An Introduction to Programming with C++

Sixth Edition

Chapter 2

Beginning the Problem-Solving Process
Objectives

• Explain the problem-solving process used to create a computer program
• Analyze a problem
• Complete an IPO chart
• Plan an algorithm using pseudocode and flowcharts
• Desk-check an algorithm
Problem Solving

- People solve hundreds of simple problems every day without thinking about how they do it.
- Understanding the thought process involved can help in solving more complex problems.
- You can also use a similar process to design a computer solution to a problem (computer program).
Solving Everyday Problems

• First step in solving a problem: analyze it
  – Example: paying and mailing a bill
• Next, you plan, review, implement, and evaluate the solution
• After this, it may be necessary to modify the solution
Figure 2-1 Summary of the analysis and planning steps for the bill paying problem
Solving Everyday Problems (cont’d.)

<table>
<thead>
<tr>
<th>Items used to accomplish the goal</th>
<th>Algorithm</th>
<th>Goal</th>
</tr>
</thead>
<tbody>
<tr>
<td>bill</td>
<td>1. use the pen to fill in the bank check’s date, payee, numerical amount, and written amount</td>
<td>pay the bill</td>
</tr>
<tr>
<td>preaddressed envelope</td>
<td>2. use the pen to sign the bank check</td>
<td></td>
</tr>
<tr>
<td>bank check</td>
<td>3. use the pen to write the customer account number on the bank check</td>
<td></td>
</tr>
<tr>
<td>pen</td>
<td>4. put the return address label on the preaddressed envelope</td>
<td></td>
</tr>
<tr>
<td>return address label</td>
<td>5. put the postage stamp on the preaddressed envelope</td>
<td></td>
</tr>
<tr>
<td>postage stamp</td>
<td>6. if (the bill has a return stub)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>tear off the return stub</td>
<td></td>
</tr>
<tr>
<td></td>
<td>put the return stub and bank check in the preaddressed envelope</td>
<td></td>
</tr>
<tr>
<td>modifications made to the original algorithm in Figure 2-1</td>
<td>else</td>
<td></td>
</tr>
<tr>
<td></td>
<td>make a copy of the bill for your records</td>
<td></td>
</tr>
<tr>
<td></td>
<td>put the bill and bank check in the preaddressed envelope</td>
<td></td>
</tr>
<tr>
<td></td>
<td>end if</td>
<td></td>
</tr>
<tr>
<td></td>
<td>7. seal the preaddressed envelope</td>
<td></td>
</tr>
<tr>
<td></td>
<td>8. mail the preaddressed envelope</td>
<td></td>
</tr>
</tbody>
</table>

Figure 2-2 Modified algorithm for the bill paying problem
Creating Computer Solutions to Problems

- A similar process to everyday problem solving is used to create computer programs
- A computer program is a solution implemented on a computer
- There are six steps to creating a computer solution to a problem
Creating Computer Solutions to Problems (cont’d.)

HOW TO Create a Computer Solution to a Problem

1. Analyze the problem
2. Plan the algorithm
3. Desk-check the algorithm
4. Code the algorithm into a program
5. Desk-check the program
6. Evaluate and modify (if necessary) the program

Figure 2-3 How to create a computer solution to a problem
Step 1–Analyzing the Problem

• It is essential to understand a problem before creating a solution to it
• Analyze a problem to:
  – Determine the goal of solving it
    • Output
    – Determine the items needed to achieve that goal
    • Input
• Always search first for the output
Some programmers use an **IPO chart** to organize and summarize the results of a problem analysis. 

- **IPO**: Input, processing, and output

![IPO chart example](image) 

Figure 2-5 Partially completed IPO chart showing the input and output items
Hints for Analyzing Problems (cont’d.)

• Some problem specifications contain incomplete information

Figure 2-7 Problem specification that does not contain enough information

Jack Osaki earns $7 per hour. Last week, Jack worked 50 hours. He wants a program that calculates and displays his weekly gross pay.
Hints for Analyzing Problems (cont’d.)

• Distinguish between information that is missing and information that is implied

Caroline Casey wants a program that calculates and displays the area of any rectangle.

Figure 2-8 Problem specification in which the input is not explicitly stated
Step 2—Planning the Algorithm

• Algorithm: set of instructions that will transform the problem’s input into its output
  – Record in the Processing column of the IPO chart
  – Can be written as pseudocode or a flowchart

• **Pseudocode**: tool programmers use to help plan an algorithm
  – Short English statements
  – Not standardized
  – Not understandable by a computer
Step 2−Planning the Algorithm (cont’d.)

**Problem specification**
Treyson Mobley wants a program that calculates and displays the amount he should tip a waiter at a restaurant. The program should subtract any liquor charge from the total bill and then calculate the tip (using a percentage) on the remainder.

<table>
<thead>
<tr>
<th>Input</th>
<th>Processing</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>total bill</td>
<td>Processing items: none</td>
<td>tip</td>
</tr>
<tr>
<td>liquor charge</td>
<td></td>
<td></td>
</tr>
<tr>
<td>tip percentage</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Algorithm:**
1. enter the total bill, liquor charge, and tip percentage
2. calculate the tip by subtracting the liquor charge from the total bill and then multiplying the remainder by the tip percentage
3. display the tip

Figure 2-9 Problem specification and IPO chart for the Treyson Mobley problem
Step 2—Planning the Algorithm (cont’d.)

• **Flowcharts** are also used to plan an algorithm
  – Use standardized symbols
  – Symbols connected with **flowlines**
  – Oval: **start/stop symbol**
    • Represents beginning and end of algorithm
  – Rectangle: **process symbol**
    • Represents tasks such as calculations
  – Parallelogram: **input/output symbol**
    • Represents I/O tasks
Figure 2-10 Figure 2-9’s algorithm in flowchart form
Step 2—Planning the Algorithm (cont’d.)

