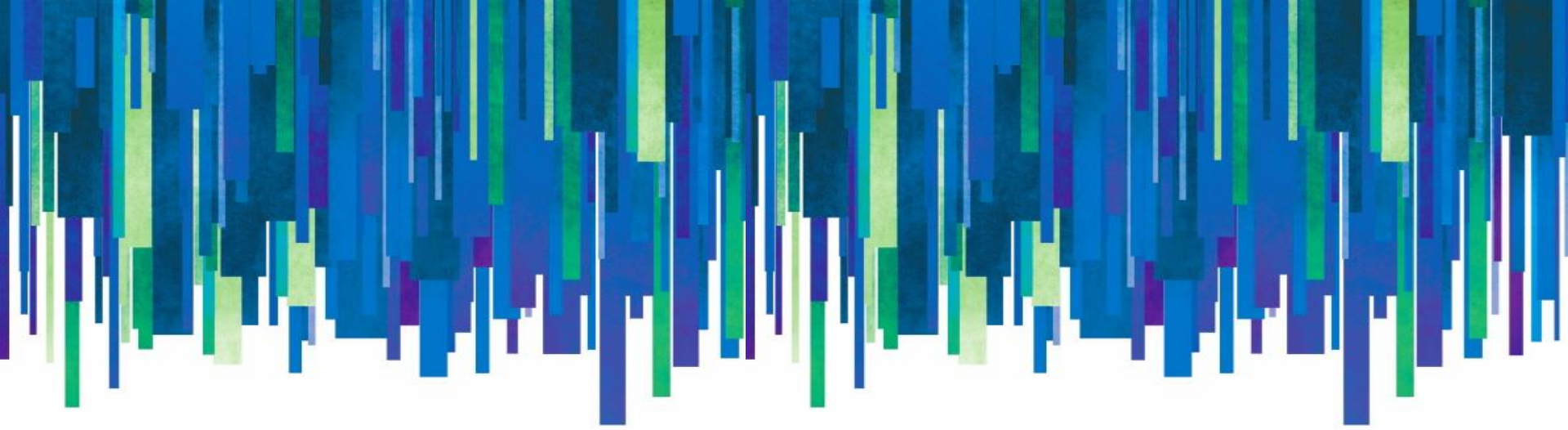

Introduction to Programming in C++
Seventh Edition

Chapter 6:
More on the Selection Structure

Objectives

- Include a nested selection structure in pseudocode and in a flowchart
- Code a nested selection structure
- Recognize common logic errors in selection structures
- Include a multiple-alternative selection structure in pseudocode and in a flowchart
- Code a multiple-alternative selection structure in C++

Making Decisions

- True and false paths of a selection structure can contain other selection structures
- Inner selection structures are referred to as **nested selection structures**; contained (nested) within an outer selection structure
- Nested selection structures are used when more than one decision needs to be made before choosing an instruction
- Inner (nested) selection structures are indented within their outer selection structures

Making Decisions (cont'd.)

Problem specification

Maleek is practicing for an upcoming basketball game. Write the instructions that direct Maleek to shoot the basketball and then say one of two phrases, depending on whether or not the basketball went through the hoop.

Result of shot	Phrase
Basketball went through the hoop	I did it!
Basketball did not go through the hoop	Missed it!


```
1. shoot the basketball
                    condition
2. if (the basketball went through the hoop)
true path     say "I did it!"
   else
false path    say "Missed it!"
   end if
```

Figure 6-1 Problem that requires a selection structure

Making Decisions (cont'd.)

Problem specification

Maleek is practicing for an upcoming basketball game. Write the instructions that direct Maleek to shoot the basketball and then say either one or two of four phrases, depending on whether or not the basketball went through the hoop and also where Maleek was standing when he made the basket.

Result of shot	Phrase
Basketball went through the hoop	I did it!
Maleek made the basket from either inside or on the 3-point line	2 points for me
Maleek made the basket from behind the 3-point line	3 points for me
Basketball did not go through the hoop	Missed it!

```
1. shoot the basketball
2. if (the basketball went through the hoop)
        say "I did it!"
        if (Maleek was either inside or on the 3-point line)
                say "2 points for me"
        else
                say "3 points for me"
        end if
   else
        say "Missed it!"
   end if
```

nested dual-alternative selection structure

outer dual-alternative selection structure

Figure 6-2 Problem that requires a nested selection structure

Making Decisions (cont'd.)

