C++ Programming: From Problem Analysis to Program Design, Fourth Edition

Chapter 10: Classes and Data Abstraction

Objectives

In this chapter, you will:

- Learn what a stream is and examine input and output streams
- Explore how to read data from the standard input device
- Learn how to use predefined functions in a program
- Explore how to use the input stream functions `get`, `ignore`, `putback`, and `peek`

Objectives (continued)

- Become familiar with input failure
- Learn how to write data to the standard output device
- Discover how to use manipulators in a program to format output
- Learn how to perform input and output operations with the `string` data type
- Become familiar with file input and output

Classes

A **class** is a collection of a fixed number of components. The components of a class are called the **members** of the class.

The general syntax for defining a class is:

```
class classIdentifier
{
    classMembersList
};
```

in which **classMembersList** consists of variable declarations and/or functions.

The members of a class are classified into three categories: private, public, and protected.

Classes

Following are some facts about `public` and `private` members of a class:

- By default, all members of a class are `private`.
- If a member of a class is `private`, you cannot access it outside of the class. (Example 10-1 illustrates this concept.)
- A `public` member is accessible outside of the `class`. (Example 10-1 illustrates this concept.)
- To make a member of a class `public`, you use the member access specifier `public` with a colon, :.

Classes

```
class clockType
{
public:
    void setTime(int, int, int);
    void getTime(int&, int&, int&) const;
    void printTime() const;
    void incrementSeconds();
    void incrementMinutes();
    void incrementHours();
    bool equalTime(const clockType&) const;

private:
    int hr;
    int min;
    int sec;
};
```

Accessor and Mutator

Accessor function: A member function of a class that only accesses (that is, does not modify) the value(s) of the member variable(s).

Mutator function: A member function of a class that modifies the value(s) of the member variable(s).

A member function of a class is called a **constant function** if its heading contains the reserved word `const` at the end. For example, the member functions **`getTime`**, **`printTime`**, and

Accessor and Mutator

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

class clockType
{
public:
    void setTime(int, int, int);
    void getTime(int&, int&, int&) const;
    void printTime() const;
    void incrementSeconds();
    void incrementMinutes();
    void incrementHours();
    bool equalTime(const clockType&) const;

private:
    int hr;
    int min;
    int sec;
};
```

Accessor and Mutator

```
int main()
{
    clockType myClock;
    clockType yourClock;

    int hours;
    int minutes;
    int seconds;

        //Set the time of myClock
    myClock.setTime(5, 4, 30);                          //Line 1

    cout << "Line 2: myClock: ";                        //Line 2
    myClock.printTime();  //print the time of myClock    Line 3
    cout << endl;                                       //Line 4

    cout << "Line 5: yourClock: ";                      //Line 5
    yourClock.printTime(); //print the time of yourClock Line 6
    cout << endl;                                       //Line 7

        //Set the time of yourClock
    yourClock.setTime(5, 45, 16);                       //Line 8

    cout << "Line 9: After setting, yourClock: ";       //Line 9
    yourClock.printTime(); //print the time of yourClock Line 10
    cout << endl;                                       //Line 11

        //Compare myClock and yourClock
    if (myClock.equalTime(yourClock))                   //Line 12
        cout << "Line 13: Both times are equal."
             << endl;                                   //Line 13
```

Md. Sharif Hossen
Lecturer, Dept. of ICT, Comilla University, Bangladesh

Accessor and Mutator

```
else                                                    //Line 14
    cout << "Line 15: The two times are not equal."
         << endl;                                       //Line 15

cout << "Line 16: Enter the hours, minutes, and "
     << "seconds: ";                                    //Line 16
cin >> hours >> minutes >> seconds;                     //Line 17
cout << endl;                                           //Line 18

    //Set the time of myClock using the value of the
    //variables hours, minutes, and seconds
myClock.setTime(hours, minutes, seconds);               //Line 19

cout << "Line 20: New myClock: ";                       //Line 20
myClock.printTime();   //print the time of myClock     Line 21
cout << endl;                                           //Line 22

