```
namespace HelloWorld;

internal static class Program
{
    private static void Main()
    {
        Console.WriteLine("Hello, world!");
    }
}
```

Introduction to C#

C# Fundamentals – .NET Development

Agenda

- C# Background
  - What Is C#?
  - Compilation
  - History
  - Design Principles
- C# Design
  - Language Type
  - Why Static Types?
- C# Project Structure

C# Background

What Is C#?

C# is a just-in-time (JIT) compiled, statically-typed, general-purpose programming language.

What Is C#?

C# is a just-in-time (JIT) compiled, statically-typed, general-purpose programming language.

What Is C#?

C# is a just-in-time (JIT) compiled, statically-typed, general-purpose programming language.

> Language that takes the code you write and translates (compiles) it into different code

What Is C#?

C# is a just-in-time (JIT) compiled, statically-typed, general-purpose programming language.

> Language that takes the code you write and translates (compiles) it into different code

- Written code is human-readable and information-dense

What Is C#?

C# is a just-in-time (JIT) compiled, statically-typed, general-purpose programming language.

> Language that takes the code you write and translates (compiles) it into different code

- Written code is human-readable and information-dense
- Compiled code is super efficient due to compiler optimizations

What Is C#?

C# is a just-in-time (JIT) compiled, statically-typed, general-purpose programming language.

> Language that takes the code you write and translates (compiles) it into different code

- Written code is human-readable and information-dense
- Compiled code is super efficient due to compiler optimizations
- Compiled code is typically not human-readable

What Is C#?

C# is a just-in-time (JIT) compiled, statically-typed, general-purpose programming language.

What Is C#?

C# is a just-in-time (JIT) compiled, statically-typed, general-purpose programming language.

- Written code is compiled into bytecode and run on a virtual machine

What Is C#?

C# is a just-in-time (JIT) compiled, statically-typed, general-purpose programming language.

- Written code is compiled into bytecode and run on a virtual machine
- Bytecode code is compiled into machine code as needed

What Is C#?

C# is a just-in-time (JIT) compiled, statically-typed, general-purpose programming language.

- Written code is compiled into bytecode and run on a virtual machine
- Bytecode code is compiled into machine code as needed
- Slower than code that is compiled directly to machine code

What Is C#?

C# is a just-in-time (JIT) compiled, statically-typed, general-purpose programming language.

- Written code is compiled into bytecode and run on a virtual machine
- Bytecode code is compiled into machine code as needed
- Slower than code that is compiled directly to machine code
- System independent, and code can be analyzed and changed at runtime

What Is C#?

C# is a just-in-time (JIT) compiled, statically-typed, general-purpose programming language.

What Is C#?

C# is a just-in-time (JIT) compiled, statically-typed, general-purpose programming language.

> Variables have fixed data types which must be known at compile-time

What Is C#?

C# is a just-in-time (JIT) compiled, statically-typed, general-purpose programming language.

What Is C#?

C# is a just-in-time (JIT) compiled, statically-typed, general-purpose programming language.

> Create many types of applications

What Is C#?

C# is a just-in-time (JIT) compiled, statically-typed, general-purpose programming language.

> Create many types of applications

- Object-oriented programming (OOP) (mainly)

What Is C#?

C# is a just-in-time (JIT) compiled, statically-typed, general-purpose programming language.

> Create many types of applications

- Object-oriented programming (OOP) (mainly)
- Imperative programming

What Is C#?

C# is a just-in-time (JIT) compiled, statically-typed, general-purpose programming language.

Create many types of applications

- Object-oriented programming (OOP) (mainly)
- Imperative programming
- Some functional-programming features

What Is C#?

C# is a just-in-time (JIT) compiled, statically-typed, general-purpose programming language.

