Classes in C++

Chapter 2

Objectives

- **Explain how the C++ class extends the data encapsulation facilities of C.**
- **Use member data to represent data encapsulated in a   class.**
- **Use member functions to implement a class' operations and provide access to its data.**
- **Use the *this* pointer to refer to the invoking object.**
- **Implement an abstract data type using C++ classes.**
- **Organize code for C++ classes into code files and header files.**
- **Gain experience through code walk-throughs and lab exercises.**

  - The example programs are in the **chapter directory.**
  - Labs located in Labs/Lab2

Data Encapsulation in C

- **An "object" is created by defining a data structure and associated operations (functions) in a file.**
    - The data structure being encapsulated is defined as static data, having file scope.
    - Operations on the data are provided by functions defined in the same file.
    - The data is encapsulated. No outside module can directly access it.

C-Style Data Encapsulation Example

- **We will start by examining "C-Style" encapsulation by implementing a stack.**
- **The application is in folder CStack.**
  - Review the code in CStack.cpp, CStack.h, and CTest.cpp.
  - Build and run.
- **Note that this is NOT an Abstract Data Type (ADT), because only one instance of such a stack can be used in a calling program without cloning this module.**

The C++ Class

- **Key concept of C++. (The original name of C++ was "C with Classes".)**
- **Derived from Simula (1967).**
- **Generalizes the C structure:**
    - **Functions as well as data.**
    - **Member access control**

Structures and Classes in C++

- **C++ provides user-defined data types *class* and *struct***
- **Both can have data members and function members**
- **Members are accessed using the "dot" notation:**
  - s.top;   // data access – top of stack
  - s.Pop();   // function access -- pop function

Member Access Control

- **Members can have *public* or *private* visibility.**
- ***class* hides the implementation details of its members and by default makes all the members private.**
- ***struct* does not hide the implementation details of its members and by default makes all the members public.**
- **Private members cannot be accessed from outside the class (except by "friend function" to be discussed later).**
- **Folder UDT has a partially complete application using a *struct* and a *class*. Review the code then run the application. The struct is complete, you will try to implement the class.**

this Pointer

- **Each class member function contains a pointer of its type named *this*.**
- **The *this* pointer contains the address of the class object through which the member function has been invoked.**
  - ****this* will refer to the invoking object itself.**
  - ***this-> is an equivalent way to refer to the invoking object.***

```
void IntStack::Push(int x){
    this->stack[this->top++] = x;
}
```

```
void IntStack::Push(int x){
    (*this).stack[(*this).top++] = x;
}
```

Code Organization

- **Class *definition* (or *specification*) is placed in a header file (*.h* extension):**

```
// IntStack.h
//
// Specification of Integer Stack class
class IntStack
{
...
}
```

Code Organization (continued)

- Clients of the class include this header file:

```
//       TstStack.cpp
//
#include "IntStack.h"

int main()
{
         ...
}
```

Code Organization (continued)

- Class *implementation* is placed in a code file (*.cpp* extension):

```
// IntStack.cpp
#include <iostream>
#include "IntStack.h"

void IntStack::InitStack()
{
...
}
```

Scope Resolution Operator

- ***Scope resolution operator*** **:: allows a member function to be defined outside the class definition.**
- **Scope resolution operator is needed by the compiler to identify for which class a member function is defined.**
- ***void FloatStack::Push(float x)*** **allows the "Push" function name to be reused as a member function of the *FloatStack* class.**
- **The scope resolution operator used by itself (not preceded by a class name) means that the following symbol is *global*.**

WALKTHOUGH: Use of a C++ Class

- Open the folder IntStack which contains a working C++ application.
- Examine the file intstack.h which contains the C++ class specification.
- Examine the file intstack.cpp which contains the implementation of integer stack class.
- Build and run the program.

Abstract Data Types

- **A class with a private representation of data and a public set of operations on the data is referred to as an abstract data type (ADT):**
  - The **IntStack** class has operations **Push**, **Pop**, **Print**.
  - The internal representation of the stack is hidden.
- **Abstract data types can be used in the same way as built-in data types.**
  - Later, we will see how other features of C++ such as operator overloading can be used to make the usage of abstract data types identical to usage of built-in data types.

Test Programs for C++ Classes

- **On advantage of object-oriented programming is the opportunity for thorough unit testing.**
  - Related functionality is encapsulated in a class, which is a logical unit for testing.
- **Write a test program for each class that exercises each member function.**
  - During development phase, test program can be exercised interactively.
  - During test phase, a test program can be exercised by scripts.

Summary

- In C++ a structure can have both data members and function members.
- Members can have public or private visibility.
- A class is a structure with default visibility of private.
- An abstract data type (ADT) is a class with private data and a public set of operations.
- Classes can be used to implement abstract data types. Multiple objects or instances of an ADT can be created.
- The *this* pointer is used to refer to the invoking object.
- Code is organized as a specification in a header file and implementation in a code file.
- You should write a test program for each class to exercise each member function.