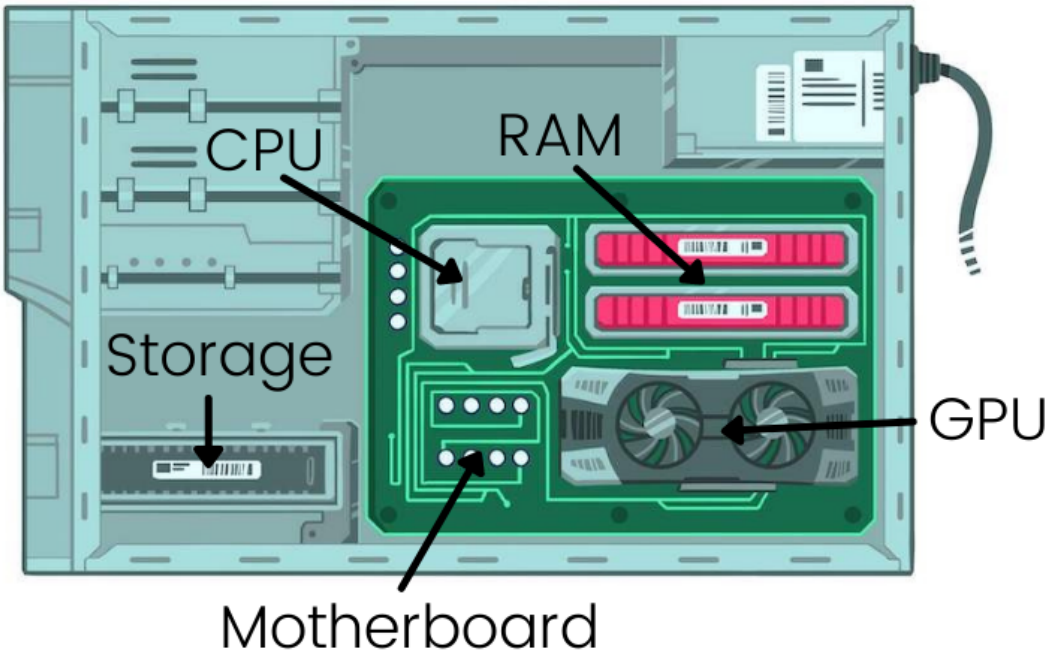
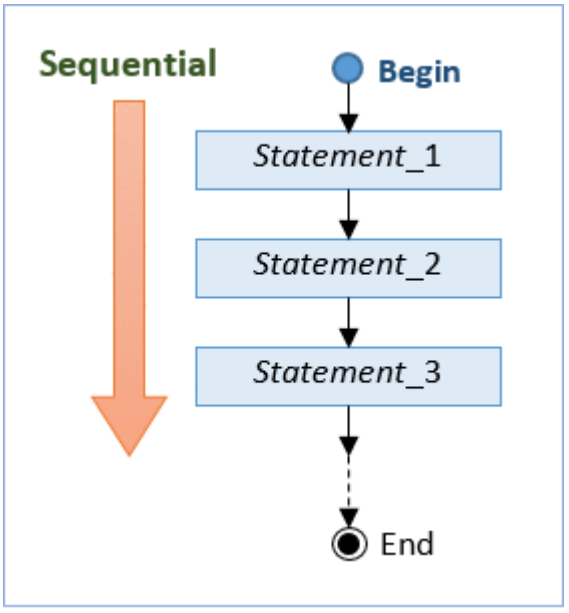
CRE132 - Week 2 - Introduction to C# Coding

1. What is a computer?



Being aware of computer hardware is important for programming. When we write our code, it is typically stored in temporary RAM until we store it more permanently in persistent disk storage. When we want to run our program or game, it will generally be loaded from the disk into RAM and executed by the CPU one line of code after another. The CPU is generally responsible for manipulating numbers (bits/bytes of text and numbers), while the GPU is a powerful graphics processing unit for manipulating images and graphics for display on a screen. These days, a GPU can also run code, such as graphics shaders for visual effects, and perform modern Artificial Intelligence operations.

