Northeastern University London

Object Oriented Design: Introduction, Basics, and Beyond

LCSCI5202: Object Oriented Design

Week 1

Northeastern University London

Agenda

- Introductions
- Teaching plan
- Assessments
- Useful resources
- Introduction to Programming Concepts
- Why C#?
- Aims of the course
- Practical Setup & First Code
- Core Programming Constructs

Northeastern University London

Teaching Team

Instructor and Course Leader

Dr. Waleed Iqbal

Assistant Professor of Data Science

w.Iqbal@nulondon.ac.uk

Northeastern University London

Teaching plan

- 11 x 1.5 hours of full-cohort lectures (Portsoken 802)
  - Reading weeks (Week 7, October 21, 2025 and Week 12, Nov 25, 2025)
- 11 x 1.5 hours of C# Labs (Portsoken 814)
  - Reading weeks (Week 7, October 21, 2025 and Week 12, Nov 25, 2025)
- 1.5 hours of office hours per teaching week (via Zoom, Link will be uploaded in CELCAT)

Assessments

- AE1: Set Exercises worth 70% (29 October 2025, 1PM).
  - Individual submission
- AE2: Coding Assignment worth 30% (Due on 3 December 2025, 1PM).
  - Group Submission but report should be independently written.
- Submit a report on your program justifying your design choices.
- You must also submit your fully commented source code.
- The report and code should be submitted as a .zip file.
- You have three submission attempts, but only the last submission will be graded. If your last submission attempt is late, you will receive the late penalty even if you have a previous submission that was on time.
- Make sure you start project from week 1 and add up on it as we progress through the semester.
- Respect the deadlines

Labs

- Write code on your favourite IDE (VSCode is highly recommended).
- You can solve it either in the lab session or at home.

Northeastern University London

Useful resources

Slides and Lab Sheets

C# Yellow Book

C# Official Documentation

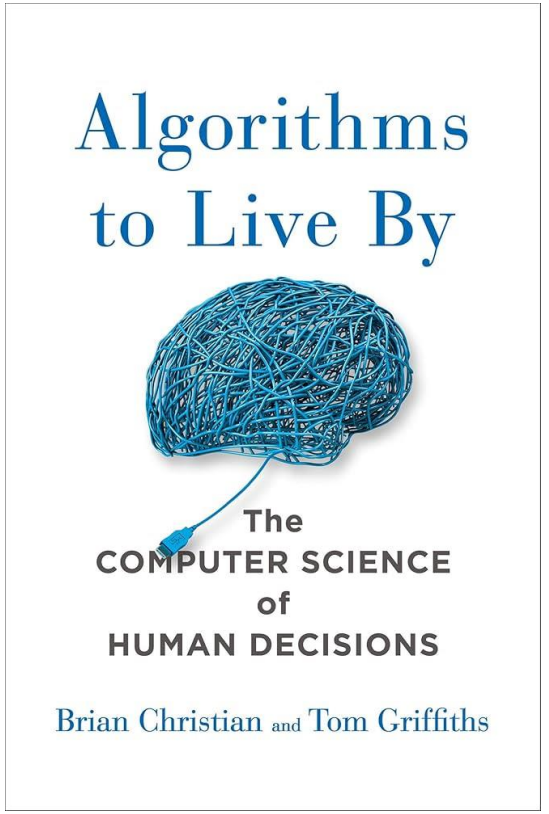
Stack Overflow

About programming

- Programming is not science
  - Programming is a skill
  - Programming takes some time to master
  - Programming is a tool used often by everyone
- It’s ok to make mistakes
  - You’re not alone
  - Every expert was once a beginner
- Programming is like a language
- Concepts Over Syntax
- Practise every concept your learn during this course

What is programming

- The process of solving problems using algorithms
- Everything running on your computer is a program
- Every program is a piece of specifically written text called the “source code”
- Programming is simply writing text which will get executed by your computer.



Process of writing a program

- Think
- Write
- Execute
- Document

Interpreted vs compiled languages

- Compiled languages
  - Translated to machine code by a compiler
  - Typically faster
  - Often tied to specific machine architectures
- Interpreted languages
  - Source code is executed on-the-fly
  - Generally slower
  - Not tied to a specific machine's architecture

Interpreted vs compiled languages

- Compiled languages
  - C, C++
  - C# (to bytecode)
  - Java (to bytecode)
- Interpreted languages
  - Python
  - JavaScript
  - PHP

What is Object Oriented Programming?

- It is a programming paradigm, which means a way of writing and organising code, centred on theconcept of object, which is a union of code + data.
- What would objects (code and data) be for the following tasks:
  - A bank, holding people's assets
  - A chess playing program

What is the point of organising programs this way? What do we achieve?

