

# BÀI 1. TỔNG QUAN VỀ MICROSOFT .NET VÀ NGÔN NGỮ LẬP TRÌNH C#.

## 1. Giới thiệu về Microsoft .NET

### 1.1. .NET và lịch sử phát triển

.NET là một nền tảng phát triển mã nguồn mở, chạy trên nhiều hệ điều hành khác nhau, miễn phí để xây dựng các loại ứng dụng, được phát triển bởi hãng phần mềm khổng lồ Microsoft. Với .NET, ta có thể sử dụng các ngôn ngữ, trình soạn thảo và thư viện khác nhau để xây dựng các ứng dụng chạy trên web, thiết bị di động, máy tính để bàn, hay phát triển các trò chơi, các ứng dụng Internet vạn vật (IoT – Internet of Things).

Microsoft bắt đầu làm việc trên .NET vào đầu thế kỷ 21 và phát hành phiên bản .NET đầu tiên vào năm 2002. Sau đó, họ phát hành phiên bản 1.1 vào năm 2003, phiên bản 2.0 vào năm 2005, v.v. Phiên bản cuối cùng là 4.8 được phát hành vào năm 2019.

Ban đầu, nền tảng này được gọi là Microsoft .NET Framework, nhưng sau này, họ thường gọi nó là .NET Framework. Khởi đầu .NET Framework được xây dựng chỉ dành cho Windows, hỗ trợ nhiều ngôn ngữ lập trình khác nhau, trong đó có C# và VB.NET. Nó được cung cấp như một phần của hệ điều hành Windows và các bản cập nhật được thực hiện thông qua Windows Update hoặc thông qua trình cài đặt độc lập tùy chọn. .NET Framework bao gồm một bộ thư viện lớp (.NET Framework Class Library) cung cấp thư viện các lớp để lập trình và một máy ảo gọi là Common Language Runtime (CLR), là môi trường nơi mã chương trình được thực thi. CLR ngoài việc được Microsoft phát hành còn được công bố dưới dạng đặc tả mở và được các tổ chức ISO và Ecma tiêu chuẩn hóa. Với tiêu chuẩn này, một phiên bản .NET Framework đã được phát triển để chạy trên Linux và nó được gọi là Mono. Mono đã phát triển để hỗ trợ nhiều loại hệ điều hành và ứng dụng. Có thể hiểu, Mono là một biến thể đa nền tảng của .NET Framework và nó được sử dụng để xây dựng các ứng dụng dành cho máy tính để bàn và thiết bị di động cho bất kỳ hệ điều hành nào.

Xamarin là một công ty được thành lập vào năm 2011 với trọng tâm là xây dựng một bộ công cụ cho phép phát triển đa nền tảng với ngôn ngữ C#. Họ duy trì dự án Mono, bao gồm MonoTouch cho phép các nhà phát triển xây dựng ứng dụng iOS bằng C# và Mono cho Android để xây dựng ứng dụng Android bằng C#. Dự án Mono đã phát triển mạnh kể từ khi Xamarin tiếp quản và họ đã phát hành Xamarin.iOS, Xamarin.Android, Xamarin.Mac và các công cụ khác cho phép phát triển các ứng dụng đa nền tảng bằng C#. Xamarin.iOS và Xamarin.Android là các phiên bản Mono cho iPhone và điện thoại thông minh dựa trên Android.

Vào tháng 2 năm 2016, Microsoft đã mua lại Xamarin Inc. và đã dần dần tích hợp nền tảng và công cụ Xamarin với việc cung cấp công cụ Visual Studio và .NET. Vì vậy,

ngày nay, Xamarin là một nền tảng mở rộng của nền tảng phát triển .NET với các công cụ và thư viện để xây dựng ứng dụng cho Android, iOS, tvOS, watchOS, macOS và Windows. Ứng dụng Xamarin sử dụng Mono như một phiên bản .NET để chạy và người dùng có thể tạo các dự án Xamarin từ phần mềm Visual Studio hoặc Visual Studio cho Mac.

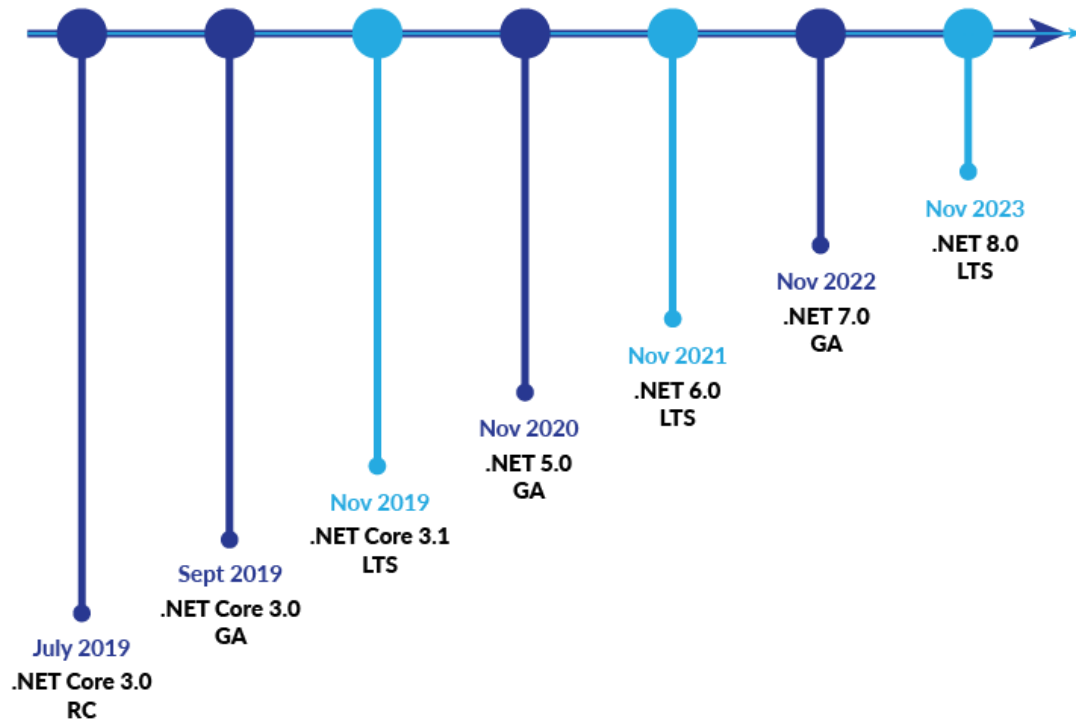
Trước khi mua lại Xamarin, vào năm 2014 Microsoft cho phát triển .NET Core, một phiên bản .NET được thiết kế lại dựa trên phiên bản đơn giản hóa của các thư viện lớp. .NET Core được thiết kế cho phép hỗ trợ đa nền tảng cho .NET, bao gồm Linux và macOS đồng thời khắc phục các nhược điểm của .NET Framework và được giải thích là sẽ trở thành nền tảng của tất cả các nền tảng .NET. .NET Framework là một framework nguyên khối chỉ chạy trên Windows và phải được cài đặt cho mọi máy cần chạy các ứng dụng .NET trên Windows. Bản chất nguyên khối của nó cũng gây ra chu kỳ phát hành chậm. Trong thế giới ngày nay, nơi mà các kiến trúc máy chủ và microservices được sử dụng nhiều, chúng ta cần một framework nhẹ với ít bộ nhớ để chạy các ứng dụng của mình và .NET Framework không hoạt động tốt trong không gian này. Tuy nhiên, Microsoft muốn đổi mới và đáp ứng yêu cầu này từ phía người dùng. Do đó, .NET Core được xây dựng từ đầu để đặt nền tảng cho tương lai của .NET, bao gồm một kiến trúc dạng mô-đun mang tính linh hoạt, cho phép Microsoft đổi mới trên các dịch vụ của họ.

.NET Core là một framework nhẹ, có kiến trúc mô-đun và đa nền tảng của .NET, cho phép tạo ra các ứng dụng web, microservices, thư viện và ứng dụng console hiện đại chạy trên Windows, Linux và macOS. .NET Core bao gồm các gói Nuget để người dùng chỉ phải cài đặt các mô-đun hoặc gói cần thiết để chạy ứng dụng, có thể đóng gói chúng cùng với ứng dụng của mình và người dùng cuối không cần cài đặt framework trên máy của họ. Kiến trúc mô-đun cũng cho phép đưa ra chu kỳ phát hành nhanh hơn từ nhóm .NET. Phiên bản .NET Core 1.0 được phát hành vào năm 2016, phiên bản cuối là .NET Core 3.1 được phát hành vào năm 2019.

Đến năm 2020, tầm nhìn của Microsoft là One.NET. Điều này có nghĩa là họ muốn tránh nhầm lẫn và chuyển sang mã nguồn mở hoàn toàn với sự hỗ trợ cho tất cả các nền tảng trên tất cả các bộ tính năng, nói cách khác, họ hợp nhất .NET Framework và .NET Core. Do đó, bản phát hành lớn tiếp theo sau .NET Framework 4.8 và .NET Core 3.1 là bản hợp nhất mang tên .NET 5. Từ quan điểm của nhà phát triển, .NET là một trình chạy đa nền tảng với một thư viện lớp không lồ chứa tất cả các yếu tố cần thiết để viết mã các ứng dụng web, máy tính để bàn hoặc di động, có thể dành riêng cho một nền tảng cụ thể, cũng có thể chạy trên đám mây với nhiều công nghệ khác nhau.

Các chương trình trong .NET có thể được phát triển với nhiều ngôn ngữ lập trình khác nhau như C++, Visual Basic và Fortran (F#), trong đó C# là ngôn ngữ phổ biến nhất.

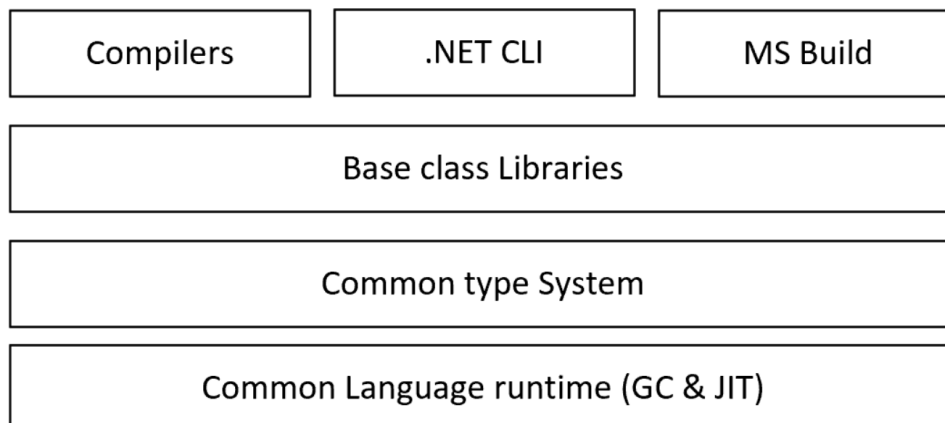
C# là một ngôn ngữ lập trình hướng đối tượng, giàu tính năng, ban đầu dựa trên cú pháp của C/C++ và Java.