> Create many types of applications

- Object-oriented programming (OOP) (mainly)
- Imperative programming
- Some functional-programming features
- Some query-language features

C# Terminology

C# Terminology

- *.NET* is a set of compilers and tools supported by underlying runtime libraries. C#, F#, and Visual Basic (VB) are the languages supported by .NET

C# Terminology

- *.NET* is a set of compilers and tools supported by underlying runtime libraries. C#, F#, and Visual Basic (VB) are the languages supported by .NET
- *.NET Core* is the current implementation of .NET

C# Terminology

- *.NET* is a set of compilers and tools supported by underlying runtime libraries. C#, F#, and Visual Basic (VB) are the languages supported by .NET
- *.NET Core* is the current implementation of .NET
- .NET's bytecode is called *common-intermediate-language (CIL) code*

C# Terminology

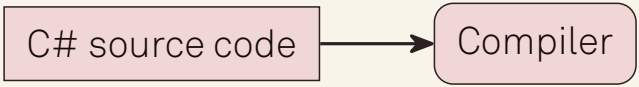
- *.NET* is a set of compilers and tools supported by underlying runtime libraries. C#, F#, and Visual Basic (VB) are the languages supported by .NET
- *.NET Core* is the current implementation of .NET
- .NET's bytecode is called *common-intermediate-language (CIL) code*
- The virtual machine used to execute .NET applications is known as the *common language runtime (CLR)*

C# Compilation

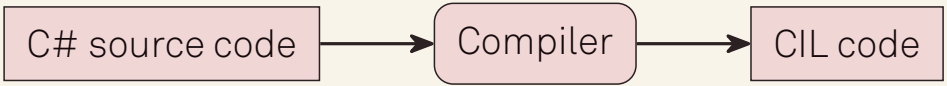
C# Compilation

C# source code

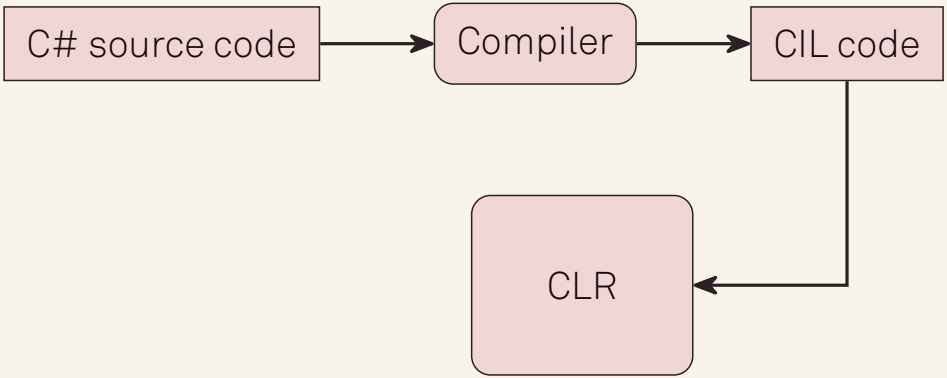
C# Compilation



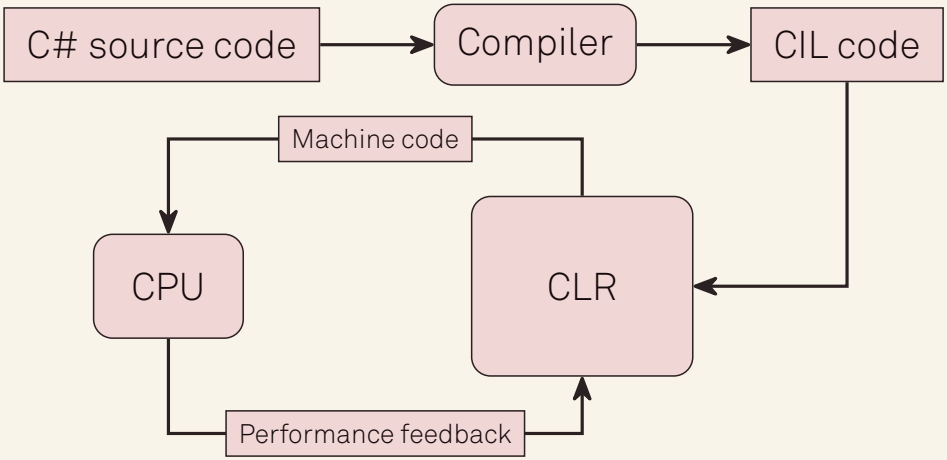
C# Compilation



C# Compilation



C# Compilation



A Brief History

A Brief History

- Developed from January 1999 to July 2000 by Anders Hejlsberg, Scott Wiltamuth, and Peter Golde at Microsoft

