main() {
    // ***** define 2 car objects *****
    car dangerous_dan('D');
    car entropic_eddy('E');

    // ***** role the dice five times for each car *****
    dangerous_dan.moveCar();
    entropic_eddy.moveCar();
    dangerous_dan.moveCar();
    entropic_eddy.moveCar();
    dangerous_dan.moveCar();
    entropic_eddy.moveCar();
    dangerous_dan.moveCar();
    entropic_eddy.moveCar();
    dangerous_dan.moveCar();
    entropic_eddy.moveCar();

    // ***** output the final position of each car *****
    cout << dangerous_dan.returnCar() << ": "
         << dangerous_dan.returnLocation() << endl;
    cout << entropic_eddy.returnCar() << ": "
         << entropic_eddy.returnLocation() << endl;
}
```

8. A Day a the Races

The *class car* can now be used to build a more complex *class*. This class will function as a racetrack and will depend on the *class car* to function.

The *class track* will be created in the file *track.h* and will start with the usual preprocessor instructions.

```
#ifndef _TRACK    // if this has been included,
#define _TRACK    // do not include again
#include <iostream> // to display the track
#include <conio.h>  // for getch() used in pausing
#include "car.h"
using namespace std;

                // ***** All other code goes here! *****

#endif
```

The *public* area of *class track* includes the *constructor* and the *member functions* *moveCars* and *display*. The *private* area of *class track* includes the *data member* objects *X* and *Y*.

```
class track {
public:
    track();                // creates two cars
    void moveCars();        // moves two cars a random distance
    void display();         // displays the track and car
private:
    car X, Y;              // creates competing vehicles
};
```

The *constructor* for *class track* is responsible for creating two objects of *class car* and assigning a symbol to each. Since the variables *X* and *Y* are already declared as objects of *class car* via a call to the default constructor of *class car*, both *X* and *Y* must be reassigned. This is done by a call to the preferred constructor of *class car* in an assignment statement. This has the effect of setting the state information of *X* and *Y* to the desired symbol values. Here is the *class track constructor*:

```
track::track() {
    X = car('X');          // set the state information of X
    Y = car('Y');          // set the state information of Y
}
```


The following is a complete listing of class track:

```

#ifndef _TRACK           // if this has been included,
#define _TRACK          // do not include again
#include <iostream>      // to display the track
#include <conio.h>       // for getch() used in pausing
#include "car.h"
using namespace std;

class track {
public:
    track();             // creates two cars
    void moveCars();    // moves two cars a andom distance
    void display();     // displays the track and car
private:
    car X, Y;           // creates competing vehicles
};

track::track() {
    X = car('X');      // set the state information of X
    Y = car('Y');      // set the state information of Y
}

void track::moveCars() {
    X.moveCar();
    Y.moveCar();
}

void track::display() {
    cout << "\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n";
    cout.fill('.');

    cout.width(X.returnLocation());
    cout << X.returnCar() << endl;

    cout.width(Y.returnLocation());
    cout << Y.returnCar() << endl;

    cout << "\n. . . Press any key to continue . . . ";
    getch();
}

#endif

```

Programming Assignment 12.3

Implement class car and class track. Create a program to use and test all the functions in the classes.

Programming Assignment 12.4

Add a third car object to class track so that the race is run between three cars.

Programming Assignment 12.5

Implement the member functions of the following class declaration:

```
class student {
public:
    student();                // default constructor, creates student "noone"
    student(char * initName,  // preferred constructor
            char * initId,
            int initYearInSchool);
    char * returnName();     // returns the name of the student
    char * returnId();       // returns the Id of the student
    int returnYearInSchool(); // returns the grade (level) of the student
    void promote();          // adds one to the grade (level) of the student
private:
    char name[30];
    char id[10];
    int yearInSchool;
};
```

Use this class in a program that demonstrates all member functions.

Programming Assignment 12.6

Create and implement a class that models the properties of teachers. Private data information should include name, id, and subject taught. Use this class in a program that demonstrates all member functions.

Programming Assignment 12.7

Create and implement a class that models the properties of dinosaurs. Use this class in a program that demonstrates all member functions.