Chapter 4: Making Decisions

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

The if Statement

- General Format:
  \[
  \text{if (expression)} \quad \text{statement;}
  \]

The if Statement - What Happens

To evaluate:

- 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

```cpp
// This program averages three test scores
#include <iostream>
#include <cassert>
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

```cpp
if Statement Notes

1. Do not place ; after (expression)
2. Place statement; on a separate line after (expression), indented:
   
```cpp
   if (score > 90)
   
```cpp
   grade = 'A';

3. Be careful testing floats and doubles for equality
4. 0 is false; any other value is true

### Flowchart for Program 4-2 Lines 21 and 22

```
average > 96

Display "Congratulations! That's a high score!"

average < 96

Program Output with Example Input Shown in Bold

Enter 3 test scores and I will average them: 80 90 70 [Enter]
Your average is 83.3

Program Output with Other Example Input Shown in Bold

Enter 2 test scores and I will average them: 100 100 100 [Enter]
Your average is 100.

Expanding the if Statement

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

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

4.3

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.

General Format:

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

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.

Flowchart for Program 4-8 Lines 14 through 18

Directed to 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.
// Hard code constraints
// using subscripts and;
// if main():
// double num1, num2, quotient;
```

Continued...
4.5
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

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";
        } // endl else
}
4.6

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

```cpp
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

```cpp
21    // Determine the letter grade.
22    if (testScore >= A_SCORE)  
23       cout << "Your grade is A.\n";
24    else if (testScore >= B_SCORE)  
25       cout << "Your grade is B.\n";
26    else if (testScore >= C_SCORE)  
27       cout << "Your grade is C.\n";
28    else if (testScore >= D_SCORE)  
29       cout << "Your grade is D.\n";
30    else  
31       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.

```cpp
21    // Determine the letter grade.
22    if (testScore >= A_SCORE)  
23       cout << "Your grade is A.\n";
24    else if (testScore >= B_SCORE)  
25       cout << "Your grade is B.\n";
26    else if (testScore >= C_SCORE)  
27       cout << "Your grade is C.\n";
28    else if (testScore >= D_SCORE)  
29       cout << "Your grade is D.\n";
30    else if (testScore == 0)  
31       cout << "Your grade is F.\n";
32    else  
33       cout << "Invalid test score.\n";
```

This trailing else catches invalid test scores.
4.7

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

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

<table>
<thead>
<tr>
<th>Operator</th>
<th>Meaning</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&amp;&amp;</code></td>
<td><strong>AND</strong></td>
<td>New relational expression is true if both expressions are true</td>
</tr>
<tr>
<td>`</td>
<td></td>
<td>`</td>
</tr>
<tr>
<td><code>!</code></td>
<td><strong>NOT</strong></td>
<td>Reverses the value of an expression – true expression becomes false, and false becomes true</td>
</tr>
</tbody>
</table>

Logical Operators-Examples

```c
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

```c
21 // Determine the user's loan qualifications.
22 if (employed == 'Y' && recentGrad == 'Y')
23 { cout << "You qualify for the special " << interestRate << "% interest rate.\n";
24 } else
25 { cout << "You must be employed and have\n" << "graduated from college in the last two years to qualify.\n";
26 } }
```
The logical || Operator in Program 4-16

```cpp
23  // Determine the user's loan qualifications.
24  if (income >= MIN_INCOME || years > MIN_YEARS)
25    cout << "You qualify.\n";
26  else
27    {
28      cout << "You must earn at least $" << MIN_INCOME << " or have been "
29          << "employed more than " << MIN_YEARS << " years.\n";
30    }
31  
32  
```

The logical ! Operator in Program 4-17

```cpp
23  // Determine the user's loan qualifications.
24  if (!income || years <= MIN_YEARS)
25    {
26      cout << "You must earn at least $" << MIN_INCOME << " or have been "
27          << "employed more than " << MIN_YEARS << " years.\n";
28    }
29  else
30    
31    cout << "You qualify.\n";
32  
```

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)

Checking Numeric Ranges with Logical Operators

- Used to test to see if a value falls inside a range:
  ```cpp
  if (grade >= 0 && grade <= 100)
  cout << "Valid grade";
  ```
- Can also test to see if value falls outside of range:
  ```cpp
  if (grade <= 0 || grade >= 100)
  cout << "Invalid grade";
  ```
- Cannot use mathematical notation:
  ```cpp
  if (0 <= grade <= 100) //doesn't work!
  ```

4.9

Checking Numeric Ranges with Logical Operators

4.10

Menus
Menus

- **Menu-driven program**: program execution controlled by user selecting from a list of actions
- **Menu**: 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

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

```c++
1  #include <iostream>
2  using namespace std;
3
4  int main()
5  { // Validate the letter grade.
6      char grade;
7      cout << "Enter your letter grade (A-F): ";
8      cin >> grade;
9      if (grade == 'A' || grade == 'A+')
10         cout << "Excellent!" << endl;
11      else if (grade == 'B' || grade == 'B+')
12         cout << "Good!" << endl;
13      else if (grade == 'C' || grade == 'C+')
14         cout << "Average."
15      else if (grade == 'D' || grade == 'D+')
16         cout << "Needs improvement."
17      else if (grade == 'F' || grade == 'F+')
18         cout << "Very poor."
19      else
20         cout << "Invalid grade."
21  }
22
23  return 0;
```

Comparing Characters and Strings

4.12
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 ASCII value of '2' (50).
- Lowercase letters have higher ASCII codes than uppercase letters, so 'a' > 'Z'.

Comparing string Objects

- Like characters, strings are compared using their ASCII values.

