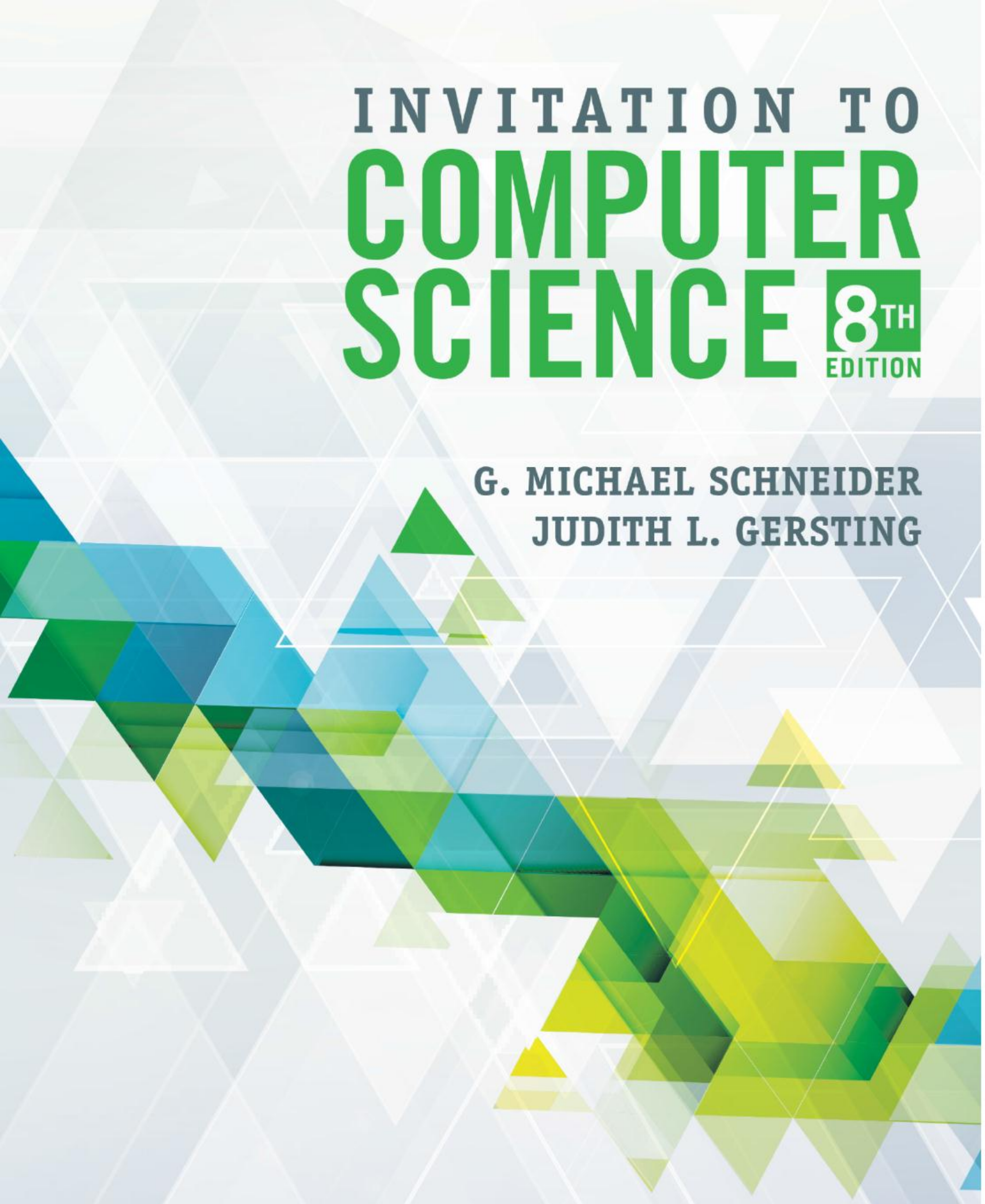


Chapter 2

Algorithm Discovery and Design

Learning Objectives (1 of 2)

