CPSC 1012

Programming Fundamentals
Lessons 6-8

Let’s review

What we've learned already...

1. The `Write`, `WriteLine` , and `ReadLine` methods
2. Integral, floating point, decimal, `bool`, `char`, and `string` data types
    a. Converting between simple data types
    b. Simple data type ranking
    c. Suffixes
3. Arithmetic operators
    a. Integer division
    b. Precedence
    c. The `Math` class
    d. Combined assignment operators
4. Constants

This week's agenda

By the end of this week, you will be able to:

- ❑ Use relational, logical, and conditional operators
- ❑ Create programs using `if` and `switch` statements

Introduction to Decision Structures

Programs can decide which statements to execute based on a condition.

```
if (radius < 0) {
    Console.WriteLine("Incorrect input");
}
else {
    double area = radius * radius * 3.14159;
    Console.WriteLine("Area is " + area);
}
```

Relational Operators

C# Operator	Math Symbol	Name	Example	Result if radius = 6
<	<	Less than	`radius < 0`	
<=	≤	Less than or equal to	`radius <= 0`	
>	>	Greater than	`radius > 0`	
>=	≥	Greater than or equal to	`radius >= 0`	
==	=	Equal to	`radius == 0`	
!=	≠	Not equal to	`radius != 0`	

Relational Operators

C# Operator	Math Symbol	Name	Example	Result if radius = 6
<	<	Less than	`radius < 0`	`false`
<=	≤	Less than or equal to	`radius <= 0`	`false`
>	>	Greater than	`radius > 0`	`true`
>=	≥	Greater than or equal to	`radius >= 0`	`true`
==	=	Equal to	`radius == 0`	`false`
!=	≠	Not equal to	`radius != 0`	`true`

AdditionQuiz.cs

```
static void Main(string[] args){
    const int minNumber= 1;
    const int maxNumber= 9;
    Random keygen = new Random();
    // Generate two random numbers between 1 and 9
    int number1 = keygen.Next(minNumber, maxNumber+ 1);
    int number2 = keygen.Next(minNumber, maxNumber+ 1);
    // Show question
    Console.Write($"What is {number1} + {number2} = ? ");
    // Receive answer
    int answer = int.Parse(Console.ReadLine());
    // Display result
    Console.WriteLine($"{number1} + {number2} = {answer} is " +
        $"{(number1 + number2) == answer}");
}
```

One-way `if` statements

- An `if` statement is a construct that enables a program to specify alternatives paths of execution.
- C# has several types of selection statements: one-way `if` statements, two-way `if-else` statements, nested `if` statements, multi-way `if-else` statement, `switch` statement, and conditional operators.
- A one-way if statement executes an action if the specified condition is `true`. If the condition is `false`, nothing is done.

One-way if statements

The syntax for a one-way `if` statement is as follows:

```
if (boolean_expression) {
    statement(s)
}
```

Example:

```
if (radius > 0) {
    double area = Math.Pow(radius,Math.PI);
    Console.WriteLine($"The area for the circle of radius {radius} is {area}");
}
```

SimpleIfDemo.cs

```
static void Main(string[] args){
    Console.Write("Enter an integer: ");
    int number = int.Parse(Console.ReadLine());
    if (number % 5 == 0){
        Console.WriteLine("HiFive");
    }
    if (number % 2 == 0){
        Console.WriteLine("HiEven");
    }
}
```

Two way `if-else` statements

- An `if-else` statement decides the execution path based on whether the condition is `true` or `false`.
- Syntax:

```
if (boolean-expression) {
    statement(s)-for-true-case
}
else {
    statement(s)-for-false-case
}
```

Two way `if-else` statements

```
if (number % 2 == 0) {
    Console.WriteLine($"{number} is even");
}
else {
    Console.WriteLine($"{number} is odd");
}
```

Nested `if` and multi-way `if-else` statements

An if statement can be inside another if statement to form a nested if statement.

```
if (isHungry)
    if (money > 10)
        Console.WriteLine("I'll buy lunch.");
    else
        Console.WriteLine("no lunch for me");
else
    Console.WriteLine("no lunch for me");
```

SubtractionQuiz.cs

```
// 1. Generate two random single-digit integers

Random keygen = new Random();
int number1 = keygen.Next(1, 10);
int number2 = keygen.Next(1, 10);

// 2. If number 1 < number 2, swap number 1 with number 2

if (number1 < number2){
    int temp = number1;
    number1 = number2;
    number2 = temp;
}
```

SubtractionQuiz.cs

```
// 3. Prompt the student to answer: "What is number1 - number2?"

Console.Write($"What is {number1} - {number2} = ? ");
int answer = int.Parse(Console.ReadLine());

// 4. Check the student's answer and display whether the answer is correct.

if (number1 - number2 == answer){
    Console.WriteLine("You are correct.");
}
else {
    Console.WriteLine("You answer is wrong.");
    Console.WriteLine($"{number1} - {number2} = {number1 - number2}");
}
```