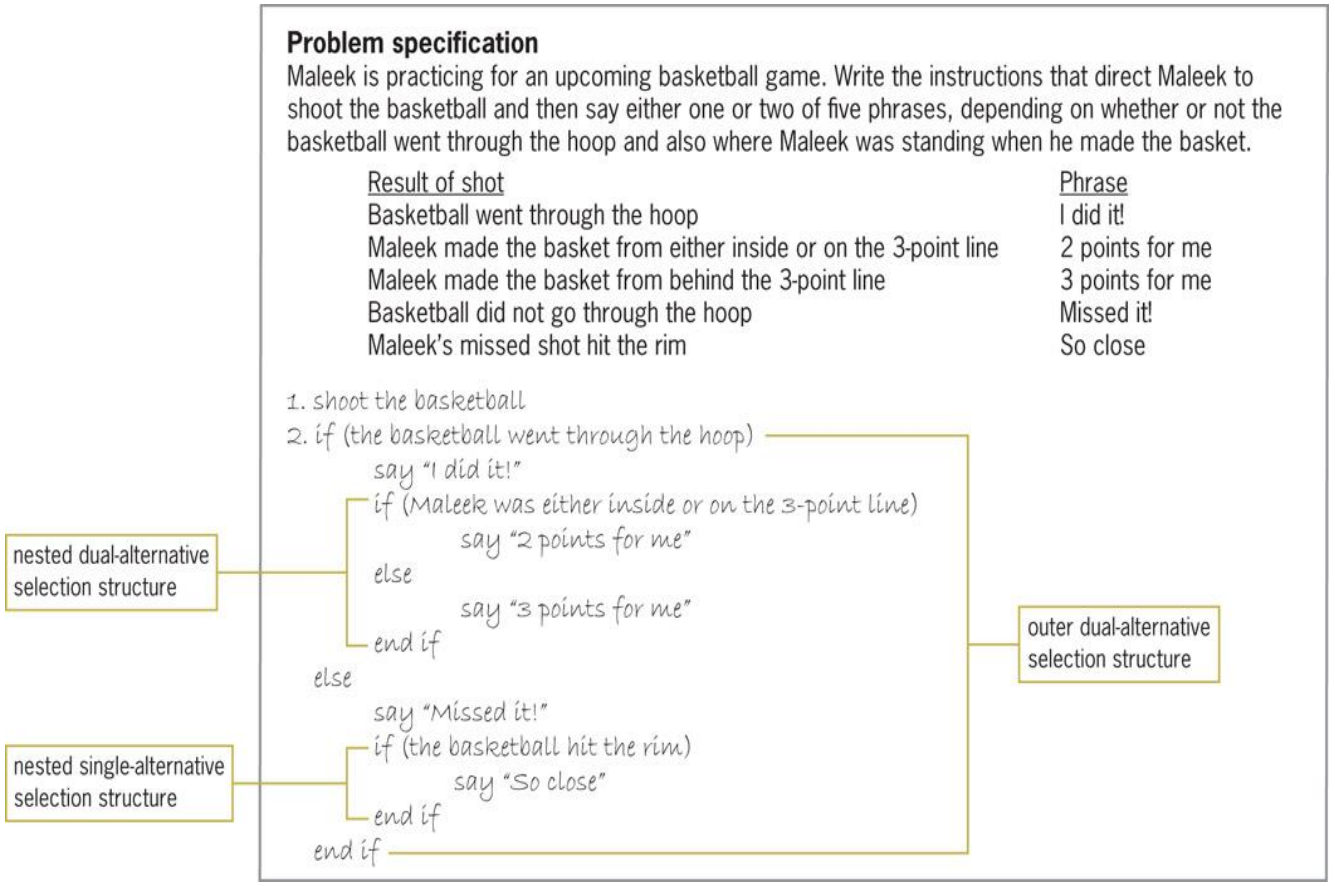

Figure 6-3 Problem that requires two nested selection structures

Flowcharting a Nested Selection Structure

- Outer and inner selection structures can be thought of as making primary and secondary decisions, respectively
- Secondary decision is called such because whether it needs to be made depends on the result of a primary decision

Flowcharting a Nested Selection Structure (cont'd.)

Problem specification

The Danville city manager wants a program that determines voter eligibility and displays one of three messages. The messages and the criteria for displaying each message are as follows:

Message	Criteria
You are too young to vote.	person is younger than 18 years old
You can vote.	person is at least 18 years old and is registered to vote
You must register before you can vote.	person is at least 18 years old but is not registered to vote

Figure 6-6 Problem specification for voter eligibility problem

Flowcharting a Nested Selection Structure (cont'd.)

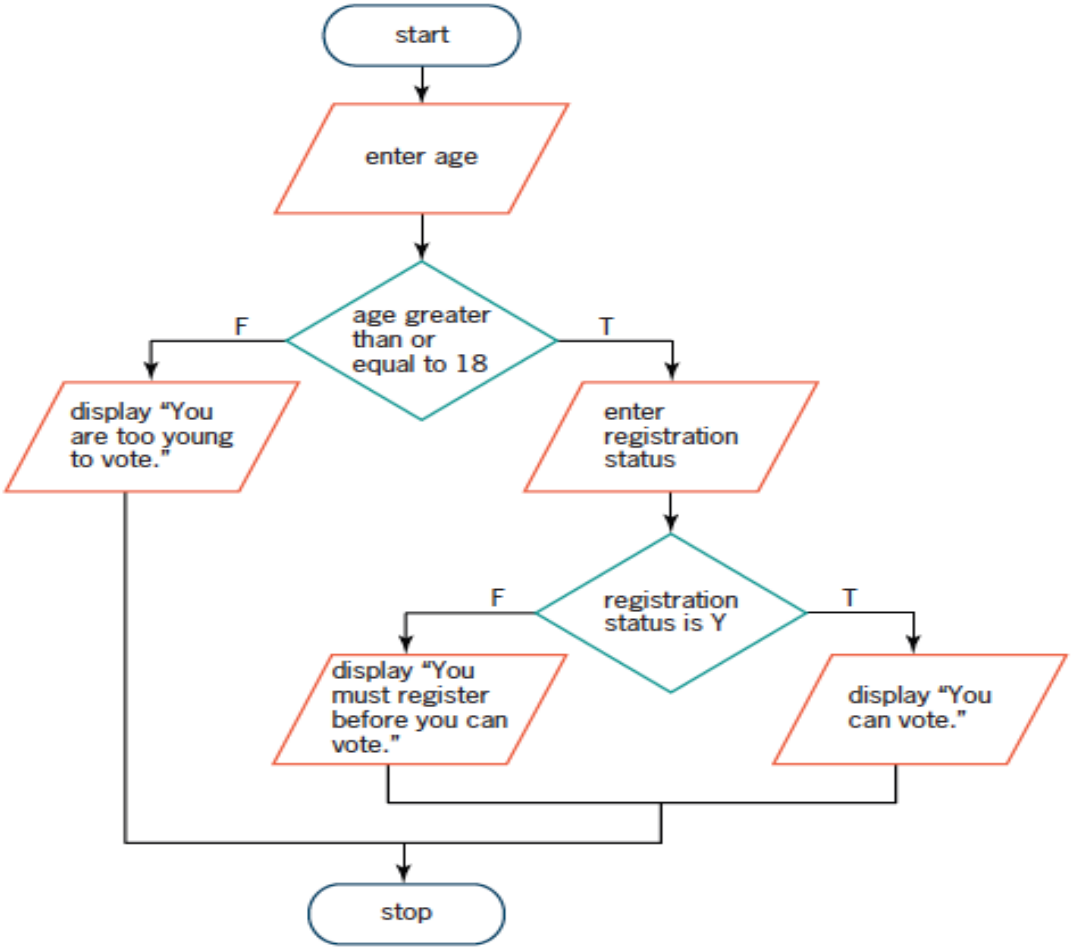

Figure 6-6 A correct solution to the voter eligibility problem

Flowcharting a Nested Selection Structure (cont'd.)