- Explain the benefits of pseudocode over natural language or a programming language
- Represent algorithms using pseudocode
- Identify algorithm statements as sequential, conditional, or iterative
- Define abstraction and top-down design, and explain their use in breaking down complex problems

Learning Objectives (2 of 2)

- Illustrate the operation of algorithms for:
  - Multiplication by repeated addition
  - Sequential search of a collection of values
  - Finding the maximum element in a collection
  - Finding a pattern string in a larger piece of text

Introduction

- Everyday algorithms, such as hair washing, may not be suitable for computers to perform (as in Chapter 1).
- Algorithmic problem solving focuses on algorithms suitable for computers such as searching lists and matching patterns.
- Pseudocode is a tool for designing algorithms but does not run on a computing device.
- This chapter will use a set of problems to illustrate algorithmic problem solving, including those with conditional statements and loops.

Representing Algorithms (1 of 5)

- Natural language is:
  - Expressive and easy to use
  - Verbose, unstructured, and ambiguous
- Programming languages are:
  - Structured and designed for computers
  - Grammatically fussy and cryptic
- **Pseudocode** lies somewhere between these two and is used to design algorithms prior to coding them

Representing Algorithms Natural Language (1 of 2)

FIGURE 2.1

Initially, set the value of the variable carry to 0 and the value of the variable i to 0. When these initializations have been completed, begin looping as long as the value of the variable *i* is less than or equal to (*m* - 1). First, add together the values of the two digits a_i and b_i and the current value of the carry digit to get the result called c_i Now check the value of c_i to see whether it is greater than or equal to 10. If c_i is greater than or equal to 10, then reset the value of carry to 1 and reduce the value of c_i by 10; otherwise, set the value of carry to 0. When you are

Representing Algorithms Natural Language (2 of 2)

finished with that operation, add 1 to i and begin the loop all over again. When the loop has completed execution, set the leftmost digit of the result c_m to the value of carry and print out the final result, which consists of the digits c_m c_{m-1}…c_0. After printing the result, the algorithm is finished, and it terminates.

The addition algorithm of Figure 1.2 expressed in natural language

Representing Algorithms Programming Language

FIGURE 2.2

```
{
Scanner inp = new Scanner(System.in);
int i, m, carry;
int[] a = new int[100];
int[] b = new int[100];
int[] c = new int[100];
m = inp.nextInt();
for (int j = 0;j <= m-1;j++) {
       a[j] = inp.nextInt();
       b[j] = inp.nextInt();
}
carry = 0;
i = 0;
while (i < m) {
    c[i] = a[i] + b[i] + carry;
    if (c[i] >= 10)
             .
             .
             .
```

The beginning of the addition algorithm of Figure 1.2 expressed in a high-level programming language

Representing Algorithms (2 of 5)

- Sequential operations perform a single task
- The three basic sequential operations:
  - **Computation**: a single numeric calculation
  - **Input**: gets data values from outside the algorithm
  - **Output**: sends data values to the outside world
- A **sequential algorithm** is made up only of sequential operations
- A **variable** is a named storage location to hold a data value
- Example: computing average miles per gallon

CENGAGE

Representing Algorithms Sequential Algorithm

FIGURE 2.3

Step	Operation
1	Get values for *gallons used, starting mileage, ending mileage*
2	Set value of *distance driven* to *(ending mileage - starting mileage)*
3	Set value of *average miles per gallon* to *(distance driven ÷ gallons used)*
4	Print the value of *average miles per gallon*
5	Stop

