CAB201 - Week 2 Notes

Oliver Vu

August 1, 2024

1 Control Structures

1.1 If-Else Statements

The `if-else` statement is a fundamental control structure used to execute code based on a condition. It allows the program to choose between different paths of execution.

```
int number = 10;

if (number > 0) {
    Console.WriteLine("Positive number");
} else if (number == 0) {
    Console.WriteLine("Zero");
} else {
    Console.WriteLine("Negative number");
}
```

1.2 Switch Statements

The `switch` statement is used to select one of many code blocks to be executed. It is often used as an alternative to multiple `if-else` statements when checking the same variable.

```
int grade = 6;

switch (grade) {
    case 7:
        Console.WriteLine("High Distinction");
        break;
    case 6:
        Console.WriteLine("Distinction");
        break;
    case 5:
        Console.WriteLine("Credit");
        break;
    case 4:
        Console.WriteLine("Pass");
        break;
    case 3:
    case 2:
    case 1:
        Console.WriteLine("Fail");
        break;
    default:
        Console.WriteLine("Invalid grade");
        break;
}
```

1.3 For Loops

The `for` loop is used for iterating over a range of values. It is particularly useful when the number of iterations is known in advance.

```
for (int i = 1; i <= 1000; i++) {
    Console.WriteLine(i);
}

for (int i = 1000; i >= 1; i--) {
    Console.WriteLine(i);
}
```

1.4 While Loops

The `while` loop is used to execute a block of code as long as a specified condition is true. It is useful when the number of iterations is not known beforehand.

```
int i = 1;

while (i <= 1000) {
    Console.WriteLine(i);
    i++;
}

int j = 1000;

while (j >= 1) {
    Console.WriteLine(j);
    j--;
}
```

2 Additional Important Notes

2.1 Specifying Data Types for Variables

In C#, whenever you define a variable, you must specify its data type. This ensures that the variable is explicitly typed and can only store values of that specific type. **Don't forget to specify the data type!**

```
// Correctly defining a variable with a specified data type
int number = 10; // 'int' specifies that 'number' is an integer

string name = "Oliver"; // 'string' specifies that 'name' is a text string

double temperature = 36.6; // 'double' specifies that 'temperature' is a double-precision floating point
```

Failing to specify a data type when defining a variable will result in a compilation error.

2.2 Console.ReadLine() Returns a String

The `Console.ReadLine()` method in C# reads the next line of characters from the standard input stream and returns it as a `string`. This means any input from the user will need to be parsed into the appropriate data type if necessary.

2.3 Difference between Int32.Parse() and Convert.ToInt32()

Both `Int32.Parse()` and `Convert.ToInt32()` are used to convert strings to integers, but they differ in how they handle `null` values and exceptions:

- `Int32.Parse()`:
  - Only works with strings.
  - Throws an exception if the input is `null` or invalid.
  - Does not handle `null` values gracefully.
- `Convert.ToInt32()`:
  - Works with a variety of types, including strings.
  - Returns `0` if the input is `null`.
  - Handles `null` values without throwing an exception.

2.4 Difference between Float, Double, and Decimal

C# provides three data types for storing floating-point numbers: `float`, `double`, and `decimal`. They differ in precision and range:

- `float`:
  - 32-bit floating point type.
  - Approximately 7 decimal digits of precision.
  - Suitable for scientific calculations and graphics.
  - Suffix: `f` (e.g., `3.14f`).
- `double`:
  - 64-bit floating point type.
  - Approximately 15-16 decimal digits of precision.
  - Default type for floating-point literals.
  - Suitable for general calculations where high precision is required.
- `decimal`:
  - 128-bit floating point type.
  - Approximately 28-29 decimal digits of precision.
  - Suitable for financial calculations requiring high precision.
  - Suffix: `m` (e.g., `3.14m`).