• A problem can have more than one solution

**Problem specification**
Treyson Mobley wants a program that calculates and displays the amount he should tip a waiter at a restaurant. The program should subtract any liquor charge from the total bill and then calculate the tip (using a percentage) on the remainder.

<table>
<thead>
<tr>
<th>Input</th>
<th>Processing</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>total bill</td>
<td>total bill without liquor charge</td>
<td>tip</td>
</tr>
<tr>
<td>liquor charge</td>
<td></td>
<td></td>
</tr>
<tr>
<td>tip percentage</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Algorithm (pseudocode):**
1. enter the total bill, liquor charge, and tip percentage
2. calculate the total bill without liquor charge by subtracting the liquor charge from the total bill
3. calculate the tip by multiplying the total bill without liquor charge by the tip percentage
4. display the tip

Figure 2-11 A different solution to the Treyson Mobley problem (pseudocode)
Step 2—Planning the Algorithm (cont’d.)

- Processing item: an intermediate value (neither input nor output) the algorithm uses to transform input into output

Figure 2-11 A different solution to the Treyson Mobley problem (flowchart)
Step 3—Desk-Checking the Algorithm

• **Desk-checking** an algorithm verifies that it is correct
  – Refers to checking an algorithm by hand, rather than with a computer
  – Also called **hand-tracing**
• Choose sample data and manually compute the expected output value
• Creating a desk-check table can be helpful
Step 3—Desk-Checking the Algorithm (cont’d.)

Figure 2-12 Manual tip calculation for the first desk-check

\[
\begin{array}{c}
45 \quad \text{(total bill)} \\
- 10 \quad \text{(liquor charge)} \\
\hline
35 \quad \text{(total bill without liquor charge)} \\
\times 0.2 \quad \text{(tip percentage)} \\
\hline
7 \quad \text{(tip)}
\end{array}
\]
Step 3−Desk-Checking the Algorithm (cont’d.)

![Diagram of input, processing, and output for desk-checking algorithm]

Figure 2-13 Treyson Mobley solution and partially completed desk-check table
### Step 3—Desk-Checking the Algorithm (cont’d.)

#### Figure 2-14 Input values entered in the desk-check table

<table>
<thead>
<tr>
<th>total bill</th>
<th>liquor charge</th>
<th>tip percentage</th>
<th>total bill without liquor charge</th>
<th>tip</th>
</tr>
</thead>
<tbody>
<tr>
<td>45</td>
<td>10</td>
<td>.2</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### Figure 2-15 Processing item’s value entered in the desk-check table

<table>
<thead>
<tr>
<th>total bill</th>
<th>liquor charge</th>
<th>tip percentage</th>
<th>total bill without liquor charge</th>
<th>tip</th>
</tr>
</thead>
<tbody>
<tr>
<td>45</td>
<td>10</td>
<td>.2</td>
<td>35</td>
<td></td>
</tr>
</tbody>
</table>

#### Figure 2-16 Output value entered in the desk-check table

<table>
<thead>
<tr>
<th>total bill</th>
<th>liquor charge</th>
<th>tip percentage</th>
<th>total bill without liquor charge</th>
<th>tip</th>
</tr>
</thead>
<tbody>
<tr>
<td>45</td>
<td>10</td>
<td>.2</td>
<td>35</td>
<td>7</td>
</tr>
</tbody>
</table>
Step 3—Desk-Checking the Algorithm (cont’d.)

Figure 2-17 Manual tip calculation for the second desk-check

$ 30  \quad \text{(total bill)}
- \quad 0  \quad \text{(liquor charge)}
\hline
30  \quad \text{(total bill without liquor charge)}
\times \quad .15  \quad \text{(tip percentage)}
\hline
$ 4.50  \quad \text{(tip)}
<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>total bill</td>
<td>liquor charge</td>
<td>tip percentage</td>
<td>total bill without liquor charge</td>
<td>tip</td>
<td></td>
</tr>
<tr>
<td>45</td>
<td>10</td>
<td>.2</td>
<td>35</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>0</td>
<td>.15</td>
<td>30</td>
<td>30</td>
<td></td>
</tr>
</tbody>
</table>

**Figure 2-18** Second set of input values entered in the desk-check table

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>total bill</td>
<td>liquor charge</td>
<td>tip percentage</td>
<td>total bill without liquor charge</td>
<td>tip</td>
<td></td>
</tr>
<tr>
<td>45</td>
<td>10</td>
<td>.2</td>
<td>35</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>0</td>
<td>.15</td>
<td>30</td>
<td>30</td>
<td></td>
</tr>
</tbody>
</table>

**Figure 2-19** Value of the second desk-check’s processing item entered in the desk-check table

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>total bill</td>
<td>liquor charge</td>
<td>tip percentage</td>
<td>total bill without liquor charge</td>
<td>tip</td>
<td></td>
</tr>
<tr>
<td>45</td>
<td>10</td>
<td>.2</td>
<td>35</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>0</td>
<td>.15</td>
<td>30</td>
<td>30</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>4.50</td>
</tr>
</tbody>
</table>

**Figure 2-20** Value of the second desk-check’s output item entered in the desk-check table
Step 3—Desk-Checking the Algorithm (cont’d.)

• **Valid data**: data that the algorithm is expecting the user to enter

• **Invalid data**: data that the algorithm is not expecting the user to enter

• You should test an algorithm with invalid data
  – Users may make mistakes when entering data