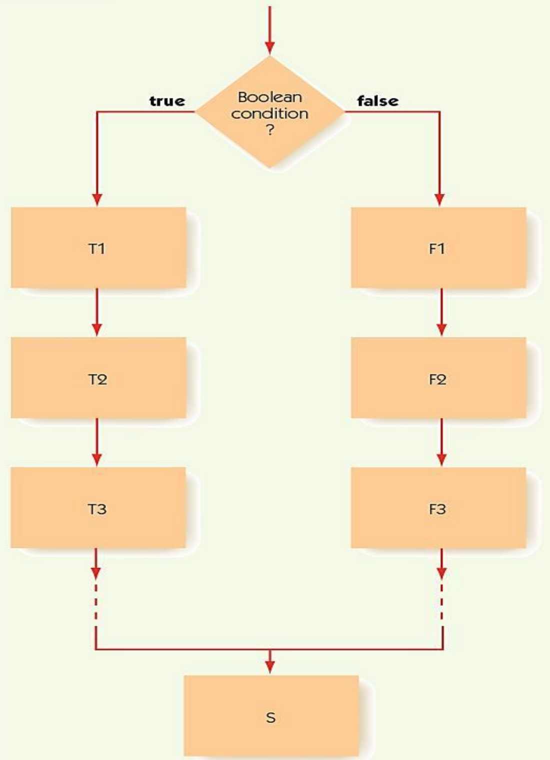
Algorithm for computing average miles per gallon (version 1)

Representing Algorithms (3 of 5)

- **Control operation:** changes the normal flow of control
- **Conditional statement**: asks a question and selects among alternative options:
    1. Evaluate the true/false condition
    2. If the condition is true, then do the first set of operations and skip the second set
    3. If the condition is false, skip the first set of operations and do the second set
- Example: check for good or bad gas mileage

Representing Algorithms Conditional Statement (1 of 2)

FIGURE 2.4



The if/then/else pseudocode statement

Representing Algorithms Conditional Statement (2 of 2)

FIGURE 2.5

Step	Operation
1	Get values for *gallons used, starting mileage, ending mileage*
2	Set value of *distance driven* to (*ending mileage* – *starting mileage*)
3	Set value of *average miles per gallon* to (*distance driven* ÷ *gallons used*)
4	Print the value of *average miles per gallon*
5	If *average miles per gallon* is >25.0 then
6	Print the message 'You are getting good gas mileage'
	Else
7	Print the message 'You are NOT getting good gas mileage'
8	Stop

Second version of the average miles per gallon algorithm

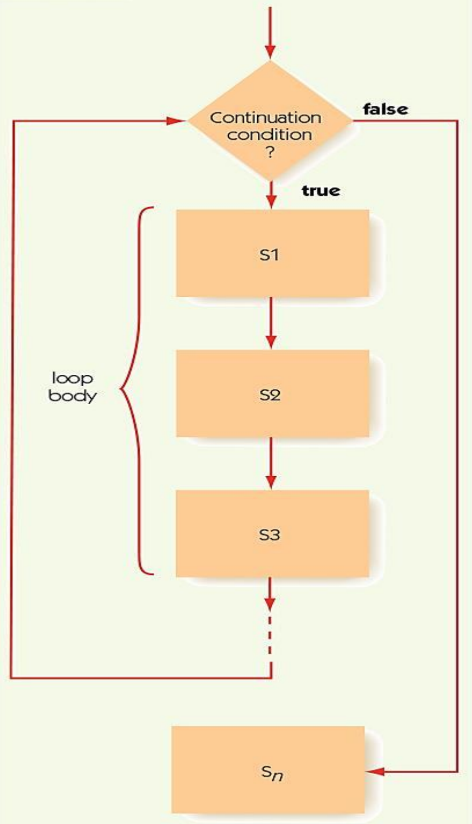
CENGAGE

Representing Algorithms (4 of 5)

- **Iteration:** an operation that causes looping, repeating a block of instructions
- While statement repeats while a condition remains true
  - **Continuation condition**: a test to see if while loop should continue
  - **Loop body**: instructions to perform repeatedly
- Example: repeated mileage calculations

Representing Algorithms Iteration and Loop Body (1 of 2)