2. What is a computer program?



A new programmer must learn that program code (generally) runs one line at a time and that the order in which code executes each line of code is important. Sometimes, a program will have loops, branches or conditions, but the execution order is still important.

For example, look at the following pseudocode to add two numbers together.

> 1. Line 1: Input the first number
> 2. Line 2: Input second number
> 3. Line 3: Calculate the sum of the two numbers
> 4. Line 4: Output the result

We need to assign two numbers to variables that can hold these numbers in memory. When we have these in memory, we can add them and output the results.

This is what the pseudocode would look like:

```
# Line 1: Input first number
num1 = int(input("Enter the first number: "))

# Line 2: Input second number
num2 = int(input("Enter the second number: "))

# Line 3: Calculate sum of the two numbers
sum_result = num1 + num2

# Line 4: Output the result
print("The sum is:", sum_result)
```

This is what the code would look like in C#

```
using System;

class Program
{
    static void Main()
    {
        // Line 1: Input first number
        Console.Write("Enter the first number: ");
        int num1 = int.Parse(Console.ReadLine());

        // Line 2: Input second number
        Console.Write("Enter the second number: ");
        int num2 = int.Parse(Console.ReadLine());

        // Line 3: Calculate sum of the two numbers
        int sumResult = num1 + num2;

        // Line 4: Output the result
        Console.WriteLine("The sum is: " + sumResult);
    }
}
```

- Python is a nice programming language and is extremely popular for Artificial Language and utility programming.
- C# is an object oriented language that runs faster. C# is like Java and C++ in that it uses a semicolon (`;`) to indicate the end of a command, and it uses curly braces (`{}`) to contain code together that is in the same *scope*.
- In C# (plus C++ and Java), we need to define the type of variable that we create, e.g., integer (whole numbers), string (text), float (decimal numbers), or bool (logical state ~ true/false).
- If we write code instructions in the wrong order or try to put data of the wrong type into a variable (e.g. int aNumber = "My Name"; would not compile and run), then we can get an error in our code when we try to compile it. Sometimes, an error is only found after the program starts running, which is called a runtime error.

---

3. Different Types of Programming Languages

- **1957**: FORTRAN (First high-level language aimed at scientific computing)
- **1959**: COBOL (Designed for business data processing)

- **1960s**: Assembly language (Assembler) (Enables direct hardware manipulation and precise control over system resources with high efficiency)
- **1960**: ALGOL (Influenced many modern languages with structured programming concepts)
- **1964**: BASIC (Designed to enable students in fields other than science and mathematics to use computers)

- **1970s**: C (Developed for system programming and embedded systems)
- **1972**: Smalltalk (Introduced object-oriented programming)

- **1983**: C++ (Extended C with object-oriented features)
- **1987**: Perl (Useful for text processing and system administration)

- **1991**: Python (Known for its readability and succinctness)
- **1995**: Java (Widely used for its write-once-run-anywhere capabilities)
- **1995**: JavaScript (Became the scripting language of the web)

- **2000**: C# (Developed by Microsoft as part of its .NET framework for building a wide range of applications)
- **2009**: Go (Designed at Google to improve programming productivity in the era of multicore, networked machines and large codebases)

- **2011**: Kotlin (Introduced by JetBrains, later officially supported for Android development by Google)
- **2015**: Rust (Designed by Mozilla to provide memory safety without garbage collection, known for its safe concurrency and memory efficiency)

4. Hello World

```
using System;

namespace HelloWorld
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Hello, World!");
        }
    }
}
```

5. C# Syntax and Data Types

C# (pronounced "C sharp") is a modern, object-oriented programming language developed by Microsoft as part of its .NET framework. It is designed to be simple, powerful, and versatile, suitable for various applications, including web, mobile, and desktop development.

