Introduction to Classes

CS 1: Problem Solving & Program Design Using C++

Objectives

- Introduce the concept of object-based programming
- Take a peek at classes
- Construct the beginnings of classes with constructors
- Find out about object-oriented programming through an example
- Finally, look at common programming errors

Object-Based Programming

- OBJECT: well suited to programming representation
- Can be specified by two basic characteristics:
  - STATE: how an object appears at the moment
  - BEHAVIOR: how object reacts to external inputs
- Example of an object: A rectangle
  - STATE: shape and location
    - Shape specified by length and width
    - Location defined by corner positions
  - BEHAVIOR: can be displayed on a screen, can change its size

Object-Based Programming (2)

- MODELING: defining object in terms of its state (attributes) and behavior
  - We choose only those attributes and behaviors that are of interest and relevant
  - Important to distinguish between attributes in model and values the attributes can have
- CLASS: attributes and behavior of model
- OBJECT (OF A CLASS): comes into existence when specific values are assigned to attributes

Classes

- CLASS: programmer-defined data type
  - Also called an abstract data type
  - Combination of data and their associated operations (capabilities)
- A C++ class provides mechanism for packaging data structure and functions together in self-contained unit

Class Construction

- Two components
  - DECLARATION SECTION: declares data types and function for class
  - IMPLEMENTATION SECTION: defines functions whose prototypes were declared in declaration section
- Class members
  - Data members (instance variables)
  - Member functions

Format of a Class Definition

```
// Class declaration section
class classname
{
        data members            // Instance variables
        function members        // Inline and prototypes
};

// Class implementation section
function definitions
```

Class Declaration Example

```
class Date
{
   private:
            int month;
            int day;
            int year;
   public:
            Date(int = 7, int = 4, int = 2006);
            void setDate(int, int, int);
            int getDate();
            void showDate(void);
}; // this is a declaration - don't forget the semicolon
```

About Class Declaration

- Enclosed in braces
- Includes variables (data members) and function declarations
- Keywords private and public: Define access rights
  - private: Class members (month, day, and year) can only be accessed using class functions
    - Enforces data security (data hiding)
  - public: class members (functions Date(), setDate(), showDate() can be used outside of the class

About Class Declaration (2)

- CONSTRUCTOR FUNCTION: Initializes class data members with values
  - Constructor function (Date()) has same name as the class
  - Default values for constructor are 7, 4, 2006
  - Constructor function has no return type (a requirement for this special function)
- Two remaining member functions: setDate() and showDate() declared as returning no value (void)

Format of a Member Function

```
return-type className::functionName (parameter list)
{
        function body
}

// Class implementation section
Date::Date (int mm, int dd, int yyyy)
{
        month = mm;
        day = dd;
        year = yyyy;
}
```

Format of a Member Function (2)

```
void Date::setDate (int mm, int dd, int yyyy)
{
        month = mm;
        date = dd;
        year = yy;

        return;
}

int Date::getDate()
{
        return date;
}
```

Format of a Member Function (3)

```
void Date::showDate(void)
{
        cout << "The date is " << setfill('0')
                << setw(2) << month << '/'
                << setw(2) << day << '/'
                << setw(2) << year % 100 << '.' << endl;

        return;
}
```

Implementation Section of Class Definition

- FUNCTIONS: defined same as other C++ functions but also includes scope resolution operator
  - To identify function as member of class
- Implementation and declaration sections declare a class
  - Variables of the class (objects) must still be defined

Example of Main Function Using the Date Class

```
int main()
{
  Date a, b, c(4,1,1998); // declare 3 objects
  b.setDate(12,25,2007);  // assign values to b's data members
  cout << endl;
  a.showDate();      // display object a's values
  b.showDate();      // display object b's values
  c.showDate();      // display object c's values
  cout << endl;
  return 0;
}
```

Description of Main Function

- Three objects of class Date defined
- Constructor function: Date(), automatically called
- Memory allocated for each object
  - Data members of the objects initialized
- Object a: No parameters assigned therefore defaults are used:
  - a.month = 7
  - a.day = 4
  - a.year = 2006

Description of Main Function (2)

- Notation to create objects:
  - objectName.attributeName
- Object b: No parameters assigned, same defaults used
- Object c: Defined with arguments 4, 1, and 1998
  - Three arguments passed into constructor function resulting in initialization of c's data members as:
    - c.month = 4
    - c.day = 1
    - c.year = 1998

Description of Main Function (3)

- All function of class Date are public, therefore
  - b.setDate(12, 25, 2007) is a valid statement inside main() function
  - Calls setDate() function with arguments 12 ,25, 2007
- Important distinction: Data members of class Date are private
  - The statement b.month = 12 is invalid in main()

Description of Main Function (4)

- Last three statements in main() call showDate() to operate on a, b, and c objects
  - Calls result in output displayed