FIGURE 2.6



Execution of the while loop

Representing Algorithms Iteration and Loop Body (2 of 2)

FIGURE 2.7

Step	Operation
1	*response* = *Yes*
2	While (*response* = *Yes*) do Steps 3 through 11
3	Get values for *gallons used, starting mileage, ending mileage*
4	Set value of *distance driven* to (*ending mileage* – *starting mileage*)
5	Set value of *average miles per gallon* to (*distance driven* ÷ *gallons used*)
6	Print the value of *average miles per gallon*
7	If average miles per gallon > 25.0 then
8	Print the message 'You are getting good gas mileage'
	Else
9	Print the message 'You are NOT getting good gas mileage'
10	Print the message 'Do you want to do this again? Enter Yes or No'
11	Get a new value for *response* from the user
12	Stop

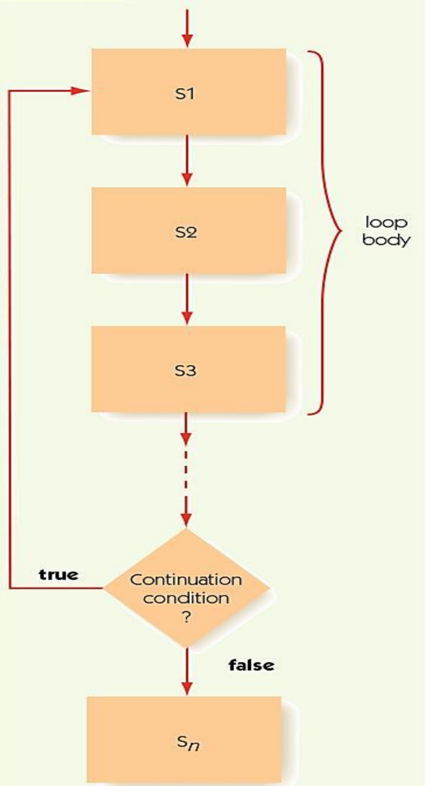
Third version of the average miles per gallon algorithm

Representing Algorithms (5 of 5)

- Do/while, alternate iterative operation
  - Continuation condition appears at the end
  - Loop body always performed at least once
- **Primitive operations:** sequential, conditional, and iterative are all that is needed

Representing Algorithms Do/While Posttest Loop (1 of 2)

FIGURE 2.8



Execution of the do/while posttest loop

Repre osttest

FIGURE 2.9

Computation:

```
Set the value of "variable" to "arithmetic expression"
```

Input/Output:

```
Get a value for "variable", "variable"...
Print the value of "variable", "variable", ...
Print the message 'message'
```

Conditional:

```
If "a true/false condition" is true then
    first set of algorithmic operations
Else
    second set of algorithmic operations
```

Iterative:

```
While ("a true/false condition") do Step i through Step j
    Step i: operation
    .
    .
    .
    Step j: operation

While ("a true/false condition") do
    operation
    .
    .
    .
    operation
End of the loop

Do
    operation
    operation
    .
    .
    .
    operation
While ("a true/false condition")
```

Summary of pseudocode language instructions

Examples of Algorithmic Problem Solving Example 1: Go Forth and Multiply (1 of 5)

Given two nonnegative integer values, $a \geq 0$, $b \geq 0$, compute and output the product ($a \times b$) using the technique of repeated addition. That is, determine the value of the sum $a + a + a + \ldots + a$ (b times).

