3

Structured Program Development in C

OBJECTIVES

In this chapter you will learn:

- Basic problem-solving techniques.
- To develop algorithms through the process of top-down, stepwise refinement.
- To use the `if` selection statement and `if...else` selection statement to select actions.

Outline

3.1 Introduction

- **Before writing a program:**
  - **Have a thorough understanding of the problem**
  - **Carefully plan an approach for solving it**
- **While writing a program:**
  - **Know what "building blocks" are available**
  - **Use good programming principles**

3.2 Algorithms

- **Computing problems**
  - **All can be solved by executing a series of actions in a specific order**
- **Algorithm: procedure in terms of**
  - **Actions to be executed**
  - **The order in which these actions are to be executed**
- **Program control**
  - **Specify order in which statements are to be executed**

3.3 Pseudocode

- **Pseudocode**
  - **Artificial, informal language that helps us develop algorithms**
  - **Similar to everyday English**
  - **Not actually executed on computers**
  - **Helps us "think out" a program before writing it**
    - **Easy to convert into a corresponding C++ program**
    - **Consists only of executable statements**

3.4 Control Structures

- **Sequential execution**
  - **Statements executed one after the other in the order written**
- **Transfer of control**
  - **When the next statement executed is not the next one in sequence**
  - **Overuse of `goto` statements led to many problems**
- **Bohm and Jacopini**
  - **All programs written in terms of 3 control structures**
    - **Sequence structures: Built into C. Programs executed sequentially by default**
    - **Selection structures: C has three types: `if`, `if…else`, and `switch`**
    - **Repetition structures: C has three types: `while`, `do…while` and `for`**

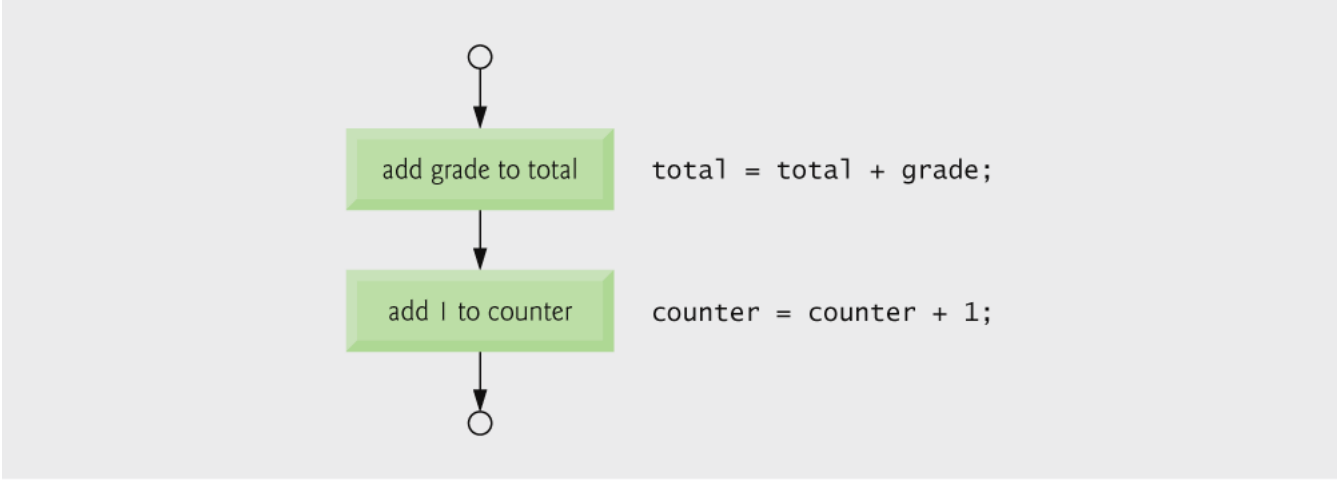

Fig. 3.1 | Flowcharting C's sequence structure.

