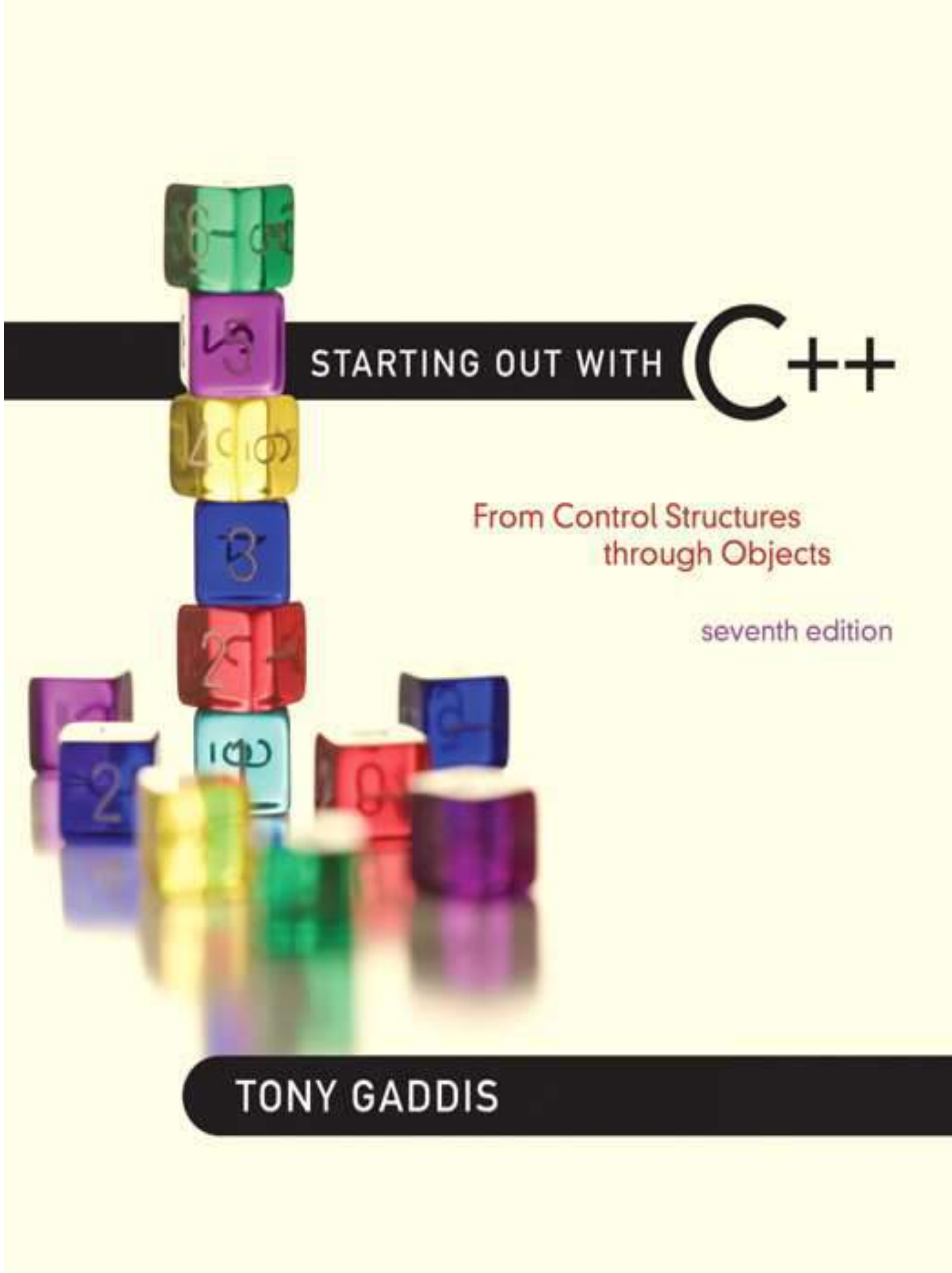
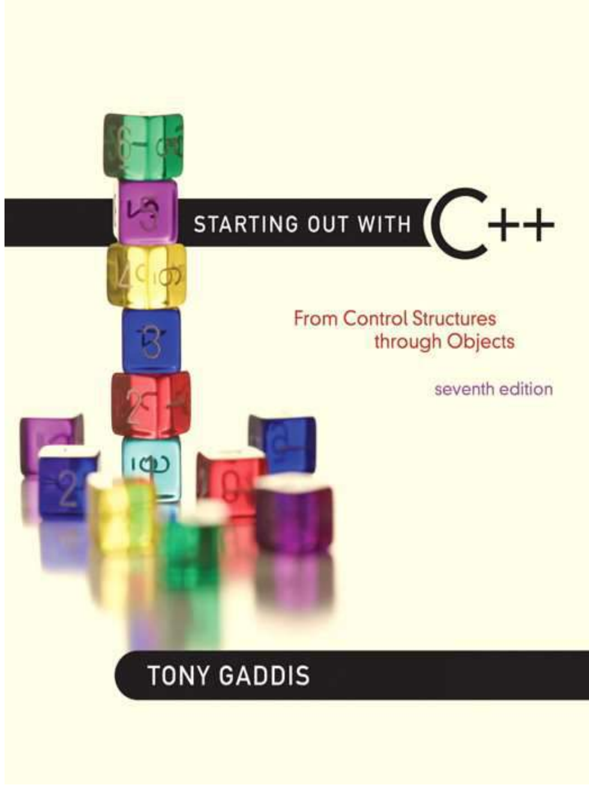
Chapter 4:

Making Decisions



Addison-Wesley is an imprint of PEARSON



4.1

Relational Operators

Relational Operators

- Used to compare numbers to determine relative order
- Operators:

>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to
==	Equal to
!=	Not equal to

Relational Expressions

- Boolean expressions – `true` or `false`
- Examples:

  `12 > 5` is `true`

  `7 <= 5` is `false`

  if `x` is 10, then

  `x == 10` is `true`,

  `x != 8` is `true`, and

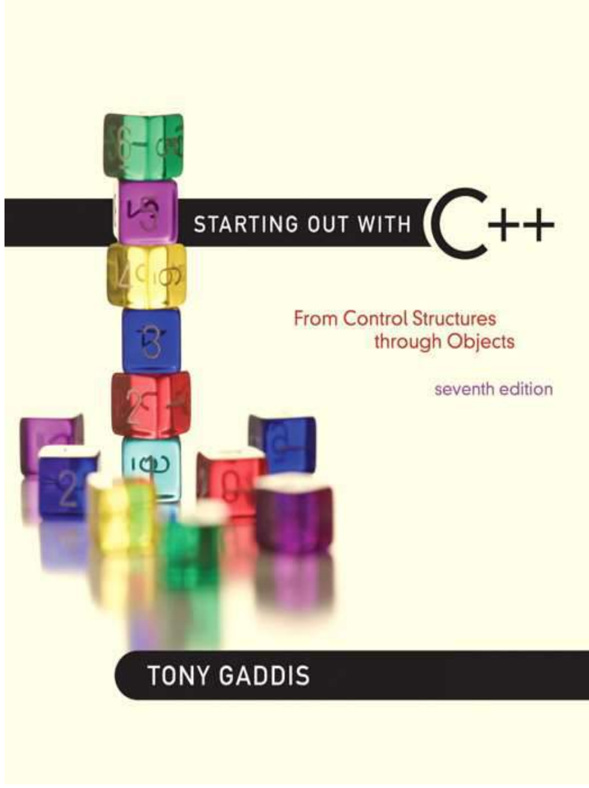
  `x == 8` is `false`

Relational Expressions

- Can be assigned to a variable:

```
result = x <= y;
```

- Assigns `0` for `false`, `1` for `true`
- Do not confuse = and ==



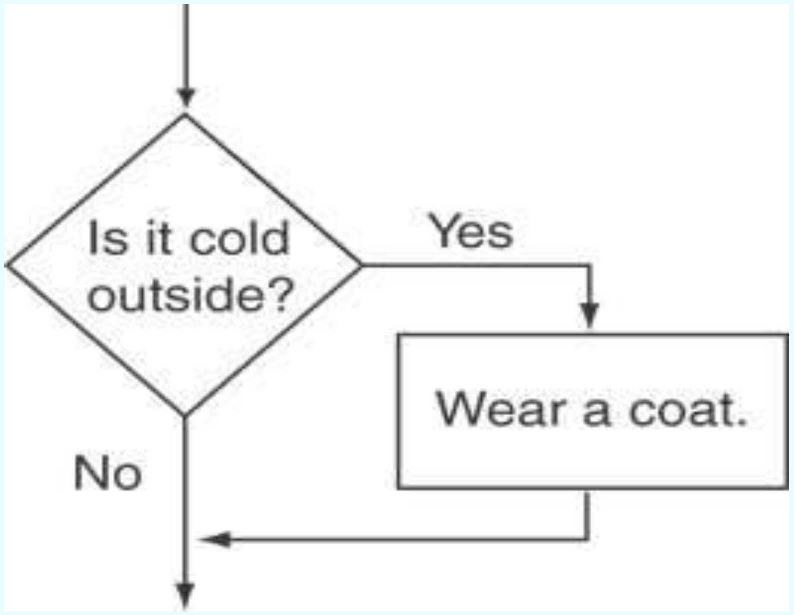
4.2

The `if` Statement

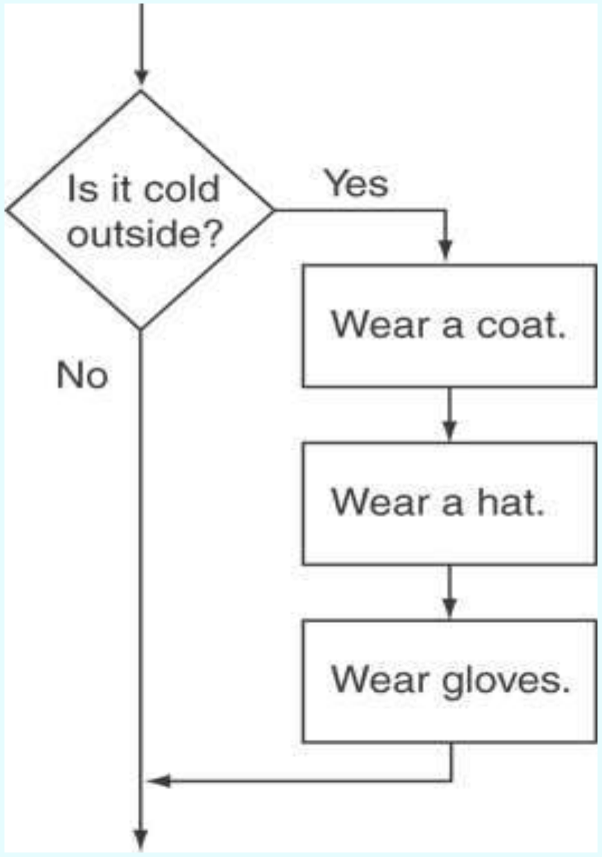
The `if` Statement

- Allows statements to be conditionally executed or skipped over
- Models the way we mentally evaluate situations:
  - "If it is raining, take an umbrella."
  - "If it is cold outside, wear a coat."

Flowchart for Evaluating a Decision



Flowchart for Evaluating a Decision



The **`if`** Statement

- General Format:

```
    if (expression)
          statement;
```

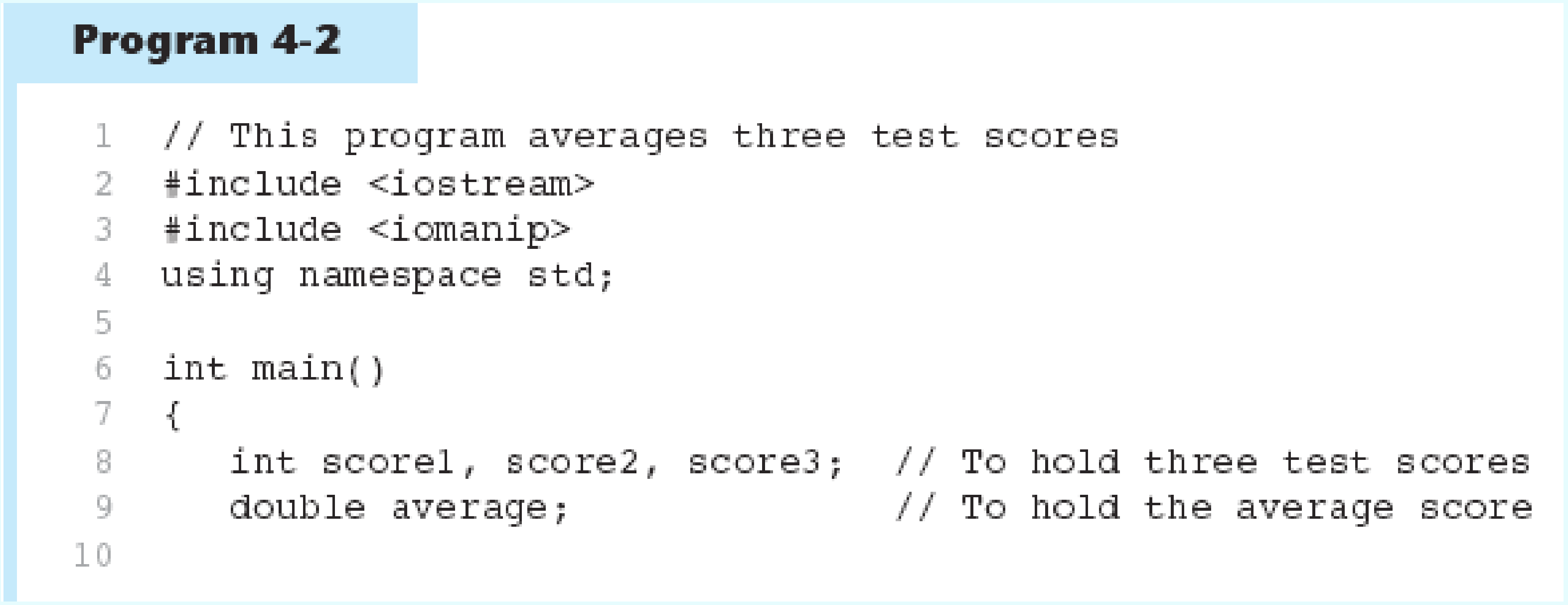
The if Statement-What Happens

To evaluate:

```
   if (expression)
       statement;
```

- If the *`expression`* is `true`, then *`statement`* is executed.
- If the *`expression`* is `false`, then *`statement`* is skipped.