Basic Syntax

1. **Variables and Data Types**:
    - Variables in C# must be declared with a type. The basic data types include `int`, `double`, `char`, `bool`, and `string`.

```
int age = 25;
float price = 19.95f;
char grade = 'A';
bool isCSharpFun = true;
string name = "John";
```

2. **Control Structures**:
    - Conditional statements and loops control the flow of execution.

```
if (age > 18) {
    Console.WriteLine("Adult");
} else {
    Console.WriteLine("Minor");
}

for (int i = 0; i < 5; i++) {
    Console.WriteLine(i);
}
```

3. **Methods**:
    - Methods in C# are blocks of code that perform tasks. They are declared with a return type, name, and parameters.

```
static void Main(string[] args) {
    int answer = Add(1, 2);
    Console.WriteLine( answer );
}

static int Add(int x, int y) {
    return x + y;
}
```

Enum

An `enum` (enumeration) is a special "class" that represents a group of constants (unchangeable/read-only variables). Enums are used to store related items under one type, making the code more readable and maintainable.

```
enum Day { Sunday, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday };

Day today = Day.Friday;

if (today == Day.Friday) {
    Console.WriteLine("It's the end of the workweek!");
}
```

Foreach

The `foreach` loop is used to iterate over collection elements (like arrays or lists), providing a simple and efficient way to loop through each item without manually managing the index (as in the `for` example above).

```
string[] fruits = { "Apple", "Banana", "Cherry" };

foreach (string fruit in fruits) {
    Console.WriteLine(fruit);
}
```

Switch Case

The `switch` statement performs different actions based on different conditions. It's a more efficient and cleaner method for handling multiple conditional branches (`if`-`else` chains). Each `case` is checked against the variable's value, and the matching case executes its associated code block.

```
int dayOfWeek = 3; // Let's assume 1 is Monday, 2 is Tuesday, etc.

switch (dayOfWeek) {
    case 1:
        Console.WriteLine("Monday");
        break;
    case 2:
        Console.WriteLine("Tuesday");
        break;
```

```
    case 3:
        Console.WriteLine("Wednesday");
        break;
    case 4:
        Console.WriteLine("Thursday");
        break;
    case 5:
        Console.WriteLine("Friday");
        break;
    case 6:
        Console.WriteLine("Saturday");
        break;
    case 7:
        Console.WriteLine("Sunday");
        break;
    default:
        Console.WriteLine("Invalid day");
        break;
}
```

Language Concepts

1. **Object-Oriented Programming (OOP)**:
    - C# supports OOP principles such as encapsulation, inheritance, and polymorphism.

```
class Animal {
    public void eat() {
        Console.WriteLine("Eating...");
    }
}

class Dog : Animal {
    public void bark() {
        Console.WriteLine("Barking...");
    }
}
```

2. **Namespaces**:
    - Namespaces in C# are used to organise code into groups and to prevent name collisions.

```
using System;
```

3. **Error Handling**:
    - C# uses exceptions to handle errors. The `try`, `catch`, and `finally` blocks are used for this purpose.

```
try {
    int[] myNumbers = {1, 2, 3};
    Console.WriteLine(myNumbers[10]);  // Error
} catch (Exception e) {
    Console.WriteLine("Something went wrong.");
} finally {
    Console.WriteLine("The 'try catch' is finished.");
}
```

Operators

Below is a list summarising the main operators in C#, categorised by their functionality. These operators are essential for performing various operations on data within your programs.

Arithmetic Operators

Operator	Example and Outcome
+ (Addition)	`5 + 3 = 8`
- (Subtraction)	`5 - 3 = 2`
* (Multiplication)	`5 * 3 = 15`
/ (Division)	`6 / 3 = 2`
% (Modulus)	`7 % 3 = 1` (Remainder of 7 divided by 3)

Assignment Operators

Operator	Example and Outcome
= (Assignment)	`int a = 5;` (Sets `a` to 5)
+= (Add and Assign)	`a += 3;` (Now `a` is 8)
-= (Subtract and Assign)	`a -= 2;` (Now `a` is 6)
*= (Multiply and Assign)	`a *= 2;` (Now `a` is 12)
/= (Divide and Assign)	`a /= 3;` (Now `a` is 4)
%= (Modulus and Assign)	`a %= 2;` (Now `a` is 0)

Comparison Operators

Operator	Example and Outcome
== (Equal to)	`5 == 5` (true)
!= (Not Equal to)	`5 != 4` (true)