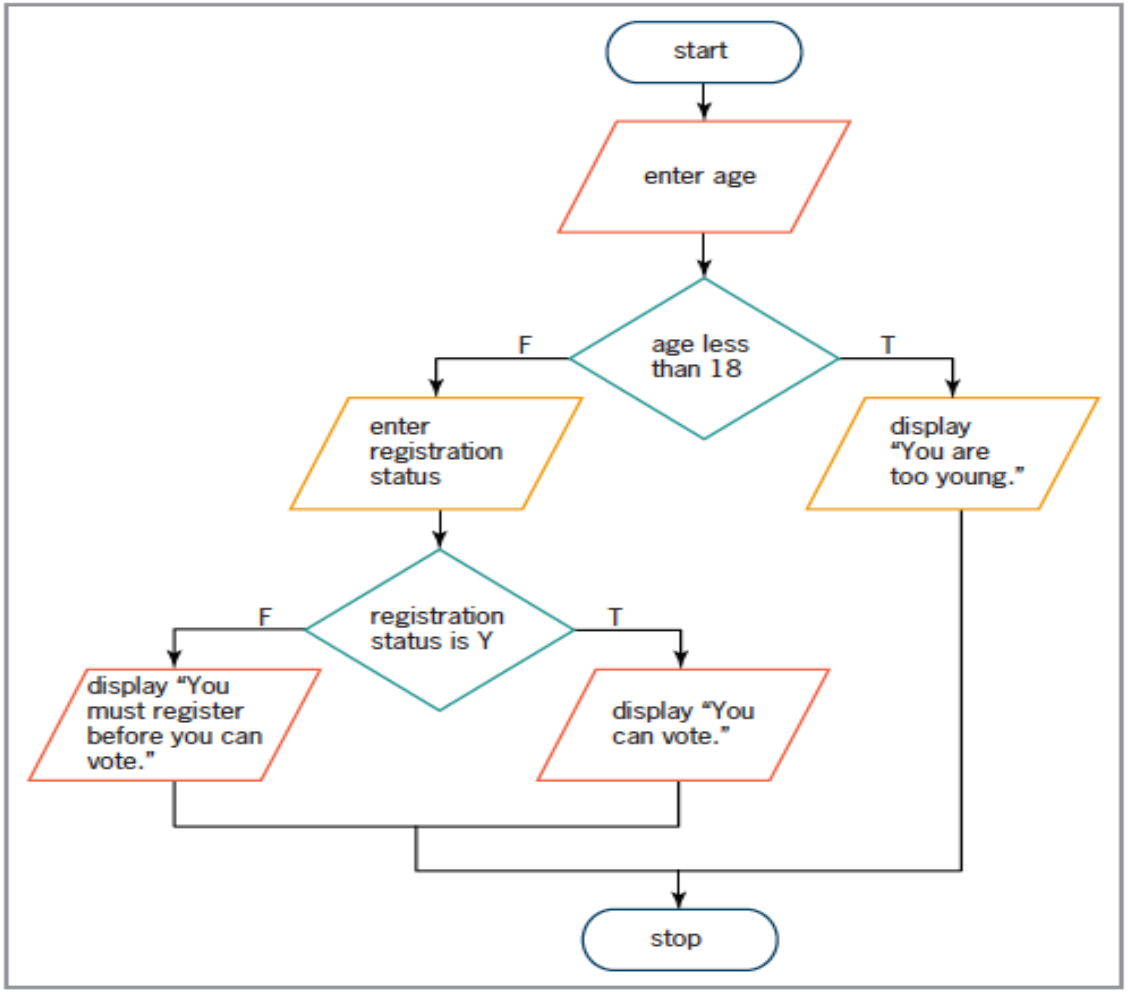

Figure 6-7 Another correct solution to the voter eligibility problem

Coding a Nested Selection Structure

- Code for nested selection structures uses the `if` and `else` statements
- Nested selection structures can be placed in either `if` or `else` statement blocks
- Correct tabbing makes code easier to read

Coding a Nested Selection Structure (cont'd.)

Problem specification

The manager of Willow Springs Health Club wants a program that allows her to enter the number of calories and grams of fat contained in a specific food. The program should calculate and display two values: the food's fat calories and its fat percentage. The fat calories are the number of calories attributed to fat and are calculated by multiplying the food's fat grams by the number 9; this is because each gram of fat contains nine calories. The fat percentage is the ratio of the food's fat calories to its total calories. You calculate the fat percentage by dividing the food's fat calories by its total calories and then multiplying the result by 100. The fat percentage should be displayed with zero decimal places. If the fat percentage is greater than 30%, the program should display the message "High in fat"; otherwise, it should display the message "Not high in fat". The program should display an appropriate error message if either or both input values are less than 0.

```
 1  //Fig6-8.cpp - displays a food's fat
 2  //calories and fat percentage
 3  //Created/revised by <your name> on <current date>
 4
 5  #include <iostream>
 6  #include <iomanip>
 7  using namespace std;
 8
 9  int main()
10  {
11      //declare variables
12      int totalCals      = 0;
13      int fatGrams       = 0;
14      int fatCals        = 0;
15      double fatPercent = 0.0;
16
17      //enter input items
18      cout << "Total calories: ";
19      cin >> totalCals;
20      cout << "Grams of fat: ";
21      cin >> fatGrams;
22
23      //determine whether the data is valid
24      if (totalCals >= 0 && fatGrams >= 0)
25      {
26            //calculate and display the output
27            fatCals = fatGrams * 9;
28            fatPercent = static_cast<double>(fatCals)
29                  / static_cast<double>(totalCals) * 100;
30
31            cout << "Fat calories: " << fatCals << endl;
32            cout << fixed << setprecision(0);
33            cout << "Fat percentage: " << fatPercent
34                 << "%" << endl;
```

Figure 6-8 Modified problem specification for the health club problem from Chapter 5's Lab 5-2

Coding a Nested Selection Structure (cont'd.)

```
35          if (fatPercent > 30.0)
36              cout << "High in fat" << endl;
37          else
38              cout << "Not high in fat" << endl;
39          //end if
40      }
41      else
42          cout << "Input error" << endl;
43      //end if
44
45      //system("pause");
46      return 0;
47  }  //end of main function
```

nested selection structure

your C++ development tool may require this statement

Figure 6-8 Modified problem specification for the health club problem from Chapter 5's Lab 5-2 (cont'd)

Coding a Nested Selection Structure (cont'd.)

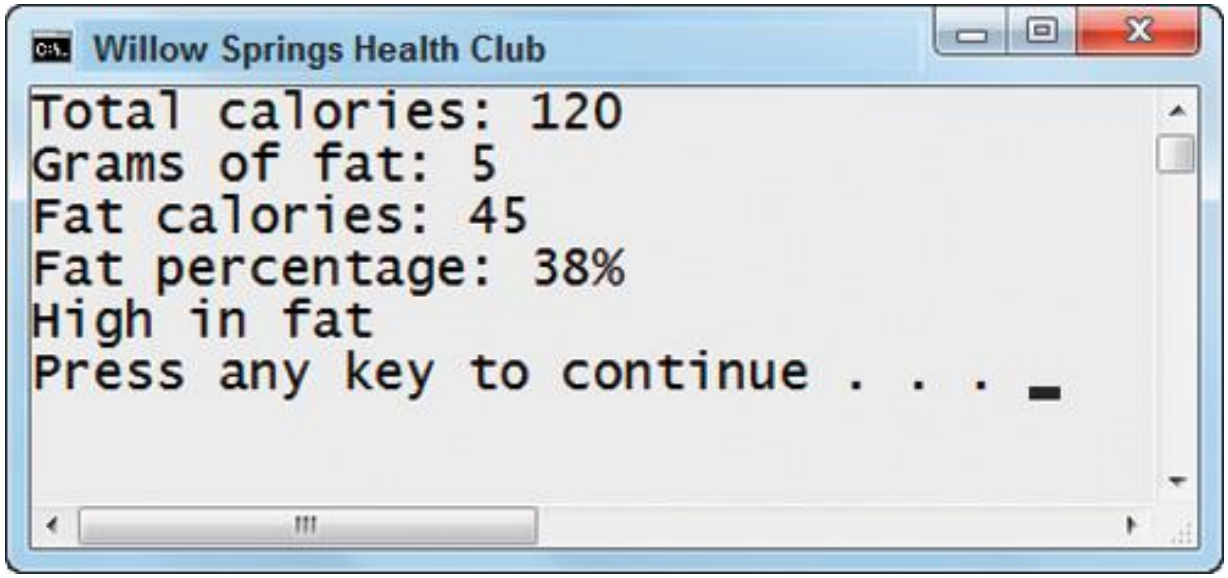

Figure 6-9 Sample run of the modified health club program

Coding a Nested Selection Structure (cont'd.)

