ITT DUBLIN
Institute of Technology Tallaght
Institiúid Teicneolaíochta Tamhlacht

Classes and Structures

Structures - motivation

- Think about our lab problem to pass the price of 3 items to a function

```
displayPrices(ostream&, double price1,
                        double price2, double price3);
```

- To use this function, the programmer would need a description of what was required for each argument.
  - Can we come up with descriptions that give sufficient information?
    - e.g say in what order the prices are passed in? Maybe widget, then toggle, then gadget?
- Could we improve this by using an array?
  - Would we need the same amount of description to the function?
- Could we make the function more flexible by using more arguments?
  - so that the names/order of the 3 items could be specified in the function call
  - Maybe pass in 2 arrays – one of 3 strings (names) and one of 3 doubles (prices)
- Even better if we could use an array of 2-element-things, where the first element was a name and the second was a price
  - Can we define a type which is a "two-element thing" where the two elements are themselves of different types?

The struct data type

- The struct is very like a class, but used in more ‘light-weight’ situations, were we don’t want all the overhead of a class
  - We want to store a few data-members of different types …
  - … but no constructors or methods to muddy the waters

```
struct Product
{
    string name;
    double price;
};
```

Definition of the data type Product says it is a structure, and each variable of this type has 2 data members, a name and a price

Don’t forget the semi-colon here

```
Product prod1;
prod1.name = “Toggle”;
prod1.price = 5.5;
```

prod1 is a variable of type Product

By default, the data members of a struct can be directly accessed (‘public access’) anywhere in our code

Improved `displayPrices` implementations

١. Pass 3 separate Product items

- A separate function would still be needed if there were 4 different products!

```
displayPrices(Product prod1, Product prod2, Product prod3)
{
    cout << setw(10) << "Name" << setw(10) << "Price" << endl;
    cout << setw(10) << prod1.name << setw(10) << prod1.price
                                                    << endl;
    cout << setw(10) << prod2.name << setw(10) << prod2.price
                                                    << endl;
    cout << setw(10) << prod2.name << setw(10) << prod2.price
                                                    << endl;
}
```

٣. Pass an array of Product items

- Now any number of products can be handled by the same function!

```
displayPrices(const Product prods[] , int numProds)
{
    cout << setw(10) << "Name" << setw(10) << "Price" << endl;
    for (int i = 0; i < numProds, i++)
            cout << setw(10) << prods[i].name << setw(10)
                                        << prods[i].price << endl;
}
```

Classes - intro

- The ideas about classes are similar to those in Java
- But syntax and some fundamentals are different
- They are similar in operation to the C++ structure
  - Add member FUNCTIONS as well as member data
- A class definition

```
class DayOfYear ← name of new class type
{
public: ← the access specifier
        void output(); ← member function!
        int month;
        int day;
};
```

Notice the semi-colon

- Notice this only has the prototype (declaration) of the member function/s
  - The implementation of the member functions is elsewhere
- 'public' keyword says 'everything declared after this is public in the object, until you reach a different keyword.
  - It is an '*access specifier*'

Declaring and using objects

- Objects are variables of a class type
  - Can use class type like any other type!
  - Just like data types int, double, etc.

- Declared in the same way as all variables
  - `DayOfYear today, birthday;`
    - Declares two objects of class type `DayOfYear`
    - When an object is declared like this, space is allocated in memory for the whole object
    - This is done statically (at compile time)
    - the variable name is bound to the actual memory for the object, not a *reference* to the object as it is in java

- Can have function parameters of a class type
  - Pass-by-value: local copy of the object is used within the function
  - Pass-by-reference: refers back to the object variable passed in

- Can return a class type from a function

- Members are accessed in same way as in structures

```
today.month
today.day
```

  - And to access member function:

```
today.output();
```

Can use the dot notation to directly access these outside the class only because they have been declared 'public'

definition (implementation) of member functions

- output() member function's definition

```
void DayOfYear::output()
{
   switch (month)
   {
      case 1:
         cout << "January";
         break;
      case 2:
      // etc
   }
   cout << day;
}
```

- Refers to data members `month` and `day` without qualifiers
- values are known because the method is called on a particular object

```
DayOfYear today;
today.output()
```

Displays the object data in the variable '`today`'

- Dot and Scope Resolution Operator
  - Used to specify "of what thing" they are members

Dot operator:

Specifies member of particular object

```
thisDate.day
thisDate.output();
```

Scope resolution operator:

Specifies what class the function definition comes from

```
DayOfYear::output()
{
  //implementation
}
```

Complete Class Example: 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: 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: 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: 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 public member functions

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

'private' access specifier

- If no access specifier is given, all members are by default private until another access specifier is reached
- Public items (usually just member functions) are "user-accessible"
- Accessor member functions
  - Also called "'*get*' member functions"
  - Request the object for information
  - Simple retrieval of member data
  - They *return* the data
    - they do *not* print it to a screen or file
- Mutator member functions
  - simple 'set' operation on member data
  - Request the object to change the value in a data member (and pass the new value)
    - But the set function may also check that the new value is reasonable!

These are 2 special categories of member functions

Encapsulation

- Encapsulation principle:
  - Separate how class is used by programmer (its interface) from the details of its implementation
  - This is a basic principle of OOP
  - The implementation should be used as a 'black box' (as should the implementation of a function)
  - Change to implementation → NO impact on the code which uses the class
- Rules to ensure separation:
  - All member variables should be private
  - Member functions should be public if to be called from outside the class
    - But private if only called from code in other methods of the class.
  - The interface for the class shows
    - the data members
    - *Only the declarations* of member functions
  - The implementation of member functions is provided outside of the class interface
    - This 'class implementation' should be unavailable to users of class
    - All they need to know is the interface of the class

Recap: Structure and Class data types

- Structure is a collection of data
  - can be of different types
  - usually used without member functions
  - And by default, all member data is public
- Class used to combine data and functions into single unit
  - objects are variables of the class type
- Member variables and member functions
  - Can be public → accessed outside class
  - Can be private → accessed only within the implementation of a member function
  - By default, all data members and functions are private
- Class and structure types can be formal parameters to functions
  - And passed by value or by reference
- C++ class definition
  - Must separate two key parts
    - Interface: what user needs
    - Implementation: details of how class works