

INTRODUCTION TO C#

C# is a multi-paradigm programming language encompassing strong typing, imperative, declarative, functional, generic, object-oriented (class-based), and component-oriented programming disciplines.

It was developed by Microsoft within its .NET initiative and later approved as a standard by European Computer Manufacturers Association (ECMA-334) and International Organization for Standardization/ International Electrotechnical Commission (ISO/IEC 23270:2006).

C# is a general-purpose, Object-Oriented Programming Language and web enabled programming language.

Its development team is led by Anders Hejlsberg and Scott Wiltamuth.

The most recent version is C# 9.0, which was released in November 10 2020 along with Visual Studio 2019 version 16.8 and .Net Framework version 5.0.

History of C#

During the development of the .NET Framework, the class libraries were originally written using a managed code compiler system called Simple Managed C (SMC).

In January 1999, Anders Hejlsberg formed a team to build a new language at the time called **Cool**, which stood for "**C-like Object Oriented Language**".

Microsoft had considered keeping the name "Cool" as the final name of the language, but chose not to do so for trademark reasons. By the time the .NET project was publicly announced at the July 2000 Professional Developers Conference, the language had been renamed C#, and the class libraries and ASP.NET runtime had been ported to C#.

Hejlsberg is C#'s principal designer and lead architect at Microsoft, and was previously involved with the design of Turbo Pascal, Embarcadero Delphi (formerly CodeGear Delphi, Inprise Delphi and Borland Delphi), and Visual J++.

In interviews and technical papers he has stated that flaws in most major programming languages (e.g. C++, Java, Delphi, and Smalltalk) drove the fundamentals of the

Common Language Runtime (CLR), which, in turn, drove the design of the C# language itself.

Microsoft first used the name C# in 1988 for a variant of the C language designed for incremental compilation. That project was not completed but the name lives on.

Why C#

C# is pronounced as "C-Sharp". It is an object-oriented programming language provided by Microsoft that runs on .Net Framework.

By the help of C# programming language, we can develop different types of secured and robust applications:

- ✓ Console Applications.
- ✓ Window applications.
- ✓ Web applications.
- ✓ Developing windows controls.
- ✓ Creating web controls.
- ✓ Distributed applications.
- ✓ Web service applications.
- ✓ Database applications etc.

C# is approved as a standard by European Computer Manufacturers Association (ECMA) and International Standards Organization (ISO).

C# was developed by Anders Hejlsberg and his team during the development of .Net Framework. C# is designed for Common Language Infrastructure (CLI), which consists of the executable code and runtime environment that allows use of various high-level languages on different computer platforms and architectures.

C# programming language is influenced by C++, Java, Eiffel, Modula-3, Pascal etc. languages.

Strong Programming Features of C#

Although C# constructs closely follow traditional high-level languages, C and C++ and being an object-oriented programming language. It has strong resemblance with Java, it has numerous strong programming features that make it endearing to a number of programmers worldwide.

Following is the list of few important features of C# –

- ✓ First component-oriented language in C/C++ family.
- ✓ Pure object orientation without compromising efficiency.
- ✓ Cross language interaction with Visual C++, Visual Basic and other .Net enabled languages.
- ✓ Explicit support for unsafe code.
- ✓ Boxing and Unboxing
- ✓ Boolean Conditions
- ✓ Automatic Garbage Collection
- ✓ Standard Library
- ✓ Assembly Versioning
- ✓ Properties and Events
- ✓ Delegates and Events Management
- ✓ Indexers
- ✓ Simple Multithreading
- ✓ Integration with Windows

Characteristics of C#

The main design goal of C# was simplicity rather than pure power. C# fulfils the need for a language that is easy to write, read and maintain and also provides the power and flexibility of C++. The language that is designed for both computing and communications is characterized by several key features.

It is

- ✓ Simple
- ✓ Object-oriented
- ✓ Compatible
- ✓ Consistent
- ✓ Type-safe
- ✓ Interoperable and
- ✓ Modern
- ✓ Versionable
- ✓ Flexible

Simple

C# simplifies C++ by eliminating irksome operators such as `->`, `::` and pointers. C# treats integer and Boolean data types as two entirely different types. This means that the use of `=` in place of `==` in If statements will be caught by the compiler.

Consistent

C# supports an unified type system which eliminates the problem of varying ranges of integer types. All types are treated as objects and developers can extend the type system simply and easily.

Modern

C# is called a modern language due to a number of features it supports. It supports

- ✓ Automatic garbage collection
- ✓ Modern approach to debugging and
- ✓ Rich intrinsic model for error handling
- ✓ Robust security model
- ✓ Decimal data type for financial applications

Object-Oriented

C# is truly object-oriented. It supports all the three tenets of object-oriented systems, namely, • Encapsulation • Inheritance • Polymorphism The entire C# class model is built on top of the Virtual Object System (VOS) of the NET Framework In C#, everything is an object. There are no more global functions, variables and constants.

Type-safe

Type-safety promotes robust programs. C# incorporates a number of type-safe measures.

- ✓ All dynamically allocated objects and arrays are initialized to zero
- ✓ Use of any uninitialized variables produces an error message by the compiler
- ✓ Access to arrays are range-checked and warned if it goes out-of-bounds
- ✓ C# does not permit unsafe casts
- ✓ C# enforces overflow checking in arithmetic operations
- ✓ Reference parameters that are passed are type-safe
- ✓ C# supports automatic garbage collection

Versionable

Making new versions of software modules work with the existing applications is known as versioning. C# provides support for versioning with the help of new and override

keywords. With this support, a programmer can guarantee that his new class library will maintain binary compatibility with the existing client applications.

Flexible

Although C# does not support pointers, we may declare certain classes and methods as 'unsafe' and then use pointers to manipulate them. However, these codes will not be type-safe.

Inter-operability

C# provides support for using COM objects, no matter what language was used to author them. C# also supports a special feature that enables a program to call out any native API.

HOW DOES C# DIFFER FROM C++?

As stated earlier. C# was derived from C++ to make it the language of choice for C and C++ programmers. C#, therefore, shares major parts of syntax with C++. However, the C# designers introduced a few changes in the syntax of C++ and removed a few features primarily to reduce the common pitfalls that occurred in C++ program development. They also added a number of additional features to make C# a type-safe and web-enabled language.

Changes Introduced

- ✓ C# compiles straight from source code to executable code, with no object files.
- ✓ C# does not separate class definition from implementation. Classes are defined and implemented in the same place and therefore there is no need for header files.
- ✓ C# does not support #include statement. (Note that using is not the same as #include). All data types in C# are inherited from the object superclass and therefore they are objects.
- ✓ All the basic value types will have the same size on any system. This is not the case in C or C++. Thus C# is more suitable for writing distributed applications.
- ✓ In C#, data types belong to either value types or reference types.

- ✓ C# checks for uninitialized variables and gives error messages at compile time. In C++, an uninitialized variable goes undetected thus resulting in unpredictable output.
- ✓ In C#, structs are value types.
- ✓ C# supports a native string type. Manipulation of strings is easy.
- ✓ C# supports a native Boolean data type and bool-type data cannot be implicitly or explicitly cast to any data type except object.
- ✓ C# declares null as a keyword and considers it as an intrinsic value.
- ✓ C# does not support pointer types for manipulating data. However, they are used in what is known as 'unsafe' code.
- ✓ Variable scope rules in C# are more restrictive. In C#, duplicating the same name within a routine is illegal, even if it is in a separate code block.
- ✓ C# permits declaration of variables between goto and label.
- ✓ We can only create objects in C# using the new keyword.
- ✓ Arrays are classes in C# and therefore they have built-in functionality for operations such as sorting, searching, and reversing.
- ✓ Arrays in C# are declared differently and behave very differently compared to C++ arrays.
- ✓ C# provides special syntax to initialize arrays efficiently.
- ✓ Arrays in C# are always reference types rather than value types, as they are in C++ and therefore stored in a heap.
- ✓ In C#, expressions in if and while statements must resolve to a bool value. Accidental use of the assignment operator (=) instead of equality operator == will be caught by the compiler.
- ✓ C# supports four iteration statements rather than three in C++ . The fourth one is the foreach statement.
- ✓ C# does not allow silent fall-through in switch statements. It requires an explicit jump statement at the end of each case statement.
- ✓ In C#, switch can also be used on string values.
- ✓ The set of operators that can be overloaded in C# is smaller compared to C++.
- ✓ C# can check overflow of arithmetic operations and conversions using checked and unchecked keywords.
- ✓ C# does not support default arguments.
- ✓ Variable method parameters are handled differently in C#.

- ✓ In exception-handling, unlike in C++, we cannot throw any type in C#. The thrown value has to be a reference to a derived class or System.Exception object.
- ✓ C# requires ordering of catch blocks correctly.
- ✓ General catch statement catch (...) in C++ is replaced by simple catch in C#
- ✓ C# does not provide any defaults for constructors.
- ✓ Destructors in C# behave differently than in C++.
- ✓ In C#, we cannot access static members via an object, as we can in C++.
- ✓ C# does not support multiple code inheritance.
- ✓ Casting in C# is much safer than in C++.
- ✓ When overriding a virtual method, we must use the override keyword.
- ✓ Abstract methods in C# are similar to virtual functions in C++, but C# abstract methods cannot have implementations.
- ✓ Command-line parameters array behave differently in C# as compared to C++.

C++ features dropped

The following C++ features are missing from C#:

- ✓ Macros
- ✓ Multiple inheritance
- ✓ Templates
- ✓ Pointers
- ✓ Global variables
- ✓ Typedef statement
- ✓ Default arguments
- ✓ Constant member functions or parameters.

Enhancements to C++

C# modernizes C++ by adding the following new features:

- ✓ Automatic garbage collection
- ✓ Versioning support
- ✓ Strict type-safety
- ✓ Properties to access data members
- ✓ Delegates and events
- ✓ Boxing and unboxing

- ✓ Web services

HOW DOES C# DIFFER FROM JAVA?

Like C#, Java was also derived from C++ and therefore they have similar roots. Moreover, C# was developed by Microsoft as an alternative to Java for web programming. C# has borrowed many good features from Java, which has already become a popular Internet language. However, there exist a number of differences between C# and Java:

- ✓ Although C# uses .NET runtime that is similar to Java runtime, the C# compiler produces an executable code.
- ✓ C# has more primitive data types.
- ✓ Unlike Java, all C# data types are objects.
- ✓ Arrays are declared differently in C#.
- ✓ Although C# classes are quite similar to Java classes, there are a few important differences relating to constants, base classes and constructors, static constructors, versioning, accessibility of members etc.
- ✓ Java uses static final to declare a class constant while C# uses const.
- ✓ The convention for Java is to put one public class in each file and in fact, some compilers require this. C# allows any source file arrangement.
- ✓ C# supports the struct type and Java does not.
- ✓ Java does not provide for operator overloading.
- ✓ In Java, class members are virtual by default and a method having the same name in a derived class overrides the base class member. In C#, the base member is required to have the virtual keyword and the derived member is required to use the override keyword.
- ✓ The new modifier used for class members has no complement in Java.
- ✓ C# provides better versioning support than Java.
- ✓ C# provides static constructors for initialization.
- ✓ C# provides built-in delegates and events. Java uses interfaces and inner classes to achieve a similar result.
- ✓ In Java, parameters are always passed by value. C# allows parameters to be passed by reference by using the ref keyword.

- ✓ C# adds internal, a new accessibility modifier. Members with internal accessibility can be accessed from other classes within the same project, but not from outside the project.
- ✓ C# includes native support for properties, Java does not.
- ✓ Java does not directly support enumerations.
- ✓ Java does not have any equivalent to C# indexers.
- ✓ Both Java and C# support interfaces. But, C# does not allow type definitions in interfaces, while Java interfaces can have const type data.
- ✓ In Java, the switch statement can have only integer expression, while C# supports either an integer or string expressions.
- ✓ C# does not allow free fall_through from case to case.
- ✓ C# provides a fourth type of iteration statement, foreach for quick and easy iterations over collections and array type data.
- ✓ Catch blocks should be ordered correctly in C#.
- ✓ C# checks overflows using checked statements.
- ✓ C# uses is operator instead of instanceof operator in Java.
- ✓ C# allows a variable number of parameters using the params keyword.
- ✓ There is no labeled break statement in C#. The goto is used to achieve this.

The .Net Framework

.Net Framework is a software development platform developed by Microsoft for building and running Windows applications. The .Net framework consists of developer tools, programming languages, and libraries to build desktop and web applications. It is also used to build websites, web services, and games. The .Net framework applications are multi-platform applications. The framework has been designed in such a way that it can be used from any of the following languages: C#, C++, Visual Basic, Jscript, COBOL, etc. All these languages can access the framework as well as communicate with each other.

The .Net framework consists of an enormous library of codes used by the client languages such as C#. Following is some of the components of the .Net framework –

- ✓ Common Language Runtime (CLR)
- ✓ The .Net Framework Class Library
- ✓ Common Language Specification
- ✓ Common Type System
- ✓ Metadata and Assemblies
- ✓ Windows Forms

- ✓ ASP.Net and ASP.Net AJAX
- ✓ ADO.Net
- ✓ Windows Workflow Foundation (WF)
- ✓ Windows Presentation Foundation
- ✓ Windows Communication Foundation (WCF)
- ✓ LINQ

Common Language Runtime (CLR)

The Common Language Runtime (CLR) is programming that manages the execution of programs written in any of several supported languages, allowing them to share common object-oriented classes written in any of the languages. It is a part of Microsoft's .NET Framework. Microsoft refers to its CLR as a "managed execution environment." A program compiled for the CLR does not need a language-specific execution environment and can easily be moved to and run on any system with Windows 2000 or Windows XP.

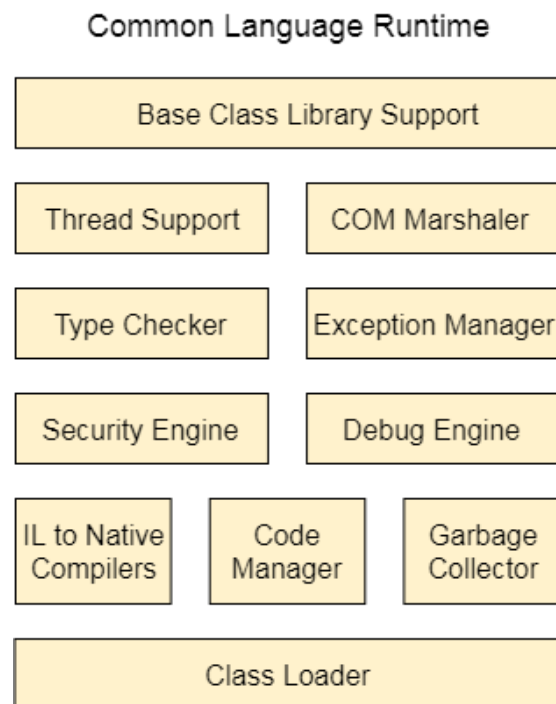
Programmers writing in Visual Basic, Visual C++, or C# compile their programs into an intermediate form of code called Common Intermediate Language (CIL) in a portable execution file that can then be managed and executed by the CLR. The programmer and the environment specify descriptive information about the program when it is compiled and the information is stored with the compiled program as metadata. Metadata, stored in the compiled program, tells the CLR what language was used, its version and what class libraries will be needed by the program. The CLR allows an instance of a class written in one language to call a method of a class written in another language. It also provides garbage collecting (returning unneeded memory to the computer), exception handling and debugging services.

Following are the functions of the CLR.

- ✓ It converts the program into native code.
- ✓ Handles Exceptions
- ✓ Provides type-safety
- ✓ Memory management
- ✓ Provides security
- ✓ Improved performance
- ✓ Language independent
- ✓ Platform independent
- ✓ Garbage collection
- ✓ Provides language features such as inheritance, interfaces, and overloading for object-oriented programming.

[.NET CLR Structure](#)

Following is the component structure of Common Language Runtime.



Base Class Library Support: It is a class library that provides support of classes to the .NET application.

Thread Support: It manages the parallel execution of the multi-threaded application.

COM Marshaler: It provides communication between the COM objects and the application.

Type Checker: It checks types used in the application and verifies that they match to the standards provided by the CLR.

Code Manager: It manages code at execution run-time.

Garbage Collector: It releases the unused memory and allocates it to a new application.

Exception Handler: It handles the exception at runtime to avoid application failure.

ClassLoader: It is used to load all classes at run time.

[.NET Framework Class Library](#)

.NET Framework Class Library is the collection of classes, namespaces, interfaces and value types that are used for .NET applications.

It contains thousands of classes that supports the following functions.

- ✓ Base and user-defined data types
- ✓ Support for exceptions handling
- ✓ Input/output and stream operations
- ✓ Communications with the underlying system
- ✓ Access to data
- ✓ Ability to create Windows-based GUI applications
- ✓ Ability to create web-client and server applications
- ✓ Support for creating web services

.NET Framework Class Library Namespaces

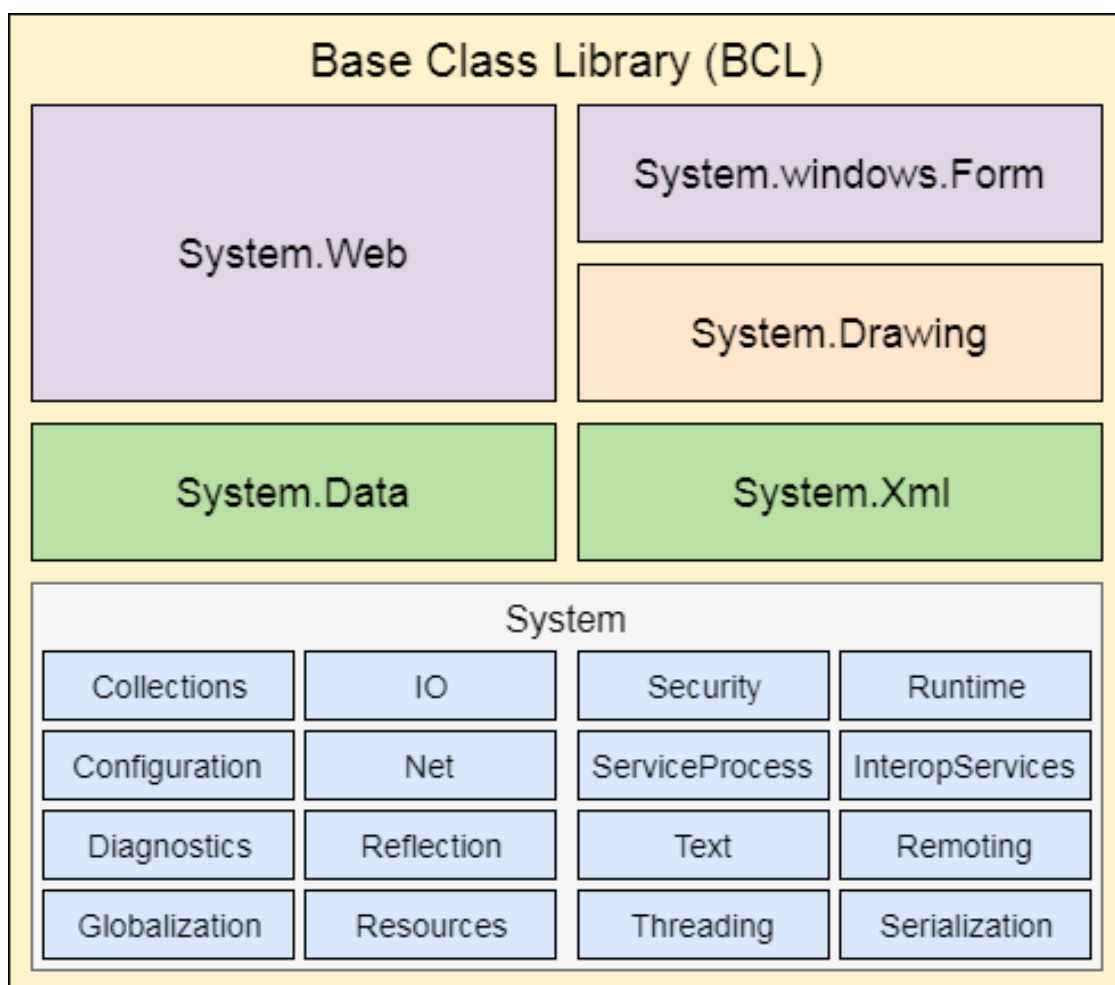
Following is the commonly used namespaces that contains useful classes and interfaces and defined in Framework Class Library.

Namespaces	Description
System	It includes all common datatypes, string values, arrays and methods for data conversion.
System.Data, System.Data.Common, System.Data.OleDb, System.Data.SqlClient, System.Data.SqlTypes	These are used to access a database, perform commands on a database and retrieve database.
System.IO, System.DirectoryServices, System.IO.IsolatedStorage	These are used to access, read and write files.
System.Diagnostics	It is used to debug and trace the execution of an application.
System.Net, System.Net.Sockets	These are used to communicate over the Internet when creating peer-to-peer applications.
System.Windows.Forms, System.Windows.Forms.Design	These namespaces are used to create Windows-based applications using Windows user interface components.
System.Web, System.WebCaching, System.Web.UI, System.Web.UI.Design, System.Web.UI.WebControls, System.Web.UI.HtmlControls, System.Web.Configuration, System.Web.Hosting, System.Web.Mail, System.Web.SessionState	These are used to create ASP.NET Web applications that run over the web.

System.Web.Services, System.Web.Services.Description, System.Web.Services.Configuration, System.Web.Services.Discovery, System.Web.Services.Protocols	These are used to create XML Web services and components that can be published over the web.
System.Security, System.Security.Permissions, System.Security.Policy, System.WebSecurity, System.Security.Cryptography	These are used for authentication, authorization, and encryption purpose.
System.Xml, System.Xml.Schema, System.Xml.Serialization, System.Xml.XPath, System.Xml.Xsl	These namespaces are used to create and access XML files.

[.NET Framework Base Class Library](#)

.NET Base Class Library is the sub part of the Framework that provides library support to Common Language Runtime to work properly. It includes the System namespace and core types of the .NET framework.



Common Language Specification

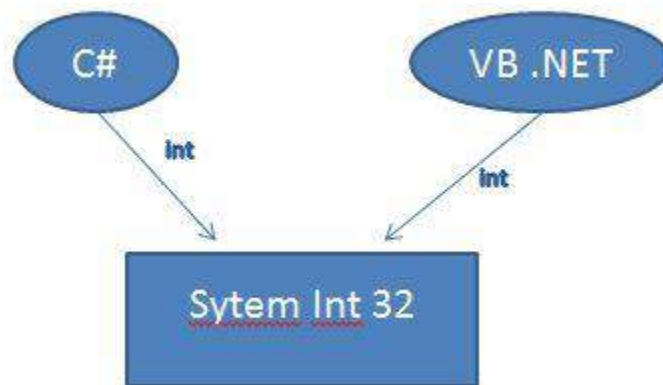
A Common Language Specification (CLS) is a document that says how computer programs can be turned into Common Intermediate Language (CIL) code. When several languages use the same bytecode, different parts of a program can be written in different languages. Microsoft uses a Common Language Specification for their .NET Framework. To fully interact with other objects regardless of the language they were used in, objects must expose to callers only those features that are common to all the languages they must exchange information with. It was always a dream of Microsoft to unite all different languages into one umbrella and CLS is one step towards that. Microsoft has defined CLS which are nothing but guidelines for languages to follow so that it can communicate with other .NET languages in a seamless manner.

Most of the members defined by types in the .NET Framework class library are able to work with CLS. However, some types in the class library have one or more members that are not able to work with CLS. These members allow support for language features that are not in the CLS.

The CLS was designed to be large enough to include the language constructs that are commonly needed by developers, yet small enough that most languages are able to support it. Any language construct that makes it impossible to quickly confirm the type safety of code was excluded from the CLS so that all languages that can work with CLS can produce verifiable code if they choose to do so.

Common Type System (CTS)

The Common Type System (CTS) standardizes the data types of all programming languages using .NET under the umbrella of .NET to a common data type for easy and smooth communication among these .NET languages. For example, when we declare an int type data type in C# and VB.Net then they are converted to int32. In other words, now both will have a common data type that provides flexible communication between these two languages.



CTS is designed as a singly rooted object hierarchy with System.Object as the base type from which all other types are derived. CTS supports two different kinds of types:

1. Value Types: Contain the values that need to be stored directly on the stack or allocated inline in a structure. They can be built-in (standard primitive types), user-defined (defined in source code) or enumerations (sets of enumerated values that are represented by labels but stored as a numeric type).
2. Reference Types: Store a reference to the value's memory address and are allocated on the heap. Reference types can be any of the pointer types, interface types or self-describing types (arrays and class types such as user-defined classes, boxed value types and delegates).

Although operations on variables of a value type do not affect any other variable, operations on variables of a reference type can affect the same object referred to by another variable. When references are made within the scope of an assembly, two types with the same name but in different assemblies are defined as two distinct types, whereas when using namespaces, the run time recognizes the full name of each type (such as System.Object, System.String, etc.). The rich set of types in CTS has well-designed semantics such that they can be widely used as a base type in Common Language Runtime (CLR) -based languages.

C# Version History

C# was first introduced with .NET Framework 1.0 in the year 2002 and evolved much since then. The following table lists important features introduced in each version of C#:

Version	.NET Framework	Visual Studio	Important Features
C# 1.0	.NET Framework 1.0/1.1	Visual Studio .NET 2002	<ul style="list-style-type: none"> • Basic features
C# 2.0	.NET Framework 2.0	Visual Studio 2005	<ul style="list-style-type: none"> • Generics • Partial types • Anonymous methods

Version	.NET Framework	Visual Studio	Important Features
			<ul style="list-style-type: none"> Iterators Nullable types Private setters (properties) Method group conversions (delegates) Covariance and Contra-variance Static classes
C# 3.0	.NET Framework 3.0\3.5	Visual Studio 2008	<ul style="list-style-type: none"> Implicitly typed local variables Object and collection initializers Auto-Implemented properties Anonymous types Extension methods Query expressions Lambda expressions Expression trees Partial Methods
C# 4.0	.NET Framework 4.0	Visual Studio 2010	<ul style="list-style-type: none"> Dynamic binding (late binding) Named and optional arguments Generic co- and contravariance Embedded interop types
C# 5.0	.NET Framework 4.5	Visual Studio 2012/2013	<ul style="list-style-type: none"> Async features Caller information
C# 6.0	.NET Framework 4.6	Visual Studio 2013/2015	<ul style="list-style-type: none"> Expression Bodied Methods Auto-property initializer nameof Expression Primary constructor Await in catch block Exception Filter String Interpolation
C# 7.0	.NET Core 2.0	Visual Studio 2017	<ul style="list-style-type: none"> out variables Tuples Discards Pattern Matching Local functions Generalized async return types
C# 8.0	.NET Core 3.0	Visual Studio 2019	<ul style="list-style-type: none"> Readonly members Default interface methods Using declarations Static local functions

Version	.NET Framework	Visual Studio	Important Features
			<ul style="list-style-type: none"> • Disposable ref structs • Nullable reference types

WinForms

Windows Forms is a smart client technology for the .NET Framework, a set of managed libraries that simplify common application tasks such as reading and writing to the file system.

ASP.NET

ASP.NET is a web framework designed and developed by Microsoft. It is used to develop websites, web applications, and web services. It provides a fantastic integration of HTML, CSS, and JavaScript. It was first released in January 2002.

ADO.NET

ADO.NET is a module of .Net Framework, which is used to establish a connection between application and data sources. Data sources can be such as SQL Server and XML. ADO .NET consists of classes that can be used to connect, retrieve, insert, and delete data.

WPF (Windows Presentation Foundation)

Windows Presentation Foundation (WPF) is a graphical subsystem by Microsoft for rendering user interfaces in Windows-based applications. WPF, previously known as "Avalon", was initially released as part of .NET Framework 3.0 in 2006. WPF uses DirectX.

WCF (Windows Communication Foundation)

It is a framework for building service-oriented applications. Using WCF, you can send data as asynchronous messages from one service endpoint to another.

WF (Workflow Foundation)

Windows Workflow Foundation (WF) is a Microsoft technology that provides an API, an in-process workflow engine, and a rehostable designer to implement long-running processes as workflows within .NET applications.

LINQ (Language Integrated Query)

It is a query language, introduced in .NET 3.5 framework. It is used to make the query for data sources with C# or Visual Basics programming languages.

Entity Framework

It is an ORM based open-source framework which is used to work with a database using .NET objects. It eliminates a lot of developer's effort to handle the database. It is Microsoft's recommended technology to deal with the database.

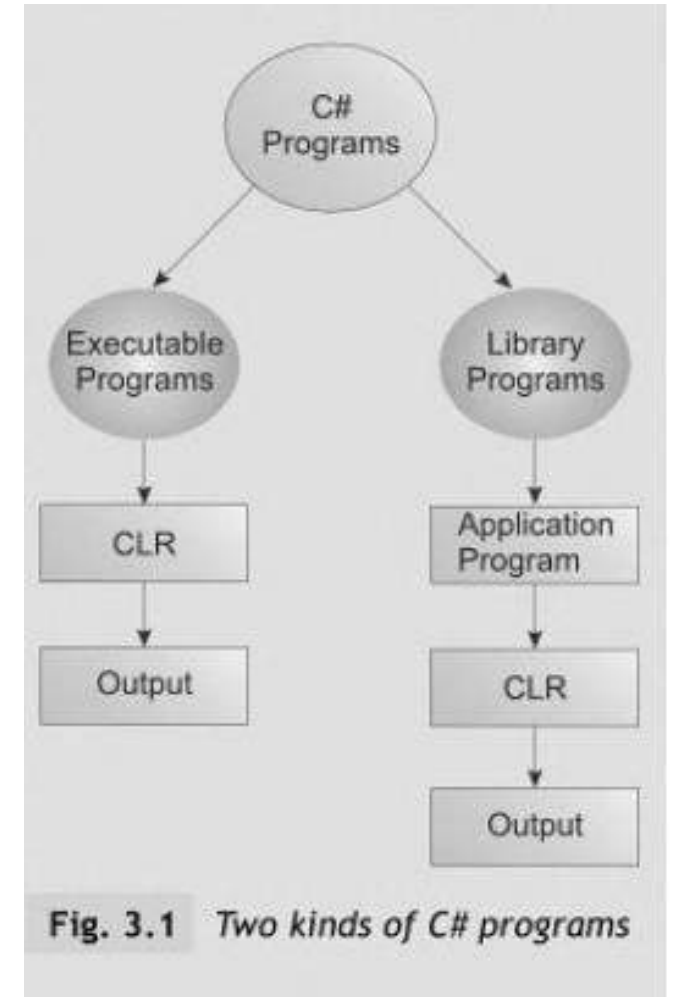
OVERVIEW OF C#

Dr P.V. Praveen Sundar
Assistant Professor,
Department of Computer Science
Adhiparasakthi College of Arts & Science,
Kalavai.

