Objectives

After studying this chapter, students will be able to:

• 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
Objectives (continued)

After studying this chapter, students will be able to:
• Illustrate the operation of sample algorithms
  – 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

• Algorithms for everyday may not be suitable for computers to perform (as in Chapter 1)
• Algorithmic problem solving focuses on algorithms suitable for computers
• Pseudocode is a tool for designing algorithms
• This chapter will use a set of problems to illustrate algorithmic problem solving
Representing Algorithms

• **Pseudocode** is used to design algorithms
• Natural language is:
  – expressive, easy to use
  – verbose, unstructured, and ambiguous
• Programming languages are:
  – structured, designed for computers
  – grammatically fussy, cryptic
• Pseudocode lies somewhere between these two
FIGURE 2.1

Initially, set the value of the variable \( \text{carry} \) 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 \( \text{carry} \) to 1 and reduce the value of \( c_i \) by 10; otherwise, set the value of \( \text{carry} \) to 0. When you are 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 \( \text{carry} \) and print out the final result, which consists of the digits \( c_m \, c_{m-1} \, \ldots \, c_0 \). After printing the result, the algorithm is finished, and it terminates.

The addition algorithm of Figure 1.2 expressed in natural language
```java
{  
    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 (continued)

- Sequential operations perform a single task
  - **Computation**: a single numeric calculation
  - **Input**: gets data values from outside the algorithm
  - **Output**: sends data values to the outside world

- A **variable** is a named location to hold a value

- A **sequential algorithm** is made up only of sequential operations

- Example: computing average miles per gallon
<table>
<thead>
<tr>
<th>Step</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Get values for gallons used, starting mileage, ending mileage</td>
</tr>
<tr>
<td>2</td>
<td>Set value of distance driven to (ending mileage – starting mileage)</td>
</tr>
<tr>
<td>3</td>
<td>Set value of average miles per gallon to (distance driven ÷ gallons used)</td>
</tr>
<tr>
<td>4</td>
<td>Print the value of average miles per gallon</td>
</tr>
<tr>
<td>5</td>
<td>Stop</td>
</tr>
</tbody>
</table>

**Algorithm for computing average miles per gallon (version 1)**
Representing Algorithms (continued)

- **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
<table>
<thead>
<tr>
<th>Step</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Get values for gallons used, starting mileage, ending mileage</td>
</tr>
<tr>
<td>2</td>
<td>Set value of distance driven to (ending mileage – starting mileage)</td>
</tr>
<tr>
<td>3</td>
<td>Set value of average miles per gallon to (distance driven ÷ gallons used)</td>
</tr>
<tr>
<td>4</td>
<td>Print the value of average miles per gallon</td>
</tr>
<tr>
<td>5</td>
<td>If average miles per gallon is greater than 25.0 then</td>
</tr>
<tr>
<td>6</td>
<td>Print the message ‘You are getting good gas mileage’</td>
</tr>
<tr>
<td></td>
<td>Else</td>
</tr>
<tr>
<td>7</td>
<td>Print the message ‘You are NOT getting good gas mileage’</td>
</tr>
<tr>
<td>8</td>
<td>Stop</td>
</tr>
</tbody>
</table>

Second version of the average miles per gallon algorithm
Representing Algorithms (continued)

• **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
FIGURE 2.6

Execution of the while loop

Contuation condition?

true

false

S1

S2

S3

S\text{\#}
**FIGURE 2.7**

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

Third version of the average miles per gallon algorithm
Representing Algorithms (continued)

• Do/while, alternate iterative operation
  – continuation condition appears at the end
  – loop body always performed at least onc

• Primitive operations: sequential, conditional, and iterative are all that is needed
FIGURE 2.8

Execution of the do/while posttest loop

- **S1**
- **S2**
- **S3**
- **S_n**

Loop body

Continuation condition? [true] [false]
Summary of pseudocode language instructions:

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
  ...
  ...
While (“a true/false condition”)
Examples of Algorithmic Problem Solving
Example 1: Go Forth and Multiply

“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

- 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 (continued)

• Loop b times, adding each time
  – Set the value of \( \text{count} \) to 0
  – While \( \text{count} < b \) do
    • … the rest of the loop*
    • Set the value of \( \text{count} \) to \( \text{count} + 1 \)