    The date is 07/04/06

    The date is 12/25/07

    The date is 04/01/98

- The statement cout << a; is invalid within main()
  - cout does not know how to handle an object of class Date

Terminology

- CLASS: programmer-defined data type
- OBJECTS (INSTANCES): created from classes
  - Relation of objects to classes similar to relation of variables to C++ built-in data types

```
int a;  // a is a variable of type integer

Date a;  // a is an object of class Date
```

- INSTANTIATION: process of creating a new object
  - Creates new set of data members belonging to new object: Determines the object's state

Terminology (2)

- INTERFACE: part of class declaration section
  - Includes:
    - Class's public member function declarations
    - Supporting comments
- IMPLEMENTATION: consists of
  - Implementation section of class definition
    - Private member functions
    - Public member functions
  - Private data members from class declaration section

Terminology (3)

- Information hiding
  - Internal construction of class is not relevant to programmer who just wishes to use class
  - Implementation can and should be hidden from all users
    - Ensures that class is not altered or compromised in any way
  - Information needed by programmer to use class is provided by interface

Constructors

- A function that has same name as its class
  - Multiple constructors can be defined for a class
    - Each must be distinguishable by the number and types of its parameters
    - This is an example of function overloading
  - If no constructor function is written, compiler assigns default constructor
- Purpose: Initialize a new object's data members
  - May perform other tasks

Constructors (2)

- Format: Same name as class to which it belongs
  - Must have no return type (not even void)
- Default constructor: Does not require arguments when called
- Two cases of constructors
  - No parameters declared
    - As with compiler-supplied default constructor
  - Arguments have already been given default values in valid prototype statement:
    - Date (int = 7, int = 4, int = 2006)
    - Declaration Date a; initializes the a object with default values: 7, 4, 2006

Sample Class Declaration

```
class Date
{
  private:
    int month, day, year;
  public:
    void setDate(int, int, int);
    void showDate()
};
```

If No Constructor Has Been Included

- Compiler assigns a do-nothing default constructor equivalent to: Date (void) { }
- This constructor expects no parameters, and has an empty body

Constructor Format

```
classname::className (parameter list)
{
        function body
}
```

Use of Constructor in main()

```
int main()
{
  Date a;              // declare an object
  Date b;                        // declare an object
  Date c (4,1,2007);             // declare an object
  return 0;
}
```

Constructor Output

```
Created a new data object with data values 7, 4, 2006
Created a new data object with data values 7, 4, 2006
Created a new data object with data values 4, 1, 2007
```

Destructor

- Counterpart of constructor function
  - Has same name as constructor
  - Preceded with a tilde (~)
  - Date class constructor: ~Date()
- Default destructor
  - Do-nothing destructor provided by C++ compiler in absence of explicit destructor
- Can only be one destructor per class
  - Destructors take no values and return no arguments

Accessor Function

- Nonconstructor member function
- Accesses a class's private data members
- EXAMPLE: showDate() in the Date class
- Complete set of accessor functions should always be provided during class construction

Mutator Function

- Nonconstructor member function
- Changes an object's data members
- After object initialized by a constructor method
- Class can have multiple mutators
- EXAMPLE: setDate()

Arrays of Objects

- Declaring array of objects same as declaring array of C++ built-in type
- EXAMPLE: Date theDate[5];
  - Creates five objects named theDate[0] through theDate[4]
- Member functions for theDate array objects are called using:

```
objectName.functionName
```

Example

- Create class from which room type objects can be constructed
- Solution:
  - One type of object – rectangular shaped room
    - Represented by two double precision variables: length and width
  - Functions needed:
    - Constructor: to specify an actual length and width value when a room is instantiated
    - Accessor: to display room's length and width
    - Mutator: to change above values
    - Function to calculate room's floor area from its length and width values

Common Programming Errors

- Failing to terminate class declaration with semicolon
- Including return type with constructor's prototype
- Failing to include return type with other function's prototype
- Using same name for data member as for member function

Common Programming Errors (2)

- Defining more than one default constructor for class
- Forgetting to include class name and scope operator, ::, in the header line of all member functions defined in class implementation section
- All above errors will result in a compiler error message

Summary

- Class: Programmer defined data type
  - Class definition: Includes declaration and implementation sections
  - Class members: Variables and functions
- Objects: Same relation to class as variables do to C++ built-in data types
- private keyword: private class members can only be accessed by member functions

Summary (2)

- Class functions: May be written inline or defined in class implementation section
- Constructor function: Automatically called each time an object is declared
  - If none defined, compiler will supply default constructor
- Default constructor: Constructor that does not require arguments
  - One default constructor per class

Summary (3)

- Objects created using either C or C++ style of declaration:
  - C++: Date a, b, c(12, 25, 2002)
  - C: Date c = Date(12,25,2006)
- Constructors may be overloaded
- If constructor is defined, user-defined default constructor should be written
  - Compiler will not provide it
- Destructor: Called when object goes out of scope
  - Takes no arguments, returns no value