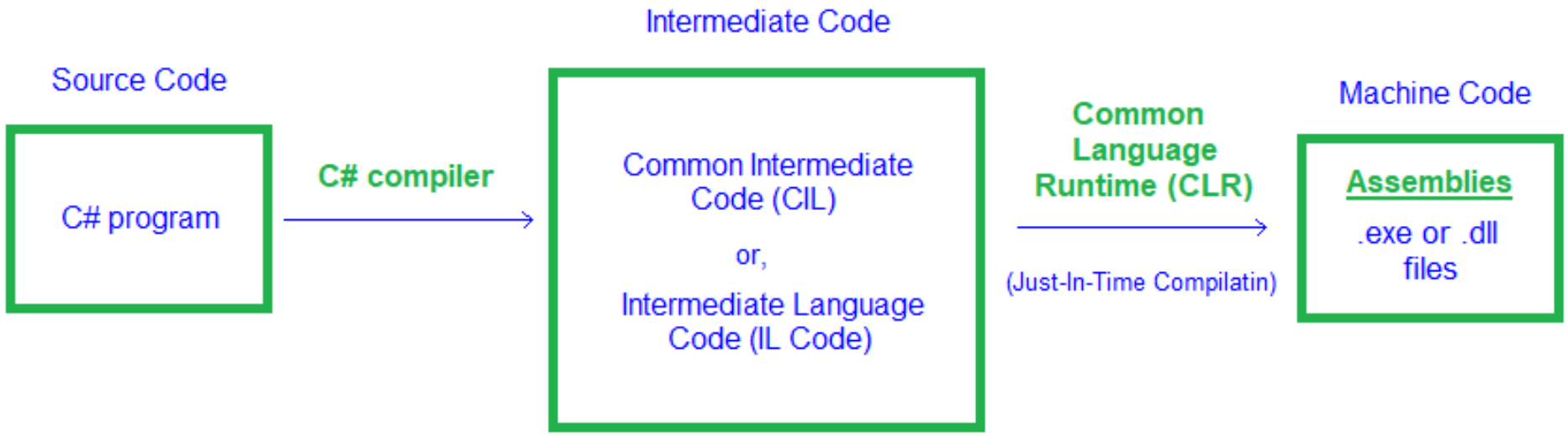
- The purpose is to tackle complexity - in a way the purpose of all programming languages is to give you tools to tackle complexity.
- Many software projects (especially modern ones) are gigantic, understanding and maintaining them is one of the central engineering challenges of our time.
- Today's programs are gigantic and maintaining them is one of the central software engineering challenge
- A well-designed object oriented program is easier to read, understand, modify, than a simply procedural one

Why learn C#

- **Universal Syntax:** C# has a syntax similar to other popular languages, making it easier to switch to or learn languages like Java, C++, or JavaScript.
- **Versatility:** Used in desktop, web, mobile, and game development (Unity); supports a wide range of applications.
- **Cross-Platform Development:** C# with .NET Core allows building applications that run on Windows, macOS, Linux, iOS and Android (Xamarian).
- **Modern Features:** Offers powerful language features like LINQ, async/await, and pattern matching for clean, efficient code.

C# teaches you OOP and how to build apps, games, and systems you'll actually use

How C# program compiles up



Aims of the course

- Implementing solutions for problems using C#
- Review typical object-oriented concepts
- An intensive tour of programming and design
- Writing object oriented code in C#

Setting up your environment

- Install .Net SDK
  - https://dotnet.microsoft.com/en-us/download
  - Verify installation by running **dotnet --version** in the command line
- Visual Studio Code (VSCode)
  - https://code.visualstudio.com/
- Install C# Dev Kit Extension

Visual Studio Code

- Search and run commands with Ctrl/Cmd + Shift + P
  - .Net: New Project
  - .Net: Close Solution
  - Workspaces: Close Workspace
- Ctrl/Cmd + F5 to compile and run your code

Visual Studio Code

- Create Your First Project
  - Open your terminal (inside VS Code or system terminal):
  - *dotnet new console -o MyApp*
  - *cd MyApp*
- Run Your Program
  - *dotnet build*
  - *dotnet run*
  - *dotnet watch run*
- Install packages
  - *dotnet add package <package_name>*

Hello World App

- When a new C# project is created in VS Code, code can be written immediately.
- No need for a Main method or additional boilerplate code.
- This is called Top-Level Statements, allowing code to be written directly without being wrapped in a class or method

Hello World with Boilerplate

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

Variables

- A variable is a named storage location in memory that holds a value.
- Declare a variable with a type, name, and optional initial value.
- Variables store data that can be used and modified throughout the program.

```
int age = 25;
string name = "Alice";
bool isStudent = true;
```

Variables

- Variables are declared using the format, initial value is optional:

  [DataType] [VariableName] = [InitialValue];

- Choose a Data Type: Specifies the kind of data the variable will store (e.g., int, string, bool).
- Name the Variable: Provide a meaningful name following C# naming conventions.
- (Optional) Initialize: Assign an initial value using the = operator.
- Variables must be assigned before use.

