COIS1020H: Programming for Computing Systems

Chapter 4
Making Decisions

Understanding Logic-Planning Tools and Decision Making

- **Pseudocode**
  - Tool that helps programmers plan a program's logic by writing plain English statements
- **Flowchart**
  - You write the steps in diagram form as a series of shapes connected by arrows

Understanding Logic-Planning Tools and Decision Making

- **Sequence structure**
  - One step follows another unconditionally
  - Sometimes, logical steps do not follow in an unconditional sequence

Understanding Logic-Planning Tools and Decision Making (cont'd.)

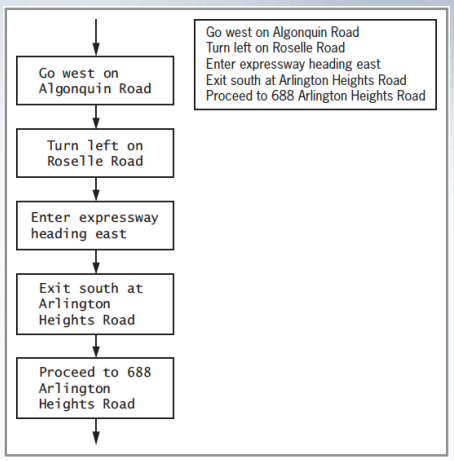


Figure 4-1 Flowchart and pseudocode for a series of sequential steps

Understanding Logic-Planning Tools and Decision Making (cont'd.)

- **Decision structure**
  - Involves choosing between alternative courses of action based on some value within a program
  - All computer decisions are yes-or-no decisions
    - When reduced to their most basic form

Understanding Logic-Planning Tools and Decision Making (cont'd.)

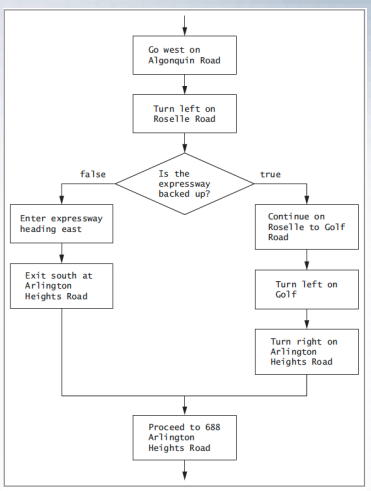


Figure 4-2 Flowchart including a decision

Making Decisions Using the `if` Statement

- **`if` statement**
  - Used to make a single-alternative decision
- **Block**
  - One or more statements contained within a pair of curly braces

Making Decisions Using the `if` Statement (cont'd.)

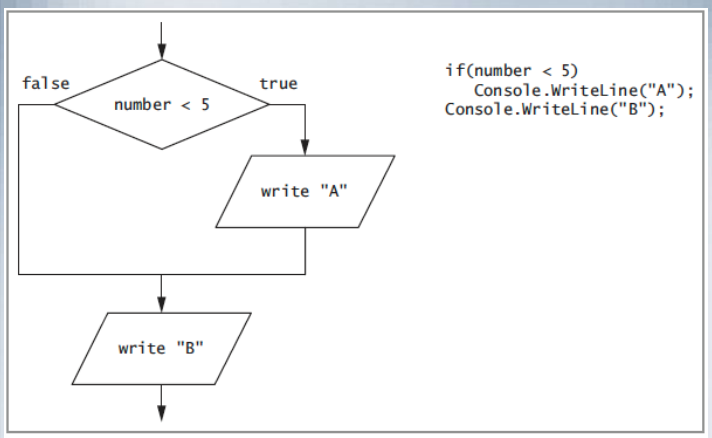


Figure 4-3 Flowchart and code including a typical `if` statement followed by a separate statement

Making Decisions Using the `if` Statement (cont'd.)