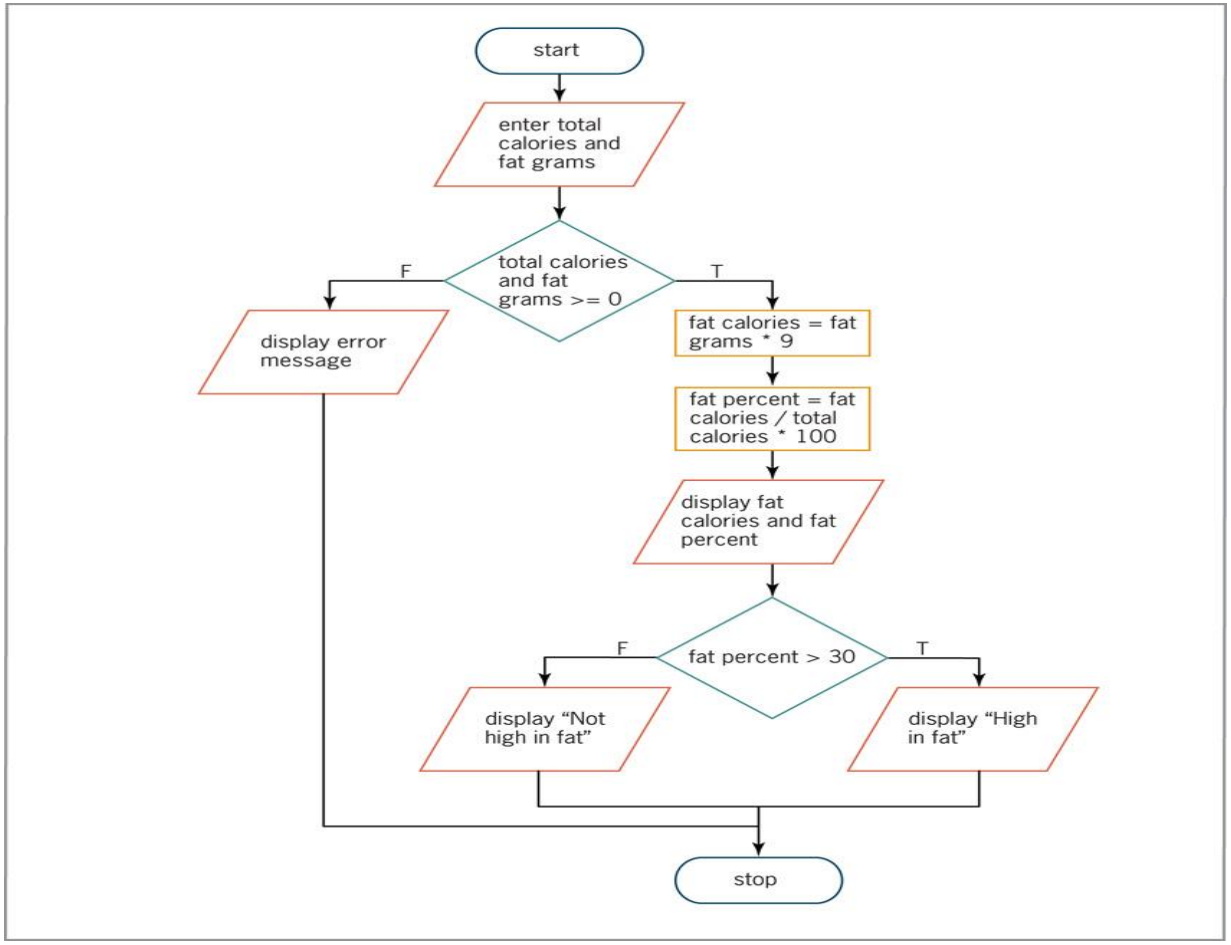

Figure 6-10 Flowchart for the modified health club program

Logic Errors in Selection Structures

- Three common logic errors made when writing selection structures
  - Using a compound condition rather than a nested selection structure
  - Reversing the outer and nested decisions
  - Using an unnecessary nested selection structure

Logic Errors in Selection Structures (cont'd.)

```
1. enter the code and sales
2. calculate the bonus by multiplying the sales by .08
3. if (the code is X)
        if (the sales are greater than or equal to 10000)
                add 150 to the bonus
        else
                add 125 to the bonus
        end if
   end if
4. display the bonus
```

Figure 6-11 A correct algorithm for the bonus problem

Logic Errors in Selection Structures (cont'd.)

Code	Sales ($)	Bonus ($)
X	15000	1350
X	9000	845
A	13000	1040

Figure 6-12 Test data and manually calculated results

code	sales	bonus
X	15000	1200

Figure 6-13 Current status of the desk-check table

code	sales	bonus
X	15000	~~1200~~
		1350

Figure 6-14 Desk-check table after completing the first desk-check

Logic Errors in Selection Structures (cont'd.)

code	sales	bonus
~~X~~	~~15000~~	~~1200~~
		~~1350~~
X	9000	~~720~~
		845

Figure 6-15 Desk-check table after completing the second desk-check

code	sales	bonus
~~X~~	~~15000~~	~~1200~~
		~~1350~~
~~X~~	~~9000~~	~~720~~
		~~845~~
A	13000	1040

Figure 6-16 Desk-check table after completing the third desk-check

First Logic Error

- Using a compound condition rather than a nested selection structure
- Ignores the hierarchy between two sub-conditions – One applies only if the other is a certain value

First Logic Error (cont'd.)

