Chapter 10
Introduction to Classes
Objectives

• In this chapter you will learn about:
  – Abstract data types in C++ (Classes)
  – Constructors
  – Destructors
  – Common programming errors
Procedural vs Object-Oriented Programming

• A procedural program consists of one or more algorithms that have been written in computer-readable language
  – Input and display of program output take a back seat to processing
    • Clear emphasis on formulas and calculations
• An object oriented approach fits graphically windowed environments
• Abstract data types: Central to creation of objects; a user-defined data type rather than a built-in data type
Data Types

• **Data type**: Combination of data and associated operations

• A data type defines *both* the types of data and the types of operations that can be performed on the data
  
  – *Data type = Allowable Data Values + Operational Capabilities*

• Operations in C++ are an inherent part of each data type
## Built-in Data Types

<table>
<thead>
<tr>
<th>Capability</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Define one or more variables of the data type</td>
<td>int a, b;</td>
</tr>
<tr>
<td>Initialize a variable at definition</td>
<td>int a = 5;</td>
</tr>
<tr>
<td>Assign a value to a variable</td>
<td>a = 10;</td>
</tr>
<tr>
<td>Assign one variable’s value to another variable</td>
<td>a = b;</td>
</tr>
<tr>
<td>Perform mathematical operations</td>
<td>a + b</td>
</tr>
<tr>
<td>Perform relational operations</td>
<td>a &gt; b</td>
</tr>
<tr>
<td>Convert from one data type to another</td>
<td>a = int (7.2);</td>
</tr>
</tbody>
</table>

**Table 10.1 C++ Built-In Data Type Capabilities**
Abstract Data Types (continued)

- **Abstract data type (ADT):** *User defined type* that specifies both a type of data and the operations that can be performed on it
  - User defined types are required when you want to create objects that are more complex than simple integers and characters
- **Data structure:** How data is stored
- **Class:** C++ name for an abstract data type
Class Construction

• A class is usually constructed in two parts
  – Declaration section
    • Declares both the data types and functions for the class
    • We usually store this in a header file (file ends in a .h)
  – Implementation section
    • Defines the functions whose prototypes have been declared in the declaration section
    • We usually store this in a separate .cpp file (from our main.cpp)
Class Construction (continued)

- Class declaration section:

```cpp
// class declaration section
class className
{
    data members // the variables
    function members // prototypes
};
// class implementation section
function definitions
```
Class Construction (continued)

- **Class members**: Both the **variables** and the **functions** listed in the declaration section
- **Data members** or **instance variables**: **Variables** listed in the declaration section
- **Member functions**: **Functions** listed in the declaration section
- When a function is part of a class it is referred to as a **method** to denote class membership
Class Construction (continued)

- Initial upper case letter in class name *Date* not required but followed by convention
- **Keywords** `public` and `private` are access specifiers that define access rights
  - `private`: Indicates that class member can only be accessed by class functions
- Restricting user from access to data storage implementation details is called **data hiding**
- After a class category like `private` is designated, it remains in force until a new category is specified
Class Construction (continued)

• **public functions** *can* be called from outside the class

• In general, all class functions should be **public** so that they provide capabilities to manipulate class variables from outside the class

• The function with same name as class is the class’s **constructor function**
  – Used to initialize class data members with values
Class Construction (continued)

• Implementation section is where member functions declared in the declaration section are written.