Overview of C#

2

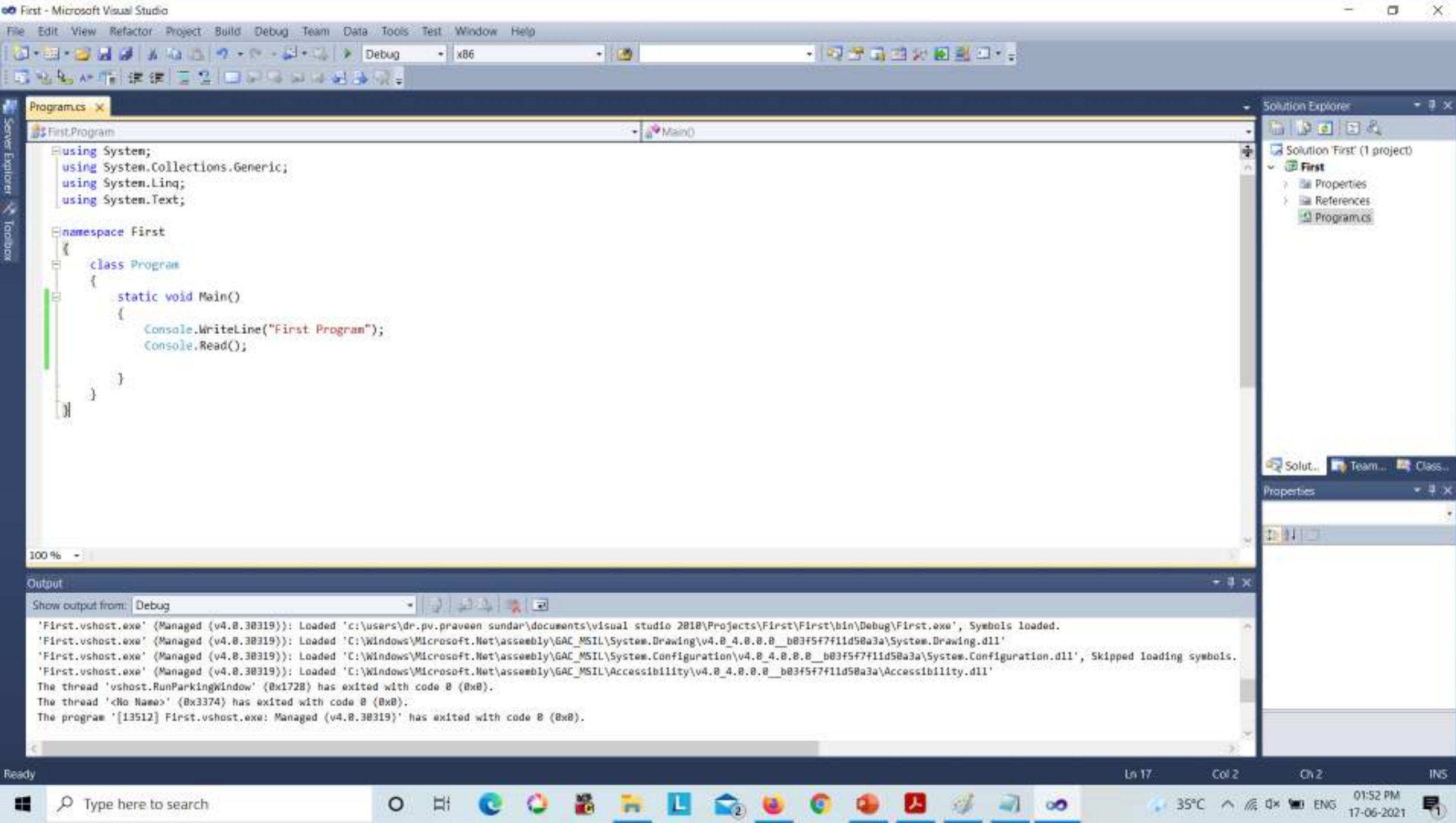
- C# can be used to develop two categories of programs, namely,
 - ✓ Executable application programs and
 - ✓ Component libraries as illustrated in Fig. 3.1.
- Executable programs are written to carry out certain tasks and require the method `Main` in one of the classes.
- In contrast, component libraries do not require a `Main` declaration because they are not standalone application programs. They are written for use by other applications.
- This concept is something similar to applets and application programs in Java.



Main()

3

- ❑ Every C# executable program must include the Main() method in one of the classes. This is the 'starting point' for executing the program.
- ❑ A C# application can have any number of classes but 'only one' class can have the Main method to initiate the execution.
- ❑ Main() contains a number of keywords: public, static and void.
- ❑ **public** : The keyword public is an access modifier that tells the C# compiler that the Main method is accessible by anyone
- ❑ **static**: The keyword static declares that the Main method is a global one and can be called without creating an instance of the class. The compiler stores the address of the method as the entry point and uses this information to begin execution before any objects are created.
- ❑ **void**: The keyword void is a type modifier that states that the Main method does not return any value.



Creating and Running C# Program

5

- Generally, the programs created using programming languages like C, C++, Java, C# etc., are written using a high-level language like English. But, the computer cannot understand the high-level language. It can understand only low-level language. So, the program written in the high-level language needs to be converted into the low-level language to make it understandable for the computer. This conversion is performed using either Interpreter or Compiler.
- Popular programming languages like C, C++, Java, C# etc., use the compiler to convert high-level language instructions into low-level language instructions.
- A compiler is a program that converts high-level language instructions into low-level language instructions. Generally, the compiler performs two things, first it verifies the program errors, if errors are found, it returns a list of errors otherwise it converts the complete code into the low-level language.

Stages of Compilation

6

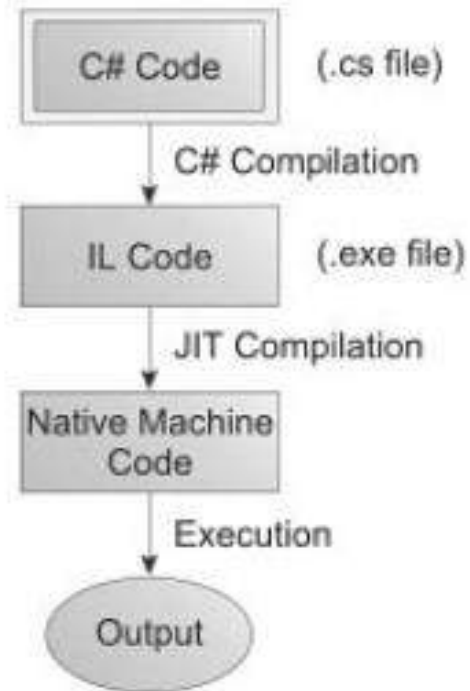


Fig. 3.2 *Three stages of code transformation*

Namespace

7

- ❑ A namespace is a declarative region that provides a scope to the identifiers (the names of types, functions, variables, etc) inside it.
- ❑ Namespaces are used to organize code into logical groups and to prevent name collisions that can occur especially when your code base includes multiple libraries.
- ❑ The namespaces in C# can be nested. That means one namespace can contain other namespaces also.
- ❑ The .NET framework already contains number of standard namespaces like System, System.Net, System.IO etc. In addition to these standard namespaces the user can define their own namespaces.



```
Sample.cs
example.Sample
test0

namespace MainClass
{
    class Sample
    {
        public void test()
        {
            System.Console.WriteLine("First Class");
        }
    }
}

namespace example
{
    class Sample
    {
        public void test()
        {
            System.Console.WriteLine("Second Class");
        }
    }

    class cname
    {
        MainClass.Sample S = new MainClass.Sample();
        example.Sample D = new example.Sample ();
        void mod1()
        {
            S.test();
            D.test();
        }
        static void Main()
        {
            cname c = new cname();
            c.mod1();
            System.Console.Read();
        }
    }
}
```

Solution Explorer

- Solution 'Second' (1 project)
 - Second
 - Properties
 - References
 - Sample.cs

Properties

Solut... Team... Class...

Comments

9

- ❑ Comments are used to provide a small description of the program.
- ❑ The main reason for developers to write comments is to clarify or explain some part of the code.
- ❑ The comment lines are simply ignored by the compiler, that means they are not executed.
- ❑ In C#, there are two types of comments.
 - ❑ **Single Line Comments:** Single line comment begins with `//` symbol. We can write any number of single line comments.
 - ❑ **Multiple Lines Comments:** Multiple lines comment begins with `/*` symbol and ends with `*/`. We can write any number of multiple lines comments in a program.

3.5 MAIN RETURNING A VALUE

Another important aspect is the return type of **Main()**. We have used **void** as the return type in earlier programs. **Main()** can also return a value if it is declared as **int** type instead of **void**. When the return type is **int**, we must include a **return** statement at the end of the method as shown in Program 3.3.

Program 3.3 | MAIN RETURNING A VALUE

```
// Main returning a value
using System;
class SampleThree
{
    public static int Main( )
    {
        Console.WriteLine ("Hello!");
        return 0; // Return statement
    }
}
```

Program 3.3 returns an integer-type value to the system. The value returned serves as the program's *termination status code*. The purpose of this code is to allow communication of success or failure to the execution environment.

USING ALIASES FOR NAMESPACE CLASSES

11

3.6 ——— USING ALIASES FOR NAMESPACE CLASSES ———

We have seen how we can avoid the prefix **System** to the **Console** class by implementing the **using System;** statement. Can we use this approach to avoid prefixing of **System.Console** to the method **WriteLine ()**? The answer is “no.”

System is a namespace and **Console** is a class. The **using** directive can be applied only to namespaces and cannot be applied to classes. Therefore the statement

```
using System.Console;
```

is illegal. However, we can overcome this problem by using aliases for namespace classes. This takes the form:

```
using alias-name = class-name;
```

Program 3.4 illustrates the use of aliases for classes.



Program.cs

First.Program

Main()

```
using A= System.Console;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace First
{
    class Program
    {
        public void test()
        {
        }
        static void Main()
        {
            A.WriteLine("First Program");
            A.Read();
        }
    }
}
```

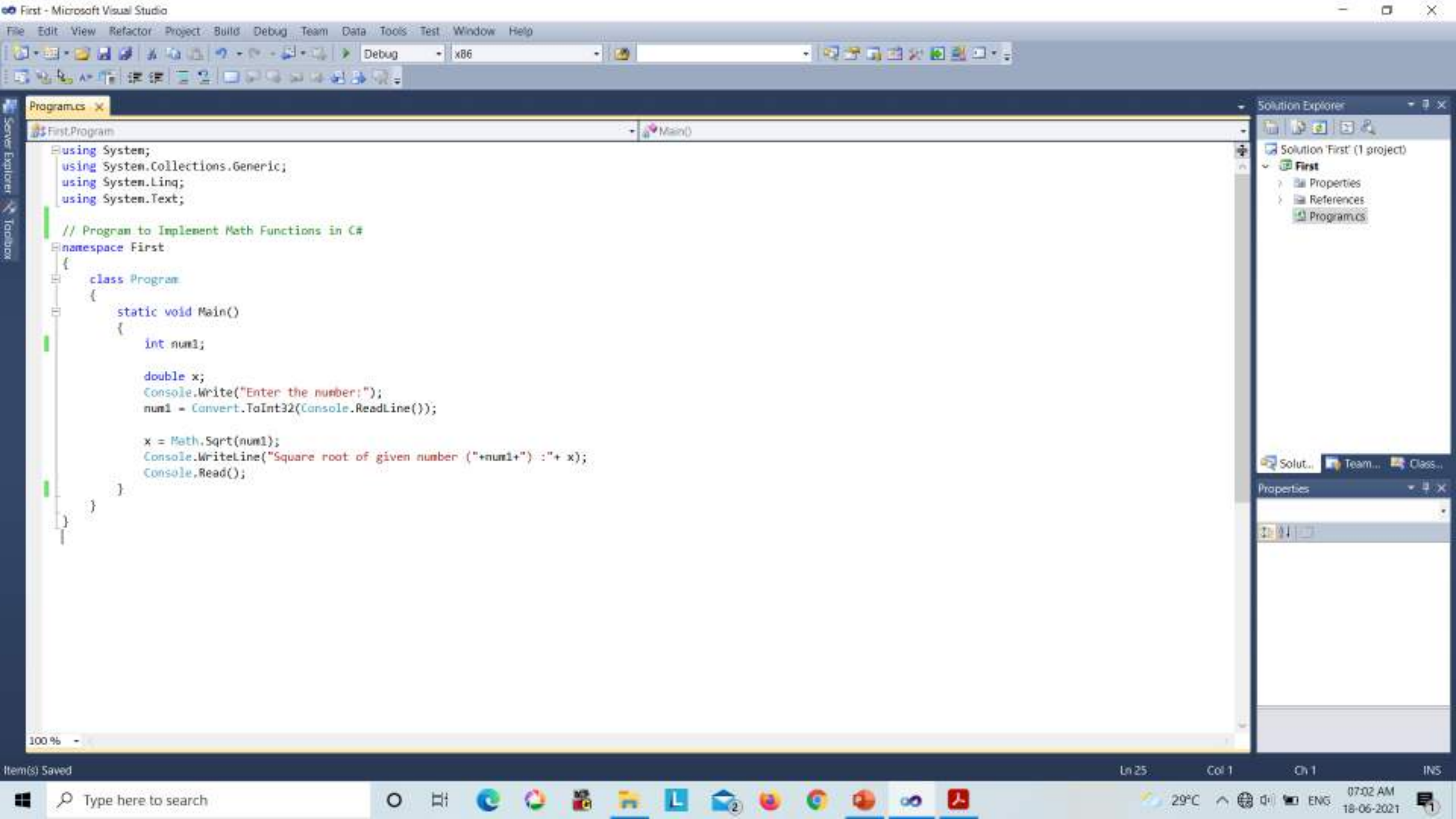
Solution Explorer

Solution 'First' (1 project)

- First
 - Properties
 - References
 - Program.cs

Solut... Team... Class...

Properties



Character Sets

14

- ❑ The set of characters used in a language is known as its **Character sets**.
- ❑ The characters of the character set must be represented in the memory of computers.
- ❑ The representation of a character in computer memory is known as **Character Encoding**.
- ❑ Different character encoding methods are available as standards.(ASCII, EBCDIC, ISO LATIN-1,Unicode)
- ❑ Unicode is a universal encoded character set that allows you to store information from any language using a single character set.
- ❑ It provides a unique code value for every character, regardless of the platform, program, or language.
- ❑ The character set for C# is taken from the Unicode encoding standard.
- ❑ C# Character sets supports Unicode Transformation Format -16 (UTF-16) encoding standard.

□ A subset of Character set of C#.

Upper Case Alphabets	A-Z
Lower case Alphabets	a-z
Decimal Digits	0-9
Special Characters	+ - * / % = < > _ Blank : ; , . ' " ? ! # \ () [] { } & ^ ~

These are Control Characters

ASCII Value	Character	Meaning
0	NULL	null
1	SOH	Start of header
2	STX	start of text
3	ETX	end of text
4	EOT	end of transaction
5	ENQ	enquiry
6	ACK	acknowledgement
7	BEL	bell
8	BS	back Space
9	HT	Horizontal Tab
10	LF	Line Feed
11	VT	Vertical Tab
12	FF	Form Feed
13	CR	Carriage Return
14	SO	Shift Out
15	SI	Shift In
16	DLE	Data Link Escape
17	DC1	Device Control 1
18	DC2	Device Control 2
19	DC3	Device Control 3
20	DC4	Device Control 4
21	NAK	Negative Acknowledgement
22	SYN	Synchronous Idle
23	ETB	End of Trans Block
24	CAN	Cancel
25	EM	End of Medium
26	SUB	Substitute
27	ESC	Escape
28	FS	File Separator
29	GS	Group Separator
30	RS	Record Separator
31	US	Unit Separator

These are Printable Characters

ASCII Value	Character	ASCII Value	Character	ASCII Value	Character
32	Space	64	@	96	`
33	!	65	A	97	a
34	"	66	B	98	b
35	#	67	C	99	c
36	\$	68	D	100	d
37	%	69	E	101	e
38	&	70	F	102	f
39	'	71	G	103	g
40	(72	H	104	h
41)	73	I	105	i
42	*	74	J	106	j
43	+	75	K	107	k
44	,	76	L	108	l
45	-	77	M	109	m
46	.	78	N	110	n
47	/	79	O	111	o
48	0	80	P	112	p
49	1	81	Q	113	q
50	2	82	R	114	r
51	3	83	S	115	s
52	4	84	T	116	t
53	5	85	U	117	u
54	6	86	V	118	v
55	7	87	W	119	w
56	8	88	X	120	x
57	9	89	Y	121	y
58	:	90	Z	122	z
59	;	91	[123	{
60	<	92	\	124	
61	=	93]	125	}
62	>	94	^	126	~
63	?	95	_	127	DEL

Tokens

17

- Token is an individual entity of a program. A compiler identifies and splits a program into a number of tokens. A token may be a single character or a group of characters which has a specific meaning according to its syntax. The following are the tokens can be identified by a C# Compiler during the translation process.
 - ▣ Identifiers
 - ▣ Keywords
 - ▣ Constants or Literals
 - ▣ Operators
 - ▣ Punctuators.
- For example, Consider the C# statement (if income >= 10000)
- where if is a keyword Followed by (operator, income is a variable followed by >= is an operator and finally 10000 is a numeric constant.

C# Tokens

18

- C# program is a collection of instructions and every instruction is a collection of some individual units. Every smallest individual unit of a C# program is called tokens. Tokens are used to construct C# programs and they are said to be the basic building blocks of a C# program.
- In a C# program tokens may contain the following...
 - ▣ Keywords
 - ▣ Identifiers
 - ▣ Operators
 - ▣ Punctuators
 - ▣ Special Symbols
 - ▣ Constants or Literals
 - ▣ Strings
 - ▣ Data values
- **In a C# program, a collection of all the keywords, identifiers, operators, special symbols, constants, strings, and data values are called tokens.**

C# Keywords

19

- As every language has words to construct statements, C# programming also has words with a specific meaning which are used to construct C# program instructions. In the C# programming language, keywords are special words with predefined meaning. Keywords are also known as reserved words in C# programming.
- In the C# programming language, there are **77 keywords**. All the 77 keywords have their meaning which is already known to the compiler.
- Keywords are the reserved words with predefined meaning which already known to the compiler
- Whenever C# compiler come across a keyword, automatically it understands its meaning.

□ Properties of Keywords

- ▣ All the keywords in C# programming language are defined as lowercase letters so they must be used only in lowercase letters
- ▣ Every keyword has a specific meaning, users can not change that meaning.
- ▣ Keywords can not be used as user-defined names like class, variable, functions, arrays, pointers, interface, namespace etc...
- ▣ Every keyword in C# programming language represents something or specifies some kind of action to be performed by the compiler.
- ▣ The following table specifies all the 77 keywords.

abstract
as
base
bool
break
byte
case
catch
char
checked
class
const
continue
decimal
default
delegate
do
double
else
enum

event
explicit
extern
false
finally
fixed
float
for
foreach
goto
if
implicit
in
int
interface
internal
is
lock
long

namespace
new
null
object
operator
out
override
params
private
protected
public
readonly
ref
return
sbyte
sealed
short
sizeof
stackalloc

static
string
struct
switch
this
throw
true
try
typeof
uint
ulong
unchecked
unsafe
ushort
using
virtual
void
volatile
while

C# Identifiers

22

- In C# programming, programmers can specify their name to a variable, array, pointer, function, class, interface, namespace etc... An identifier is a collection of characters which acts as the name of variable, function, array, pointer, structure, etc... In other words, an identifier can be defined as the user-defined name to identify an entity uniquely in the C# programming that name may be of the class name, variable name, function name, array name, pointer name, structure name, Interface name, namespace name or a label.
- The identifier is a user-defined name of an entity to identify it uniquely during the program execution.
- Example
int marks;
char studentName[30];
 - ▣ Here, marks and studentName are identifiers.

Rules for Creating Identifiers

23

- ❑ An identifier can contain letters (UPPERCASE and lowercase), numeric & underscore symbol only.
- ❑ An identifier should not start with a numerical value. It can start with a letter or an underscore.
- ❑ We should not use any special symbols in between the identifier even whitespace. However, the only underscore symbol is allowed.
- ❑ Keywords should not be used as identifiers.
- ❑ There is no limit for the length of an identifier. However, the compiler considers the first 31 characters only.
- ❑ An identifier must be unique in its scope.

Rules for Creating Identifiers for better programming

24

- ❑ The following are the commonly used rules for creating identifiers for better programming...
 - ❑ The identifier must be meaningful to describe the entity.
 - ❑ Since starting with an underscore may create conflict with system names, so we avoid starting an identifier with an underscore.
 - ❑ We start every identifier with a lowercase letter. If an identifier contains more than one word then the first word starts with a lowercase letter and second word onwards first letter is used as an UPPERCASE letter. We can also use an underscore to separate multiple words in an identifier.

Valid Identifiers

- ☐ `int a,b;`
- ☐ `float _a;`
- ☐ `char _123;`
- ☐ `double pi;`
- ☐ `int value,Value,vAlue;`
- ☐ `int Auto;`

Invalid Identifiers

- ☐ `int a b;`
- ☐ `float 123a;`
- ☐ `char str-;`
- ☐ `double pi, a;`
- ☐ `int break;`

Datatypes

26

- Data used in C# programming is classified into different types based on its properties. In the C# programming, a data type can be defined as a set of values with similar characteristics. All the values in a data type have the same properties.
- Data types in the C# programming are used to specify what kind of value can be stored in a variable. The memory size and type of the value of a variable are determined by the variable data type. In a C# programming, each variable or constant or array must have a data type and this data type specifies how much memory is to be allocated and what type of values are to be stored in that variable or constant or array. The formal definition of a data type is as follows...
- **The Data type is a set of value with predefined characteristics. data types are used to declare variable, constants, arrays, pointers, and functions.**

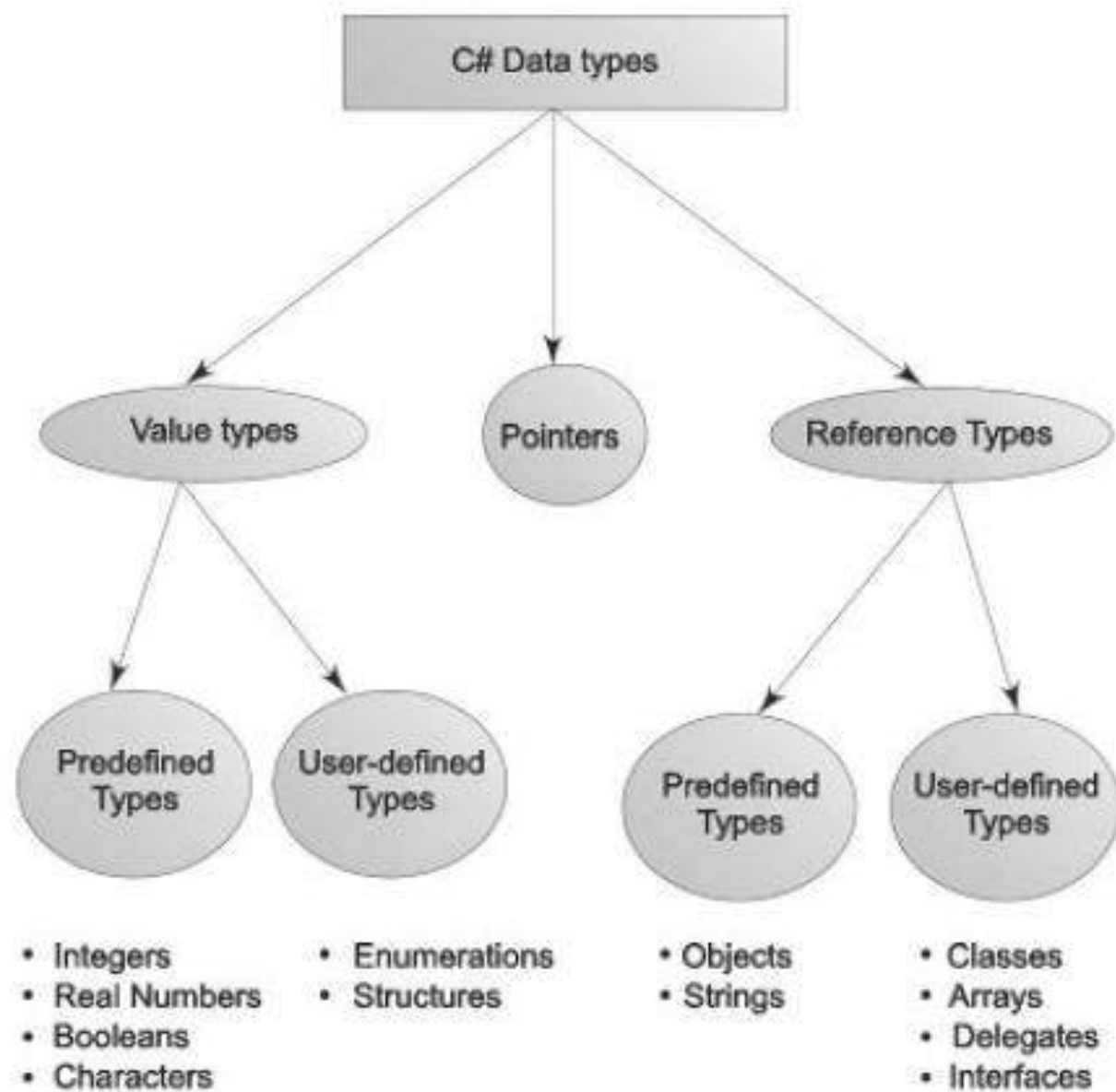
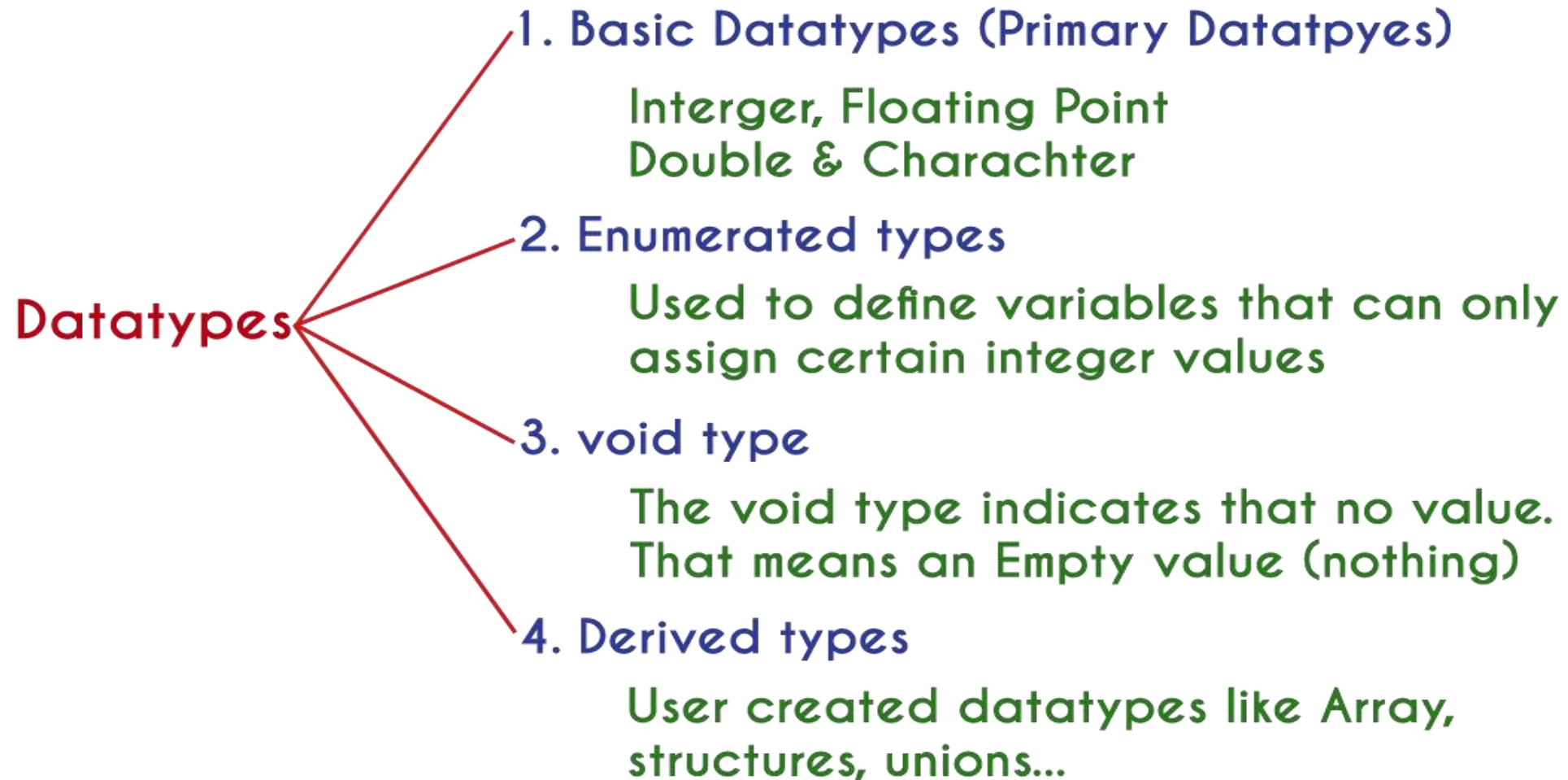


Fig. 4.2 *Taxonomy C# data types*



- In the C# programming , data types are classified as follows...
 - ▣ Value types (Basic data types or Predefined data types)
 - ▣ Pointers
 - ▣ Reference data types (Objects, Strings, Class, Arrays, Delegates, Interfaces)
 - ▣ Derived data types (Secondary data types OR User-defined data types)
 - ▣ Enumeration data types
 - ▣ Void data type
- **Primary data types**
- The primary data types in the C# programming are the basic data types. All the primary data types are already defined in the system. Primary data types are also called as Built-In data types. The following are the primary data types in C# programming ...
 - ▣ Integer data type
 - ▣ Floating Point data type
 - ▣ Double data type
 - ▣ Character data type

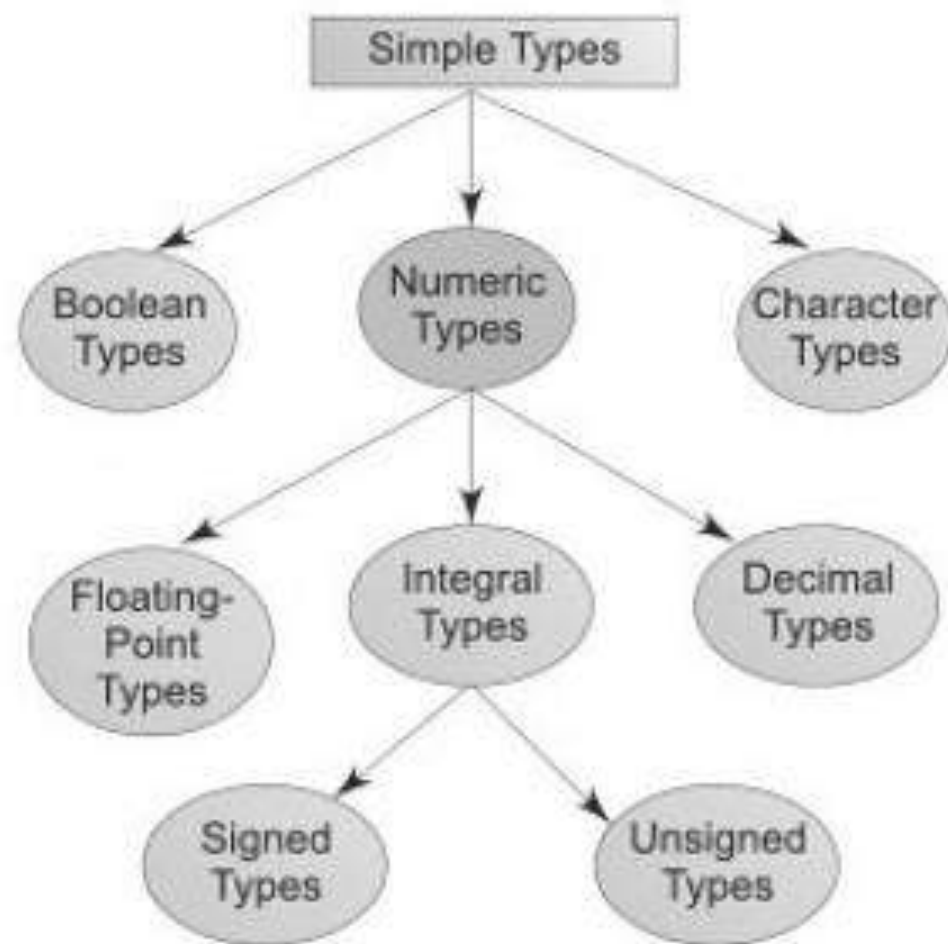


Fig. 4.3 *Categories of simple type data*

Integer Data type

31

- ❑ The integer data type is a set of whole numbers.
- ❑ Every integer value does not have the decimal value.
- ❑ We use the keyword "int" to represent integer data type in C# Programming.
- ❑ We use the keyword int to declare the variables and to specify the return type of a function.
- ❑ The integer data type is used with different type modifiers like short, long, signed and unsigned.
- ❑ The following figure and table provides complete details about the integer data type.

