TT TEXTBOOK TESTBANKS

Solutions Manual for Programming with C++ 1st Mcmullen

SOLUTIONS MANUAL SAMPLE PREVIEW

PUBLISHER
Cengage

COPYRIGHT
2022

RESOURCE TYPE
Solutions Manual

ISBN
9780357421901

Instructor Manual

McMullen, Matthews, Parsons, Programming with C++ © 2022, 978-0-357-63775-3, Module 1: Computational Thinking

Table of Contents

Purpose and Perspective of the Module

The purpose of this module is to introduce algorithms to students. Students will learn steps and terminology related to creating an algorithm and apply this knowledge to the programming environment. Programming algorithms are specifically reviewed with an emphasis on how to assess a problem and decompose the processes into smaller subcomponents or modules. The importance of decomposition as an important tool for computer scientists is emphasized. Specifically, three types of decomposition are covered in more detail: structural, functional, and object-oriented decomposition. The importance of algorithm efficiency is presented along with a list of characteristics an effective algorithm possesses. Students will also learn how to recognize the various types of patterns frequently found in an algorithm: fill-in-the-blank, repetitive, and classification patterns. The module concludes with a discussion of abstraction. Students will learn why abstraction is an important computer science concept.

Cengage Supplements

The following product-level supplements provide additional information that may help you in preparing your course. They are available in the Instructor Resource Center.

- Test Bank: Cengage Testing, powered by Cognero® (contains assessment questions and problems)
- PowerPoint (provides text-based lectures and presentations)
- Master List of Learning Objectives (provides a comprehensive list of learning objectives for all modules)
- C++ Code Files (includes code files of figures from the modules and fully executable C++ programs)
- MindTap Educator Guide (contains a detailed outline of the MindTap)
- Guide to Teaching Online (includes pedagogical considerations and resources for teaching online)

List of Student Downloads

Students should download the following items from the Student Companion Center to complete the activities and assignments related to this module:

- C++ Snippets ReadMe (provides helpful information related to the embedded coding Snippets)

Module Objectives

The following objectives are addressed in this module:

1.1 Algorithms

1.1.1 Define the term “algorithm” as a series of steps for solving a problem or carrying out a task.

1.1.2 State that algorithms are the underlying logic for computer programs.

1.1.3 Define the term “computer program.”

1.1.4 Provide examples of algorithms used in everyday technology applications.

1.1.5 Confirm that there can be more than one algorithm for a task or problem and that some algorithms may be more efficient than others.

1.1.6 Explain why computer scientists are interested in algorithm efficiency.

1.1.7 List the characteristics of an effective algorithm.

1.1.8 Write an algorithm for accomplishing a simple, everyday technology application.

1.1.9 Write an alternate algorithm for an everyday technology task.

1.1.10 Select the more efficient of the two algorithms you have written.

1.2 Decomposition

1.2.1 Define the term “decomposition” as a technique for dividing a complex problem or solution into smaller parts.

1.2.2 Explain why decomposition is an important tool for computer scientists.

1.2.3 Differentiate the concepts of algorithms and decomposition.

1.2.4 Identify examples of structural decomposition.

1.2.5 Identify examples of functional decomposition.

1.2.6 Identify examples of object-oriented decomposition.

1.2.7 Provide examples of decomposition in technology applications.

1.2.8 Explain how dependencies and cohesion relate to decomposition.

1.3 Pattern Identification

1.3.1 Define the term “pattern identification” as a technique for recognizing similarities or characteristics among the elements of a task or problem.