A Brief History

- Developed from January 1999 to July 2000 by Anders Hejlsberg, Scott Wiltamuth, and Peter Golde at Microsoft
- Heavily influenced by Java and previous Microsoft Java imitations (J++, J#)

A Brief History

- Developed from January 1999 to July 2000 by Anders Hejlsberg, Scott Wiltamuth, and Peter Golde at Microsoft
- Heavily influenced by Java and previous Microsoft Java imitations (J++, J#)
- Current versions of C# are significantly different from Java

A Brief History

- Developed from January 1999 to July 2000 by Anders Hejlsberg, Scott Wiltamuth, and Peter Golde at Microsoft
- Heavily influenced by Java and previous Microsoft Java imitations (J++, J#)
- Current versions of C# are significantly different from Java
- C# is ECMA-standardized (ECMA-334)

Design Principles

Design Principles

- The language is intended to be a simple, modern, general-purpose programming language

Design Principles

- The language is intended to be a simple, modern, general-purpose programming language
- The language, and implementations thereof, should provide support for software engineering principles such as strong type checking, array bounds checking, detection of attempts to use uninitialized variables, and automatic garbage collection. Software robustness, durability, and programmer productivity are important.

Design Principles

- The language is intended for use in developing software components suitable for deployment in distributed environments.

Design Principles

- The language is intended for use in developing software components suitable for deployment in distributed environments.
- Portability is very important for source code and programmers, especially those already familiar with C and C++.

Design Principles

- The language is intended for use in developing software components suitable for deployment in distributed environments.
- Portability is very important for source code and programmers, especially those already familiar with C and C++.
- Support for internationalization is very important.

Design Principles

- C# is intended to be suitable for writing applications for both hosted and embedded systems, ranging from the very large that use sophisticated operating systems, down to the very small having dedicated functions.

Design Principles

- C# is intended to be suitable for writing applications for both hosted and embedded systems, ranging from the very large that use sophisticated operating systems, down to the very small having dedicated functions.
- Although C# applications are intended to be economical with regard to memory and processing power requirements, the language was not intended to compete directly on performance and size with C or assembly language.

C# Design

What Type of Language Is C#

What Type of Language Is C#

- C# is an application programming language

What Type of Language Is C#

- C# is an application programming language
  - Console apps

What Type of Language Is C#

- C# is an application programming language
  - Console apps
  - Games (Unity)

What Type of Language Is C#

- C# is an application programming language
  - Console apps
  - Games (Unity)
  - Web apps (Blazer)

What Type of Language Is C#

- C# is an application programming language
  - Console apps
  - Games (Unity)
  - Web apps (Blazer)
  - Desktop and mobile apps (.NET MAUI)

What Type of Language Is C#

- C# is an application programming language
  - Console apps
  - Games (Unity)
  - Web apps (Blazer)
  - Desktop and mobile apps (.NET MAUI)
- It's a *high-level* language compared to C and a *very high-level* language compared to Assembly language

Language-level Breakdown

Language-level Breakdown

- Relative to Assembly language

Language-level Breakdown

- Relative to Assembly language
    - Low level (machine code, Assembly)

Language-level Breakdown

- Relative to Assembly language
    - Low level (machine code, Assembly)
        - “Close to the hardware”

Language-level Breakdown

- Relative to Assembly language
    - Low level (machine code, Assembly)
        - “Close to the hardware”
        - Specific control over hardware (stack)

Language-level Breakdown

- Relative to Assembly language
    - Low level (machine code, Assembly)
        - “Close to the hardware”
        - Specific control over hardware (stack)
        - Hard to read

Language-level Breakdown

- Relative to Assembly language
  - Low level (machine code, Assembly)
    - “Close to the hardware”
    - Specific control over hardware (stack)
    - Hard to read
  - High level (BASIC, C, C++, COBOL, FORTRAN)

Language-level Breakdown

- Relative to Assembly language
  - Low level (machine code, Assembly)
    - “Close to the hardware”
    - Specific control over hardware (stack)
    - Hard to read
  - High level (BASIC, C, C++, COBOL, FORTRAN)
    - Add abstractions

Language-level Breakdown

- Relative to Assembly language
  - Low level (machine code, Assembly)
    - “Close to the hardware”
    - Specific control over hardware (stack)
    - Hard to read
  - High level (BASIC, C, C++, COBOL, FORTRAN)
    - Add abstractions
    - Automate some control over hardware (stack)

