# C# and Visual Studio

Prepared for V$^{th}$ semester DDU-CE students
2022-23 WDDN

Apurva A Mehta

# Features of C#

- It is a modern, general-purpose programming language
- It is type safe, strongly typed.
- It is object oriented.
- It is component oriented.
- It is easy to learn.
- It is a structured language.
- It produces efficient programs.
- It can be compiled on a variety of computer platforms.
- It is a part of .NET Framework.

# Program Structure

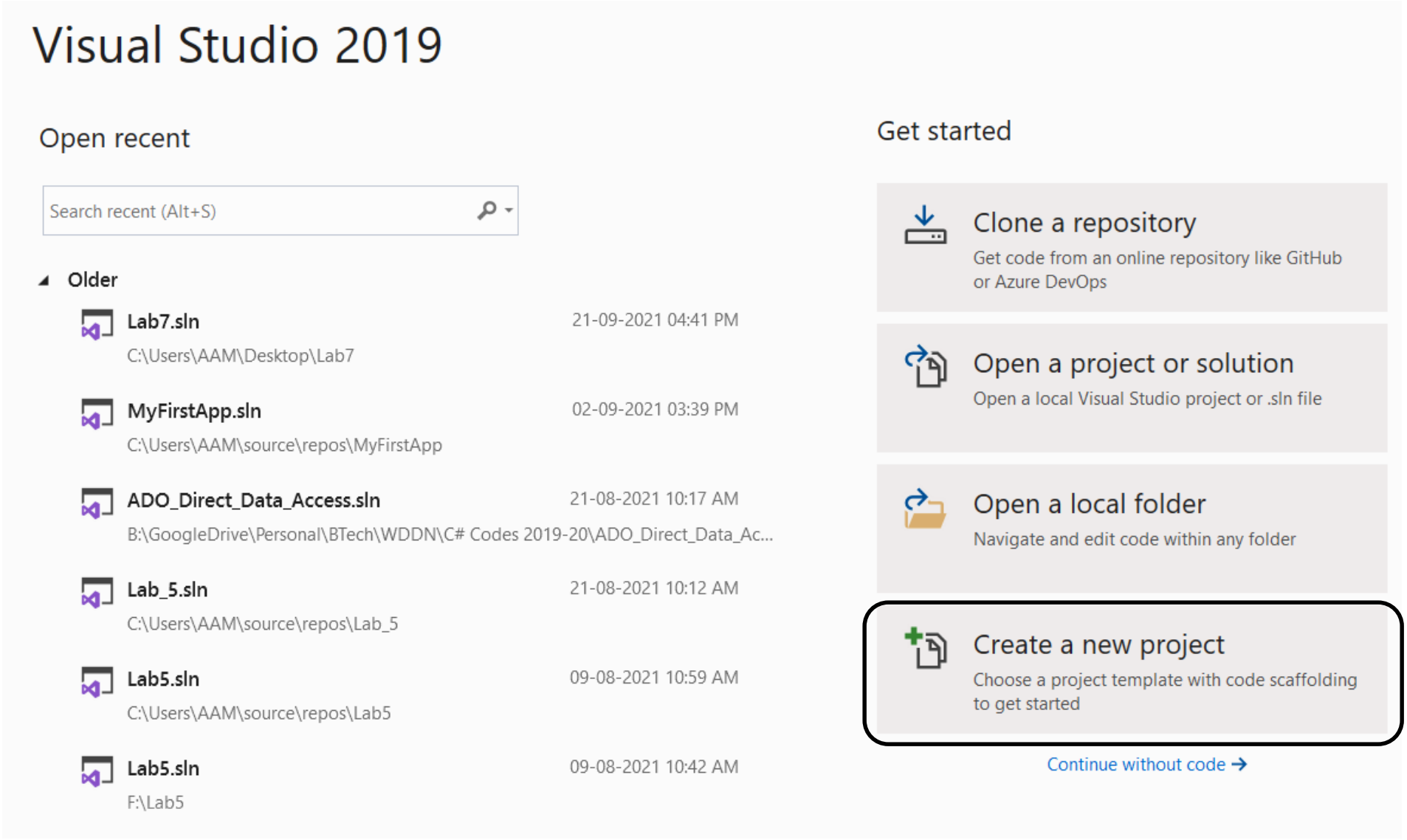
A C# program consists of the following parts:

- Namespace declaration
- A class
- Class methods
- Class attributes
- A Main method
- Statements and Expressions
- Comments