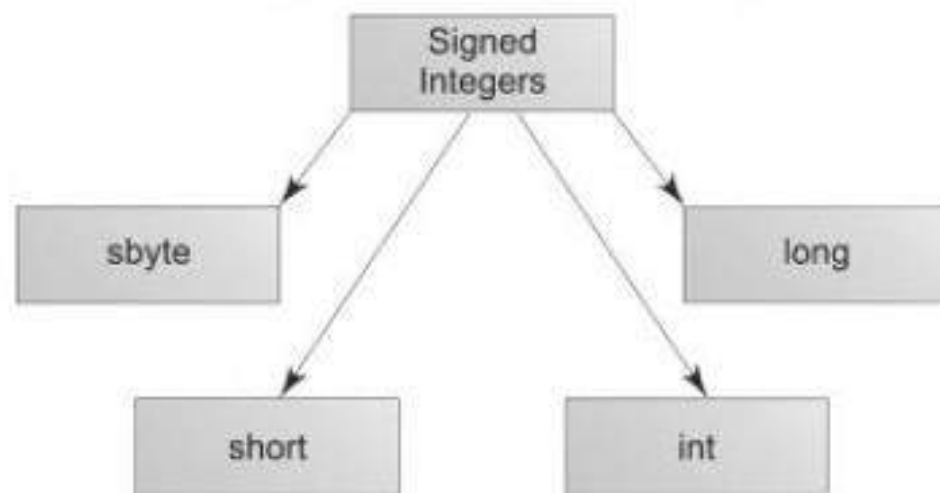


Fig. 4.4 *Signed integer types*

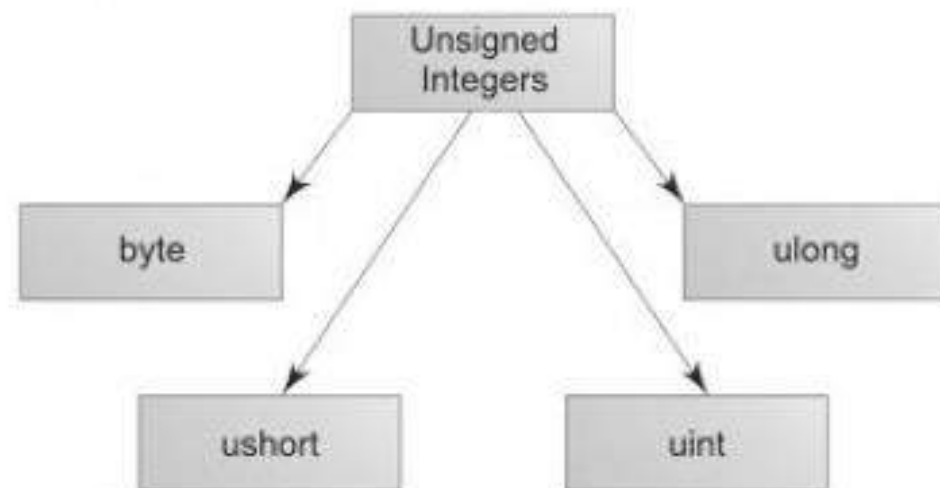


Fig. 4.5 *Unsigned integer types*

C# supports the following predefined integral types:

C# type/keyword	Range	Size	.NET type
sbyte	-128 to 127	Signed 8-bit integer	System.SByte
byte	0 to 255	Unsigned 8-bit integer	System.Byte
short	-32,768 to 32,767	Signed 16-bit integer	System.Int16
ushort	0 to 65,535	Unsigned 16-bit integer	System.UInt16
int	-2,147,483,648 to 2,147,483,647	Signed 32-bit integer	System.Int32
uint	0 to 4,294,967,295	Unsigned 32-bit integer	System.UInt32
long	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807	Signed 64-bit integer	System.Int64
ulong	0 to 18,446,744,073,709,551,615	Unsigned 64-bit integer	System.UInt64
nint	Depends on platform	Signed 32-bit or 64-bit integer	System.IntPtr
nuint	Depends on platform	Unsigned 32-bit or 64-bit integer	System.UIntPtr

Floating Point Data Types

34


- Floating-point data types are a set of numbers with the decimal value. Every floating-point value must contain the decimal value. The floating-point data type has two variants...
 - ▣ Float
 - ▣ Double
 - ▣ Decimal
- We use the keyword "float" to represent floating-point data type and "double" to represent double data type in c. Both float and double are similar but they differ in the number of decimal places. The float value contains 6 decimal places whereas double value contains 15 or 19 decimal places. The following table provides complete details about floating-point data types.

C# supports the following predefined floating-point types:

C# type/keyword	Approximate range	Precision	Size	.NET type
<code>float</code>	$\pm 1.5 \times 10^{-45}$ to $\pm 3.4 \times 10^{38}$	~6-9 digits	4 bytes	System.Single
<code>double</code>	$\pm 5.0 \times 10^{-324}$ to $\pm 1.7 \times 10^{308}$	~15-17 digits	8 bytes	System.Double
<code>decimal</code>	$\pm 1.0 \times 10^{-28}$ to $\pm 7.9228 \times 10^{28}$	28-29 digits	16 bytes	System.Decimal

In the preceding table, each C# type keyword from the leftmost column is an alias for the corresponding .NET type. They are interchangeable. For example, the following declarations declare variables of the same type:

C#

 Copy

```
double a = 12.3;  
System.Double b = 12.3;
```


Character Data Type

36

- The character data type is a set of characters enclosed in single quotations. The following table provides complete details about the character data type.

The `char` type keyword is an alias for the .NET `System.Char` structure type that represents a Unicode UTF-16 character.

Type	Range	Size	.NET type
<code>char</code>	U+0000 to U+FFFF	16 bit	<code>System.Char</code>

The default value of the `char` type is `\0`, that is, U+0000.

The `char` type supports `comparison`, `equality`, `increment`, and `decrement` operators. Moreover, for `char` operands, `arithmetic` and `bitwise logical` operators perform an operation on the corresponding character codes and produce the result of the `int` type.

The `string` type represents text as a sequence of `char` values.

void data type

- The void data type means nothing or no value. Generally, the void is used to specify a function which does not return any value. We also use the void data type to specify empty parameters of a function.

Enumerated data type

- An enumerated data type is a user-defined data type that consists of integer constants and each integer constant is given a name. The keyword "enum" is used to define the enumerated data type.

Derived data types

- Derived data types are user-defined data types. The derived data types are also called as user-defined data types or secondary data types. In the C# Programming, the derived data types are created using the following concepts...
 - ▣ Arrays
 - ▣ Structures
 - ▣ Unions
 - ▣ Enumeration

Enumeration types

38

- An enumeration type (or enum type) is a value type defined by a set of named constants of the underlying integral numeric type.
- To define an enumeration type, use the enum keyword and specify the names of enum members: for eg:

```
enum Season  
{  
    Spring,  
    Summer,  
    Autumn,  
    Winter  
}
```

- By default, the associated constant values of enum members are of type int; they start with zero and increase by one following the definition text order.
- It is possible that we can explicitly specify any other integral numeric type as an underlying type of an enumeration type.
- We can also explicitly specify the associated constant values, as the following example shows:

```
enum ErrorCode : ushort
{
    None = 0,
    Unknown = 1,
    ConnectionLost = 100,
    OutlierReading = 200
}
```

Reference types

40

- Variables of reference types store references to their data (objects), With reference types, two variables can reference the same object; therefore, operations on one variable can affect the object referenced by the other variable. With value types, each variable has its own copy of the data, and it is not possible for operations on one variable to affect the other.
- The following keywords are used to declare reference types:
 - ▣ Class
 - ▣ Interface
 - ▣ Delegate
 - ▣ Record
- C# also provides the following built-in reference types:
 - ▣ Dynamic
 - ▣ Object
 - ▣ String

Object

41

- ❑ The object type is an alias for `System.Object` in .NET.
- ❑ In the unified type system of C#, all types, predefined and user-defined, reference types and value types, inherit directly or indirectly from `System.Object`.
- ❑ It is possible to assign values of any type to variables of type object.
- ❑ Any object variable can be assigned to its default value using the literal `null`.
- ❑ When a variable of a value type is converted to object, it is said to be boxed. (Boxing)
- ❑ When a variable of type object is converted to a value type, it is said to be unboxed. (Unboxing)

Variables

42

- ❑ Variables in a C# Programming are the named memory locations where the user can store different values of the same datatype during the program execution. In other words, a variable can be defined as a storage container to hold values of the same datatype during the program execution.
- ❑ The formal definition of a variable is as follows...
- ❑ **Variable is a name given to a memory location where we can store different values of the same datatype during the program execution.**
- ❑ Every variable in C# Programming must be declared before it is used. Every variable must have a datatype that determines the range and type of values be stored and the size of the memory to be allocated.
- ❑ A variable name may contain letters, digits and underscore symbol. The following are the rules to specify a variable name...
 - ❑ Variable name should not start with a digit.
 - ❑ Keywords should not be used as variable names.
 - ❑ A variable name should not contain any special symbols except underscore(_).
 - ❑ A variable name can be of any length but compiler considers only the first 31 characters of the variable name.

Declaration of Variable

- ▣ Declaration of a variable tells the compiler to allocate the required amount of memory with the specified variable name and allows only specified datatype values into that memory location. In C# Programming, the declaration can be performed either before the function as global variables or inside any block or function. But it must be at the beginning of block or function.

Declaration Syntax:

`datatype variableName;`

Example

`int number;`

- ▣ The above declaration tells to the compiler that allocates **2 bytes** of memory with the name **number** and allows only integer values into that memory location.

Constants or Literals

44

- In C# programming, a constant is similar to the variable but the constant hold only one value during the program execution. That means, once a value is assigned to the constant, that value can't be changed during the program execution. Once the value is assigned to the constant, it is fixed throughout the program. A constant can be defined as follows...
- A constant is a named memory location which holds only one value throughout the program execution.
- In C# programming, a constant can be of any data type like integer, floating-point, character, string and double, etc.,

Literals

45

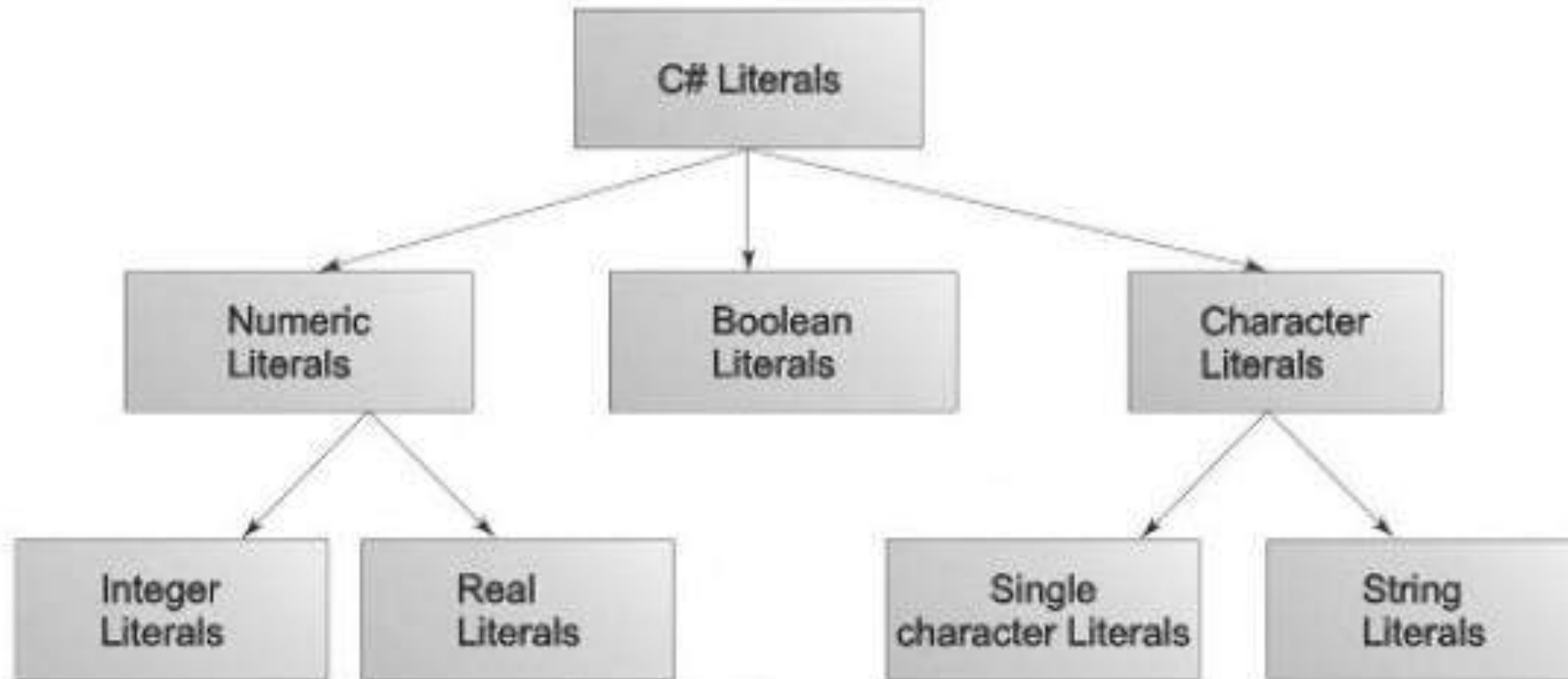


Fig. 4.1 *C# literals*

Integer constants

46

- ❑ An integer constant can be a decimal integer or hexadecimal integer. A decimal integer value is specified as direct integer value whereas hexadecimal value is prefixed with '0X'.
- ❑ If the literal is suffixed by U or u, its type is the first of the following types in which its value can be represented: uint, ulong.
- ❑ If the literal is suffixed by L or l, its type is the first of the following types in which its value can be represented: long, ulong.
- ❑ Example
 - ❑ 125 → Decimal Integer Constant
 - ❑ 0X3A → Hexa Decimal Integer Constant
 - ❑ 50u → Unsigned Integer Constant
 - ❑ 30l → Long Integer Constant
 - ❑ 100ul → Unsigned Long Integer Constant

Floating Point Constants

47

- A floating-point constant must contain both integer and decimal parts. Some times it may also contain the exponent part.
- The default value of each floating-point type is zero, 0.
- Each of the floating-point types has the `MinValue` and `MaxValue` constants that provide the minimum and maximum finite value of that type.
- The `float` and `double` types also provide constants that represent not-a-number and infinity values.
- For example, the `double` type provides the following constants: `Double.NaN`, `Double.NegativeInfinity`, and `Double.PositiveInfinity`.
- The type of a real literal is determined by its suffix as follows:
 - The literal without suffix or with the `d` or `D` suffix is of type `double`
 - The literal with the `f` or `F` suffix is of type `float`
 - The literal with the `m` or `M` suffix is of type `decimal`

- When a floating-point constant is represented in exponent form, the value must be suffixed with 'e' or 'E'.

Example

The floating-point value 3.14 is represented as 3E-14 in exponent form.

```
const double d = 3D;  
const double e = 0.42e2;  
Console.WriteLine(e); // output 42  
  
const float f = 134.45E-2f;  
Console.WriteLine(f); // output: 1.3445  
  
const decimal m = 1.5E6m;  
Console.WriteLine(m); // output: 1500000
```

Character Constants

49

- A character constant is a symbol enclosed in single quotation. A character constant has a maximum length of one character.
- We can specify a char value with:
 - ▣ a character literal.
 - ▣ a Unicode escape sequence, which is `\u` followed by the four-symbol hexadecimal representation of a character code.
 - ▣ a hexadecimal escape sequence, which is `\x` followed by the hexadecimal representation of a character code.

```
var chars = new[]  
{  
    'j',  
    '\u006A',  
    '\x006A',  
    (char)106,  
};  
Console.WriteLine(string.Join(" ", chars)); //  
output: j j j j
```

String Constants

51

- ❑ The string type represents a sequence of zero or more Unicode characters. string is an alias for `System.String` in .NET.
- ❑ Although string is a reference type, the equality operators `==` and `!=` are defined to compare the values of string objects, not references. This makes testing for string equality more intuitive.
- ❑ For example:

```
const string a = "hello";  
string b = "h";  
// Append to contents of 'b'  
b += "ello";  
Console.WriteLine(a == b);  
Console.WriteLine(object.ReferenceEquals(a, b));
```


- This displays "True" and then "False" because the content of the strings are equivalent, but a and b do not refer to the same string instance.
- The + operator concatenates strings:

eg:

```
const string a = "good " + "morning";  
const string s = "Good Morning";  
const string s = "123+456";
```

Escape Sequences

53

- ❑ Character combinations consisting of a backslash (\) followed by a letter or by a combination of digits are called "escape sequences."
- ❑ To represent a newline character, single quotation mark, or certain other characters in a character constant, we use escape sequences.
- ❑ An escape sequence is regarded as a single character and is therefore valid as a character constant.
- ❑ They are also used to provide literal representations of nonprinting characters and characters that usually have special meanings, such as the double quotation mark ("). The following table lists the ANSI escape sequences and what they represent.

Escape Sequence	Represents
<code>\a</code>	Bell (alert)
<code>\b</code>	Backspace
<code>\f</code>	Form feed
<code>\n</code>	New line
<code>\r</code>	Carriage return
<code>\t</code>	Horizontal tab
<code>\v</code>	Vertical tab
<code>\'</code>	Single quotation mark
<code>\"</code>	Double quotation mark
<code>\\</code>	Backslash
<code>\?</code>	Literal question mark
<code>\ooo</code>	ASCII character in octal notation
<code>\x hh</code>	ASCII character in hexadecimal notation
<code>\x hhhh</code>	Unicode character in hexadecimal notation if this escape sequence is used in a wide-character constant or a Unicode string literal.

Operators

55

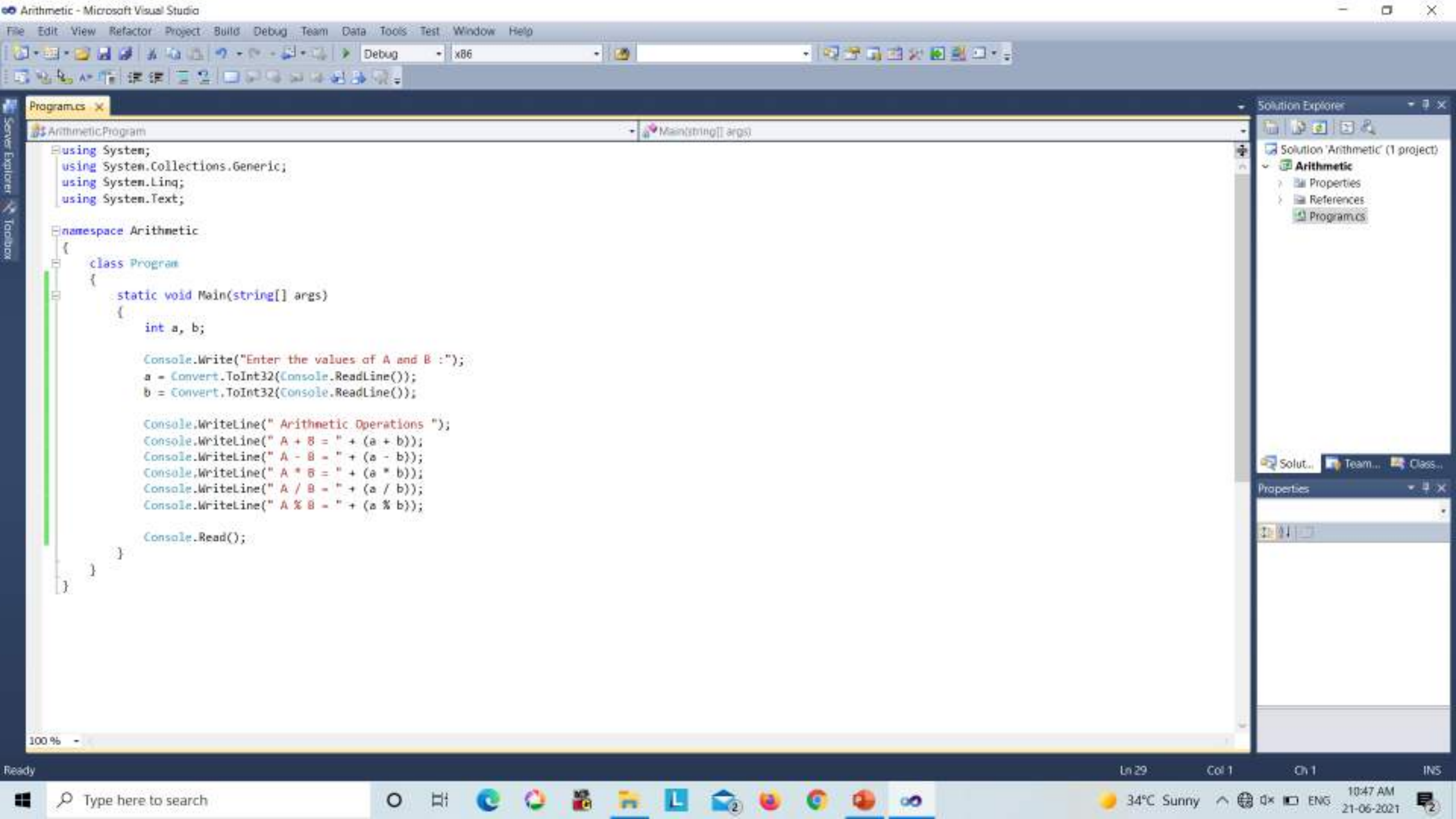
- An operator is a symbol used to perform arithmetic and logical operations in a program. That means an operator is a special symbol that tells the compiler to perform mathematical or logical operations. C# programming language supports a rich set of operators that are classified as follows.
 - ▣ Arithmetic Operators
 - ▣ Relational Operators
 - ▣ Logical Operators
 - ▣ Increment & Decrement Operators
 - ▣ Assignment Operators
 - ▣ Bitwise Operators
 - ▣ Conditional Operator
 - ▣ Member access operators
 - ▣ Special Operators

Arithmetic Operators (+, -, *, /, %)

56

- The arithmetic operators are the symbols that are used to perform basic mathematical operations like addition, subtraction, multiplication, division and percentage modulo. The following table provides information about arithmetic operators.
- The addition operator can be used with numerical data types and character data type. When it is used with numerical values, it performs mathematical addition and when it is used with character data type values, it performs concatenation (appending).
- The remainder of the division operator is used with integer data type only.

Operator	Meaning	Example
+	Addition	$10 + 5 = 15$
-	Subtraction	$10 - 5 = 5$
*	Multiplication	$10 * 5 = 50$
/	Division	$10 / 5 = 2$
%	Remainder of the Division	$5 \% 2 = 1$





file:///c:/users/dr.pv.praveen sundar/documents/visual studio 2010/Projects/Arithmetic/Arithmetic/bin/Debug/Arithmetic.EXE

