Lecture 2

Structures and Classes
Learning Objectives

• Classes
  – Defining, member functions
  – Public and private members
  – Accessor and mutator functions
  – Structures vs. classes
Classes

• Similar to structures
  – Adds member FUNCTIONS
  – Not just member data

• Integral to object-oriented programming
  – Focus on objects
    • Object: Contains data and operations
    • In C++, variables of class type are objects
Class Definitions

• Defined similar to structures

• Example:

```cpp
class DayOfYear
{
public:
    void output();
    int month;
    int day;
};
```

• Notice only member function’s prototype
  – Function’s implementation is elsewhere
Declaring Objects

• Declared same as all variables
  – Predefined types, structure types

• Example:
  ```cpp
  DayOfYear today, birthday;
  • Declares two objects of class type DayOfYear
  ```

• Objects include:
  – Data
    • Members month, day
  – Operations (member functions)
    • output()
Class Member Access

• Members accessed same as structures
• Example:
  today.month
  today.day
  – And to access member function:
    today.output(); ← Invokes member function
Class Member Functions

• Must define or "implement" class member functions

• Like other function definitions
  – Can be after main() definition
  – Must specify class:
    void DayOfYear::output()
    {
      ...
    }
    • :: is scope resolution operator
    • Instructs compiler "what class" member is from
    • Item before :: called type qualifier
Class Member Functions Definition

• Notice output() member function’s definition (in next example)

• Refers to member data of class
  – No qualifiers

• Function used for all objects of the class
  – Will refer to "that object’s" data when invoked
  – Example:
    today.output();
    • Displays "today" object’s data
Complete Class Example:

**Display 6.3** Class With a Member Function (1 of 4)

```cpp
//Program to demonstrate a very simple example of a class.
//A better version of the class DayOfYear will be given in Display 6.4.
#include <iostream>
using namespace std;

class DayOfYear
{
public:
    void output();
    int month;
    int day;
};

int main()
{
    DayOfYear today, birthday;
    cout << "Enter today's date:\n";
    cout << "Enter month as a number: ";
    cin >> today.month;
    cout << "Enter the day of the month: ";
    cin >> today.day;
    cout << "Enter your birthday:\n";
    cout << "Enter month as a number: ";
    cin >> birthday.month;
    cout << "Enter the day of the month: ";
    cin >> birthday.day;
    (continued)
```
Complete Class Example:

Display 6.3 Class With a Member Function (2 of 4)

```cpp
Display 6.3  Class with a Member Function

25    cout << "Today's date is ";
26    today.output( );
27    cout << endl;
28    cout << "Your birthday is ";
29    birthday.output( );
30    cout << endl;
31    if (today.month == birthday.month && today.day == birthday.day)
32       cout << "Happy Birthday!\n";
33    else
34       cout << "Happy Unbirthday!\n";
35    return 0;
36 }
37 //Uses iostream:
38 void DayOfYear::output( )
39 {
40    switch (month)
41    {
42       case 1:
43          cout << "January "; break;
44       case 2:
45          cout << "February "; break;
46       case 3:
47          cout << "March "; break;
48       case 4:
49          cout << "April "; break;
```

Calls to the member function output

Member function definition
Complete Class Example:

**Display 6.3** Class With a Member Function (3 of 4)

```cpp
50     case 5:
51         cout << "May "; break;
52     case 6:
53         cout << "June "; break;
54     case 7:
55         cout << "July "; break;
56     case 8:
57         cout << "August "; break;
58     case 9:
59         cout << "September "; break;
60     case 10:
61         cout << "October "; break;
62     case 11:
63         cout << "November "; break;
64     case 12:
65         cout << "December "; break;
66     default:
67         cout << "Error in DayOfYear::output. Contact software vendor."
68             } }
69         cout << day;
70     } }
```
Complete Class Example:

**Display 6.3** Class With a Member Function (4 of 4)

---

**Sample Dialogue**

Enter today's date:
Enter month as a number: 10
Enter the day of the month: 15
Enter your birthday:
Enter month as a number: 2
Enter the day of the month: 21
Today's date is October 15
Your birthday is February 21
Happy Unbirthday!
Dot and Scope Resolution Operator

• Used to specify "of what thing" they are members

• Dot operator:
  – Specifies member of particular object

• Scope resolution operator:
  – Specifies what class the function definition comes from
A Class’s Place

• Class is full-fledged type!
  – Just like data types int, double, etc.