Correct algorithm

```
1. enter the code and sales
2. calculate the bonus by multiplying
   the sales by .08
3. if (the code is X)
        if (the sales are greater than or
        equal to 10000)
                add 150 to the bonus
        else
                add 125 to the bonus
        end if
   end if
4. display the bonus
```

Incorrect algorithm

```
1. enter the code and sales
2. calculate the bonus by multiplying
   the sales by .08
3. if (the code is X and the sales are greater
   than or equal to 10000)
        add 150 to the bonus
   else
        add 125 to the bonus
   end if
4. display the bonus
```

uses a compound condition instead of a nested selection structure

Figure 6-17 Correct algorithm and incorrect algorithm containing the first logic error

First Logic Error (cont'd.)

code	sales	bonus	
~~X~~	~~15000~~	~~1200~~	
		~~1350~~	(correct result for the first desk-check)
~~X~~	~~9000~~	~~720~~	
		~~845~~	(correct result for the second desk-check)
A	13000	~~1040~~	
		1165	(incorrect result for the third desk-check)

Figure 6-18 Desk-check table for the incorrect algorithm in Figure 6-17

Second Logic Error

- Reversing outer and nested selection structures

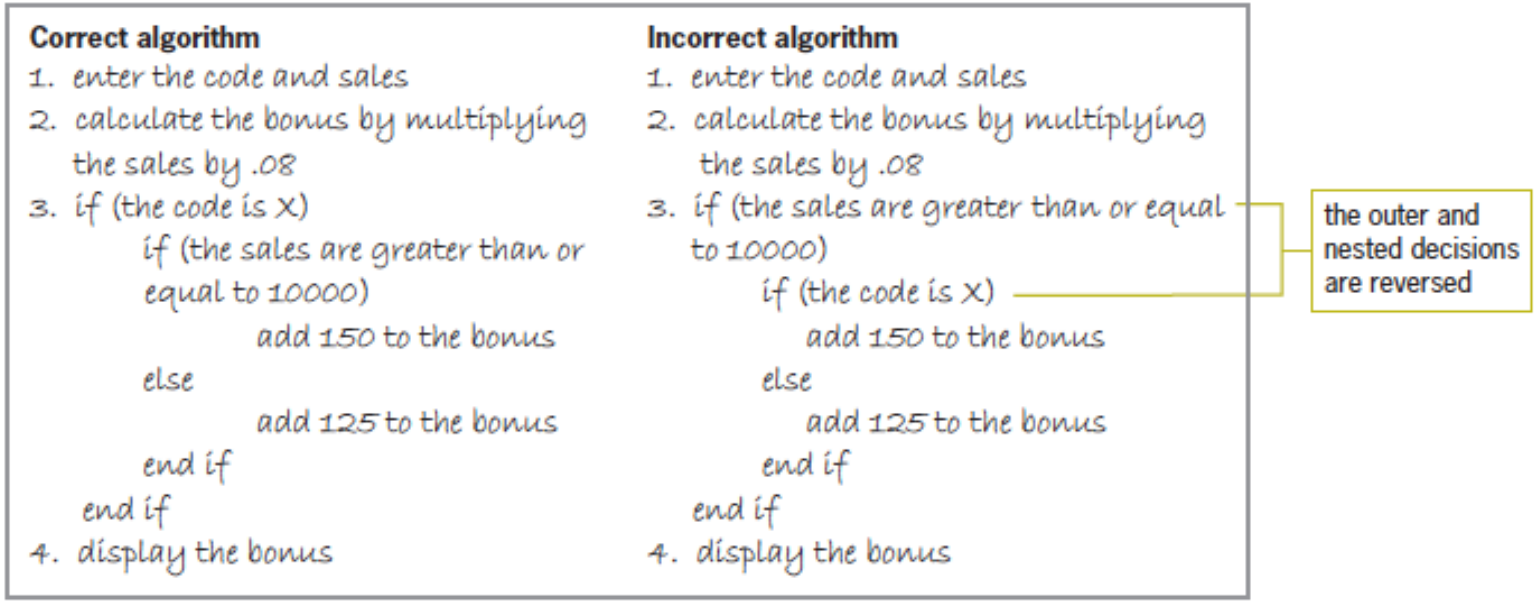

Figure 6-19 Correct algorithm and an incorrect algorithm containing the second logic error

Second Logic Error (cont'd.)

code	sales	bonus	
~~X~~	~~15000~~	~~1200~~	
		~~1350~~	(correct result for the first desk-check)
~~X~~	~~9000~~	~~720~~	(incorrect result for the second desk-check)
A	13000	~~1040~~	
		1165	(incorrect result for the third desk-check)

Figure 6-20 Desk-check table for the incorrect algorithm in Figure 6-19

Third Logic Error

- Using an unnecessary nested selection structure
- Often will produce the correct result, but will be inefficient

Third Logic Error (cont'd.)

Correct algorithm

```
1. enter the code and sales
2. calculate the bonus by multiplying
   the sales by .08
3. if (the code is X)
        if (the sales are greater than or
        equal to 10000)
            add 150 to the bonus
        else
            add 125 to the bonus
        end if
   end if
4. display the bonus
```

Inefficient algorithm

```
1. enter the code and sales
2. calculate the bonus by multiplying
   the sales by .08
3. if (the code is X)
        if (the sales are greater than or
        equal to 10000)
            add 150 to the bonus
        else
            if (the sales are less than 10000)
                add 125 to the bonus
            end if
        end if
   end if
4. display the bonus
```

unnecessary nested selection structure

Figure 6-21 Correct algorithm and an incorrect algorithm containing the third logic error

Third Logic Error (cont'd.)

code	sales	bonus	
~~X~~	~~15000~~	~~1200~~	
		~~1350~~	(correct result for the first desk-check)
~~X~~	~~9000~~	~~720~~	
		~~845~~	(correct but inefficient result for the second desk-check)
A	13000	1040	(correct result for the third desk-check)

Figure 6-22 Desk-check table for inefficient algorithm in Figure 6-21

Multiple-Alternative Selection Structures

- Sometimes problems require a selection structure that chooses between several alternatives
- Called **multiple-alternative selection structures** or **extended selection structures**
- In a flowchart, diamond symbol is used; has multiple flowlines leading out, not just two
- Each flowline represents a possible path, marked with the value that represents that path
- `if/else` statements can be used to implement it; uses multiple `if else` clauses

Multiple-Alternative Selection Structures (cont'd.)

Problem specification

Mr. Jacoby teaches math at Kindlon High School. He wants a program that displays a message based on a letter grade he enters. The valid letter grades and their corresponding messages are shown below. If the letter grade is not valid, the program should display the "Invalid grade" message.

Letter grade	Message
A	Excellent
B	Above Average
C	Average
D	Below Average
F	Below Average

Figure 6-25 Problem specification for Kindlon High School problem

Multiple-Alternative Selection Structures (cont'd.)

Input	Processing	Output
grade	Processing items: none	message

```
Algorithm:
1. enter the grade
2. if (the grade is one of the following:)
        A       display "Excellent" message
        B       display "Above Average" message
        C       display "Average" message
        D or F  display "Below Average" message
   else
        display "Invalid grade"
   end if
```

Figure 6-25 IPO chart for the Kindlon High School problem

Multiple-Alternative Selection Structures (cont'd.)