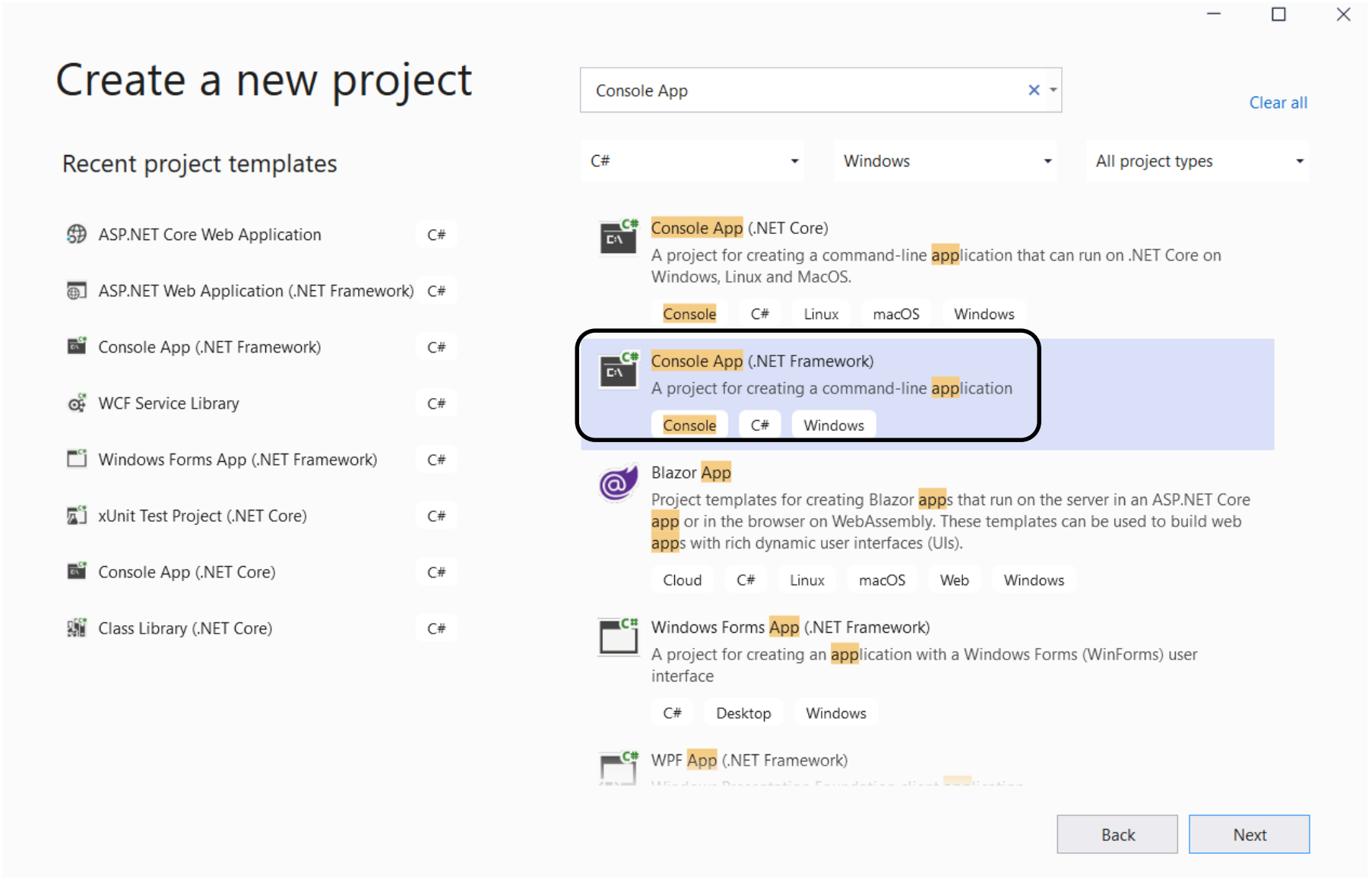
# Visual Studio 2019 Community edition

- Open Visual Studio 2019

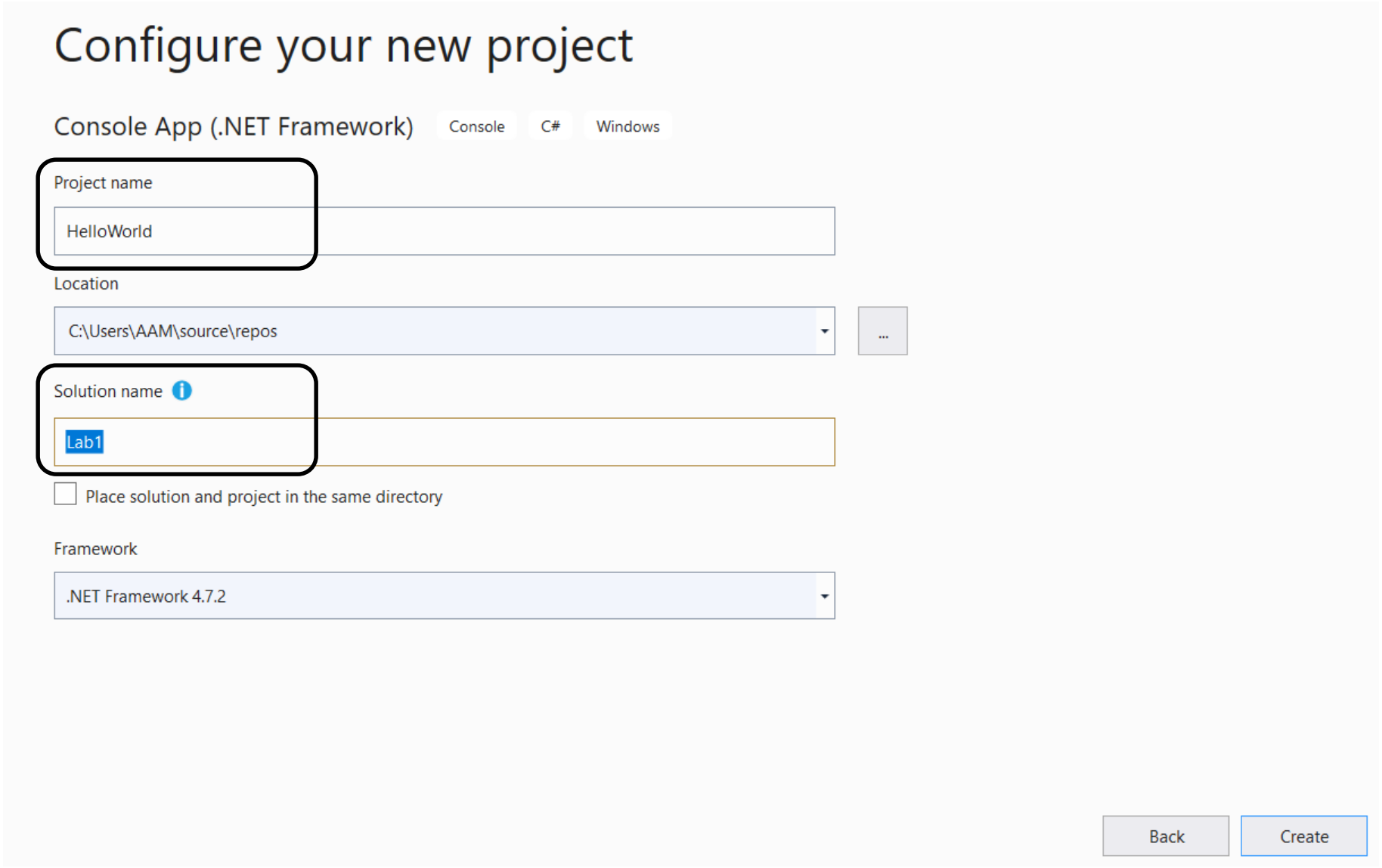
# Visual Studio 2019 Community edition



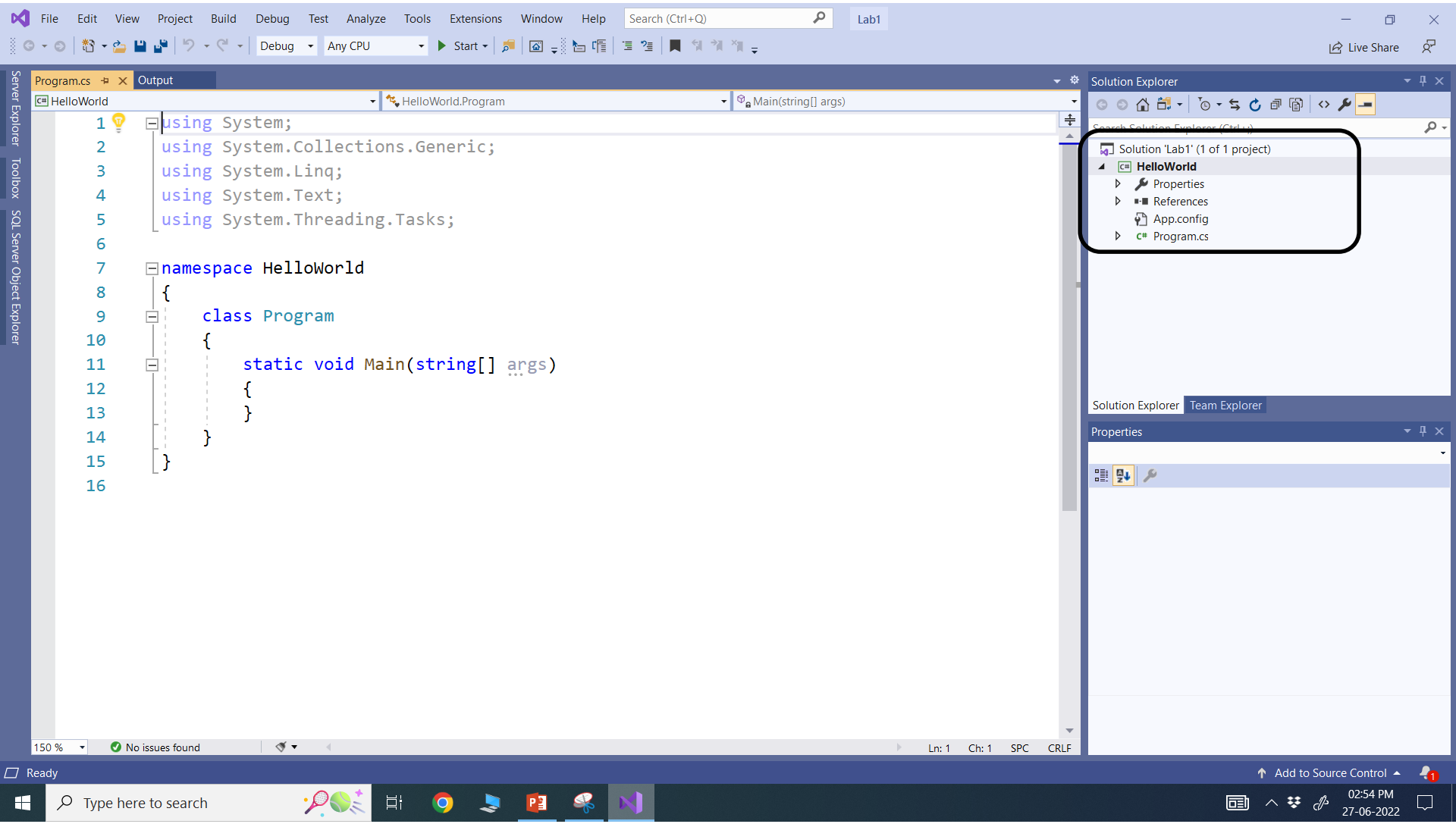
# Visual Studio 2019 Community edition



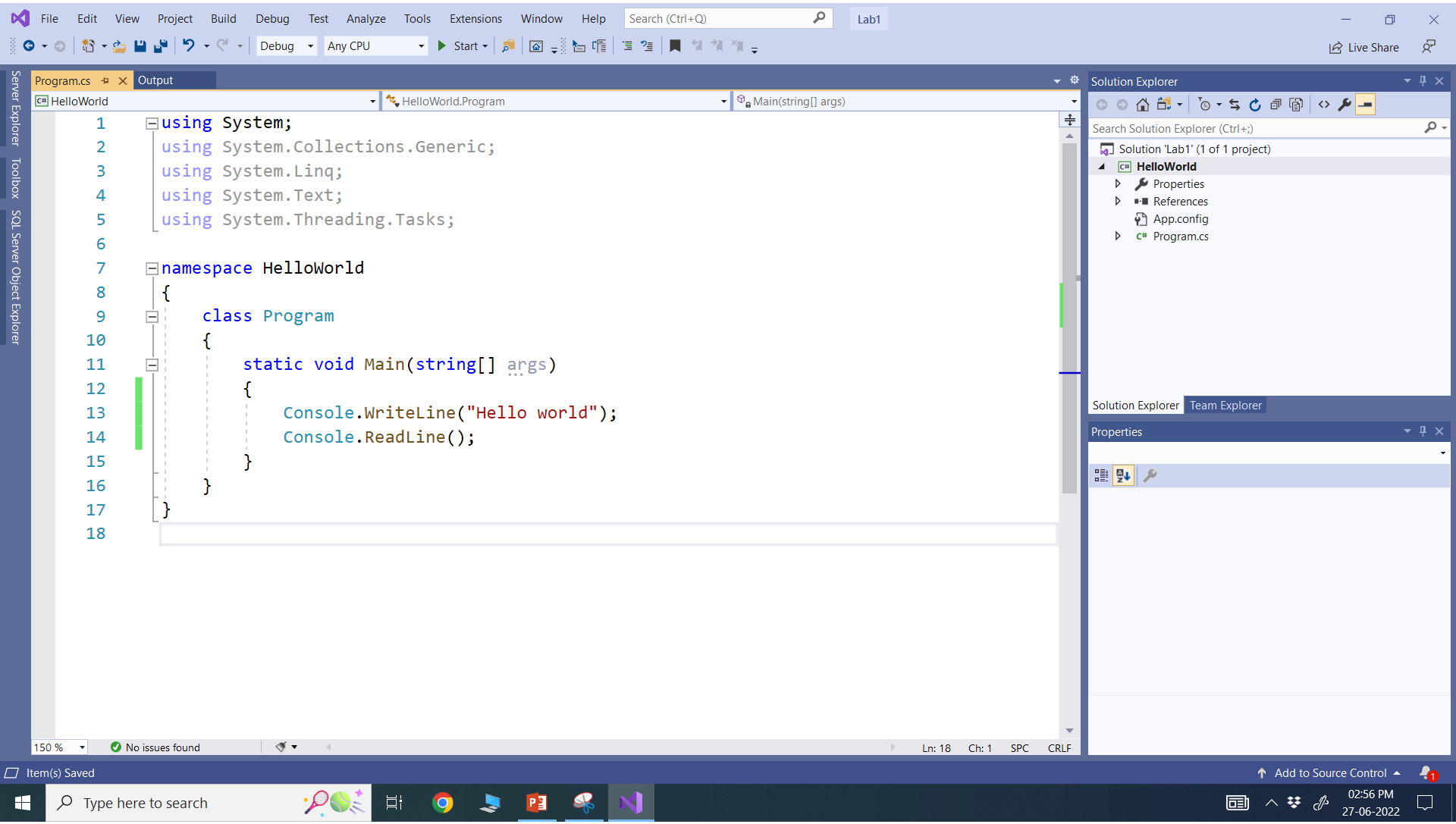
# Hello World!


Configure your new project
Console App (.NET Framework)
Console
C#
Windows
Project name
HelloWorld
Location
C:\Users\AAM\source\repos
...
Solution name
Lab1
Place solution and project in the same directory
Framework
.NET Framework 4.7.2
Back
Create

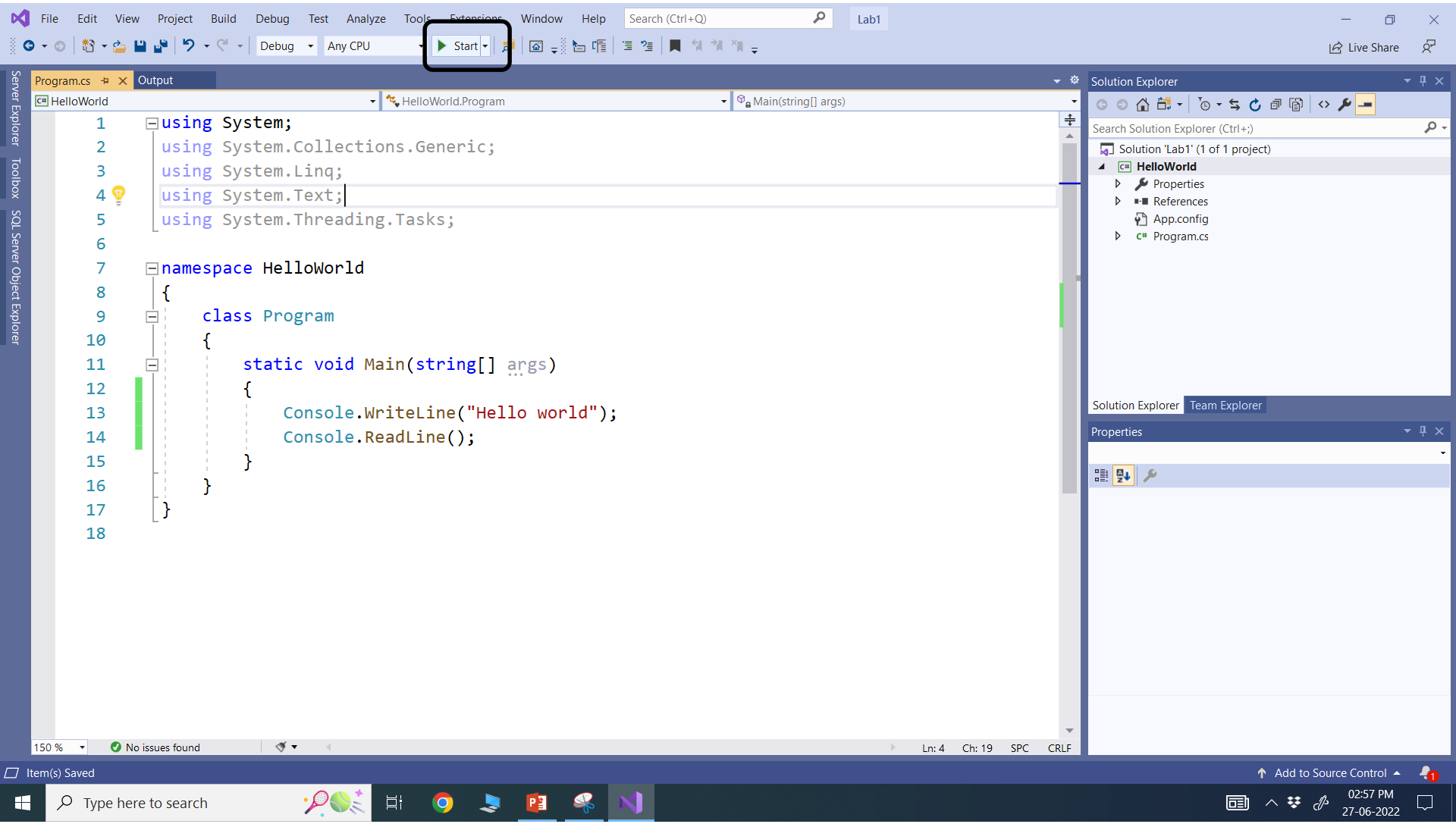
# Hello World!


File Edit View Project Build Debug Test Analyze Tools Extensions Window Help
Search (Ctrl+Q)
Lab1
Debug
Any CPU
Start
Live Share
Program.cs
Output
HelloWorld
HelloWorld.Program
Main(string[] args)
1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Text;
5 using System.Threading.Tasks;
6
7 namespace HelloWorld
8 {
9 class Program
10 {
11 static void Main(string[] args)
12 {
13 }
14 }
15 }
16
Solution Explorer
Solution 'Lab1' (1 of 1 project)
HelloWorld
Properties
References
App.config
Program.cs
Solution Explorer
Team Explorer
Properties
150 %
No issues found
Ln: 1
Ch: 1
SPC
CRLF
Ready
Add to Source Control
Type here to search
02:54 PM
27-06-2022