1.3.2 Identify examples of fill-in-the-blank patterns.

1.3.3 Identify examples of repetitive patterns.

1.3.4 Identify examples of classification patterns.

1.3.5 Provide examples of pattern identification in the real world and in technology applications.

1.4 Abstraction

1.4.1 Define the term “abstraction” as a technique for generalization and for simplifying levels of complexity.

1.4.2 Explain why abstraction is an important computer science concept.

1.4.3 Provide an example illustrating how abstraction can help identify variables.

1.4.4 Provide examples of technology applications that have abstracted or hidden details.

1.4.5 Provide an example illustrating the use of a class as an abstraction of a set of objects.

1.4.6 Explain how the black box concept is an implementation of abstraction.

1.4.7 Identify appropriate levels of abstraction.

[return to top]

Key Terms

abstraction: hides details, simplifies complexity, substitutes a generalization for something specific, and allows an algorithm to work for multiple inputs.
algorithm: a series of steps for solving a problem or carrying out a task.
attributes: relate to the characteristics of an object.
classes: templates that specify the attributes for a classification of objects.
classification patterns: found in attributes, they describe any person or object.
computational thinking: a set of techniques designed to formulate problems and their solutions.
computer program: a set of instructions, written in a programming language such as C++, Python, or Java, that performs a specific task when executed by a digital device.
decomposition: the process of dividing an algorithm for a complex app into smaller parts so that the algorithm is easier to formulate.
functional decomposition: a process that breaks down modules into smaller actions, processes, or steps.
level of abstraction: relates to the amount of detail that is hidden.
methods: actions that an object can perform.
modules: structural units that perform distinct tasks.
objects: represent people, places, or things.

object-oriented decomposition: another way to apply decomposition to a module by looking for logical and physical objects that a computer program will manipulate.
pattern identification: the process of finding similarities in procedures and tasks.
programming algorithm: a set of steps that specifies the underlying logic and structure for the statements in a computer program.
structural decomposition: a process that identifies a hierarchy of structural units.
[return to top]

Module Outline

1.1 Algorithms

I. Algorithm Basics (1.1.1, 1.1.4)

a. Remind students that passwords can be used to protect online accounts and review the basic process. Emphasize that in today's world, passwords alone cannot adequately protect online accounts from perpetrators with bad intentions and additional steps need to be considered.
b. Introduce the concept of two-factor authentication. Note that two-factor authentication is a common way to add an extra layer of account protection from perpetrators with bad intentions.
    i. Note that students may hear the term *multi-factor authentication*. Explain that two-factor authentication always uses two forms of verification while multi-factor authentication may use two or more forms.
c. Refer to Figure 1-1 to illustrate a common form of two-factor authentication that sends a personal identification number (PIN) to a user's cell phone. Work through the two-factor authentication process by reviewing the series of steps that are implemented when using it.
d. Define an **algorithm** as a series of steps for solving a problem or carrying out a task. Note that the procedure for two-factor authentication can be considered an example of an algorithm.
e. Mention that algorithms exist for everyday tasks and tasks that involve technology. Review the following example tasks:
    - A recipe for baking brownies
    - The steps for changing a tire
    - The instructions for pairing a smart watch with a phone
    - The payment process at an online store
    - The procedure for posting a tweet
f. Take 5 minutes to help students gain experience writing a short algorithm. Break the class into small groups of students. Select two of the task examples above (or select a new idea). Assign each of the selected tasks to at

least two student groups. Ask each student group to devise a short algorithm to complete their assigned task. Have each student group present their solution to the class for a short discussion. Those groups that completed the same task can compare their results.

II. Programming Algorithms (1.1.2, 1.1.3, 1.1.5)

a. Remind students that in general, an algorithm is a series of steps for solving a problem or carrying out a task. Note that algorithms are an important programming tool.
b. Define a **programming algorithm** as a set of steps that specifies the underlying logic and structure for the statements in a computer program. Explain that programming algorithms can be considered as the blueprints for computer programs.
c. Define a **computer program** as a set of instructions, written in a programming language such as C++, Python, or Java, that performs a specific task when executed by a digital device. Mention that a computer program is an implementation of an algorithm.
d. To further explain a programming algorithm, refer to the first question and answer set on page 3 that illustrates two programming algorithms. Note that because algorithm two provides instructions that should be for the computer and not the user, it represents a programming algorithm.
e. To further illustrate algorithm efficiency, refer to the second question and answer set on page 3. Explain why algorithm two in this example is more efficient.
f. Explain that programming algorithm efficiency is important so that programs run as quickly as possible using the least amount of resources. Be sure to emphasize that programmers must code their algorithms efficiently with the consideration for future data growth.