Hình 1.1. Lộ trình phát hành .NET

## 1.2. Các thành phần của .NET

.NET có hai thành phần chính: .NET runtime và các thư viện lớp cơ sở. Runtime bao gồm trình thu gom rác (GC) và trình biên dịch just-in-time (JIT), quản lý việc thực thi các ứng dụng .NET và thư viện lớp cơ sở (Base Class Libraries - BCL), còn được gọi là *runtime libraries* hoặc *framework libraries*, chứa các khối xây dựng cho các ứng dụng .NET.



Hình 1.2. Minh họa .NET SDK

.NET SDK có sẵn để tải xuống tại <https://dotnet.microsoft.com/download/dotnet/>. Nó chứa một bộ thư viện và công cụ để phát triển và chạy các ứng dụng .NET. Người dùng có thể chọn cài đặt SDK hoặc .NET runtime. Để phát triển các ứng dụng .NET, ta nên cài đặt SDK trên máy phát triển, còn để chạy các ứng dụng .NET thì chỉ cần .NET runtime. .NET runtime được bao gồm trong .NET SDK, do đó người dùng không phải cài đặt riêng .NET runtime nếu đã cài đặt .NET SDK.

.NET SDK chứa các thành phần sau:

- **Common Language Runtime (CLR):** CLR thực thi mã và quản lý cấp phát bộ nhớ. Các ứng dụng .NET, khi được biên dịch, tạo ra một ngôn ngữ trung gian (IL – Intermediate Language). CLR sử dụng trình biên dịch JIT để chuyển đổi mã đã biên dịch sang mã máy. Nó là một runtime đa nền tảng có sẵn cho Windows, Linux và macOS.
- **Quản lý bộ nhớ:** trình thu gom rác quản lý việc cấp phát và giải phóng bộ nhớ cho các ứng dụng .NET. Đối với mọi đối tượng mới được tạo, bộ nhớ được cấp phát trong vùng heap và khi không còn đủ dung lượng trống, trình thu gom rác sẽ kiểm tra các đối tượng trong heap và xóa chúng nếu chúng không còn được sử dụng trong ứng dụng. Để biết thêm thông tin, có thể tham khảo tại: <https://docs.microsoft.com/en-us/dotnet/standard/garbage-collection>.
- **JIT:** Khi mã .NET được biên dịch, nó được chuyển đổi thành IL. IL không phụ thuộc vào nền tảng và ngôn ngữ, vì vậy khi runtime chạy ứng dụng, JIT sẽ chuyển IL thành mã máy mà bộ xử lý hiểu được.
- **Hệ thống kiểu chung (CTS – Common Type System):** Hệ thống này xác định cách các kiểu được định nghĩa, sử dụng và quản lý trong CLR. Nó cho phép tích hợp nhiều ngôn ngữ và đảm bảo an toàn cho kiểu.
- **Thư viện lớp cơ sở:** Thư viện này chứa các triển khai cho các kiểu nguyên thủy như System.String và System.Boolean, các collections như List<T>, Dictionary<TKey, TValue> và các hàm tiện ích để thực hiện các hoạt động I/O, HTTP, serialization, và nhiều thứ khác nữa. Nó đơn giản hóa việc phát triển ứng dụng .NET.
- **Trình biên dịch Roslyn:** Roslyn là trình biên dịch C # và Visual Basic mã nguồn mở với các API phân tích mã phong phú. Nó cho phép xây dựng các công cụ phân tích mã với cùng một API được Visual Studio sử dụng.
- **MSBuild:** Đây là một công cụ để build các ứng dụng .NET. Visual Studio sử dụng MSBuild để build các ứng dụng .NET.
- **NuGet:** Đây là một công cụ quản lý gói mã nguồn mở, cho phép tạo, xuất bản và sử dụng lại mã. Một gói NuGet chứa mã đã biên dịch, các tệp phụ thuộc của nó và một tệp kê khai bao gồm thông tin số phiên bản gói.

### 1.3. Các đặc trưng của .NET:

Dưới đây là một số đặc trưng cốt lõi của .NET:

- **Mã nguồn mở:** .NET là nền tảng nhà phát triển mã nguồn mở miễn phí (không có chi phí cấp phép, kể cả cho mục đích sử dụng thương mại), cung cấp nhiều công cụ phát triển cho Linux, macOS và Windows. Mã nguồn của nó được duy trì bởi Microsoft và cộng đồng .NET trên GitHub. Người dùng có thể truy cập kho lưu trữ .NET tại địa chỉ:  
<https://github.com/dotnet/core/blob/main/Documentation/core-repos.md>
- **Đa nền tảng:** Các ứng dụng .NET chạy trên nhiều hệ điều hành, bao gồm Linux, macOS, Android, iOS, tvOS, watchOS và Windows. Chúng cũng chạy nhất quán trên các kiến trúc vi xử lý như x86, x64, ARM32 và ARM64.

Với .NET, chúng ta có thể xây dựng các loại ứng dụng sau:

Loại ứng dụng	Mô tả
Web	Ứng dụng API dựa trên web và REST
Microservices	Các dịch vụ vi mô có thể triển khai độc lập, có khả năng mở rộng cao, chạy trên Docker container
Cloud	Các chức năng không cần máy chủ và các ứng dụng cloud-native
Mobile	Ứng dụng chạy trên thiết bị di động iOS, Android, Windows
Desktop	Ứng dụng Winform và WPF trên hệ điều hành Windows
IoT	Ứng dụng IoT có thể chạy trên Raspberry Pi và HummingBoard
Game	Các ứng dụng game 2D và 3D cho Windows, Android, iOS
Machine Learning	Xây dựng các mô hình học máy tùy chỉnh và dễ dàng tích hợp với các ứng dụng .NET

- **Ngôn ngữ lập trình:** .NET hỗ trợ nhiều ngôn ngữ lập trình. Mã được viết bằng một ngôn ngữ có thể truy cập vào các ngôn ngữ khác. Bảng sau đây hiển thị các ngôn ngữ được hỗ trợ:

Ngôn ngữ hỗ trợ	Mô tả
C#	Ngôn ngữ lập trình đơn giản, hiện đại, hướng đối tượng, an toàn kiểu
F#	Ngôn ngữ lập trình hướng chức năng và hướng đối tượng
Visual Basic	Ngôn ngữ lập trình đơn giản, an toàn kiểu, hướng đối tượng

- **IDE:** .NET hỗ trợ nhiều IDE, cụ thể được chỉ ra dưới đây:

- Visual Studio: là một IDE giàu tính năng có sẵn trên nền tảng Windows để xây dựng, gỡ lỗi và xuất bản các ứng dụng .NET. Microsoft cung cấp ba phiên bản: Community, Professional và Enterprise. Visual Studio 2019 Community Edition miễn phí cho sinh viên, nhà phát triển cá nhân và tổ chức đóng góp cho các dự án nguồn mở.
- Visual Studio cho Mac: miễn phí và có sẵn cho macOS. Nó có thể được sử dụng để phát triển các ứng dụng và trò chơi đa nền tảng cho iOS, Android và web sử dụng .NET.
- Visual Studio Code: là một trình soạn thảo miễn phí, mã nguồn mở, nhẹ nhưng mạnh mẽ, chạy được trên Windows, macOS và Linux. Nó hỗ trợ tích hợp sẵn JavaScript, TypeScript và Node.js. Đồng thời, với các phần mở rộng, người dùng có thể bổ sung hỗ trợ cho nhiều ngôn ngữ lập trình phổ biến.
- Codespaces: là một môi trường phát triển đám mây được cung cấp bởi Visual Studio Code và được lưu trữ bởi GitHub để phát triển các ứng dụng .NET.
- **Mô hình triển khai:** .NET hỗ trợ hai chế độ triển khai:
  - Chế độ độc lập: Khi ứng dụng .NET được xuất bản ở chế độ độc lập, cấu phần được xuất bản chứa .NET runtime, thư viện và ứng dụng cũng như các thành phần phụ thuộc của nó. Các ứng dụng độc lập chạy trên tảng cụ thể và không cần cài đặt .NET runtime lên máy đích. Máy sử dụng .NET runtime được đính kèm cùng với ứng dụng để chạy ứng dụng.
  - Phụ thuộc vào framework: Khi ứng dụng .NET được xuất bản ở chế độ phụ thuộc vào framework, cấu phần được xuất bản chỉ chứa ứng dụng và các phụ thuộc của nó. .NET runtime phải được cài đặt trên máy đích để chạy ứng dụng.

#### 1.4. Các framework của .NET:

.NET đơn giản hóa việc phát triển ứng dụng bằng cách cung cấp nhiều framework kèm theo. Mỗi framework chứa một tập hợp các thư viện để phát triển các ứng dụng phù hợp. Dưới đây là một số framework của .NET:

- **ASP.NET Core:** Đây là một framework đa nền tảng và mã nguồn mở cho phép xây dựng các ứng dụng hiện đại, dựa trên đám mây, được kết nối internet, chẳng hạn như các ứng dụng web, IoT và API. ASP.NET Core được xây dựng dựa trên .NET Core, do đó có thể xây dựng và chạy trên các nền tảng như Linux, macOS và Windows.
- **WPF:** Đây là một framework giao diện người dùng cho phép tạo các ứng dụng máy tính để bàn cho Windows. WPF sử dụng Ngôn ngữ XAML (Extensible Application Markup Language), một dạng mô hình khai báo, để phát triển ứng dụng.
- **Entity Framework (EF) Core:** Đây là một framework ánh xạ quan hệ đối tượng (ORM - object-relational mapping) mã nguồn mở, đa nền tảng, nhẹ, để làm việc với cơ sở dữ liệu sử dụng các đối tượng .NET. Nó hỗ trợ các truy vấn



LINQ, theo dõi thay đổi và chuyển đổi các lược đồ. Nó hoạt động với các cơ sở dữ liệu phổ biến như SQL Server, SQL Azure, SQLite, Azure Cosmos DB, MySQL và nhiều cơ sở dữ liệu khác.

- **Language-Integrated Query (LINQ):** Đây là một sự bổ sung khả năng truy vấn cho các ngôn ngữ lập trình .NET. LINQ cho phép truy vấn dữ liệu từ cơ sở dữ liệu, XML, mảng trong bộ nhớ và collections với cùng một API.