`if` Statement in Program 4-2

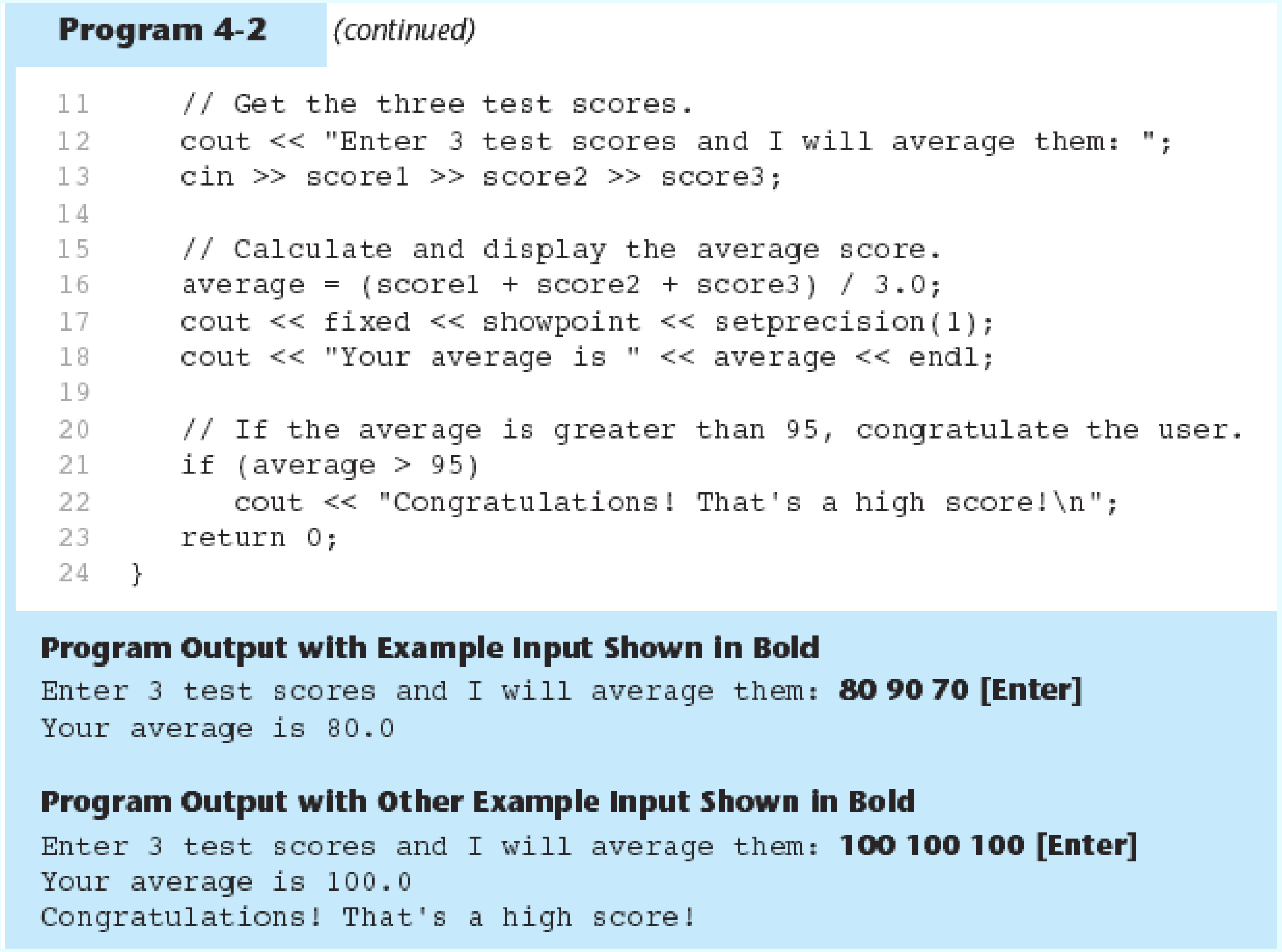


Program 4-2

```
// This program averages three test scores
#include <iostream>
#include <iomanip>
using namespace std;

int main()
{
   int score1, score2, score3;   // To hold three test scores
   double average;               // To hold the average score

```

Continued…

if Statement in Program 4-2



Program 4-2 *(continued)*

```
   // Get the three test scores.
   cout << "Enter 3 test scores and I will average them: ";
   cin >> score1 >> score2 >> score3;

   // Calculate and display the average score.
   average = (score1 + score2 + score3) / 3.0;
   cout << fixed << showpoint << setprecision(1);
   cout << "Your average is " << average << endl;

   // If the average is greater than 95, congratulate the user.
   if (average > 95)
      cout << "Congratulations! That's a high score!\n";
   return 0;
}
```

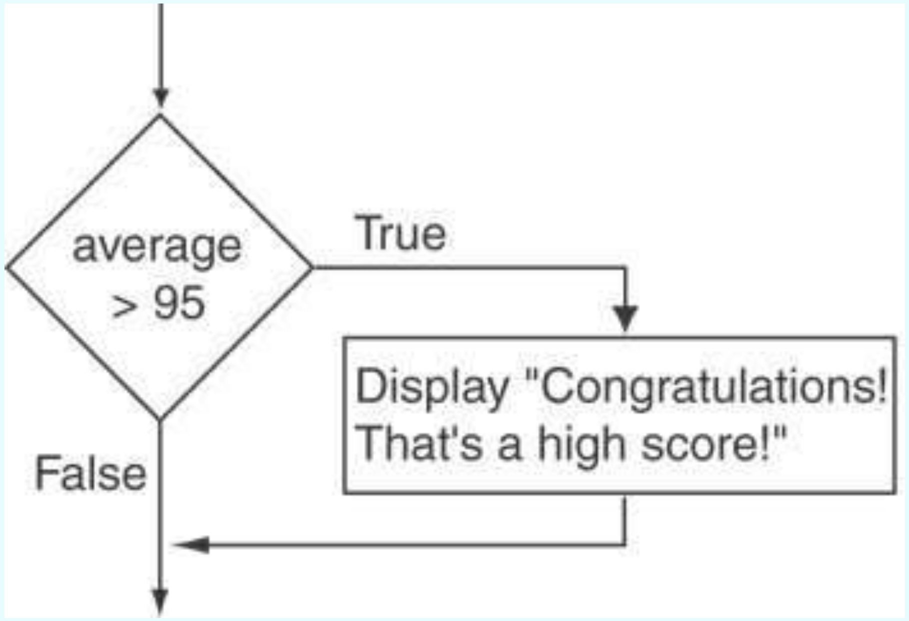
Program Output with Example Input Shown in Bold

```
Enter 3 test scores and I will average them: 80 90 70 [Enter]
Your average is 80.0
```

Program Output with Other Example Input Shown in Bold

```
Enter 3 test scores and I will average them: 100 100 100 [Enter]
Your average is 100.0
Congratulations! That's a high score!
```

Flowchart for Program 4-2 Lines 21 and 22

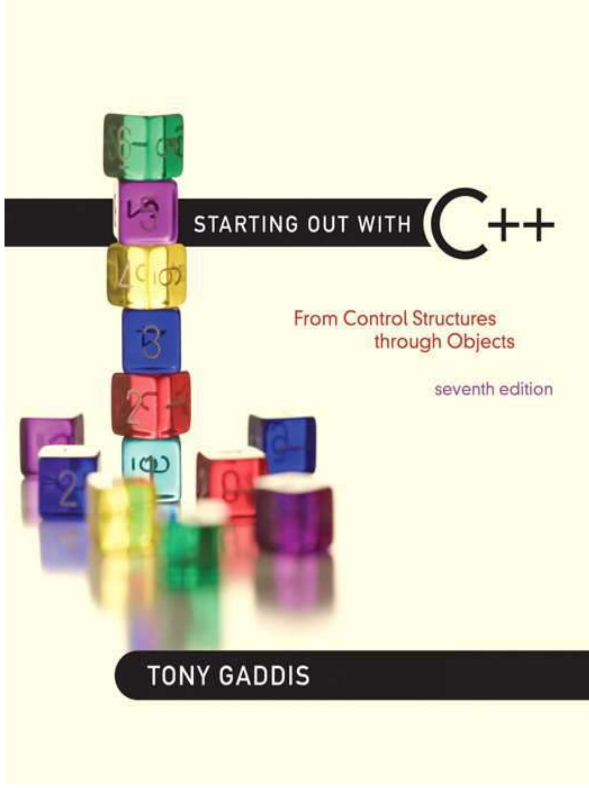


`if` Statement Notes

- Do not place `;` after `(`*`expression`*`)`
- Place *`statement;`* on a separate line after `(`*`expression`*`)`, indented:

```
if (score > 90)
    grade = 'A';
```

- Be careful testing `floats` and `doubles` for equality
- `0` is `false`; any other value is `true`



4.3

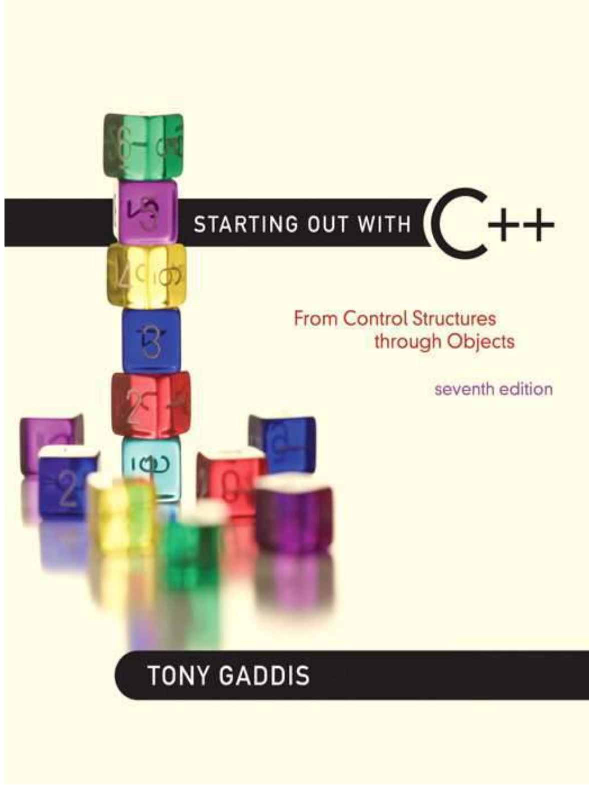
Expanding the `if` Statement

Expanding the `if` Statement

- To execute more than one statement as part of an `if` statement, enclose them in `{ }`:

```
if (score > 90)
{
    grade = 'A';
    cout << "Good Job!\n";
}
```

- `{ }` creates a <u>block</u> of code



4.4

The `if/else` Statement

The `if/else` statement

- Provides two possible paths of execution
- Performs one statement or block if the *`expression`* is true, otherwise performs another statement or block.