• General form for functions written in the implementation section is the same as all C++ functions with the addition of the class name and the scope resolution operator ::.

```cpp
returnType className::functionName(parameter list)
{
    function body
}
```
Class Construction (continued)

- Variables of a user-declared class type must:
  - Be defined before use in a program
  - Are referred to as objects
- An object name’s attribute is referenced with the period operator
  - `objectName.attributeName`
    - `objectName` is the name of a specific object
    - `attributeName` is the name of a data member defined for the object’s class
Class Construction (continued)

• The syntax for referring to an object’s method is:
  - `objectName.methodName(parameters)`
    • `objectName` is the name of the specific object
    • `methodName` is name of a function defined for the object’s class
Terminology

- **Class**: Programmer defined data type from which objects can be created
- **Objects**: Created from classes
  - Referred to as *instances* of a class
- Process of creating a new object is called **instantiation** of the object
- Each time a new object instantiated a new set of data members belonging to the object is created
  - Values contained in these data members determine the object’s **state**
Constructors

- A **constructor function** is any function with the **same name as its class**
- **Multiple constructors** can be defined for each class as long as they can be distinguished by number and types of their parameters
- A constructor’s intended purpose is to initialize a new object’s data members
- If no constructor function is written the compiler supplies a **default constructor**
- In addition to initialization, a constructor can perform other tasks when it is called
Constructors (continued)

- General format of a constructor includes:
  - The same name as the class to which it belongs
  - No return type (not even `void`)
  - A constructor that does not require arguments is called the **default constructor**

```cpp
className::className(parameter list)
{
    // function body
}
```
Constructors (continued)

Program 10.2

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

// class declaration section
class Date
{
private:
    int month;
    int day;
    int year;
public:
    Date(int = 7, int = 4, int = 2005);  // constructor
};

// class implementation section
Date::Date(int mm, int dd, int yyyy)
{
    month = mm;
    day = dd;
    year = yyyy;
    cout << "Created a new date object with data values "
        << month << ", " << day << ", " << year << endl;
}

int main()
{
    Date a;  // declare an object
    Date b;  // declare an object
    Date c(4,1,2006);  // declare an object
    return 0;
}
```
Calling Constructors

• Constructors are called when an object is created
• Declaration can be made in a variety of ways
  - `Date c(4,1,2006);`
  - `Date c = Date(4,1,2006);`
• An object should never be declared with empty parentheses
  - `Date a();`
    • Not the same as the declaration `Date a;`
    • Does not result in an object being created
Overloaded and Inline Constructors

• Primary difference between a constructor and other user-written functions is how the constructor is called
  – Constructors are called automatically each time an object is created
  – Most other functions must be called explicitly by name

• Inline functions are functions defined in the class declaration section
Destructors

• **Destructor functions**: Counterpart to the constructor functions

• Destructors:
  – Are functions with the same name as constructors but are preceded with a tilde (~)
    • For the `Date` class the destructor name is `~Date()`
  – Take *no parameters and return no values*

• There can **only be one destructor per class**
Destructors (continued)

• Destructors:
  – Called automatically when an object goes out of existence
  – Clean up any undesirable effects the object might leave, such as releasing memory stored in a pointer
Common Programming Errors

• Failing to terminate class declaration section with semicolon
• Including return type with constructor’s prototype or failing to include return type with other functions’ prototypes
• Using same name for a data member as for a member function
• Defining more than one default constructor
• Forgetting to include class name and scope operator, ::, in the function header
Chapter Summary

• A class
  – Is a programmer-defined data type
  – Consists of a declaration and implementation section
• Class functions can be written inline or included in the class implementation section
• A constructor function is a special function that is called automatically each time an object is declared
  – If no constructor is declared, compiler supplies a default
Chapter Summary (continued)

- Default constructor is the term for any constructor that does not require arguments
  - Each class can have only one default constructor
- Objects are created by using a C++ or C style declaration:
  
  ```cpp
  className list-of-objectNames
  (list of initializers);
  ```

- Constructors can be overloaded
  - If a constructor is defined for a class, a user-defined default constructor should also be written, as compiler will not provide one
Chapter Summary (continued)

- A destructor function is called each time an object goes out of scope.
- Destructors have the same name as their class, but preceded with a tilde (~).
- There is only one destructor per class.
- Destructor takes no arguments and returns no value.
- If a user-defined destructor is not included in a class, compiler provides a do-nothing destructor.