## 2. Thiết lập môi trường lập trình trên Visual Studio

Visual Studio .NET là môi trường tích hợp phát triển phần mềm (Integrated Development Environment – IDE) của Microsoft, là công cụ cho phép viết mã, gỡ rối và biên dịch chương trình trong nhiều ngôn ngữ lập trình .NET khác nhau.

### 2.1. Hướng dẫn cài đặt Visual Studio

Bước 1 - Đảm bảo máy tính đã sẵn sàng cho Visual Studio

Trước khi bạn bắt đầu cài đặt Visual Studio:

1. Kiểm tra các yêu cầu hệ thống . Những yêu cầu này giúp bạn biết liệu máy tính của mình có hỗ trợ Visual Studio 2019 hay không.
2. Áp dụng các bản cập nhật Windows mới nhất. Các bản cập nhật này đảm bảo rằng máy tính của bạn có cả bản cập nhật bảo mật mới nhất và các thành phần hệ thống cần thiết cho Visual Studio.
3. Khởi động lại. Việc khởi động lại đảm bảo rằng mọi bản cập nhật hoặc cài đặt đang chờ xử lý không cản trở quá trình cài đặt Visual Studio.
4. Giải phóng dung lượng. Xóa các tệp và ứng dụng không cần thiết khỏi %SystemDrive% của bạn, chẳng hạn bằng cách chạy ứng dụng Disk Cleanup.

Bước 2 - Tải xuống Visual Studio

Tiếp theo, tải xuống tệp khởi động Visual Studio.

Để làm như vậy, hãy mở link <https://visualstudio.microsoft.com/downloads/>, chọn phiên bản Visual Studio mà bạn muốn tải xuống và lưu về máy.

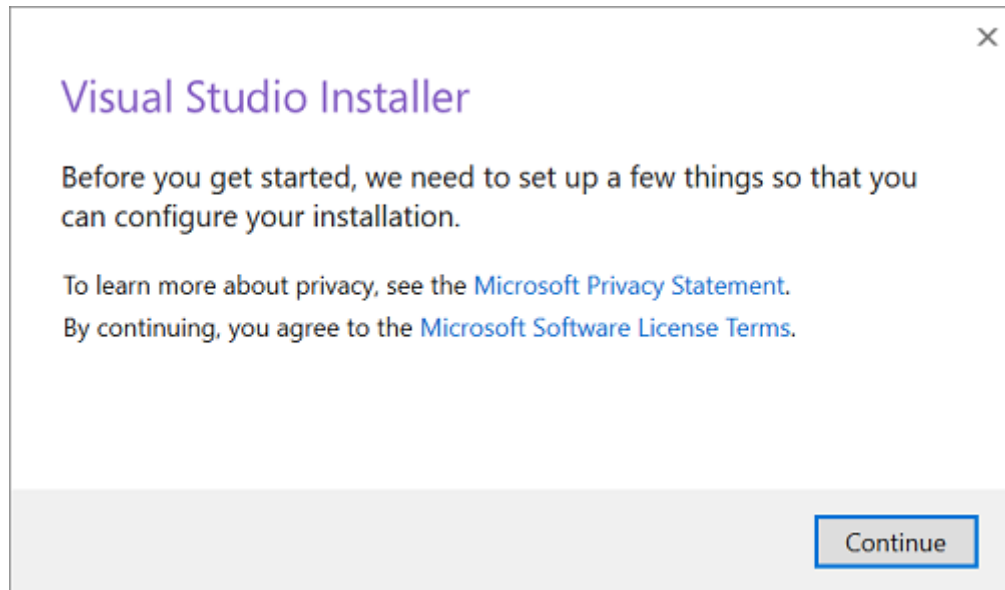
Bước 3 - Cài đặt Visual Studio Installer

Chạy tệp bootstrapper để cài đặt Visual Studio Installer. Trình cài đặt nhẹ này bao gồm mọi thứ bạn cần để cài đặt và tùy chỉnh Visual Studio.

1. Từ thư mục vừa tải Visual Studio về, bấm đúp vào bộ khởi động phù hợp hoặc tương tự với một trong các tệp sau:
  - vs\_community.exe cho bản Visual Studio Community
  - vs\_professional.exe cho bản Visual Studio Professional
  - vs\_enterprise.exe cho bản Visual Studio Enterprise

Nếu bạn nhận được thông báo User Account Control, hãy chọn Yes.

2. Khi nhận được yêu cầu xác nhận Microsoft Privacy Statement và Microsoft Software License Term, chọn Continue.

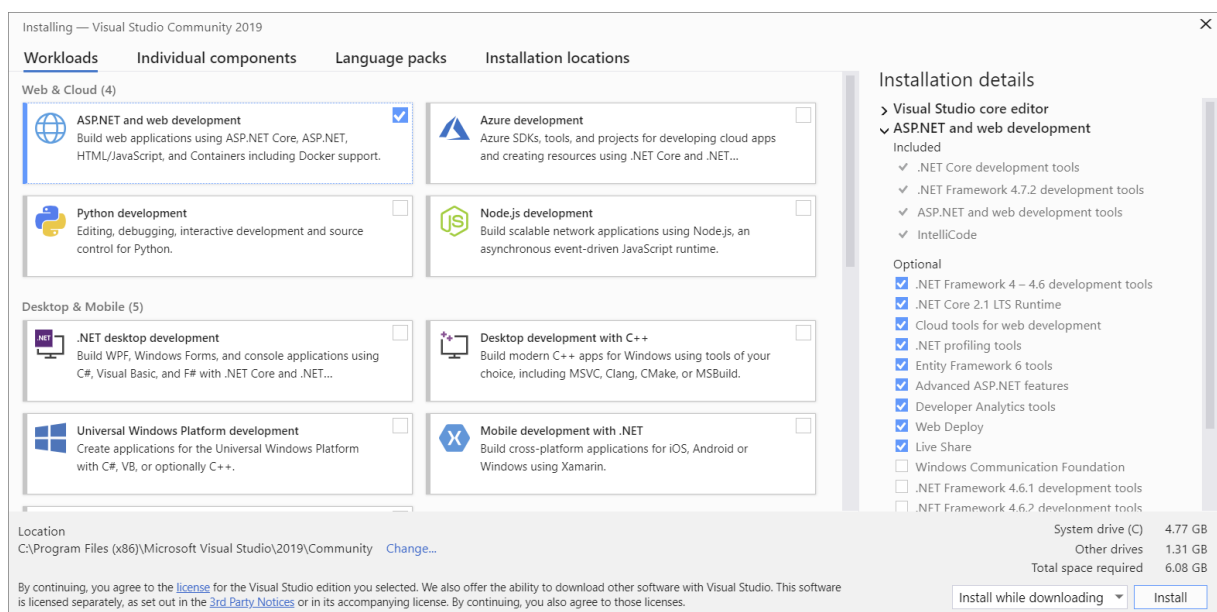


Hình 1.3 - Visual Studio Installer

#### Bước 4 - Chọn các gói cần cài đặt

Sau khi trình cài đặt được cài xong, bạn có thể sử dụng nó để tùy chỉnh cài đặt của mình bằng cách chọn bộ tính năng hoặc gói phần mềm mà bạn muốn ở tab Workloads. Dưới đây là cách thức thực hiện:

1. Tìm workload bạn muốn trong Visual Studio Installer.



Hình 1.4 – Tùy chỉnh Workload trong Visual Studio Installer



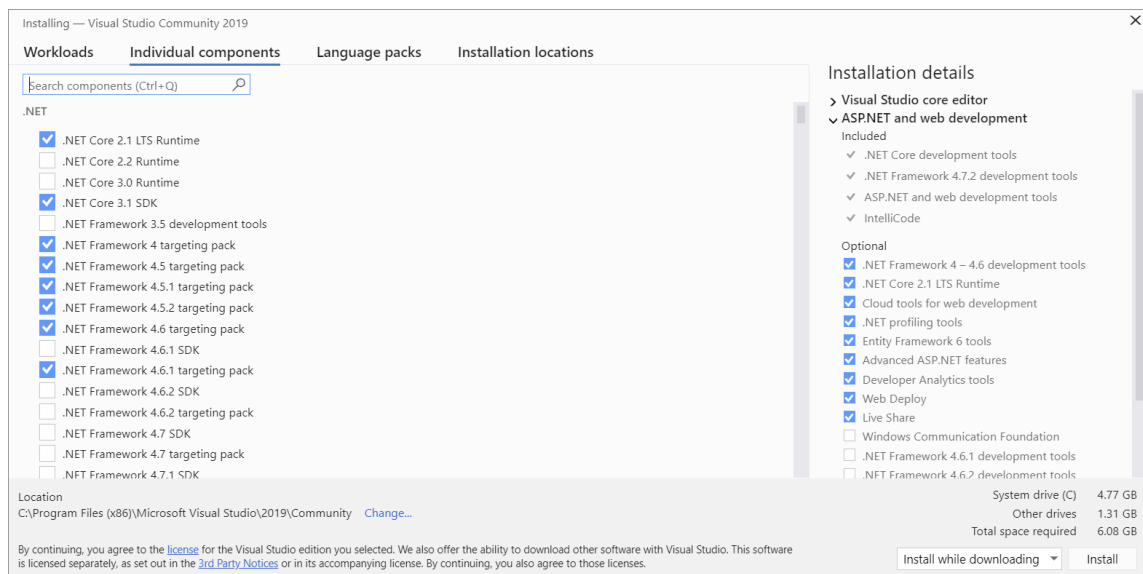
Ví dụ: chọn gói "ASP.NET and web development". Nó đi kèm với core editor mặc định, bao gồm hỗ trợ viết mã cơ bản cho hơn 20 ngôn ngữ, khả năng mở và chỉnh sửa mã từ bất kỳ thư mục nào.

2. Sau khi bạn chọn các gói phần mềm bạn muốn, hãy chọn Install.

Tiếp theo, màn hình trạng thái xuất hiện cho biết tiến trình cài đặt Visual Studio của bạn.