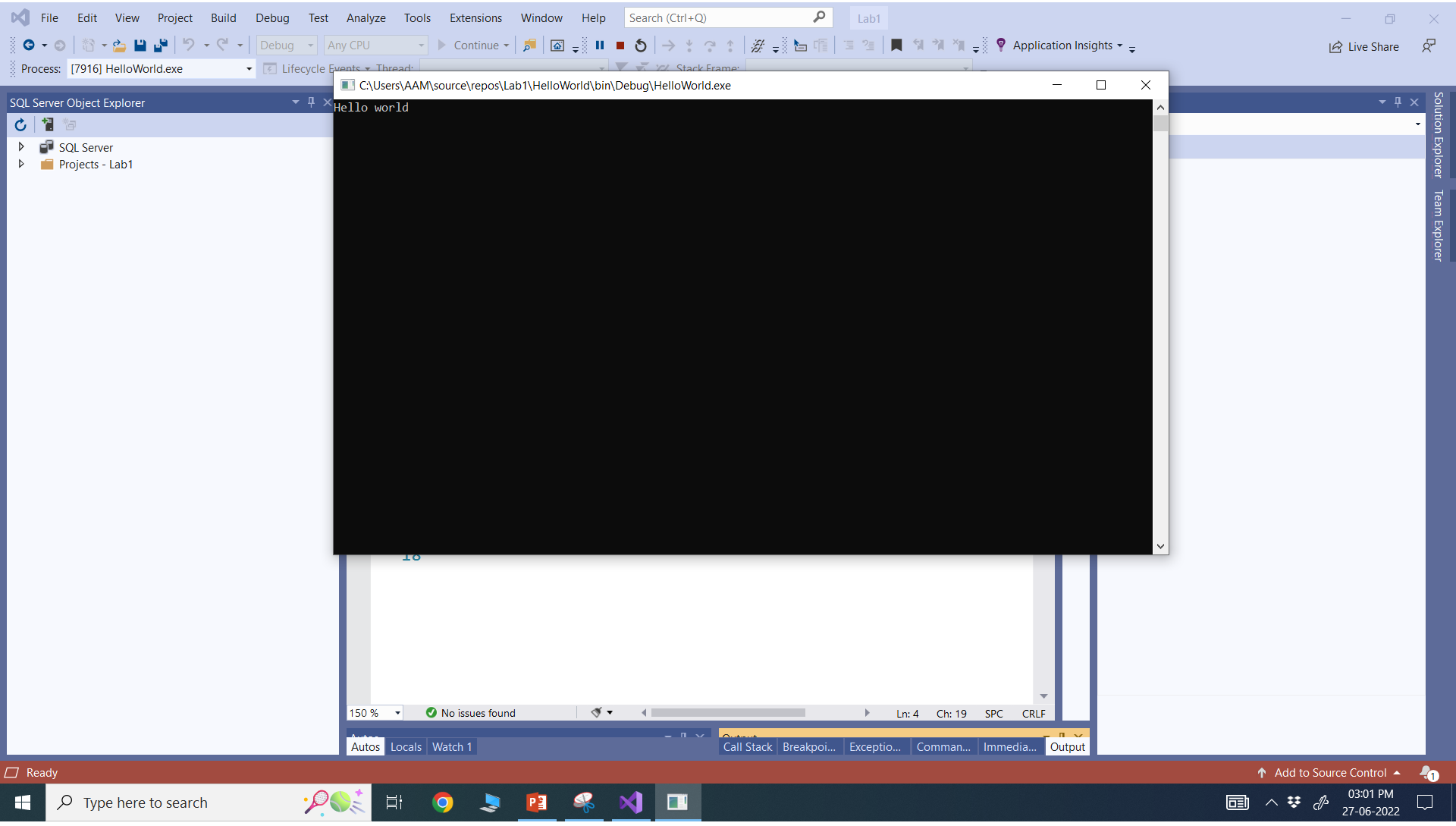
# Hello World!


File Edit View Project Build Debug Test Analyze Tools Extensions Window Help
Search (Ctrl+Q)
Lab1
Debug
Any CPU
Start
Live Share
Program.cs
Output
HelloWorld
HelloWorld.Program
Main(string[] args)
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
namespace HelloWorld
{
class Program
{
static void Main(string[] args)
{
Console.WriteLine("Hello world");
Console.ReadLine();
}
}
}
Solution Explorer
Search Solution Explorer (Ctrl+;)
Solution 'Lab1' (1 of 1 project)
HelloWorld
Properties
References
App.config
Program.cs
Solution Explorer
Team Explorer
Properties
150 %
No issues found
Ln: 18
Ch: 1
SPC
CRLF
Item(s) Saved
Add to Source Control
Type here to search
02:56 PM
27-06-2022

# Hello World!


File Edit View Project Build Debug Test Analyze Tools Extensions Window Help
Search (Ctrl+Q)
Lab1
Debug
Any CPU
Start
Live Share
Program.cs
Output
HelloWorld
HelloWorld.Program
Main(string[] args)
1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Text;
5 using System.Threading.Tasks;
6
7 namespace HelloWorld
8 {
9 class Program
10 {
11 static void Main(string[] args)
12 {
13 Console.WriteLine("Hello world");
14 Console.ReadLine();
15 }
16 }
17 }
18
Solution Explorer
Search Solution Explorer (Ctrl+;)
Solution 'Lab1' (1 of 1 project)
HelloWorld
Properties
References
App.config
Program.cs
Solution Explorer
Team Explorer
Properties
150 %
No issues found
Ln: 4
Ch: 19
SPC
CRLF
Item(s) Saved
Add to Source Control
Type here to search
02:57 PM
27-06-2022

# Hello World!


File Edit View Project Build Debug Test Analyze Tools Extensions Window Help
Search (Ctrl+Q)
Lab1
Debug
Any CPU
Continue
Application Insights
Live Share
Process: [7916] HelloWorld.exe
SQL Server Object Explorer
SQL Server
Projects - Lab1
C:\Users\AAM\source\repos\Lab1\HelloWorld\bin\Debug\HelloWorld.exe
Hello world
Solution Explorer
Team Explorer
150 %
No issues found
Ln: 4
Ch: 19
SPC
CRLF
Autos
Locals
Watch 1
Call Stack
Breakpoi...
Exceptio...
Comman...
Immedia...
Output
Ready
Add to Source Control
Type here to search
03:01 PM
27-06-2022

# Keywords

- Keywords
  - @ as a prefix
- Contextual keywords

There are total 78 keywords in C# as follows:

| | | | | |
|---|---|---|---|---|
| abstract | do | in | protected | throw |
| as | double | int | public | true |
| base | else | interface | readonly | try |
| bool | enum | internal | ref | typeof |
| break | event | is | return | unit |
| byte | explicit | lock | sbyte | ulong |
| case | extern | long | sealed | unchecked |
| catch | false | namespace | short | unsafe |
| char | finally | new | sizeof | ushort |
| checked | fixed | null | stackalloc | using |
| class | float | object | static | using static |
| const | for | operator | string | virtual |
| continue | foreach | out | struct | void |
| decimal | goto | override | switch | volatile |
| default | if | params | this | while |
| delegate | implicit | private | | |

| | | |
|---|---|---|
| add | alias | ascending |
| async | await | by |
| descending | dynamic | equals |
| from | get | global |
| group | into | join |
| let | nameof | on |
| orderby | partial (type) | partial (method) |
| remove | select | set |
| unmanaged (generic type constraint) | value | var |
| when (filter condition) | where (generic type constraint) | where (query clause) |
| yield | | |

# Reading and Writing on console

- Console.ReadLine()
- Console.WriteLine()
  - Concatenation

  Console.WriteLine(“Hello ”+userName)
  - Place holder

  Console.WriteLine(“Hello {0}”,userName)

  Console.WriteLine("FirstName: {0}, LastName:{1}", fName, lName);

  Console.WriteLine($”FirstName: {fName} LastName: {lName}”);

# Built in types in C#

- Boolean types
  - bool  b = true;
- Integral types
  - byte age = byte.MinValue;
  - uint populationOfIndia = uint.MinValue;
- Floating types
  - float temp = float.MinValue;
  - float n1 = 1.234f;
  - double pressure = double.MaxValue;
- Decimal types
  - decimal myMoney = 300.5m;
- String types
  - string name = “Apurva A Mehta”;

# Data Types

- *Value Type*
  - *Variable that directly contains the value*
  - *Store data in memory area called stack*
  - *Like all basic data types, structure, enum*
- *Reference Type*
  - *Reference types store the address of their data*
  - *Also known as a pointer*
  - *Stored in the area of memory called heap*
  - *Like class, array, string, delegate, interface*

# Continue…

```
0 references
static void Main(string[] args)
{
    int i = 100;
    Console.WriteLine(i);
    ChangeValue(i);
    Console.WriteLine(i);
}
1 reference
static void ChangeValue(int i)
{
    i = 200;
    Console.WriteLine(i);
}
```

```
static void Main(string[] args)
{
    Student std1 = new Student();
    std1.Name = "Ram";
    Console.WriteLine(std1.Name);
    ChangeReference(std1);
    Console.WriteLine(std1.Name);
}
1 reference
static void ChangeReference(Student std1)
{
    std1.Name = "Bharat";
    Console.WriteLine(std1.Name);
}
```

# Type Conversion

- Implicit conversions
  - Occur automatically
  - Guaranteed to succeed
  - No information (precision) loss
  - Smaller to larger integral types
  - Derived classes to base classes
- Explicit conversions
  - Require a cast
  - May not succeed
  - Information (precision) might be lost