Operator	Example and Outcome
> (Greater than)	`5 > 3` (true)
< (Less than)	`3 < 5` (true)
>= (Greater than or Equal to)	`5 >= 5` (true)
<= (Less than or Equal to)	`3 <= 5` (true)

Logical Operators

Operator	Example and Outcome
&& (Logical AND)	`true && false` (false)
|| (Logical OR)	`true || false` (true)
! (Logical NOT)	`!false` (true)

Bitwise Operators

Operator	Example and Outcome
& (Bitwise AND)	`5 & 3` (equals 1)
| (Bitwise OR)	`5 | 3` (equals 7)
^ (Bitwise XOR)	`5 ^ 3` (equals 6)
~ (Bitwise NOT)	`~5` (Typically results in a large negative number for an `int`)
<< (Left Shift)	`5 << 1` (equals 10)
>> (Right Shift)	`5 >> 1` (equals 2)

Special Operators

Operator	Example and Outcome
?? (Null Coalescing Operator)	`int? x = null; int y = x ?? -1;` (y is -1)
?. (Null-conditional Operator)	`string s = null; int? length = s?.Length;` (length is null)

Summary of Basic Keywords

Certainly! Here's a refined and more comprehensive summary of key C# keywords and concepts, useful for beginners to get a solid grasp of essential features in C# programming:

Basic Syntax Keywords

1. **using**: Declares the namespaces containing a program's external or system classes.

```
using System;
```

2. **namespace**: Organizes classes and other types into a structured scope, helping avoid naming conflicts.

```
namespace SampleApp {
    class Program { }
}
```

3. **class**: Defines a blueprint from which individual objects are created.

```
class Car {
    public void Drive() {
        Console.WriteLine("Driving");
    }
}
```

4. **static**: Specifies that the method or variable belongs to the class blueprint rather than instances of the class.

```
static void Main(string[] args) { }
```

5. **void**: Indicates that a method doesn't return a value.

```
void Display() {
    Console.WriteLine("Hello World");
}
```

6. **public**, **private**: Access modifiers that control the visibility and access of class members.
    - **public**: Accessible from any part of the program.
    - **private**: Accessible only within the class itself.

Control Flow Keywords

7. **if**, **else**: Implements conditional branching.

```
if (age >= 18) {
    Console.WriteLine("Adult");
} else {
    Console.WriteLine("Minor");
}
```

8. **switch**: Allows a variable to be tested for equality against a list of values, making complex conditional branches simpler and cleaner.

```
switch (day) {
    case "Monday":
        break;
    default:
        break;
}
```

9. **for**, **foreach**: Looping constructs that iterate over a code block multiple times.
    - **for**: Used for a known number of iterations.
    - **foreach**: Iterates over items in a collection or array.

```
foreach (var item in collection) {
    Console.WriteLine(item);
}
```

10. **while**, **do-while**: Loops that continue until a condition is no longer true.

```
while (condition) {
    // Code
}
```

Data Structure and Management Keywords

11. **enum**: Defines a variable that can hold a set of predefined constants.

```
enum Level { Low, Medium, High }
```

12. **try**, **catch**, **finally**: Exception handling blocks to catch and handle errors in a safe way.

```
try {
    // Code that may throw an exception
} catch (Exception e) {
    // Code to handle the exception
} finally {
    // Code that runs after try/catch, regardless of the result
}
```

Advanced Concepts

13. Keywords for defining immutable (means fixed) values.
    - **const**: Value must be set at declaration and cannot change.
    - **readonly**: Value can be set at declaration or within the constructor at runtime but not mutable afterwards.

---

6. Live Demo

1. Create a C# console program (see separate instruction sheet).
2. Compile and run to observe "Hello World" message in the console window.
3. Find the executable in the project bin folder and run it from a terminal.
4. Try typing in some example types, some with incorrect assignment of values.
5. Discuss console messages and add breakpoints for debugging code issues.
6. Go through the basic operator examples above.
7. Control structures and methods/functions.
8. Fix broken examples.
9. Basic introduction to a Class.
10. Basic overview of Enum and Switch.