Enter the values of A and B :20

30

Arithmetic Operations

A + B = 50

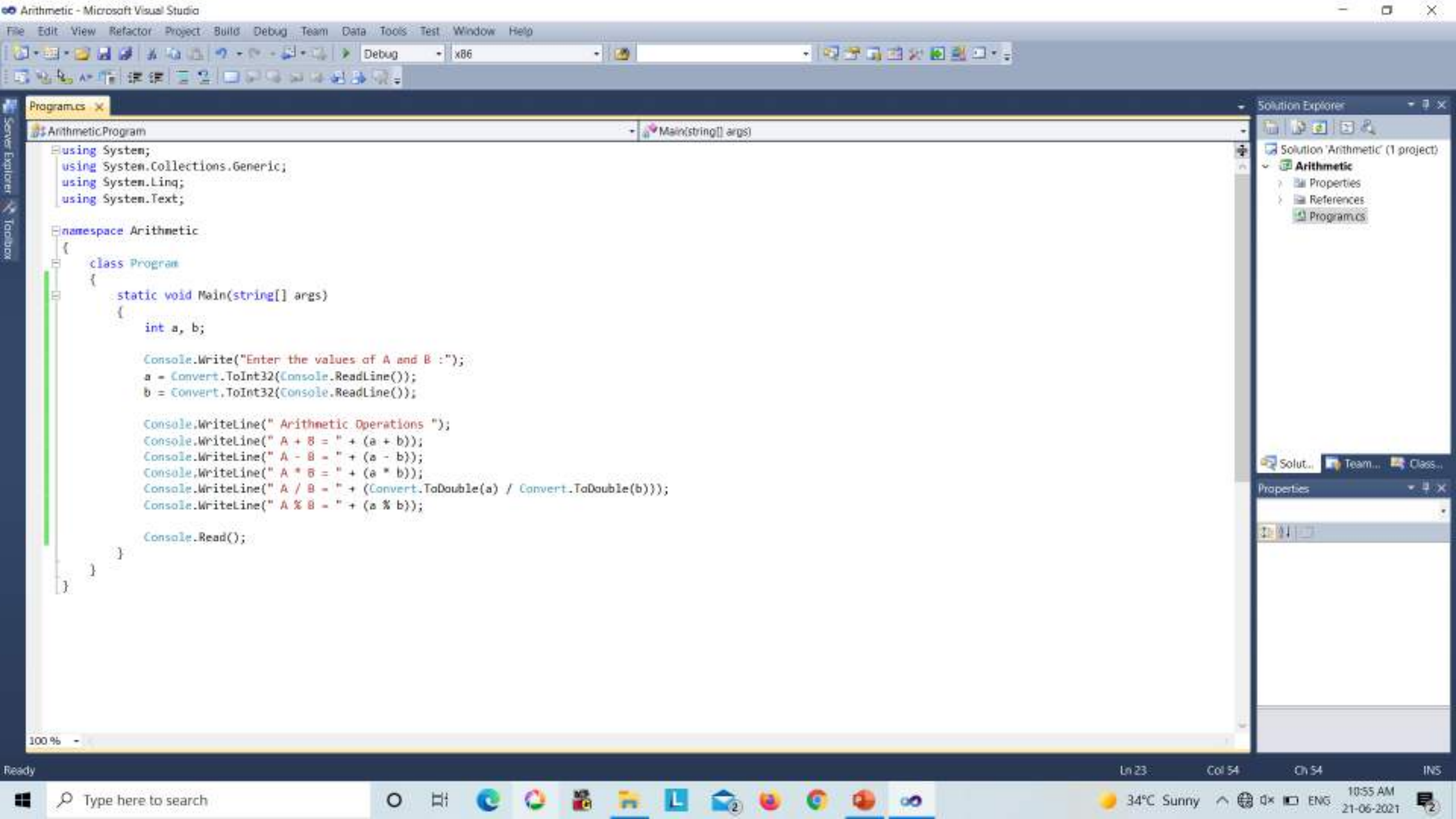
A - B = -10

A * B = 600

A / B = 0

A % B = 20

■



Enter the values of A and B :20

30

Arithmetic Operations

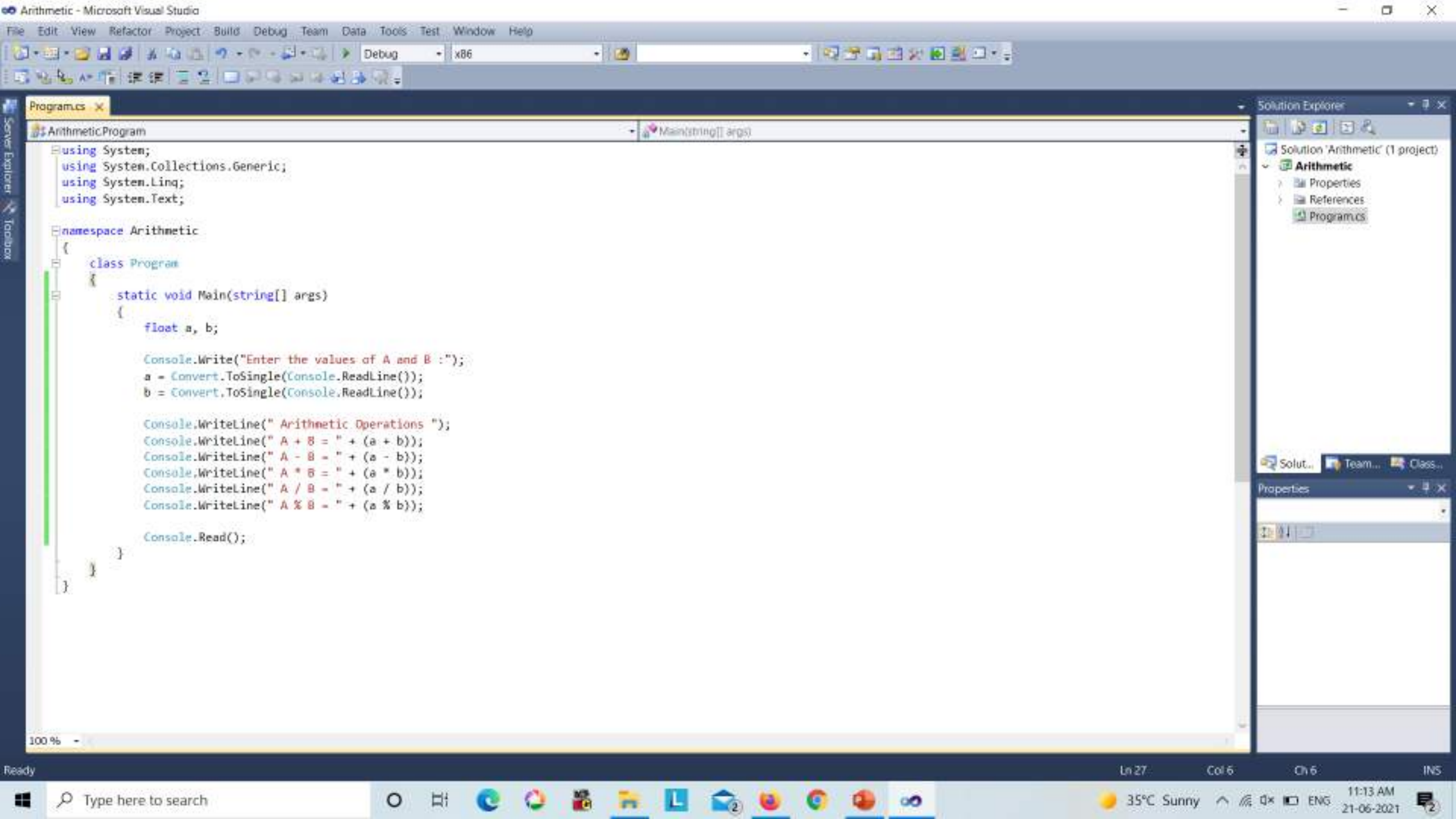
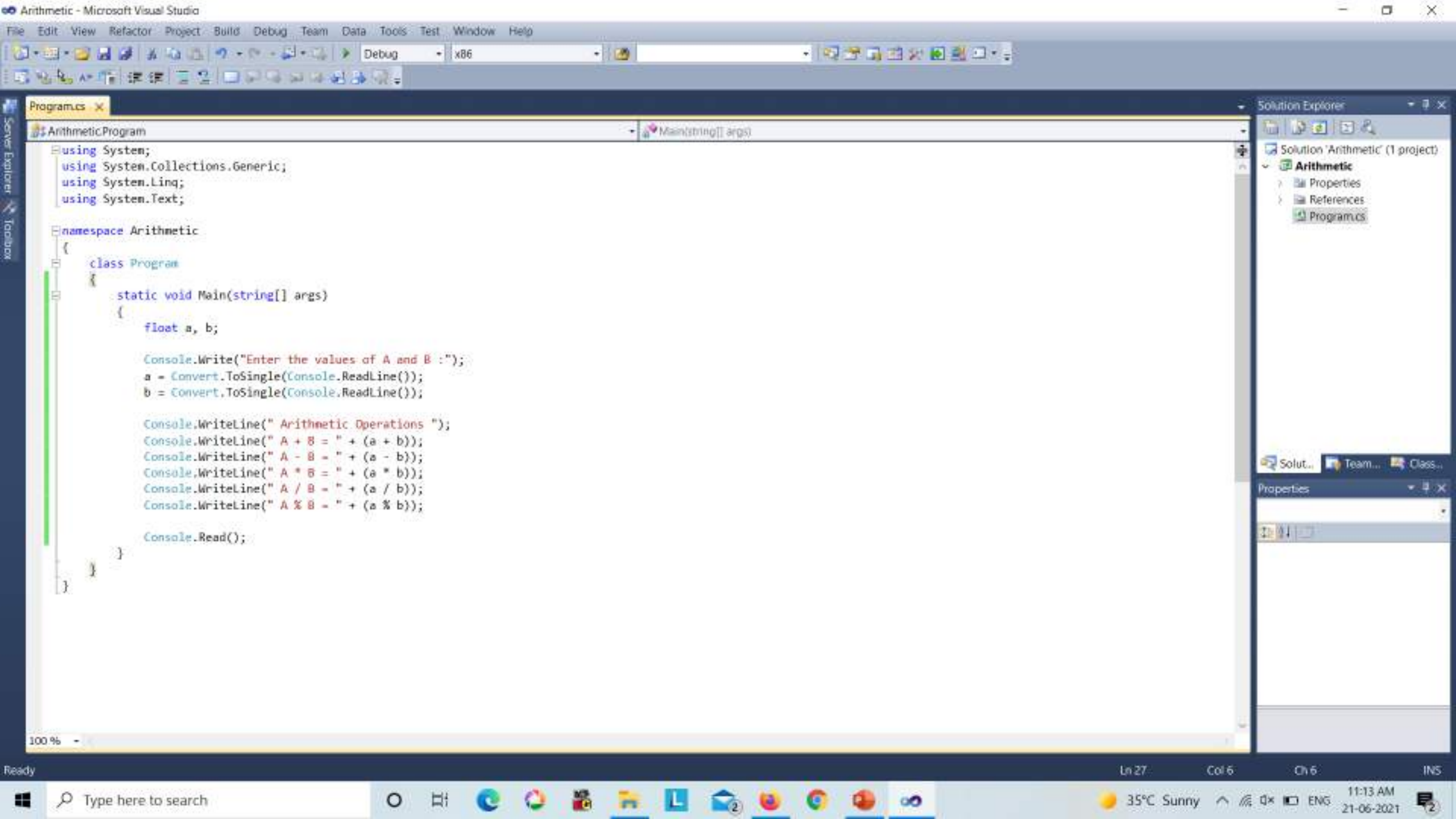
A + B = 50

A - B = -10

A * B = 600

A / B = 0.6666666666666667

A % B = 20



file:///c:/users/dr.pv.praveen sundar/documents/visual studio 2010/Projects/Arithmetic/Arithmetic/bin/Debug/Arithmetic.EXE

Enter the values of A and B :10.32

20.45

Arithmetic Operations

A + B = 30.77

A - B = -10.13

A * B = 211.044

A / B = 0.5046455

A % B = 10.32

file:///c:/users/dr.pv.praveen.sundar/documents/visual.studio.2010/projects/Arithmetic/Arithmetic/bin/Debug/Arithmetic.EXE

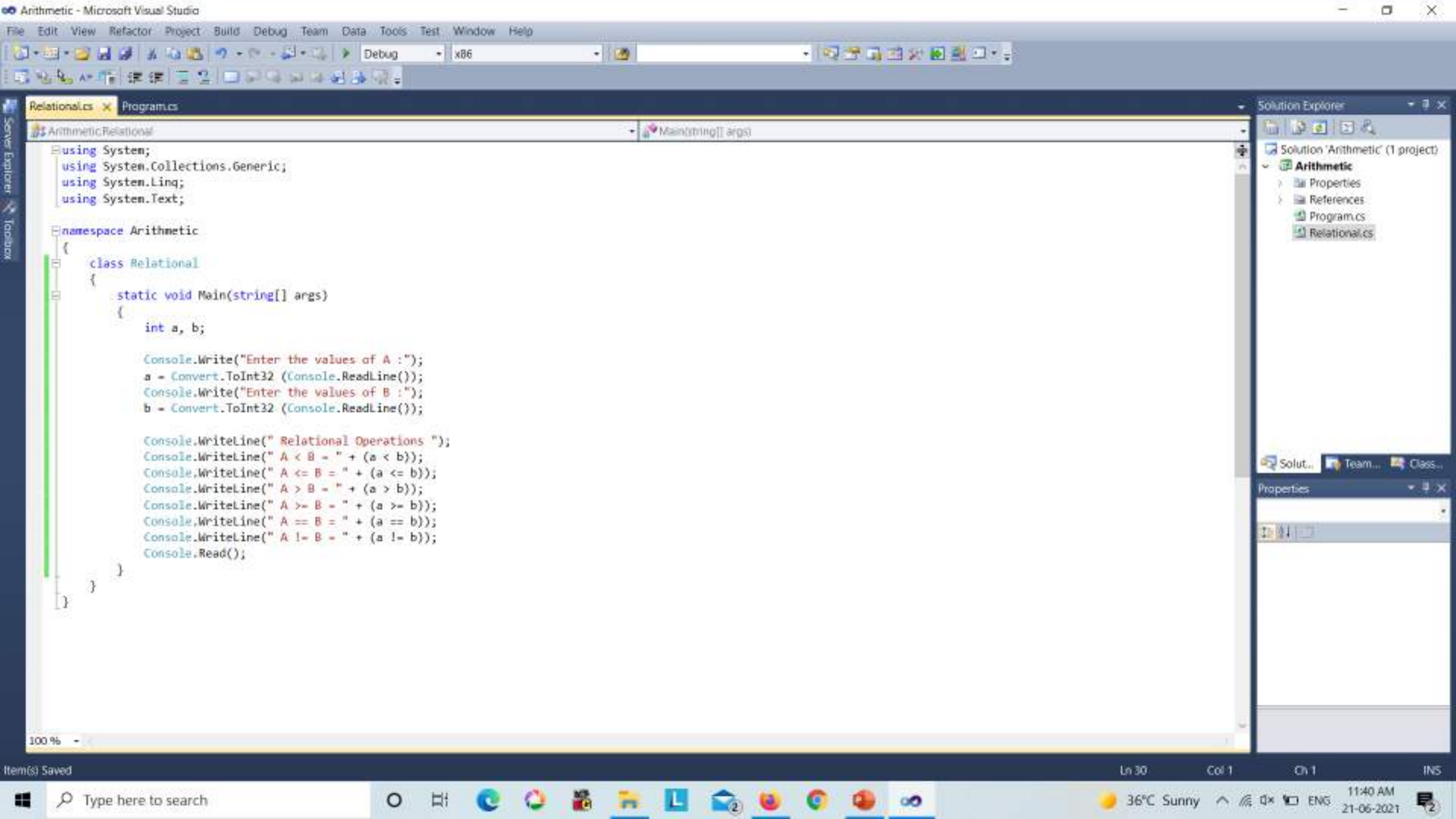
Enter the values of A and B :20.45
10.32
Arithmetic Operations
A + B = 30.77
A - B = 10.13
A * B = 211.044
A / B = 1.981589
A % B = 10.13

Relational Operators (<, >, <=, >=, ==, !=)

64

- The relational operators are the symbols that are used to compare two values. That means the relational operators are used to check the relationship between two values. Every relational operator has two results TRUE or FALSE. In simple words, the relational operators are used to define conditions in a program. The following table provides information about relational operators.

Operator	Meaning	Example
<	Returns TRUE if the first value is smaller than second value otherwise returns FALSE	10 < 5 is FALSE
>	Returns TRUE if the first value is larger than second value otherwise returns FALSE	10 > 5 is TRUE
<=	Returns TRUE if the first value is smaller than or equal to second value otherwise returns FALSE	10 <= 5 is FALSE
>=	Returns TRUE if the first value is larger than or equal to second value otherwise returns FALSE	10 >= 5 is TRUE
==	Returns TRUE if both values are equal otherwise returns FALSE	10 == 5 is FALSE
!=	Returns TRUE if both values are not equal otherwise returns FALSE	10 != 5 is TRUE



Enter the values of A :20

Enter the values of B :30

Relational Operations

A < B = True

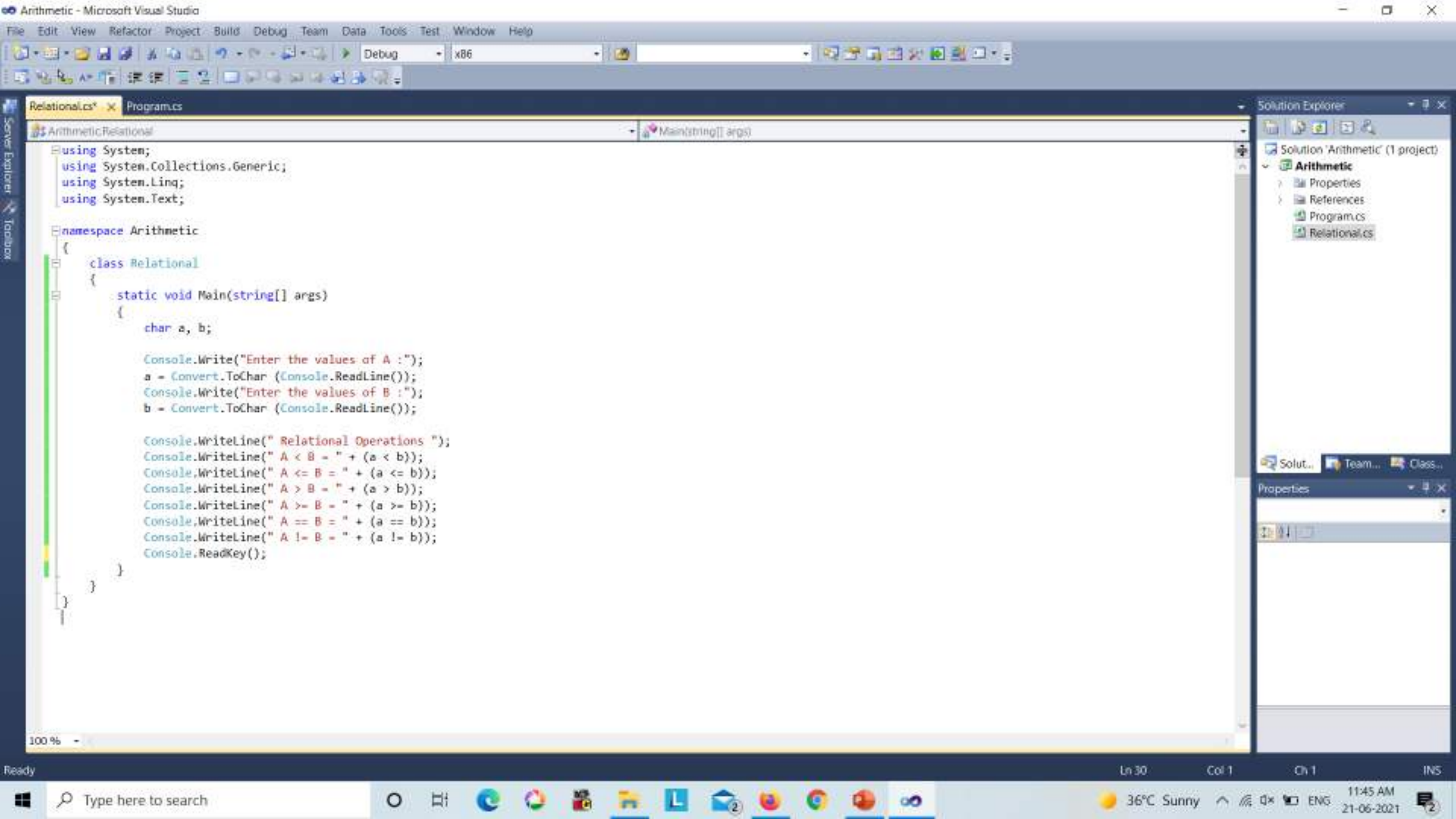
A <= B = True

A > B = False

A >= B = False

A == B = False

A != B = True



```
Arithmetic - Microsoft Visual Studio
File Edit View Refactor Project Build Debug Team Data Tools Test Window Help
Debug x86
Relational.cs Program.cs
Arithmetic.Relational Main(string[] args)
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Arithmetic
{
    class Relational
    {
        static void Main(string[] args)
        {
            char a, b;

            Console.Write("Enter the values of A :");
            a = Convert.ToChar(Console.ReadLine());
            Console.Write("Enter the values of B :");
            b = Convert.ToChar(Console.ReadLine());

            Console.WriteLine(" Relational Operations ");
            Console.WriteLine(" A < B = " + (a < b));
            Console.WriteLine(" A <= B = " + (a <= b));
            Console.WriteLine(" A > B = " + (a > b));
            Console.WriteLine(" A >= B = " + (a >= b));
            Console.WriteLine(" A == B = " + (a == b));
            Console.WriteLine(" A != B = " + (a != b));
            Console.ReadKey();
        }
    }
}
```



```
file:///c:/users/dr.pv.praveen sundar/documents/visual studio 2010/Projects/Arithmetic/Arithmetic/bin/Debug/Arithmetic.EXE
Enter the values of A :f
Enter the values of B :k
Relational Operations
A < B = True
A <= B = True
A > B = False
A >= B = False
A == B = False
A != B = True
```

Logical Operators (&&, ||, !)

69

- The logical operators are the symbols that are used to combine multiple conditions into one condition. The following table provides information about logical operators.
- Logical AND - Returns TRUE only if all conditions are TRUE, if any of the conditions is FALSE then complete condition becomes FALSE.
- Logical OR - Returns FALSE only if all conditions are FALSE, if any of the conditions is TRUE then complete condition becomes TRUE.

Operator	Meaning	Example
&&	Logical AND - Returns TRUE if all conditions are TRUE otherwise returns FALSE	10 < 5 && 12 > 10 is FALSE
	Logical OR - Returns FALSE if all conditions are FALSE otherwise returns TRUE	10 < 5 12 > 10 is TRUE
!	Logical NOT - Returns TRUE if condition is FALSE and returns FALSE if it is TRUE	!(10 < 5 && 12 > 10) is TRUE

Arithmetic - Microsoft Visual Studio

File Edit View Refactor Project Build Debug Team Data Tools Test Window Help

Debug x86

Logicals.cs Relational.cs Program.cs

Arithmetic.Logical Main(string[] args)

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Arithmetic
{
    class Logical
    {
        static void Main(string[] args)
        {
            int a, b, c;

            Console.Write("Enter the values of A :");
            a = Convert.ToInt16(Console.ReadLine());
            Console.Write("Enter the values of B :");
            b = Convert.ToInt16(Console.ReadLine());
            Console.Write("Enter the values of C :");
            c = Convert.ToInt16(Console.ReadLine());

            Console.WriteLine(" Logical Operations ");
            Console.WriteLine(" (A < B) && (C < A) = " + ((a < b) && (c < a)));
            Console.WriteLine(" (A < B) || (C < A) = " + ((a < b) || (c < a)));
            Console.WriteLine(" !(A < B) && (C < A) = " + (! (a < b) && (c < a)));
            Console.WriteLine(" (A < B) && !(C < A) = " + ((a < b) && !(c < a)));
            Console.ReadKey();
        }
    }
}
```

Solution Explorer

Solution 'Arithmetic' (1 project)

- Arithmetic
 - Properties
 - References
 - Logicals.cs
 - Program.cs
 - Relational.cs

Properties

100 %

file:///c:/users/dr.pv.praveen sundar/documents/visual studio 2010/Projects/Arithmetic/Arithmetic/bin/Debug/Arithmetic.EXE

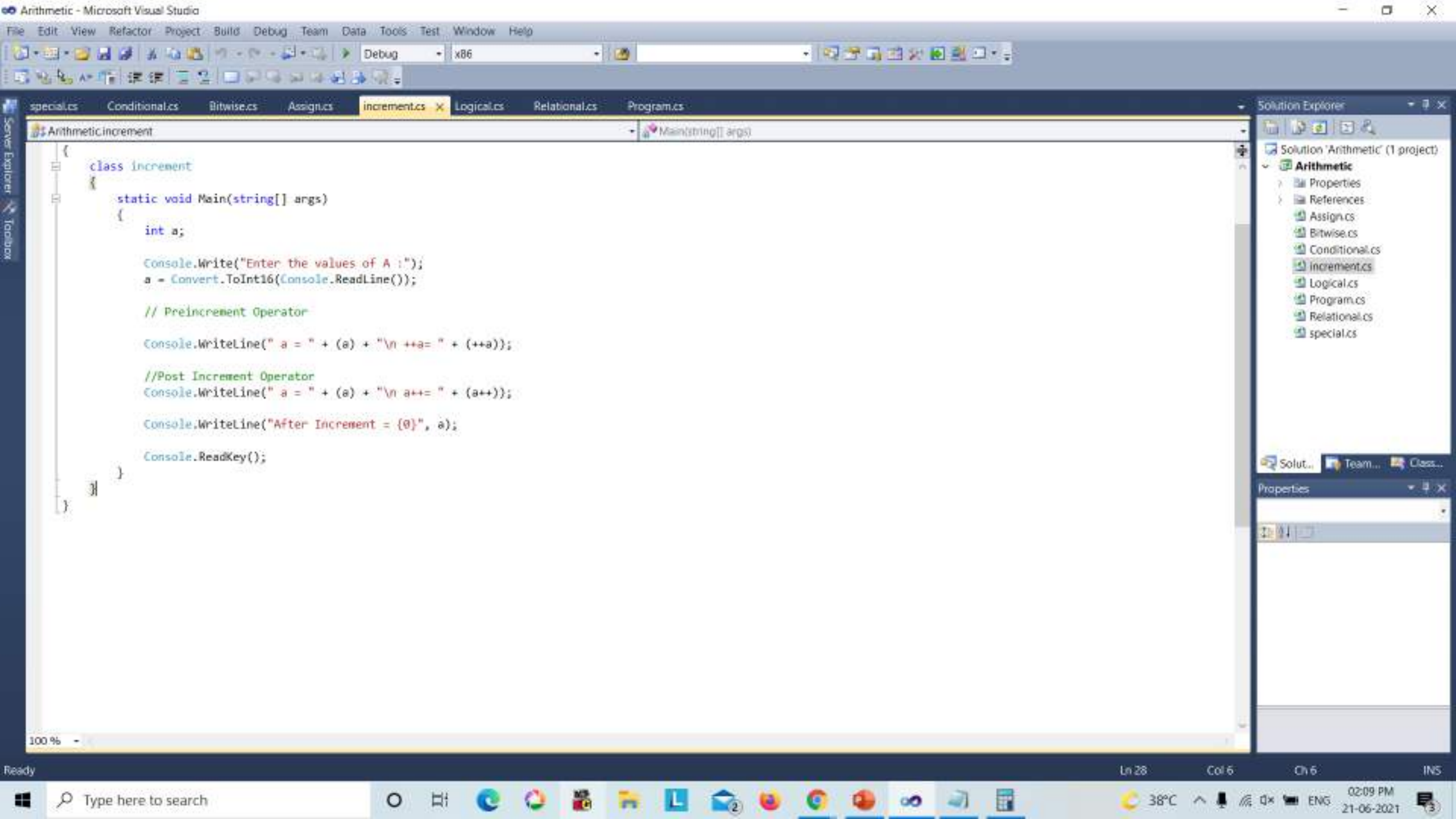
Enter the values of A :20
Enter the values of B :30
Enter the values of C :10
Logical Operations
(A < B) && (C < A) = True
(A < B) || (C < A) True
!(A < B) && (C < A) False
(A < B) && !(C < A) = False

Increment & Decrement Operators (++ & --)

72

- The increment and decrement operators are called unary operators because both need only one operand. The increment operators adds one to the existing value of the operand and the decrement operator subtracts one from the existing value of the operand. The following table provides information about increment and decrement operators.
- The increment and decrement operators are used Infront of the operand (++a) or after the operand (a++). If it is used in front of the operand, we call it as pre-increment or pre-decrement and if it is used after the operand, we call it as post-increment or post-decrement.

Operator	Meaning	Example
++	Increment - Adds one to existing value	int a = 5; a++; \Rightarrow a = 6
--	Decrement - Subtracts one from existing value	int a = 5; a--; \Rightarrow a = 4



file:///c:/users/dr.pv.praveen sundar/documents/visual studio 2010/Projects/Arithmetic/Arithmetic/bin/Debug/Arithmetic.EXE

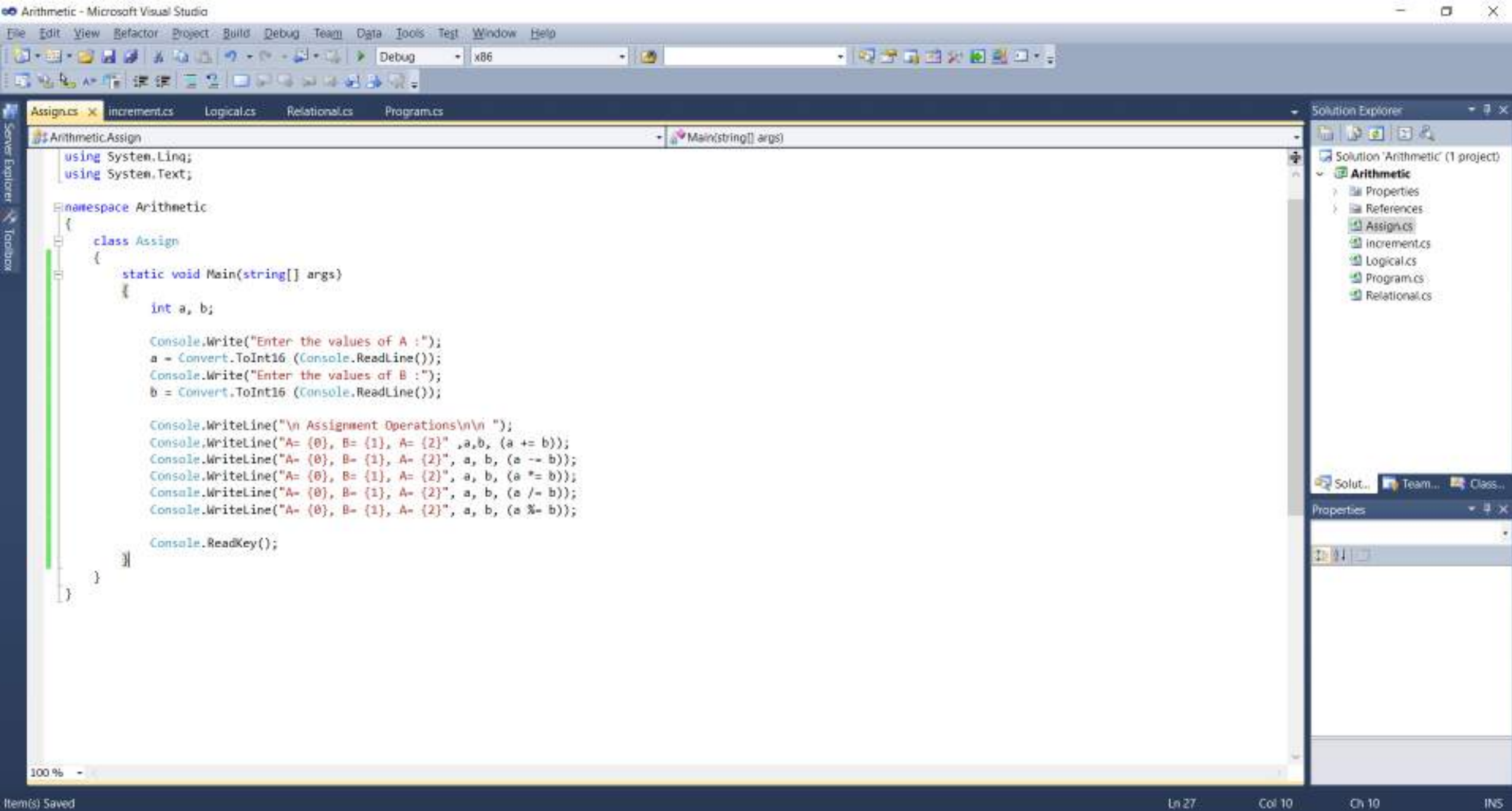
Enter the values of A :10
a = 10
++a= 11
a = 11
a++= 11
After Increment = 12

Assignment Operators (=, +=, -=, *=, /=, %=)

75

- The assignment operators are used to assign right-hand side value (Rvalue) to the left-hand side variable (Lvalue). The assignment operator is used in different variants along with arithmetic operators. The following table describes all the assignment operators in the C programming language.

Operator	Meaning	Example
=	Assign the right-hand side value to left-hand side variable	A = 15
+=	Add both left and right-hand side values and store the result into left-hand side variable	A += 10 ⇒ A = A+10
-=	Subtract right-hand side value from left-hand side variable value and store the result into left-hand side variable	A -= B ⇒ A = A-B
*=	Multiply right-hand side value with left-hand side variable value and store the result into left-hand side variable	A *= B ⇒ A = A*B
/=	Divide left-hand side variable value with right-hand side variable value and store the result into the left-hand side variable	A /= B ⇒ A = A/B
%=	Divide left-hand side variable value with right-hand side variable value and store the remainder into the left-hand side variable	A %= B ⇒ A = A%B



file:///c:/users/dr.pv.praveen sundar/documents/visual studio 2010/Projects/Arithmetic/Arithmetic/bin/Debug/Arithmetic.EXE

Enter the values of A :10
Enter the values of B :5

Assignment Operations

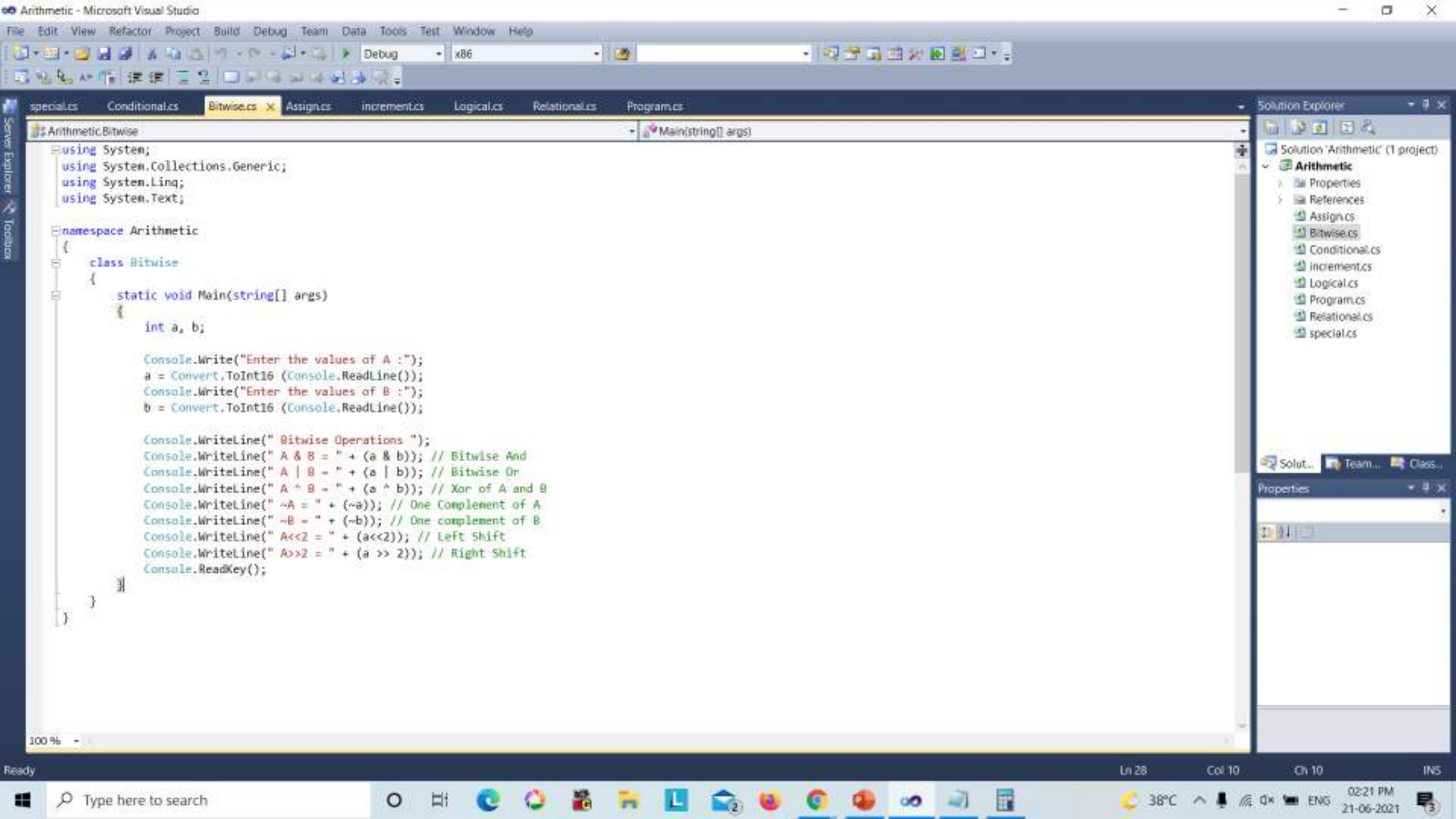
A= 10, B= 5, A= 15
A= 15, B= 5, A= 10
A= 10, B= 5, A= 50
A= 50, B= 5, A= 10
A= 10, B= 5, A= 0

Bitwise Operators (&, |, ^, ~, >>, <<)

78

- The bitwise operators are used to perform bit-level operations in the c programming language. When we use the bitwise operators, the operations are performed based on the binary values. The following table describes all the bitwise operators in the C programming language. Let us consider two variables A and B as A = 25 (11001) and B = 20 (10100).

Operator	Meaning	Example
&	the result of Bitwise AND is 1 if all the bits are 1 otherwise it is 0	A & B ⇒ 16 (10000)
	the result of Bitwise OR is 0 if all the bits are 0 otherwise it is 1	A B ⇒ 29 (11101)
^	the result of Bitwise XOR is 0 if all the bits are same otherwise it is 1	A ^ B ⇒ 13 (01101)
~	the result of Bitwise once complement is negation of the bit (Flipping)	~A ⇒ -26
<<	the Bitwise left shift operator shifts all the bits to the left by the specified number of positions	A << 2 ⇒ 100 (1100100)
>>	the Bitwise right shift operator shifts all the bits to the right by the specified number of positions	A >> 2 ⇒ 6 (00110)



file:///c:/users/dr.pv.praveen sundar/documents/visual studio 2010/Projects/Arithmetic/Arithmetic/bin/Debug/Arithmetic.EXE

Enter the values of A :25
Enter the values of B :20
Bitwise Operations
A & B = 16
A | B = 29
A ^ B = 13
~A = -26
~B = -21
A<<2 = 100
A>>2 = 6

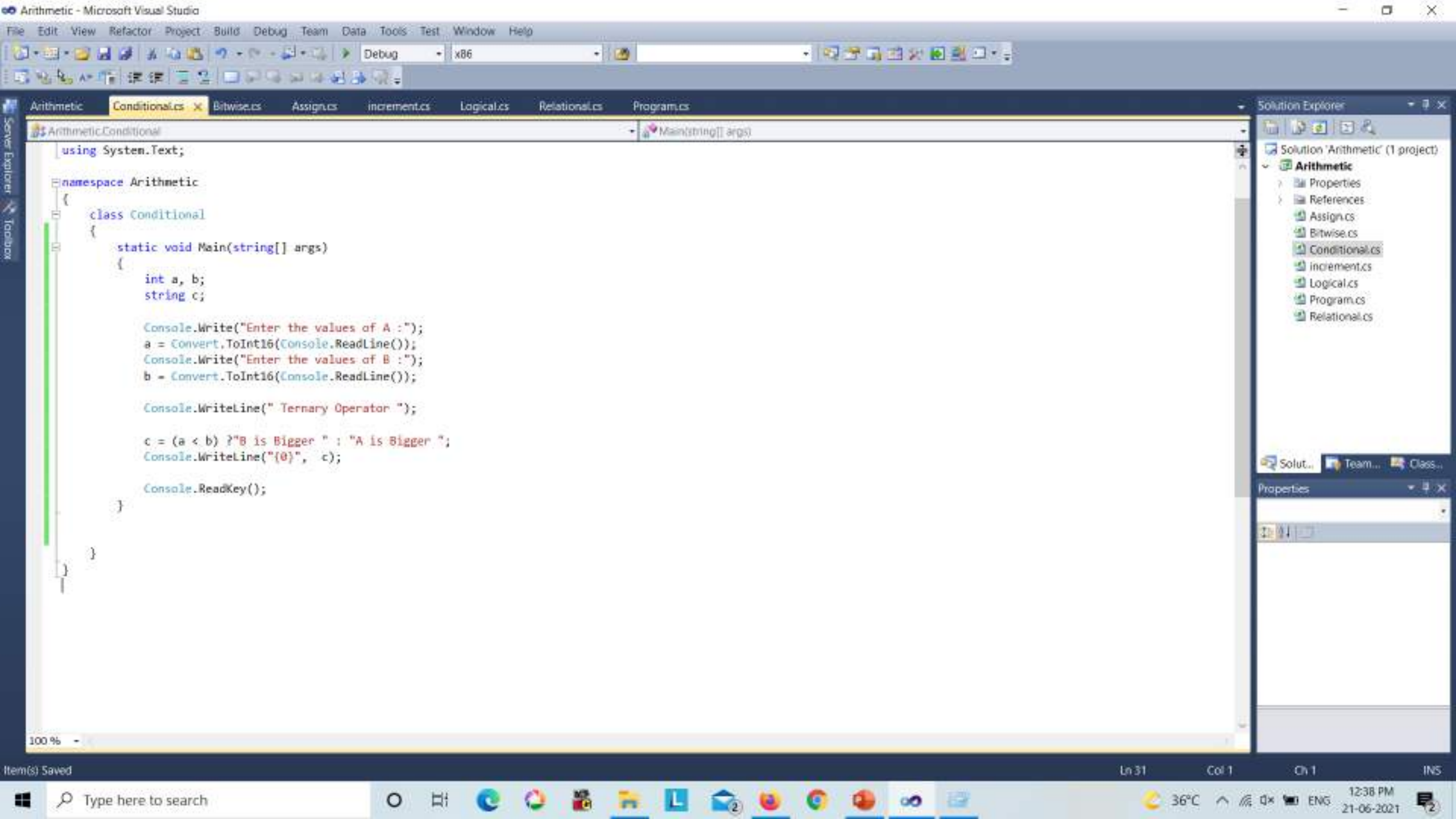
Conditional Operator (?:)

81

- The conditional operator is also called a ternary operator because it requires three operands. This operator is used for decision making. In this operator, first we verify a condition, then we perform one operation out of the two operations based on the condition result. If the condition is TRUE the first option is performed, if the condition is FALSE the second option is performed. The conditional operator is used with the following syntax.
- Condition ? TRUE Part : FALSE Part;

Example

A = (10<15)?100:200; ⇒ A value is 100



Enter the values of A :20

Enter the values of B :10

Ternary Operator

A is Bigger

file:///c:/users/dr.pv.praveen sundar/documents/visual studio 2010/Projects/Arithmetic/Arithmetic/bin/Debug/Arithmetic.EXE

Enter the values of A :10
Enter the values of B :20
Ternary Operator
B is Bigger

Special Operators (sizeof, is, as, typeof, etc.)

- The following are the special operators in C# programming language.

sizeof operator

- **sizeof()** is an operator in C#, it is used to get the size in bytes of compile-time known types, it does not work with the variables or instances. This operator is used with the following syntax.