Bước 5 - Chọn các thành phần riêng lẻ (Tùy chọn)

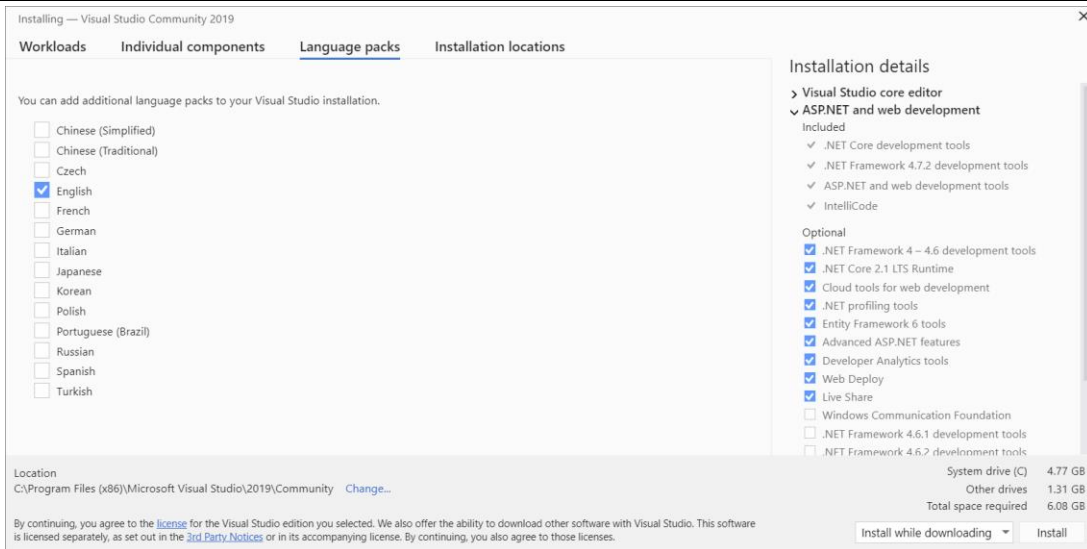
Nếu bạn không muốn sử dụng tính năng Workloads để tùy chỉnh cài đặt Visual Studio của mình hoặc bạn muốn thêm nhiều thành phần hơn để cài đặt, bạn có thể thêm bớt các thành phần riêng lẻ từ tab Individual components. Chọn những gì bạn muốn, sau đó làm theo lời nhắc.



Hình 1.5 – Tùy chỉnh Individual component trong Visual Studio Installer

Bước 6 - Cài đặt gói ngôn ngữ (Tùy chọn)

Theo mặc định, chương trình cài đặt cố gắng khớp với ngôn ngữ của hệ điều hành khi nó chạy lần đầu tiên. Để cài đặt Visual Studio bằng ngôn ngữ khác, hãy chọn tab Language Packs từ Visual Studio Installer, sau đó làm theo lời nhắc.



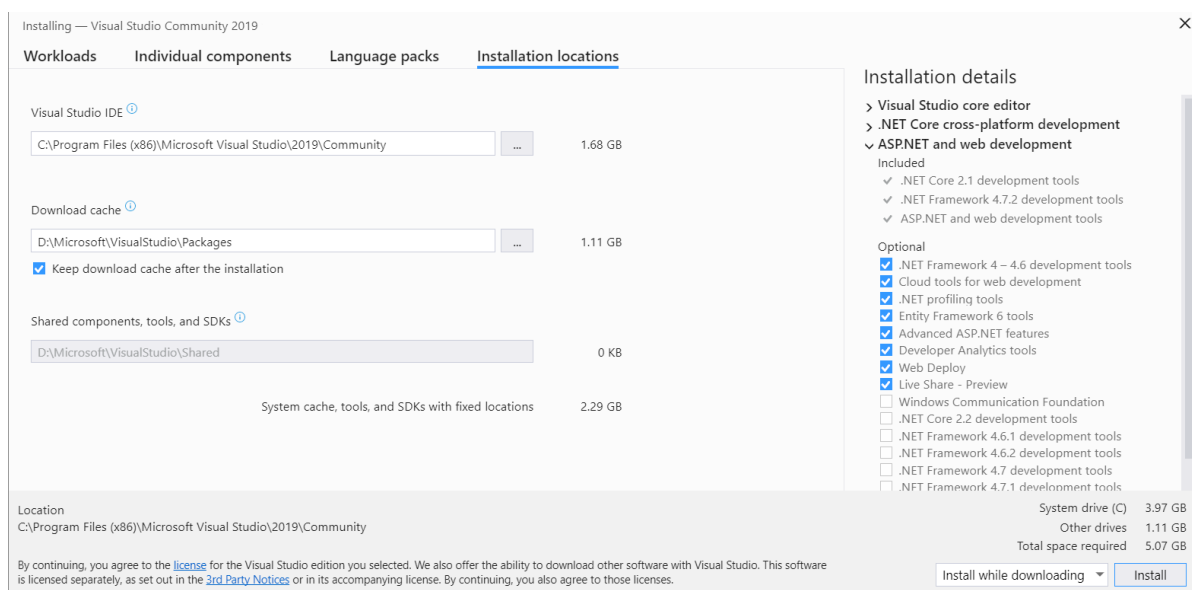
Hình 1.6 – Tùy chỉnh Language Packs trong Visual Studio Installer

### Thay đổi ngôn ngữ trình cài đặt từ dòng lệnh:

Một cách khác mà bạn có thể thay đổi ngôn ngữ mặc định là chạy trình cài đặt từ dòng lệnh. Ví dụ, bạn có thể buộc trình cài đặt để chạy trong tiếng Anh bằng cách sử dụng lệnh sau: `vs_installer.exe --locale en-US`. Trình cài đặt sẽ ghi nhớ cài đặt này khi nó được chạy lần sau. Trình cài đặt hỗ trợ các mã ngôn ngữ sau: zh-cn, zh-tw, cs-cz, en-us, es-es, fr-fr, de-de, it-it, ja-jp, ko-kr, pl-pl, pt-br, ru-ru và tr-tr.

### Bước 7 - Chọn vị trí cài đặt (Tùy chọn)

Bạn có thể giảm dung lượng cài đặt Visual Studio trên ổ đĩa hệ thống của mình. Bạn có thể chọn di chuyển bộ nhớ cache tải xuống, các thành phần được chia sẻ, SDK và các công cụ sang các ổ đĩa khác nhau và giữ Visual Studio trên ổ đĩa mà nó chạy nhanh nhất.



Hình 1.7 – Tùy chỉnh Instalation Locations trong Visual Studio Installer

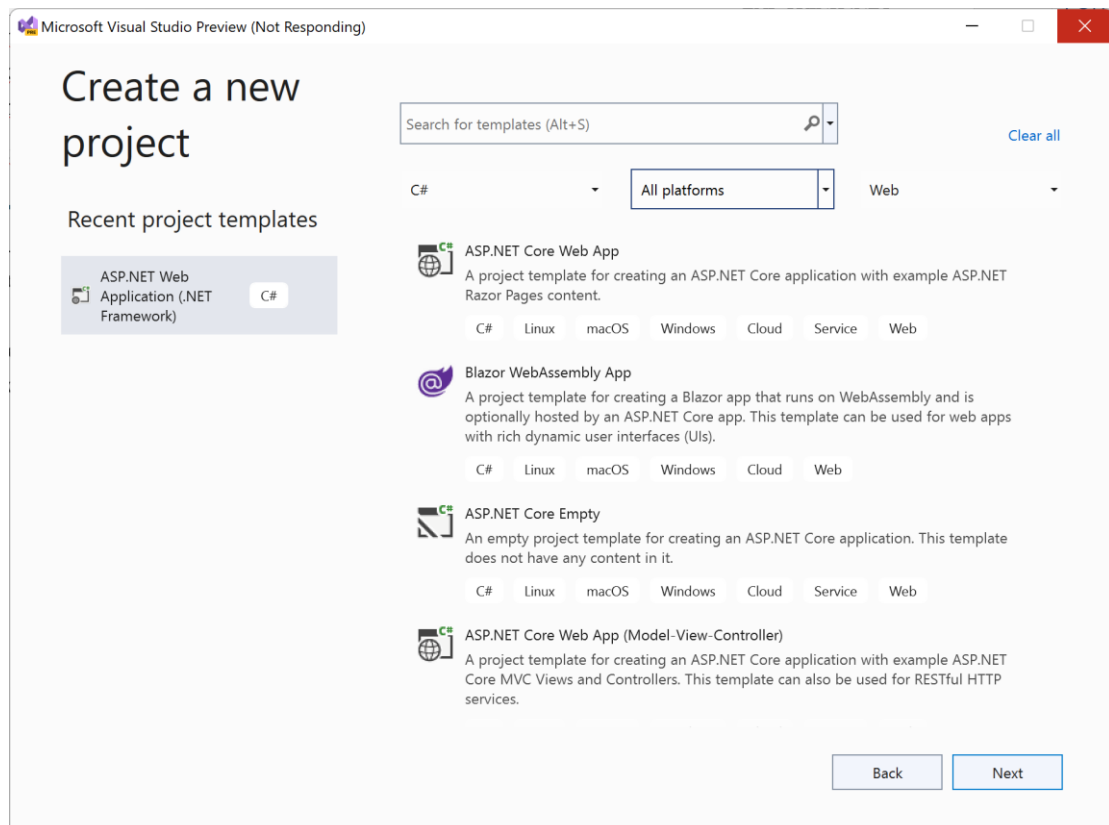
### Lưu ý:

Bạn chỉ có thể chọn một ổ đĩa khác khi lần đầu tiên cài đặt Visual Studio. Nếu bạn đã cài đặt nó và muốn thay đổi ổ đĩa, bạn phải gỡ cài đặt Visual Studio rồi cài đặt lại.

### Bước 8 - Bắt đầu lập trình

Sau khi cài đặt Visual Studio hoàn tất, hãy chọn nút Launch để bắt đầu làm việc với Visual Studio.

Trên cửa sổ bắt đầu, chọn Create a new project.