# Continue…

```
int num = 2147483647;
long bigNum = num;

Derived d = new Derived();
Base b = d;

double x = 1234.7;
int a;
a = (int)x;

Giraffe g = new Giraffe();
Animal a = g;
Giraffe g2 = (Giraffe) a;
```

# Ways of type conversion

**Microsoft .NET provides three ways of type conversion:**

1. Parsing
2. Convert Class
3. Explicit Cast Operator ()

```
static void Main(string[] args)
{
    int age;
    double weight;

    Console.WriteLine("Please, enter your real integer age (in years)");
    age = int.Parse(Console.ReadLine());

    Console.WriteLine("Please, enter your real fractional weight (in KGs)");
    weight = double.Parse(Console.ReadLine());

    Console.WriteLine("Age: {0}, Weight: {1}", age, weight);
    Console.ReadLine();

}
```

```
static void Main(string[] args)
{
    string ageS = "29";
    int ageI = Convert.ToInt32(ageS);

    float weightf = 65.3f;
    string weightS = Convert.ToString(weightf);

    Console.WriteLine("AgeS: {0}, AgeI: {1}, WeightS: {2}, Weightf: {3}",
        ageS, ageS, weightS, weightf);

}
```

```
static void Main(string[] args)
{
    int n1, n2;
    float avg;
    n1 = 10;
    n2 = 20;
    avg = (float)(n1 + n2) / 2;
    Console.WriteLine("Avg: {0}", avg);

}
```

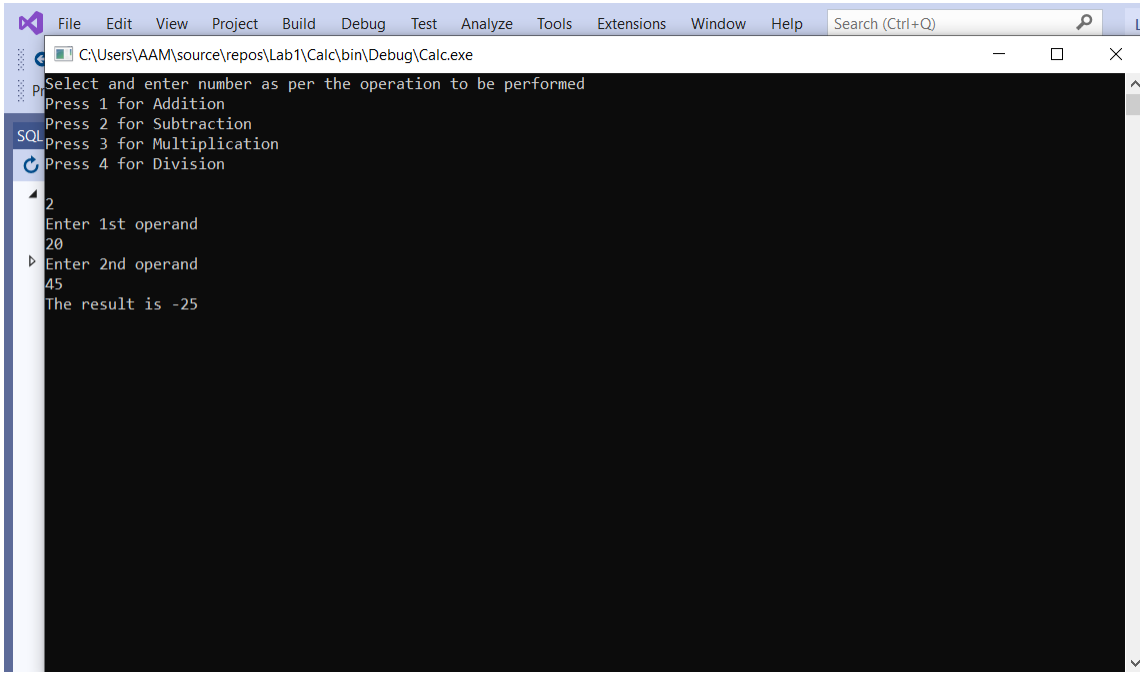
# C# Type Conversion Methods

| Sr.No | Methods & Description |
|---|---|
| 1 | **ToBoolean**<br>Converts a type to a Boolean value, where possible. |
| 2 | **ToByte**<br>Converts a type to a byte. |
| 3 | **ToChar**<br>Converts a type to a single Unicode character, where possible. |
| 4 | **ToDateTime**<br>Converts a type (integer or string type) to date-time structures. |
| 5 | **ToDecimal**<br>Converts a floating point or integer type to a decimal type. |
| 6 | **ToDouble**<br>Converts a type to a double type. |
| 7 | **ToInt16**<br>Converts a type to a 16-bit integer. |

| | |
|---|---|
| 8 | **ToInt32**<br>Converts a type to a 32-bit integer. |
| 9 | **ToInt64**<br>Converts a type to a 64-bit integer. |
| 10 | **ToSbyte**<br>Converts a type to a signed byte type. |
| 11 | **ToSingle**<br>Converts a type to a small floating point number. |
| 12 | **ToString**<br>Converts a type to a string. |
| 13 | **ToType**<br>Converts a type to a specified type. |
| 14 | **ToUInt16**<br>Converts a type to an unsigned int type. |
| 15 | **ToUInt32**<br>Converts a type to an unsigned long type. |
| 16 | **ToUInt64**<br>Converts a type to an unsigned big integer. |

# Ex.1

- Create a Console application that implements facility of simple calc.
- Note: Keep the project in same solution of Lab1.



# Summary of Variable Types

| Type | Example |
| --- | --- |
| Integral types | sbyte, byte, short, ushort, int, uint, long, ulong, and char |
| Floating point types | float and double |
| Decimal types | decimal |
| Boolean types | true or false values, as assigned |
| Nullable types | Nullable data types |

# Defining Constants

- Defined using **const** keyword.
- const <data_type> <constant_name> = value;

```
static void Main(string[] args)
{
    const double pi = 3.14159;
    double r;
    Console.WriteLine("Enter Radius");
    //double.TryParse(Console.ReadLine(),out r);
    r = Convert.ToDouble(Console.ReadLine());
    double areaCircle = pi * r * r;
    Console.WriteLine("Area: {0}", areaCircle);

}
```

# Escape sequences in C#

- string path = "C:\\Program Files\\Microsoft Visual Studio 10.0\\";

\' – single quote, needed for character literals

\" – double quote, needed for string literals

\\ – backslash

\0 – Unicode character 0

\a – Alert (character 7)

\b – Backspace (character 8)

\f – Form feed (character 12)

\n – New line (character 10)

\r – Carriage return (character 13)

\t – Horizontal tab (character 9)

\v – Vertical quote (character 11)

\uxxxx – Unicode escape sequence for character with hex value xxxx

\xn[n][n][n] – Unicode escape sequence for character with hex value nnnn (variable length version of \uxxxx)

\Uxxxxxxxx – Unicode escape sequence for character with hex value xxxxxxxx (for generating surrogates)

# Verbatim literal in C#

```
static void Main(string[] args)
{
    string path= @"C:\Program Files\Microsoft Visual Studio 10.0\";
    string name = @"Hello""World""!";
    Console.WriteLine("Path: {0}, Name: {1}", path,name);
}
```

# Common operators in C#

- Assignment operator =
- Arithmetic operator +, -, *, /, %
- Comparison operator ==, !=, >, <, >=, <=
- Conditional operator &&, ||
- Ternary operator ?:
- Null coalescing operator ??

```
static void Main(string[] args)
{
    int? x = null;
    int y = x ?? -1;
    Console.WriteLine("X: {0}, Y: {1}", x, y);
}
```

# Misc Operators

| Operator | Description | Example |
|---|---|---|
| sizeof() | Returns the size of a data type. | sizeof(int), returns 4. |
| typeof() | Returns the type of a class. | typeof(StreamReader); |
| & | Returns the address of an variable. | &a; returns actual address of the variable. |
| * | Pointer to a variable. | *a; creates pointer named 'a' to a variable. |
| ? : | Conditional Expression | If Condition is true ? Then value X : Otherwise value Y |
| is | Determines whether an object is of a certain type. | If( Ford is Car) // checks if Ford is an object of the Car class. |
| as | Cast without raising an exception if the cast fails. | Object obj = new StringReader("Hello"); StringReader r = obj as StringReader; |

# Flow Control

- The statements that allow us to control the *flow* of our program
  - Conditional (Selection) Statements
  - Loop (Iteration) Statements
  - Jump Statements

# Conditional Statements

- branch on certain condition
- && (optimization in evaluation) and &
- ||(optimization in evaluation) and |

- The IF Statement

```
if(condition)
        statement s1
 else
        statement s2
```