```
int sizeof(type);
```

- It accepts the type and returns an int value – which is the size of that type in bytes.

Arithmetic - Microsoft Visual Studio

File Edit View Refactor Project Build Debug Team Data Tools Test Window Help

Debug x86

Checked.cs special.cs Conditional.cs Bitwise.cs Assign.cs Increment.cs Logical.cs Relational.cs Program.cs

Arithmetic.special Main(string[] args)

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Arithmetic
{
    class special
    {
        static void Main(string[] args)
        {
            Console.WriteLine(" Size of Byte={0}", sizeof(byte));
            Console.WriteLine(" Size of Bool={0}", sizeof(bool));
            Console.WriteLine(" Size of Char={0}", sizeof(char));
            Console.WriteLine(" Size of Short={0}", sizeof(short));
            Console.WriteLine(" Size of Int= {0}", sizeof(int));
            Console.WriteLine(" Size of UInt= {0}", sizeof(uint));
            Console.WriteLine(" Size of UInt32= {0}", sizeof(UInt32));
            Console.WriteLine(" Size of Int32= {0}", sizeof(Int32));
            Console.WriteLine(" Size of Float={0}", sizeof(float));
            Console.WriteLine(" Size of Double={0}", sizeof(double));
            Console.WriteLine(" Size of Decimal={0}", sizeof(decimal));

            Console.ReadKey();
        }
    }
}
```

Solution Explorer

Solution 'Arithmetic' (1 project)

- Arithmetic
 - Properties
 - References
 - Assign.cs
 - Bitwise.cs
 - Checked.cs
 - Conditional.cs
 - Increment.cs
 - Logical.cs
 - Program.cs
 - Relational.cs
 - special.cs

Properties

Ln 13 Col 66 Ch 66 INS

Ready

Type here to search

32°C 12:59 AM 22-06-2021

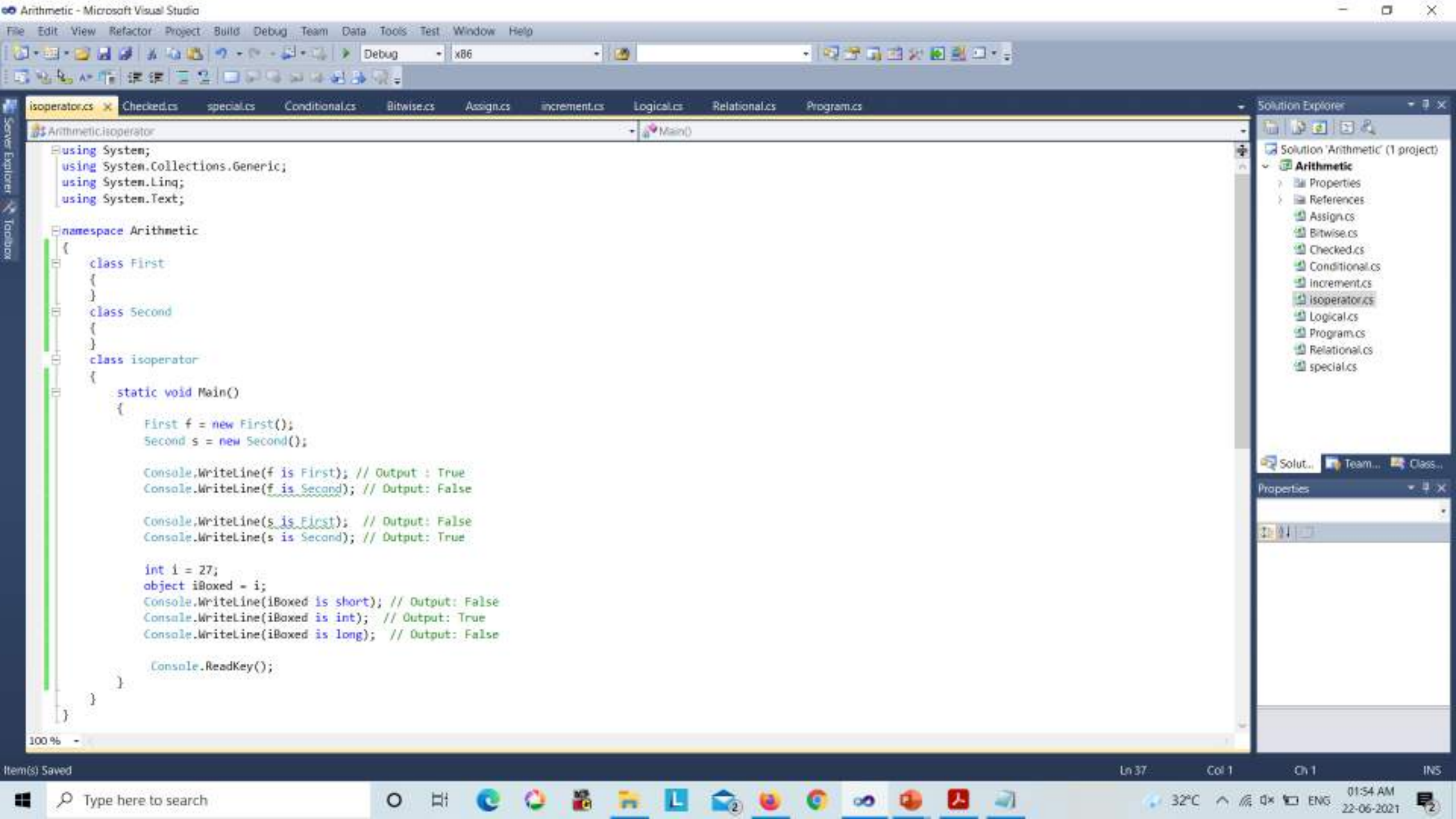
is operator

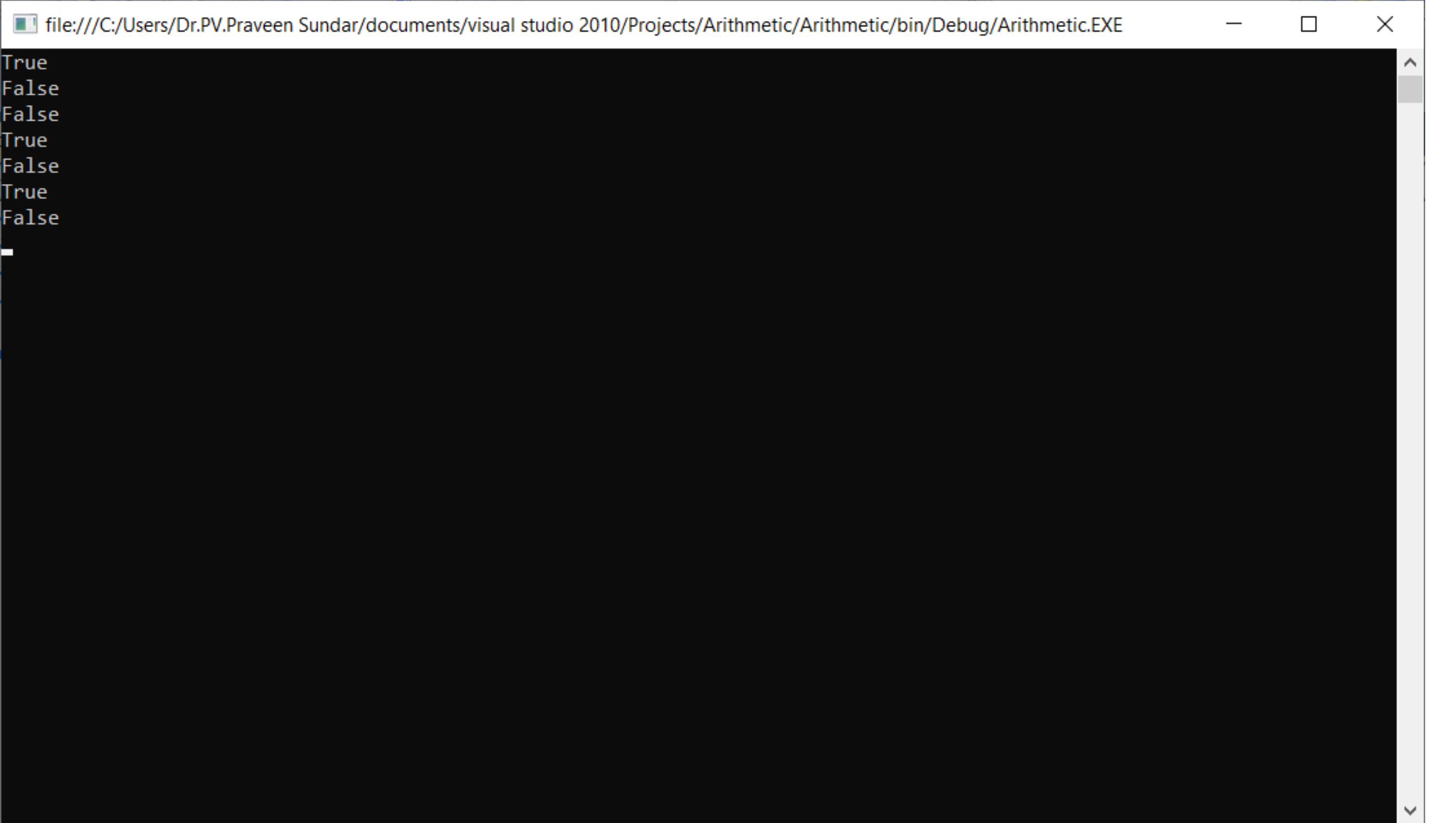
87

- The is operator is used to check if the run-time type of an object is compatible with the given type or not.
- The expression with the type-testing is operator has the following form.

E is T

- where E is an expression that returns a value and T is the name of a type or a type parameter.
- The is operator returns true when an expression result is non-null and any of the following conditions are true:
 - ▣ The run-time type of an expression result is T.
 - ▣ The run-time type of an expression result derives from type T, implements interface T, or another implicit reference conversion exists from it to T.
 - ▣ A boxing or unboxing conversion exists from the run-time type of an expression result to type T.





True
False
False
True
False
True
False

As operator

90

- ❑ The as operator explicitly converts the result of an expression to a given reference or nullable value type.
- ❑ If the conversion is not possible, the as operator returns null. Unlike a cast expression, the as operator never throws an exception.
- ❑ The expression of the form

E is T

Arithmetic - Microsoft Visual Studio

File Edit View Refactor Project Build Debug Team Data Tools Test Window Help

Debug x86

asoperator.cs isoperator.cs Checked.cs special.cs Conditional.cs Bitwise.cs Assign.cs increment.cs Logical.cs Relational.cs Program.cs

Arithmetic.asoperator Main()

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Arithmetic
{
    class asoperator
    {
        static void Main()
        {
            object[] myobjects = new object[4];

            myobjects[0] = new First();
            myobjects[1] = 123;
            myobjects[2] = 123.ToString();
            myobjects[3] = "Student";

            for(int i = 0; i < myobjects.Length; i++)
            {
                string s = myobjects[i] as string;
                Console.WriteLine("Inspecting element: {MyObjects[i]}");

                if (s == null)
                {
                    Console.WriteLine("->> Incompatible type");
                }
                else
                {
                    Console.WriteLine("->> Compatible type");
                }

                Console.WriteLine(", with string!");
            }
            Console.ReadKey();
        }
    }
}
```

Solution Explorer

Solution 'Arithmetic' (1 project)

- Arithmetic
 - Properties
 - References
 - asoperator.cs
 - Assign.cs
 - Bitwise.cs
 - Checked.cs
 - Conditional.cs
 - increment.cs
 - isoperator.cs
 - Logical.cs
 - Program.cs
 - Relational.cs
 - special.cs

Properties

100%