Examples of Algorithmic Problem Solving Example 1: Go Forth and Multiply (2 of 5)

- Get input values
  - Get values for *a* and *b*
- Compute the answer
  - Loop *b* times, adding each time*
- Output the result
  - Print the final value*

* steps need elaboration

Examples of Algorithmic Problem Solving Example 1: Go Forth and Multiply (3 of 5)

- Loop b times, adding each time
  - Get values for *a* and *b*
  - Set the value of *count* to 0
  - While (*count* < *b*) do
    - … the rest of the loop*
    - Set the value of *count* to (*count* + 1)
  - End of the loop

* steps need elaboration

Examples of Algorithmic Problem Solving Example 1: Go Forth and Multiply (4 of 5)

- Loop b times, adding each time
  - Get values for *a* and *b*
  - Set the value of *count* to 0
  - Set the value of *product* to 0
  - While (*count* < *b*) do
    - Set the value of *product* to (*product* + *a*)
    - Set the value of *count* to (*count* + 1)
  - End of the loop
- Output the result
  - Print the value of *product*

Examples of Algorithmic Problem Solving Example 1: Go Forth and Multiply (5 of 5)

FIGURE 2.10

```
Get values for a and b
If (either a = 0 or b = 0) then
    Set the value of product to 0
Else
    Set the value of count to 0
    Set the value of product to 0
    While (count < b) do
        Set the value of product to (product + a)
        Set the value of count to (count + 1)
    End of loop
Print the value of product
Stop
```

Algorithm for multiplication of nonnegative values via repeated addition

Examples of Algorithmic Problem Solving Example 2: Looking, Looking, Looking (1 of 5)

Assume that we have a list of 10,000 names that we define as N_1, N_2, N_3, … , $N_{10,000}$, along with the 10,000 telephone numbers of those individuals, denoted as T_1, T_2, T_3, … , $T_{10,000}$. To simplify the problem, we initially assume that all names in the book are unique and that the names need not be in alphabetical order.

Examples of Algorithmic Problem Solving Example 2: Looking, Looking, Looking (2 of 5)

- Three versions here illustrate **algorithm discovery**, working toward a correct, efficient solution:
  - A sequential algorithm (no loops or conditionals)
  - An incomplete iterative algorithm
  - A correct algorithm

Examples of Algorithmic Problem Solving Example 2: Looking, Looking, Looking (3 of 5)

FIGURE 2.11

Step	Operation
1	Get values for *NUMBER*, $T_1, \ldots, T_{10,000}$, and $N_1, \ldots, N_{10,000}$
2	If $NUMBER = T_1$ then print the value of N_1
3	If $NUMBER = T_2$ then print the value of N_2
4	If $NUMBER = T_3$ then print the value of N_3
.	.
.	.
.	.
10,000	If $NUMBER = T_{9,999}$ then print the value of $N_{9,999}$
10,001	If $NUMBER = T_{10,000}$ then print the value of $N_{10,000}$
10,002	Stop

First attempt at designing a sequential search algorithm

Examples of Algorithmic Problem Solving Example 2: Looking, Looking, Looking (4 of 5)

FIGURE 2.12

Step	Operation
1	Get values for *NUMBER*, $T_1, \ldots, T_{10,000}$, and $N_1, \ldots, N_{10,000}$
2	Set the value of *i* to 1 and set the value of *Found* to NO
3	While (*Found* = NO) do Steps 4 through 7
4	If *NUMBER* is equal to the *i*th number on the list, T_i, then
5	Print the name of the corresponding person, N_i
6	Set the value of *Found* to YES
	Else (*NUMBER* is not equal to T_i)
7	Add 1 to the value of *i*
8	Stop

Second attempt at designing a sequential search algorithm

Examples of Algorithmic Problem Solving Example 2: Looking, Looking, Looking (5 of 5)

FIGURE 2.13

Step	Operation
1	Get values for *NUMBER*, $T_1, \dots, T_{10,000}$, and $N_1, \dots, N_{10,000}$
2	Set the value of *i* to 1 and set the value of *Found* to NO
3	While both (*Found* = NO) and ($i \le 10{,}000$) do Steps 4 through 7
4	If NUMBER is equal to the ith number on the list T_i then
5	Print the name of the corresponding person, N_i
6	Set the value of *Found* to YES
	Else (*NUMBER* is not equal to T_i)
7	Add 1 to the value of *i*
8	If (*Found* = NO) then
9	Print the message 'Sorry, this number is not in the directory'
10	Stop

The sequential search algorithm

Examples of Algorithmic Problem Solving Example 3: Big, Bigger, Biggest (1 of 2)

- **Library:** A collection of prewritten, useful algorithms
- A “building-block” algorithm used in many libraries:

  Given a value $n \geq 1$ and a list containing exactly n unique numbers called A_1, A_2, … , A_n, find and print out both the largest value in the list and the position in the list where that largest value occurred.