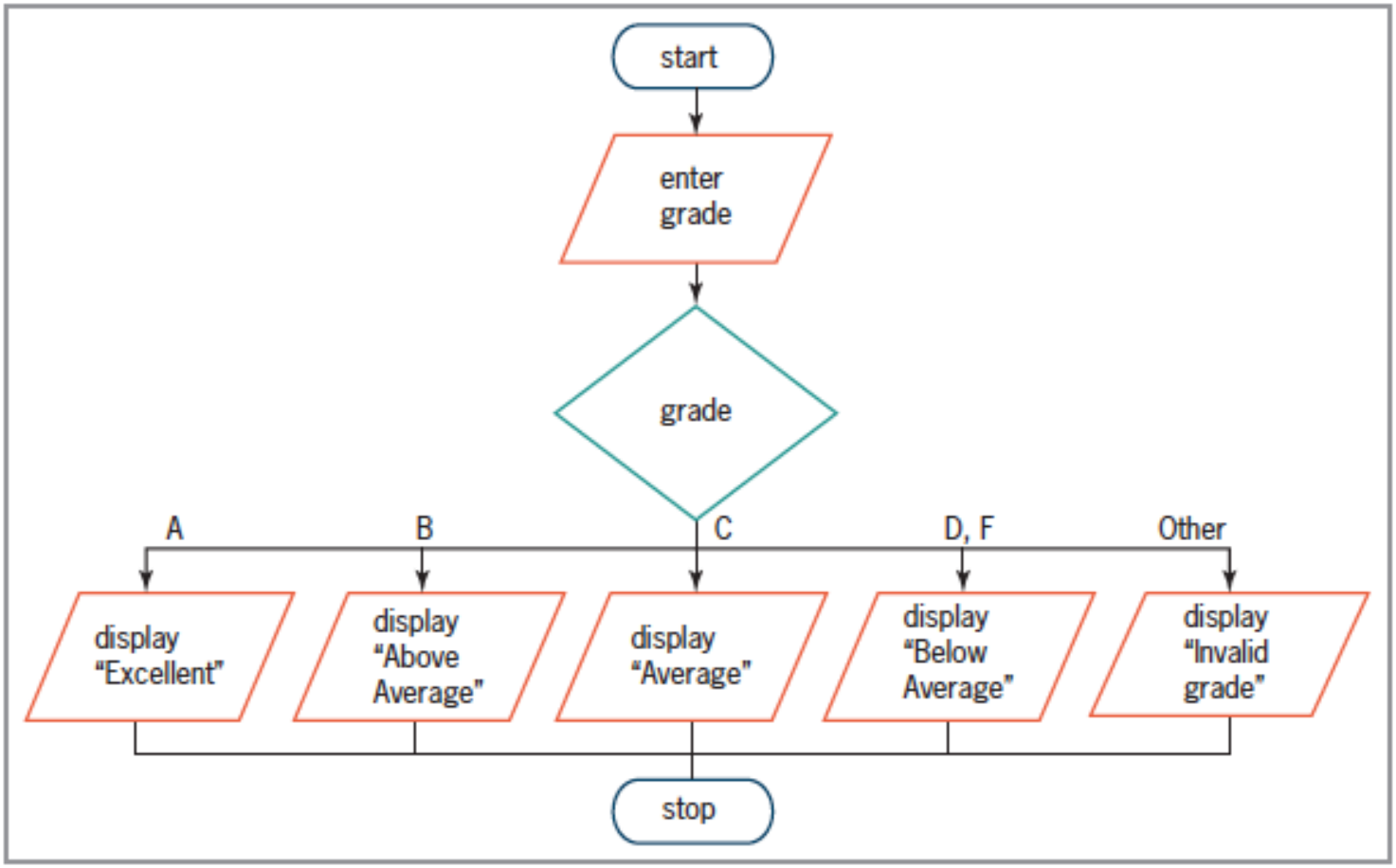

Figure 6-26 Flowchart for the Kindlon High School problem

Multiple-Alternative Selection Structures (cont'd.)

Example 1

```
grade = toupper(grade);
if (grade == 'A')
      cout << "Excellent";
else
      if (grade == 'B')
            cout << "Above Average";
      else
            if (grade == 'C')
                  cout << "Average";
            else
                  if (grade == 'D' || grade == 'F')
                        cout << "Below Average";
                  else   //default
                         cout << "Invalid grade";
                  //end if
            //end if
      //end if
//end if
```

Example 2

```
grade = toupper(grade);
if (grade == 'A')
     cout << "Excellent";
else if (grade == 'B')
     cout << "Above Average";
else if (grade == 'C')
     cout << "Average";
else if (grade == 'D' || grade == 'F')
     cout << "Below Average";
else //default
     cout << "Invalid grade";
//end if
```

you get here when the grade is not A

you get here when the grade is not A and not B

you get here when the grade is not A, B, or C

you get here when the grade is not A, B, C, D, or F

you can use one comment to mark the end of the entire structure

Figure 6-27 Two ways of coding the multiple-alternative selection structure from Figures 6-25 and 6-26

The `switch` Statement

- Can sometimes use the **`switch` statement** to code a multiple-alternative selection structure
- Statement begins with `switch` keyword followed by a selector expression in parentheses
- Selector expression can contain any combination of variables, constants, functions, and operators
- Must result in a data type that is `bool`, `char`, `short`, `int`, or `long`
- Between opening and closing braces (after selector expression), there are one or more `case` clauses

The `switch` Statement (cont'd.)

- Each `case` clause represents a different alternative and contains a value followed by a colon
- Can include as many `case` clauses as necessary
- Value for each `case` clause can be a literal constant, named constant, or an expression composed of literal and named constants
- Data type of the value should be the same data type as the selector expression

The `switch` Statement (cont'd.)

- Each case clause contains one or more statements processed when selector expression matches that `case`'s value
- **`break` statement** tells computer to break out of `switch` at that point; must be the last statement of a case clause
- Without a break statement, computer continues to process instructions in later case clauses
- After processing `break`, computer processes next instruction after `switch` statement's closing brace

The `switch` Statement (cont'd.)

- Good programming practice to document end of `switch` with a comment (`//end switch`)
- Can also include one `default` clause; processed if selector expression does not match any values in `case` clauses
- `default` clause can appear anywhere, but usually entered as last clause
  - If it is the last clause, a `break` statement is not needed at its end
  - Otherwise, a `break` statement is needed to prevent computer from processing later case clauses

The `switch` Statement (cont'd.)

HOW TO Use the `switch` Statement

Syntax

```
switch (selectorExpression)
{
case value1:
        one or more statements
        [break;]
[case value2:
        one or more statements
        [break;]]
[case valueN:
        one or more statements
        [break;]]
[default:
        one or more statements to be processed when the selector-
        Expression does not match any of the values in the case clauses
        [break;]]
}       //end switch
```

Figure 6-28 How to use the `switch` statement

The `switch` Statement (cont'd.)