```c++
string name1 = "Mary";
string name2 = "Mark";
name1 > name2 // true
name1 <= name2 // false
name1 == name2 // true
name1 < "Mary Jane" // true
```

Relational Operators Compare Characters in Program 4-20

```c++
10    // Get a character from the user.
11    cout << "Enter a digit or a letter: ";
12    ch = cin.get();
13    // Determine what the user entered.
14    if (ch >= '0' && ch <= '9')
15        cout << "You entered a digit.";
16    else if (ch >= 'a' && ch <= 'z')
17        cout << "You entered a lowercase letter.";
18    else if (ch >= 'A' && ch <= 'Z')
19        cout << "You entered an uppercase letter.";
20    else
21        cout << "That is not a digit or a letter.";
```

Relational Operators Compare Strings in Program 4-21

```c++
26    // Determine and display the correct price
27    if (partNum == "S-Mark")
28        cout << "The price is $" << PRICE_A << endl;
29    else if (partNum == "S-29")
30        cout << "The price is $" << PRICE_B << endl;
31    else
32        cout << partNum << " is not a valid part number.";
```

The Conditional Operator

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

```
x<0 ? y=10 : z=20;
```

The characters in each string must match before they are equal.
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

```c
// This program calculates a consultant's charges at 950
// per hour, for a minimum of 5 hours. The if operator
// allows for charges of less than 5 hours worked.
#include <iostream>
#define HOURS 5
int main()
{
    int hours worked;
    double hourly rate = 950.0;
    double total charges = 0.0;

    // Get the hours worked.
    std::cin >> hours worked;

    // Determine the hours to charge for.
    if (hours worked < HOURS)
    {
        total charges = hours worked * hourly rate;
    }
    else
    {
        total charges = HOURS * hourly rate;
    }

    // Display the charges for.
    std::cout << "The charges are ", total charges << "
    // total charges;
    return 0;
}
```

4.14

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

```c
// The switch statement in this program tells the user something
// about how much memory the user uses per hour of use.
#include <iostream>

int main()
{
    int memory used;
    char choice;

    std::cout << "How much memory do you want to use: A, B, or C? \n    // using uppercase only."
    std::cin >> memory used;

    switch (memory used)
    {
        case 'A':
            std::cout >> "you selected A."
            break;
        case 'B':
            std::cout >> "you selected B."
            break;
        case 'C':
            std::cout >> "you selected C."
            break;
        default:
            std::cout >> "you did not select A, B, or C!"
            break;
    }
    return 0;
}
```
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 **exp1** through **expn**
3) If **expression** matches value **expi**, the program branches to the statement following **expi** 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

1. Used to exit a switch statement
2. 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 4-23

```c
// This program is carefully constructed to use the "fall through"
// behavior of the switch statement.
switch (score) {  // conditional expression
    case 100:  // body of first statement
        printf("A final grade of \"A\".");
        break;  // terminate the switch
    case 90:  // body of second statement
        printf("Final grade is \"B\".");
        break;  // terminate the switch
    case 80:  // body of third statement
        printf("Final grade is \"C\".");
        break;  // terminate the switch
    default:  // body of default statement
        printf("The score is \"D\" or below."));
        break;  // terminate the switch
    }  // end switch
return 0;  // end of main function
```

---

Using switch in Menu Systems

1. **switch** statement is a natural choice for menu-driven program:
2. **display** the menu
3. **then**, get the user’s menu selection
4. **use** user input as **expression** in switch statement
5. **use** menu choices as **expr** in **case** statements
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.

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

```cpp
16 if (Income >= MIN_INCOME) {
17   // Get the number of years at the current job.
18   cout << "How many years have you worked at " << "your current job? ";
19   int years; // Variable definition
20   cin >> years;
21   if (years > MIN_YEARS) {
22     cout << "You qualify.\n";
23     } else {
24       cout << "You must have been employed for\n" << "more than " "MIN_YEARS\n" << "years to qualify\n";
25     }
26   }
```

Variables with the Same Name

- Variables defined inside `{ }` have local or block 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.

Variables with the Same Name

- Variables defined inside `{ }` have local or block 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

```cpp
16 Program 4-30:
17 // This program has two variables with the same name.
18 // Name them carefully.
19 // Note: "number" is a reserved word.
20 int number1;
21 int number2;
22 // Define a variable named number.
23 if (number1 > number2) {
24   cout << "First number greater than second.\n";
25   } else {
26     cout << "Second number greater than first.\n";
27     }
28 // Define a variable named number.
29 int number3;
30 number3 = number1;
31 // Define another variable named number.
32 char name[10]; // Reserve name for a string.
33 char name[10] = "name"; // Reserve name for a string.
34 // Define another variable named number.
35 int number4;
36 number4 = number3;
37 // Define another variable named number.
38 int number5;
39 number5 = number4;
```

Program Output with Example Input: Name is Held

```
Enter a number greater than 0: 0 [ERROR]
```