ComputeAndInterpretBMI.cs

```
// Prompt the user to enter the weight in pounds

Console.Write("Enter weight in pounds: ");
double weight = double.Parse(Console.ReadLine());

// Prompt the user to enter the height in inches

Console.Write("Enter height in inches: ");
double height = double.Parse(Console.ReadLine());
const double kilogramPerPound = 0.45359537, metersPerInch = 0.0254;

// Compute BMI

double weightInKilograms = weight * kilogramPerPound;
double heightInMeters = height * metersPerInch;
double bmi = weightInKilograms/ ( heightInMeters * heightInMeters);
```

ComputeAndInterpretBMI.cs

```
// Display result

Console.WriteLine("BMI is " + bmi);
if (bmi < 18.5)
    Console.WriteLine("BMI Category: Underweight");
else if (bmi < 25)
    Console.WriteLine("BMI Category: Normal");
else if (bmi < 30)
    Console.WriteLine("BMI Category: Overweight");
else
    Console.WriteLine("BMI Category: Obese");
```

Logical Operators

The logical operators `!`, `&&`, `||`, and `^` can be used to create a compound Boolean expression.

`!` represents “not” and is a logical negation.

`&&` represents “and” and is a logical conjunction.

`||` represents “or” and is a logical disjunction.

`^` represents “exclusive or” and is a logical exclusion.

! Operator

- If `p` is `true`, what is `!p` ?
- If `p` is `false`, what is `!p` ?
- If `age = 24`, what is `!(age > 18)` ?
- If `weight = 140`, what is `!(weight == 150)` ?

! Operator

- If `p` is `true`, what is `!p`? `false`
- If `p` is `false`, what is `!p`? `true`
- If `age = 24`, what is `!(age > 18)`? `false`
- If `weight = 140`, what is `!(weight == 150)`? `true`

&& Operator

- If `p1` is `false`, and `p2` is `false`, what is `p1 && p2` ?
- If `p1` is `false`, and `p2` is `true`, what is `p1 && p2` ?
- If `p1` is `true`, and `p2` is `false`, what is `p1 && p2` ?
- If `p1` is `true`, and `p2` is `true`, what is `p1 && p2` ?

If `age = 24` and `weight = 140`:

- What is `(age > 28) && (weight <= 140)` ?
- What is `(age > 18) && (weight <= 140)` ?

&& Operator

- If `p1` is `false`, and `p2` is `false`, what is `p1 && p2`? `false`
- If `p1` is `false`, and `p2` is `true`, what is `p1 && p2`? `false`
- If `p1` is `true`, and `p2` is `false`, what is `p1 && p2`? `false`
- If `p1` is `true`, and `p2` is `true`, what is `p1 && p2`? `true`

If `age = 24` and `weight = 140`:

- What is `(age > 28) && (weight <= 140)`? `false`
- What is `(age > 18) && (weight <= 140)`? `true`

|| Operator

- If `p1` is `false`, and `p2` is `false`, what is `p1 || p2`?
- If `p1` is `false`, and `p2` is `true`, what is `p1 || p2`?
- If `p1` is `true`, and `p2` is `false`, what is `p1 || p2`?
- If `p1` is `true`, and `p2` is `true`, what is `p1 || p2`?

If `age = 24` and `weight = 140`:

- What is `(age > 34) || (weight >= 150)`?
- What is `(age > 18) || (weight < 140)`?

|| Operator

- If `p1` is `false`, and `p2` is `false`, what is `p1 || p2`? `false`
- If `p1` is `false`, and `p2` is `true`, what is `p1 || p2`? `true`
- If `p1` is `true`, and `p2` is `false`, what is `p1 || p2`? `true`
- If `p1` is `true`, and `p2` is `true`, what is `p1 || p2`? `true`

If `age = 24` and `weight = 140`:

- What is `(age > 34) || (weight >= 150)`? `false`
- What is `(age > 18) || (weight < 140)`? `true`

^ Operator

- If `p1` is `false`, and `p2` is `false`, what is `p1 ^ p2` ?
- If `p1` is `false`, and `p2` is `true`, what is `p1 ^ p2` ?
- If `p1` is `true`, and `p2` is `false`, what is `p1 ^ p2` ?
- If `p1` is `true`, and `p2` is `true`, what is `p1 ^ p2` ?

If `age = 24` and `weight = 140`:

- What is `(age > 34) ^ (weight > 140)` ?
- What is `(age > 34) ^ (weight >= 140)` ?

