

INTRODUCTION TO CLASSES, OBJECTS, MEMBER FUNCTIONS AND STRINGS

Chapter 3

Deitel 10th Edition



TOPICS

Introduction – Classes

Using a Class: Test-Driving an Account Object

- Instantiating an object, headers & source code files, gets & sets, getline function

Parts of a Class Definition: Account Class

- Class definition, parts of a class, access specifiers

Initializing Objects with Constructors: Account Class

- Purpose of constructor, defining a constructor

TOPICS, CONTINUED

Software Engineering with Set and Get Member Functions

- Purpose of gets and sets

Demonstrating Data Validation: Account Class

- Two-parameter constructors

INTRODUCTION – CLASSES

INTRODUCTION

Remember

- Classes are reusable software components
- Often an abstraction of a “thing”, like a car, a student, an item in inventory

Each class you create becomes a new data type

- C++ is **extensible**

Attributes of a class defined with **data members**

Behaviors of a class defined with **member functions**

USING A CLASS: TEST-DRIVING AN ACCOUNT OBJECT

INSTANTIATING AN ACCOUNT OBJECT IN A DRIVER CLASS

Fig03.01 AccountTest.cpp

Driver program

- **main** function defined in its own file
- Usually instantiate objects
- Sometimes called the “client” of the class it instantiates

Can't call functions of a class before an object of the class is created

- Called **instantiation**

Instantiating an Account object

- Line 10
- The type of the variable **myAccount** is **Account**

HEADER FILES AND SOURCE-CODE FILES

Header file

- File that contains a reusable class definition
- Uses file extension **.h**

Included where needed using **#include**

- If it's missing, compiler doesn't know what Account is

Source code file

- In this example, it's the driver
- Uses file extension **.cpp**

Headers included in source code files

- And sometimes in other headers

CALLING A MEMBER FUNCTION OF A CLASS

Account's getName member function call

- Line 13

General format of a call to an object's function:

objectName.functionName(arguments if needed)

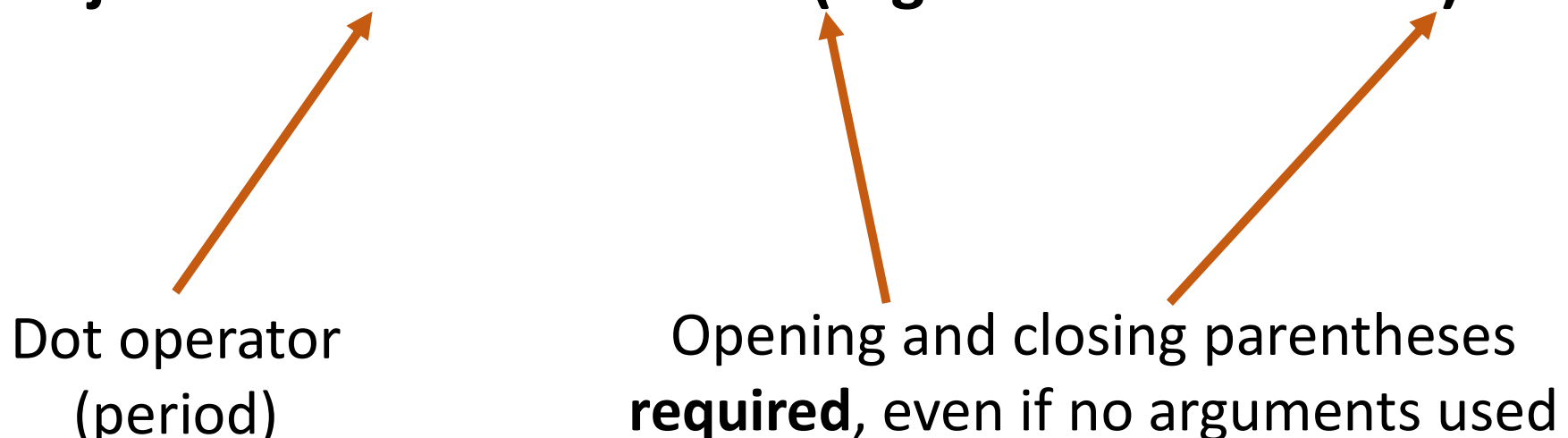
CALLING A MEMBER FUNCTION OF A CLASS – OPERATORS REQUIRED

Account's getName member function call

- Line 13

General format of a call to an object's function:

objectName.functionName(arguments if needed)



Dot operator
(period)

Opening and closing parentheses
required, even if no arguments used

CALLING A MEMBER FUNCTION OF A CLASS – TRANSFER OF CONTROL

Transfer of control

- The order in which code is executed

What is the transfer of control in a function call?

1. Arguments evaluated
2. Object is identified
3. Function is identified
4. Control moves to the function's parameter list
5. Argument values assigned to parameters
6. Function's body executes
7. If present, return value returned to calling code

INPUTTING A STRING WITH GETLINE

Part of **string** library, defined in header **<string>**

Part of **std namespace**, so need **std::** or **using** statement

Global function

Reads user input until **Enter key**

- Enter key is discarded

cin reads user input to first **white space**

- But anything after the space is not lost – can still be read

CALLING CLASS ACCOUNT'S SETNAME MEMBER FUNCTION

Stores the string passed in as the name

“**theName**” is the argument to the set function

Remember:

- **Argument** is Actual data
- **Parameter** is the description of the data

PARTS OF A CLASS DEFINITION:

ACCOUNT CLASS WITH DATA MEMBER AND SET & GET MEMBER FUNCTIONS

ACCOUNT CLASS DEFINITION

Fig. 3.2: Account.h

Begins with keyword **class**

Class definition enclosed in **curly braces**

Terminating semicolon after last brace

ACCOUNT CLASS DEFINITION – DATA MEMBERS

Identifiers in C++ use **camel case**

- Variable and function names begin with **lowercase letter**
- Class names begin with **uppercase letter**

Do not put **using** statements in **headers**

- use the **namespace** with the double-colon **::**

Typically, data members are **last** in the class definition

DATA MEMBERS AND MEMBER FUNCTIONS

Data member **name** of type **string**

- Line 18

Each object has its **own copy** of data members and member functions

Data members exist separately from execution of member functions

Declared inside class, but **outside** of any member functions

Order of items in a class:

- Public functions first
- Private functions next
- Data members last

SET MEMBER FUNCTIONS

setName Member Function

- Lines 9-11

Sets are also called setters

Used to **assign a value to a data member**

STRUCTURE OF A MEMBER FUNCTION

First line is the **function header**

Headers contain:

- **Return type** – a fundamental or programmer-defined data type; required; use **void** if none
- **Function name** – required
- **Opening parenthesis** – required
- **Parameter** or comma-separated list of parameters (optional)
- **Closing parenthesis** – required

PARAMETERS

Parameters contain:

- Data type
- Identifier

Parameters are **local variables**

- In scope only within that function
- Lost when function ends

PARAMETERS VS. ARGUMENTS

Parameters = variable declarations

Arguments = actual data passed when function executes

Argument types must be consistent with parameter types

Consistent with

- Data type of argument is equal to or smaller in bytes than data type of parameter
- Int argument → int parameter – ok
- Int argument → double parameter – ok
- Double argument → int parameter – ?

GET MEMBER FUNCTIONS

getName Member Function

- Line 14-16

Gets are also called getters

Returns data to the calling code, of the return type specified

return statement required if return type other than void

CONST KEYWORD

Line 14

Used to right of parameter list

Indicates function should not modify the object

- Only return info about the object (in this case, the account's name)

Compilation error if name is modified in getName() function

ACCESS SPECIFIERS PRIVATE AND PUBLIC

Private

- Private members can only be accessed inside the class
- Data members should always be private (data hiding)
- Allow access via public functions that can control access to private data members

Public

- Public members can be accessed both inside and outside of the class
- Outside only if there is a reference to an object of the class

Default access is private, unless public is explicitly specified

ACCESS SPECIFIERS PRIVATE AND PUBLIC – BEST PRACTICES

Use private or public on a line by itself, followed by a colon

Everything that follows an access modifier uses that access

- Until there is another access modifier listed

Group everything appropriately

- Only one mention of each without repeating
- Repeating access modifiers is confusing and sloppy

INITIALIZING OBJECTS WITH CONSTRUCTORS: ACCOUNT CLASS

CONSTRUCTORS

Data members in a class may or may not be initialized

- **Fundamental types are not initialized by default**
- Objects are initialized to **null**, unless another value is indicated

Constructors provide custom initialization

- Special functions within a class

Must have **exactly the same name** as the class

Normally public

DEFINING A CONSTRUCTOR IN THE ACCOUNT CLASS 1

Fig. 3.4: Account.h – lines 8-11

Receives a string to be assigned to data member **name**

- Parameter list defines data required to provide custom initialization for this object

Member initializer – line 9

: name{accountName}

- Value of parameter name in {} or () assigned to data member next to the colon
- More than one initializer – separate with commas
- More efficient than initializing within constructor, if no other logic needed (such as validation)

DEFINING A CONSTRUCTOR IN THE ACCOUNT CLASS 2

explicit keyword

- Line 8
- All single-parameter constructors must use explicit – we'll learn why later

Constructors do not have return types

- Don't even put void there!

Constructors cannot be const

- Initializing an object modifies it

Parameter names in different functions and constructors

- Not visible to each other; each has local scope

INITIALIZING ACCOUNT OBJECTS WHEN THEY'RE CREATED

Fig. 3.5: AccountTest.cpp

Lines 11-13

- Creates objects using the **single-parameter constructor** with a string argument

DEFAULT CONSTRUCTORS

Previous example Fig. 3.1: AccountTest.cpp

Created Account object like this:

Account account;

All object instantiations use a constructor

But this version of Account has no constructor explicitly written

DEFAULT CONSTRUCTORS AUTOMATIC IN SOME CASES

If a class does not explicitly define any constructor, a **default constructor** is automatically created/invoked

DOES NOT automatically initialize fundamental data members

If you write **any constructor**, the default is no longer automatically included

Should **always provide custom initialization**

SOFTWARE ENGINEERING WITH SET AND GET MEMBER FUNCTIONS

PURPOSE OF SET AND GET MEMBER FUNCTIONS

Validate attempts to change private data

Control how data is displayed to user of class

Sets – contain **validation**

Gets – contain control of **presentation**

DEMONSTRATING DATA VALIDATION: ACCOUNT CLASS

IN-CLASS INITIALIZER

Fig. 3.8: Account.h

Note **in-class initializer** on line 42:

```
int balance{0};
```

Data member **balance**

- Assigned the default value 0
- Can be used in any member function (see lines 15, 22, 28)

TWO-PARAMETER CONSTRUCTOR WITH VALIDATION

Constructor includes balance

Constructor not declared explicit

- Because it includes two parameters
- Explicit can only be used if there is only one parameter

Note validation of balance using an if statement

DEMONSTRATING DATA VALIDATION IN OTHER FUNCTIONS

deposit member function with validation

- Lines 21-23

getBalance member function

- Declared const

Fig. 3.9: AccountTest.cpp

- Note line 20 – variable not initialized here – Why not?
- Lines 14-40 – what could a problem be with this approach?