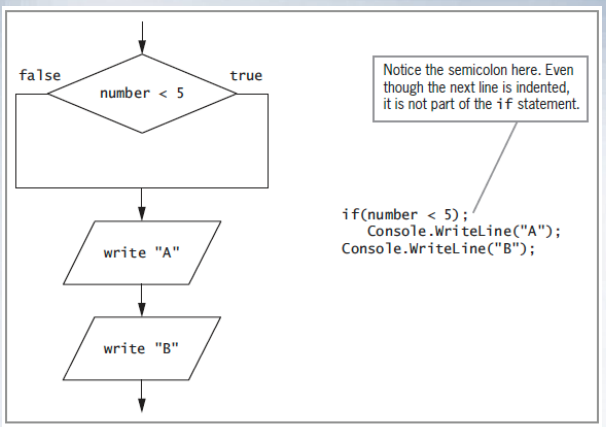


Figure 4-4 Flowchart and code including an `if` statement with a semicolon following the `if` expression

Making Decisions Using the `if` Statement (cont'd.)

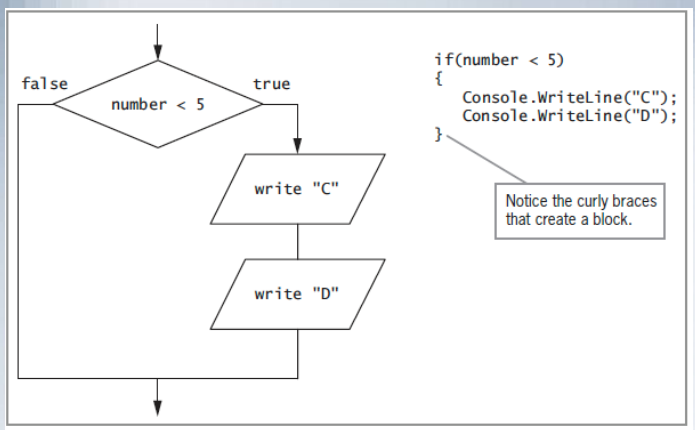


Figure 4-5 Flowchart and code including a typical `if` statement containing a block

Making Decisions Using the `if` Statement (cont'd.)

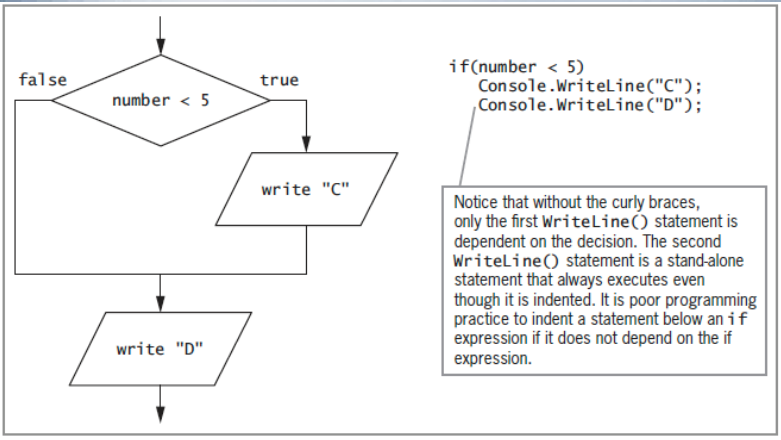


Figure 4-6 Flowchart and code including an `if` statement that is missing curly braces or that has inappropriate indenting

Making Decisions Using the `if` Statement (cont'd.)