^ Operator

- If `p1` is `false`, and `p2` is `false`, what is `p1 ^ p2`? `false`
- If `p1` is `false`, and `p2` is `true`, what is `p1 ^ p2`? `true`
- If `p1` is `true`, and `p2` is `false`, what is `p1 ^ p2`? `true`
- If `p1` is `true`, and `p2` is `true`, what is `p1 ^ p2`? `false`

If `age = 24` and `weight = 140`:

- What is `(age > 34) ^ (weight > 140)`? `false`
- What is `(age > 34) ^ (weight >= 140)`? `true`

Precedence of Logical Operators

`!` has the highest precedence

then `&&`

and `||` has the lowest precedence.

Precedence of all operators

1. `-` `!` (unary negation, logical NOT)
2. `*` `/` `%` (multiplication, division, modulus)
3. `+` `-` (addition, subtraction)
4. `<` `>` `<=` `>=` (less than, greater than, less than or equal to, greater than or equal to)
5. `==` `!=` (equal to, not equal to)
6. `&&` (logical AND)
7. `||` (logical OR)
8. `=` `+=` `-=` `*=` `/=` `%=` (assignment, combined assignment)

TestBooleanOperators.cs

```
Console.Write("Enter an integer: ");
int number = int.Parse(Console.ReadLine());

if (number % 2 == 0 && number % 3 == 0) {
    Console.WriteLine($"{number} is divisible by 2 and 3.");
}
if (number % 2 == 0 || number % 3 == 0) {
    Console.WriteLine($"{number} is divisible by 2 or 3.");
}
if (number % 2 == 0 ^ number % 3 == 0) {
    Console.WriteLine($"{number} is divisible by 2 or 3, but not both.");
}
```

LeapYear.cs

```
Console.Write("Enter a year: ");
int year = int.Parse(Console.ReadLine());

// Check if the year is a leap year
// A leap year is divisible by 4
bool isLeapYear = (year % 4 == 0);
// A leap year is divisible by 4 but not by 100
isLeapYear =  isLeapYear && (year % 100 != 0);
// A leap year is divisible by 4 but not by 100 or divisible by 400
isLeapYear =  isLeapYear || (year % 400 == 0);
// bool isLeapYear= (year % 4 == 0 && year % 100 != 0) || (year % 400 == 0);

// Display the result
Console.WriteLine($"{year} is a leap year? {isLeapYear}");
```

switch statements

A `switch` executes statements based on the value of a variable or expression.

```
switch (switch-expression) {
    case value1:
        statements(s)1;
        break;
    case value2:
        statements(s)2;
        break;
    case valueN:
        statements(s)N;
        break;
    default:
        statement(s)-for-default;
}
```

ChineseZodiac.cs

```
Console.Write("Enter a year: ");
int year = int.Parse(Console.ReadLine());

switch (year % 12)
{
    case 0: Console.WriteLine("monkey"); break;
    case 1: Console.WriteLine("rooster"); break;
    case 2: Console.WriteLine("dog"); break;
    case 3: Console.WriteLine("pig"); break;
    case 4: Console.WriteLine("rat"); break;
    case 5: Console.WriteLine("ox"); break;
    case 6: Console.WriteLine("tiger"); break;
    case 7: Console.WriteLine("rabbit"); break;
    case 8: Console.WriteLine("dragon"); break;
    case 9: Console.WriteLine("snake"); break;
    case 10: Console.WriteLine("horse"); break;
    case 11: Console.WriteLine("sheep");  break;
}
```

Conditional operators

A conditional operator evaluates an expression based on a condition.

```
boolean-expression ? expression1 : expression 2;
```

e.g.

```
max = (num1 > num2) ? num1 : num2;
```

AverageScore.cs

Let’s write a program that:

1. Prompts the user to enter three scores.
2. Calculates the average score.
3. Displays the average score.
4. Congratulates the user `if` the average is greater than 95.

AverageScore.cs

```
double score1;      // to hold score #1
double score2;      // to hold score #2
double score3;      // to hold score #3
double average;     // to hold the average score

// Prompt and read in the first test score.
Console.WriteLine("Enter score #1:");
score1 = double.Parse(Console.ReadLine());

// Prompt and read in the second score.
Console.WriteLine("Enter score #2:");
score2 = double.Parse(Console.ReadLine());

// Prompt and read in the third score.
Console.WriteLine("Enter score #3:");
score3 = double.Parse(Console.ReadLine());
```

AverageScore.cs

```
// Calculate the average score.
average = (score1 + score2 + score3) / 3.0;

// Display the average score.
Console.WriteLine($"The average is {average}");

// If the score was greater than 95, let the user know
// that's a great score.
if (average > 95)
{
    Console.WriteLine("That's a great score!");
}
```

Division.cs

Let’s write a program that:

1. Reads in two numbers.
2. If the second number is zero, returns an error, otherwise calculates and displays the quotient of the two numbers.