III. "Good" Algorithms (1.1.6, 1.1.7)

a. Mention that a good algorithm tends to produce a computer program that operates efficiently, quickly, and reliably.
b. Spend time reviewing characteristics of good algorithms.
   - Input: The algorithm applies to a set of specified inputs.
   - Output: The algorithm produces one or more outputs.
   - Finite: The algorithm terminates after a finite number of steps.
   - Precise: Each step of the algorithm is clear and unambiguous.
   - Effective: The algorithm successfully produces the correct output.

c. Refer to Figure 1-2 and explain how a programmer might easily check to make sure an algorithm satisfies all the criteria for a good algorithm.

IV. Selecting and Creating Algorithms (1.1.8, 1.1.9, 1.1.10)

a. Mention that before coding, programmers consider various algorithms that might apply to a problem. Review the three ways a programmer might come up with an algorithm.
   - Use a standard algorithm
   - Perform the task manually
   - Apply computational thinking techniques

b. Explain that when programmers use **computational thinking**, they employ a set of techniques designed to formulate problems and their solutions.

c. Note that programmers can use computational thinking techniques such as decomposition, pattern identification, and abstraction to devise efficient algorithms.

d. For more information on computational thinking, see the following BBC Web site: **https://www.bbc.co.uk/bitesize/topics/z7tp34j**.

1.2 Decomposition

V. Decomposition Basics (1.2.1)

a. Remind students that decomposition is a computational thinking technique programmers can use.

b. Reference Figure 1-3 to illustrate how an app can have many components. In this example, review some possible mobile banking components.

c. Note that all of these components together can lead to an extensive algorithm. Define **decomposition** as the process of dividing the algorithm for an extensive app into smaller parts so that the algorithm is easier to formulate. Discuss how decomposition can help a programmer deal with smaller, more manageable pieces of a problem. Be sure to emphasize that a programmer will decompose a problem to make it easier to solve.

VI. Structural Decomposition (1.2.2, 1.2.3, 1.2.4, 1.2.7)

a. Point out that the first step in decomposition is to identify structural units that perform distinct tasks.

b. Continue working with the mobile banking app containing many components by referring to Figure 1-4 to illustrate how the app might be broken into structural units, called **modules**.

c. Define **structural decomposition** as a process that identifies a hierarchy of structural units. Explain that at the lowest levels of the hierarchy are modules that have a manageable scope for creating algorithms. Point out that in Figure 1-4 the lowest levels are indicated in yellow.
d. To further explain structural decomposition, refer to the question and answer set on page 6.
e. Take the time to review the provided tips for creating a structural decomposition diagram.

VII. Functional Decomposition (1.2.5)

a. Explain that **functional decomposition** breaks down modules into smaller actions, processes, or steps.
b. Refer to Figure 1-5 to illustrate a functional decomposition of the two-factor authentication module in an online banking app. Point out that the levels of the functional decomposition diagram get more specific until the nodes in the lowest levels begin to reveal instructions that should be incorporated into an algorithm.
c. Take the time to review the provided tips for constructing functional decomposition diagrams and deriving algorithms from them.

VIII. Object-Oriented Decomposition (1.2.6)

a. Introduce the concept of object-oriented decomposition by explaining that it involves another way to apply decomposition to a module by looking for logical and physical objects that a computer program will manipulate.
b. Refer to Figure 1-6 to illustrate an object-oriented decomposition of the two-factor authentication module for the mobile banking app. Explain that an object-oriented decomposition does not produce a hierarchy. Instead, it produces a collection of objects that can represent people, places, or things.
c. Take the time to review the provided tips for object-oriented decomposition.