- **Nested `if`**
  - One decision structure is contained within another

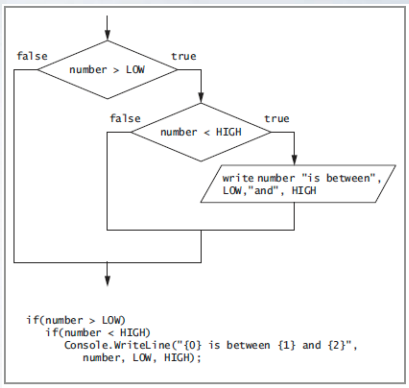


Figure 4-7 Flowchart and code showing the logic of a nested `if`

Making Decisions Using the `if` Statement (cont'd.)

```
using System;
public class NestedDecision
{
   public static void Main()
   {
      const int HIGH = 10, LOW = 5;
      string numberString;
      int number;
      Console.Write("Enter an integer ");
      numberString = Console.ReadLine();
      number = Convert.ToInt32(numberString);
        if(number > LOW)
          if(number < HIGH)
            Console.WriteLine("{0} is between {1} and {2}",
              number, LOW, HIGH);
   }
}
```

Enter an integer 8
8 is between 5 and 10

Enter an integer 4

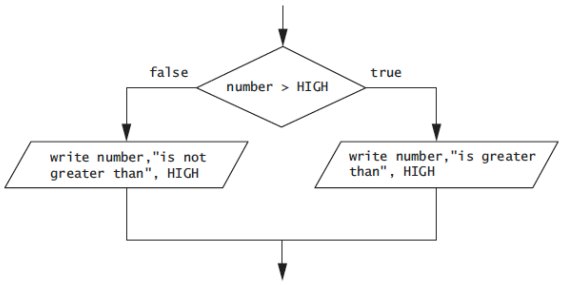
Enter an integer 10

Figure 4-8 Program using nested `if`

Making Decisions Using the `if-else` Statement

- **Dual-alternative decisions**
  - Have two possible outcomes
- **`if-else` statement**
  - Used to perform one action when a Boolean expression evaluates as `true`
    - And an alternate action when it evaluates as `false`

Making Decisions Using the `if-else` Statement (cont'd.)




```
if(number > HIGH)
   Console.WriteLine("{0} is greater than {1}",
      number, HIGH);
else
   Console.WriteLine("{0} is not greater than {1}",
      number, HIGH);
```

Figure 4-10 Flowchart and code showing the logic of a dual-alternative `if-else` statement

Making Decisions Using the `if-else` Statement (cont'd.)

```
using System;
public class IfElseDecision
{
   public static void Main()
   {
      const int HIGH = 10;
      string numberString;
      int number;
      Console.Write("Enter an integer ");
      numberString = Console.ReadLine();
      number = Convert.ToInt32(numberString);
      if(number > HIGH)
        Console.WriteLine("{0} is greater than {1}",
          number, HIGH);
      else
        Console.WriteLine("{0} is not greater than {1}",
          number, HIGH);
   }
}
```

Enter an integer 8
8 is not greater than 10

Enter an integer 32
32 is greater than 10

Figure 4-11 Program with a dual-alternative `if-else` statement

Using Compound Expressions in `if` Statements

- You can combine multiple decisions into a single `if` statement
  - Using a combination of AND (&&) and OR (||) operators

17

Using the Conditional AND Operator

- **Conditional AND operator**
  - Determines whether two expressions are both true
  - Written as two ampersands (`&&`)
  - You must include a complete Boolean expression on each side of the operator
- **Short-circuit evaluation**
  - Expressions in each part of an AND expression are evaluated only as much as necessary
    - if the first condition is `false`, the second condition is not evaluated (overall condition will always be `false`)

18

Using the Conditional AND Operator (cont'd.)

```
// using &&
  if(age >= 0 && age < 120)
     Console.WriteLine("Age is valid");
// using nested ifs
  if(age >= 0)
     if(age < 120)
       Console.WriteLine("Age is valid");
```

Figure 4-13 Comparison of the && operator and nested `if` statements

- Common application of Conditional AND
  - Determine if a number falls into a range (0 ≤ grade ≤ 100)

    if ((grade >= 0) && (grade <= 100)) // correct

    if (0 <= grade <= 100) // incorrect

Using the Conditional OR Operator

- **Conditional OR operator**
  - Used when you want some action to occur even if only one of two conditions is `true`
  - Short circuited as well (if first condition is `true`)
  - Written as `||`

```
// using ||
 if(age < 0 || age >= 120)
   Console.WriteLine("Age is not valid");
// using nested ifs
 if(age < 0)
   Console.WriteLine("Age is not valid");
 else
   if(age >= 120)
      Console.WriteLine("Age is not valid");
```

