Chapter 4: Selection Structures

Objectives

- In this chapter, you will learn about:
  - Selection criteria
  - The **`if-else`** statement
  - Nested **`if`** statements
  - The **`switch`** statement
  - Program testing
  - Common programming errors

Selection Criteria

- **`if-else`** statement: Implements a decision structure for two alternatives

  Syntax:

  if (condition)

  statement executed if condition is true;

  else

  statement executed if condition is false;

Selection Criteria (continued)

- The condition is evaluated to its numerical value:
  - A non-zero value is considered to be true
  - A zero value is considered to be false
- The **`else`** portion is optional
  - Executed only if the condition is false
- The condition may be any valid C++ expression

Relational Operators

- **Relational expression:** Compares two operands or expressions using **relational operators**

Relational Operator	Meaning	Example
<	Less than	`age < 30`
>	Greater than	`height > 6.2`
<=	Less than or equal to	`taxable <= 20000`
>=	Greater than or equal to	`temp >= 98.6`
==	Equal to	`grade == 100`
!=	Not equal to	`number != 250`

Table 4.1 C++'s Relational Operators

Relational Operators (continued)

- Relational expressions are evaluated to a numerical value of 1 or 0 only:
  - If the value is 1, the expression is true
  - If the value is 0, the expression is false
- **`char`** values are automatically coerced to **`int`** values for comparison purposes
- Strings are compared on a character by character basis
  - The string with the first lower character is considered smaller

Refer to pages 181,182 for more explanations and examples

Relational Operators (continued)

- Examples of string comparisons

Expression	Value	Interpretation	Comment
`"Hello"> "Good-bye"`	1	true	The first H in Hello is greater than the first G in Good-bye.
`"SMITH" > "JONES"`	1	true	The first S in SMITH is greater than the first J in JONES.
`"123" > "1227"`	1	true	The third character in 123, the 3, is greater than the third character in 1227, the 2.
`"Behop" > "Beehive"`	1	true	The third character in Behop, the h, is greater than the third character in Beehive, the second e.

Logical Operators

- AND (`&&`): Condition is true only if both expressions are true
- OR (`||`): Condition is true if either one or both of the expressions is true
- NOT (`!`): Changes an expression to its opposite state; true becomes false, false becomes true

Refer to page 183 for more explanations and examples

Logical Operators (continued)

Operator	Associativity
! unary - ++ --	Right to left
* / %	Left to right
+ -	Left to right
< <= > >=	Left to right
== !=	Left to right
&&	Left to right
||	Left to right
= += -= *= /=	Right to left

Table 4.2 Operator Precedence and Associativity

Refer to page 184 for more explanations and examples

A Numerical Accuracy Problem

- Comparing single and double precision values for equality (==) can lead to errors because values are stored in binary
- Instead, test that the absolute value of the difference is within an acceptable range
  - Example:

```
operandOne == operandTwo
abs(operandOne - operandTwo) < 0.000001
x/y == 0.35
```

The `if-else` Statement

- **`if-else`** performs instructions based on the result of a comparison
- Place statements on separate lines for readability
- Syntax:

```
if (expression)          <-- no semicolon here
  statement1;
else                     <-- no semicolon here
  statement2;
```

The `if-else` Statement (cont'd)