IX. Dependencies and Cohesion (1.2.8)

a. Introduce this topic by explaining that in practice, there may be several viable ways to apply decomposition. Point out that an effective breakdown minimizes dependencies and maximizes cohesion among the various parts.
b. Review the two principles of decomposition:
   - Minimize dependencies. Although input and output may flow between nodes, changing the instructions in one module or object should not require changes to others.

- Maximize cohesion. Each object or module contains attributes, methods, or instructions that perform a single logical task or represent a single entity.

1.3 Pattern Identification

X. Pattern Identification Basics (1.3.1, 1.3.2)

a. Remind students that pattern identification is a computational thinking technique programmers can use.
b. Introduce the Amaze-Your-Friends math trick at the top of page 8, explaining that it is used to simply and quickly calculate the sum of numbers.
c. Ask each student to work through the four steps in the Amaze-Your-Friends math trick to quickly calculate the sum of the numbers from 1 to 10. Verify students calculated 55 as the answer.
d. Refer to the question and answer set on page 8 to challenge your students even more. Ask students to work through the four steps in the Amaze-Your-Friends math trick to quickly calculate the sum of the number from 1 to 200. Verify students multiply 100 x 201 to calculate a result of 20,100.
e. Emphasize that there is a pattern allowing the Amaze-Your-Friends math trick to work successfully. Note that in this case there is a fill-in-the-blank pattern.
f. Define **pattern identification** as the process of finding similarities in procedures and tasks.
g. Point out that pattern identification is a useful computational thinking technique for creating algorithms that can be used and reused on different data sets.
h. Explain that by recognizing the pattern in the Amaze-Your-Friends math trick, the students can use the algorithm to find the total of any series of numbers.

XI. Repetitive Patterns (1.3.3)

a. Explain that repetitive patterns can also be used as a programmer analyzes tasks and solves problems.
b. Review the algorithm on the bottom of page 8 that describes how to handle logins to a social media site. Read each line aloud emphasizing the repetitive lines of code.
c. Use the question and answer series set at the top of page 9 to challenge students to think about and recognize the number of times the repetitive lines of code occur.
d. Explain how the algorithm can be streamlined by using a repeat statement. Review the updated, streamlined algorithm in the answer section on page 9.

XII. Classification Patterns (1.3.4, 1.3.5)

a. Remind students that we are looking for patterns in algorithm steps to see if we can streamline the algorithms for completing tasks. Fill-in-the-blank and repetitive patterns are two pattern types we can look for and utilize in our algorithm streamlining process.
b. Note that in addition to fill-in-the-blank and repetitive patterns, a programmer might find classification patterns as they analyze algorithms.
c. Explain that programmers can often discover **classification patterns** in the attributes that describe any person or object. Emphasize that identifying classification patterns can help a programmer design programs that involve databases because the template identifies fields, such as User ID, that contain data.
d. To illustrate classification patterns, review the login credential sets for two fictitious users, Lee and Priya, who subscribe to a social media site.
e. Point out that each user has three attributes: a user ID, a password, and a mobile number. Emphasize that by recognizing a pattern like this, a programmer can create a template for any user's login credentials. Review the example template provided for this example on page 9.
f. Explain that classification patterns are useful if a programmer wants to design programs based on the interactions among a variety of objects, rather than a step-by-step algorithm. Note that in some programming circles, templates are called classes because they specify the attributes for a classification of objects.
g. Provide examples of classification patterns that are based on the interactions among a variety of objects.
    - People classified as social media subscribers have attributes for login credentials.
    - Vehicles classified as cars have attributes such as color, make, model, and VIN number.
    - Businesses classified as restaurants have a name, hours of operation, and a menu.