Figure 4-14 Comparison of the `||` operator and nested `if` statements

Using the Logical AND and OR Operators

- **Boolean logical AND** (&) and **Boolean logical inclusive OR** (|) operators
  - Work similar to their && and || (*conditional* AND and OR) counterparts
  - They do not support short-circuit evaluation
  - Can lead to a **side effect** (unintended consequence)
  - they are applied in a bit-wise approach

21

Combining AND and OR Operators

```
using System;
public class MovieDiscount
{
   public static void Main()
   {
      int age = 10;
      char rating = 'R';
      const int CHILD_AGE = 12;
      const int SENIOR_AGE = 65;
      Console.WriteLine("When age is {0} and rating is {1}",
        age, rating);
      if((age <= CHILD_AGE || age >= SENIOR_AGE) &&
        rating == 'G')
        Console.WriteLine("Discount applies");
      else
        Console.WriteLine("Full price");
   }
}
```

Figure 4-15 Movie ticket discount program using parentheses to alter precedence of Boolean evaluations

22

Multi-Alternative `If` Statements

- Multi-alternative `if` statements are used if it is necessary to have more than just two alternatives
- `if-else-if` *statements* can become very complex.
- Imagine the following decision set.

  if it is very cold, wear a heavy coat,
  else, if it is chilly, wear a light jacket,
  else, if it is windy, wear a windbreaker,
  else, if it is hot, wear no jacket.

Multi-Alternative `If` Statements

```
if (expression)
  statement or block
else if (expression)
  statement or block
else if (expression)
  statement or block
. . .              // Put as many else ifs as needed
else
  statement or block
```

- Care must be used since `else` statement matches up with the immediately preceding unmatched `if` statement.
- Example: Assume you are to assign a letter grade based on an numeric grade that has been input

LetterGrade.cs

```
// This program asks the user to enter a numeric mark and assigns and prints a
// letter grade for the score. The program displays an error message if an invalid
// mark (> 100) is entered.

using System;

public static class LetterGrade
{
  public static void Main()
  {
    int mark;        // numeric mark
    char grade;      // letter grade
    // Get the numeric test score.
    Console.Write("Enter your numeric mark => ");
    mark = Convert.ToInt32(Console.ReadLine());
```

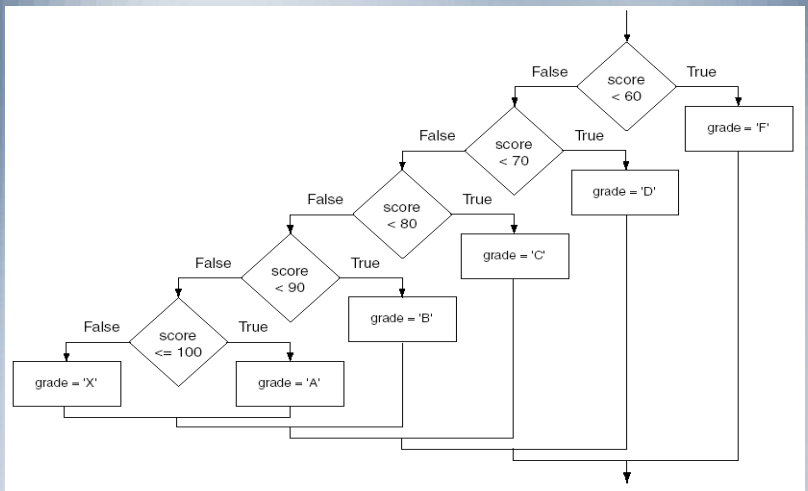
LetterGrade.cs (cont'd)

```
    // Display the grade.
    if (mark < 60)
        grade = 'F';
    else if (mark < 70)
        grade = 'D';
    else if (mark < 80)
        grade = 'C';
    else if (mark < 90)
        grade = 'B';
    else if (mark <= 100)
        grade = 'A';
    else        // Invalid score
    {
        Console.WriteLine("Invalid mark.");
        grade = 'X';
    }
    Console.WriteLine("The letter grade for {0} is {1}.", mark, grade);
    Console.ReadLine();
  }
}
```

Output 1:
Enter your numeric mark => 78
The letter grade for 78 is C.