# Continue…

- The switch statements
  - Conditional statements
  - To replace multiple if else statements
  - Case value must be a constant expression

```
static void Main(string[] args)
{
    Console.WriteLine("Do you enjoy C#? (yes/no/may be)");
    string input = Console.ReadLine();

    switch(input.ToLower())
    {
        case "yes":
        case "may be":
            Console.WriteLine("Great");
            break;
        case "no":
            Console.WriteLine("So Sorry");
            break;
        default:
            Console.WriteLine("Read Carefully");
            break;
    }
}
```

# Loop statement -While

- While loop checks condition first
- When condition is true, statements within the loop are executed
- This process is repeated as long as the condition evaluates to true
- Do not forget to update the variable participating in the condition, so the loop can end, at some point

# Do while loop

- A do loop checks its condition at the end of the loop
- This means that the do loop is guaranteed to execute at least one time
- Do loops are used to present a menu to the user

# for loop

- For loop is very similar to while loop
- In a while loop we do the initialization at one place, condition check at another place and the variable modification at another place, where as for loop has all of these at one place

```
for (initializer; condition; iterator)
    body
```

# foreach loop

- Foreach loop is used to iterate through the item in a collection.
- Example of Collection include C# arrays
- Collection classes available in System.Collection namespace.

```
foreach (element in iterable-item)
{//body}
```

```
static void Main(string[] args)
{
    int n = 0;
    while (n < 5)
    {
        Console.WriteLine(n);
        n++;
    }
    int m = 0;
    do
    {
        Console.WriteLine(m);
        m++;
    } while (m < 5);
}
```

```
static void Main(string[] args)
{
    for (int i = 0; i < 5; i++)
    {
        Console.WriteLine(i);
    }

    char[] myArray = { 'H', 'e', 'l', 'l', 'o' };
    foreach (char ch in myArray)
    {
        Console.WriteLine(ch);
    }

}
```

# Jump statements

- Branching is performed using jump statements, which cause an immediate transfer of the program control
  - Break
  - Continue
  - Go to
  - Return
  - Throw

# break

- The break statement terminates the closest enclosing loop or switch statement in which it appears.
- Control is passed to the statement that follows the terminated statement, if any.

```
static void Main()
{
    for (int i = 1; i <= 100; i++)
    {
        if (i == 5)
        {
            break;
        }
        Console.WriteLine(i);
    }
}
```

# continue

- The continue statement passes control to the next iteration of the enclosing while, do, for, or foreach statement in which it appears.

```
static void Main()
{
    for (int i = 1; i <= 10; i++)
    {
        if (i < 9)
        {
            continue;
        }
        Console.WriteLine(i);
    }
}
```

# go to

- The goto statement transfers the program control directly to a labelled statement.
- A common use of goto is to transfer control to a specific switch-case label or the default label in a switch statement.
- The goto statement is also useful to get out of deeply nested loops.
- Can’t Jump in to loop
- Can’t jump out of class
- Can’t exit finally block after Try..Catch Block

# Continue…

```
static void Main(string[] args)
{
    Console.WriteLine("Do you enjoy C#? (yes/no/may be)");
    string input = Console.ReadLine();
    switch (input.ToLower())
    {
        case "yes":
            Console.WriteLine("Great");
            break;
        case "may be":
            goto case "yes";
        case "no":
            Console.WriteLine("So Sorry");
            break;
        default:
            Console.WriteLine("Read Carefully");
            break;
    }
    goto Finish;
    Finish: Console.WriteLine("End");
}
```

# return

- The return statement terminates execution of the method in which it appears and returns control to the calling method.

```
static double CalculateArea(int r)
{
    double area = r * r * Math.PI;
    return area;
    Console.WriteLine("Hello");
}

0 references
static void Main()
{
    int radius = 5;
    double result = CalculateArea(radius);
    Console.WriteLine("The area is {0:0.00}", result);
}
```

# throw

- Signals the occurrence of an exception during program execution.

```
public class NumberGenerator
{
    int[] numbers = { 2, 4, 6, 8, 10, 12, 14, 16, 18, 20 };
    1 reference
    public int GetNumber(int index)
    {
        if (index < 0 || index >= numbers.Length)
        {
            throw new IndexOutOfRangeException();
        }
        return numbers[index];
    }

    0 references
    static void Main(string[] args)
    {
        NumberGenerator n = new NumberGenerator();
        Console.WriteLine(n.GetNumber(10));
    }
}
```

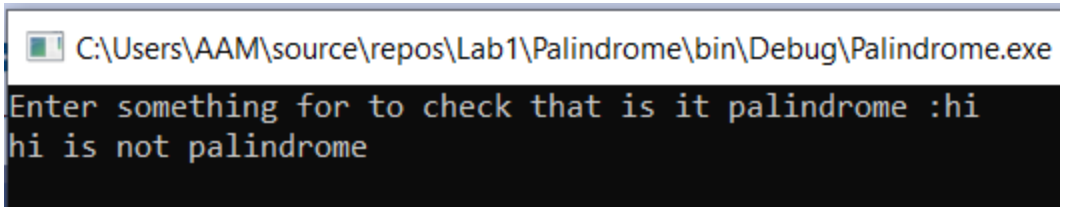
# Enumerations

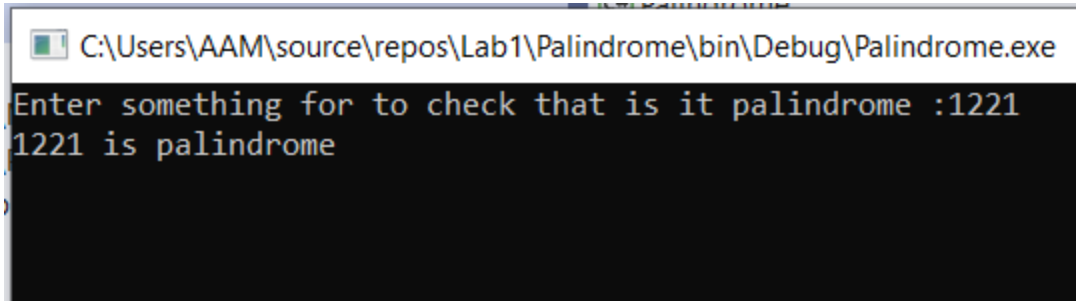
- Enums are strongly typed constants
- If a program uses set of integral numbers, consider replacing them with enums. Otherwise the program becomes less Readable, Maintainable

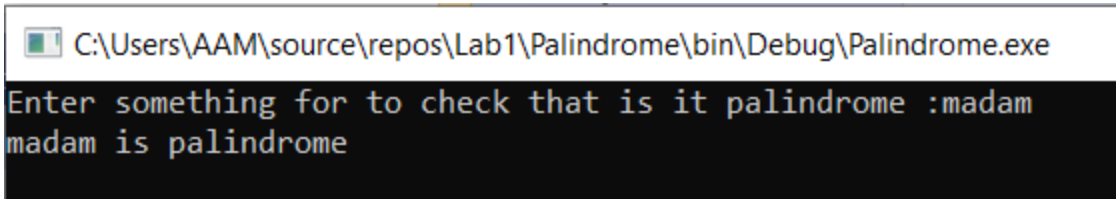
```
public enum Gender
{
    unknown = 0,
    male = 1,
    female = 2
}
0 references
static void Main(string[] args)
{
    Console.WriteLine(Gender.female);
    Console.WriteLine("\n");
    Console.WriteLine((int)Gender.female);
}
```

# Ex.2

- Create a Console application that implements facility for verification of palindrome input.
- Note: Keep the project in same solution of Lab1.

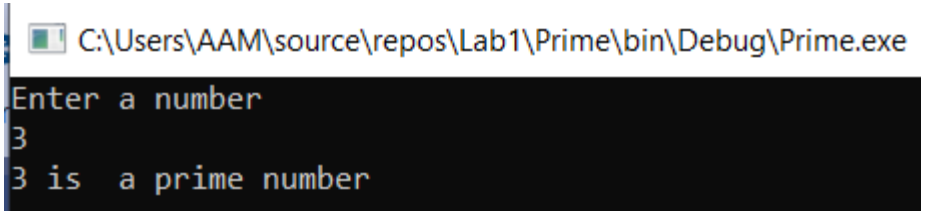


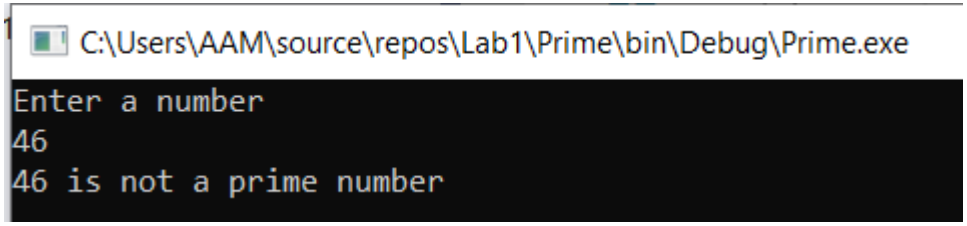



# Ex.3

- Create a Console application that implements facility for verification of prime input.
- Note: Keep the project in same solution of Lab1.