Division.cs

```
double number1, number2;     // Division operands
double quotient;             // Result of division

// Prompt and read in the first number
Console.WriteLine("Enter a number:");
number1 = double.Parse(Console.ReadLine());

// Prompt and read in the second number
Console.WriteLine("Enter another number:");
number2 = double.Parse(Console.ReadLine());
```

Division.cs

```
if (number2 == 0)
{
    Console.WriteLine("Division by zero is not possible.");
    Console.WriteLine("Please run the program again and ");
    Console.WriteLine("enter a number other than zero.");
}
else
{
    quotient = number1 / number2;
    Console.WriteLine($"The quotient of {number1}");
    Console.WriteLine($" divided by {number2} ");
    Console.WriteLine($" is {quotient}");
}
```

LoanQualifier.cs

To test more than one condition, an `if` statement can be nested inside another `if` statement.

Write a program that checks if a user is qualified for a loan.

- To qualify, the user must make at least $30,000/year and have been at their current job for at least 2 years.
- Let the user know whether they are qualified, and if not, why.

TestResults.cs

The `if-else-if` statement tests a series of conditions.

It's often simpler to test with `if-else-if` than with a set of nested `if-else` statements.

Write a program that gets a user's test score and returns their grade.

- 80% or higher gets an A
- 70% or higher gets a B
- 60% or higher gets a C
- 50% or higher gets a D
- Less than 50% gets an F

TestResults.cs

```
int testScore;  // Numeric test score

// Get the numeric test score
testScore = int.Parse(Console.ReadLine());

// Display the grade
if (testScore < 50) {
     Console.WriteLine("Your grade is F.");
}
else if (testScore < 60) {
     Console.WriteLine("Your grade is D.");
}
else if (testScore < 70) {
     Console.WriteLine("Your grade is C.");
}
else if (testScore < 80) {
     Console.WriteLine("Your grade is B.");
}
else {
     Console.WriteLine("Your grade is A.");
}
```

SwitchDemo.cs

The `switch` statement lets the value of a variable or expression determine where the program will branch to.

Write a program that gets a number from a user and either returns a confirmation that they entered 1, 2, or 3, or returns an error message for any other input.

SwitchDemo.cs

```
int number; // A number entered by the user
// Get one of the numbers 1, 2, or 3 from the user
Console.Write("Enter 1, 2, or 3: ");
number = int.Parse(Console.ReadLine());
// Determine the number entered
switch(number){
    case 1:
        Console.Write("You entered 1.");
        break;
    case 2:
        Console.Write("You entered 2.");
        break;
    case 3:
        Console.Write("You entered 3.");
        break;
    default:
        Console.Write("That's not 1, 2, or 3!");
    break;
}
```

PetFood.cs

Write a program that prompts the user for a grade of pet food:

```
Our pet food is available in three grades:
A, B, and C. Which do you want pricing for?
```

A is 30 cents per pound, B is 20 cents per pound, and C is 15 cents per pound.

PetFood.cs

```
// Prompt the user for a grade of pet food.
Console.WriteLine("Our pet food is available in three grades:");
Console.Write("A, B, and C. Which do you want pricing for?");
char foodGrade= Console.ReadKey().KeyChar
// Display pricing for the selected grade.
switch(foodGrade)
{
     case 'a':
     case 'A':
          Console.WriteLine("30 cents per lb.");
          break;
     case 'b':
     case 'B':
          Console.WriteLine("20 cents per lb.");
          break;
     case 'c':
     case 'C':
          Console.WriteLine("15 cents per lb.");
          break;
     default:
          Console.WriteLine("Invalid choice.");
          break;
}
```

Common Errors

1. Using `=` instead of `==` to compare primitive values
2. Forgetting to enclose an `if` statement's boolean expression in parentheses
3. Writing a semicolon at the end of an `if` clause
4. Forgetting to enclose multiple conditionally executed statements in braces
5. Omitting the trailing `else` in an `if-else-if` statement
6. Not writing complete Boolean expressions on both sides of a logical `&&` or `||` operator
7. Using a `switch` expression that is not an `int`, `char`, or `string`
8. Using a `case` expression that is not a literal or `const` variable
9. Forgetting to write a colon at the end of a `case` statement
10. Forgetting to write a `default` section in a `switch` statement
11. Reversing the `?` and the `:` when using the conditional operator

Let’s review

This week's agenda

By the end of this week, you will be able to:

- ❑ Use relational, logical, and conditional operators
- ❑ Create programs using `if` and `switch` statements

What's next?

- Loops
  - `while` loops
  - `do-while` loops
  - `for` loops
  - Nested loops
- Files
  - Input & output
- Generating random numbers