Examples of Algorithmic Problem Solving Example 3: Big, Bigger, Biggest (2 of 2)

FIGURE 2.14

Get a value for n, the size of the list
Get values for $A_1, A_2, \ldots, A_n$, the list to be searched
Set the value of *largest so far* to A_1
Set the value of *location* to 1
Set the value of i to 2
While ($i \leq n$) do
    If A_i > *largest so far* then
        Set *largest so far* to A_i
        Set *location* to i
    Add 1 to the value of i
End of the loop
Print out the values of *largest so far* and *location*
Stop

Algorithm to find the largest value in a list

Examples of Algorithmic Problem Solving Example 4: Meeting Your Match (1 of 4)

- Pattern matching: common across many applications, such as:
  - Word processor search, web search, image analysis, and human genome project

Let’s formally define the pattern-matching problem as follows:

You will be given some text composed of n characters that will be referred to as $T_1\ T_2\ \ldots\ T_n$. You will also be given a pattern of m characters, $m \le n$, that will be represented as $P_1\ P_2\ \ldots\ P_m$. The algorithm must locate every occurrence of the given pattern within the text. The output of the algorithm is the location in the text where each match occurred.

Examples of Algorithmic Problem Solving Example 4: Meeting Your Match (2 of 4)

- Algorithm has two parts:
  1. Sliding the pattern along the text, aligning it with each position in turn
  2. Given a particular alignment, determine if there is a match at that location
- Solve parts separately and use
  - **Abstraction:** focus on high level, not details
  - **Top-down design:** start with big picture, gradually elaborate parts

Examples of Algorithmic Problem Solving Example 4: Meeting Your Match (3 of 4)

FIGURE 2.15

Get values for n and m, the size of the text and the pattern, respectively
Get values for both the text $T_1\ T_2 \ldots T_n$ and the pattern $P_1\ P_2 \ldots P_m$
Set k, the starting location for the attempted match, to 1
Keep going until we have fallen off the end of the text
    Attempt to match every character in the pattern beginning at
        position k of the text (this is Step 1 from the previous page)
    If there was a match then
        Print the value of k, the starting location of the match
    Add 1 to k, which slides the pattern forward one position (this is Step 2)
End of the loop
Stop

First draft of the pattern-matching algorithm

Examples of Algorithmic Problem Solving Example 4: Meeting Your Match (4 of 4)

FIGURE 2.16

Get values for n and m, the size of the text and the pattern, respectively
Get values for both the text $T_1\ T_2 \ldots T_n$ and the pattern $P_1\ P_2 \ldots P_m$
Set k, the starting location for the attempted match, to 1
While ($k \leq (n - m + 1)$) do
    Set the value of i to 1
    Set the value of *Mismatch* to NO
    While both ($i \leq m$) and (*Mismatch* = NO) do
        If $P_i \neq T_{k+(i-1)}$ then
            Set *Mismatch* to YES
        Else
            Increment i by 1 (to move to the next character)
    End of the loop
    If *Mismatch* = NO then
        Print the message 'There is a match at position'
        Print the value of k
    Increment k by 1
End of the loop
Stop

Final draft of the pattern-matching algorithm

Summary

- Pseudocode is used for algorithm design: structured like code but allows English and mathematical phrasing and notation
- Pseudocode is made up of sequential, conditional, and iterative operations
- Algorithmic problem solving involves:
  - Step-by-step development of algorithm pieces
  - Use of abstraction and top-down design