Variable Naming Conventions in C#

- Camel Case:
  - Example: int myScore; or string userName;
- Meaningful Names
  - Choose descriptive names that clearly indicate the purpose of the variable.
  - Example: int age; instead of int a;
- Prefix for Boolean Variable
  - Prefix Boolean variables with is, has, or can to indicate a true/false value.
  - Example: bool isActive; or bool hasPassed;

Basic Data Types

- Integer (int)
  - Stores whole numbers.
  - Example: int count = 10;
- Floating-Point (float, double):
  - Stores numbers with decimals.
  - Example: float price = 19.99f; double distance = 123.456;
- String (string):
  - Stores a sequence of characters.
  - Example: string name = "Alice";
- Boolean (bool):
  - Stores true or false.
  - Example: bool isComplete = true;

Converting variables

- Implicit Conversion:
  - Automatically occurs when converting a smaller type to a larger type.

```
int num = 10;
double result = num;
```

- Explicit Conversion (Casting):
  - Required when converting a larger type to a smaller type or incompatible types.

```
double price = 9.99;
int roundedPrice = (int)price;
```

Converting variables - Conversion Methods

Using Conversion Methods

- Parse(): Converts a string to a numeric type.
- Convert.ToType(): Converts between various types.

```
string strA = "123";
int number = int.Parse(strA);
string strB = "123.45";
double num = Convert.ToDouble(strB);
```

Converting variables - Exceptions

Handling Conversion Errors

- Use TryParse() to safely convert strings, avoiding exceptions.

```
string input = "abc";
bool success = int.TryParse(input, out int result);
```

Reading Input

Using Console.ReadLine():

- Console.ReadLine() is used to read input from the user via the command line.
- The input is always read as a string.

```
Console.WriteLine("Enter your name:");
string userName = Console.ReadLine();
```

Converting Input

Converting Input:

- Input can be converted to other data types using methods like int.Parse().

```
Console.WriteLine("Enter your favorite number:");
string input = Console.ReadLine();
int favoriteNumber = int.Parse(input);
Console.WriteLine("Your favorite number is: " + favoriteNumber);
```

Brackets and Parentheses

- **( ) Parentheses:** Enclose method parameters and control conditions (e.g., if, for, while).
- **{ } Curly Brackets:** Define the body of classes, methods, and control structures, grouping code blocks.
- **[ ] Square Brackets:** Indicate arrays and access elements by index. Also used for attributes.
- **< > Angle Brackets:** Define generic types and methods, allowing flexible and type-safe code (e.g., List<T>).

Symbols in C#

- Used for mathematical calculations.
  - + (Addition)
  - - (Subtraction)
  - * (Multiplication)
  - / (Division)
  - % (Modulus → remainder after division)
- Used to compare values (results in true or false).
  - == (Equal to)
  - != (Not equal to)
  - > (Greater than)
  - < (Less than)
  - >= (Greater than or equal to)
  - <= (Less than or equal to)

Symbols in C#

- Used for combining conditions.
  - && (AND → both conditions true)
  - || (OR → at least one condition true)
  - ! (NOT → reverses condition)
- Assign values to variables.
  - = (Assign)
  - += (Add and assign)
  - -= (Subtract and assign)
  - *= (Multiply and assign)
  - /= (Divide and assign)
- Increment & Decrement
  - ++ (Increase by 1)
  - -- (Decrease by 1)

Conditionals

```
if (condition) {
        Console.WriteLine("Condition met.");
}

if (condition) {
        Console.WriteLine("Condition met 1.");
}
else if (condition) {
        Console.WriteLine("conditions met 2.");
}
else {
        Console.WriteLine("Two conditions met 3.");
}
```

Loops

- **For Loop:**

```
for (int i = 1; i <= 5; i++)
{
   Console.WriteLine("Iteration: " + i);
}
```

- **While Loop:**

```
int count = 1;
while (count <= 5)
{
   Console.WriteLine("Count: " + count);
   count++;
}
```

Loops

- **Do-While Loop:**

```
int number = 1;
do
{
   Console.WriteLine("Number: " + number);
   number++;
} while (number <= 5);
```

- **Foreach Loop:**

```
string[] names = { "Alice", "Bob", "Charlie" };
foreach (string name in names)
{
   Console.WriteLine("Hello, " + name);
}
```

Functions

- A function is a block of code that performs a specific task.
- It is reusable, meaning you can call it multiple times without rewriting the same code.
- Functions can take inputs (called parameters) and return outputs (called return values).

Function

```
// No return value, no parameters
static void DisplayText() {
   Console.WriteLine("Some text");
}
// Return value
static int ReturnNumber() {
   return 5;
}
// Return value + parameters
static int AddNumbers(int a, int b) {
   return a + b;
}
```

Call in Main Function

```
static void Main()
{
        DisplayText();
}
```

Summary

- **Course Logistics**
- **Introduction to programming concepts** (what programming is, compiled vs interpreted languages, writing programs)
- **Why C# and its benefits**
- **Development setup** (installing .NET SDK, VS Code, extensions)
- **Programming Fundamentals**