CHAPTER 6
Structures and Classes

LEARNING OBJECTIVES

- Structures
  - Structure types
  - Structures as function arguments
  - Initializing structures
- Classes
  - Defining, member functions
  - Public and private members
  - Accessor and mutator functions
  - Structures vs. classes

STRUCTURES

- 2nd aggregate data type: **struct**
- Recall: aggregate meaning "grouping"
  - Recall **array**: collection of values of **same** type
  - **Structure**: collection of values of **different** types
- Treated as a single item, like arrays
- Major difference: Must first "define" struct
  - Prior to declaring any variables

STRUCTURE TYPES

- Define struct globally (typically)
- No memory is allocated
  - Just a “placeholder” for what our struct will "look like"
- Definition:

```
struct CDAccountV1   // Name of new struct "type"
{
        double balance;   // member names
        double interestRate;
        int term;
};
```

DECLARE STRUCTURE VARIABLE

- With structure type defined, now declare variables of this new type:

CDAccountV1 account;

  - Just like declaring simple types
  - Variable *account* now of type CDAccountV1
  - It contains "member values"
    - Each of the struct "parts"

ACCESSING STRUCTURE MEMBERS

- **Dot Operator** to access members
  - account.balance
  - account.interestRate
  - account.term
- Called "member variables"
  - The "parts" of the structure variable
  - Different structs can have same name member variables
    - No conflicts

STRUCTURE EXAMPLE: DISPLAY 6.1 A STRUCTURE DEFINITION (1 OF 3)

Display 6.1 A Structure Definition

```
1   //Program to demonstrate the CDAccountV1 structure type.
2   #include <iostream>
3   using namespace std;

4   //Structure for a bank certificate of deposit:
5   struct CDAccountV1
6   {
7       double balance;
8       double interestRate;
9       int term;//months until maturity
10  };


11  void getData(CDAccountV1& theAccount);
12  //Postcondition: theAccount.balance, theAccount.interestRate, and
13  //theAccount.term have been given values that the user entered at the keyboar
```

An improved version of this structure will be given later in this chapter.

```
14  int main( )
15  {
16      CDAccountV1 account;
17      getData(account);

18      double rateFraction, interest;
19      rateFraction = account.interestRate/100.0;
20      interest = account.balance*(rateFraction*(account.term/12.0));
21      account.balance = account.balance + interest;

22      cout.setf(ios::fixed);
23      cout.setf(ios::showpoint);
24      cout.precision(2);
25      cout << "When your CD matures in "
26           << account.term << " months,\n"
27           << "it will have a balance of $"
28           << account.balance << endl;

29      return 0;
30  }
```

(continued)

Display 6.1 A Structure Definition

```
31  //Uses iostream:
32  void getData(CDAccountV1& theAccount)
33  {
34      cout << "Enter account balance: $";
35      cin >> theAccount.balance;
36      cout << "Enter account interest rate: ";
37      cin >> theAccount.interestRate;
38      cout << "Enter the number of months until maturity: ";
39      cin >> theAccount.term;
40  }
```

SAMPLE DIALOGUE

Enter account balance: **$100.00**
Enter account interest rate: **10.0**
Enter the number of months until maturity: **6**
When your CD matures in 6 months,
it will have a balance of $105.00

STRUCTURE PITFALL

- Semicolon after structure definition
  - ; MUST exist:

```
struct WeatherData
{
   double temperature;
   double windVelocity;
};
```

← REQUIRED semicolon!

  - Required since you "can" declare structure variables in this location

STRUCTURE ASSIGNMENTS

- Given **structure** named CropYield
- Declare two structure variables:
  CropYield apples, oranges;
  - Both are variables of "struct type CropYield"
  - Simple assignments are legal:

    apples = oranges;
    - Simply copies each member variable from apples into member variables from oranges

STRUCTURES AS FUNCTION **ARGUMENTS**

- Passed like any simple data type
  - **Pass-by-value**
  - **Pass-by-reference**
  - Or combination
- Can also be **returned** by function
  - Return-type is structure type
  - Return statement in function definition sends structure variable back to caller