3.4 Control Structures

- **Flowchart**
  - **Graphical representation of an algorithm**
  - **Drawn using certain special-purpose symbols connected by arrows called flowlines**
  - **Rectangle symbol (action symbol):**
    - **Indicates any type of action**
  - **Oval symbol:**
    - **Indicates the beginning or end of a program or a section of code**
- **Single-entry/single-exit control structures**
  - **Connect exit point of one control structure to entry point of the next (control-structure stacking)**
  - **Makes programs easy to build**

3.5 The `if` selection statement

- **Selection structure:**
  - **Used to choose among alternative courses of action**
  - **Pseudocode:**

    If student's grade is greater than or equal to 60
    Print "Passed"
- **If condition `true`**
  - **Print statement executed and program goes on to next statement**
  - **If `false`, print statement is ignored and the program goes onto the next statement**
  - **Indenting makes programs easier to read**
    - **C ignores whitespace characters**

Good Programming Practice 3.1

Consistently applying responsible indentation conventions greatly improves program readability. We suggest a fixed-size tab of about 1/4 inch or three blanks per indent. In this book, we use three blanks per indent.

Good Programming Practice 3.2

Pseudocode is often used to "think out" a program during the program design process. Then the pseudocode program is converted to C.

3.5 The if selection statement

- **Pseudocode statement in C:**

```
if ( grade >= 60 )
   printf( "Passed\n" );
```

  - C code corresponds closely to the pseudocode
- **Diamond symbol (decision symbol)**
  - Indicates decision is to be made
  - Contains an expression that can be `true` or `false`
  - Test the condition, follow appropriate path

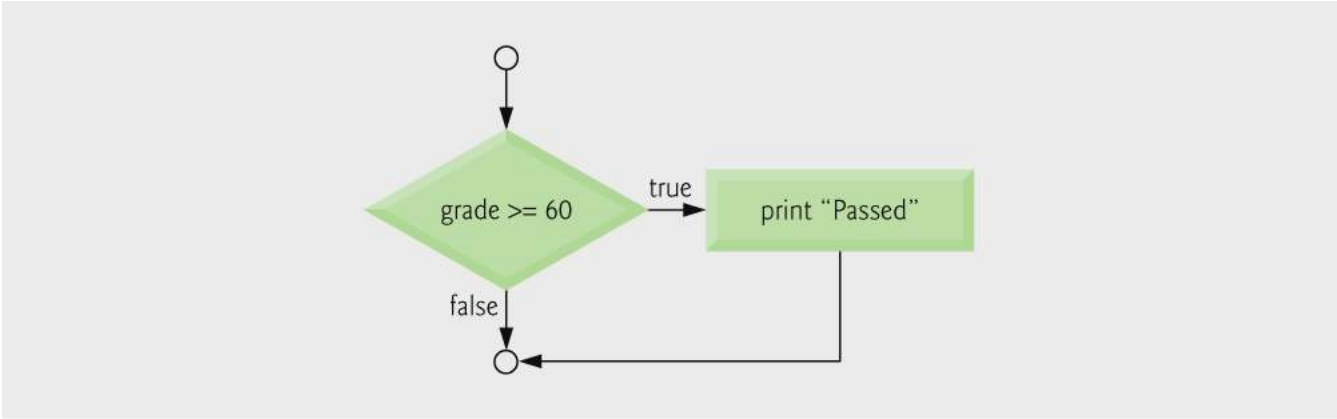

Fig. 3.2 | Flowcharting the single-selection `if` statement.

3.6 The `if…else` selection statement

- `if`
  - **Only performs an action if the condition is `true`**
- `if...else`
  - **Specifies an action to be performed both when the condition is `true` and when it is `false`**
- **Psuedocode:**

  If student's grade is greater than or equal to 60
  Print "Passed"

  else
  Print "Failed"

  - **Note spacing/indentation conventions**

Good Programming Practice 3.3

Indent both body statements of an `if...else` statement.

Good Programming Practice 3.4

If there are several levels of indentation, each level should be indented the same additional amount of space.