Output 2:
Enter your numeric mark => 103
The letter grade for 103 is X.

if-else-if Flowchart



Decisions Using the `switch` Statement

- **`switch` structure**
  - Tests a single variable against a series of exact matches
- Keywords
  - **`switch`, `case`, `break`**, and **`default`**
- C# supports "No fall through rule"
  - Not allowing code to reach the end of a `case`
  - Use a `break` statement at the end of each `case` (or `case`s)

Decisions Using the `switch` Statement

```
if(year == 1)
   Console.WriteLine("Freshman");
else
   if(year == 2)
      Console.WriteLine("Sophomore");
   else
      if(year == 3)
        Console.WriteLine("Junior");
      else
        if(year == 4)
          Console.WriteLine("Senior");
        else
          Console.WriteLine("Invalid year");
```

Figure 4-18 Executing multiple alternatives using a series of `if` statements

Decisions Using the `switch` Statement

```
switch(year)
{
   case 1:
      Console.WriteLine("Freshman");
      break;
   case 2:
      Console.WriteLine("Sophomore");
      break;
   case 3:
      Console.WriteLine("Junior");
      break;
   case 4:
      Console.WriteLine("Senior");
      break;
   default:
      Console.WriteLine("Invalid year");
      break;
}
```

Figure 4-19 Executing multiple alternatives using a `switch` statement

Decisions Using the `switch` Statement

- The selector (`year` in the previous example) must be an integral data type (integer, character, enumerated)
- The keyword `case` is followed by one of the possible values that might equal the selector
- A `switch` does not need a `default` case
  - Good programming practice to include one
- You can use multiple labels to govern a list of statements

Decisions Using the `switch` Statement

```
switch(year)
{
   case 1:
      Console.WriteLine("Freshman");
      break;
   case 2:
      Console.WriteLine("Sophomore");
      break;
   case 3:
   case 4:
      Console.WriteLine("Upperclass");
      break;
   default:
      Console.WriteLine("Invalid year");
      break;
}
```

Cases 3 and 4 are both "Upperclass".

Figure 4-20 Example `switch` structure using multiple labels to execute a single statement block

PetFood.cs

```
// This program demonstrates a menu-driven interface using a switch statement
using System;

public static class PetFood
{
  public static void Main()
  {
     char foodGrade;   // Grade of pet food

     // Prompt the user for a grade of pet food.
     Console.WriteLine("Our pet food is available in three grades: A, B, and C");
     Console.Write("Which do you want pricing for => ");
     foodGrade = Convert.ToChar(Console.ReadLine());
```

PetFood.cs (cont'd)

```
// Display pricing for the selected grade.
switch(foodGrade)
{
  case 'a':   case 'A':
      Console.WriteLine("75 cents per kg.");
      break;
  case 'b':   case 'B':
      Console.WriteLine("50 cents per kg.");
      break;
  case 'c':   case 'C':
      Console.WriteLine("35 cents per kg.");
      break;
  default:
      Console.WriteLine("Invalid choice.");
      break;
}
Console.ReadLine();
  }
}
```

Output 1:
Our pet food is available in three grades:
A, B, and C.
Which do you want pricing for? b
50 cents per kg.

Output 2:
Our pet food is available in three grades:
A, B, and C.
Which do you want pricing for? A
75 cents per kg.