    //Increment the time of myClock by one second
myClock.incrementSeconds();                             //Line 23

cout << "Line 24: After incrementing myClock by "
     << "one second, myClock: ";                        //Line 24
myClock.printTime();   //print the time of myClock     Line 25
cout << endl;                                           //Line 26

    //Retrieve the hours, minutes, and seconds of the
    //object myClock
myClock.getTime(hours, minutes, seconds);               //Line 27

    //Output the value of hours, minutes, and seconds
cout << "Line 28: hours = " << hours
     << ", minutes = " << minutes
     << ", seconds = " << seconds << endl;              //Line 28

    return  0;
}//end main
```

Accessor and Mutator

```
void clockType::setTime(int hours, int minutes, int seconds)
{
    if (0 <= hours && hours < 24)
        hr = hours;
    else
        hr = 0;

    if (0 <= minutes && minutes < 60)
        min = minutes;
    else
        min = 0;

    if (0 <= seconds && seconds < 60)
        sec = seconds;
    else
        sec = 0;
}
```

Constructors

There are two types of constructors: with parameters and without parameters. The constructor without parameters is called the default constructor. Constructors have the following properties:

- The name of a constructor is the same as the name of the class.
- A constructor, even though it is a function, has no type. That is, it is neither a value-returning function nor a `void` function.
- A class can have more than one constructor. However, all constructors of a class have the same name.
- If a class has more than one constructor, the constructors must have different formal parameter lists. That is, either they have a different number of formal parameters or, if the number of formal parameters is the same, then the data type of the formal parameters, in the order you list, must differ in at least one position.
- Constructors execute automatically when a class object enters its scope. Because they have no types, they cannot be called like other functions.
- Which constructor executes depends on the types of values passed to the class object when the class object is declared.

Constructors

class clockType by including two constructors:

```
class clockType
{
public:
    void setTime(int, int, int);
    void getTime(int&, int&, int&) const;
    void printTime() const;
    void incrementSeconds();
    void incrementMinutes();
    void incrementHours();
    bool equalTime(const clockType&) const;
    clockType(int, int, int);  //constructor with parameters
    clockType();  //default constructor

private:
    int hr;
    int min;
    int sec;
};
```

Md. Sharif Hossen
Lecturer, Dept. of ICT, Comilla University, Bangladesh

Constructors

class clockType includes two constructors: one with three parameters and one without any parameters.

```
clockType::clockType(int hours, int minutes, int seconds)
{
    if (0 <= hours && hours < 24)
        hr = hours;
    else
        hr = 0;

    if (0 <= minutes && minutes < 60)
        min = minutes;
    else
        min = 0;

    if (0 <= seconds && seconds < 60)
        sec = seconds;
    else
        sec = 0;
}

clockType::clockType()  //default constructor
{
   hr = 0;
   min = 0;
   sec = 0;
}
```

Md. Sharif Hossen
Lecturer, Dept. of ICT, Comilla University, Bangladesh

Constructors

class clockType includes two constructors: one with three parameters and one without any parameters.

```
clockType::clockType(int hours, int minutes, int seconds)
{
    setTime(hours, minutes, seconds);
}
```

Invoking Constructors

Recall that when a class object is declared, a constructor is automatically executed. Because a class might have more than one constructor, including the default constructor,

Invoking the Default Constructor

Suppose that a class contains the default constructor. The syntax to invoke the default constructor is:

```
className classObjectName;
```

For example, the statement:

```
clockType yourClock;
```

declares **yourClock** to be an object of type **clockType**.