Language-level Breakdown

- Relative to Assembly language
  - Low level (machine code, Assembly)
    - “Close to the hardware”
    - Specific control over hardware (stack)
    - Hard to read
  - High level (BASIC, C, C++, COBOL, FORTRAN)
    - Add abstractions
    - Automate some control over hardware (stack)
    - Use natural-language elements

Language-level Breakdown

- Relative to Assembly language
  - Low level (machine code, Assembly)
    - “Close to the hardware”
    - Specific control over hardware (stack)
    - Hard to read
  - High level (BASIC, C, C++, COBOL, FORTRAN)
    - Add abstractions
    - Automate some control over hardware (stack)
    - Use natural-language elements
  - Very high level (Java, JavaScript, Python, Ruby, C#)

Language-level Breakdown

- Relative to Assembly language
  - Low level (machine code, Assembly)
    - “Close to the hardware”
    - Specific control over hardware (stack)
    - Hard to read
  - High level (BASIC, C, C++, COBOL, FORTRAN)
    - Add abstractions
    - Automate some control over hardware (stack)
    - Use natural-language elements
  - Very high level (Java, JavaScript, Python, Ruby, C#)
    - Add many abstractions

Language-level Breakdown

- Relative to Assembly language
  - Low level (machine code, Assembly)
    - “Close to the hardware”
    - Specific control over hardware (stack)
    - Hard to read
  - High level (BASIC, C, C++, COBOL, FORTRAN)
    - Add abstractions
    - Automate some control over hardware (stack)
    - Use natural-language elements
  - Very high level (Java, JavaScript, Python, Ruby, C#)
    - Add many abstractions
    - Remove details about hardware control

Language-level Breakdown

Language-level Breakdown

- Relative to C

Language-level Breakdown

- Relative to C
    - Low level (C)

Language-level Breakdown

- Relative to C
    - Low level (C)
        - A lot of hardware control

Language-level Breakdown

- Relative to C
    - Low level (C)
        - A lot of hardware control
        - Not many abstractions

Language-level Breakdown

- Relative to C
  - Low level (C)
    - A lot of hardware control
    - Not many abstractions
  - Middle level (C++)

Language-level Breakdown

- Relative to C
  - Low level (C)
    - A lot of hardware control
    - Not many abstractions
  - Middle level (C++)
    - A blend of abstractions and hardware control

Language-level Breakdown

- Relative to C
  - Low level (C)
    - A lot of hardware control
    - Not many abstractions
  - Middle level (C++)
    - A blend of abstractions and hardware control
    - “Smart” memory management

Language-level Breakdown

- Relative to C
  - Low level (C)
    - A lot of hardware control
    - Not many abstractions
  - Middle level (C++)
    - A blend of abstractions and hardware control
    - “Smart” memory management
  - High level (Java, Python, C#)

Language-level Breakdown

- Relative to C
  - Low level (C)
    - A lot of hardware control
    - Not many abstractions
  - Middle level (C++)
    - A blend of abstractions and hardware control
    - “Smart” memory management
  - High level (Java, Python, C#)
    - Many abstractions

Language-level Breakdown

- Relative to C
  - Low level (C)
    - A lot of hardware control
    - Not many abstractions
  - Middle level (C++)
    - A blend of abstractions and hardware control
    - “Smart” memory management
  - High level (Java, Python, C#)
    - Many abstractions
    - Automated memory management (garbage collector)

Why Static Types?

Why Static Types?

- Data types are important to programmers

Why Static Types?

- Data types are important to programmers
- Type errors are caught at *compile-time*, not runtime

Why Static Types?

- Data types are important to programmers
- Type errors are caught at *compile-time*, not runtime
- Meaning of code is clearer

Why Static Types?

- Data types are important to programmers
- Type errors are caught at *compile-time*, not runtime
- Meaning of code is clearer
- What do Python and JavaScript do?

Why Static Types?

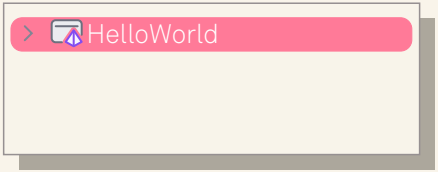
- Data types are important to programmers
- Type errors are caught at *compile-time*, not runtime
- Meaning of code is clearer
- What do Python and JavaScript do?
    - mypy

Why Static Types?