# Boxing and unboxing

- Boxing is the process of converting a value type to the type object or to any interface type implemented by this value type.
- When the CLR boxes a value type, it wraps the value inside a System.Object and stores it on the managed heap.
- Unboxing extracts the value type from the object.

# Cont…

- Boxing is implicit, unboxing is explicit.
- The concept of boxing and unboxing underlies the C# unified view of the type system in which a value of any type can be treated as an object.

```
int i = 123;
object o = i; // i is boxed and assigned to object


o = 123;
i = (int)o; // o is unboxed and assigned to i
```

```
// String.Concat example.
// String.Concat has many versions. Rest the mouse pointer on
// Concat in the following statement to verify that the version
// that is used here takes three object arguments. Both 42 and
// true must be boxed.
Console.WriteLine(String.Concat("Answer", 42, true));


// List example.
// Create a list of objects to hold a heterogeneous collection
// of elements.
List<object> mixedList = new List<object>();

// Add a string element to the list.
mixedList.Add("First Group:");

// Add some integers to the list.
for (int j = 1; j < 5; j++)
{
    // Rest the mouse pointer over j to verify that you are adding
    // an int to a list of objects. Each element j is boxed when
    // you add j to mixedList.
    mixedList.Add(j);
}
```

```
// Display the elements in the list. Declare the loop variable by
// using var, so that the compiler assigns its type.
foreach (var item in mixedList)
{
    // Rest the mouse pointer over item to verify that the elements
    // of mixedList are objects.
    Console.WriteLine(item);
}
```

```
int sum = mixedList[1] + mixedList[2];
int sum = (int)mixedList[1] + (int)mixedList[2];
```

```
public interface IEntity {
    bool Validate();
}

public class EmployeeClass : IEntity {
    public bool Validate() { return true; }
}

public struct EmployeeStruct : IEntity {
    public bool Validate() { return true; }
}

IEntity emp2 = new EmployeeStruct();

IEntity emp1 = new EmployeeClass();

var empStruct = (EmployeeStruct)emp2;

var empClass = (EmployeeClass)emp2;
```

# Methods in C#

- Static Methods vs Instance Methods
- Method parameters
  - Value
  - Reference --- *ref*
  - Out--- *out*
  - Parameters arrays --- *params*

```
1 reference
public static void PassByValue(int j)
{
    j = 101;
}

1 reference
public static void PassByReference(ref int j)
{
    j = 101;
}

1 reference
public static void Calculate(int fN,int sN, out int sum, out int product)
{
    sum = fN + sN;
    product = fN * sN;
}
```

```
0 references
public static void ParamMethod(params int[] Numbers, int i)
{ }
1 reference
public static void ParamMethod(params int[] Numbers)
{
    Console.WriteLine("There are {0} elements", Numbers.Length);

    foreach(int i in Numbers)
    {
        Console.WriteLine(i);
    }
}
```

```
static void Main(string[] args)
{
    int i = 0;
    int sumTotal,productTotal=0;
    int[] numbers = new int[3];
    numbers[0] = 101;
    numbers[1] = 102;
    numbers[2] = 102;
    PassByValue(i);
    Console.WriteLine("Pass by value: {0}",i);
    PassByReference(ref i);
    Console.WriteLine("Pass by reference: {0}", i);
    Calculate(10, 20,out sumTotal,out productTotal);
    Console.WriteLine("Out parameters:: {0}  {1}", sumTotal,productTotal);

    //ParamMethod();
    //ParamMethod(numbers);
    ParamMethod(1, 2, 3, 4, 5, 6);
    Console.ReadLine();
}
```

# C# Arrays

- Collection of similar data types
- Strongly typed
- Cannot grow in size
- Rely on integral indices to store or to retrieve
- To declare an array in C#, you can use the following syntax:

  datatype[] arrayName;
- Array is a reference type, so you need to use the **new** keyword to create an instance of the array.

  double[] balance = new double[10];

# Continue…

- The Array class is the base class to all arrays, and provides various properties and methods for working with arrays.

```
int[ ] a1 = new int[10];
int[ , ] a2 = new int[10, 5];
int [ , , ] a3 = new int[10, 5, 2];
int[] a = new int[] {1, 2, 3};
int[] t = new int[3];
t[0] = 1;
t[1] = 2;
t[1] = 3;
```

```
// Declare the array of two elements:
int[][] arr = new int[2][];

// Initialize the elements:
arr[0] = new int[5] { 1, 3, 5, 7, 9 };
arr[1] = new int[4] { 2, 4, 6, 8 };


// Display the array elements:
for (int i = 0; i < arr.Length; i++)
{
    Console.Write("Element({0}): ", i);

    for (int j = 0; j < arr[i].Length; j++)
    {
        Console.Write("{0} ", arr[i][j]);
    }
    Console.WriteLine();
}
// Keep the console window open in debug mode.
Console.WriteLine("Press any key to exit.");
Console.ReadKey();
```

# C# Strings

- In C#, you can use strings as array of characters.
- However, more common practice is to use the **string** keyword to declare a string variable.
- The string keyword is an alias for the **System.String** class.

  char a = ‘A’;

  string b = “Here’s a tab: \t”;
- Property of String class
  - Length : - Get Length of String.

```
string s1 = "Hello ";
string s2 = 2.ToString();
string s3 = s2.Insert(1," World!");
string s4 = string.Concat(s1, s3);
Console.WriteLine(s4);

string s5 = string.Copy(s4);
if(s5.Equals(s4))
{
    Console.WriteLine("S4: {0} S5:{1}  Equals!", s4, s5);
}
if((string.Equals(s1,s2)) == false)
{
    Console.WriteLine("S1: {0} S2:{1}  Not equals!", s1, s2);
}

Console.WriteLine("Substring:{0}", s4.Substring(5));
Console.WriteLine("Substring:{0}", s4.Substring(0,5));

Console.ReadLine();
```

# C# Structure

- Provides unique way of packing together data of different types

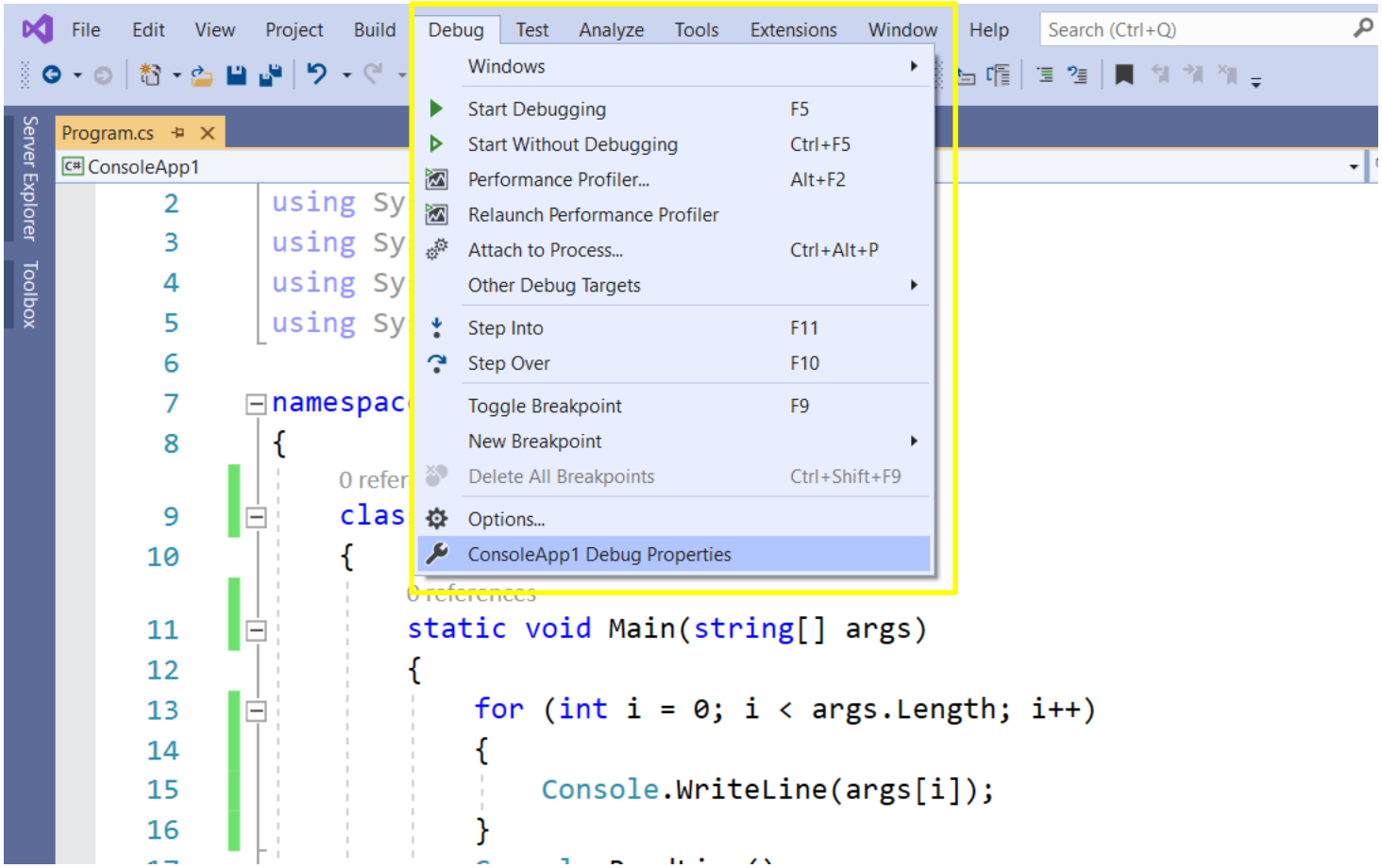
```
struct Student
{
  public string name;
  public int rollNumber;
  public double totalMarks;
}
```