```
Inspecting element: {MyObjects[i]} ->> Incompatible type, with string!  
Inspecting element: {MyObjects[i]} ->> Incompatible type, with string!  
Inspecting element: {MyObjects[i]} ->> Compatible type, with string!  
Inspecting element: {MyObjects[i]} ->> Compatible type, with string!
```

Difference between is and as operator

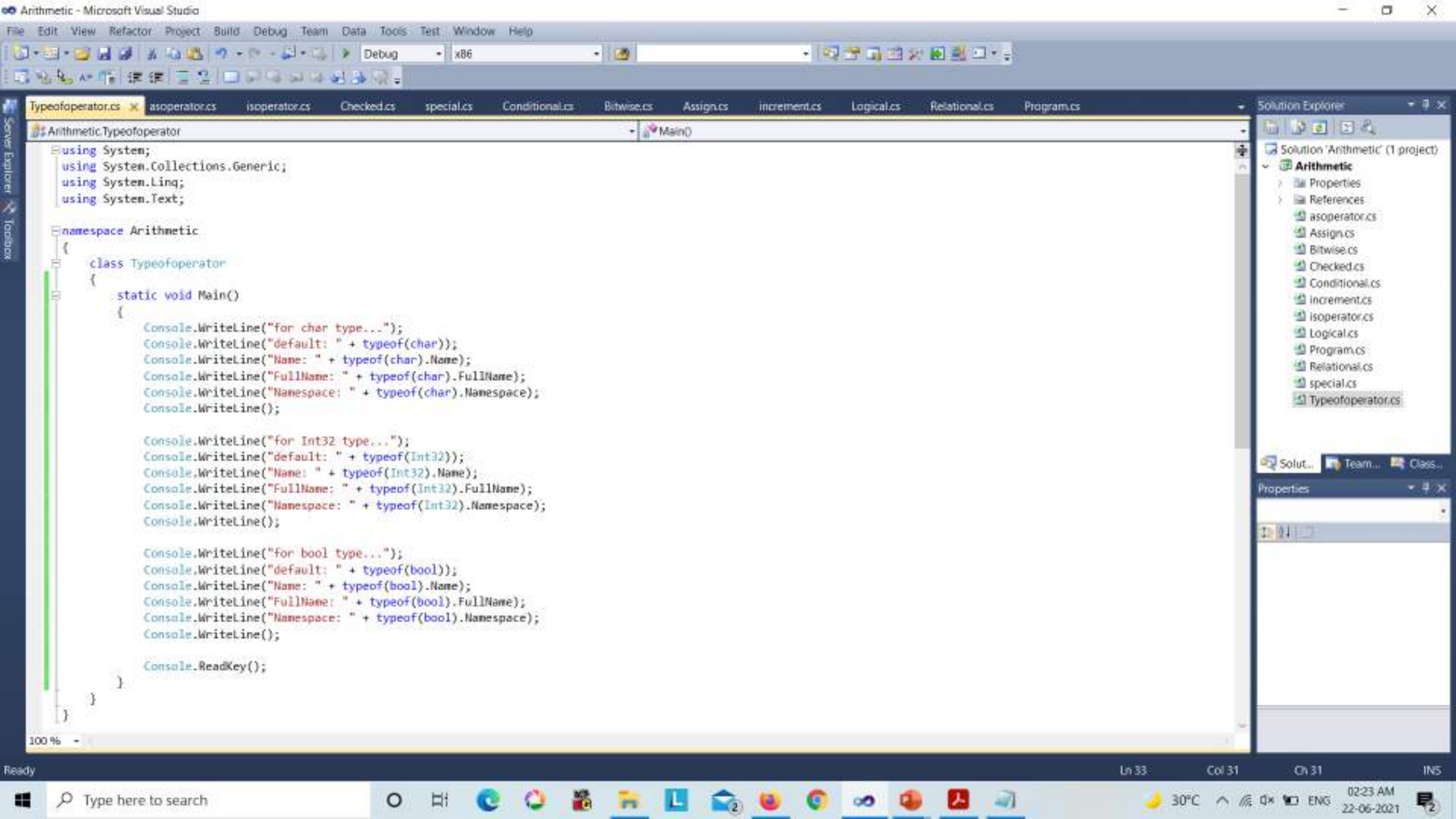
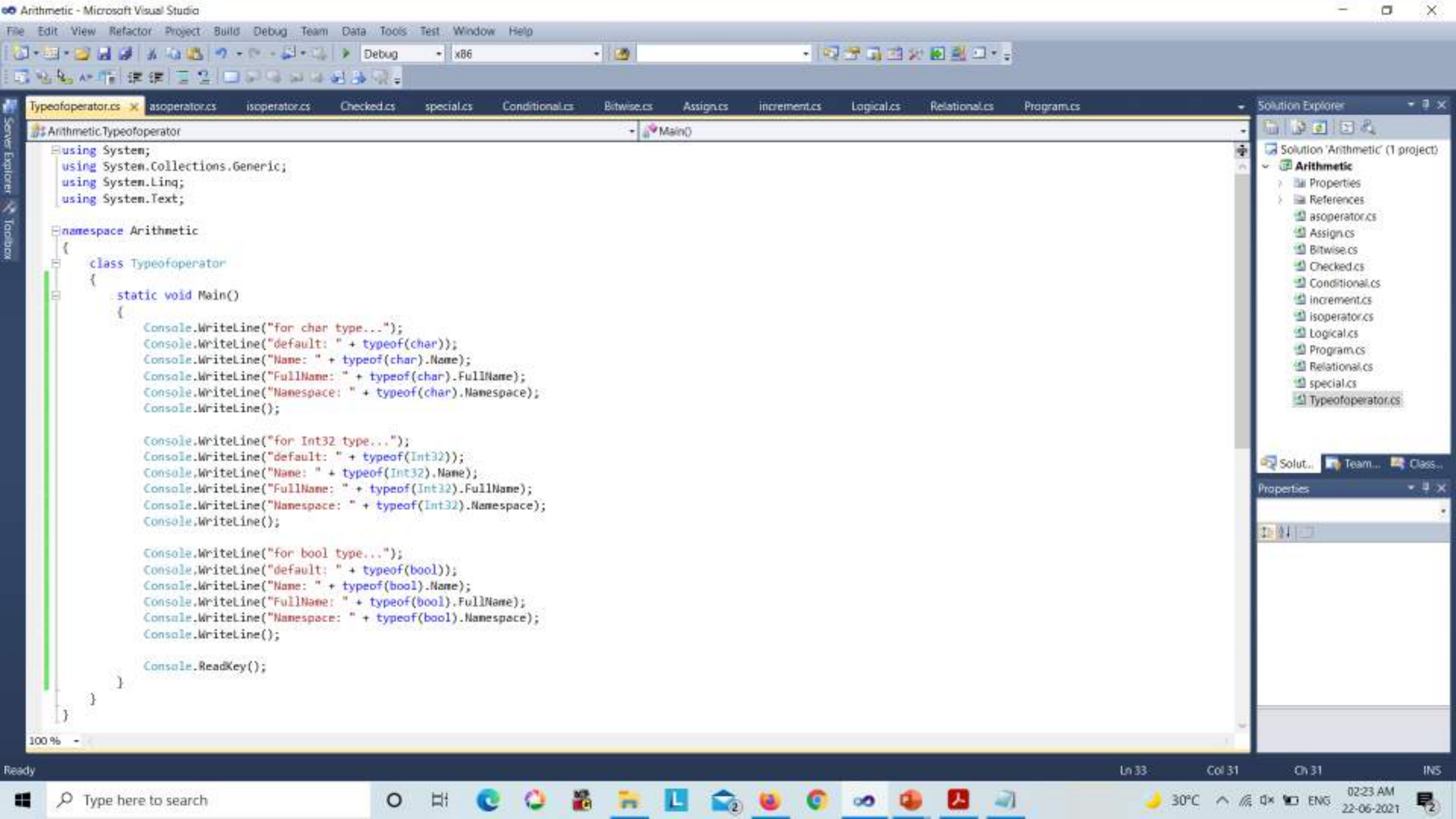
93

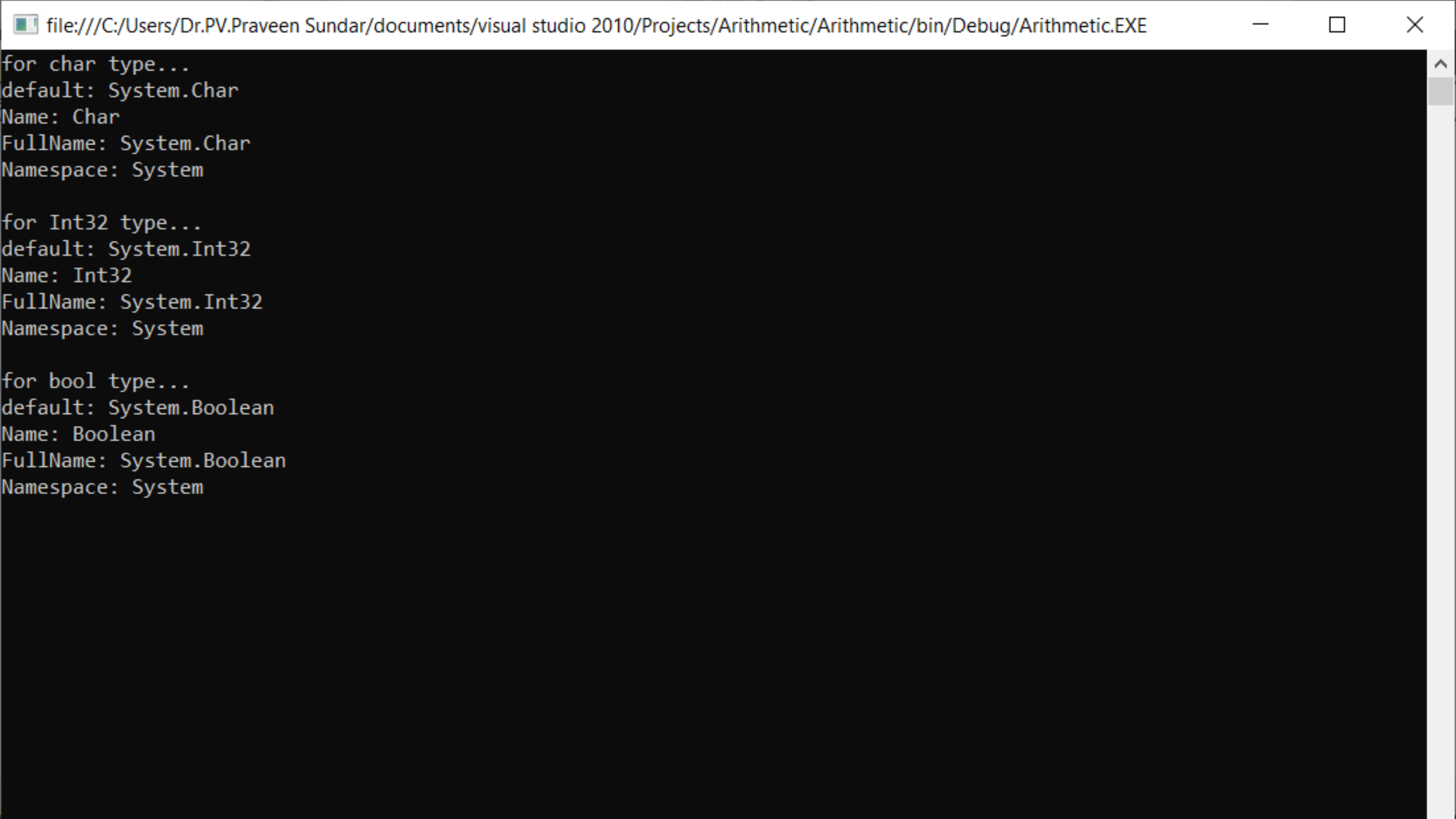
- ❑ The **is** operator is used to check if the run-time type of an object is compatible with the given type or not, whereas the **as** operator is used to perform conversion between compatible reference types or nullable types.
- ❑ The **is** operator is of Boolean type, whereas the **as** operator is not.
- ❑ The **is** operator returns true if the given object is of the same type, whereas the **as** operator returns the object when they are compatible with the given type.
- ❑ The **is** operator returns false if the given object is not of the same type, whereas the **as** operator returns null if the conversion is not possible.
- ❑ The **is** operator is used for only reference, boxing, and unboxing conversions, whereas the **as** operator is used only for nullable, reference, and boxing conversions.

typeof operator

94

- ❑ `typeof()` is an operator in C#, it is used to get the type (system type) of with class name of a given type.
- ❑ By using `typeof()` operator, we can get the name of the type, namespace name. It works with only compile-time known types.
- ❑ `typeof()` operator does not work with the variables or instances. If you want to get the type of a variable, you can use `GetType()` method.
- ❑ There are main 3 properties to get the details about the type:
 - ▣ `typeof(type).Name` or `this.GetType().Name` – It returns the class name only.
 - ▣ `typeof(type).FullName` or `this.GetType().FullName` – It returns the class name along with the namespace.
 - ▣ `typeof(type).Namespace` or `this.GetType().Namespace` – It returns the namespace only.

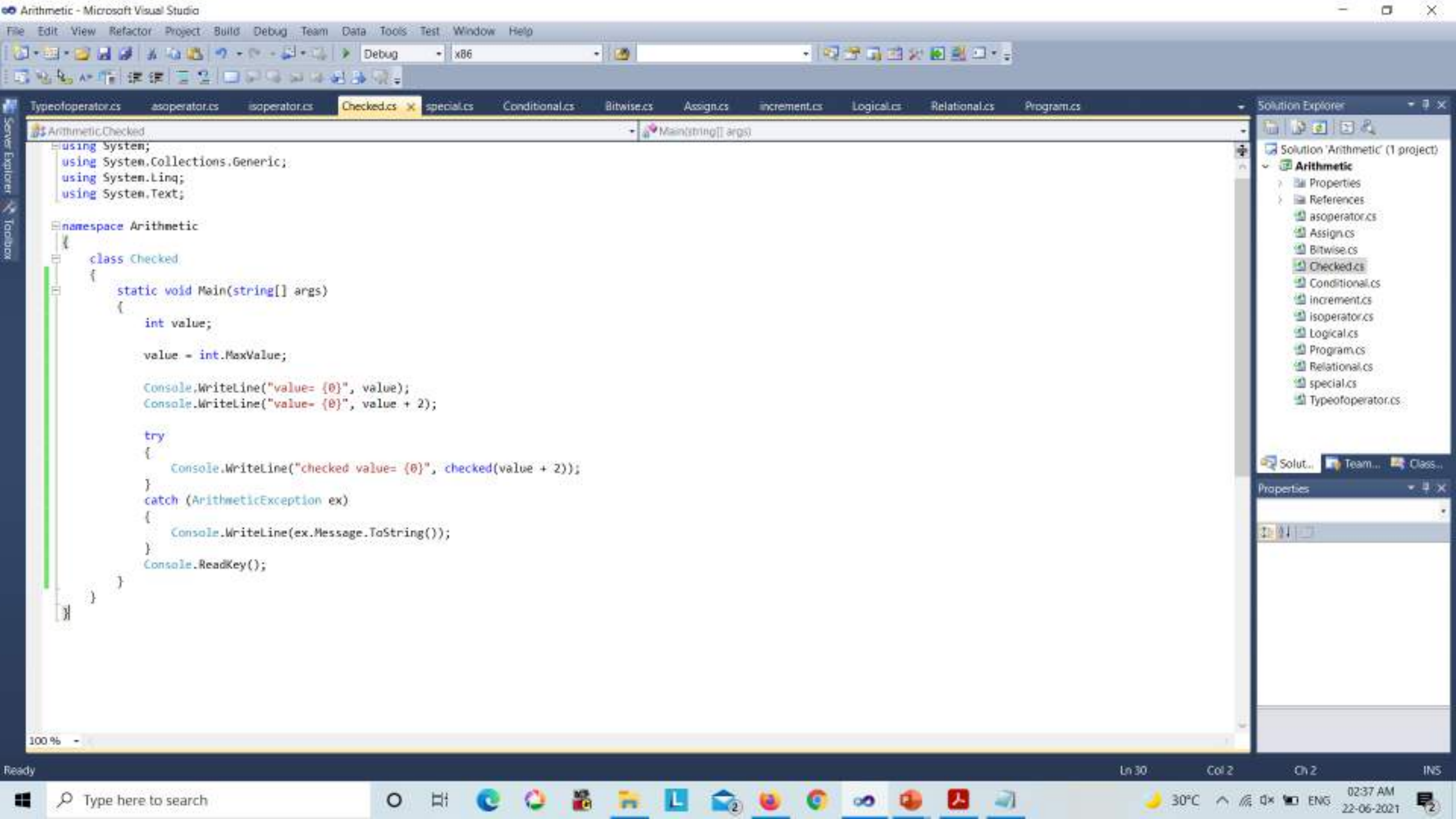




Checked and unchecked operator

97

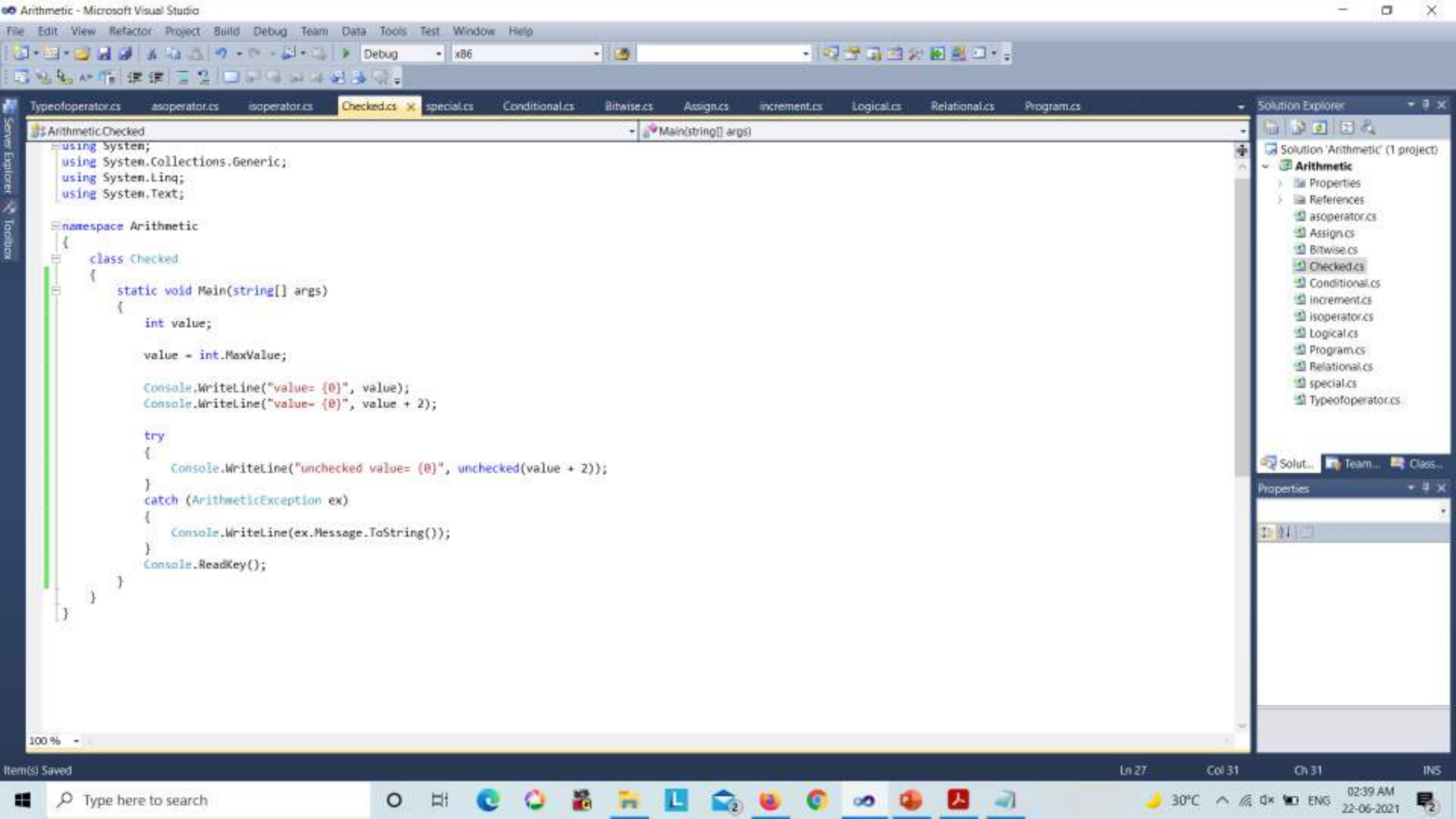
- ❑ C# provides checked and unchecked keyword to handle integral type exceptions.
- ❑ Checked and unchecked keywords specify checked context and unchecked context respectively.
- ❑ In checked context, arithmetic overflow raises an exception whereas, in an unchecked context, arithmetic overflow is ignored and result is truncated.
- ❑ The `checked` keyword is used to explicitly check overflow and conversion of integral type values at compile time.
- ❑ The `Unchecked` keyword ignores the integral type arithmetic exceptions. It does not check explicitly and produce result that may be truncated or wrong.

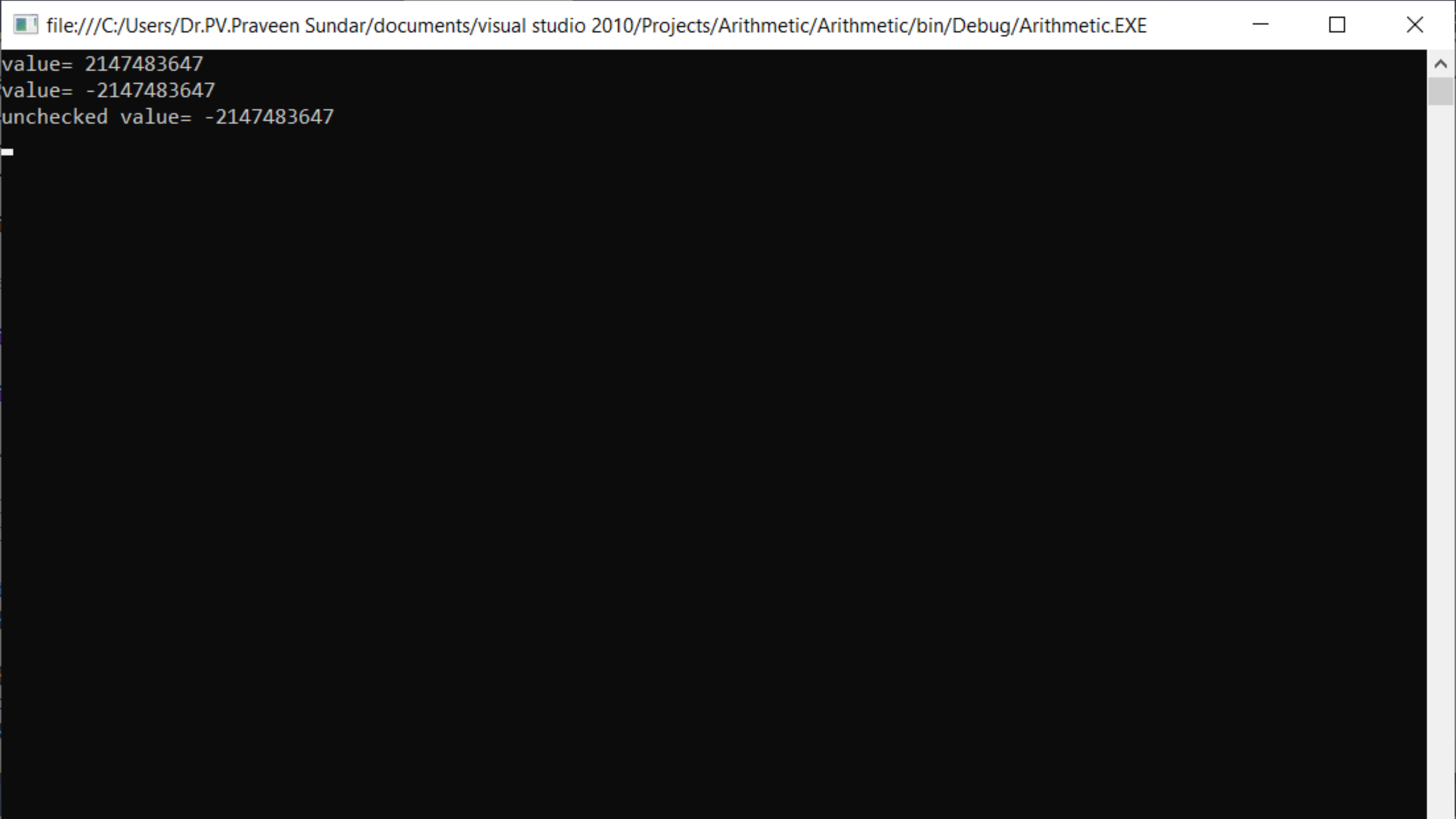


value= 2147483647

value= -2147483647

Arithmetic operation resulted in an overflow.





value= 2147483647

value= -2147483647

unchecked value= -2147483647

Thank you



EXPRESSIONS IN C#

Dr P.V. Praveen Sundar
Assistant Professor,
Department of Computer Science
Adhiparasakthi College of Arts & Science,
Kalavai.

Expressions

2

- ❑ The simplest C# expressions are literals (for example, integer and real numbers) and names of variables. We can combine them into complex expressions by using operators.
- ❑ An expression in C# is a combination of operands (variables, literals, method calls) and operators that can be evaluated to a single value. To be precise, an expression must have at least one operand but may not have any operator.
- ❑ Typically, an expression produces a result and can be included in another expression.

- The expressions are classified on based on the operations are
 - ▣ Primary Expression
 - ▣ Unary Expression
 - ▣ Arithmetic Expression
 - ▣ String Concatenation Expression
 - ▣ Relational Expression
 - ▣ Logical Expression
 - ▣ Conditional Expression
 - ▣ Assignment Expression
 - ▣ Constant Expression
 - ▣ Boolean Expression
 - ▣ Type Conversion Expression.
- In the following code, examples of expressions are at the right-hand side of assignments:



Program.cs

Expressions.Program

Main()

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Expressions
{
    class Program
    {
        static void Main()
        {
            int a, b, c;
            a = 7;
            b = a;
            c = b++;
            b = a + b * c;
            c = a >= 100 ? b : c / 10;
            a = (int)Math.Sqrt(b * b + c * c);

            string s = "String literal";
            char l = s[s.Length - 1];
        }
    }
}
```

Solution Explorer

Solution 'Expressions' (1 project)

- Expressions
 - Properties
 - References
 - Program.cs

Solut... Team... Class...

Properties

Primary Expression

5

- ❑ Literals or Constants.
- ❑ Variable names.
- ❑ Expression written within parentheses.
- ❑ Method access using dot notation.
- ❑ Method Invocation.
- ❑ Array element access.
- ❑ This access
- ❑ Postfix increment expression
- ❑ Postfix decrement expression
- ❑ Expressions using new operator
- ❑ Expressions using typeof operator
- ❑ Expressions using sizeof operator.
- ❑ Checked expression
- ❑ Unchecked expression.

Unary Expression

6

- Unary Plus Expressions
- Unary Minus Expressions
- Unary Not(!) Expressions
- Unary one's Complement Expressions. (~)
- Unary * Expressions
- Unary & Expressions
- Prefix Increment Expressions. (++)
- Prefix Decrement Expressions (--)
- Cast Expressions.(Cast Operator)

Interpolated String Expression:

7

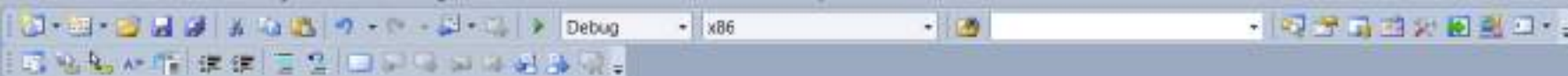
- ❑ The \$ special character identifies a string literal as an interpolated string.
- ❑ An interpolated string is a string literal that might contain interpolation expressions.
- ❑ When an interpolated string is resolved to a result string, items with interpolation expressions are replaced by the string representations of the expression results.
- ❑ String interpolation provides a more readable and convenient syntax to create formatted strings than a string composite formatting feature.

```
var r = 2.3;  
var message = $"The area of a circle with radius {r} is {Math.PI * r * r:F3}.";   
Console.WriteLine(message);  
// Output:  
// The area of a circle with radius 2.3 is 16.619.
```

Lambda Expressions

8

- ❑ lambda expression is used to create an anonymous function.
- ❑ Use the lambda declaration operator `=>` to separate the lambda's parameter list from its body.
- ❑ A lambda expression can be of any of the following two forms:
 - ❑ Expression lambda that has an expression as its body:
 - ❑ Statement lambda that has a statement block as its body:
- ❑ To create a lambda expression, we must specify input parameters (if any) on the left side of the lambda operator and an expression or a statement block on the other side.



Program.cs

Expressions.Program

Main()

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Expressions
{
    class Program
    {
        static void Main()
        {
            int[] numbers = { 2, 3, 4, 5 };
            var maximumSquare = numbers.Max(x => x * x);
            Console.WriteLine(maximumSquare);
            Console.ReadKey();
        }
    }
}
```

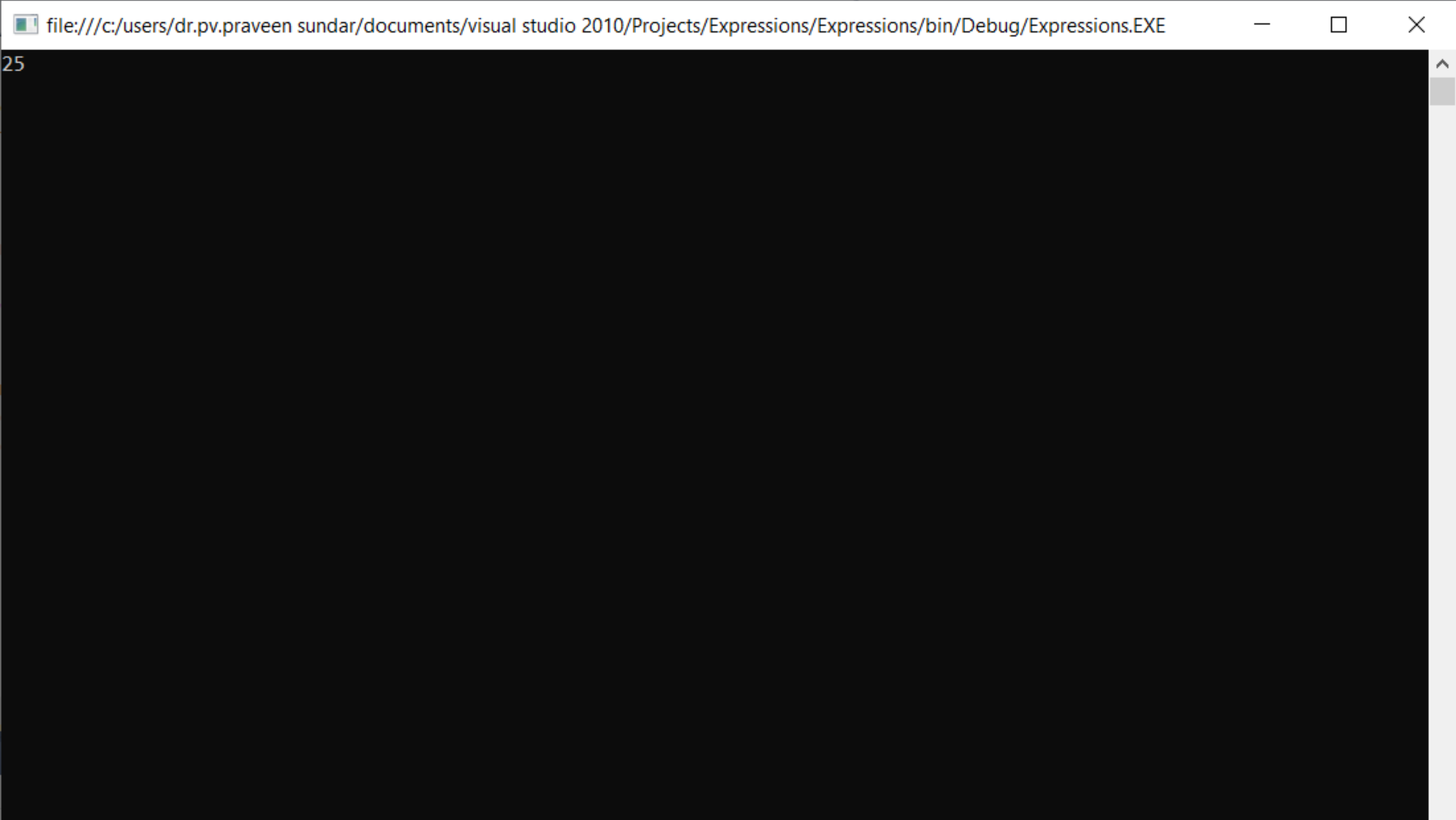
Solution Explorer

Solution 'Expressions' (1 project)

- Expressions
 - Properties
 - References
 - Program.cs

Solut... Team... Class...

Properties



Type Conversion

11

- Type conversion is converting one type of data to another type. It is also known as Type Casting.
- In C#, type casting has two forms –
 - ▣ Implicit type conversion – These conversions are performed by C# in a type-safe manner. For example, are conversions from smaller to larger integral types and conversions from derived classes to base classes.
 - ▣ Explicit type conversion – These conversions are done explicitly by users using the pre-defined functions. Explicit conversions require a cast operator.

Implicit numeric conversions

The following table shows the predefined implicit conversions between the built-in numeric types:

From	To
sbyte	short, int, long, float, double, decimal, Or nint
byte	short, ushort, int, uint, long, ulong, float, double, decimal, nint, Or nuint
short	int, long, float, double, Or decimal, Or nint
ushort	int, uint, long, ulong, float, double, Or decimal, nint, Or nuint
int	long, float, double, Or decimal, nint
uint	long, ulong, float, double, Or decimal, Or nuint
long	float, double, Or decimal
ulong	float, double, Or decimal
float	double
nint	long, float, double, Or decimal
nuint	ulong, float, double, Or decimal

- Any integral numeric type is implicitly convertible to any floating-point numeric type.
- There are no implicit conversions to the byte and sbyte types. There are no implicit conversions from the double and decimal types.
- There are no implicit conversions between the decimal type and the float or double types.
- A value of a constant expression of type int (for example, a value represented by an integer literal) can be implicitly converted to sbyte, byte, short, ushort, uint, ulong, nint, or nuint, if it's within the range of the destination type:

Explicit numeric conversions

The following table shows the predefined explicit conversions between the built-in numeric types for which there is no [implicit conversion](#):

From	To
<code>sbyte</code>	<code>byte</code> , <code>ushort</code> , <code>uint</code> , <code>ulong</code> , <code>nuint</code>
<code>byte</code>	<code>sbyte</code>
<code>short</code>	<code>sbyte</code> , <code>byte</code> , <code>ushort</code> , <code>uint</code> , <code>ulong</code> , <code>nuint</code>
<code>ushort</code>	<code>sbyte</code> , <code>byte</code> , <code>short</code>
<code>int</code>	<code>sbyte</code> , <code>byte</code> , <code>short</code> , <code>ushort</code> , <code>uint</code> , <code>ulong</code> , <code>nuint</code>
<code>uint</code>	<code>sbyte</code> , <code>byte</code> , <code>short</code> , <code>ushort</code> , <code>int</code>
<code>long</code>	<code>sbyte</code> , <code>byte</code> , <code>short</code> , <code>ushort</code> , <code>int</code> , <code>uint</code> , <code>ulong</code> , <code>nint</code> , <code>nuint</code>
<code>ulong</code>	<code>sbyte</code> , <code>byte</code> , <code>short</code> , <code>ushort</code> , <code>int</code> , <code>uint</code> , <code>long</code> , <code>nint</code> , <code>nuint</code>
<code>float</code>	<code>sbyte</code> , <code>byte</code> , <code>short</code> , <code>ushort</code> , <code>int</code> , <code>uint</code> , <code>long</code> , <code>ulong</code> , <code>decimal</code> , <code>nint</code> , <code>nuint</code>
<code>double</code>	<code>sbyte</code> , <code>byte</code> , <code>short</code> , <code>ushort</code> , <code>int</code> , <code>uint</code> , <code>long</code> , <code>ulong</code> , <code>float</code> , <code>decimal</code> , <code>nint</code> , <code>nuint</code>
<code>decimal</code>	<code>sbyte</code> , <code>byte</code> , <code>short</code> , <code>ushort</code> , <code>int</code> , <code>uint</code> , <code>long</code> , <code>ulong</code> , <code>float</code> , <code>double</code> , <code>nint</code> , <code>nuint</code>

S.No	Method	Description
1	ToBoolean	Converts a type to a Boolean value, where possible.
2	ToByte	Converts a type to a byte.
3	ToChar	Converts a type to a single Unicode character, where possible.
4	ToDateTime	Converts a type (integer or string type) to date-time structures.
5	ToDecimal	Converts a floating point or integer type to a decimal type.
6	ToDouble	Converts a type to a double type.
7	ToInt16	Converts a type to a 16-bit integer.
8	ToInt32	Converts a type to a 32-bit integer.
9	ToInt64	Converts a type to a 64-bit integer.
10	ToSbyte	Converts a type to a signed byte type.
11	ToSingle	Converts a type to a small floating point number.
12	ToString	Converts a type to a string.
13	ToType	Converts a type to a specified type.
14	ToUInt16	Converts a type to an unsigned int type.
15	ToUInt32	Converts a type to an unsigned long type.
16	ToUInt64	Converts a type to an unsigned big integer.

namespace Expressions

{
class Program

{

static void Main()

{

int value;

short value2;

float value3;

long value4;

Console.WriteLine("Enter the Value");

value = Convert.ToInt16(Console.ReadLine()); // Implicit conversion

Console.WriteLine("Value is " + value);

value2 = (short) (value*value); // Explicit Conversion

Console.WriteLine("Value2 is " + value2);

value4 = value * value; // Implicit Conversion

Console.WriteLine("Value2 is " + value4);

value3 = value2; // Implicit Conversion

Console.WriteLine("Value3 is " + value3);

value3 = value4; // Implicit Conversion

Console.WriteLine("Value3 is " + value3);

value3 = 3.1413f;

value2 = (short)value3; // Explicit Conversion.

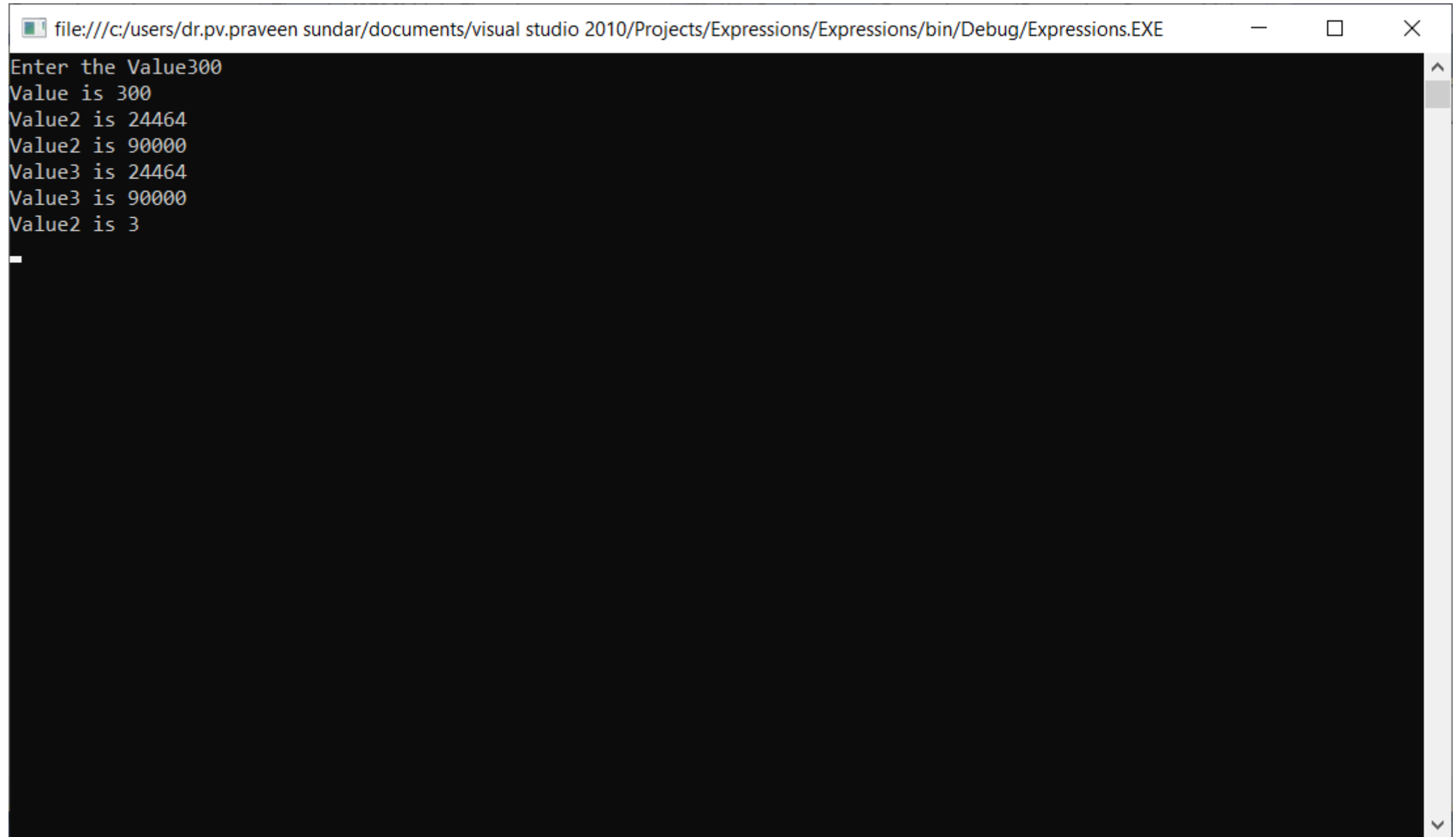
Console.WriteLine("Value2 is " + value2);

Console.ReadKey();

}

}

}