• Can have variables of a class type
  – We simply call them "objects"

• Can have parameters of a class type
  – Pass-by-value
  – Pass-by-reference

• Can use class type like any other type!
Encapsulation

• Any data type includes
  – Data (range of data)
  – Operations (that can be performed on data)

• Example:
  \( \text{int} \) data type has:
  Data: -2147483648 to 2147483647 (for 32 bit int)
  Operations: +,-,*,/,% , logical, etc.

• Same with classes
  – But WE specify data, and the operations to be allowed on our data!
Abstract Data Types

• "Abstract"
  – Programmers don’t know details

• Abbreviated "ADT"
  – Collection of data values together with set of basic operations defined for the values

• ADT’s often "language-independent"
  – We implement ADT’s in C++ with classes
    • C++ class "defines" the ADT
  – Other languages implement ADT’s as well
More Encapsulation

- Encapsulation
  - Means "bringing together as one"

- Declare a class → get an object

- Object is "encapsulation" of
  - Data values
  - Operations on the data (member functions)
Principles of OOP

• Information Hiding
  – Details of how operations work not known to "user" of class

• Data Abstraction
  – Details of how data is manipulated within ADT/class not known to user

• Encapsulation
  – Bring together data and operations, but keep "details" hidden
Public and Private Members

• Data in class almost always designated private in definition!
  – Upholds principles of OOP
  – Hide data from user
  – Allow manipulation only via operations
    • Which are member functions

• Public items (usually member functions) are "user-accessible"
Public and Private Example

• Modify previous example:
  ```cpp
class DayOfYear
{
    public:
      void input();
      void output();
    private:
      int month;
      int day;
};
```
• Data now private
• Objects have no direct access
Public and Private Example 2

• Given previous example

• Declare object:
  DayOfYear today;

• Object *today* can ONLY access
  public members
  – cin >> today.month;  // NOT ALLOWED!
  – cout << today.day;  // NOT ALLOWED!
  – Must instead call public operations:
    • today.input();
    • today.output();
Public and Private Style

• Can mix & match public & private

• More typically place public first
  – Allows easy viewing of portions that can be USED by programmers using the class
  – Private data is "hidden", so irrelevant to users

• Outside of class definition, cannot change (or even access) private data
Accessor and Mutator Functions

• Object needs to "do something" with its data

• Call accessor member functions
  – Allow object to read data
  – Also called "get member functions"
  – Simple retrieval of member data

• Mutator member functions
  – Allow object to change data
  – Manipulated based on application
Separate Interface and Implementation

• User of class need not see details of how class is implemented
  – Principle of OOP $\rightarrow$ encapsulation

• User only needs "rules"
  – Called "interface" for the class
    • In C++ $\rightarrow$ public member functions and associated comments

• Implementation of class hidden
  – Member function definitions elsewhere
  – User need not see them
Structures versus Classes

• Structures
  – Typically all members public
  – No member functions

• Classes
  – Typically all data members private
  – Interface member functions public

• Technically, same
  – Perceptionally, very different mechanisms
Thinking Objects

• Focus for programming changes
  – Before → algorithms center stage
  – OOP → data is focus

• Algorithms still exist
  – They simply focus on their data
  – Are "made" to "fit" the data

• Designing software solution
  – Define variety of objects and how they interact
Summary 1

• Structure is collection of different types

• Class used to combine data and functions into single unit -> object

• Member variables and member functions
   – Can be public → accessed outside class
   – Can be private → accessed only in a member function’s definition

• Class and structure types can be formal parameters to functions
Summary 2

- C++ class definition
  - Should separate two key parts
    - Interface: what user needs
    - Implementation: details of how class works