1.4 Abstraction

XIII. Abstraction Basics (1.4.1, 1.4.2, 1.4.3)

a. Remind students that abstraction is a computational thinking technique programmers can use.

b. Remind students of the Amaze-Your-Friends math trick, noting that by identifying a pattern, they were able to formulate a general algorithm that works for a sequence of any length.
c. Review the four lines of fill-in-the-blank code for the Amaze-Your-Friends math trick algorithm, emphasizing that the blank lines are an abstraction that represents the last number in the sequence.
d. Explain that an **abstraction** hides details, simplifies complexity, substitutes a generalization for something specific, and allows an algorithm to work for multiple inputs.
e. Emphasize that abstraction is a key element of computational thinking and helps programmers in a multitude of ways.
f. Ask students to raise their hands if they have programmed before. Mention that they may recognize that in the Amaze-Your-Friends algorithm, the blanks could become a variable with a name such as `last number`. Point out that `result` and `sum` are also variables because they represent values that change depending on the numbers in the sequence.
g. Conclude this section by noting that a variable is an abstraction because rather than representing a specific number, it can be used to represent many different numbers.

XIV. Classes and Objects (1.4.4, 1.4.5)

a. Introduce this topic by pointing out that abstraction has uses other than identifying variables.
b. Discuss why abstraction is important for understanding how to represent real world and conceptual objects.
c. Have students recall the pattern discovered for social media login credentials. Explain that with a little modification, the pattern becomes a template that can be applied to any subscriber.
   - Class: LoginCredentials
   - Attribute: user_ID
   - Attribute: user_password
   - Attribute: mobile_number
d. Mention that the LoginCredentials class is an abstraction that contains a set of attributes. Explain that the class was formed by abstracting away, or removing, details for any specific subscriber.
e. Note that abstractions are handy for any programs that deal with real-world objects in addition to technology objects, such as login credentials.
f. Refer to the question and answer set starting on page 10 to encourage students to envision a class that is an abstraction of the collection of objects

shown in Figure 1-7. Explain that the glassware class could have these attributes:

- Class: Glassware
- Attribute: Color
- Attribute: Capacity
- Attribute: Style

XV. Black Boxes (1.4.6)

a. Refer to Figure 1-9 to illustrate abstraction applied to driving a car. Explain that to drive a car, a driver does not have to know exactly what goes on under the hood. Point out that the engine is essentially a "black box" that the student controls using a few simple inputs such as the gas pedal, brake, and steering wheel. The details of the engine are hidden. Mention that in computer science terminology, these details have been "abstracted away."
b. Refer to Figure 1-9 to illustrate that in concept, a black box is anything that accepts some type of input and performs a specific process to produce output without requiring an understanding of its internal workings.
c. Explain why the black-box concept of abstraction is a fundamental aspect of computer science. Emphasize that this is possible because computer programs are abstractions.
    - A programmer can use a social media app without knowing anything about the programming that makes it work. That is, the icons that a user touches on their screen abstract away the details of the underlying programming.
d. Explain that programmers make extensive use of abstraction within programs by creating a set of instructions that functions like a black box. Refer to Figure 1-10 to illustrate how a programmer could bundle the instructions that handle login attempts into a black box.
e. Note that programming languages often have built-in abstractions that perform standard tasks. Use the example of a built-in random function that generates a random number when given a range, such as 1–100. Mention that programmers can incorporate the random function in a program without knowing how it works internally.

XVI. Levels of Abstraction (1.4.7)

a. Encourage students by indicating that applying the correct level of abstraction to their programs may take a little practice.
b. Note that a **level of abstraction** relates to the amount of detail that is hidden.

c. Refer to Figure 1-11 and remind students that abstracting out too much detail can make a program too generalized. Neglecting abstraction can produce programs that are too specific to work with a wide variety of data.
d. Conclude this topic by reminding students that with experience, they will be able to identify useful abstractions and gauge the correct level of abstraction to use..