```
file:///c:/users/dr.pv.praveen sundar/documents/visual studio 2010/Projects/Expressions/Expressions/bin/Debug/Expressions.EXE
Enter the Value300
Value is 300
Value2 is 24464
Value2 is 90000
Value3 is 24464
Value3 is 90000
Value2 is 3
_
```

Operator Precedence & Associativity

18

- ❑ Operator precedence and associativity determine the order in which the operations in an expression are performed.
- ❑ In an expression with multiple operators, the operators with higher precedence are evaluated before the operators with lower precedence.
- ❑ When operators have the same precedence, associativity of the operators determines the order in which the operations are performed:
 - ▣ Left-associative operators are evaluated in order from left to right. Except for the assignment operators, all binary operators are left-associative.
 - ▣ Right-associative operators are evaluated in order from right to left. The assignment operators and the conditional operator `?:` are right-associative.
- ❑ We can use parentheses to change the order of evaluation imposed by operator precedence and associativity.
- ❑ The following table lists the C# operators starting with the highest precedence to the lowest. The operators within each row have the same precedence.

Category	Operator	Associativity
Postfix	() [] -> . ++ --	Left to right
Unary	+ - ! ~ ++ -- (type)* & sizeof	Right to left
Multiplicative	* / %	Left to right
Additive	+ -	Left to right
Shift	<< >>	Left to right
Relational	< <= > >=	Left to right
Equality	== !=	Left to right
Bitwise AND	&	Left to right
Bitwise XOR	^	Left to right
Bitwise OR		Left to right
Logical AND	&&	Left to right
Logical OR		Left to right
Conditional	?:	Right to left
Assignment	= += -= *= /= %= >>= <<= &= ^= =	Right to left
Comma	,	Left to right

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Expressions
{
    class Operator
    {
        public static void Main()
        {
            int a, b, c, ans;

            Console.Write(" Enter the Value of A :");
            a = Convert.ToInt32(Console.ReadLine());
            Console.Write(" Enter the Value of B :");
            b = Convert.ToInt32(Console.ReadLine());
            Console.Write(" Enter the Value of C :");
            c = Convert.ToInt32(Console.ReadLine());

            // First Method
            ans = a - b / 3 + c * 2 - 1;
            Console.WriteLine("The result of the Expression (a - b / 3 + c * 2 - 1;) is {0}", ans);

            // Second Method
            ans = a - b / (3 + c) * (2 - 1);
            Console.WriteLine("The result of the Expression (a - b / (3 + c) * (2 - 1);) is {0}", ans);

            // Third Method
            ans = a - (b / (3 + c) * 2) - 1;
            Console.WriteLine("The result of the Expression (a - (b / (3 + c) * 2) - 1;) is {0}", ans);

            // Fourth Method
            ans = a - ((b / 3) + c * 2) - 1;
            Console.WriteLine("The result of the Expression (a - ((b / 3) + c * 2) - 1;) is {0}", ans);

            Console.ReadKey();
        }
    }
}
```

```
Enter the Value of A :9
Enter the Value of B :12
Enter the Value of C :3
The result of the Expression (a - b / 3 + c * 2 - 1;) is 10
The result of the Expression (a - b / (3 + c) * (2 - 1);) is 7
The result of the Expression (a - (b / (3 + c) * 2) - 1;) is 4
The result of the Expression (a - ((b / 3) + c * 2) - 1;) is -2
```



```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
namespace Expressions
{
    class MathFunctions
    {
        public static void Main()
        {
            int theta, ans;
            double value;

            Console.Write(" Enter the degree value :");
            theta = Convert.ToInt32(Console.ReadLine());
            value = (theta * Math.PI / 180);
            Math.DivRem(256, 144, out ans);

            Console.WriteLine(" Radian Value of ({0})= {1}", theta, value);
            Console.WriteLine(" Abs({0})= {1}", theta, Math.Abs(-value));
            Console.WriteLine(" floor({0})= {1}", theta, Math.Floor(value));
            Console.WriteLine(" Ceiling({0})= {1}", theta, Math.Ceiling(value));
            Console.WriteLine(" Mul(100*200) = {0}", Math.BigMul(100, 200));
            Console.WriteLine(" DivRem(256/144) = {0}", ans);
            Console.WriteLine(" The Value of E = {0}", Math.E);

            Console.WriteLine("\n Sin ({0})= {1}", theta, Math.Round(Math.Sin(value), 4));
            Console.WriteLine(" Cos ({0})= {1}", theta, Math.Round(Math.Cos(value), 4));
            Console.WriteLine(" Tan ({0})= {1}", theta, Math.Round(Math.Tan(value), 4));
            Console.WriteLine(" Sinh ({0})= {1}", theta, Math.Round(Math.Sinh(value), 4));
            Console.WriteLine(" Cosh ({0})= {1}", theta, Math.Round(Math.Cosh(value), 4));
            Console.WriteLine(" Tanh ({0})= {1}", theta, Math.Round(Math.Tanh(value), 4));
            Console.WriteLine(" ASin ({0})= {1}", theta, Math.Round(Math.Asin(value), 4));
            Console.WriteLine(" ACos ({0})= {1}", theta, Math.Round(Math.Acos(value), 4));
            Console.WriteLine(" ATan ({0})= {1}", theta, Math.Round(Math.Atan(value), 4));
            Console.WriteLine(" ATan2 ({0})= {1}", theta, Math.Round(Math.Atan2(value, 10), 4));

            Console.ReadKey();
        }
    }
}
```

Enter the degree value :45
Radian Value of (45)= 0.785398163397448
Abs(45)= 0.785398163397448
floor(45)= 0
Ceiling(45)= 1
Mul(100*200) = 20000
DivRem(256/144) = 112
The Value of E = 2.71828182845905

Sin (45)= 0.7071
Cos (45)= 0.7071
Tan (45)= 1
Sinh (45)= 0.8687
Cosh (45)= 1.3246
Tanh (45)= 0.6558
ASin (45)= 0.9033
ACos (45)= 0.6675
ATan (45)= 0.6658
ATan2 (45)= 0.0784

Control Statements

24

- The control statements are used to control the flow of execution of the program.
- If we want to execute a specific block of instructions only when a certain condition is true, then control statements are useful.
- If we want to execute a block repeatedly, then loops are useful.
- C# classifies these control statements into two categories
 - ▣ Conditional execution
 - ▣ Unconditional execution

Control Flow statements

```
graph TD; A[Control Flow statements] --> B[Conditional]; A --> C[Unconditional]; B --> D[Selection]; B --> E[Iteration]; D --> F[if]; D --> G[If-else]; D --> H[If-else-if]; D --> I[switch]; E --> J[while]; E --> K[Do-while]; E --> L[for]; E --> M[foreach]; C --> N[Jump]; N --> O[break]; N --> P[continue]; N --> Q[goto]; N --> R[throw]; N --> S[return];
```

Conditional

Selection

if

If-else

If-else-if

switch

Iteration

while

Do-
while

for

foreach

Unconditional

Jump

break

continue

goto

throw

return

Simple if

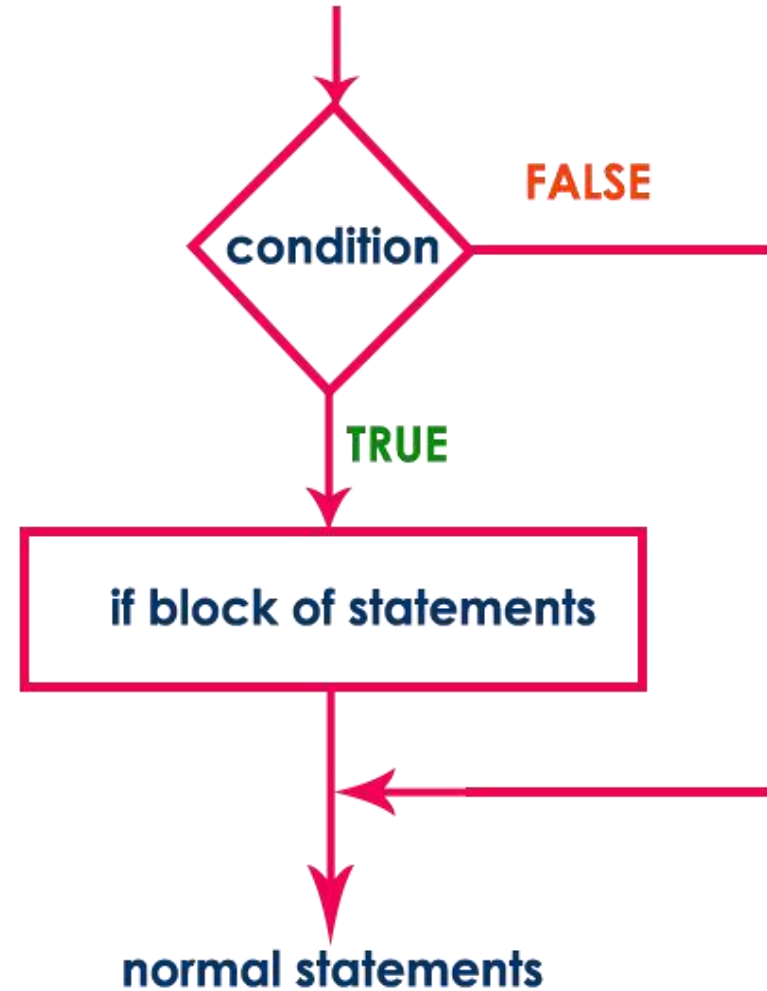
26

- ❑ Simple if statement is used to verify the given condition and executes the block of statements based on the condition result.
- ❑ The simple if statement evaluates specified condition.
- ❑ If it is **TRUE**, it executes the next statement or block of statements.
- ❑ If the condition is **FALSE**, it skips the execution of the next statement or block of statements.
- ❑ Simple if statement is used when we have only one option that is executed or skipped based on a condition.
- ❑ The general syntax and execution flow of the simple if statement is as follows.

Syntax

```
if ( condition )  
{  
    ....  
    block of statements;  
    ....  
}
```

Execution flow diagram



Program.cs · X

ControlProgram

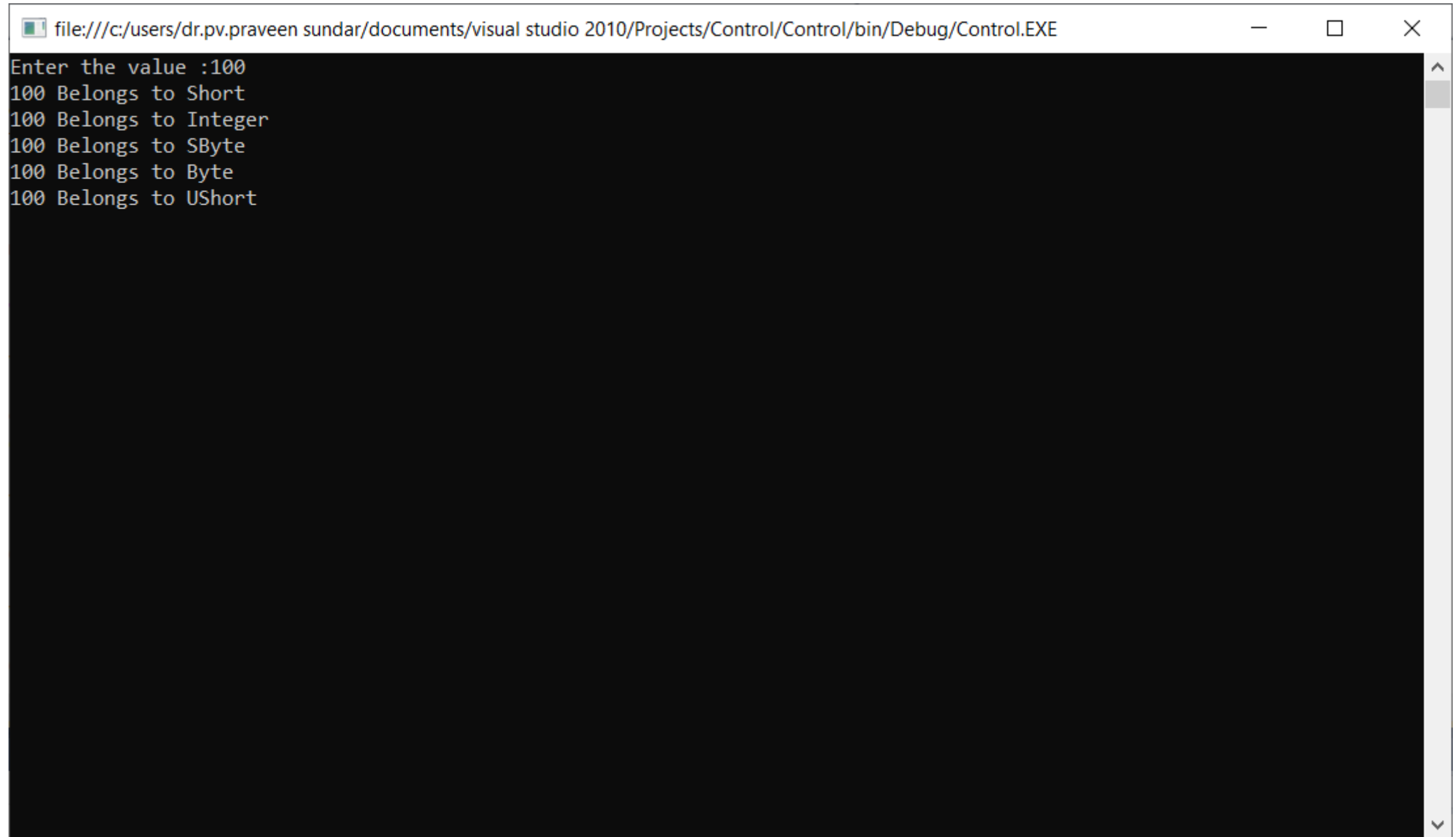
Main()

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

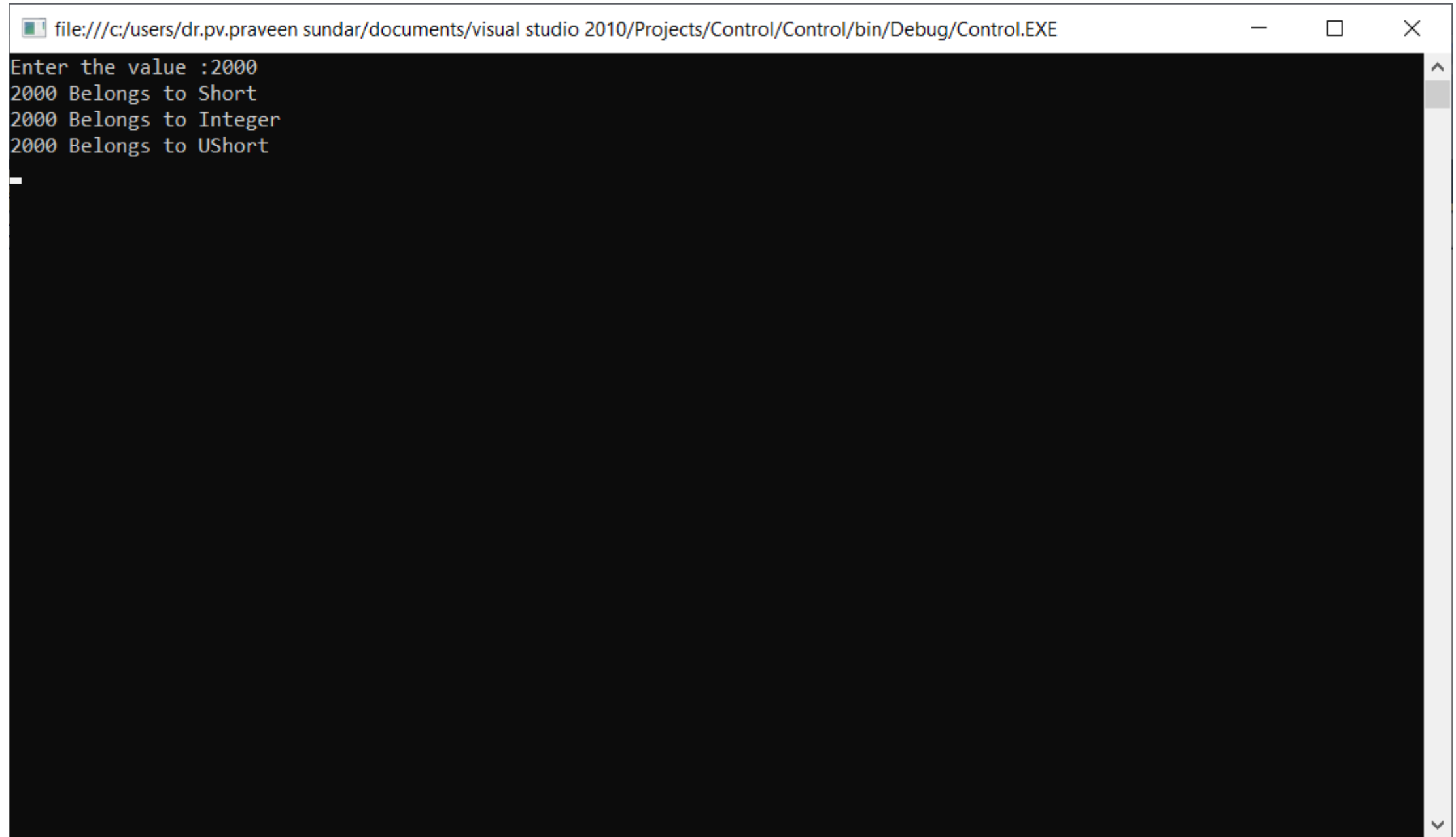
namespace Control
{
    class Program
    {
        static void Main()
        {
            int value;

            Console.Write("Enter the value :");
            value = Convert.ToInt32(Console.ReadLine());

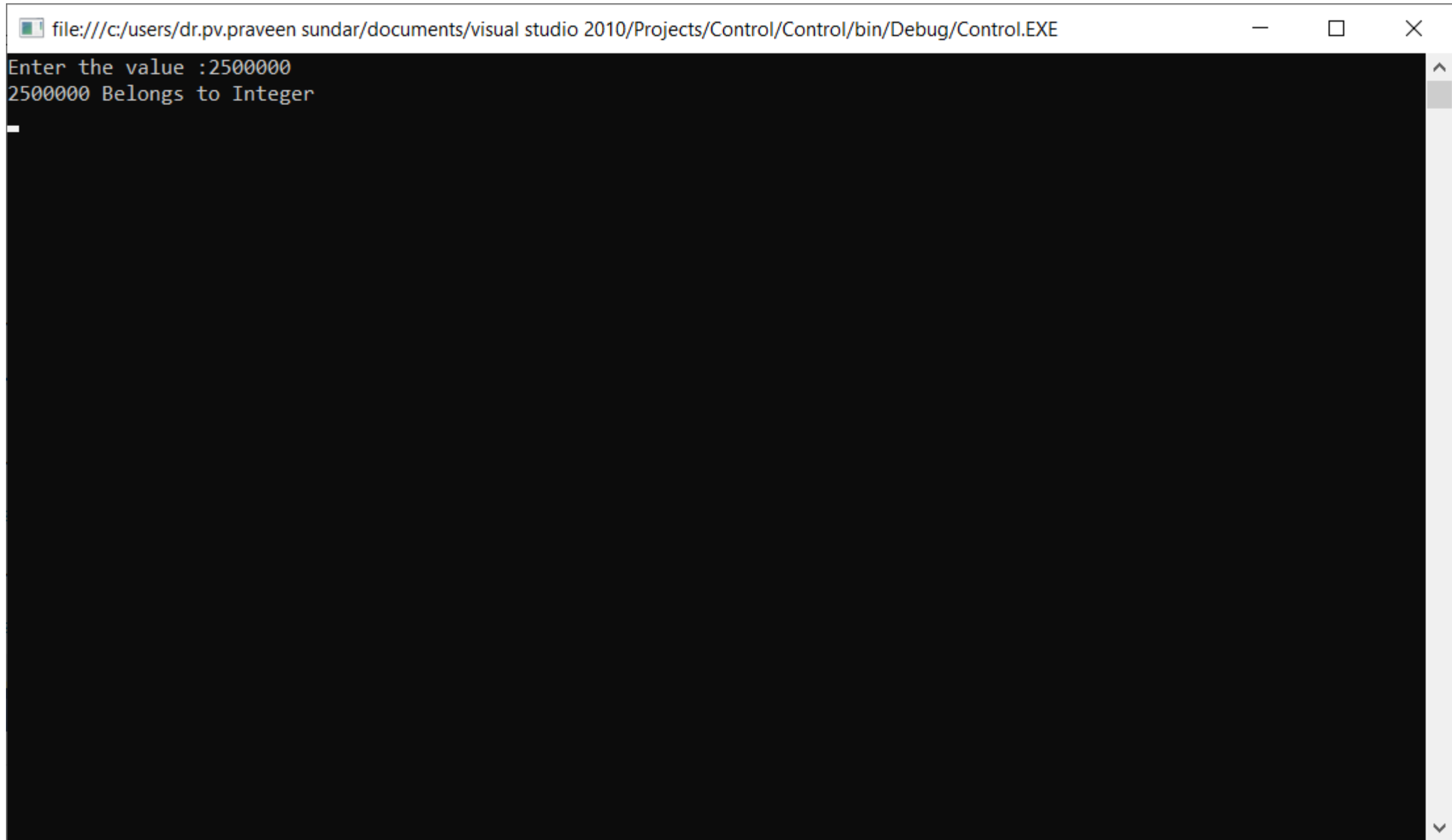
            if (value < short.MaxValue)
            {
                Console.WriteLine("{0} Belongs to Short", value);
            }
            if (value < int.MaxValue)
            {
                Console.WriteLine("{0} Belongs to Integer", value);
            }
            if (value < sbyte.MaxValue)
            {
                Console.WriteLine("{0} Belongs to SByte", value);
            }
            if (value < byte.MaxValue)
            {
                Console.WriteLine("{0} Belongs to Byte", value);
            }
            if (value < ushort.MaxValue)
            {
                Console.WriteLine("{0} Belongs to UShort", value);
            }
            Console.ReadKey();
        }
    }
}
```



```
file:///c:/users/dr.pv.praveen sundar/documents/visual studio 2010/Projects/Control/Control/bin/Debug/Control.EXE
Enter the value :100
100 Belongs to Short
100 Belongs to Integer
100 Belongs to SByte
100 Belongs to Byte
100 Belongs to UShort
```

```
file:///c:/users/dr.pv.praveen sundar/documents/visual studio 2010/Projects/Control/Control/bin/Debug/Control.EXE
Enter the value :2000
2000 Belongs to Short
2000 Belongs to Integer
2000 Belongs to UShort
_
```



```
file:///c:/users/dr.pv.praveen sundar/documents/visual studio 2010/Projects/Control/Control/bin/Debug/Control.EXE
Enter the value :2500000
2500000 Belongs to Integer
_
```

If-else statement

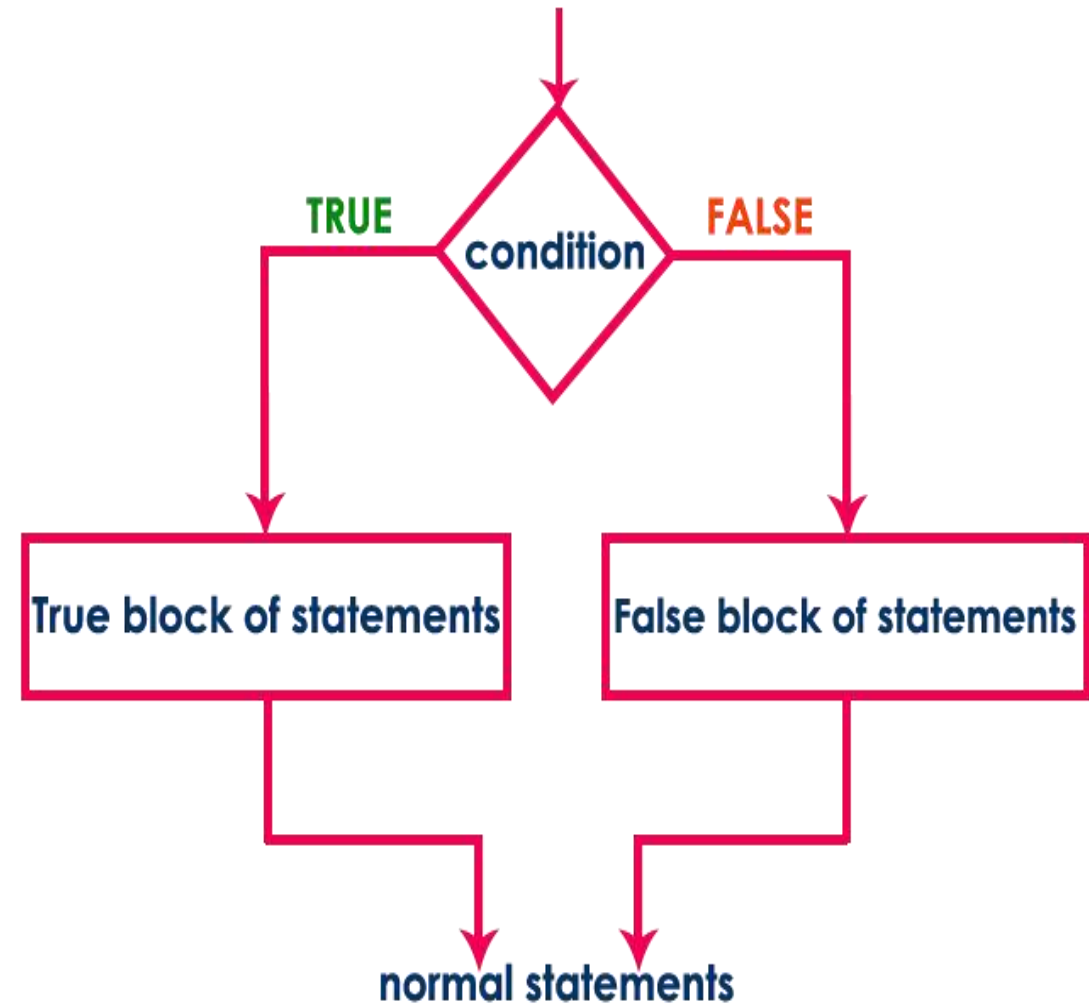
32

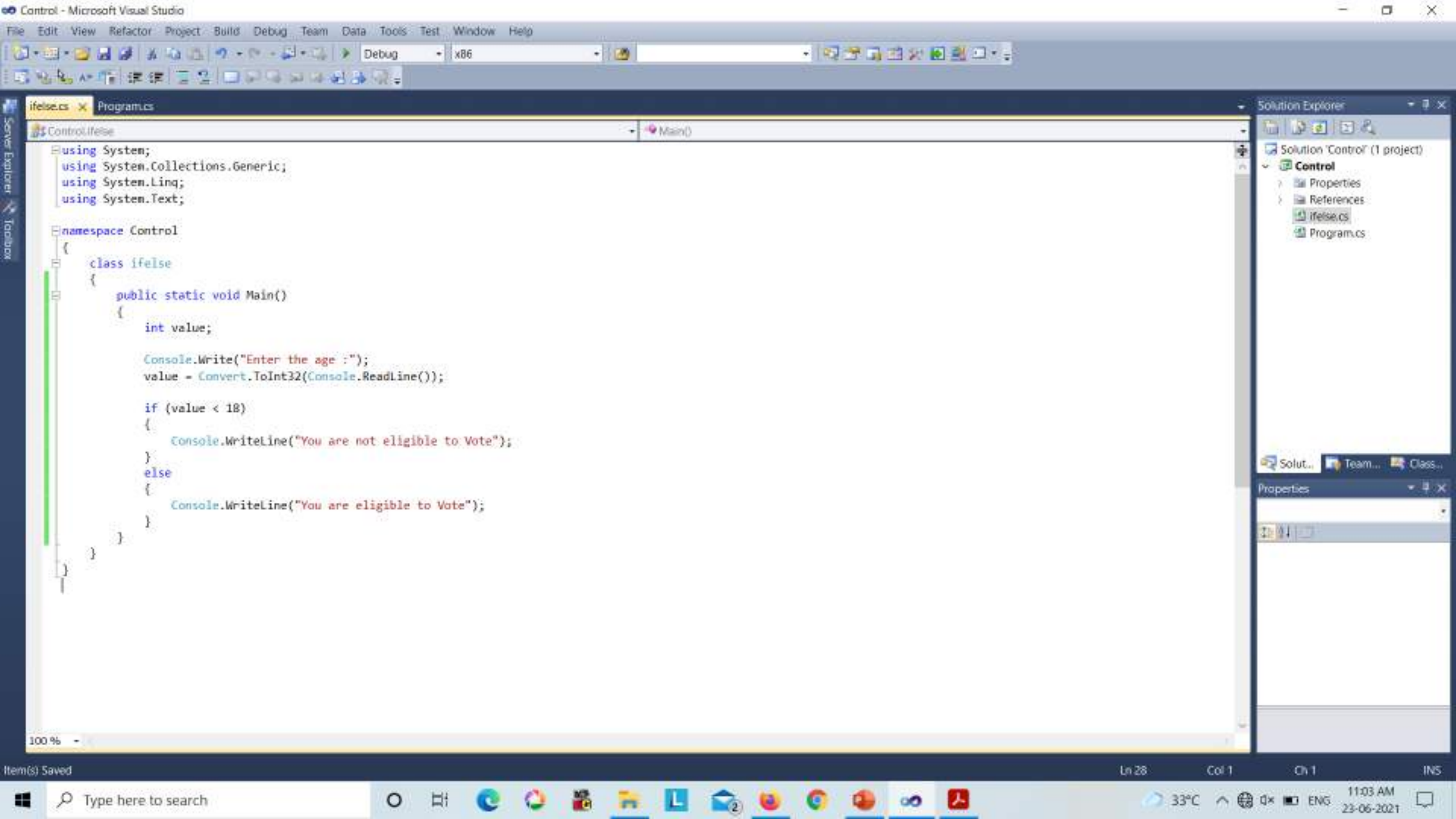
- ❑ The if-else statement is used to verify the given condition and executes only one out of the two blocks of statements based on the condition result.
- ❑ The if-else statement evaluates the specified condition.
- ❑ If it is TRUE, it executes a block of statements (True block).
- ❑ If the condition is FALSE, it executes another block of statements (False block).
- ❑ The if-else statement is used when we have two options and only one option has to be executed based on a condition result (TRUE or FALSE).
- ❑ The general syntax and execution flow of the if-else statement is as follows.

Syntax