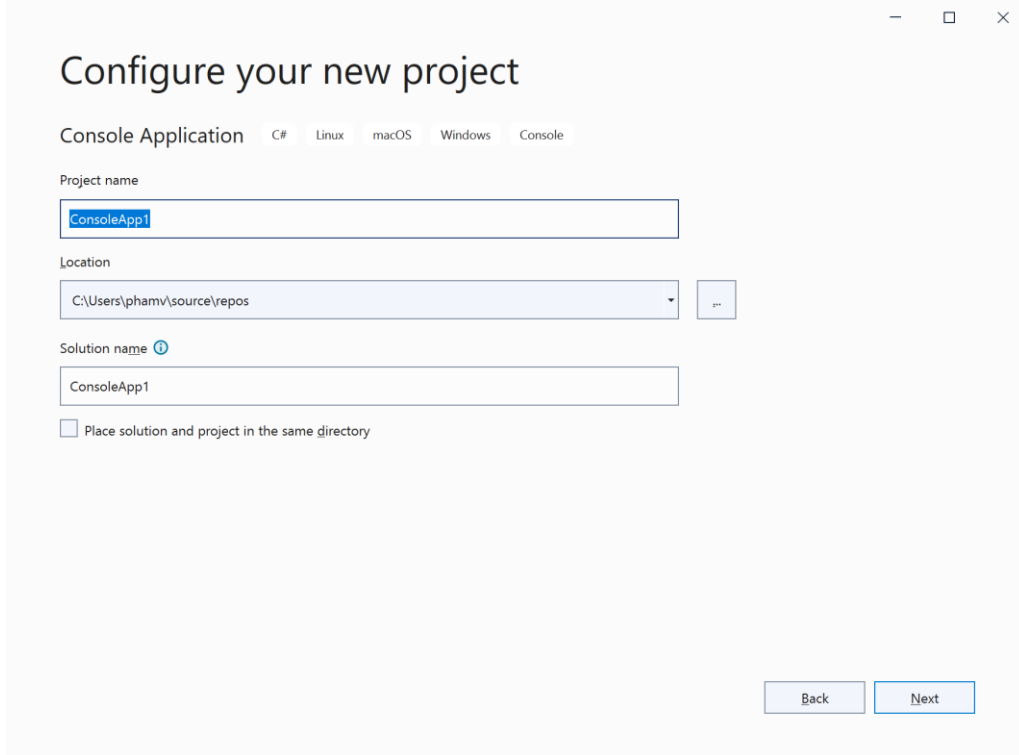


Hình 1.8 – Tạo mới một project

Trong hộp tìm kiếm, hãy nhập loại ứng dụng bạn muốn tạo để xem danh sách các template có sẵn. Danh sách các template phụ thuộc vào các workload mà bạn đã chọn trong khi cài đặt. Để xem các template khác nhau, hãy chọn các workload khác nhau.

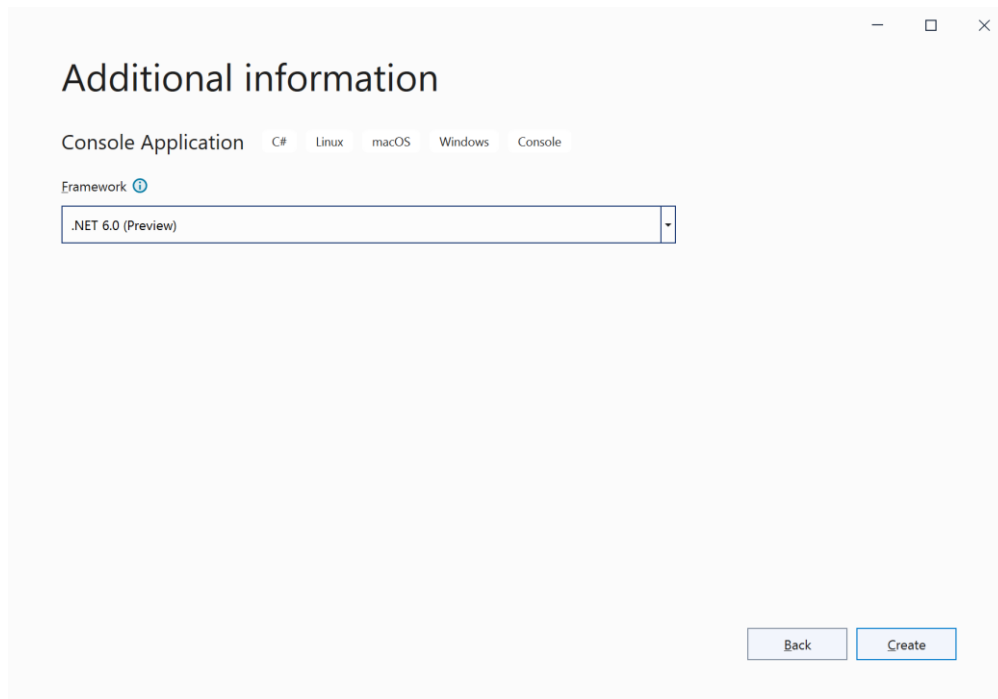
Bạn có thể lọc tìm kiếm của mình cho một ngôn ngữ lập trình cụ thể bằng cách sử dụng drop-down list có tên Language. Bạn cũng có thể lọc theo Platform và Project Types.

Sau khi nhấn Next, bạn điền thông tin về tên Project và nơi lưu trữ.



Hình 1.9 – Cấu hình cho project mới

Nhấn Next, sau đó chọn framework bạn muốn sử dụng. Cuối cùng nhấn Create.



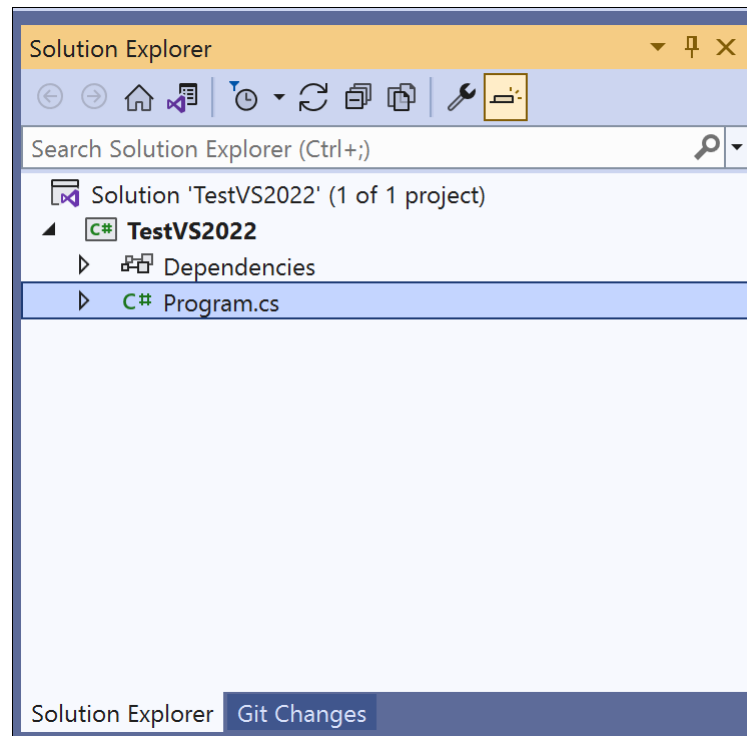
Hình 1.10 – Lựa chọn framework cho project

Visual Studio tạo dự án mới cho bạn và bạn có thể sẵn sàng để viết mã.

## 2.2. Cách tổ chức chương trình của Visual Studio

Chương trình hoặc ứng dụng hoặc thậm chí là một hệ thống đều được Visual Studio tổ chức dưới dạng Solution. Solution là tập hợp của nhiều Project. Project là tập hợp các

tập tin liên quan đến một ứng dụng và được người dùng tổ chức theo các cấp độ thư mục. Một Project của Visual Studio thông thường bao gồm 3 phần: phần thuộc tính (Properties), phần tham chiếu (References), phần người dùng tự định nghĩa.



Hình 1.11 – Cấu trúc của một project trên Visual Studio

Phần thuộc tính (Properties) chứa class AssemblyInfo trong đó mô tả các thông tin cơ bản về ứng dụng như: tên ứng dụng, tên công ty, địa chỉ công ty, bản quyền và các thông tin khác. Phần tham chiếu (References) chứa các gói hoặc các class mà ứng dụng này cần dùng. Người dùng có thể sử dụng các gói và các class có sẵn của .NET Framework hoặc sử dụng các gói và class do người dùng định nghĩa. Các gói và class này có thể được xây dựng bằng nhiều ngôn ngữ khác nhau miễn là các ngôn ngữ này cùng thuộc họ .NET. Phần người dùng định nghĩa là phần còn lại, người dùng có thể tự định nghĩa các gói, các lớp hoặc thêm vào một số file dữ liệu nếu cần.

### 2.3. Các dạng Project của Visual Studio

Hiện nay, một hệ thống thông tin thường có những dạng ứng dụng sau: Ứng dụng Console phục vụ xử lý các vấn đề liên quan đến hệ thống và giao tiếp vào ra; Ứng dụng Desktop phục vụ xây dựng các phần mềm ứng dụng với giao diện thân thiện; Ứng dụng Internet phục vụ việc xây dựng các website; Đối với mỗi dạng ứng dụng khác nhau, Visual Studio cung cấp các dạng Project khác nhau. Các dạng Project được Visual Studio cung cấp gồm có:

- Console Application: Cung cấp template cho ứng dụng Console
- Windows Application: Cung cấp template cho ứng dụng Desktop
- Class Library: Cung cấp template cho việc xây dựng thư viện liên kết động

- ASP.NET Website: Cung cấp template cho việc xây dựng Website
- ASP.NET Web Service: Cung cấp template cho việc xây dựng Web Service

### 3. Giới thiệu về ngôn ngữ lập trình C#

#### 3.1. Khái niệm

C# là một ngôn ngữ lập trình hướng đối tượng được phát triển bởi Microsoft. Microsoft phát triển C# dựa trên C++ và Java. C# được miêu tả là ngôn ngữ có được sự cân bằng giữa C++, Visual Basic, Delphi và Java. C# được thiết kế chủ yếu bởi Anders Hejlsberg kiến trúc sư phần mềm nổi tiếng với các sản phẩm Turbo Pascal, Delphi, J++, WFC.

#### 3.2. Đặc điểm

C# là ngôn ngữ lập trình phản ánh trực tiếp nhất đến .NET Framework mà tất cả các chương trình .NET chạy. C# phụ thuộc mạnh mẽ vào .NET Framework, mọi dữ liệu cơ sở đều là đối tượng, được cấp phát và hủy bỏ bởi trình dọn rác Garbage-Collector (GC). C# cung cấp nhiều kiểu trừu tượng khác chẳng hạn như class, delegate, interface, exception, v.v, phản ánh rõ ràng những đặc trưng của .NET runtime. So sánh với C và C++, ngôn ngữ này có một vài đặc điểm sau đây:

- Các con trỏ chỉ có thể được sử dụng trong chế độ không an toàn. Hầu hết các đối tượng được tham chiếu an toàn, và các phép tính đều được kiểm tra tràn bộ đệm.
- Các con trỏ chỉ được sử dụng để gọi các loại kiểu giá trị, còn những đối tượng thuộc bộ thu rác (garbage-collector) thì chỉ được gọi bằng cách tham chiếu.
- Các đối tượng không thể được giải phóng tường minh.
- Chỉ có đơn kế thừa, nhưng có thể cài đặt nhiều interface trừu tượng (abstract interfaces). Chức năng này làm đơn giản hóa sự thực thi của thời gian thực thi.
- C# thì an-toàn-kiểu (typesafe) hơn C++.
- Cú pháp khai báo mảng khác nhau(`int[] a = new int[5]` thay vì `int a[5]`).