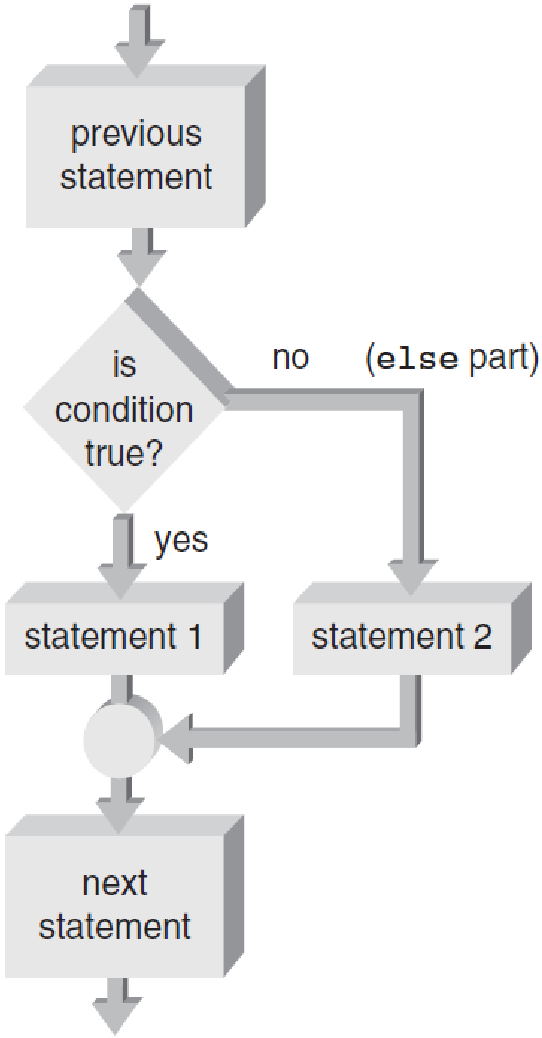


Figure 4.2
The `if-else` flowchart

The `if-else` Statement (continued)

Program 4.1

```
#include <iostream>
#include <cmath>
using namespace std;

int main()
{
  double radius;

  cout << "Please type in the radius: ";
  cin  >> radius;

  if (radius < 0.0)
    cout << "A negative radius is invalid" << endl;
  else
    cout << "The area of this circle is " << 3.1416 * pow(radius,2) << endl;

  return 0;
}
```

Refer to page 189 for more explanations and examples

Compound Statements

- **Compound statement:** A sequence of single statements contained between braces
  - Creates a block of statements
  - A block of statements can be used anywhere that a single statement is legal
  - Any variable declared within a block is usable only within that block
- **Scope:** The area within a program where a variable can be used
  - A variable's scope is based on where the variable is declared

Block Scope (continued)

```
{    // start of outer block
   int a = 25;
   int b = 17;

   cout << "The value of a is " << a
        <<" and b is " << b << endl;

   {    // start of inner block
     double a = 46.25;

     int c = 10;
     cout << "a is now " << a
          << " b is now " << b
          << " and c is " << c << endl;
   }   // end of inner block

   cout << "a is now " << a
        << " and b is " << b << endl;
}   // end of outer block
```

Refer to page 192 for more explanations and examples

One-Way Selection

- **One-way selection:** An **`if`** statement without the optional **`else`** portion

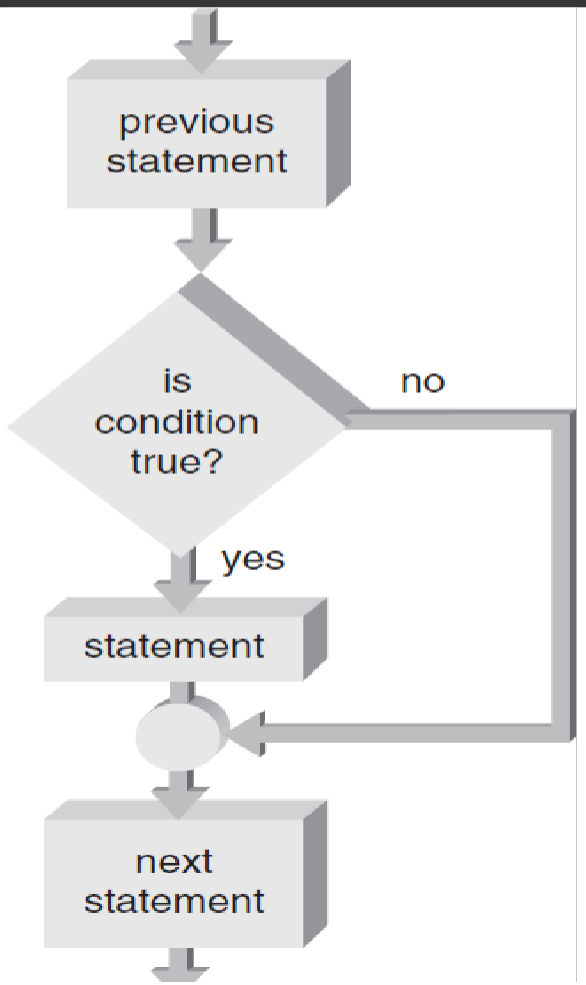


Figure 4.3 A one-way selection **`if`** statement

One-Way Selection

Program 4.3

```
#include <iostream>
using namespace std;

int main()
{
   const double LIMIT = 3000.0;
   int idNum;
   double miles;

   cout << "Please type in car number and mileage: ";
   cin  >> idNum >> miles;

   if (miles > LIMIT)
      cout << " Car " << idNum << " is over the limit.\n";
      cout << "End of program output.\n";

   return 0;
}
```