[return to top]

Discussion Questions

You can assign these questions several ways: in a discussion forum in your LMS; as whole-class discussions in person; or as a partner or group activity in class.

1. Discuss algorithms that are being used by your favorite games and apps.

2. What is the best way to ensure that an algorithm is free of logical errors?

[return to top]

Suggested Usage for Lab Activities

1. No Lab Activities for Module 1.

[return to top]

Additional Activities and Assignments

1. **Commuter Algorithm**: Divide students into smaller groups based on how they commute to school: walk, ride a bike, drive a car, drive a motorcycle, take a bus, take a train, or use multiple modes of transportation. Allow each group 5-10 minutes to work together to write an algorithm for their mode of transportation. When time is up, each group should pass their algorithm to the group of students next to them in a clockwise direction. Now, each group should evaluate and critique the algorithm handed to them. After another 5-10 minutes, each group should take turns presenting their algorithm critique to the class so the class as a whole can learn from each other.

2. **Pattern Identification**: Pattern identification is a technique for recognizing similarities or characteristics among the elements of a task or problem. For a simple class activity, have students identify the characteristics or patterns that apply to posting a message on any social media site. First, have students identify three to five social media sites and write them on the board. Leave space in between for

additional information. Second, ask students to describe the steps for posting a message to each site, documenting those steps on the board, too. Third, ask students to discuss the similarities and patterns that apply to the sites.

3. **Sort Algorithm**: Sort algorithms are extremely common. One of the simplest is the bubble sort. A common classroom activity is to sort students by height using a bubble sort algorithm until students are in a line from shortest to tallest. Use the bubble sort to sort the class by height so students gain experience following an algorithm! If necessary start with three to five students so the students gain experience with the algorithmic steps. (The steps can be written on the board or handed out if necessary.)

   Summary of bubble sort of student height:

   - Compare the first two students.
   - If the student on the right is smaller than the student on the left, they should swap places.
   - Then compare the second and third students.
   - If the student on the right is smaller than the student on the left, they should swap places.
   - When you reach the end, start over at the beginning again.
   - Continue in this way until you make it through the entire row of students without swapping anybody.

Quick Quiz 1

1. A(n) ____ is a series of steps for solving a problem or carrying out a task.

   Answer: algorithm

2. True or False: Computer programs can be considered the blueprints for computer programs.

   Answer: False

3. An app can be broken into structural units, called ____.

   Answer: modules

4. True or False: With structural decomposition, the highest levels of the hierarchy are modules that have a manageable scope for creating algorithms.

Answer: False x

5. Object-oriented decomposition involves looking for logical and physical ____ that a computer program will manipulate.

   Answer: objects

Quick Quiz 2

1. The process of finding similarities in procedures and tasks is called ____ identification.

   Answer: pattern

2. Classification patterns come in handy if a programmer wants to design programs based on the interactions among a variety of objects, rather than a step-by-step algorithm.

   Answer: True

3. True or False: Abstraction is a key element of computational thinking.

   Answer: True

4. A(n) ____ is anything that accepts some type of input and performs a specific process to produce output without requiring an understanding of its internal workings.

   Answer: black box

5. The term "level of ____" relates to the amount of detail that is hidden.

   Answer: abstraction

[return to top]

Additional Resources

Cengage Video Resources

- MindTap Videos:
  - What is Computer Programming? 1:19 min.

Internet Resources

- Geeks For Geeks C++ Tutorial:

  - https://www.geeksforgeeks.org/cpp-tutorial/
- Top 10 Algorithms and Data Structures for Competitive Programming:
  - https://www.geeksforgeeks.org/top-algorithms-and-data-structures-for-competitive-programming/#algo7
- Algorithms are everywhere but what will it take for us to trust them?:
  - https://theconversation.com/algorithms-are-everywhere-but-what-will-it-take-for-us-to-trust-them-118830

[return to top]