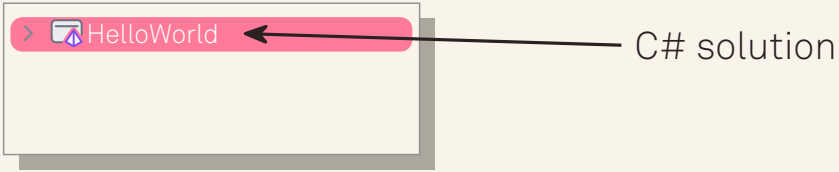
- Data types are important to programmers
- Type errors are caught at *compile-time*, not runtime
- Meaning of code is clearer
- What do Python and JavaScript do?
    - mypy
    - TypeScript/JSDoc comments

C# Project Structure

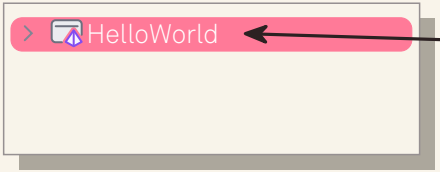
Project Structure



Project Structure

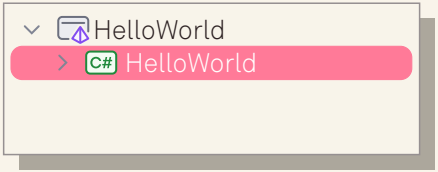


Project Structure

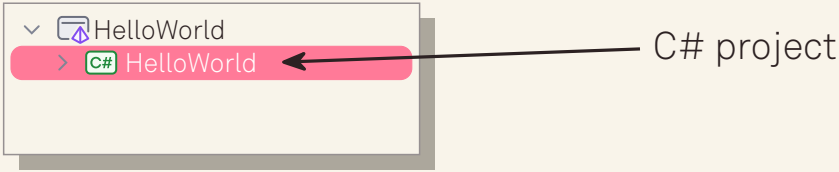


- Solutions are containers for one or more related projects

Project Structure



Project Structure

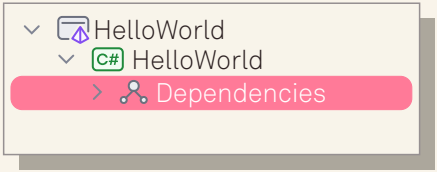


Project Structure

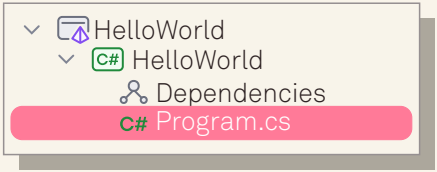
C# project

- Projects contain all code compiled into an executable, library, or website

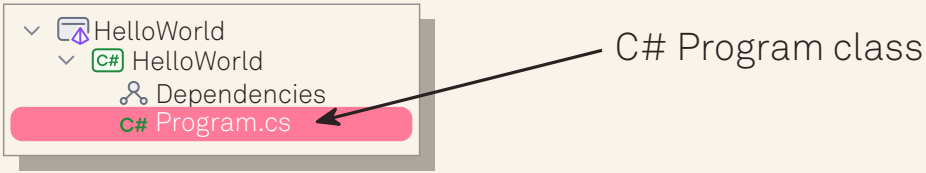
Project Structure



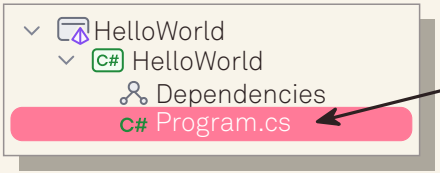
Project Structure



Project Structure



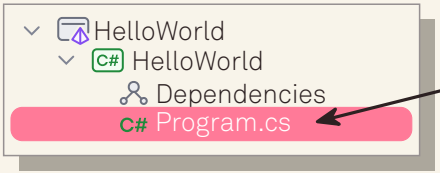
Project Structure



C# Program class

- This is the main class for a basic console application

Project Structure



C# Program class

- This is the main class for a basic console application
- This is where the code from the title slide goes

Any Questions?