Problems Associated with the `if-else` Statement

- Common problems with **`if-else`** statements:
  - Misunderstanding what an expression is
  - Using the assignment operator (=) instead of the relational operator (==)

Problems Associated with the `if-else` Statement

Program 4.4

```
#include <iostream>
using namespace std;

int main()
{
  int age = 18;

  cout << "The value of the first expression is " << (age + 5) << endl;
  cout << "The value of the second expression is " << (age = 30) << endl;
  cout << "The value of the third expression is " << (age == 40) << endl;

  return 0;
}
```

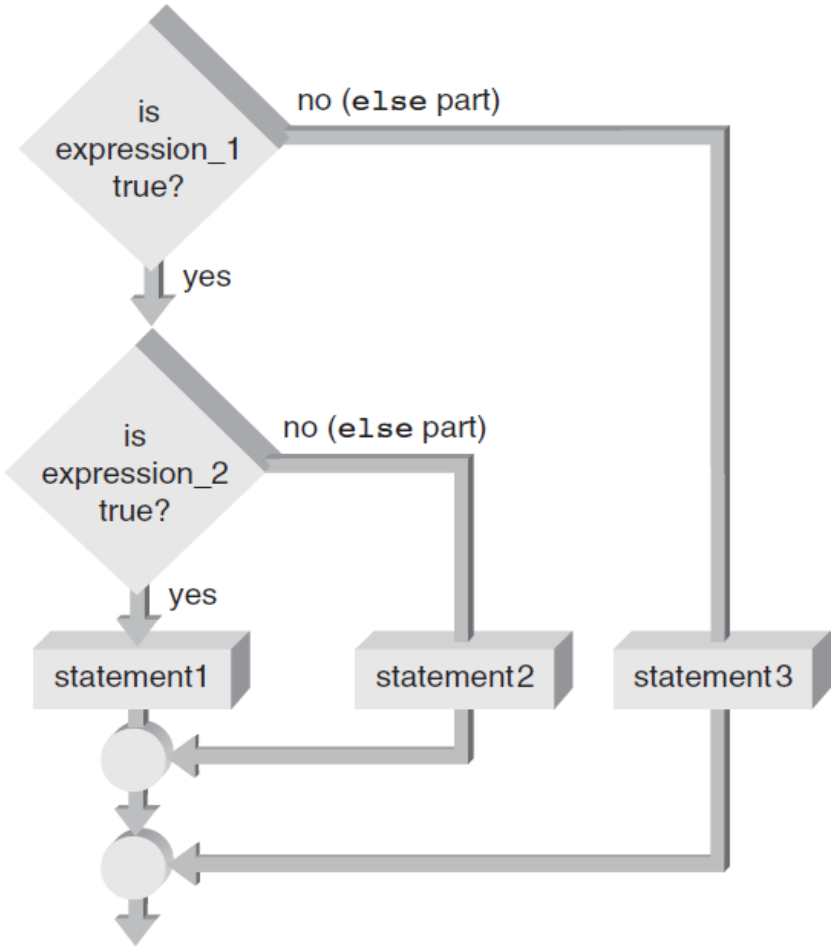
Refer to page 196 for more explanations and examples

Nested `if` Statements

- **`if-else`** statement can contain any valid C++ statement, including another **`if-else`**
- Nested **`if`** statement: an **`if-else`** statement completely contained within another **`if-else`**
- Use braces to block code, especially when inner **`if`** statement does not have its own **`else`**

Nested `if` Statements (continued)