The **`if/else`** statement

- General Format:

```
if (expression)
        statement1;  // or block
else
        statement2;  // or block
```

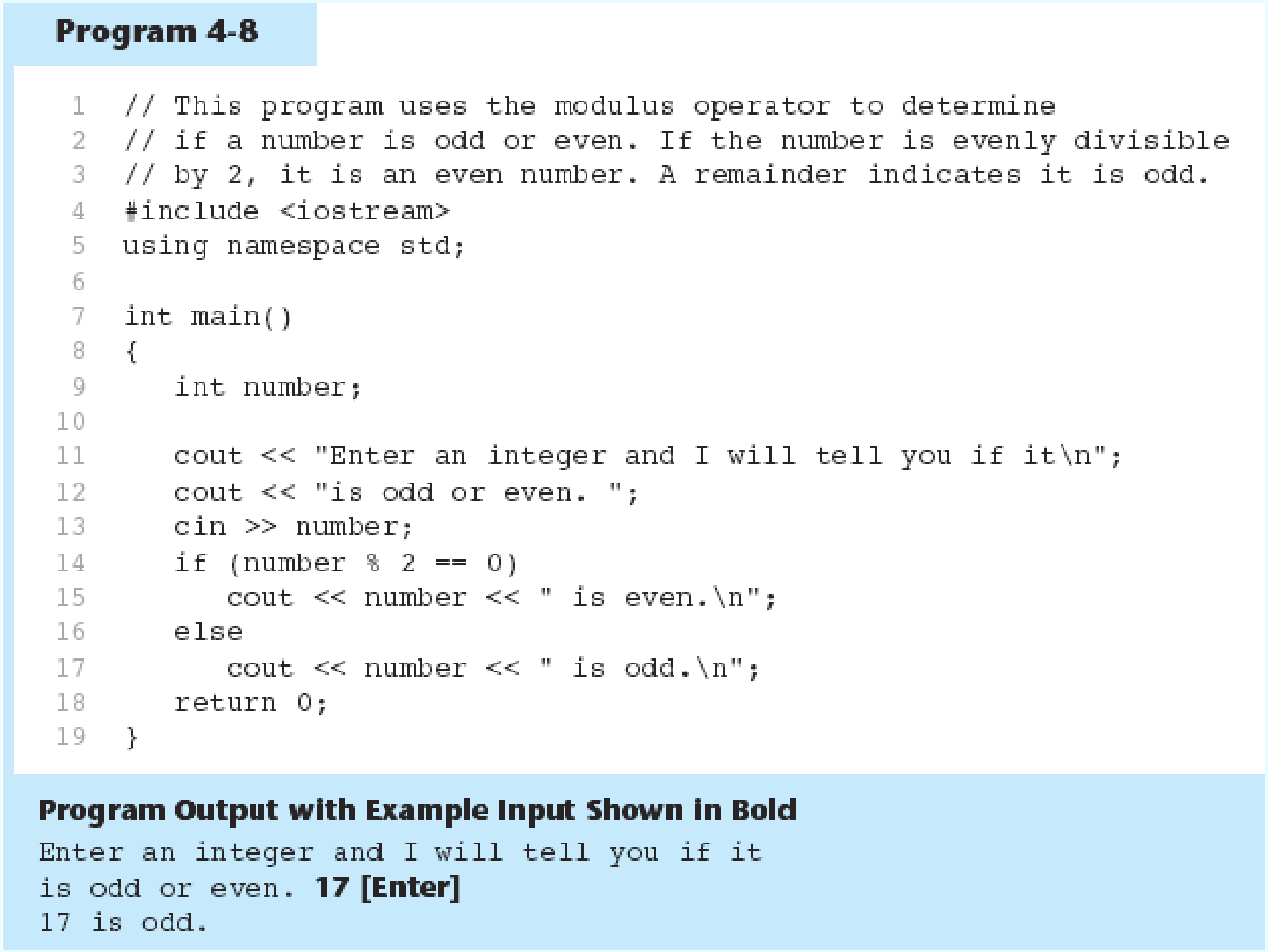
`if/else`-What Happens

To evaluate:

```
    if (expression)
       statement1;
    else
       statement2;
```

- If the *`expression`* is `true`, then *`statement1`* is executed and *`statement2`* is skipped.
- If the *`expression`* is *`false`*, then *`statement1`* is skipped and *`statement2`* is executed.

The **if/else** statement and Modulus Operator in Program 4-8



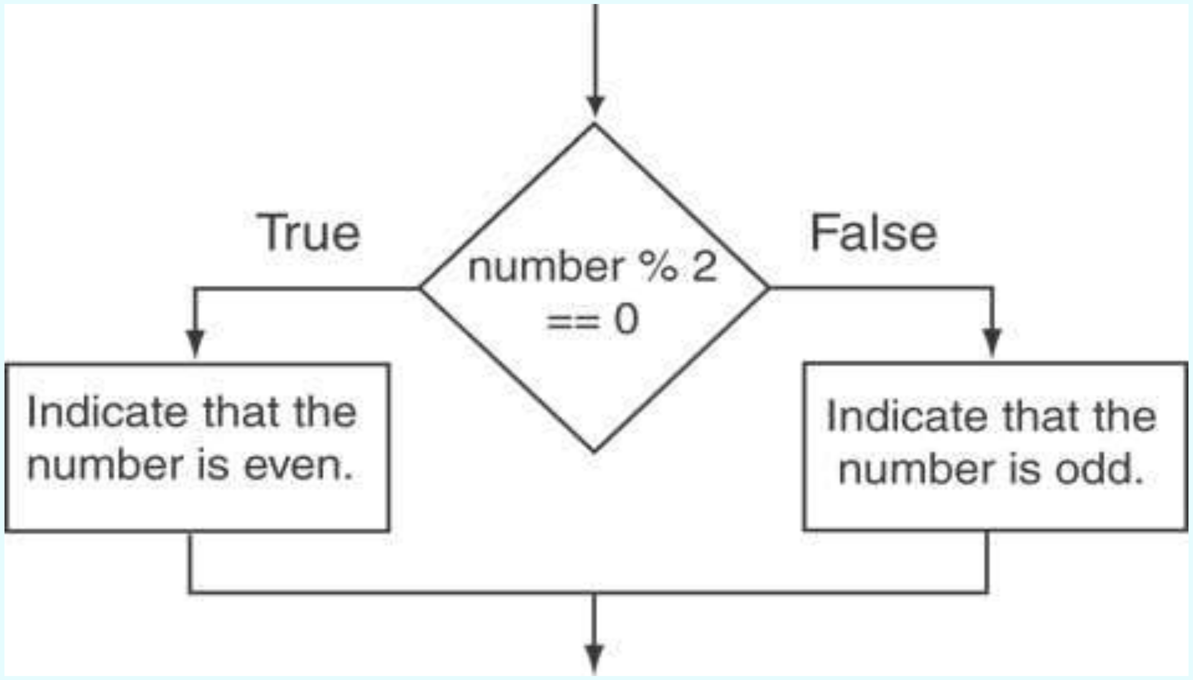
Program 4-8

```
// This program uses the modulus operator to determine
// if a number is odd or even. If the number is evenly divisible
// by 2, it is an even number. A remainder indicates it is odd.
#include <iostream>
using namespace std;

int main()
{
   int number;

   cout << "Enter an integer and I will tell you if it\n";
   cout << "is odd or even. ";
   cin >> number;
   if (number % 2 == 0)
      cout << number << " is even.\n";
   else
      cout << number << " is odd.\n";
   return 0;
}
```

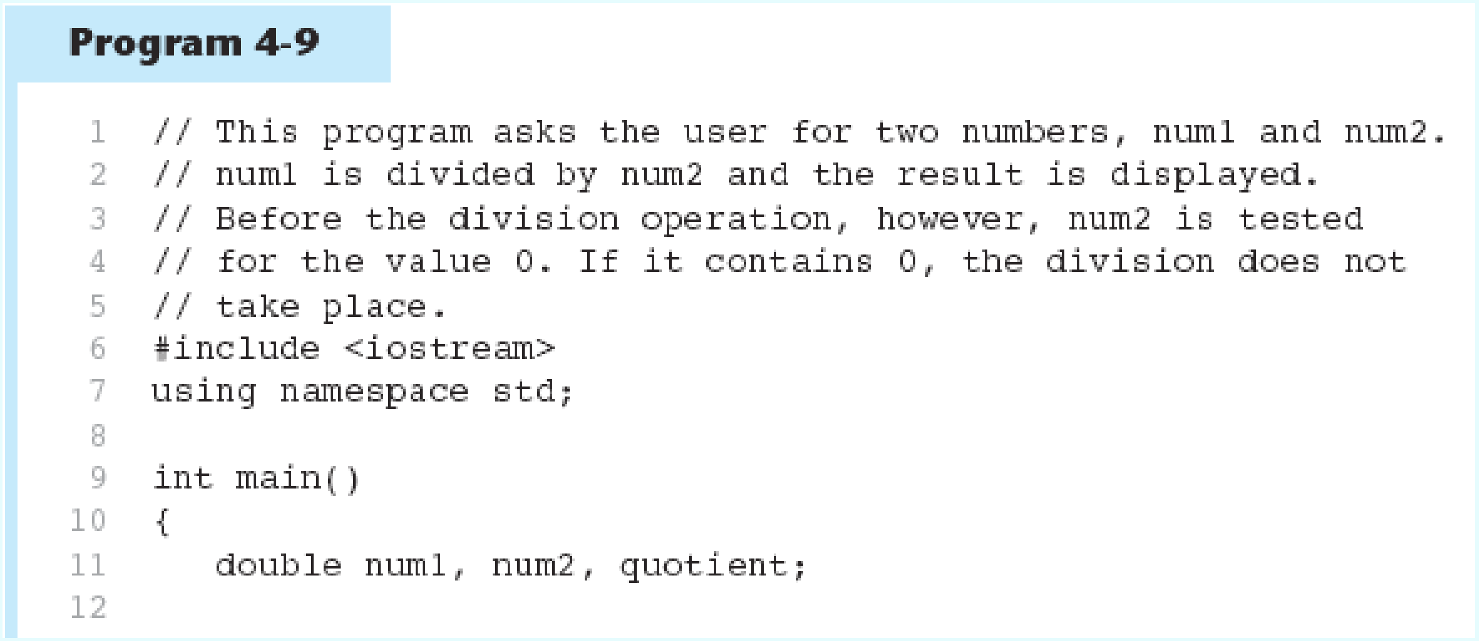
Program Output with Example Input Shown in Bold

```
Enter an integer and I will tell you if it
is odd or even. 17 [Enter]
17 is odd.
```

Flowchart for Program 4-8 Lines 14 through 18


True
number % 2 == 0
False
Indicate that the number is even.
Indicate that the number is odd.

Testing the Divisor in Program 4-9

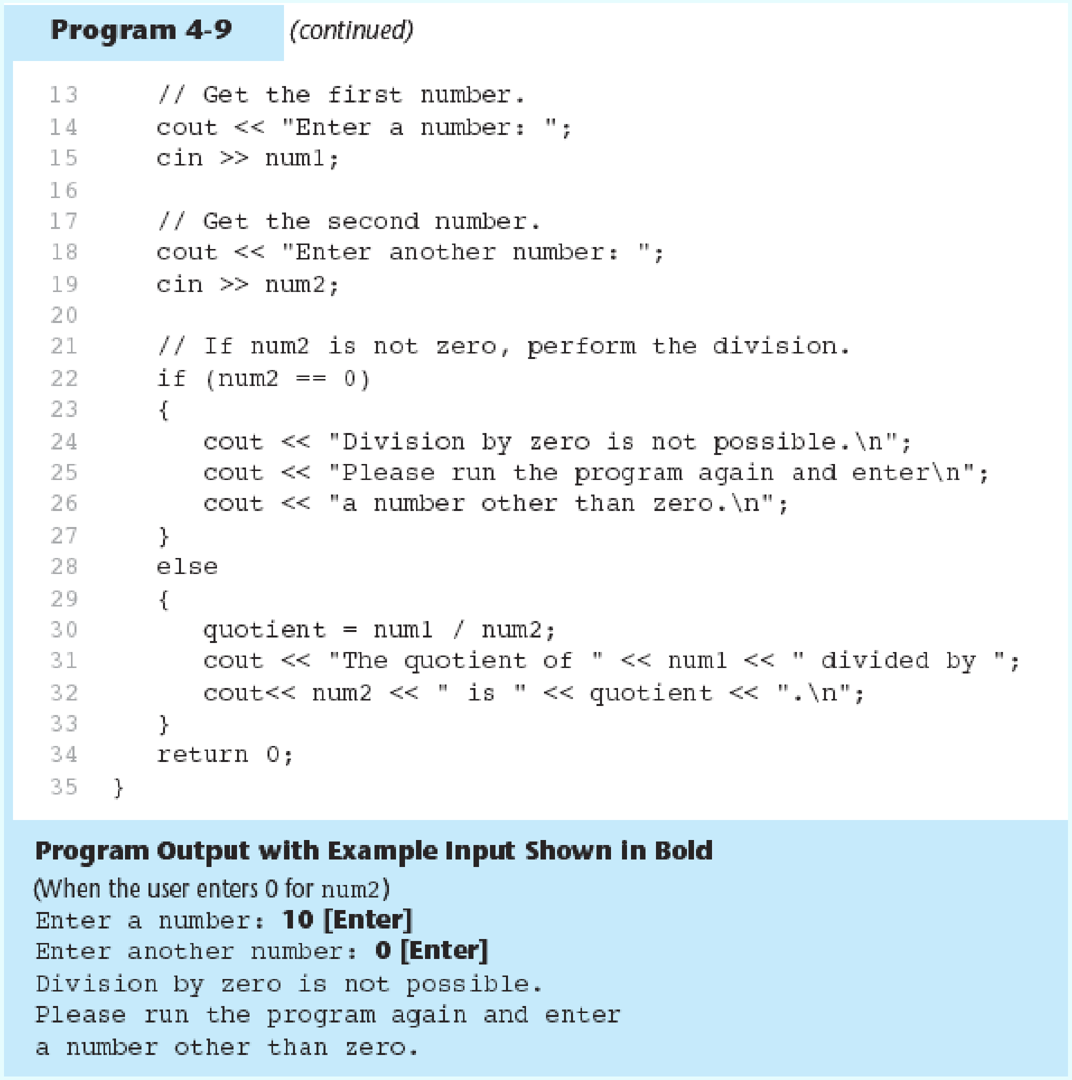


Program 4-9

```
// This program asks the user for two numbers, num1 and num2.
// num1 is divided by num2 and the result is displayed.
// Before the division operation, however, num2 is tested
// for the value 0. If it contains 0, the division does not
// take place.
#include <iostream>
using namespace std;

int main()
{
   double num1, num2, quotient;

