Abstract Data Types in C++

Introduction to Classes

Mang Juan IT learns

Abstract Data Types in C++

- Software development is labor-intensive
- Productivity in software cannot be dramatically increased like productivity in manufacturing can
- Reusable code components contribute to productivity in software development
- Object-oriented approach to software development supports the creation and reuse of software components

Abstract Data Types in C++ (continued)

- **Abstract data type:** a user-defined data type
- Each data type consists of both a type and specific operational capabilities for the type
- **Data type:** defined as the combination of data and the operations that can be performed on the data
- Built-in data types in C++ have operational capabilities

Abstract Data Types in C++ (continued)

Operational Capabilities for Built-in Data Types

Capability	Example
Define one or more variables of the data type	`int a, b;`
Initialize a variable at definition	`int a = 5;`
Assign a value to a variable	`a = 10;`
Assign one variable's value to another variable	`a = b;`
Perform mathematical operations	`a + b`
Perform relational operations	`a > b`
Convert from one data type to another	`a = int (7.2);`

Abstract Data Types in C++ (continued)

- When an abstract data type has been created, programmers can use it without knowing how it is implemented internally

Example: a date object might include operations for various display formats, comparing two dates, extracting the month, day, and year numbers, etc.

- **Class**: an abstract data type that defines both data and functions

Abstract Data Types in C++ (continued)

- A class usually contains two sections:
  - **Declarations section:** declares the data types and the function prototypes
  - **Implementation section:** defines the functions
- **Method:** a function contained in a class
- **Class members:** the variables and functions listed in the class declaration section
- **Data members:** the variables in a class; also called instance variables

Abstract Data Types in C++ (continued)

- **Member functions:** the functions in a class
- Class declaration section begins with the keyword **`class`** and the class name

Syntax:

```
class Name
{
  private:
    a list of variable declarations
  public:
    a list of function prototypes
};
```

Abstract Data Types in C++ (continued)

- Class name is usually capitalized as a convention
- **Access specifiers** define the access rights to variables and functions in the class:
  - **`private`**: class members may only be accessed by the class's own functions
  - **`public`**: class members may be called by any objects and functions outside the class

Abstract Data Types in C++ (continued)

- **`private`** designation enforces data security by requiring access through class functions; called **data hiding**
- Generally, all class functions are declared as **`public`**, because they are used to manipulate the class's data
- **Constructor**: function used to initialize class data members with values; cannot have a return value

Abstract Data Types in C++ (continued)

```
// class declaration

class Date
{
  private:
    int month;
    int day;
    int year;
  public:
    Date(int = 7, int = 4, int = 2005);  // constructor
    void setDate(int, int, int);         // member function to copy a date
    void showDate();                     // member function to display a date
};

// implementation section

Date::Date(int mm, int dd, int yyyy)
{
  month = mm;
  day = dd;
  year = yyyy;
}
void Date::setDate(int mm, int dd, int yyyy)
{
  month = mm;
  day = dd;
  year = yyyy;

  return;
}
```

Abstract Data Types in C++ (continued)

```
void Date::showDate()
{
  cout << "The date is ";
  cout << setfill('0')
       << setw(2) << month << '/'
       << setw(2) << day << '/'
       << setw(2) << year % 100; // extract the last 2 year digits
  cout << endl;

  return;
}
int main()
{
  Date a, b, c(4,1,2000);  // declare 3 objects

  b.setDate(12,25,2006);  // assign values to b's data members
  a.showDate();           // display object a's values
  b.showDate();           // display object b's values
  c.showDate();           // display object c's values

  return 0;
}
```

```
The date is 07/04/05
The date is 12/25/06
The date is 04/01/00
```

Abstract Data Types in C++ (continued)

```
// class declaration

class Date
{
  private:
    int month;
    int day;
    int year;
  public:
    Date(int = 7, int = 4, int = 2005);  // constructor
    void setDate(int, int, int);         // member function to copy a date
    void showDate();                     // member function to display a date
};

// implementation section

Date::Date(int mm, int dd, int yyyy)
{
  month = mm;
  day = dd;
  year = yyyy;
}
void Date::setDate(int mm, int dd, int yyyy)
{
  month = mm;
  day = dd;
  year = yyyy;

  return;
}
```

Abstract Data Types in C++ (continued)

```
void Date::showDate()
{
  cout << "The date is ";
  cout << setfill('0')
       << setw(2) << month << '/'
       << setw(2) << day << '/'
       << setw(2) << year % 100; // extract the last 2 year digits
  cout << endl;

  return;
}
int main()
{
  Date a, b, c(4,1,2000);  // declare 3 objects

  b.setDate(12,25,2006);  // assign values to b's data members
  a.showDate();           // display object a's values
  b.showDate();           // display object b's values
  c.showDate();           // display object c's values

  return 0;
}
```

```
The date is 07/04/05
The date is 12/25/06
The date is 04/01/00
```

Abstract Data Types in C++ (continued)

```
// class declaration

class Date
{
  private:
    int month;
    int day;
    int year;
  public:
    Date(int = 7, int = 4, int = 2005);  // constructor
    void setDate(int, int, int);         // member function to copy a date
    void showDate();                     // member function to display a date
};

// implementation section

Date::Date(int mm, int dd, int yyyy)
{
  month = mm;
  day = dd;
  year = yyyy;
}
void Date::setDate(int mm, int dd, int yyyy)
{
  month = mm;
  day = dd;
  year = yyyy;

  return;
}
```