INITIALIZING STRUCTURES

- Can initialize at declaration
  - Example:

```
struct Date
{
        int month;
        int day;
        int year;
};
Date dueDate = {12, 31, 2003};
```

  - Declaration provides initial data to all three member variables

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:

```
class DayOfYear    // name of new class type
{
public:
        void output();          // member function!
        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:

  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

DISPLAY 6.3 CLASS WITH A MEMBER FUNCTION

```
1)  #include <iostream>
2)  using namespace std;

3)  class DayOfYear
4)  {
5)  public:
6)      int month;
7)      int day;
8)      void output( );
9)  };
```

```
1)   int main( )
2)   {
3)       DayOfYear today, birthday;
4)       cout << "Enter today's date:\n"; cout << "Enter month as a number: ";
5)       cin >> today.month;
6)       cout << "Enter the day of the month: ";    cin >> today.day;
7)       cout << "Enter your birthday:\n";cout << "Enter month as a number: ";
8)       cin >> birthday.month;
9)       cout << "Enter the day of the month: ";    cin >> birthday.day;

10)      cout << "Today's date is ";    today.output( );    cout << endl;
11)      cout << "Your birthday is ";    birthday.output( );    cout << endl;

12)      if (today.month == birthday.month && today.day == birthday.day)
13)          cout << "Happy Birthday!\n";
14)      else        cout << "Happy Unbirthday!\n";

15)      return 0;
16)  }
```

```
1)  void DayOfYear::output( )
2)  {
3)      switch (month)
4)      {
5)          case 1:
6)              cout << "January "; break;
7)          case 2:
8)              cout << "February ";   break;
9)          case 3:
10)             cout << "March "; break;
11)         case 4:
12)             cout << "April "; break;
13)         case 5:
14)             cout << "May "; break;
15)         case 6:
16)             cout << "June "; break;
17)         case 7:
18)             cout << "July "; break;
19)         case 8:
20)             cout << "August "; break;
21)         case 9:
22)             cout << "September ";
                break;
23)         case 10:
24)             cout << "October ";
                break;
25)         case 11:
26)             cout << "November ";
                break;
27)         case 12:
28)             cout << "December ";
                break;
29)         default:
30)             cout << "Error in
                DayOfYear::output. Contact
                software vendor.";
31)     }
32)     cout << day;
33) }
```

Display 6.3 Class with a Member Function

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!
```

- Do you have any idea to write statements in a more efficient way for switch block in previous example?
  - E.g. in three statements

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:
  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:

```
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

```
1)  class DayOfYear
2)  {
3)  public:
4)      void input( );
5)      void output( );
6)      void set(int newMonth, int newDay);
7)      //Precondition: newMonth and newDay form a possible date.

8)      void set(int newMonth);
9)      //Precondition: 1 <= newMonth <= 12
10)     //Postcondition: The date is set to the first day of the given month.
11)
12)     int getMonthNumber( ); //Returns 1 for Jan, 2 for Feb, etc.
13)     int getDay( );
14) private:
15)     int month;
16)     int day;
17) };
```

```
1)   int main( )
2)   {
3)       DayOfYear today, bachBirthday;
4)       cout << "Enter today's date:\n";    today.input( );
5)       cout << "Today's date is ";    today.output( );    cout << endl;

6)       bachBirthday.set(3, 21);
7)       cout << "J. S. Bach's birthday is ";
8)       bachBirthday.output( );    cout << endl;

9)       if ( today.getMonthNumber( ) ==
                    bachBirthday.getMonthNumber( ) &&
10)                 today.getDay( ) == bachBirthday.getDay( ) )
11)          cout << "Happy Birthday Johann Sebastian!\n";
12)      else
13)          cout << "Happy Unbirthday Johann Sebastian!\n";

14)      return 0;
15)  }
```

```
1)  void DayOfYear::input( )
2)  {
3)      cout << "Enter the month as a number: ";
4)      cin >> month;
5)      cout << "Enter the day of the month: ";
6)      cin >> day;
7)      if ((month < 1)  | |  (month > 12)  | |  (day < 1)
                             | |  (day > 31))
8)      {
9)          cout << "Illegal date! Program aborted.\n";
10)         exit(1);
11)     }
12)}
```

Private members may be used in member function definitions (but not elsewhere).