3.6 The `if…else` selection statement

- **C code:**

```
if ( grade >= 60 )
   printf( "Passed\n");
else
   printf( "Failed\n");
```

- **Ternary conditional operator (`?:`)**
  - Takes three arguments (condition, value if `true`, value if `false`)
  - Our pseudocode could be written:

```
printf( "%s\n", grade >= 60 ? "Passed" : "Failed" );
```

  - Or it could have been written:

```
grade >= 60 ? printf( “Passed\n” ) : printf( “Failed\n”
  );
```

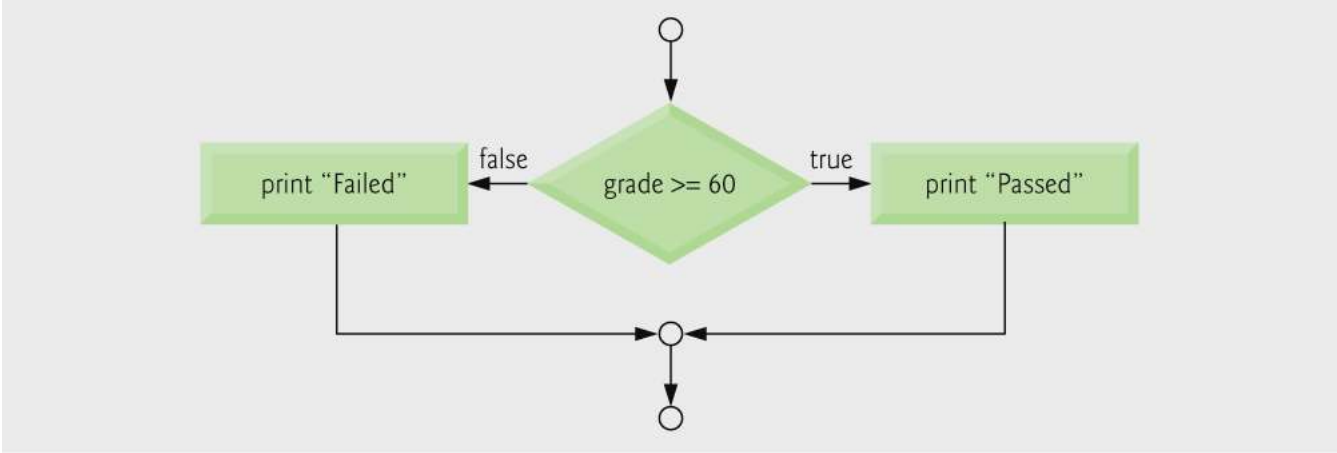

Fig. 3.3 | Flowcharting the double-selection `if...else` statement.

3.6 The `if…else` selection statement

- **Compound statement:**
  - **Set of statements within a pair of braces**
  - **Example:**

```
if ( grade >= 60 )
   printf( "Passed.\n" );
else {
   printf( "Failed.\n" );
   printf( "You must take this course
      again.\n" );
  }
```

  - **Without the braces, the statement**

```
printf( "You must take this course
   again.\n" );
```

would be executed automatically

Software Engineering Observation 3.1

A compound statement can be placed anywhere in a program that a single statement can be placed.

Common Programming Error 3.1

Forgetting one or both of the braces that delimit a compound statement.

3.6 The `if…else` selection statement

- **Block:**
  - **Compound statements with declarations**
- **Syntax errors**
  - **Caught by compiler**
- **Logic errors:**
  - **Have their effect at execution time**
  - **Non-fatal: program runs, but has incorrect output**
  - **Fatal: program exits prematurely**

Common Programming Error 3.2

Placing a semicolon after the condition in an `if` statement as in `if ( grade >= 60 );` leads to a logic error in single-selection `if` statements and a syntax error in double-selection `if` statements.

Error-Prevention Tip 3.1

Typing the beginning and ending braces of compound statements before typing the individual statements within the braces helps avoid omitting one or both of the braces, preventing syntax errors and logic errors (where both braces are indeed required).

Software Engineering Observation 3.2

Just as a compound statement can be placed anywhere a single statement can be placed, it is also possible to have no statement at all, i.e., the empty statement. The empty statement is represented by placing a semicolon (;) where a statement would normally be.