```

Continued…

Testing the Divisor in Program 4-9



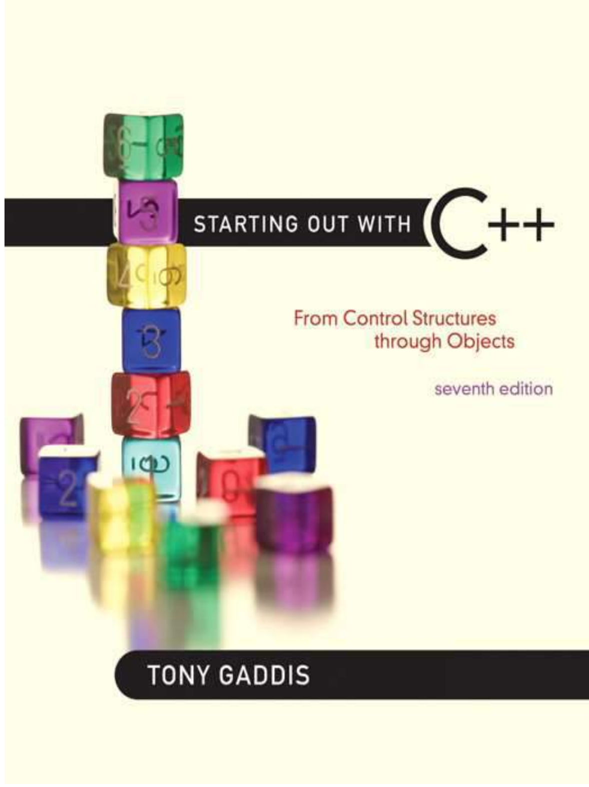
Program 4-9 *(continued)*

```
    // Get the first number.
    cout << "Enter a number: ";
    cin >> num1;

    // Get the second number.
    cout << "Enter another number: ";
    cin >> num2;

    // If num2 is not zero, perform the division.
    if (num2 == 0)
    {
       cout << "Division by zero is not possible.\n";
       cout << "Please run the program again and enter\n";
       cout << "a number other than zero.\n";
    }
    else
    {
       quotient = num1 / num2;
       cout << "The quotient of " << num1 << " divided by ";
       cout<< num2 << " is " << quotient << ".\n";
    }
    return 0;
 }
```

Program Output with Example Input Shown in Bold

(When the user enters 0 for num2)

```
Enter a number: 10 [Enter]
Enter another number: 0 [Enter]
Division by zero is not possible.
Please run the program again and enter
a number other than zero.
```



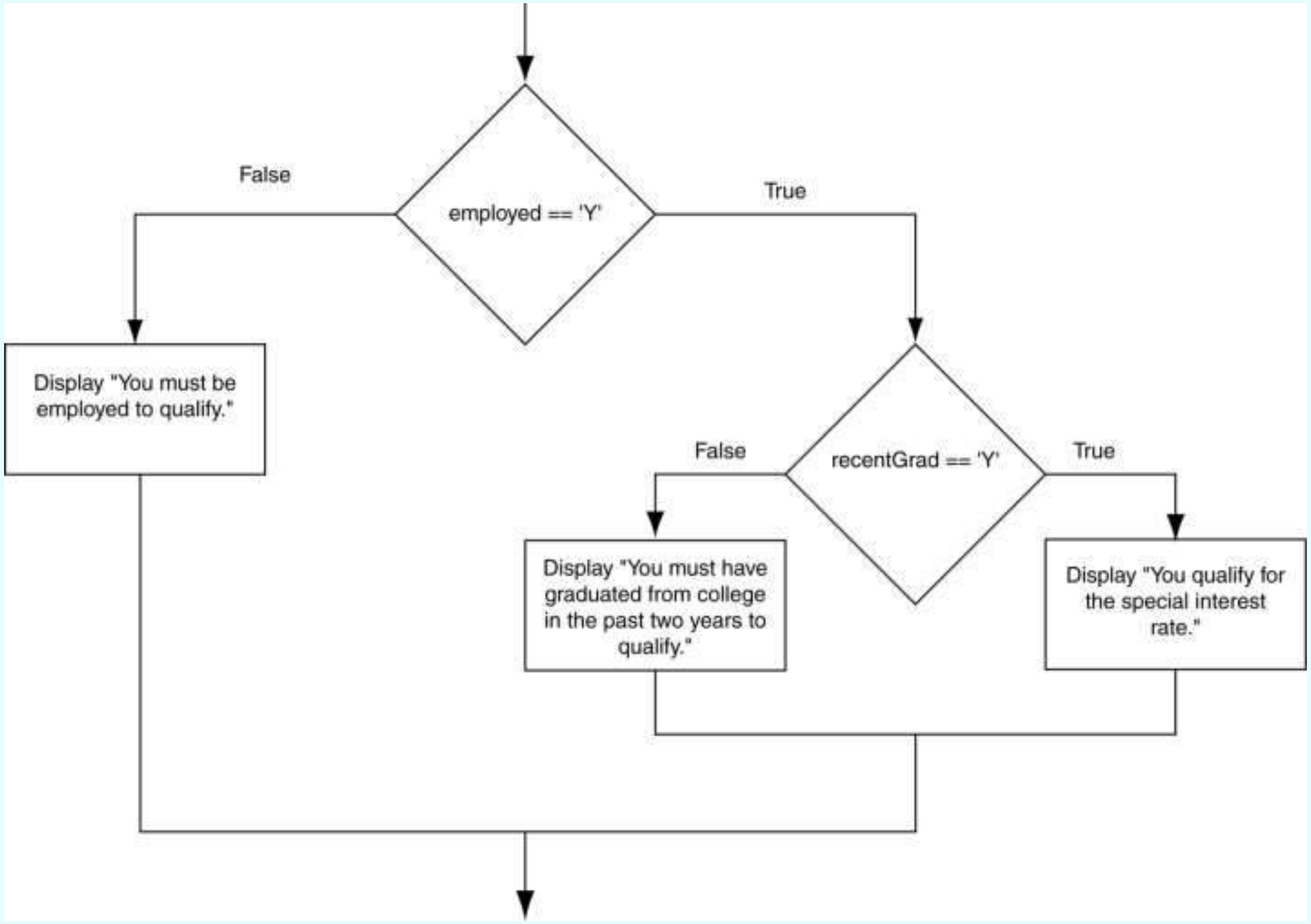
4.5

Nested `if` Statements

Nested **`if`** Statements

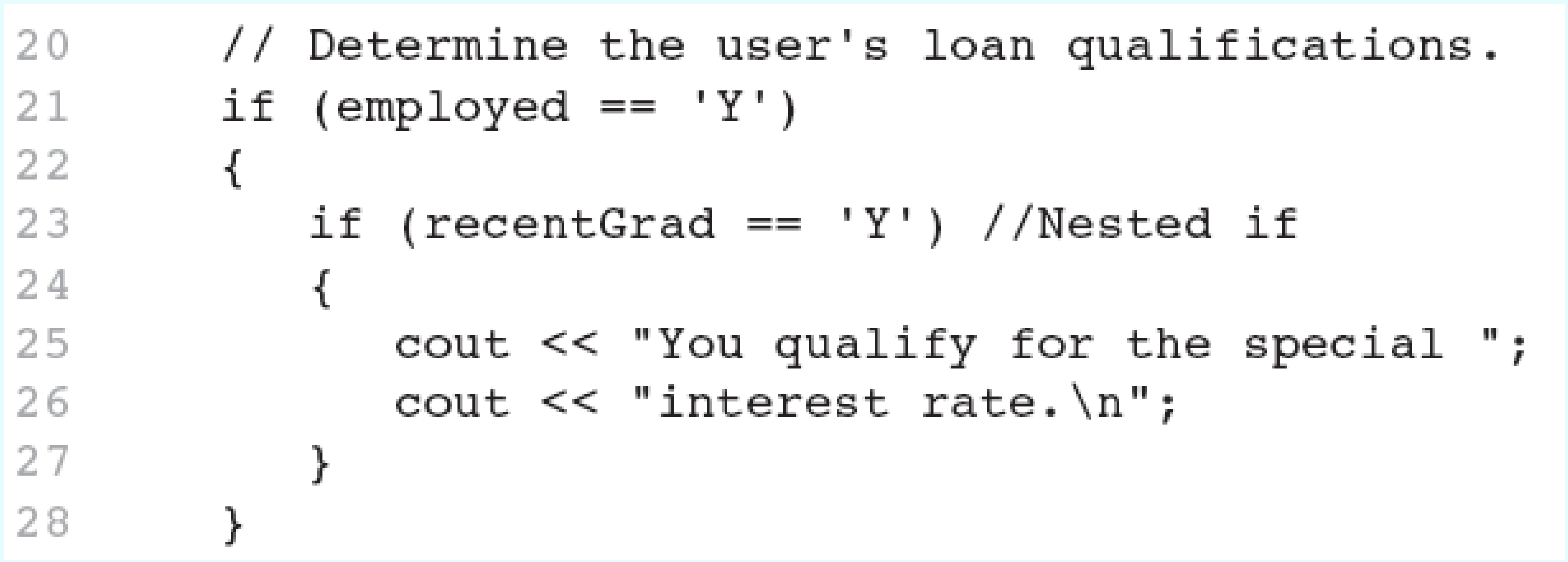
- An `if` statement that is nested inside another `if` statement
- Nested `if` statements can be used to test more than one condition

Flowchart for a Nested **if** Statement


False
employed == 'Y'
True
Display "You must be employed to qualify."
False
recentGrad == 'Y'
True
Display "You must have graduated from college in the past two years to qualify."
Display "You qualify for the special interest rate."

Nested **`if`** Statements

- From Program 4-10



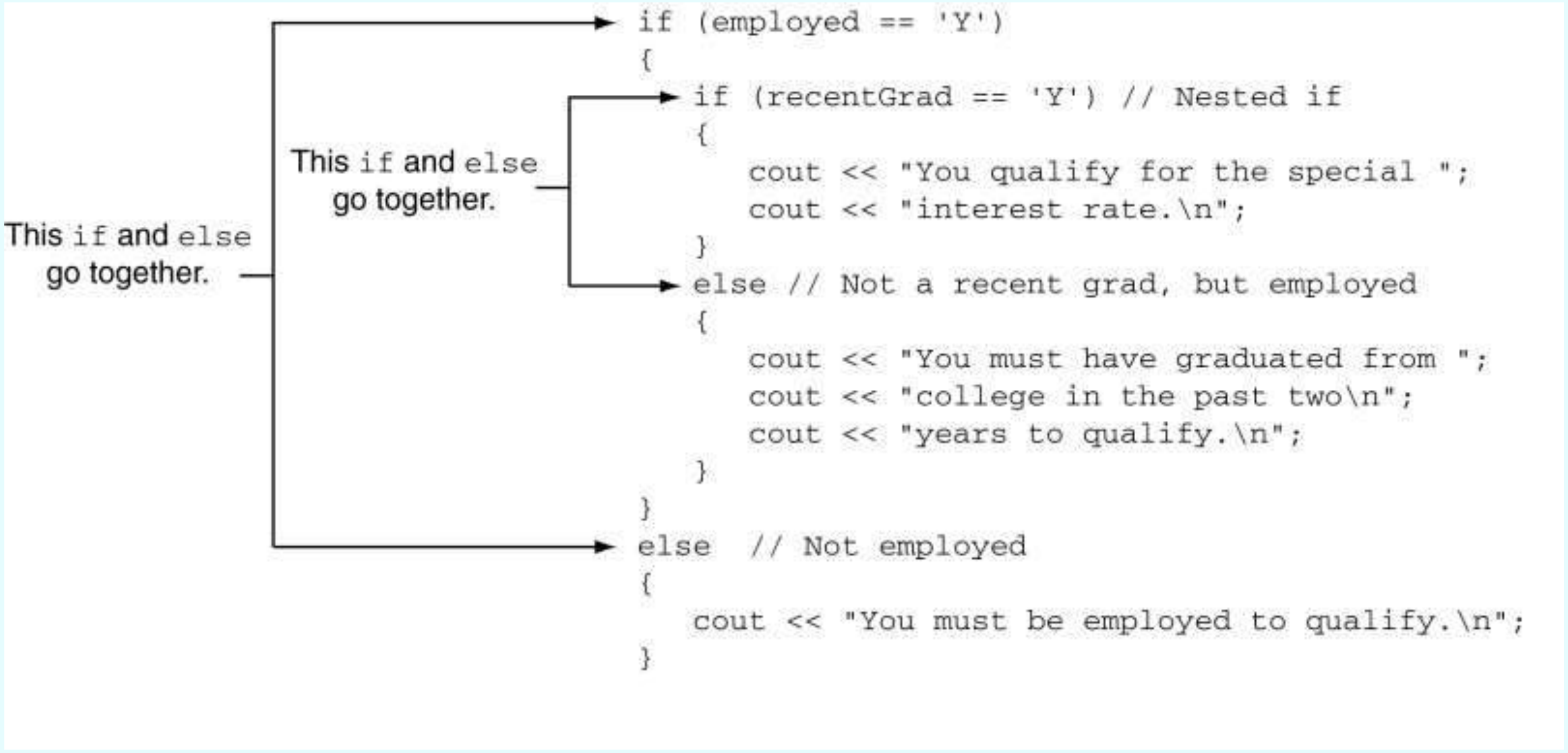
```
    // Determine the user's loan qualifications.
    if (employed == 'Y')
    {
       if (recentGrad == 'Y') //Nested if
       {
          cout << "You qualify for the special ";
          cout << "interest rate.\n";
       }
    }
```