Example

```
int main()
{
    char grade = ' ';

    //enter grade
    cout << "Letter grade: ";
    cin >> grade;
    grade = toupper(grade);

    switch (grade)
    {
    case 'A':
        cout << "Excellent";
        break;
    case 'B':
        cout << "Above Average";
        break;
    case 'C':
        cout << "Average";
        break;
    case 'D':
    case 'F':
        cout << "Below Average";
        break;
    default:
        cout << "Invalid grade";
    }   //end switch

    cout << endl;
    system("pause");
    return 0;
}   //end of main function
```

Example similar to code in Figure 6-28

The `switch` Statement (cont'd.)

Problem specification

The sales manager at Warren Company wants a program that displays a price based on a product ID she enters. The valid product IDs and their corresponding prices are shown here. If the product ID is not valid, the program should display the "Invalid product ID" message.

Product ID	Price
1	50.55
2	12.35
5	11.46
7	11.46
9	12.35
11	11.46

Figure 6-29 Problem specification for the Warren Company problem

The `switch` Statement (cont'd.)

IPO chart information

Input

product ID

Processing

none

Output

price

Algorithm

1. enter the product ID

2. if (the product ID is one of the following:)
   - 1 — assign 50.55 as the price
   - 2 or 9 — assign 12.35 as the price
   - 5, 7, or 11 — assign 11.46 as the price

   else
   assign -1 as the price
   end if

3. if (the price is -1)
   display "Invalid product ID" message
   else
   display the price
   end if

C++ instructions

```
int productId = 0;
double price = 0.0;

cout << "Product ID (1, 2, 5,
7, 9, or 11): ";
cin >> productId;

switch (productId)
{
    case 1:
        price = 50.55;
        break;
    case 2:
    case 9:
        price = 12.35;
        break;
    case 5:
    case 7:
    case 11:
        price = 11.46;
        break;
    default:
        price = -1;
} //end switch

if (price == -1)
    cout << "Invalid product ID"
    << endl;
else
    cout << "Price: $" << price
    << endl;
//end if
```

Figure 6-29 IPO chart and C++ instructions for the Warren Company problem

The `switch` Statement (cont'd.)

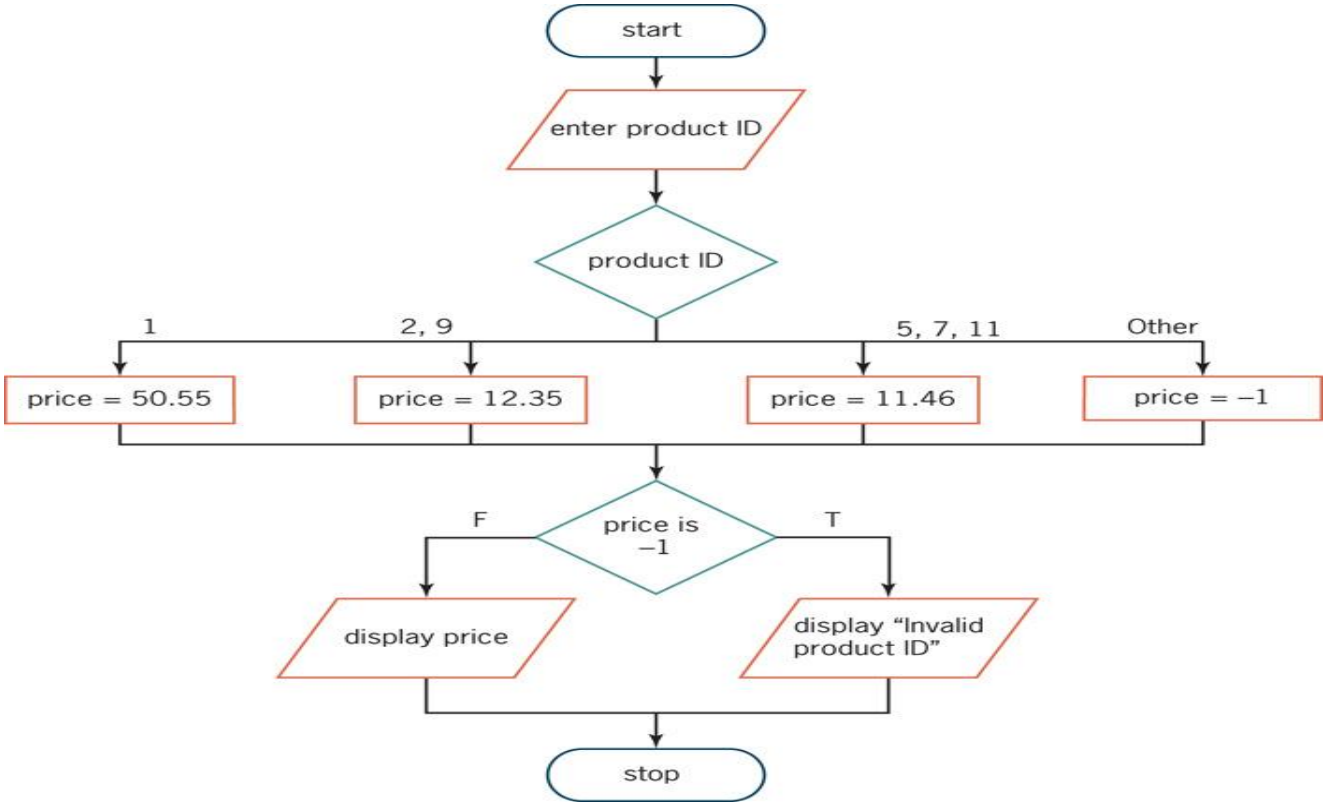

Figure 6-30 Flowchart for the Warren Company problem

Summary

- Can nest a selection structure within true or false path of another selection structure
- Three common logic errors when writing selection structures
  - Using a compound condition instead of a nested selection structure
  - Reversing the inner and outer selection structures
  - Using an unnecessary nested selection structure

Summary (cont'd.)

- Some solutions require selection structures that choose from multiple alternatives; called multiple-alternative or extended selection structures
- Can code these either with `if/else` statements or the `switch` statement
- Diamond is used to represent multiple-alternative selection structures in a flowchart
- Has multiple flowlines leading out; each representing a possible path and marked with appropriate values

Summary (cont'd.)