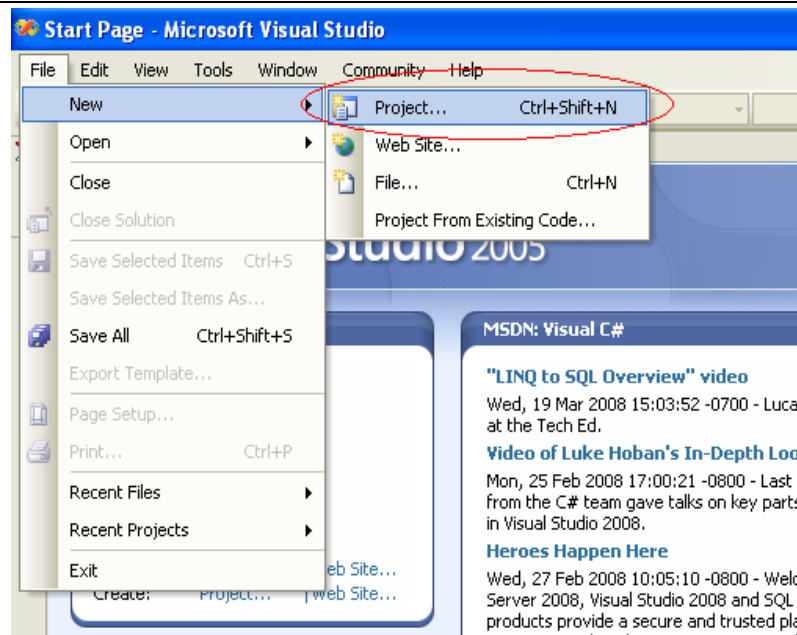
### 4. Bắt đầu với Console Application

Console Application là dạng Project phục vụ việc lập trình với các ứng dụng đơn giản. Với dạng Project này chúng ta dễ dàng thực hiện việc lập trình để mô phỏng thuật toán và mô phỏng hướng đối tượng. Các bước để khởi tạo Console Application như sau:

#### 4.1. Tạo Project

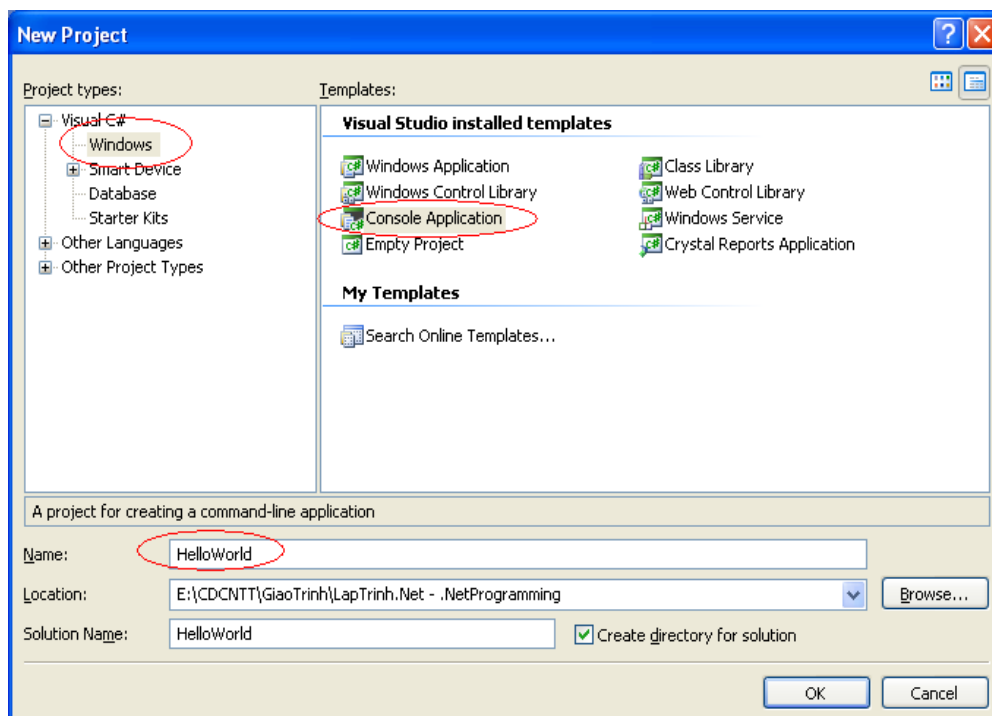
Ngay sau khi khởi động Visual Studio, chọn Menu File → New → Project.





Hình 1.12 – Khởi tạo Project

Sau khi chọn vào Project, Visual Studio hiển thị giao diện để người dùng chọn dạng Project ở mục Project types và Templates, đặt tên cho Project ở mục Name và đặt đường dẫn cho Project ở mục Location. Ở ví dụ ở hình 2.2, tài liệu trình bày cách tạo Project Console Application với tên là HelloWorld và Project này được đặt trong thư mục D: \LapTrinh.Net.

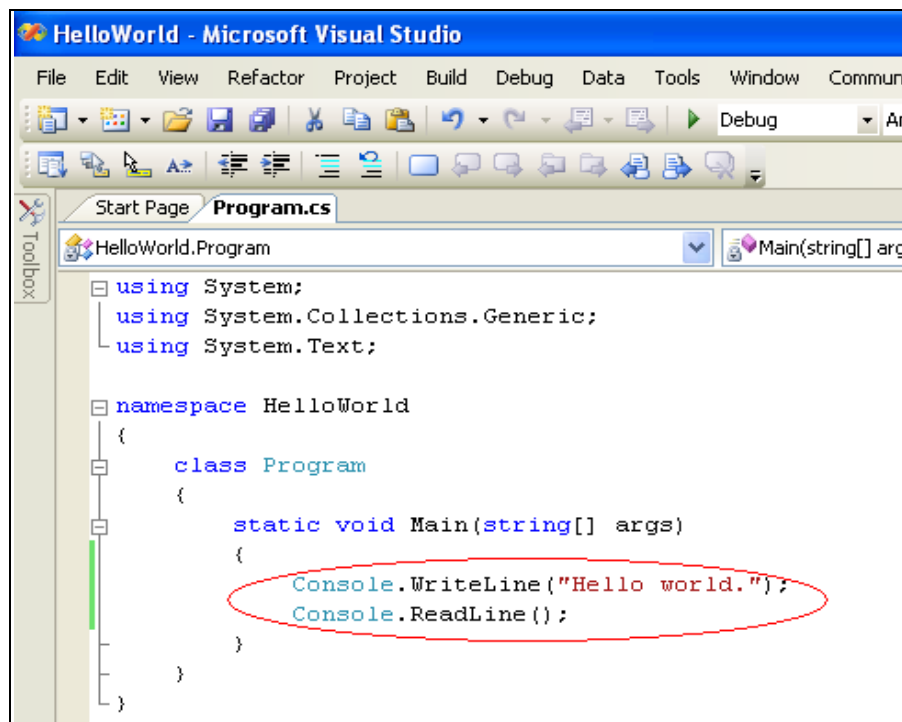


Hình 1.13 – Cấu hình Console Application Project

## 4.2. Lập trình

Ngay sau khi khởi tạo, Visual Studio sẽ tạo sẵn một Project với cấu trúc chuẩn. Trong Project, Visual Studio đã tạo sẵn một class có tên là Program nằm trong file

Program.cs, trong class này có sẵn phương thức Main(), người sử dụng chỉ cần lập trình ngay tại phương thức này.



Hình 1.14 – Lập trình trong Console Application của C#

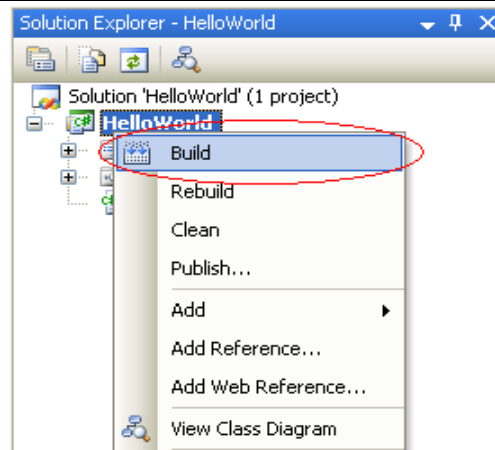
Trong ví dụ ở hình 2.3, trình bày cách lập trình để hiển thị dòng chữ “Hello world.” ra màn hình. C# cung cấp class Console để thực hiện việc xuất hoặc nhập dữ liệu. Dòng lệnh Console.WriteLine(); dùng để hiển thị một chuỗi ra màn hình. Dòng lệnh Console.ReadLine(); dùng để nhập một chuỗi từ bàn phím. Trong .NET Framework, Microsoft đưa ra thêm khái niệm namespace để quản lý các class trong cùng một thư viện. Với .NET, namespace là một thuật ngữ dùng để tham chiếu và được hiểu là một tập hợp các class. Như vậy, một thư viện (\*.dll) sẽ là tập hợp chứa các namespace và trong mỗi namespace chứa các class. Visual Studio tự tạo ra namespace mặc định trùng với tên Project mà bạn đã đặt. Trong trường hợp này, namespace mặc định là HelloWorld.

### 4.3. Biên dịch

Thông thường, một ứng dụng hoặc một dự án được tổ chức thành một Solution. Tùy vào mức độ lớn nhỏ, ta có 2 cách biên dịch đối với ứng dụng mà ta xây dựng: biên dịch từng phần và biên dịch toàn bộ.

#### 4.3.1. Biên dịch từng phần

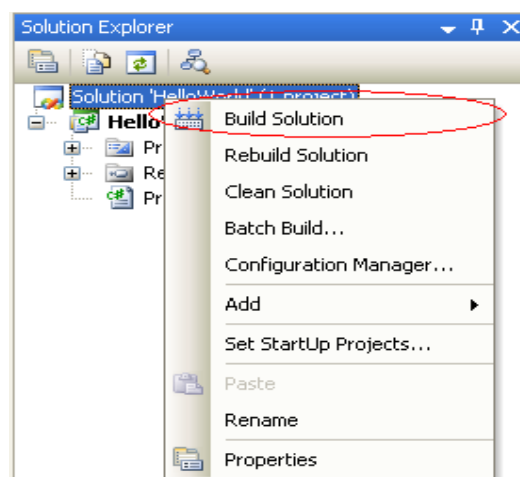
Biên dịch từng phần là hình thức biên dịch từng Project trong một Solution. Hình thức biên dịch này áp dụng đối với dự án đã được chia thành những thành phần riêng biệt. Với hình thức biên dịch này, tốc độ biên dịch sẽ nhanh và các lỗi dễ dàng được phân vùng để sửa chữa. Để thực hiện việc biên dịch từng phần, ta có thể click chuột phải (right-click) vào Project cần biên dịch và chọn Build.