Abstract Data Types in C++ (continued)

- Constructor function has the same name as the class name
- Member functions are declared with the class name

Syntax:

```
returnType  className::functionName(parameter list)
{
    function body
}
```

Abstract Data Types in C++ (continued)

```
// class declaration

class Date
{
  private:
    int month;
    int day;
    int year;
  public:
    Date(int = 7, int = 4, int = 2005);  // constructor
    void setDate(int, int, int);         // member function to copy a date
    void showDate();                     // member function to display a date
};

// implementation section

Date::Date(int mm, int dd, int yyyy)
{
  month = mm;
  day = dd;
  year = yyyy;
}
void Date::setDate(int mm, int dd, int yyyy)
{
  month = mm;
  day = dd;
  year = yyyy;

  return;
}
```

Abstract Data Types in C++ (continued)

- A class simply defines a data type; it does not create any variables of this data type
- Variables declared to be a user-defined class type are called **objects**
- When an object is defined, memory is allocated for the object and its data members are automatically initialized by the class constructor

Abstract Data Types in C++ (continued)

- **Attribute** of an object: a data member defined for the object's class

Syntax:

```
objectName.attributeName
```

- **Method** of an object: a function defined for the object's class

Syntax:

```
objectName.methodName(parameters)
```

Abstract Data Types in C++ (continued)

- Understanding the terminology:
  - **Class:** a programmer-defined data type out of which objects can be created
  - **Object:** created from a class; also called an **instance** of a class
  - **Instantiation:** the process of creating a new object from a class
  - **State:** the particular values for data members of a given instance of the class
  - **Behavior:** the operations that are permitted on an object's data members

Constructors

- **Constructor function:** a function with the same name as its class whose purpose is to initialize a new object's data members
- Constructor must not have a return type
- Multiple constructors can be defined with different number or type of parameters (overloading)
- If no constructor is defined, a compiler-supplied default constructor with no parameters and no body is used

Constructors (continued)

Syntax:

```
className::className(parameter list)
{
   function body
}
```

- **Default constructor**: any constructor that does not require arguments (none declared, or they have default values)

Constructors (continued)

Program 9.2

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

// class declaration section

class Date
{
  private:
    int month;
    int day;
    int year;
  public:
    Date(int = 7, int = 4, int = 2005);    // constructor
};

// implementation section

Date::Date(int mm, int dd, int yyyy)
{
  month = mm;
  day = dd;
  year = yyyy;
  cout << "Created a new data object with data values "
       << month << ", " << day << ", " << year << endl;
}

int main()
{
  Date a;           // declare an object
  Date b;           // declare an object
  Date c(4,1,2006); // declare an object

  return 0;
}
```

```
Created a new data object with data values 7, 4, 2005
Created a new data object with data values 7, 4, 2005
Created a new data object with data values 4, 1, 2006
```

Constructors (continued)

- Object members are initialized in the order they are declared in the class declarations section, *not* in the order they appear in the constructor function

Calling Constructors

- Constructor is called when an object is created
- Syntax for calling a constructor:
- C++ Style

```
className objectName(argument values);
```

- C Style

```
className objectName =    className(argument values)
```


Calling Constructors

- `Date c(4,1,2006);`
- `Date c = Date(4,1,2006);`
- `Date c = 8;`
- `Date a();` is not the same as the declaration `Date a;`

Constructors (continued)

- Constructors are like other functions:
  - They may be overloaded
  - They may have default arguments
  - They may be written as inline functions

Constructors (continued)

```
// class declaration

class Date
{
  private:
    int month;
    int day;
    int year;
  public:
    Date(int = 7, int = 4, int = 2005);   // constructor
    Date(long);              // another constructor
    void showDate();         // member function to display a date
};

// implementation section

Date::Date(int mm, int dd, int yyyy)
{
  month = mm;
  day = dd;
  year = yyyy;
}

Date::Date(long yyyymmdd)
{
  year = int(yyyymmdd/10000.0);    // extract the year
  month = int( (yyyymmdd - year * 10000.0)/100.00 ); // extract the month
  day = int(yyyymmdd - year * 10000.0 - month * 100.0); // extract the day
```

Destructors

- **Destructor:** a function with the same name as the class name, but preceded by a tilde (~)
- Only one destructor per class is allowed
- Destructor is called automatically when an object goes out of existence
- Purpose is to "clean up" any undesirable effects that might be left by the object

Common Programming Errors

- Failing to terminate the class declaration with a semicolon
- Including a return type with the constructor's prototype
- Failing to include a return type with other functions' prototypes
- Using the same name for a data member as for a member function
- Defining more than one default constructor for a class
- Failing to include the class name and scope operator (`::`) in the header line of all member functions defined in the class implementation section

Summary

- Class: a programmer-defined data type from which objects may be defined
- Two sections of a class definition: declaration and implementation
- Class members: the variables and functions declared in the declaration section
- Private access: members which can only be used by the class's own functions
- Public access: members can be accessed from outside the class

Summary (continued)

- Constructor: function that is automatically called each time an object is created from the class; has the same name as the class but no return type
- Default constructor: a constructor with no required arguments; only one allowed per class
- Constructors may be overloaded
- Destructor: function that is automatically called when an object goes out of scope; only one per class

Summary (continued)

- Destructor has no arguments and returns no value
- Destructor has the same name as the class name, preceded by a tilde (~)
- See the video demo on classes.