Output 3:
Our pet food is available in three grades:
A, B, and C.
Which do you want pricing for? x
Invalid choice

PetFood.cs (cont'd)

```
// Alternate form of the switch statement using the char.ToUpper() method
// to reduce the number of cases

switch(char.ToUpper(foodGrade))
{
  case 'A':
      Console.WriteLine("75 cents per kg.");
      break;
  case 'B':
      Console.WriteLine("50 cents per kg.");
      break;
  case 'C':
      Console.WriteLine("35 cents per kg.");
      break;
  default:
      Console.WriteLine("Invalid choice.");
      break;
}
```

Using the Conditional Operator

- **Conditional operator**
  - Used as an abbreviated version of the `if-else` statement
  - A ternary operator
  - Has limited application
- Syntax

```
testExpression ? trueResult : falseResult;
```

- Example

```
biggerNum = (a > b) ? a : b;
```

36

Using the NOT Operator

- **NOT operator**
  - Written as an exclamation point (`!`)
  - Negates the result of any Boolean expression
  - Used more to improve readability of a program than an absolute must especially with Boolean data types: while (!done)

- Example: consider car insurance premiums

  if (age > 25) then premium = 200; else premium = 125;

  if (!(age > 25)) then premium = 125; else premium = 200;

37

Avoiding Common Errors When Making Decisions

- Most frequent errors include:
  - Using the assignment operator instead of the comparison operator (= vs. ==)
  - Inserting a semicolon after the Boolean expression in an `if` statement (sematic versus syntax error)

    if (salary > 50000); // incorrect
  - Failing to block a set of statements with curly braces
  - Failing to include a complete Boolean expression on each side of an `&&` or `||` operator

    if(grade >= 0 && <= 100) // incorrect

    if(grade >= 0 && grade <= 100) // correct

38

Performing Accurate and Efficient Range Checks

- **Range check**
  - A series of `if` statements that determine whether a value falls within a specified range
- Problem

```
if(saleAmount >= 1000)
  commissionRate = 0.08;
if(saleAmount >= 500)
  commissionRate = 0.06;
if(saleAmount <= 499)
  commissionRate = 0.05;
```

DON'T DO IT

Although it was not the programmer's intention, both of the first two `if` statements are true for any `saleAmount` greater than or equal to 1000.

Performing Accurate and Efficient Range Checks (cont'd.)

- Solution

```
if(saleAmount >= 1000)
   commissionRate = 0.08;
else if(saleAmount >= 500)
   commissionRate = 0.06;
else commissionRate = 0.05;
```

- Never substitute a series of single if statements for a multi-alternative if statement

Using && and II Appropriately

- Problem
  - Print an error message when an employee's hourly pay rate is under $5.65 ***and*** when an employee's hourly pay rate is over $60
- Solution

```
if(payRate < 5.65 || payRate > 60)
   Console.WriteLine ("Error in pay rate");
```

41

Using the ! Operator Correctly

- Problem
  - Make sure that if the sales code is not 'A' or 'B', the customer gets a 10% discount
- Correct

```
if(salesCode != 'A' && salesCode != 'B')
   discount = 0.10;
if(!(salesCode == 'A' || salesCode == 'B'))
   discount = 0.10;
```

- Incorrect

```
if(salesCode != 'A' || salesCode != 'B')
   discount = 0.10;
```

42

Summary

- A flowchart is a pictorial tool that helps you understand a program's logic
- You use an `if` statement to make a single-alternative decision
- When you make a dual-alternative decision, you can use an `if-else` statement
- Conditional AND operator determines whether two expressions are both `true`
- Use the conditional OR operator when you want some action to occur when one or both of two conditions are `true`

43

Summary (cont'd.)

- AND operators take precedence
- The `switch` statement tests a single variable against a series of exact matches
- The conditional operator is used as an abbreviated version of the `if-else` statement
- You use the NOT operator to negate the result of any Boolean expression
- Common errors when making decisions

44