Nested `if` Statements

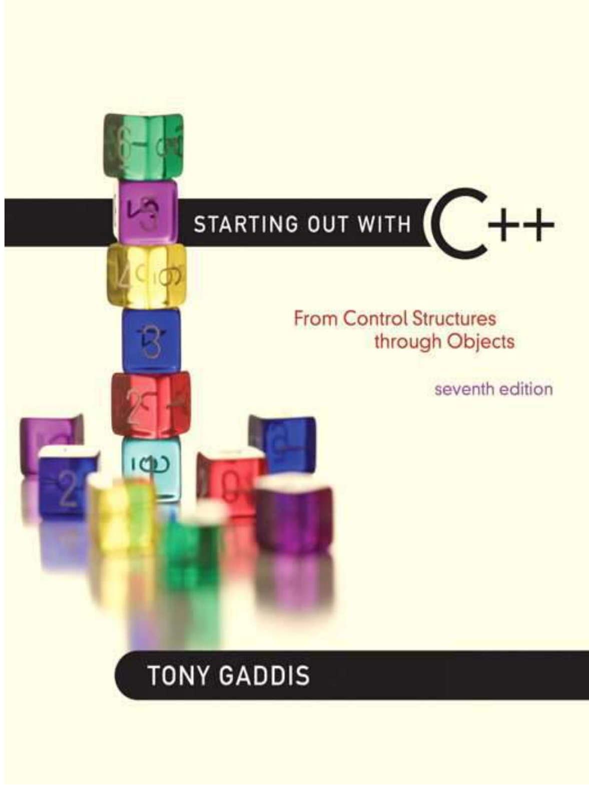
- Another example, from Program 4-1

```
    // Determine the user's loan qualifications.
    if (employed == 'Y')
    {
       if (recentGrad == 'Y') // Nested if
       {
          cout << "You qualify for the special ";
          cout << "interest rate.\n";
       }
       else // Not a recent grad, but employed
       {
          cout << "You must have graduated from ";
          cout << "college in the past two\n";
          cout << "years to qualify.\n";
       }
    }
    else // Not employed
    {
       cout << "You must be employed to qualify.\n";
    }
```

Use Proper Indentation!



```
if (employed == 'Y')
{
   if (recentGrad == 'Y') // Nested if
   {
      cout << "You qualify for the special ";
      cout << "interest rate.\n";
   }
   else // Not a recent grad, but employed
   {
      cout << "You must have graduated from ";
      cout << "college in the past two\n";
      cout << "years to qualify.\n";
   }
}
else  // Not employed
{
   cout << "You must be employed to qualify.\n";
}
```



4.6

The `if/else if` Statement

The `if/else if` Statement

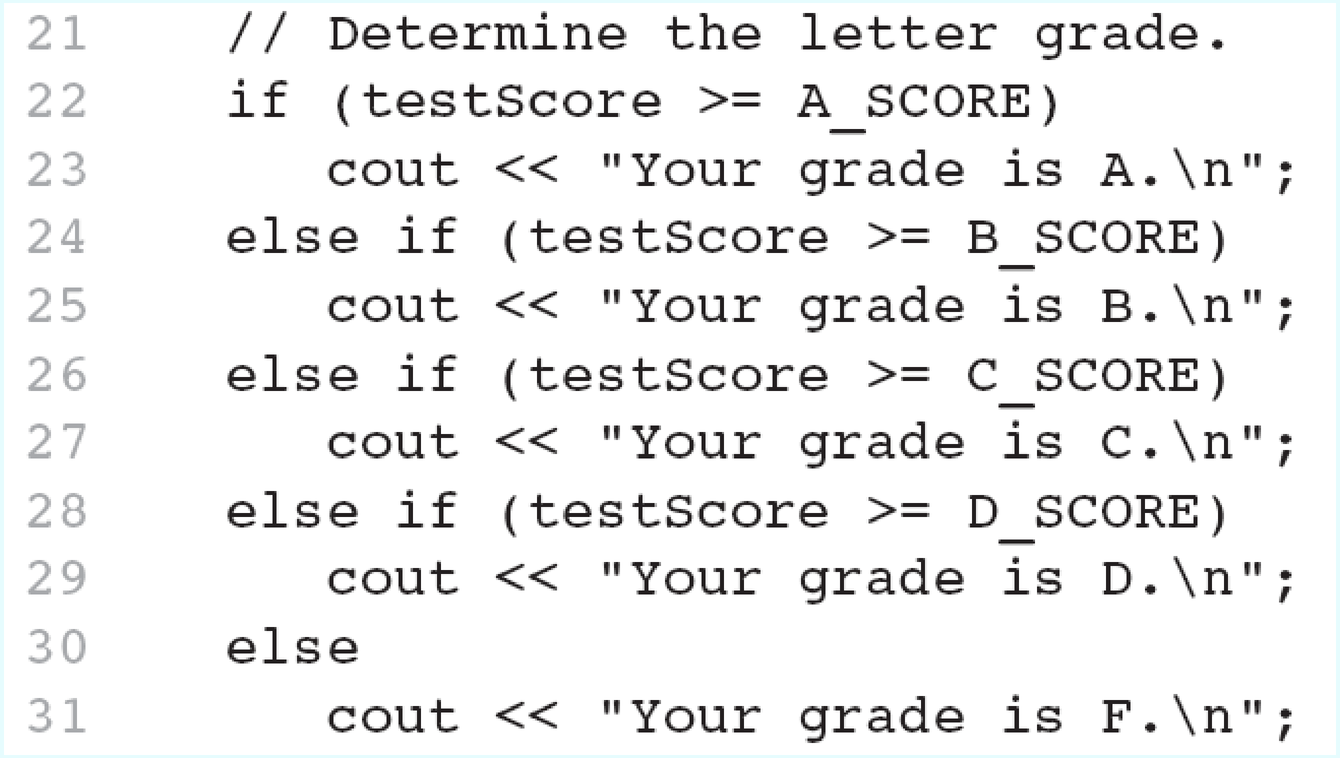
- Tests a series of conditions until one is found to be true
- Often simpler than using nested `if/else` statements
- Can be used to model thought processes such as:

  "If it is raining, take an umbrella,
  else, if it is windy, take a hat,
  else, take sunglasses”

if/else if Format

```
if (expression)
        statement1;  // or block
else if (expression)
        statement2;  // or block
       .
       . // other else ifs
       .
else if (expression)
        statementn;  // or block
```

The `if/else if` Statement in Program 4-13



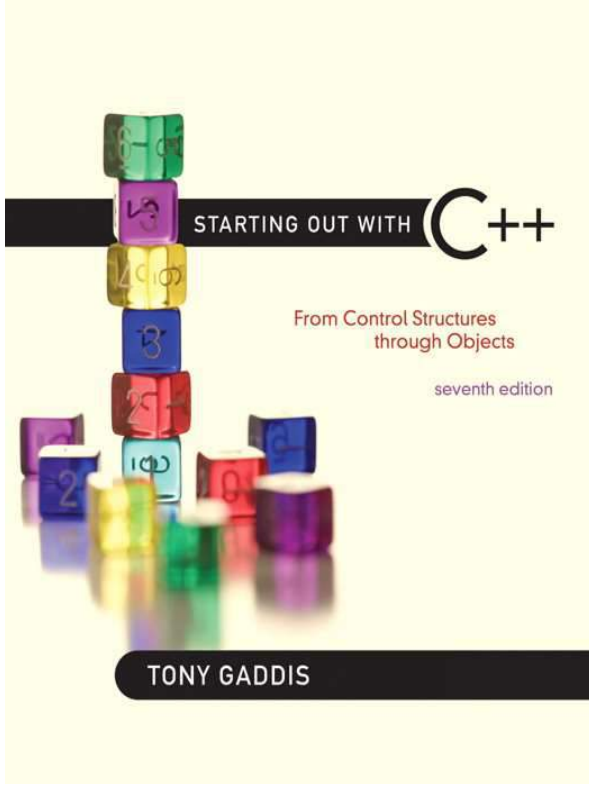
```
    // Determine the letter grade.
    if (testScore >= A_SCORE)
        cout << "Your grade is A.\n";
    else if (testScore >= B_SCORE)
        cout << "Your grade is B.\n";
    else if (testScore >= C_SCORE)
        cout << "Your grade is C.\n";
    else if (testScore >= D_SCORE)
        cout << "Your grade is D.\n";
    else
        cout << "Your grade is F.\n";
```

Using a Trailing **`else`** to Catch Errors in Program 4-14

- The trailing `else` clause is optional, but it is best used to catch errors.

```
    // Determine the letter grade.
    if (testScore >= A_SCORE)
       cout << "Your grade is A.\n";
    else if (testScore >= B_SCORE)
       cout << "Your grade is B.\n";
    else if (testScore >= C_SCORE)
       cout << "Your grade is C.\n";
    else if (testScore >= D_SCORE)
       cout << "Your grade is D.\n";
    else if (testScore >= 0)
       cout << "Your grade is F.\n";
    else
       cout << "Invalid test score.\n";
```

This trailing `else` catches invalid test scores

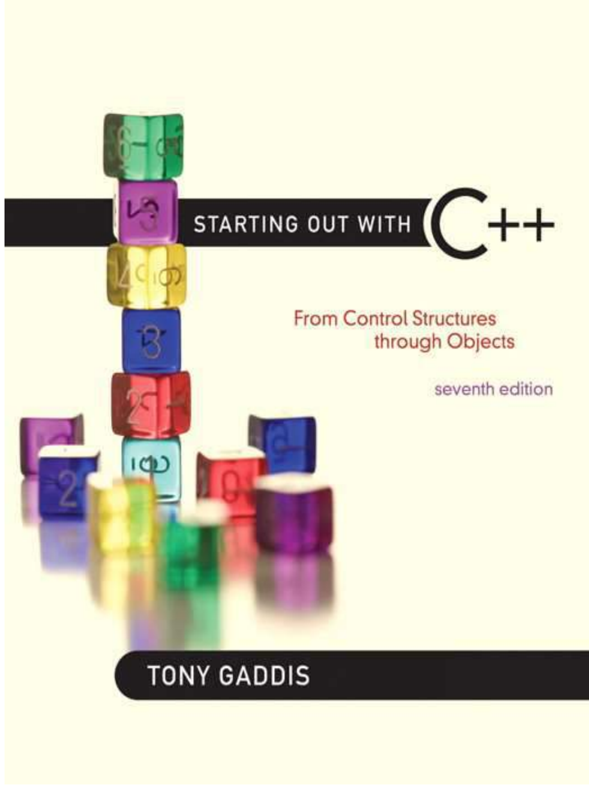

STARTING OUT WITH C++
From Control Structures through Objects
seventh edition
TONY GADDIS

4.7

Flags

Flags

- Variable that signals a condition
- Usually implemented as a `bool` variable
- Can also be an integer
  - The value `0` is considered `false`
  - Any nonzero value is considered `true`
- As with other variables in functions, must be assigned an initial value before it is used



4.8

Logical Operators

Logical Operators

- Used to create relational expressions from other relational expressions
- Operators, meaning, and explanation:

&&	AND	New relational expression is true if both expressions are true
||	OR	New relational expression is true if either expression is true
!	NOT	Reverses the value of an expression – true expression becomes false, and false becomes true

Logical Operators-Examples

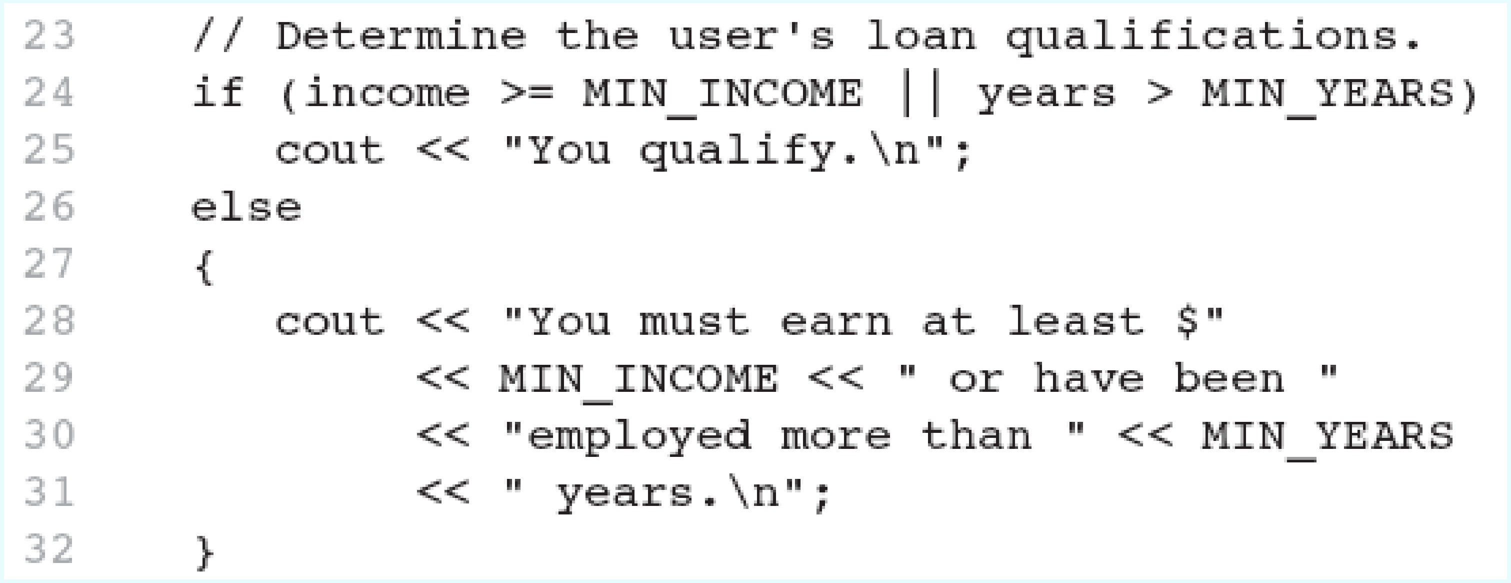
```
int x = 12, y = 5, z = -4;
```

`(x > y) && (y > z)`	`true`
`(x > y) && (z > y)`	`false`
`(x <= z) || (y == z)`	`false`
`(x <= z) || (y != z)`	`true`
`!(x >= z)`	`false`

The logical **&&** operator in Program 4-15

```
// Determine the user's loan qualifications.
if (employed == 'Y' && recentGrad == 'Y')
{
   cout << "You qualify for the special "
        << "interest rate.\n";
}
else
{
   cout << "You must be employed and have\n"
        << "graduated from college in the\n"
        << "past two years to qualify.\n";
}
```