- In a `switch` statement, the data type of the value in each `case` clause must be compatible with data type of selector expression
- Selector expression must evaluate to value of type `bool`, `char`, `short`, `int`, or `long`
- Most `case` clauses contain a `break` statement; tells the computer to leave the `switch` statement
- Good practice to mark end of `switch` statement with a comment (`//end switch`)

Lab 6-1: Stop and Analyze

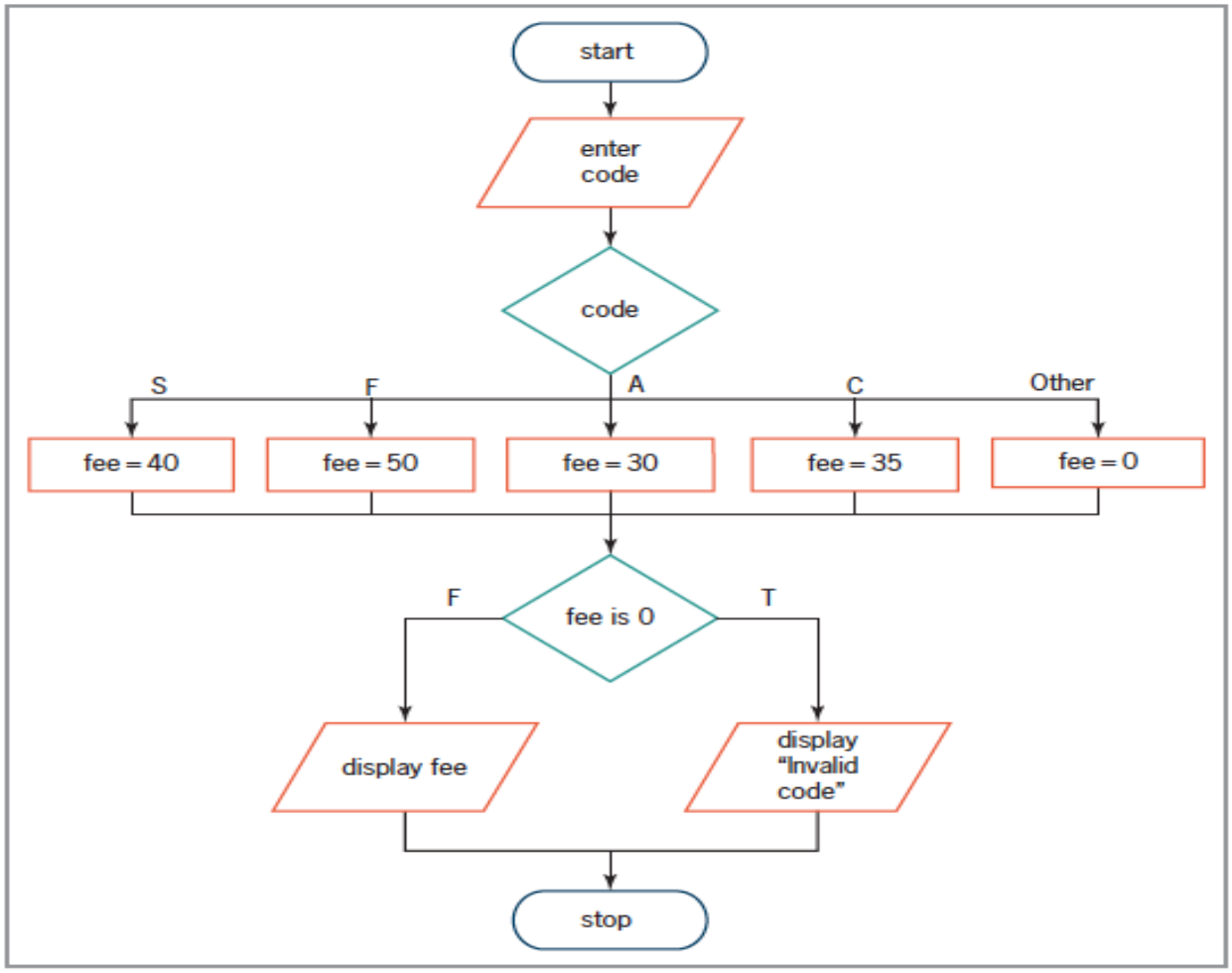

Figure 6-31 Flowchart for Lab 6-1

Lab 6-2: Plan and Create

Problem specification

Jennifer Yardley is the owner of Golf Pro, a U.S. company that sells golf equipment both domestically and abroad. She wants a program that displays the amount of a salesperson's commission. A commission is a percentage of the sales made by the salesperson. Some companies use a fixed rate to calculate the commission, while others (like Golf Pro) use a rate that varies with the amount of sales.

Golf Pro's commission schedule is shown below, along with examples of using the schedule to calculate the commission on three different sales amounts. Notice that the commission for each range in the schedule is calculated differently. The commission for sales in the first range is calculated by multiplying the sales by 2%. As Example 1 shows, if the sales are $15,000, the commission is $300. The commission for sales in the second range is calculated by multiplying the sales over $100,000 by 5% and then adding $2,000 to the result. As Example 2 shows, if the sales are $250,000, the commission is $9,500. The commission for sales starting at $400,001 is calculated by multiplying the sales over $400,000 by 10%, and then adding $17,000 to the result. Example 3 indicates that the commission for sales of $500,000 is $27,000.

If the sales do not fall in any of the sales ranges (in other words, they are less than 0), the program should display the message "The sales cannot be less than 0."

Sales range	Commission
$0 – 100,000	multiply the sales by 2%
$100,001 – 400,000	multiply the sales over 100,000 by 5% and then add 2,000 to the result
$400,001 and over	multiply the sales over 400,000 by 10% and then add 17,000 to the result

Example 1

Sales: $15,000
Commission: 15,000 * .02 = 300

Example 2

Sales: $250,000
Commission: (250,000 – 100,000) * .05 + 2,000 = 9,500

Example 3

Sales: $500,000
Commission: (500,000 – 400,000) * .1 + 17,000 = 27,000

Figure 6-32 Problem specification and calculation examples for Lab 6-2

Lab 6-3: Modify

- Modify the program in Lab 6-2 to calculate commission based on information in Figure 6-38

Code	Commission
1	multiply the sales by 2%
2	multiply the sales over 100,000 by 5% and then add 2000 to the result
3	multiply the sales over 400,000 by 10% and then add 17000 to the result

Figure 6-38 Problem specification for Lab 6-3

- If the sales are less than zero, display "The sales cannot be less than zero."
- If the code is not 1, 2, or 3, display "Invalid Code"

Lab 6-4: Desk-Check

```
//declare variable
int number = 0;

//enter input item
cout << "Enter a number: ";
cin >> number;

//perform calculations
switch (number)
{
      case 1:
      case 2:
      case 3:
            number = number * 2;
            break;
      case 4:
      case 5:
            number = number + 5;
            break;
      default:
            number = number - 50;
}     //end switch

//display number
cout << "Final number: " << number << endl;
```

Figure 6-39 Code for Lab 6-4

Lab 6-5: Debug

- Follow the instructions for starting C++ and opening the Lab6-5.cpp file
- Test the program using codes 1, 2, 3, 4, 5, 9, and -3
- Debug the program