  – End of loop

* steps need elaboration
Examples of Algorithmic Problem Solving

• Loop b times, adding each time
  – 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

• Output the result
  – Print the value of product
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 ($\text{count} < b$) do
    Set the value of product to ($\text{product} + a$)
    Set the value of count to ($\text{count} + 1$)
  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

“Assume that we have a list of 10,000 names that we define as \( N_1, N_2, N_3, \ldots, N_{10,000} \), along with the 10,000 telephone numbers of those individuals, denoted as \( T_1, T_2, T_3, \ldots, T_{10,000} \). 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 (continued)

• 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
**FIGURE 2.11**

<table>
<thead>
<tr>
<th>Step</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Get values for $NAME, N_1, \ldots, N_{10,000}$, and $T_1, \ldots, T_{10,000}$</td>
</tr>
<tr>
<td>2</td>
<td>If $NAME = N_1$ then print the value of $T_1$</td>
</tr>
<tr>
<td>3</td>
<td>If $NAME = N_2$ then print the value of $T_2$</td>
</tr>
<tr>
<td>4</td>
<td>If $NAME = N_3$ then print the value of $T_3$</td>
</tr>
<tr>
<td>\vdots</td>
<td>\vdots</td>
</tr>
<tr>
<td>10,000</td>
<td>If $NAME = N_{9,999}$ then print the value of $T_{9,999}$</td>
</tr>
<tr>
<td>10,001</td>
<td>If $NAME = N_{10,000}$ then print the value of $T_{10,000}$</td>
</tr>
<tr>
<td>10,002</td>
<td>Stop</td>
</tr>
</tbody>
</table>

First attempt at designing a sequential search algorithm
FIGURE 2.12

Step | Operation
--- | ---
1 | Get values for $NAME$, $N_1$, $\ldots$, $N_{10,000}$, and $T_1$, $\ldots$, $T_{10,000}$
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 $NAME$ is equal to the $i$th name on the list $N_i$, then
5 | Print the telephone number of that person, $T_i$
6 | Set the value of $Found$ to YES
7 | Else ($NAME$ is not equal to $N_i$)
8 | Add 1 to the value of $i$
9 | Stop

Second attempt at designing a sequential search algorithm
### FIGURE 2.13

<table>
<thead>
<tr>
<th>Step</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Get values for NAME, ( N_1, \ldots, N_{10,000} ), and ( T_1, \ldots, T_{10,000} )</td>
</tr>
<tr>
<td>2</td>
<td>Set the value of ( i ) to 1 and set the value of ( Found ) to NO</td>
</tr>
<tr>
<td>3</td>
<td>While both (( Found = NO )) and ( (i \leq 10,000) ) do Steps 4 through 7</td>
</tr>
<tr>
<td>4</td>
<td>If NAME is equal to the ( i )th name on the list ( N_i ) then</td>
</tr>
<tr>
<td>5</td>
<td>Print the telephone number of that person, ( T_i )</td>
</tr>
<tr>
<td>6</td>
<td>Set the value of ( Found ) to YES</td>
</tr>
<tr>
<td>7</td>
<td>Else (NAME is not equal to ( N_i ))</td>
</tr>
<tr>
<td>8</td>
<td>Add 1 to the value of ( i )</td>
</tr>
<tr>
<td>9</td>
<td>If (( Found = NO )) then</td>
</tr>
<tr>
<td>10</td>
<td>Print the message ‘Sorry, this name is not in the directory’</td>
</tr>
<tr>
<td></td>
<td>Stop</td>
</tr>
</tbody>
</table>

---

**The sequential search algorithm**
Examples of Algorithmic Problem Solving
Example 3: Big, Bigger, Biggest

• A “building-block” algorithm used in many libraries

• **Library**: A collection of pre-defined useful algorithms

  “Given a value $n \geq 1$ and a list containing exactly $n$ unique numbers called $A_1, A_2, \ldots, A_n$, find and print out both the largest value in the list and the position in the list where that largest value occurred.”
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 \textit{largest so far} to \( A_1 \)
Set the value of \textit{location} to 1
Set the value of \( i \) to 2
While \((i \leq n)\) do
    If \( A_i \) > \textit{largest so far} then
        Set \textit{largest so far} to \( A_i \)
        Set \textit{location} to \( i \)
        Add 1 to the value of \( i \)
End of the loop
Print out the values of \textit{largest so far} and \textit{location}
Stop

Algorithm to find the largest value in a list
Examples of Algorithmic Problem Solving

• Pattern-matching: common across many applications
  – word processor search, web search, image analysis, human genome project

“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, \( m \leq n \), 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 (continued)

• 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
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
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, we are finished

Final draft of the pattern-matching algorithm
Summary

• Pseudocode is used for algorithm design: structured like code, but allows English and math 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