2 Structures

- A **structure** can be viewed as an object
  - Contains no member functions
    (The structures used here have no member functions)
  - Contains multiple values of possibly different types
    - The multiple values are logically related as a single item
    - Example:
      A bank Certificate of Deposit (CD) has the following values:
      - a balance
      - an interest rate
      - a term (months to maturity)

2.1 The CD Definition

- The Certificate of Deposit structure can be defined as

```
struct CDAccount
{
        double balance;
        double interest_rate;
        int term; //months to maturity
};  // Remember this semicolon!
```

- Keyword **struct** begins a structure definition
- **CDAccount** is the **structure tag** or the structure's type
- **Member names** are identifiers declared in the braces

2.2 Using the Structure

- Structure definition is generally placed outside any function definition
  - This makes the structure type available to all code that follows the structure definition
- To declare two variables of type CDAccount:

```
CDAccount my_account, your_account;
```

  - My_account and your_account contain distinct member variables balance, interest_rate, and term

2.3 The Structure Value

- The **Structure Value**
  - Consists of the values of the member variables
- The value of an object of type **CDAccount**
  - Consists of the values of the member variables
    - balance
    - interest_rate
    - term

2.4 Specifying Member Variables

- Member variables are specific to the structure variable in which they are declared
  - Syntax to specify a member variable:
    Structure_Variable_Name . Member_Variable_Name
  - Given the declaration: CDAccount my_account, your_account;
    - Use the **dot operator** to specify a member variable
      - my_account.balance
      - my_account.interest_rate
      - my_account.term

2.5 Using Member Variables

- Member variables can be used just as any other variable of the same type
  - ```
    CDAccount my_account, your_account;
    my_account.balance = 1000;
    your_account.balance = 2500;
    ```
    - Notice that **my_account.balance** and **your_account.balance** are different variables!
  - ```
    my_account.balance = my_account.balance + interest;
    ```

2.6 Assignment and Structures

- The assignment operator can be used to assign values to structure types
- Using the CDAccount structure again:
  ```
  CDAccount my_account, your_account;
  my_account.balance = 1000.00;
  my_account.interest_rate = 5.1;
  my_account.term = 12;
  your_account = my_account;
  ```
  - Assigns all member variables in **your_account** the corresponding values in **my_account**

2.6.1 sample21.cpp

```
#include <iostream>
using namespace std;

struct Vector2d{
  double x;
  double y;
};

int main (int argc, char *argv[]) {
  Vector2d a;
  a.x = 1.0;
  a.y = 2.0;
  Vector2d b = a;
  cout << "b=(" << b.x << "," << b.y << ")" << endl;

  return 0;
}
```

2.7 Duplicate Names

- Member variable names duplicated between structure types are not a problem.

```
struct FertilizerStock
{
        double quantity;
        double nitrogen_content;
};

FertilizerStock  super_grow;
```

```
struct CropYield
{
        int quantity;
        double size;
};

CropYield  apples;
```

- **super_grow.quantity** and **apples.quantity** are different variables stored in different locations

2.8 Structures as Arguments

- Structures can be arguments in function calls
  - The formal parameter can be call-by-value
  - The formal parameter can be call-by-reference
- Example:

```
void out_data(CDAccount the_account); // Call-by-value
```

Uses the structure type **CDAccount** we saw earlier as the type for a call-by-value parameter

```
void get_data(CDAccount& the_account); // Call-by-reference
```

Uses the structure type **CDAccount** we saw earlier as the type for a call-by-reference parameter

2.9 Structures as Return Types

- Structures can be the type of a value returned by a function
- Example:

```
CDAccount shrink_wrap(double the_balance, double the_rate, int the_term)
{
        CDAccount temp;
        temp.balance = the_balance;
        temp.interest_rate = the_rate;
        temp.term = the_term;
        return temp;
}
```

2.10 Using Function shrink_wrap

- **shrink_wrap** builds a complete structure value in temp, which is returned by the function
- We can use **shrink_wrap** to give a variable of type **CDAccount** a value in this way:

```
CDAccount new_account;
new_account = shrink_wrap(1000.00, 5.1, 11);
```

2.11 Hierarchical Structures

- Structures can contain member variables that are also structures

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

```
struct PersonInfo{
        double height;
        int weight;
        Date birthday;
};
```

- **struct PersonInfo** contains a Date structure

2.11.1 Using PersonInfo

- A variable of type PersonInfo is declared by
  ```
  PersonInfo person1;
  ```
- To display the birth year of person1, first access the birthday member of person1
  cout << person1.birthday...
- But we want the year, so we now specify the year member of the birthday member
  ```
  cout << person1.birthday.year;
  ```

2.12 Initializing Classes

- A structure can be initialized when declared
- Example:
  ```
  struct Date
  {
          int month;
          int day;
          int year;
  };
  ```
- Can be initialized in this way
  ```
  Date due_date = {12, 31, 2004};
  ```

2.13 Sample22.cpp

```
#include <iostream>
#include <cmath>

using namespace std;

struct Vector2d{
    double x;
    double y;
};

Vector2d rotate(double a, Vector2d p){
    Vector2d q;
    q.x = p.x * cos(a) - p.y * sin(a);
    q.y = p.x * sin(a) + p.y * cos(a);

    return q;
}
```

```cpp
int main (int argc, char *argv[]) {
    Vector2d x1 = {1.0, 0.0};

    cout << "(" << x1.x << "," << x1.y << ")" << endl;

    x1 = rotate(M_PI / 4.0, x1);
    cout << "(" << x1.x << "," << x1.y << ")" << endl;

    x1 = rotate(M_PI / 4.0, x1);
    cout << "(" << x1.x << "," << x1.y << ")" << endl;

    x1 = rotate(M_PI / 4.0, x1);
    cout << "(" << x1.x << "," << x1.y << ")" << endl;

    x1 = rotate(M_PI / 4.0, x1);
    cout << "(" << x1.x << "," << x1.y << ")" << endl;

    x1 = rotate(M_PI / 4.0, x1);
    cout << "(" << x1.x << "," << x1.y << ")" << endl;

    x1 = rotate(M_PI / 4.0, x1);
    cout << "(" << x1.x << "," << x1.y << ")" << endl;

    x1 = rotate(M_PI / 4.0, x1);
    cout << "(" << x1.x << "," << x1.y << ")" << endl;

    x1 = rotate(M_PI / 4.0, x1);
    cout << "(" << x1.x << "," << x1.y << ")" << endl;

    return 0;
}
```

Compare with “**sample12.cpp**”.