```
1)  void DayOfYear::output( )
2)  {
3)      switch (month)
4)      {
5)          case 1:
6)              cout << "January "; break;
7)          case 2:
8)              cout << "February "; break;
9)              …
10)         case 12:
11)             cout << "December "; break;
12)         default:
13)             cout << "Error in DayOfYear::output. Contact
                        software vendor.";
14)     }
15)     cout << day;
16) }
```

```
1)  void DayOfYear::set(int newMonth, int newDay)
2)  {
3)      if ((newMonth >= 1) && (newMonth <= 12))
4)          month = newMonth;
5)      else
6)      {
7)          cout << "Illegal month value! Program aborted.\n";
8)          exit(1);
9)      }
10)     if ((newDay >= 1) && (newDay <= 31))
11)         day = newDay;
12)     else
13)     {
14)         cout << "Illegal day value! Program aborted.\n";
15)         exit(1);
16)     }
17) }
```

```
1)  void DayOfYear::set(int newMonth)
2)  {
3)      if ((newMonth >= 1) && (newMonth <= 12))
4)          month = newMonth;
5)      else
6)      {
7)          cout << "Illegal month value! Program aborted.\n";
8)          exit(1);
9)      }
10)     day = 1;
11)}
```

```
1)  int DayOfYear::getMonthNumber( )
2)  {
3)      return month;
4)  }

5)  int DayOfYear::getDay( )
6)  {
7)      return day;
8)  }
```

SEPARATE INTERFACE AND IMPLEMENTATION

- User of class need not see details of how class is implemented
  - Principle of OOP → encapsulation
- User only needs "rules"
  - Called "interface" for the class
    - In C++ → 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

STRUCTURE VS. CLASS IN C++

- Members of a class are private by default and members of struct are **public** by default

```
#include <stdio.h>

struct Test {
    int x; // x is public
};
int main()
{
  Test t;
  t.x = 20; // works fine
  getchar();
  return 0;
}
```

```
1) #include <stdio.h>
2) class Test {
3)     int x; // x is private
4) };
5) int main()
6) {
7)   Test t;
8)   t.x = 20; // compiler error
9)   getchar();
10)  return 0;
11)}
```

"NEW" ALWAYS RETURNS POINTERS TO DISTINCT OBJECTS

```
1)  #include<iostream>
2)  using namespace std;
3)
4)  class Empty { };
5)
6)  int main()
7)  {
8)      Empty* p1 = new Empty;
9)      Empty* p2 = new Empty;
10)
11)     if (p1 == p2)
12)         cout << "impossible " << endl;
13)     else
14)         cout << "Fine " << endl;
15)
16)     return 0;
17) }
```

```
1)   #include <iostream>
2)    using namespace std;

3)   int main(void) {
4)      Box Box1(3.3, 1.2, 1.5);    // Declare box1
5)      Box Box2(8.5, 6.0, 2.0);    // Declare box2
6)      Box *ptrBox;                // Declare pointer to a class.
7)      // Save the address of first object
8)      ptrBox = &Box1;
9)      // Now try to access a member using member access operator
10)     cout << "Volume of Box1: " << ptrBox->Volume() << endl;
11)     // Save the address of second object
12)     ptrBox = &Box2;
13)     // Now try to access a member using member access operator
14)     cout << "Volume of Box2: " << ptrBox->Volume() << endl;
15)     return 0;
16)  }
```

```
1)   class Box {
2)     public:
3)       // Constructor definition
4)       Box(double l = 2.0, double b = 2.0, double h = 2.0)
5)       {
6)          cout <<"Constructor called." << endl;
7)          length = l;
8)          breadth = b;
9)          height = h;
10)      }
11)
12)      double Volume() {
13)         return length * breadth * height;
14)      }
15)    private:
16)      double length;     // Length of a box
17)      double breadth;    // Breadth of a box
18)      double height;     // Height of a box
19)  };
```

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