Hình 1.15 – Biên dịch từng phần

#### 4.3.2. Biên dịch toàn phần

Biên dịch toàn bộ là hình thức biên dịch tất cả các Project trong một Solution. Hình thức biên dịch này được áp dụng đối với các ứng dụng vừa phải hoặc các ứng dụng mà tất cả các Project đều có liên quan mật thiết với nhau. Với hình thức biên dịch này, tốc độ biên dịch sẽ chậm, tuy nhiên tính đồng bộ được bảo đảm. Để thực hiện biên dịch toàn phần, ta có thể click phải chuột (right-click) vào Solution rồi chọn Build Solution.



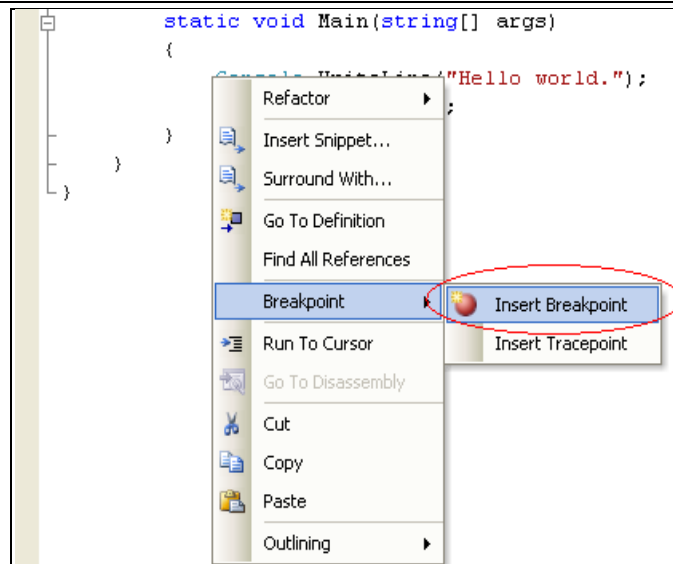
Hình 1.16 – Biên dịch toàn phần

### 4.4. Chạy chương trình

Visual Studio cung cấp 2 chế độ chạy chương trình: chế độ debug và chế độ non-debug.

#### 4.4.1. Chế độ debug

Chế độ debug là chế độ chạy từng dòng lệnh để người lập trình bắt lỗi. Trong chế độ này người lập trình quy định một số điểm dừng gọi là breakpoint, chương trình sẽ tự động dừng tại breakpoint để người dùng dễ dàng theo dõi kết quả của các lệnh chạy tiếp theo. Để tạo ra breakpoint, người lập trình chỉ cần click phải chuột (right-click) vào dòng lệnh cần dừng chọn breakpoint rồi chọn Insert Breakpoint.

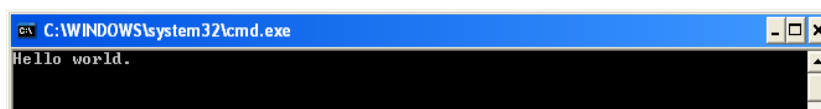


Hình 1.17 – Chế độ debug

Để thực hiện debug, người lập trình có thể bấm F5 hoặc chọn vào Menu Debug → Start Debug.

#### 4.4.2. Chế độ non-debug

Chế độ non-debug là chế độ chạy hết cả chương trình mà không dừng lại để bắt lỗi cho dù người lập trình đã thiết lập breakpoint. Để chạy chế độ này, người lập trình có thể bấm Ctrl+F5 hoặc chọn vào Menu Debug → Start Without Debugging. Với chương trình HelloWorld ở trên kết quả chạy chương trình như sau:



Hình 1.18 – Kết quả chạy chương trình

## 5. Biến và phạm vi hoạt động của biến trong C#

### 5.1. Biến

Biến là đơn vị được các ngôn ngữ lập trình tổ chức để lưu trữ và xử lý dữ liệu. Biến được khai báo theo cú pháp sau:

$$[phạm\_vi] <Kiểu\_Dữ\_Liệu> <tên\_biến> [ = <giá\_trị> ] ;$$

Trong đó: **[phạm\_vi]** là một trong những từ khóa public, private, protected, ...;

**Ví dụ 1.1:** Một biến mang tên i kiểu số nguyên int và có thể được truy cập bởi bất cứ hàm nào:

```
public int i;
```

**Ví dụ 1.2:** Ta cũng có thể khai báo biến và khởi tạo cho biến một giá trị như sau:

```
int i = 10;
```

**Ví dụ 1.3:** Nếu ta khai báo nhiều biến có cùng kiểu dữ liệu sẽ có dạng như sau:

```
int x=10; y=20;
```

## 5.2. Phạm vi hoạt động của biến

Trong C#, phạm vi hoạt động của biến là vùng đoạn mã mà từ đây biến có thể được truy xuất. Thông thường một đoạn mã được định nghĩa bằng một cặp dấu {}. Trong một phạm vi hoạt động (scope), không thể có hai biến cùng mang một tên trùng nhau.

**Ví dụ 1.4:** Thực hiện việc in ra các số từ 0 đến 9 ra màn hình rồi tiếp tục in các số từ 9 đến 0 ra màn hình:

```
using System;

static int Main()
{
    for (int i = 0; i < 10; i++)
    {
        Console.WriteLine(i);
    } // biến i ra khỏi phạm vi
    // Chúng ta có thể khai báo thêm biến i ở đây
    for (int i = 9; i >= 0; i--)
    {
        Console.WriteLine(i);
    }
    // biến i ra khỏi phạm vi ở đây
    return 0;
}
```

Với ví dụ trên, trong 2 vòng lặp for khác nhau, ta có thể khai báo cùng một biến i cho dù 2 vòng lặp này cùng nằm trong một khối lệnh. Điều này hợp lý bởi vì i được khai báo trong hai vòng lặp khác nhau và là biến cục bộ của 2 vòng lặp đó. Khi vòng lặp ngoài được thực hiện xong thì biến tự động được giải phóng và vì thế các biến ở các vòng lặp khác nhau thì có thể được đặt tên giống nhau.

## 6. Hằng

Một hằng (constant) là một biến nhưng trị không thể thay đổi được suốt thời gian thi hành chương trình. Đôi lúc ta cũng cần có những giá trị bao giờ cũng bất biến.

Hằng được khai báo như sau:

```
<const> <Kiểu_Dữ_Liệu> <tên_hằng> = <giá_trị> ;
```

### Ví dụ 1.5:

```
const double PI = 3.14159 ;
```

Hằng có những đặc điểm sau:

- Hằng bắt buộc phải được gán giá trị lúc khai báo. Một khi đã được khởi gán thì không thể viết đè lên.
- Trị của hằng phải có thể được tính toán vào lúc biên dịch, do đó không thể gán một hằng từ một trị của một biến.
- Hằng bao giờ cũng static, tuy nhiên ta không thể đưa từ khoá static vào khi khai báo hằng.

- Có ba thuận lợi khi sử dụng hằng trong chương trình của bạn:
- Hằng làm cho chương trình đọc dễ dàng hơn, bằng cách thay thế những con số vô cảm bởi những tên mang đầy ý nghĩa hơn.
- Hằng làm cho dễ sửa chương trình hơn, việc thay đổi giá trị chỉ cần thực hiện một lần ngay tại vị trí khai báo hằng.
- Hằng làm cho việc tránh lỗi dễ dàng hơn, nếu bạn gán một trị khác cho một hằng vốn đã được khai báo đâu đó trong chương trình trình biên dịch sẽ tự động thông báo lỗi vì hằng này đã được khai báo.

## 7. Kiểu dữ liệu

C# là một ngôn ngữ được kiểm soát chặt chẽ về mặt kiểu dữ liệu, ngoài ra C# còn chia các kiểu dữ liệu thành hai loại khác nhau: kiểu trị (value type) và kiểu qui chiếu (reference type). Nghĩa là trên một chương trình C# dữ liệu được lưu trữ một hoặc hai nơi tùy theo đặc thù của kiểu dữ liệu. C# cũng hỗ trợ kiểu con trỏ (pointer type) giống như C++ nhưng ít khi dùng đến và chỉ dùng khi làm việc với đoạn mã unmanaged. Đoạn mã unmanaged là đoạn mã được tạo ra ngoài .NET Framework, chẳng hạn những đối tượng COM.

### 7.1. Kiểu giá trị (Value Types)

Kiểu C#	Số byte	Kiểu .NET	Mô tả
byte	1	Byte	Số nguyên dương không dấu từ 0 đến 255
char	2	Char	Ký tự Unicode
bool	1	Boolean	Giá trị logic true / false
sbyte	1	Sbyte	Số nguyên có dấu từ -128 đến 127
short	2	Int16	Số nguyên có dấu từ -32768 đến 32767
ushort	2	UInt16	Số nguyên dương không dấu từ 0 đến 65535
int	4	Int32	Số nguyên có dấu từ -2.147.483.647 đến 2.147.483.647
uint	4	UInt32	Số nguyên có dấu từ -2.147.483.647 đến 2.147.483.647
float	4	Single	Kiểu dấu chấm động, giá trị xấp xỉ từ -3.4E-38 đến 3.4E+38, với 7 chữ số có nghĩa
double	8	Double	Kiểu dấu chấm động có độ chính xác gấp đôi, giá trị xấp xỉ từ -1.7E-308 đến 1.7E+308, với 15, 16 chữ số có nghĩa



decimal	8	Decimal	Có độ chính xác đến 28 con số và giá trị thập phân, được dùng trong tính toán tài chính, kiểu này đòi hỏi phải có hậu tố “m” hay “M”
long	8	Int64	Kiểu số nguyên có dấu có giá trị trong khoảng - 9.223.370.036.854.775.808 đến 9.223.372.036.854.775.807
ulong	8	UInt64	Số nguyên không dấu từ 0 đến 0xffffffffffffffff

## 7.2. Kiểu tham chiếu (Reference Type)

C# hỗ trợ 2 kiểu tham chiếu cơ bản: object, string. Kiểu object là kiểu dữ liệu gốc, tất cả các kiểu dữ liệu khác đều được dẫn xuất từ kiểu dữ liệu này. Kiểu string là kiểu dữ liệu trình bày chuỗi ký tự. Chuỗi string trong C# hỗ trợ hoàn toàn unicode.

Cũng giống như Ansi C và Java, C# định nghĩa giá trị của một chuỗi trong cặp dấu ngoặc kép “”, còn được gọi là Double Quote. Để tiện cho việc xử lý tất cả các ký tự, C# định nghĩa một số ký tự đặc biệt.