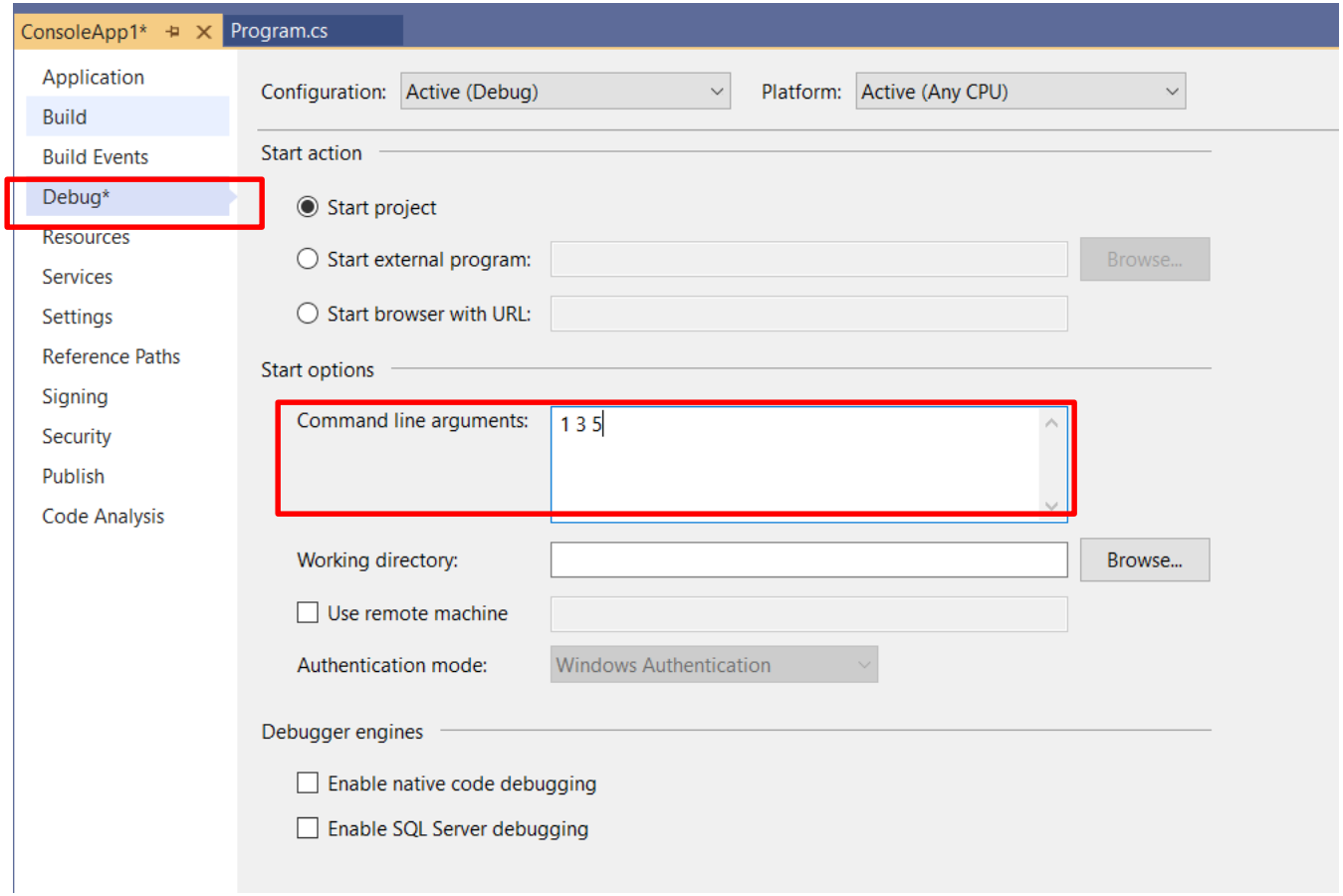
```
StringBuilder s = new StringBuilder();
s.Append("C# is an");
s.Append(" Object Oriented Language");
Console.WriteLine(s);

int n = s.Length;
s.Insert(n, ". Ok.");
Console.WriteLine(s);

s.Remove(n-2,2);
Console.WriteLine(s);

Console.ReadLine();
```

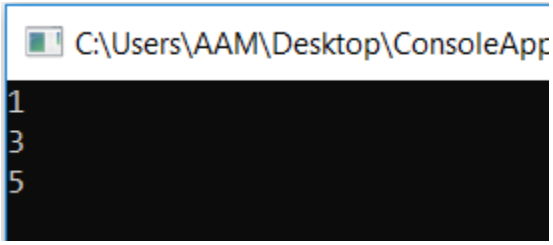
```
public struct Item
{
    public string name;
    public int code;
    public double price;
}
0 references
class Program
{
    0 references
    static void Main(string[] args)
    {
        Item fan;
        fan.name = "Master";
        fan.code = 32245;
        fan.price = 2349.99;

        Console.WriteLine("Name: {0}, Code: {1}, Price: {2}",
            fan.name, fan.code, fan.price);
    }
}
```

```
struct Rectangle
{
    int a, b;
    1 reference
    public Rectangle(int x, int y)
    {
        a = x;
        b = y;
    }
  1 reference
    public int Area()
    {
        return a * b;
    }
    1 reference
    public void Display()
    {
        Console.WriteLine("Area: {0}", Area());
    }
}

 static void Main(string[] args)
 {
     Rectangle rect = new Rectangle(10, 20);
     rect.Display();
 }
```

```
namespace ConsoleApp1
{
    0 references
    class Program
    {
        0 references
        static void Main(string[] args)
        {
            for (int i = 0; i < args.Length; i++)
            {
                Console.WriteLine(args[i]);
            }
            Console.ReadLine();
        }
    }
}
```


File
Edit
View
Project
Build
Debug
Test
Analyze
Tools
Extensions
Window
Help
Search (Ctrl+Q)
Windows
Start Debugging F5
Start Without Debugging Ctrl+F5
Performance Profiler... Alt+F2
Relaunch Performance Profiler
Attach to Process... Ctrl+Alt+P
Other Debug Targets
Step Into F11
Step Over F10
Toggle Breakpoint F9
New Breakpoint
Delete All Breakpoints Ctrl+Shift+F9
Options...
ConsoleApp1 Debug Properties
Server Explorer
Toolbox
Program.cs
ConsoleApp1
using Sy
using Sy
using Sy
using Sy
namespac
{
0 refer
clas
{
0 references
static void Main(string[] args)
{
for (int i = 0; i < args.Length; i++)
{
Console.WriteLine(args[i]);
}


ConsoleApp1*
Program.cs
Application
Build
Build Events
Debug*
Resources
Services
Settings
Reference Paths
Signing
Security
Publish
Code Analysis
Configuration: Active (Debug)
Platform: Active (Any CPU)
Start action
Start project
Start external program:
Browse...
Start browser with URL:
Start options
Command line arguments: 1 3 5
Working directory:
Browse...
Use remote machine
Authentication mode: Windows Authentication
Debugger engines
Enable native code debugging
Enable SQL Server debugging


C:\Users\AAM\Desktop\ConsoleApp
1
3
5