```
clockType yourClock();  //illegal object declaration
```

Invoking a Constructor with Parameters

Invoking a Constructor with Parameters

Suppose a class contains constructors with parameters. The syntax to invoke a constructor with a parameter is:

```
className classObjectName(argument1, argument2, ...);
```

in which **argument1**, **argument2**, and so on are either a variable or an expression.

Consider the statement:

```
clockType myClock(5, 12, 40);
```

Md. Sharif Hossen
Lecturer, Dept. of ICT, Comilla University, Bangladesh

Invoking a Constructor with Parameters : Example

EXAMPLE 10-6

Consider the following class definition:

```
class inventory
{
public:
    inventory();                                  //Line 1
    inventory(string);                            //Line 2
    inventory(string, int, double);               //Line 3
    inventory(string, int, double, int); //Line 4

    //Add additional functions

private:
    string name;
    int itemNum;
    double price;
    int unitsInStock;
};
```

Invoking a Constructor with Parameters : Example

This class has four constructors and four member variables. Suppose that the definitions of the constructors are as follows:

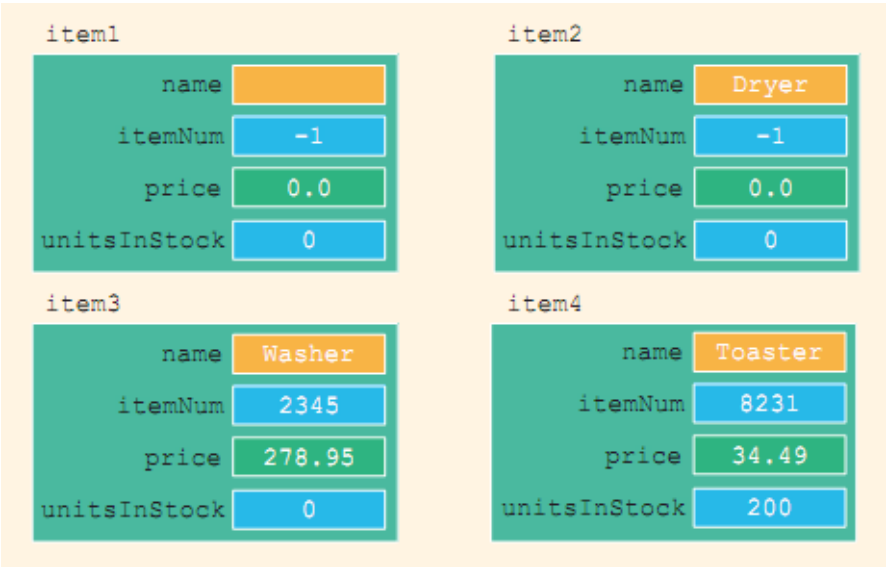
```
inventory::inventory() //default constructor
{
     name = "";
     itemNum = -1;
     price = 0.0;
     unitsInStock = 0;
}

inventory::inventory(string n)
{
     name = n;
     itemNum = -1;
     price = 0.0;
     unitsInStock = 0;
}

inventory::inventory(string n, int iNum, double cost)
{
     name = n;
     itemNum = iNum;
     price = cost;
     unitsInStock = 0;
}
```

Md. Sharif Hossen
Lecturer, Dept. of ICT, Comilla University, Bangladesh

Invoking a Constructor with Parameters : Example

```
inventory::inventory(string n, int iNum, double cost, int inStock)
{
     name = n;
     itemNum = iNum;
     price = cost;
     unitsInStock = inStock;
}
```

```
inventory item1;
inventory item2("Dryer");
inventory item3("Washer", 2345, 278.95);
inventory item4("Toaster", 8231, 34.49, 200);
```



Md. Sharif Hossen
Lecturer, Dept. of ICT, Comilla University, Bangladesh

Invoking a Constructor with Parameters : Example

```
class circleType
{
public:
    void setRadius(double r);
      //Function to set the radius.
      //Postcondition: if (r >= 0) radius = r;
      //               otherwise radius = 0;

    double getRadius();
      //Function to return the radius.
      //Postcondition: The value of radius is returned.

    double area();
      //Function to return the area of a circle.
      //Postcondition: Area is calculated and returned.

    double circumference();
      //Function to return the circumference of a circle.
      //Postcondition: Circumference is calculated and returned.