The logical || Operator in Program 4-16



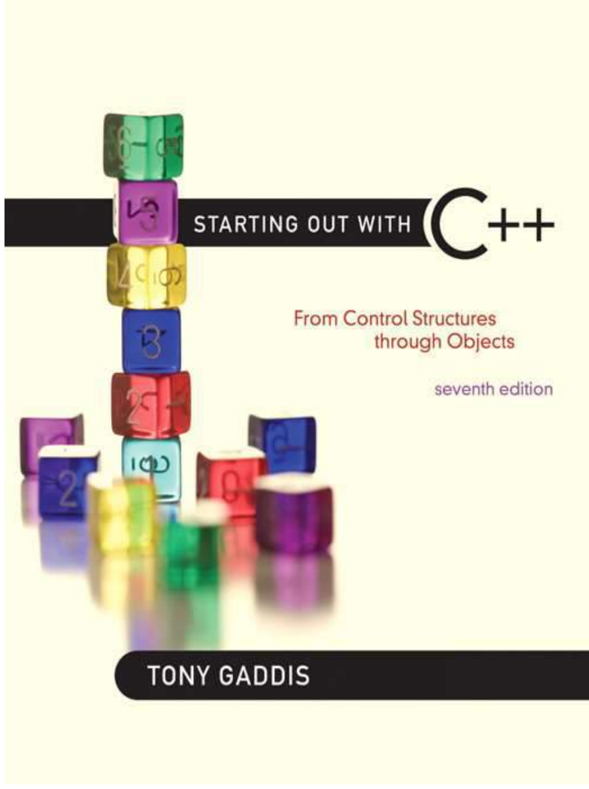
```
// Determine the user's loan qualifications.
if (income >= MIN_INCOME || years > MIN_YEARS)
   cout << "You qualify.\n";
else
{
   cout << "You must earn at least $"
        << MIN_INCOME << " or have been "
        << "employed more than " << MIN_YEARS
        << " years.\n";
}
```

The logical ! Operator in Program 4-17

```
    // Determine the user's loan qualifications.
    if (!(income >= MIN_INCOME || years > MIN_YEARS))
    {
       cout << "You must earn at least $"
            << MIN_INCOME << " or have been "
            << "employed more than " << MIN_YEARS
            << " years.\n";
    }
    else
       cout << "You qualify.\n";
```

Logical Operator-Notes

- `!` has highest precedence, followed by `&&`, then `||`
- If the value of an expression can be determined by evaluating just the sub-expression on left side of a logical operator, then the sub-expression on the right side will not be evaluated (*short circuit evaluation*)



4.9

Checking Numeric Ranges with Logical Operators

Checking Numeric Ranges with Logical Operators

- Used to test to see if a value falls **inside** a range:

```
if (grade >= 0 && grade <= 100)
   cout << "Valid grade";
```

- Can also test to see if value falls **outside** of range:

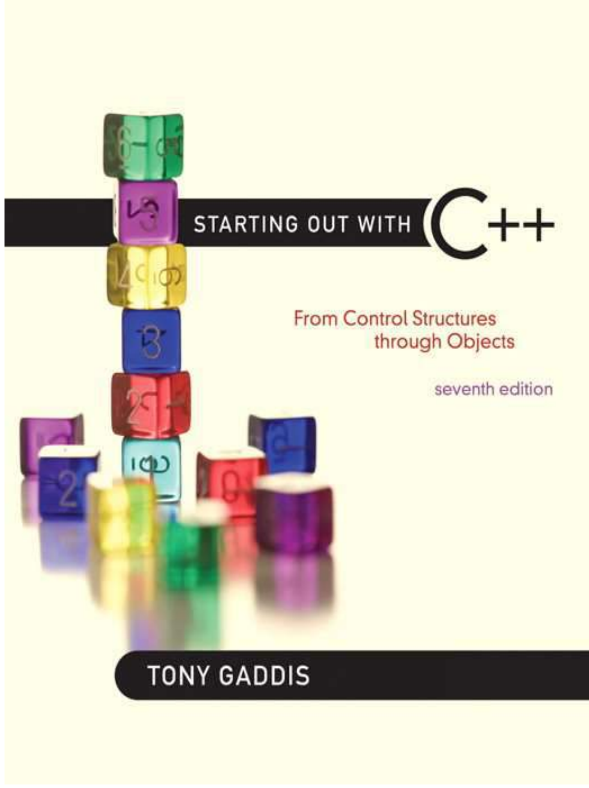
```
if (grade <= 0 || grade >= 100)
  cout << "Invalid grade";
```

- Cannot use mathematical notation:

```
if (0 <= grade <= 100) //doesn't work!
```

4.10

Menus

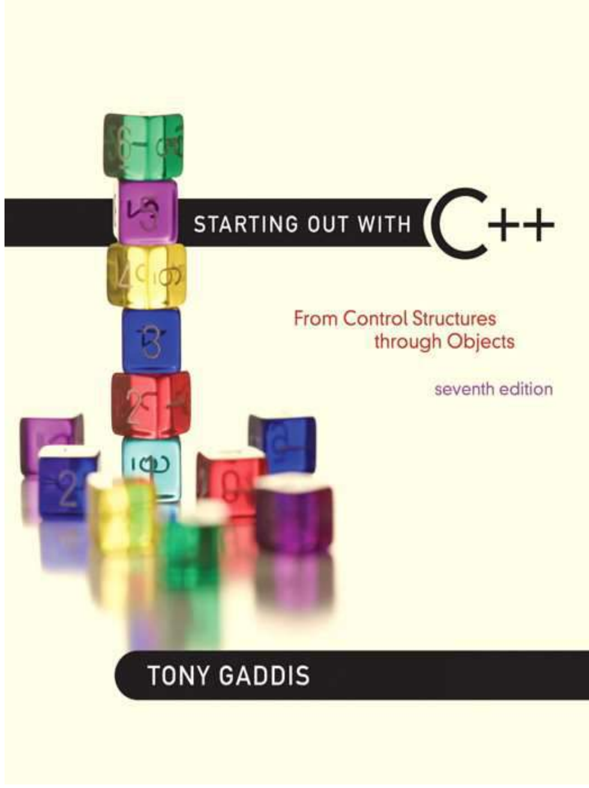

STARTING OUT WITH C++
From Control Structures through Objects
seventh edition
TONY GADDIS

Menus

- <u>Menu-driven program</u>: program execution controlled by user selecting from a list of actions
- <u>Menu</u>: list of choices on the screen
- Menus can be implemented using `if/else if` statements

Menu-Driven Program Organization

- Display list of numbered or lettered choices for actions
- Prompt user to make selection
- Test user selection in *`expression`*
  - if a match, then execute code for action
  - if not, then go on to next *`expression`*



4.11

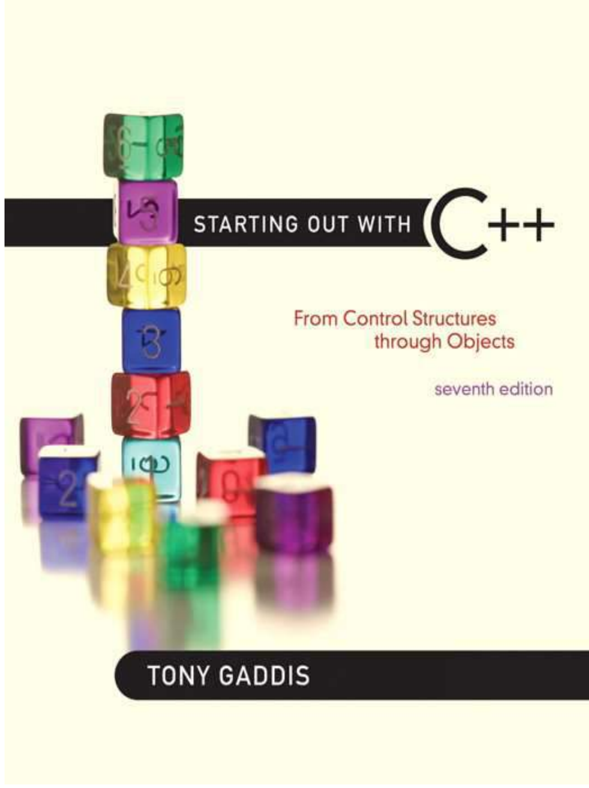
Validating User Input

Validating User Input

- Input validation: inspecting input data to determine whether it is acceptable
- Bad output will be produced from bad input
- Can perform various tests:
  - Range
  - Reasonableness
  - Valid menu choice
  - Divide by zero

Input Validation in Program 4-19

```
   int testScore;  // To hold a numeric test score

   // Get the numeric test score.
   cout << "Enter your numeric test score and I will\n"
        << "tell you the letter grade you earned: ";
   cin >> testScore;

   // Validate the input and determine the grade.
   if (testScore >= MIN_SCORE && testScore <= MAX_SCORE)
   {
      // Determine the letter grade.
      if (testScore >= A_SCORE)
        cout << "Your grade is A.\n";
      else if (testScore >= B_SCORE)
        cout << "Your grade is B.\n";
      else if (testScore >= C_SCORE)
        cout << "Your grade is C.\n";
      else if (testScore >= D_SCORE)
        cout << "Your grade is D.\n";
      else
        cout << "Your grade is F.\n";
   }
   else
   {
      // An invalid score was entered.
      cout << "That is an invalid score. Run the program\n"
           << "again and enter a value in the range of\n"
           << MIN_SCORE << " through " << MAX_SCORE << ".\n";
   }
```



4.12

Comparing Characters and Strings

Comparing Characters

- Characters are compared using their ASCII values
- 'A' < 'B'
  - The ASCII value of 'A' (65) is less than the ASCII value of 'B'(66)
- '1' < '2'
  - The ASCII value of '1' (49) is less than the ASCI value of '2' (50)
- Lowercase letters have higher ASCII codes than uppercase letters, so 'a' > 'Z'

Relational Operators Compare Characters in Program 4-20

```cpp
// Get a character from the user.
cout << "Enter a digit or a letter: ";
ch = cin.get();

// Determine what the user entered.
if (ch >= '0' && ch <= '9')
   cout << "You entered a digit.\n";
else if (ch >= 'A' && ch <= 'Z')
   cout << "You entered an uppercase letter.\n";
else if (ch >= 'a' && ch <= 'z')
   cout << "You entered a lowercase letter.\n";
else
   cout << "That is not a digit or a letter.\n";
```

Comparing **`string`** Objects

- Like characters, strings are compared using their ASCII values

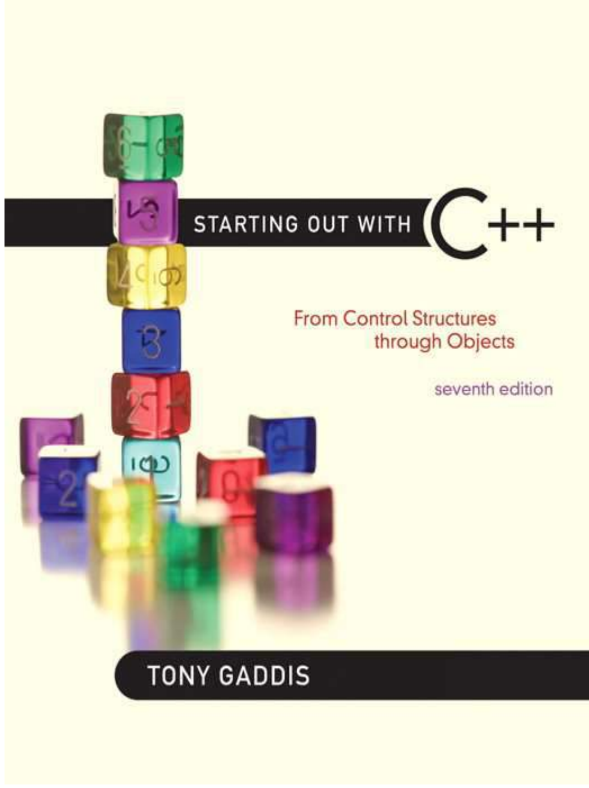
```
string name1 = "Mary";
string name2 = "Mark";

name1 > name2   // true
name1 <= name2 // false
name1 != name2  // true

name1 < "Mary Jane" // true
```

The characters in each string must match before they are equal

Relational Operators Compare Strings in Program 4-21

```
    // Determine and display the correct price
    if (partNum == "S-29A")
       cout << "The price is $" << PRICE_A << endl;
    else if (partNum == "S-29B")
       cout << "The price is $" << PRICE_B << endl;
    else
       cout << partNum << " is not a valid part number.\n";
```

4.13



The Conditional Operator

The Conditional Operator

- Can use to create short `if/else` statements
- Format: `expr ? expr : expr;`

$$\underbrace{\texttt{x<0}} \;\texttt{?}\; \underbrace{\texttt{y=10}} \;\texttt{:}\; \underbrace{\texttt{z=20}}\texttt{;}$$