Danh sách một số ký tự đặc biệt thông dụng:

Kí tự	Ý nghĩa
\'	Dấu nháy đơn
\"	Dấu nháy kép
\\	Dấu chéo
\0	Ký tự null
\a	Alert
\b	Backspace
\f	Sang trang form feed
\n	Dòng mới
\r	Đầu dòng
\t	Tab ngang
\v	Tab dọc

Hình 2.9 – Danh sách một số ký tự đặc biệt

## 8. Toán tử

- học: + , - , \* , / , %
- Toán tử tăng / giảm: += , -= , \*= , /= , %=
- Toán tử tăng / giảm 1 đơn vị: ++ , --
- Toán tử gán: =

- Toán tử quan hệ: == , != , > , >= , < , <=
- Toán tử logic: ! , && , ||
- Toán tử 3 ngôi: (Điều\_Kiện) ? (Biểu\_Thức\_1) : (Biểu\_Thức\_2) ;

## 9. Xuất nhập dữ liệu

### 9.1. Lệnh xuất dữ liệu

```
Console.Write(giá_trị);
```

Hoặc `Console.WriteLine(giá_trị);`

#### Ví dụ 1.6:

```
string str="Hà Giang";
Console.WriteLine("Chào bạn:" + str);
```

### 9.2. Lệnh nhập dữ liệu từ bàn phím

+ Nhập dữ liệu cho biến chuỗi:

```
biến=Console.ReadLine();
```

#### Ví dụ 1.7:

```
string s=Console.ReadLine();
```

+ Nhập dữ liệu cho biến không phải kiểu chuỗi:

```
biến=<Kiểu.Net>.Parse(Console.ReadLine());
```

#### Ví dụ 1.8:

```
int a;
a=Int32.Parse(Console.ReadLine());
```

## 10. Cấu trúc rẽ nhánh

### 10.1. Câu lệnh điều kiện if..else

#### 10.1.1. Cú pháp

```
if (Điều_Kiện)
```

```
<Khối lệnh>
```

```
[else
```

```
<Khối lệnh>]
```

#### 10.1.2. Cách sử dụng

Lệnh điều kiện *if* thực hiện một hoặc nhiều lệnh trong *khối lệnh* nếu kết quả trả về của biểu thức *Điều\_kiện* là *true*, ngược lại, các lệnh hoặc *khối lệnh* ngay sau từ khóa *else* sẽ được thực hiện.

#### Ví dụ 1.9:

```
using System;
class Chan_Le
{
    static void Main()
```

```

{
    // Khai báo và khởi tạo biến
    int bienDem = 9;
    // Xuất ra màn hình
    if (bienDem % 2 == 0)
        Console.WriteLine("{0} là số chẵn", bienDem);
    else
        Console.WriteLine("{0} là số lẻ", bienDem);
}
}

```

**Ví dụ 1.10:** Sử dụng cấu trúc *if* lồng nhau

```
using System;
```

```

static void Main()
{
    Console.WriteLine("Nhập điểm trung bình:");
    double dtb = Double.Parse(Console.ReadLine());
    string xl = "";
    if (dtb < 0 || dtb > 10)
        Console.WriteLine("Điểm không hợp lệ!");
    else
    {
        if (dtb < 5)
            xl = "Yếu";
        else if (dtb < 7.0)
            xl = "Trung bình";
        else if (dtb < 8)
            xl = "Khá";
        else
            xl = "Giỏi";
        Console.WriteLine("Xếp loại: { 0}", xl);
        Console.ReadLine();
    }
}

```

## 10.2. Câu lệnh lựa chọn switch..case

Trong trường hợp phải dùng nhiều lệnh *if..else* lồng nhau, chương trình sẽ trở nên phức tạp và khó gỡ rối. Để khắc phục vấn đề này, C# cung cấp cấu trúc lệnh *switch..case* để phục vụ cho việc lập trình khi xử lý một loạt các trường hợp khác nhau của cùng một điều kiện.

### 10.2.1. Cú pháp

*switch* (Biểu\_Thức)

```

{
    case <giá_trị_1>:< Khối lệnh 1>
        [break;]
    case <giá_trị_2>:< Khối lệnh 2>
        [break;]
    ....
    [default:
        < Khối lệnh khác>]
}

```

}

### 10.2.2. Cách sử dụng

Với từng trường hợp của *biểu thức* điều kiện, các lệnh tương ứng sẽ được thực hiện. Trong cấu trúc lệnh này, lệnh *break* được sử dụng để nhảy ra khỏi khối lệnh. Chỉ dẫn *default* được sử dụng để giải quyết trường hợp của điều kiện mà trường hợp này không nằm trong tất cả các trường hợp đã liệt kê trong các chỉ dẫn *case*.

**Ví dụ 1.11:** In ra màn hình thứ tương ứng:

```
using System;
class Thu
{
    static void Main()
    {
        // Khai báo và khởi tạo biến
        int thu = 5; // 0: Chu nhật, 1: Thu hai, 2: Thu ba, 3: Thu tư,
                    // 4: Thu nam, 5: Thu sáu, 6: Thu bảy
                    // Xuất ra màn hình

        switch (thu)
        {
            case 0:
                Console.WriteLine("Chu nhật");
                break;
            case 1:
                Console.WriteLine("Thu hai");
                break;
            case 2:
                Console.WriteLine("Thu ba");
                break;
            case 3:
                Console.WriteLine("Thu tư");
                break;
            case 4:
                Console.WriteLine("Thu nam");
                break;
            case 5:
                Console.WriteLine("Thu sáu");
                break;
            case 6:
                Console.WriteLine("Thu bảy");
                break;
            default:
                Console.WriteLine("Không phải là thu trong tuần");
                break;
        }
    }
}
```

## 11. Cấu trúc lặp

### 11.1. Cấu trúc lặp while

#### 11.1.1. Cú pháp

*while* (*Điều\_kiện*)

< *Khối\_lệnh* >

### 11.1.2. Cách sử dụng

Cấu trúc *while* thực hiện việc lặp một hay nhiều lệnh khi *điều kiện* vẫn còn đúng. Khi sử dụng cấu trúc này, người lập trình cần phải chủ động thực hiện các câu lệnh tạo sự biến đổi để tránh những vòng lặp vô tận.

**Ví dụ 1.12:** Xuất ra màn hình tất cả các số nguyên từ 1 đến 100.

```
using System;
class UsingWhile
{
    static void Main()
    {
        // Khai báo và khởi tạo biến đếm
        int i = 1;
        // Xuất ra màn hình
        while (i <= 100)
        {
            Console.WriteLine("i = {0}", i);
            i++; // tăng biến đếm,
        }
    }
}
```

## 11.2. Cấu trúc lặp do..while

### 11.2.1. Cú pháp

```
do
    < Khối lệnh >
while (Điều_kiện) ;
```

### 11.2.2. Cách sử dụng

Cấu trúc lặp *do..while* thực hiện việc lặp một hoặc nhiều lệnh cho tới khi *điều kiện* có giá trị *false*. Cấu trúc lặp này có đặc điểm là các lệnh được thực hiện ít nhất một lần cho dù ngay từ đầu *điều kiện* đã là *false*.

**Ví dụ 1.13:** Xuất ra màn hình tất cả các số nguyên từ 1 đến 100.

```
using System;
class UsingDoWhile
{
    static void Main()
    {
        // Khai báo và khởi tạo biến đếm
        int i = 1;
        // Xuất ra màn hình
        do
        {
            Console.WriteLine("i = {0}", i);
            i++; // tăng biến đếm
        } while (i <= 100);
    }
}
```

## 11.3. Cấu trúc lặp for

### 11.3.1. Cú pháp

```
for ([Khởi_tạo] ; [Điều_kiện] ; [Bước_lặp])
    <Khối_lệnh>
```

### 11.3.2. Cách sử dụng

Lệnh *for* thực hiện việc lặp một hoặc nhiều lệnh trong khi *điều kiện* kết thúc vẫn còn đúng.

**Ví dụ 1.14:** Chương trình xuất ra màn hình tất cả các số dương chia hết cho 10 mà nhỏ hơn 100.

```
using System;
class UsingFor
{
    static void Main()
    {
        for (int i = 1; i < 100; i++)
            if (i % 10 == 0)
                Console.Write("{0} \n\r", i);
            else
                Console.Write("{0} ", i);
    }
}
```

## 11.4. Cấu trúc lặp foreach

### 11.4.1. Cú pháp

```
foreach (<kiểu dữ liệu> <phần tử> in <tập hợp>)
{
    <Khối_lệnh>
}
```

### 11.4.2. Cách sử dụng

Lệnh *foreach* thực hiện duyệt qua từng *phần tử* trong *tập hợp* để thực hiện các lệnh tương ứng.

**Ví dụ 1.15:**

```
using System;
class Program
{
    static void Main(string[] args)
    {
        int[] arr = new int[] { 5, 1, 4, 3, 5, 8 };
        foreach (int i in arr)
        {
            Console.WriteLine(i);
        }
    }
}
```



## 11.5. Các lệnh hỗ trợ cho cấu trúc lặp

### 11.5.1. Lệnh *break*

Lệnh *break* thực hiện việc dừng vòng lặp. Khi chương trình đang chạy mà gặp lệnh *break* thì chương trình sẽ lập tức chấm dứt vòng lặp cho dù điều kiện của vòng lặp vẫn cho phép chạy tiếp. Trong trường hợp có nhiều vòng lặp lồng nhau, chương trình sẽ chấm dứt vòng lặp gần với *break* nhất.

**Ví dụ 1.16:** Sử dụng vòng lặp *for* và câu lệnh *break* để tìm số nguyên lớn nhất trong khoảng từ 1 đến 100 chia hết cho *n* ( $0 < n \leq 100$ )

```
using System;

class Program
{
    static void Main(string[] args)
    {
        int i, max = 0;
        int n = int.Parse(Console.ReadLine());
        for (i = 100; i >= 1; i--)
        {
            if (i % n == 0)
            {
                max = i;
                break; //thoát khỏi vòng lặp
            }
        }
        Console.WriteLine("Số lớn nhất chia hết cho {0} là {1}", n, max);
    }
}
```

### 11.5.2. Lệnh *continue*

Trong khi thực hiện vòng lặp, người lập trình đôi khi cần thực hiện việc bỏ qua một số dòng lệnh để tiếp tục thực hiện việc lặp cho lần tiếp theo. C# hỗ trợ lệnh *continue* để thực hiện chức năng này. Lệnh *continue* thực hiện việc chuyển sang lần lặp tiếp theo và bỏ qua các lệnh nằm trong vòng lặp nhưng nằm phía sau nó.

**Ví dụ 2.17:** Sử dụng vòng lặp *for* và lệnh *continue* để tính tổng các số nguyên chẵn từ 1 đến 100:

```
using System;

class Program
{
    static void Main(string[] args)
    {
        int tong = 0;
        for (int i = 1; i <= 100; i++)
        {
            if (i % 2 != 0)
                continue;
            tong += i;
        }
    }
}
```