    circleType(double r = 0);
      //Constructor with a default parameter.
      //Radius is set according to the parameter.
      //The default value of the radius is 0.0;
      //Postcondition: radius = r;

private:
    double radius;
};
```

Md. Sharif Hossen
Lecturer, Dept. of ICT, Comilla University, Bangladesh

Invoking a Constructor with Parameters : Example 2

```
class circleType
{
public:
    void setRadius(double r);
      //Function to set the radius.
      //Postcondition: if (r >= 0) radius = r;
      //               otherwise radius = 0;

    double getRadius();
      //Function to return the radius.
      //Postcondition: The value of radius is returned.

    double area();
      //Function to return the area of a circle.
      //Postcondition: Area is calculated and returned.

    double circumference();
      //Function to return the circumference of a circle.
      //Postcondition: Circumference is calculated and returned.

    circleType(double r = 0);
      //Constructor with a default parameter.
      //Radius is set according to the parameter.
      //The default value of the radius is 0.0;
      //Postcondition: radius = r;

private:
    double radius;
};
```

Invoking a Constructor with Parameters : Example 2

```
void circleType::setRadius(double r)
{
    if (r >= 0)
        radius = r;
    else
        radius = 0;
}

double circleType::getRadius()
{
    return radius;
}

double circleType::area()
{
    return 3.1416 * radius * radius;
}

double circleType::circumference()
{
    return 2 * 3.1416 * radius;
}

circleType::circleType(double r)
{
    setRadius(r);
}
```

Invoking a Constructor with Parameters : Example 2

```
//The user program that uses the class circleType

#include <iostream>
#include <iomanip>
#include "circleType.h"

using namespace std;

int main()                                                    //Line 1
{                                                             //Line 2
    circleType circle1(8);                                    //Line 3
    circleType circle2;                                       //Line 4

    double radius;                                            //Line 5

    cout << fixed << showpoint << setprecision(2);            //Line 6

    cout << "Line 7: circle1 - "
         << "radius: " << circle1.getRadius()
         << ", area: " << circle1.area()
         << ", circumference: " << circle1.circumference()
         << endl;                                             //Line 7

    cout << "Line 8: circle2 - "
         << "radius: " << circle2.getRadius()
         << ", area: " << circle2.area()
         << ", circumference: " << circle2.circumference()
         << endl << endl;                                     //Line 8

    cout << "Line 9: Enter the radius of a circle: ";         //Line 9
    cin >> radius;                                            //Line 10
    cout << endl;                                             //Line 11
```

Invoking a Constructor with Parameters : Example 2

```
    circle2.setRadius(radius);                                  //Line 12

    cout << "Line 13: After setting the radius." << endl;       //Line 13
    cout << "Line 14: circle2 - "
         << "radius: " << circle2.getRadius()
         << ", area: " << circle2.area()
         << ", circumference: " << circle2.circumference()
         << endl;                                               //Line 14

    return 0;                                                   //Line 15
}//end main                                                     //Line 16
```

Programming Exercise

1. Design and implement a `class` `dayType` that implements the day of the week in a program. The `class` `dayType` should store the day, such as `Sun` for Sunday. The program should be able to perform the following operations on an object of type `dayType`:

   a. Set the day.

   b. Print the day.

   c. Return the day.

   d. Return the next day.

   e. Return the previous day.

   f. Calculate and return the day by adding certain days to the current day. For example, if the current day is Monday and we add 4 days, the day to be returned is Friday. Similarly, if today is Tuesday and we add 13 days, the day to be returned is Monday.

   g. Add the appropriate constructors.

Programming Exercise

2. (Tic-Tac-Toe) Write a program that allows two players to play the tic-tac-toe game. Your program must contain the `class` `ticTacToe` to implement a `ticTacToe` object. Include a 3-by-3 two-dimensional array, as a `private` member variable, to create the board. If needed, include additional member variables. Some of the operations on a `ticTacToe` object are printing the current board, getting a move, checking if a move is valid, and determining the winner after each move. Add additional operations as needed.