First Expression:
Expression to be
tested

2nd Expression:
Executes if first
expression is true

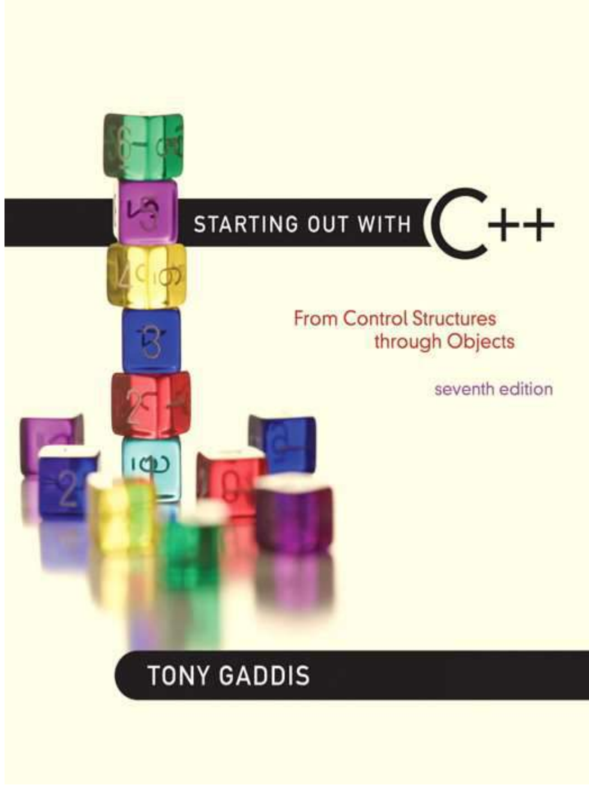
3rd Expression:
Executes if the first
expression is false

The Conditional Operator

- The value of a conditional expression is
  - The value of the second expression if the first expression is true
  - The value of the third expression if the first expression is false
- Parentheses ( ) may be needed in an expression due to precedence of conditional operator

The Conditional Operator in Program 4-22

```
// This program calculates a consultant's charges at $50
// per hour, for a minimum of 5 hours. The ?: operator
// adjusts hours to 5 if less than 5 hours were worked.
#include <iostream>
#include <iomanip>
using namespace std;

int main()
{
   const double PAY_RATE = 50.0;  // Hourly pay rate
   const int MIN_HOURS = 5;       // Minimum billable hours
   double hours,                  // Hours worked
          charges;                // Total charges

   // Get the hours worked.
   cout << "How many hours were worked? ";
   cin >> hours;

   // Determine the hours to charge for.
   hours = hours < MIN_HOURS ? MIN_HOURS : hours;

   // Calculate and display the charges.
   charges = PAY_RATE * hours;
   cout << fixed << showpoint << setprecision(2)
        << "The charges are $" << charges << endl;
   return 0;
}
```



4.14

The `switch` Statement

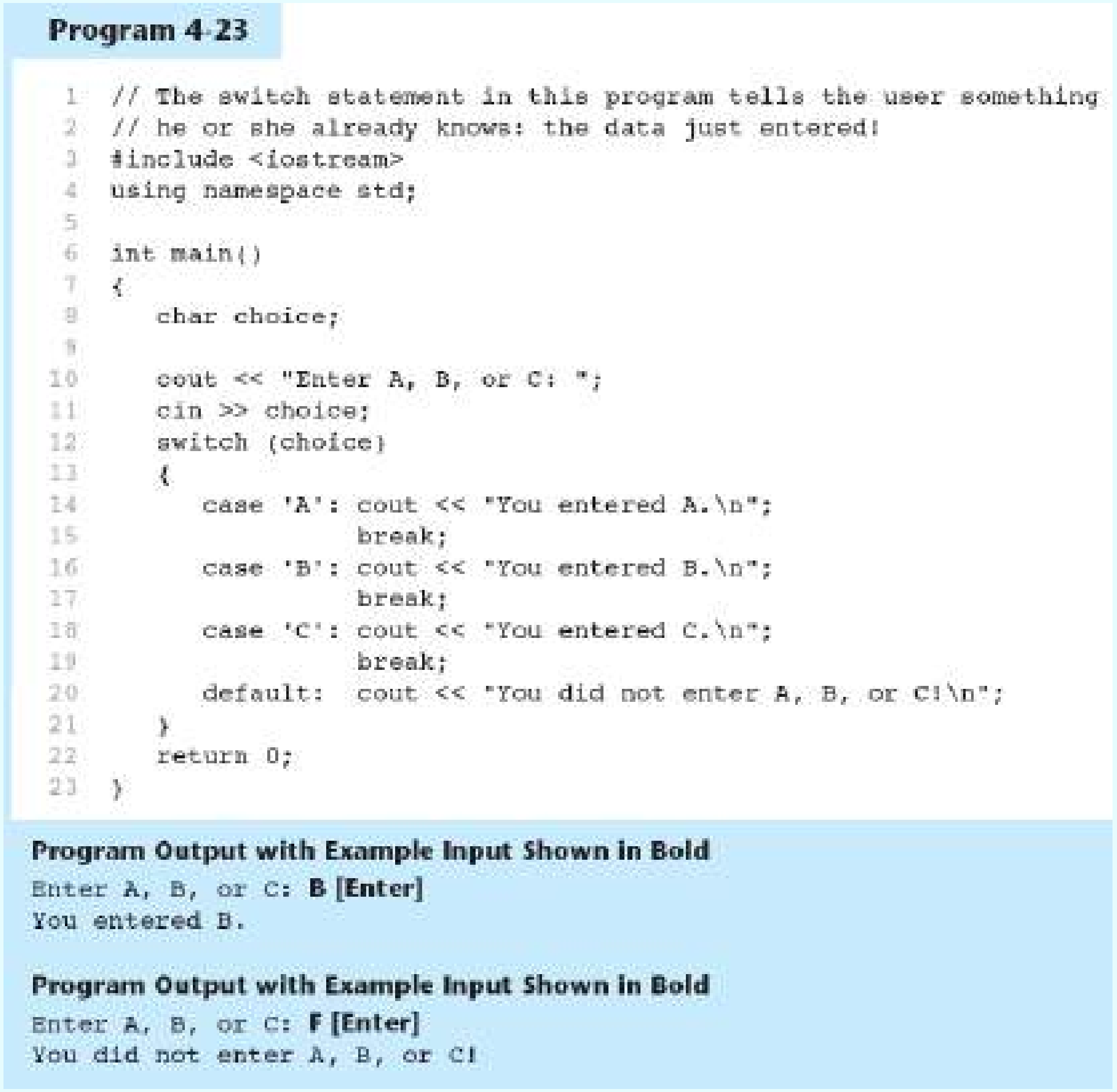
The `switch` Statement

- Used to select among statements from several alternatives
- In some cases, can be used instead of `if/else if` statements

switch Statement Format

```
switch (expression) //integer
{
  case exp1: statement1;
  case exp2: statement2;
  ...
  case expn: statementn;
  default:   statementn+1;
}
```

The **switch** Statement in Program 4-23



Program 4-23

```
// The switch statement in this program tells the user something
// he or she already knows: the data just entered!
#include <iostream>
using namespace std;

int main()
{
   char choice;

   cout << "Enter A, B, or C: ";
   cin >> choice;
   switch (choice)
   {
      case 'A': cout << "You entered A.\n";
                break;
      case 'B': cout << "You entered B.\n";
                break;
      case 'C': cout << "You entered C.\n";
                break;
      default:  cout << "You did not enter A, B, or C!\n";
   }
   return 0;
}
```

Program Output with Example Input Shown in Bold

```
Enter A, B, or C: B [Enter]
You entered B.
```

Program Output with Example Input Shown in Bold

```
Enter A, B, or C: F [Enter]
You did not enter A, B, or C!
```

`switch` Statement Requirements

1) *`expression`* must be an integer variable or an expression that evaluates to an integer value
2) *`exp1`* through *`expn`* must be constant integer expressions or literals, and must be unique in the `switch` statement
3) `default` is optional but recommended

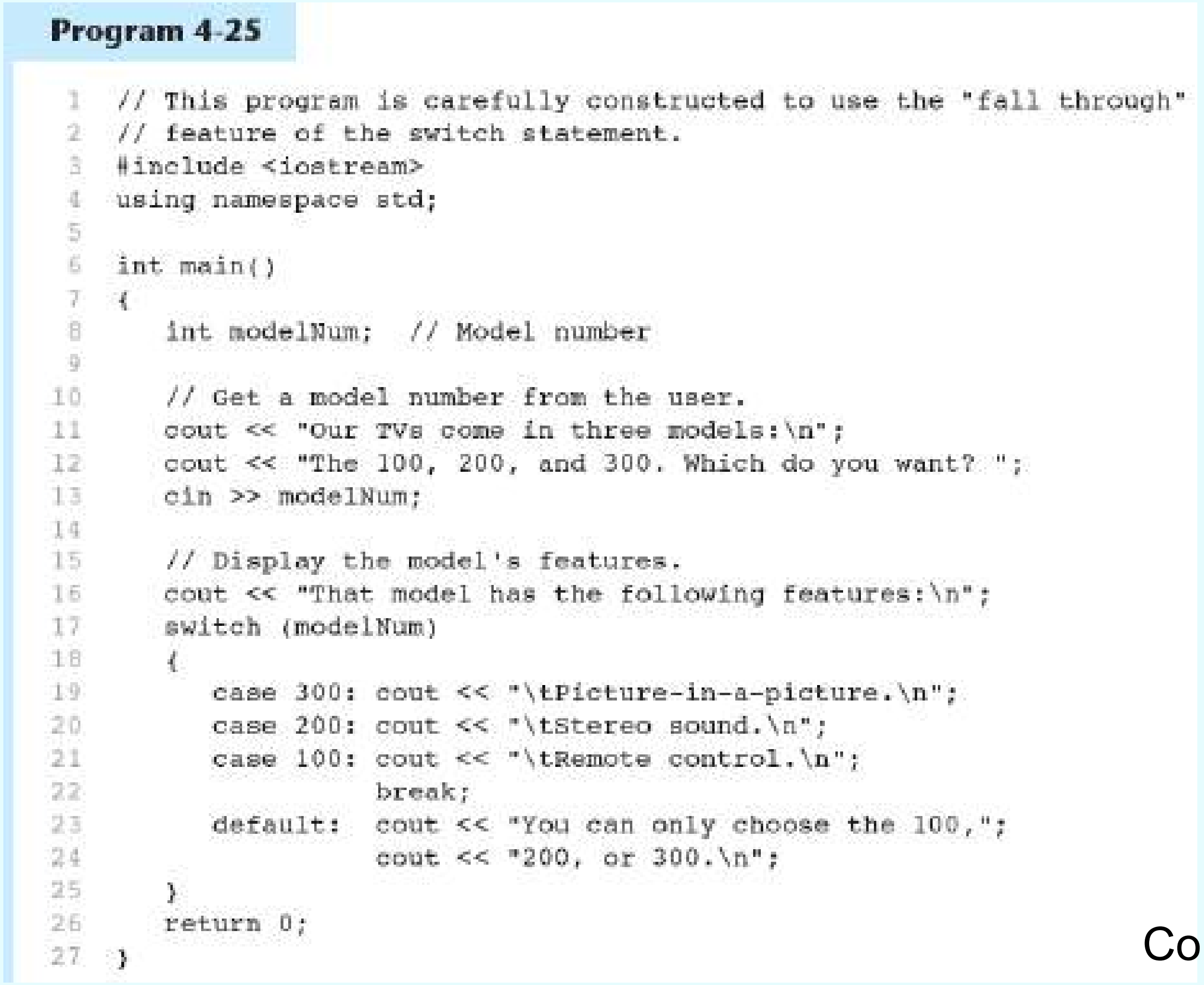
`switch` Statement-How it Works

1) *`expression`* is evaluated
2) The value of *`expression`* is compared against *`exp`***`1`** through *`exp`***`n`**.
3) If *`expression`* matches value *`exp`***`i`**, the program branches to the statement following *`exp`***`i`** and continues to the end of the `switch`
4) If no matching value is found, the program branches to the statement after `default:`

`break` Statement

- Used to exit a `switch` statement
- If it is left out, the program "falls through" the remaining statements in the `switch` statement

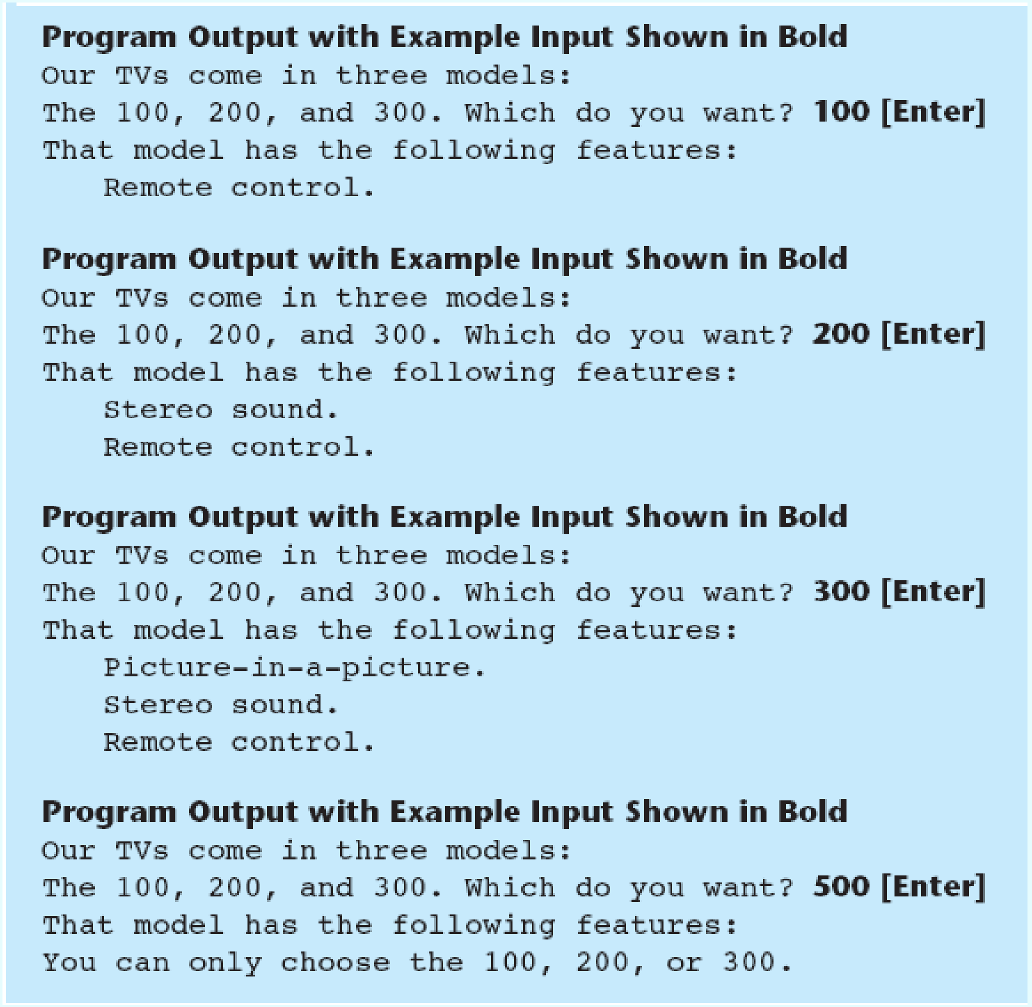
break and **default** statements in Program 4-25


