

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

- 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

- 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;
};

CDAccountV1 account;
```

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"

Structure Types

```
struct [structure tag]
```

```
{
```

```
member definition;
```

```
...
```

```
member definition;
```

```
} [one or more structure variables];
```



```
struct Books
```

```
{
```

```
char title [50];
```

```
char author [50];
```

```
char subject [100];
```

```
int book_id;
```

```
} book;
```

Initializing Structures

- Can initialize at declaration

- Example:

```
struct Date
{
    int month;
    int day;
    int year;
};
```

```
Date dueDate = {12, 31, 2018};
```

- Declaration provides initial data to all three member variables

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.

Structure Example:

Display 6.1 A Structure Definition (2 of 3)

```
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)

Structure Example:

Display 6.1 A Structure Definition (3 of 3)

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

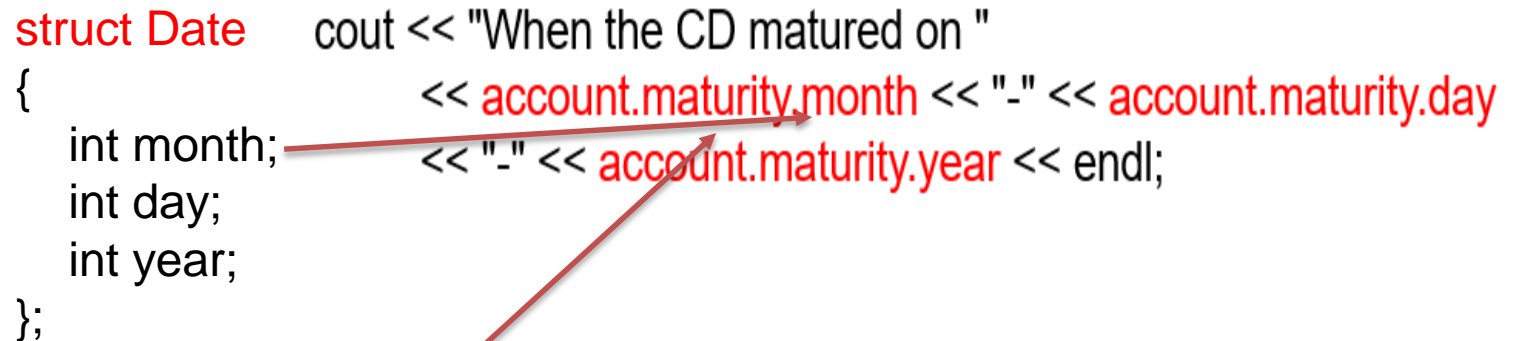
Structure Assignments

- 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 variables

Hierarchical Structures

- A member of a structure is a smaller structure

```
struct Date    cout << "When the CD matured on "  
{              << account.maturity.month << "-" << account.maturity.day  
    int month; << "-" << account.maturity.year << endl;  
    int day;  
    int year;  
};
```



```
//Improved structure for a bank certificate of deposit:  
struct CDAccount  
{  
    double initialBalance;  
    double interestRate;  
    int term;           //months until maturity  
    Date maturity;  
    double balanceAtMaturity;  
};
```

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:      ← access specifier
        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

Complete Class Example:

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

Display 6.3 Class with a Member Function

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

5  class DayOfYear
6  {
7  public:
8      void output( );
9      int month;
10     int day;
11 };

12 int main( )
13 {
14     DayOfYear today, birthday;
15     cout << "Enter today's date:\n";
16     cout << "Enter month as a number: ";
17     cin >> today.month;
18     cout << "Enter the day of the month: ";
19     cin >> today.day;
20     cout << "Enter your birthday:\n";
21     cout << "Enter month as a number: ";
22     cin >> birthday.month;
23     cout << "Enter the day of the month: ";
24     cin >> birthday.day;
```

*Normally, member variables are **private** and not **public**, as in this example. This is discussed a bit later in this chapter.*

Member function declaration

(continued)

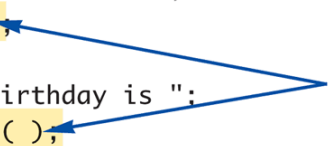
Complete Class Example:

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


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)

```
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
70     cout << day;
71 }
```

Complete Class Example:

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

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!

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

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