Figure 4.4a
Nested within the `if` part

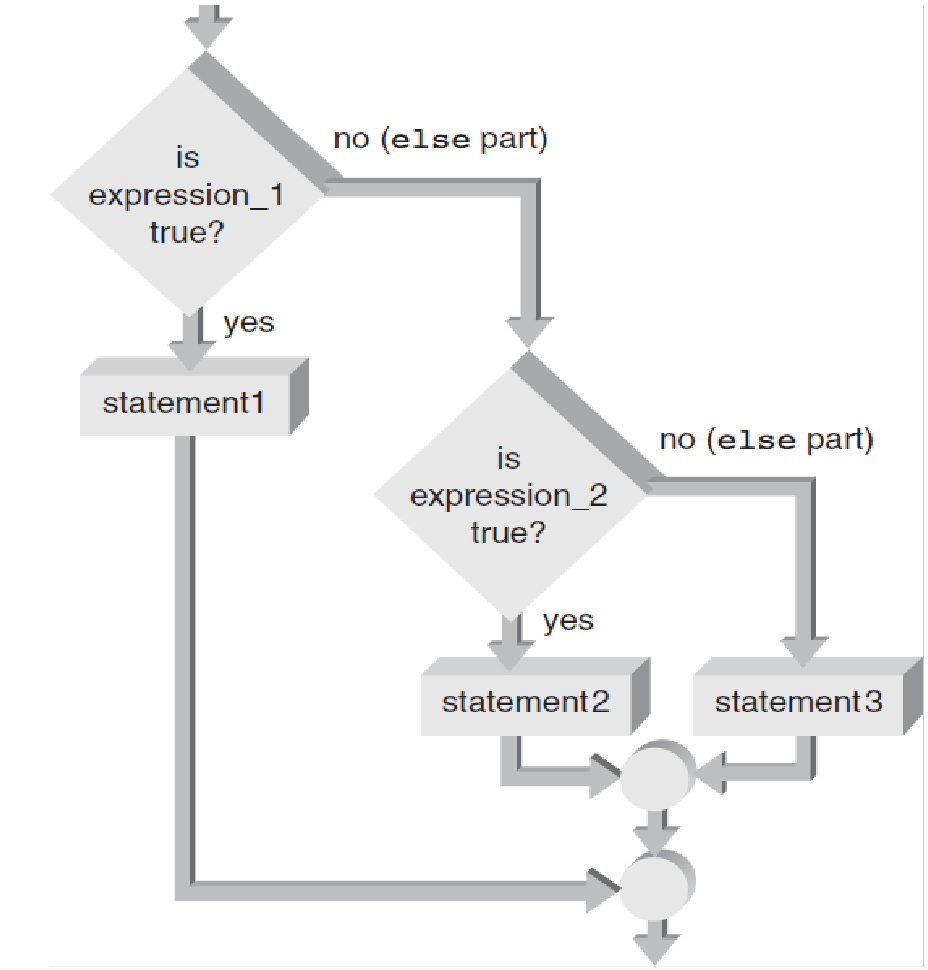


The `if-else` Chain

- **`if-else`** chain: A nested **`if`** statement occurring in the **`else`** clause of the outer **`if-else`**
- If any condition is true, the corresponding statement is executed and the chain terminates
- Final **`else`** is only executed if no conditions were true
  - Serves as a catch-all case
- **`if-else`** chain provides one selection from many possible alternatives

The `if-else` Chain (continued)

Figure 4.4b
Nested within the **`else`** part



The `if-else` Chain (continued)

- General form of an **`if-else`** chain

```
if (expression_1)
  statement1;
else if (expression_2)
  statement2;
else if (expression_3)
  statement3;
      .
      .
      .
else if (expression_n)
  statementn;
else
  last_statement;
```

The `if-else` Chain (continued)

To illustrate using an `if-else` chain, Program 4.5 displays an item's specification status corresponding to a letter input. The following input codes are used:

Specification Status	Input Code
Space exploration	S
Military grade	M
Commercial grade	C
Toy grade	T

The `if-else` Chain (continued)

Program 4.5

```
#include <iostream>
using namespace std;

int main()
{
  char code;

  cout << "Enter a specification code: ";
  cin  >> code;

  if (code == 'S')
    cout << "The item is space exploration grade.";
  else if (code == 'M')
    cout << "The item is military grade.";
  else if (code == 'C')
    cout << "The item is commercial grade.";
  else if (code == 'T')
    cout << "The item is toy grade.";
  else
    cout << "An invalid code was entered.";
  cout << endl;

  return 0;
}
```

The `switch` Statement

- **`switch`** statement: Provides for one selection from many alternatives
- **`switch`** keyword starts the statement
  - Is followed by the expression to be evaluated
- **`case`** keyword identifies a value to be compared to the switch expression
  - When a match is found, statements in this **`case`** block are executed
- All further cases after a match is found are executed unless a **`break`** statement is found

Refer to page 210 for more explanations and examples

The `switch` Statement (continued)

- **`default`** case is executed if no other case value matches were found
- **`default`** case is optional

Refer to page 212 for more explanations and examples

The `switch` Statement (continued)

Program 4.7

```
#include <iostream>
using namespace std;

int main()
{
  int opselect;
  double fnum, snum;

  cout << "Please type in tw
  cin  >> fnum >> snum;
  cout << "Enter a select co
  cout << "\n         1 for a
  cout << "\n         2 for m
  cout << "\n         3 for d
  cin  >> opselect;

  switch (opselect)
  {
    case 1:
      cout << "The sum of the numbers entered is " << fnum+snum;
      break;
    case 2:
      cout << "The product of the numbers entered is " << fnum*snum;
      break;
    case 3:
      cout << "The first number divided by the second is " << fnum/snum;
      break;
  }    // end of switch

  cout << endl;

  return 0;
}
```