Program 4-25

```
// This program is carefully constructed to use the "fall through"
// feature of the switch statement.
#include <iostream>
using namespace std;

int main()
{
   int modelNum;   // Model number

   // Get a model number from the user.
   cout << "Our TVs come in three models:\n";
   cout << "The 100, 200, and 300. Which do you want? ";
   cin >> modelNum;

   // Display the model's features.
   cout << "That model has the following features:\n";
   switch (modelNum)
   {
      case 300: cout << "\tPicture-in-a-picture.\n";
      case 200: cout << "\tStereo sound.\n";
      case 100: cout << "\tRemote control.\n";
                break;
      default:  cout << "You can only choose the 100,";
                cout << "200, or 300.\n";
   }
   return 0;
}
```

Continued…

`break` and `default` statements in Program 4-25



Program Output with Example Input Shown in Bold

```
Our TVs come in three models:
The 100, 200, and 300. Which do you want? 100 [Enter]
That model has the following features:
    Remote control.
```

Program Output with Example Input Shown in Bold

```
Our TVs come in three models:
The 100, 200, and 300. Which do you want? 200 [Enter]
That model has the following features:
    Stereo sound.
    Remote control.
```

Program Output with Example Input Shown in Bold

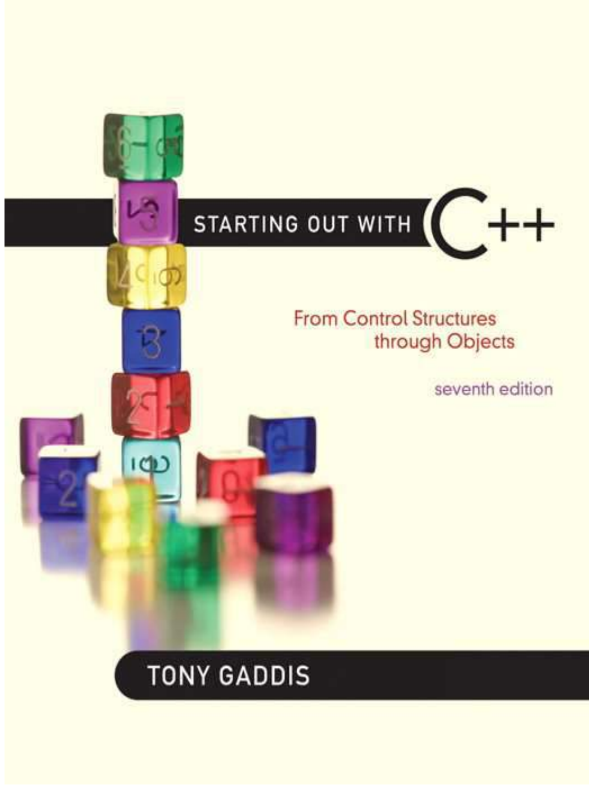
```
Our TVs come in three models:
The 100, 200, and 300. Which do you want? 300 [Enter]
That model has the following features:
    Picture-in-a-picture.
    Stereo sound.
    Remote control.
```

Program Output with Example Input Shown in Bold

```
Our TVs come in three models:
The 100, 200, and 300. Which do you want? 500 [Enter]
That model has the following features:
You can only choose the 100, 200, or 300.
```

Using `switch` in Menu Systems

- `switch` statement is a natural choice for menu-driven program:
  - display the menu
  - then, get the user's menu selection
  - use user input as `expression` in `switch` statement
  - use menu choices as *`expr`* in `case` statements



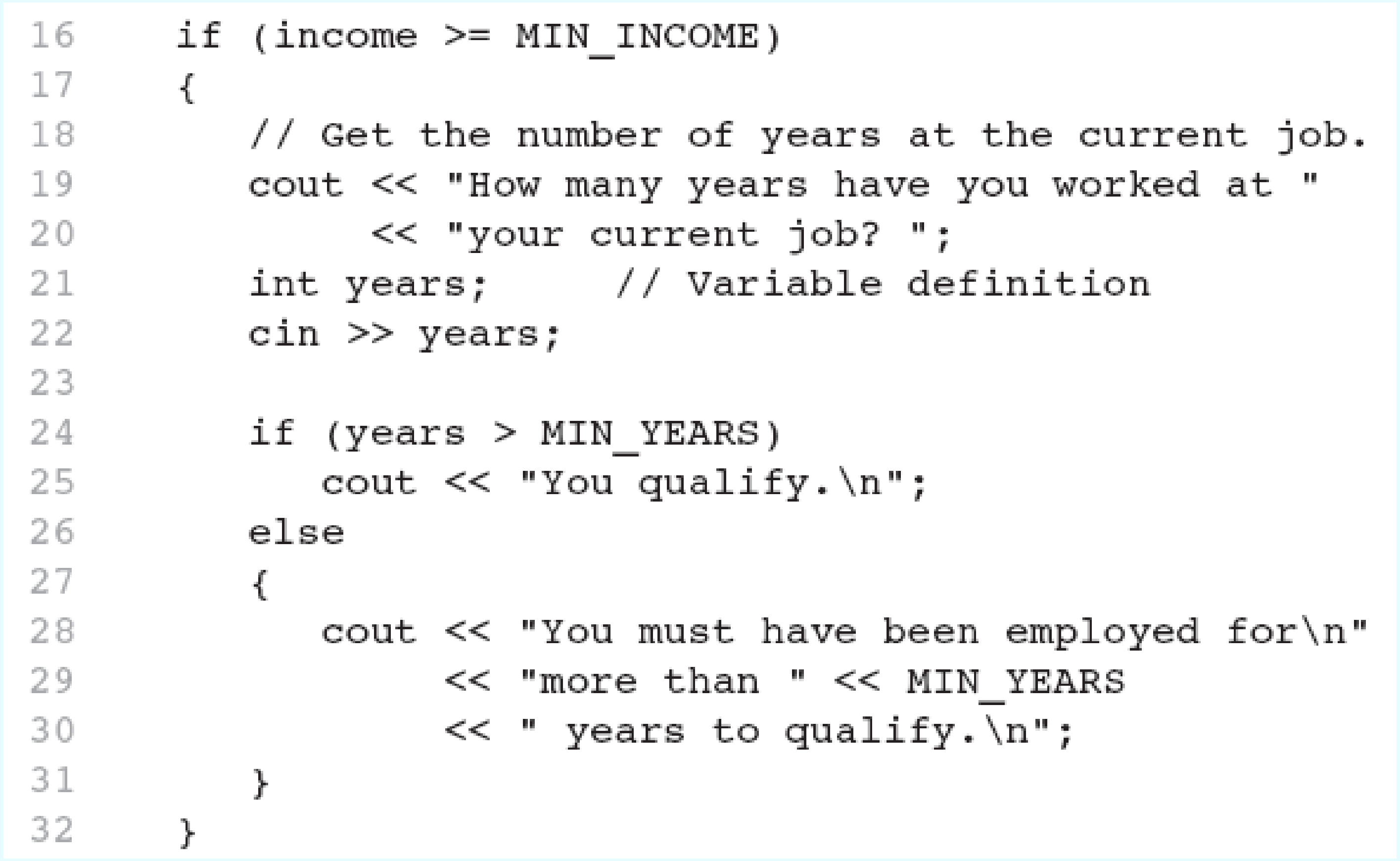
4.15

More About Blocks and Scope

More About Blocks and Scope

- Scope of a variable is the block in which it is defined, from the point of definition to the end of the block
- Usually defined at beginning of function
- May be defined close to first use

Inner Block Variable Definition in Program 4-29

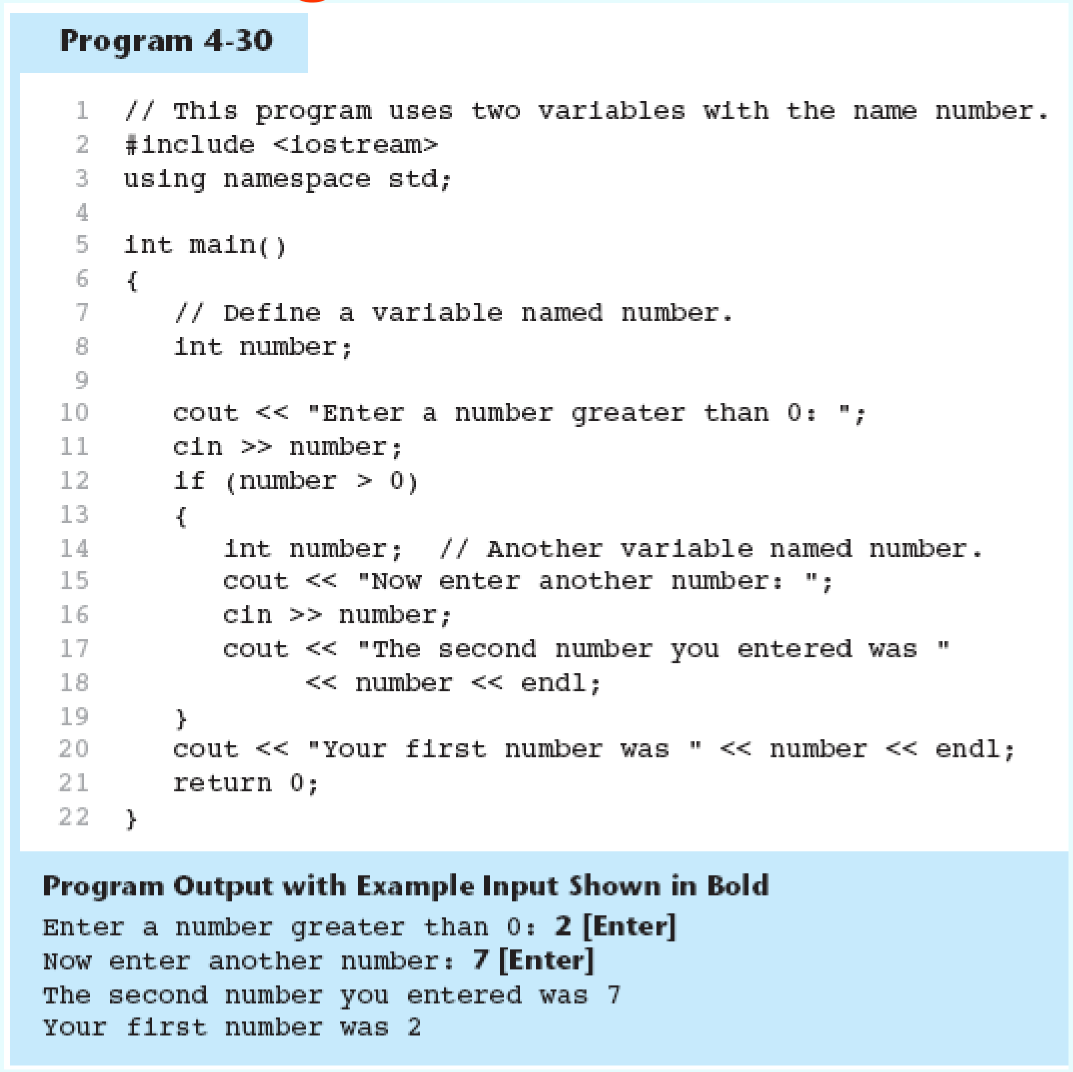


```
    if (income >= MIN_INCOME)
    {
       // Get the number of years at the current job.
       cout << "How many years have you worked at "
            << "your current job? ";
       int years;     // Variable definition
       cin >> years;

       if (years > MIN_YEARS)
          cout << "You qualify.\n";
       else
       {
          cout << "You must have been employed for\n"
               << "more than " << MIN_YEARS
               << " years to qualify.\n";
       }
    }
```

Variables with the Same Name

- Variables defined inside { } have <u>local</u> or <u>block</u> scope
- When inside a block within another block, can define variables with the same name as in the outer block.
  - When in inner block, outer definition is not available
  - Not a good idea

Two Variables with the Same Name in Program 4-30



Program 4-30

```
// This program uses two variables with the name number.
#include <iostream>
using namespace std;

int main()
{
   // Define a variable named number.
   int number;

   cout << "Enter a number greater than 0: ";
   cin >> number;
   if (number > 0)
   {
      int number;  // Another variable named number.
      cout << "Now enter another number: ";
      cin >> number;
      cout << "The second number you entered was "
           << number << endl;
   }
   cout << "Your first number was " << number << endl;
   return 0;
}
```

Program Output with Example Input Shown in Bold

```
Enter a number greater than 0: 2 [Enter]
Now enter another number: 7 [Enter]
The second number you entered was 7
Your first number was 2
```