A Case Study: Solving Quadratic Equations

- **Data validation:** Use defensive programming techniques to validate user input
  - Includes code to check for improper data before an attempt is made to process it further
- **Solving quadratic equations:** Use the software development procedure to solve for the roots of a quadratic equation

A Case Study: Solving Quadratic Equations

- **Step 1: Analyze the Problem:** The problem requires accepting three inputs—the coefficients a, b, and c of a quadratic equation. The outputs are the roots of the equation, found by using the given formulas.
- **Step 2: Develop a Solution:** A first attempt at a solution is using the user-entered values of a, b, and c to calculate a value for each root, as described by the following pseudocode:

  Display a program purpose message

  Accept user-input values for a, b, and c

  Calculate the two roots

  Display the values of the calculated roots

A Case Study: Solving Quadratic Equations

Taking into account all four limiting cases, the following pseudocode shows a refined solution for determining the roots of a quadratic equation correctly:

```
Display a program purpose message
Accept user-input values for a, b, and c
If a = 0 and b = 0 then
   Display a message saying that the equati
ElseIf a = zero then
   Calculate the single root equal to -c/b
   Display the single root
Else
   Calculate the discriminant
   If the discriminant > 0 then
     Solve for both roots using the given formulas
     Display the two roots
   ElseIf the discriminant < 0 then
     Display a message that there are no real roots
   Else
     Calculate the repeated root equal to -b/(2a)
     Display the repeated root
   EndIf
EndIf
```

A Case Study: Solving Quadratic Equations

- **Step 3: Code the Solution:**

Program 4.8

```
#include <iostream>
#include <cmath>
using namespace std;

// This program solves for the rc
int main()
{
  double a, b, c, disc, root1, rc

  cout << "This program calculate
  cout << "    quadratic equation
  cout << "              2\n";
  cout << "           ax + bx + c

  cout << "Please enter values fc
  cin  >>  a >> b >> c;

  if (a == 0.0 && b == 0.0)
    cout << "The equation is degenerate and has no roots.\n";
  else if (a == 0.0)
    cout << "The equation has the single root x = "
         << -c/b << endl;
  else
  {  // Start of compound statement for the outer else
    disc = pow(b,2.0) - 4 * a * c;     // calculate discriminant
    if (disc > 0.0)
    {
      disc = sqrt(disc);
      root1 = (-b + disc) / (2 * a);
      root2 = (-b - disc) / (2 * a);
      cout << "The two real roots are "
           << root1 << " and " << root2 << endl;
    }
    else if (disc < 0.0)
      cout << "Both roots are imaginary.\n";
    else
      cout << "Both roots are equal to " << -b / (2 * a) << endl;
  }  // End of compound statement for the outer else

  return 0;
}
```

A Closer Look: Program Testing

- Theory: A comprehensive set of test runs would test all combinations of input and computations, and would reveal all errors
- Reality: There are too many combinations to test for any program except a very simple one
- Example:
  - One program with 10 modules, each with five `if` statements, always called in the same order
  - There are 2^5 paths through each module, and more than 2^{50} paths through the program!

A Closer Look: Program Testing (continued)

- Conclusion: there is no error-free program, only one in which no errors have recently been encountered

Common Programming Errors

- Using the assignment operator (=) instead of the relational operator (==) for an equality test
- Placing a semicolon immediately after the condition
- Assuming a structural problem with an **`if-else`** causes the error instead of focusing on the data value being tested
- Using nested **`if`** statements without braces to define the structure

Summary

- Relational expressions, or conditions, are used to compare operands
- If the relation expression is true, its value is `1`; if false, its value is `0`
- Use logical operators `&&` (AND), `||` (OR), and `!` (NOT) to construct complex conditions
- **`if-else`** allows selection between two alternatives

Summary (continued)

- An `if` expression that evaluates to `0` is false; if non-zero, it is true
- `if` statements can be nested
- Chained `if` statement provides a multiway selection
- Compound statement: contains any number of individual statements enclosed in braces

Summary (continued)

- **`switch`** statement: Provides a multiway selection
- **`switch`** expression: Evaluated and compared to each **`case`** value
  - If a match is found, execution begins at that case's statements and continues unless a **`break`** is encountered