```
if ( condition )  
{  
    ....  
    True block of statements;  
    ....  
}  
else  
{  
    ....  
    False block of statements;  
    ....  
}
```

Execution flow diagram



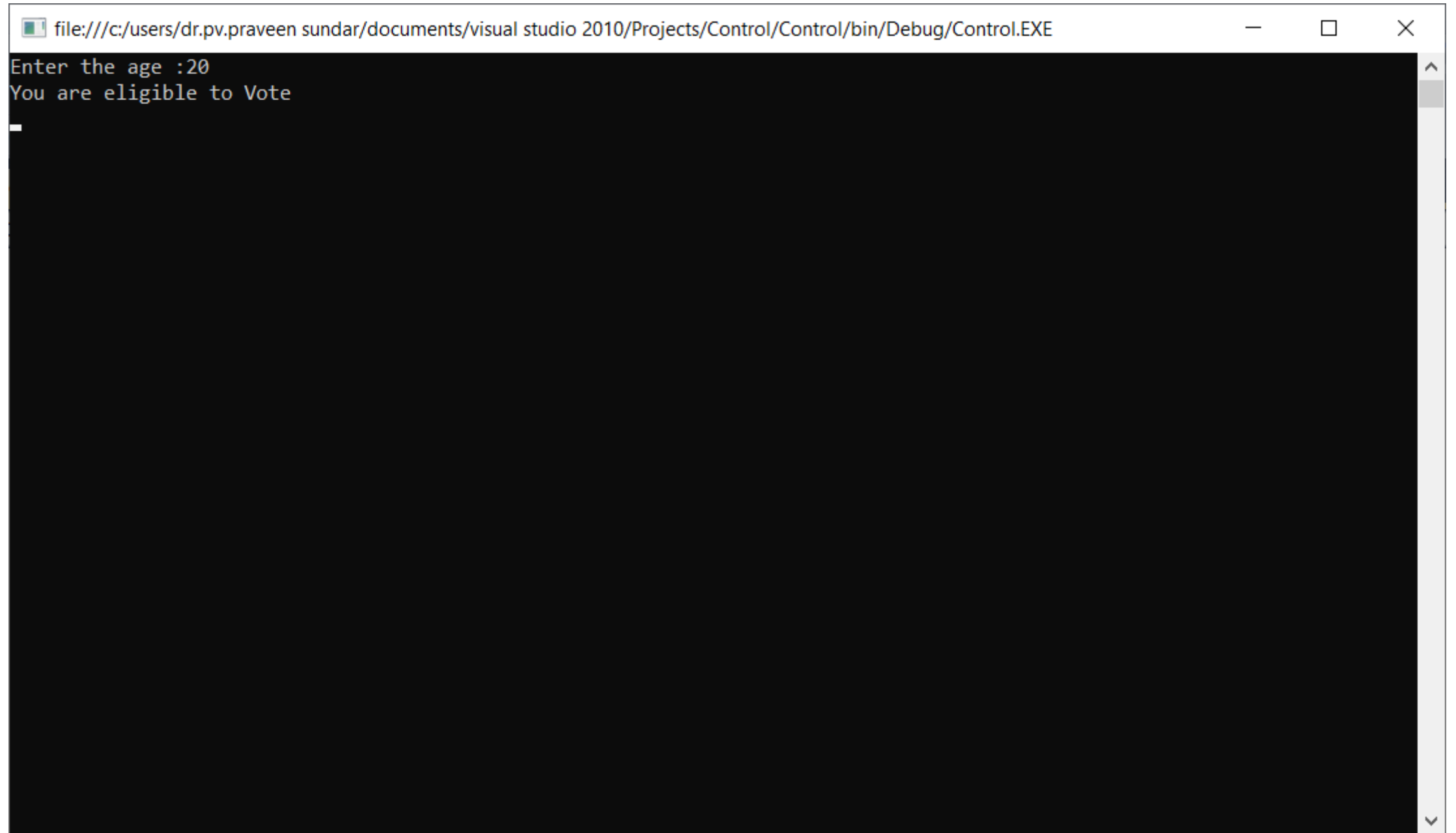


```
Control - Microsoft Visual Studio
File Edit View Refactor Project Build Debug Team Data Tools Test Window Help
Debug x86
ifelse.cs x Program.cs
Control.ifelse
Main()
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Control
{
    class ifelse
    {
        public static void Main()
        {
            int value;

            Console.Write("Enter the age :");
            value = Convert.ToInt32(Console.ReadLine());

            if (value < 18)
            {
                Console.WriteLine("You are not eligible to Vote");
            }
            else
            {
                Console.WriteLine("You are eligible to Vote");
            }
        }
    }
}
```



The image shows a Windows command prompt window with a title bar. The title bar text is "file:///c:/users/dr.pv.praveen sundar/documents/visual studio 2010/Projects/Control/Control/bin/Debug/Control.EXE". The window contains the following text:

```
Enter the age :20
You are eligible to Vote
```

The text is displayed in a monospaced font. The input "20" is on the same line as the prompt "Enter the age :". The output "You are eligible to Vote" is on the next line. A small white cursor is visible on the line following the output.

Nested if statement

36

- Writing a if statement inside another if statement is called nested if statement.
- The nested if statement can be defined using any combination of simple if & if-else statements.
- The general syntax of the nested if statement is as follows...

Syntax

```
if ( condition1 )
{
    if ( condition2 )
    {
        ....
        True block of statements 1;
    }
    ....
}
else
{
    False block of condition1;
}
```

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Control
{
    class nestedif
    {
        public static void Main()
        {
            int value;

            Console.Write("Enter the age :");
            value = Convert.ToInt32(Console.ReadLine());

            if (value > 0)
            {
                if (value > 18)
                {
                    if (value < 100)
                    {
                        Console.WriteLine("You are eligible to Vote");
                    }
                    else
                    {
                        Console.WriteLine("You are not eligible to Vote");
                    }
                }
                else
                {
                    Console.WriteLine("You are not eligible to Vote");
                }
            }
            else
            {
                Console.WriteLine("Invalid age");
            }
            Console.ReadKey();
        }
    }
}
```

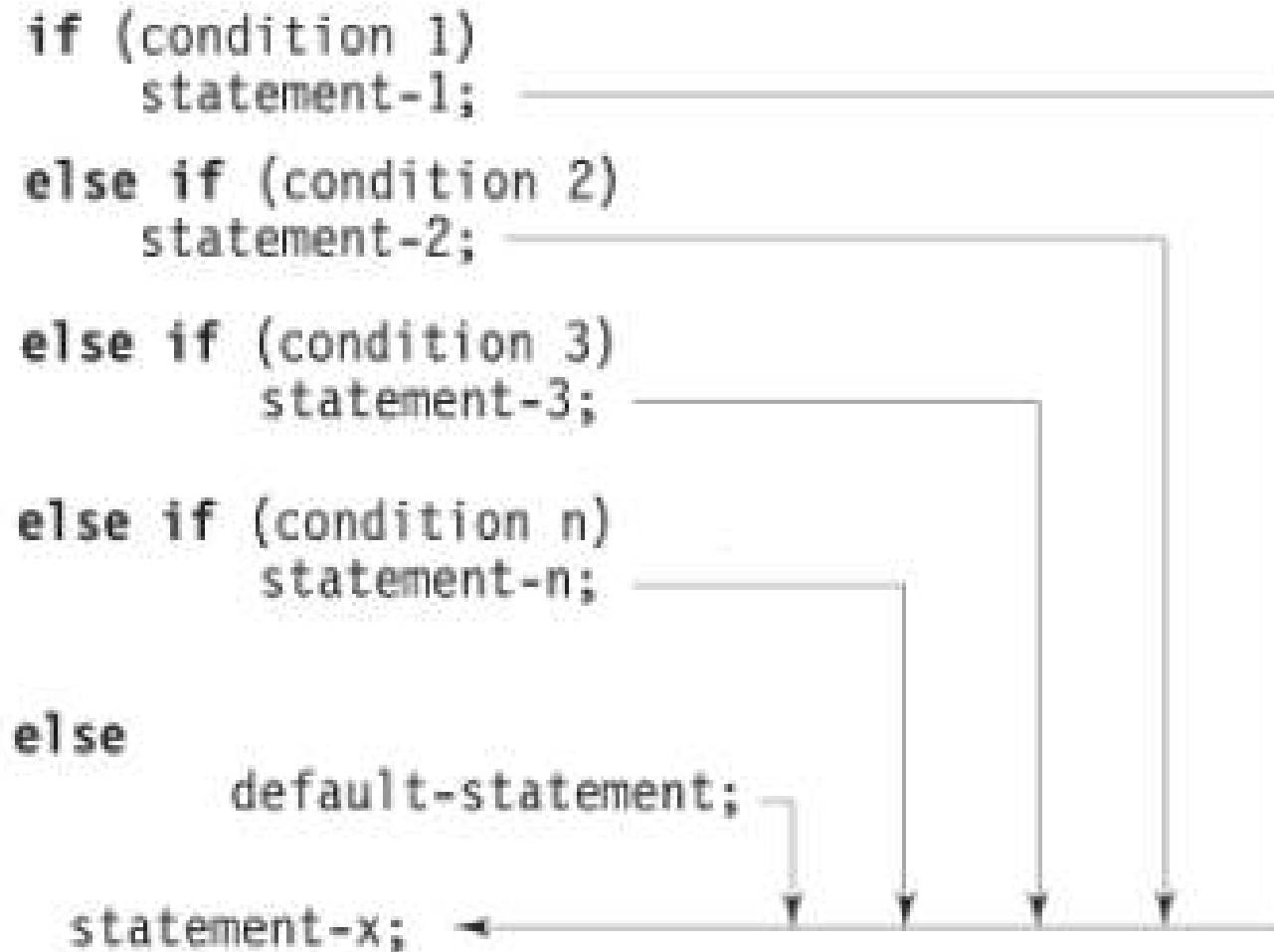

file:///c:/users/dr.pv.praveen sundar/documents/visual studio 2010/Projects/Control/Control/bin/Debug/Control.EXE

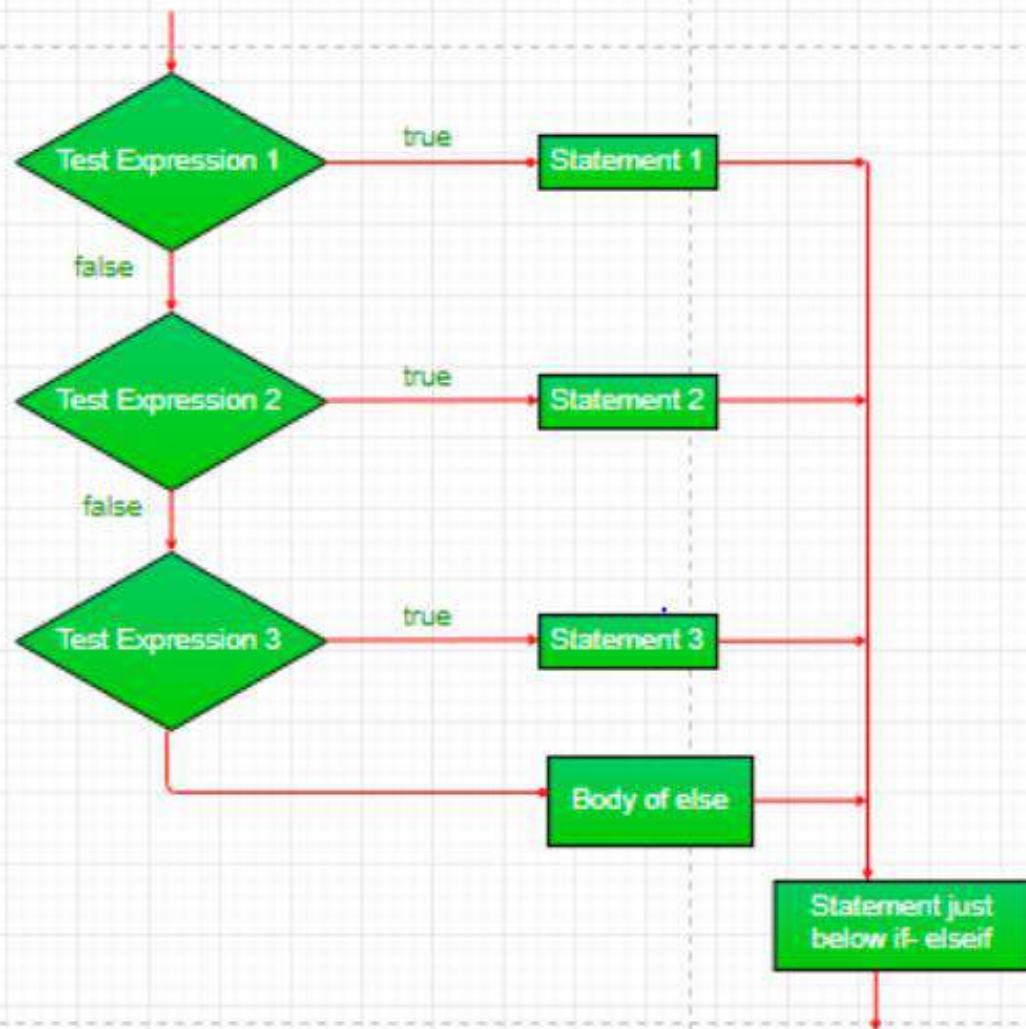
Enter the age :10
You are not eligible to Vote

if...else if...else statement

39

- ❑ The if-else-if ladder statement executes one condition from multiple statements. The execution starts from top and checked for each if condition.
- ❑ We can use multiple else if blocks to add multiple conditions but it requires atleast one if block at the beginning, we can't directly write else and else if statements without having any if block.
- ❑ The statement of if block will be executed which evaluates to be true. If none of the if condition evaluates to be true then the last else block is evaluated.
- ❑ The general syntax of the if-else-if statement is as follows...





```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Control
{
    class elseif
    {
        public static void Main()
        {
            int value;

            Console.Write("Enter the age :");
            value = Convert.ToInt32(Console.ReadLine());

            if (value < 0)
            {
                Console.WriteLine("Invalid age");
            }
            else if (value < 18)
            {
                Console.WriteLine("You are not eligible to Vote");
            }
            else if (value < 100)
            {
                Console.WriteLine("You are eligible to Vote");
            }
            else
            {
                Console.WriteLine("Invalid age");
            }

            Console.ReadKey();
        }
    }
}
```

file:///c:/users/dr.pv.praveen sundar/documents/visual studio 2010/Projects/Control/Control/bin/Debug/Control.EXE

Enter the age :20
You are eligible to Vote

Switch Statement

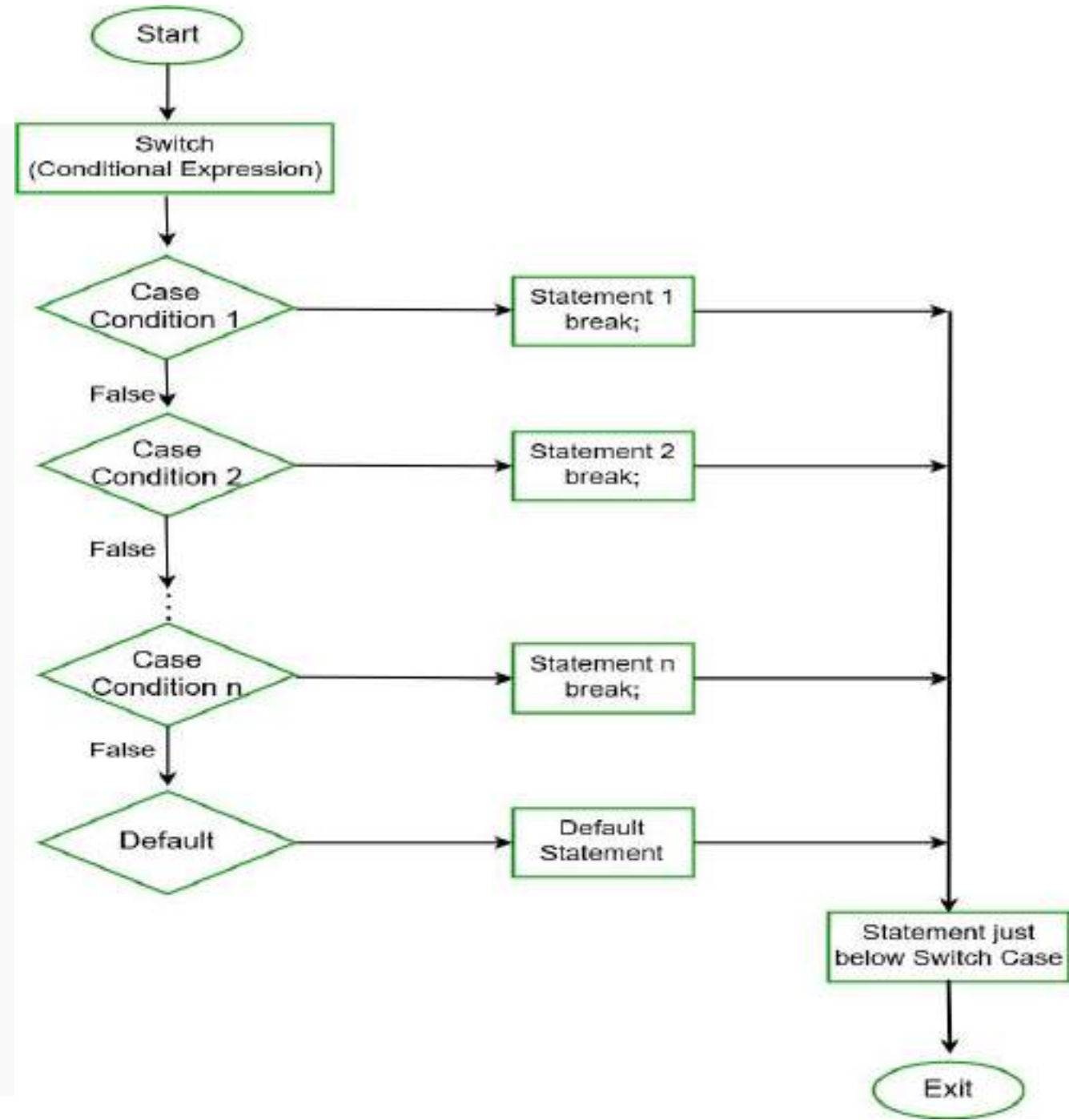
44

- ❑ In C#, Switch statement is a multiway branch statement.
- ❑ It provides an efficient way to transfer the execution to different parts of a code based on the value of the expression.
- ❑ The switch expression is of integer type such as int, char, byte, or short, or of an enumeration type, or of string type.
- ❑ The expression is checked for different cases and the one match is executed.
- ❑ The switch statement is often used as an alternative to an if-else construct if a single expression is tested against three or more conditions.
- ❑ C# doesn't allow execution to continue from one switch section to the next.
- ❑ When a break statement is reached, the switch terminates, and the flow of control jumps to the next line following the switch statement.
- ❑ Not every case needs to contain a break. If no break appears, then it will raise a compile time error.

- ❑ In C#, duplicate case values are not allowed.
- ❑ The data type of the variable in the switch and value of a case must be of the same type.
- ❑ The value of a case must be a constant or a literal. Variables are not allowed.
- ❑ The break in switch statement is used to terminate the current sequence.
- ❑ The default statement is optional and it can be used anywhere inside the switch statement.
- ❑ Multiple default statements are not allowed.

Syntax:

```
switch (expression) {  
  
    case value1: // statement sequence  
        break;  
  
    case value2: // statement sequence  
        break;  
  
    .  
    .  
    .  
  
    case valueN: // statement sequence  
        break;  
  
    default:    // default statement sequence  
}
```



```
class switchcase
{
    public static void Main()
    {
        int value;
        Console.WriteLine("Enter the value between 0 to 9:");
        value = Convert.ToInt32(Console.ReadLine());

        switch (value)
        {
            case 0:
                Console.WriteLine(" You Have Entered 0");
                break;
            case 1:
                Console.WriteLine(" You Have Entered 1");
                break;
            case 2:
                Console.WriteLine(" You Have Entered 2");
                break;
            case 3:
                Console.WriteLine(" You Have Entered 3");
                break;
            case 4:
                Console.WriteLine(" You Have Entered 4");
                break;
            case 5:
                Console.WriteLine("You Have Entered 5");
                break;
            case 6:
                Console.WriteLine(" You Have Entered 6");
                break;
            case 7:
                Console.WriteLine(" You Have Entered 7");
                break;
            case 8:
                Console.WriteLine(" You Have Entered 8");
                break;
            case 9:
                Console.WriteLine("You Have Entered 9");
                break;
            default:
                Console.WriteLine("No match found");
                break;
        }
    }
}
```

file:///c:/users/dr.pv.praveen sundar/documents/visual studio 2010/Projects/Control/Control/bin/Debug/Control.EXE

Enter the value between 0 to 9:5

You Have Entered 5

Control - Microsoft Visual Studio

File Edit View Refactor Project Build Debug Team Data Tools Test Window Help

Debug x86

Control switchcase.cs elseif.cs nestedif.cs ifelse.cs Program.cs

Control.switchcase Main()

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Control
{
    class switchcase
    {
        public static void Main()
        {
            char ch;
            Console.Write("Enter the Character :");
            ch = Convert.ToChar(Console.ReadLine());
            switch (ch)
            {
                case 'a':
                case 'A':
                case 'E':
                case 'e':
                case 'I':
                case 'i':
                case 'o':
                case 'O':
                case 'u':
                case 'U':
                    Console.WriteLine("{0} is a vowel", ch);
                    break;
                default: Console.WriteLine("{0} is not a vowel", ch);
                    break;
            }
            Console.ReadKey();
        }
    }
}
```

100 %

Solution Explorer

Solution 'Control' (1 project)

- Control
 - Properties
 - References
 - elseif.cs
 - ifelse.cs
 - nestedif.cs
 - Program.cs
 - switchcase.cs

Properties

Item(s) Saved

Ln 17

Col 26

Ch 26

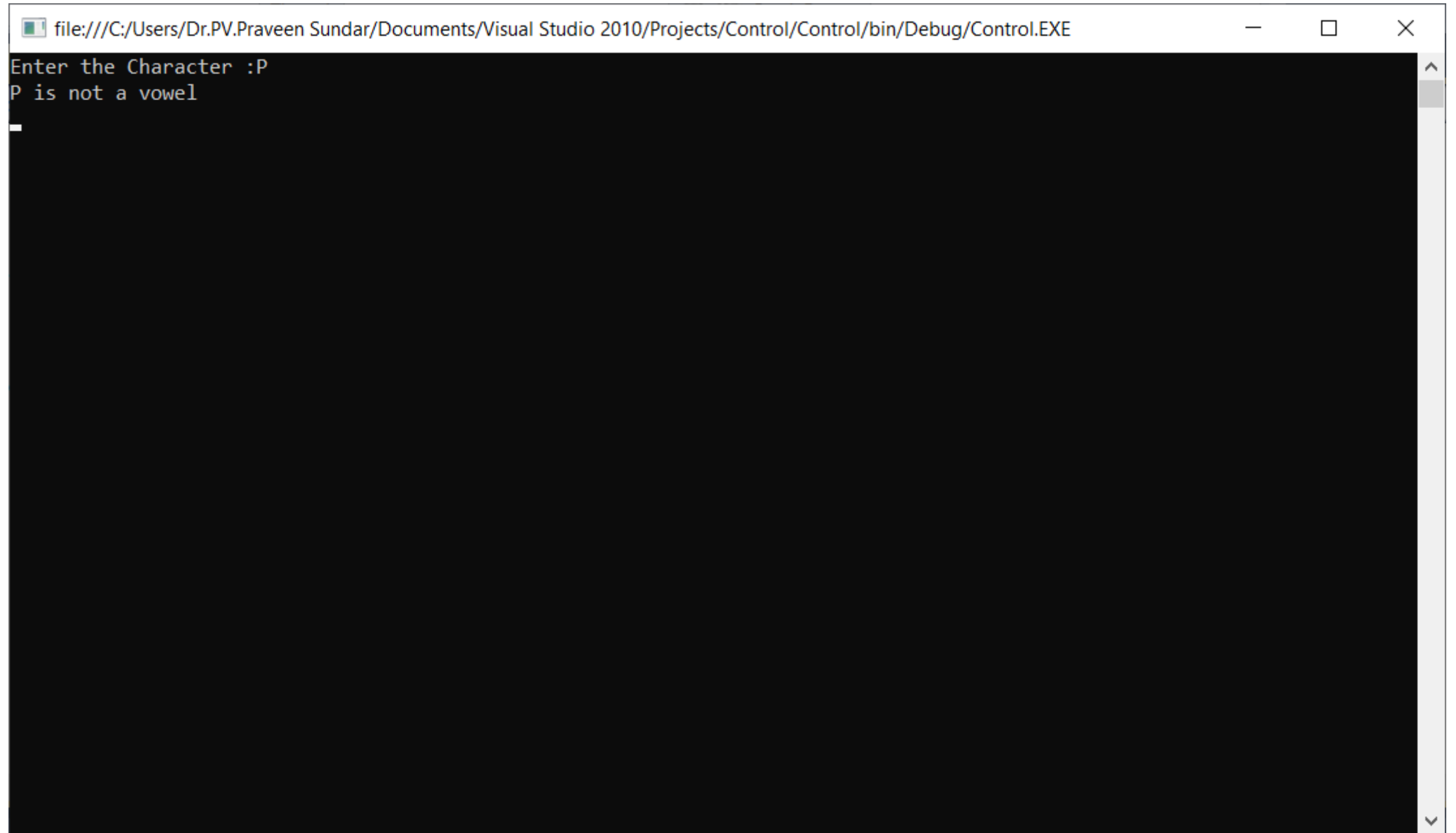
INS

Type here to search

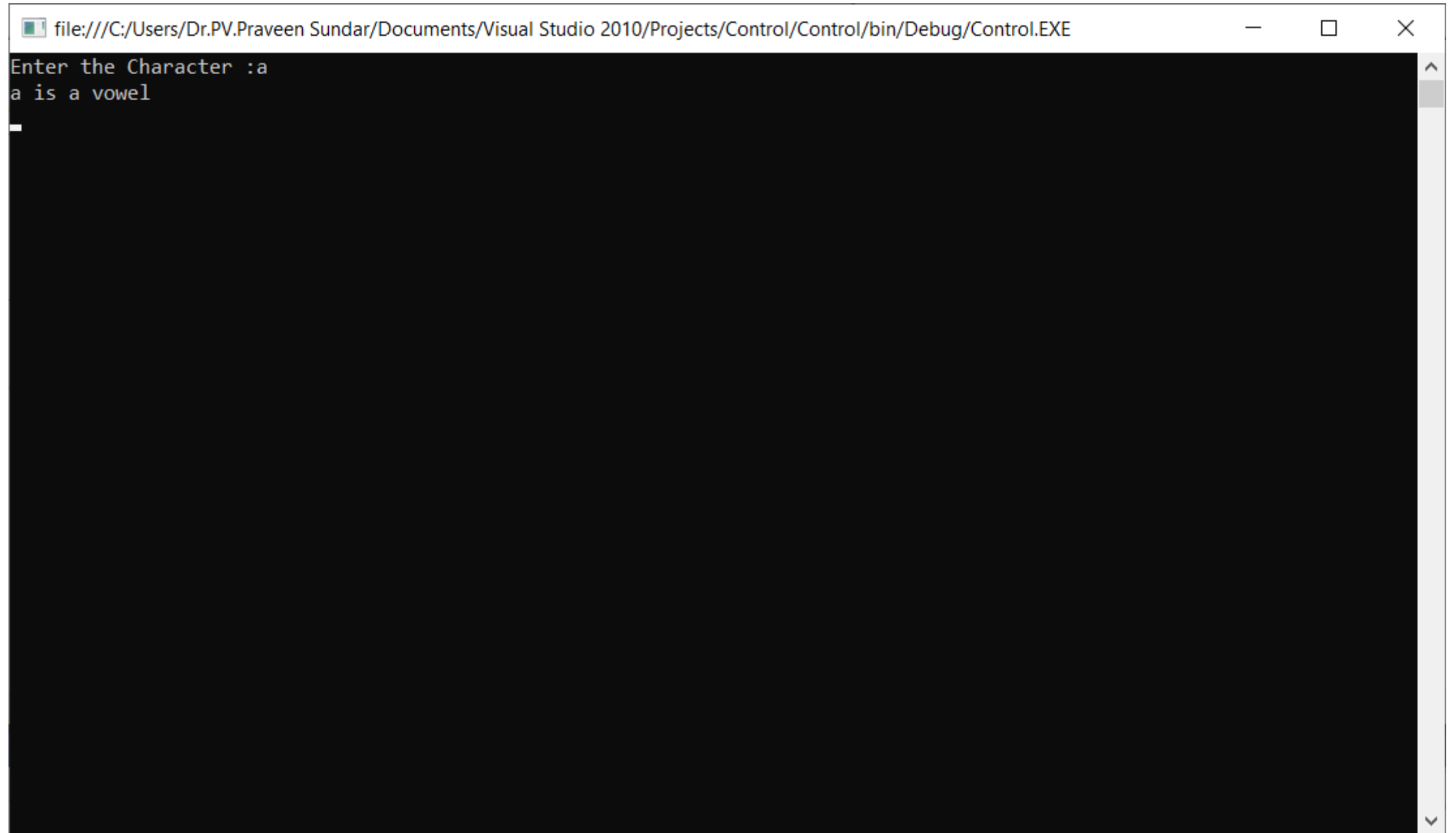
34°C Mostly cloudy

ENG

11:25 AM
24-06-2021



```
file:///C:/Users/Dr.PV.Praveen Sundar/Documents/Visual Studio 2010/Projects/Control/Control/bin/Debug/Control.EXE
Enter the Character :P
P is not a vowel
_
```



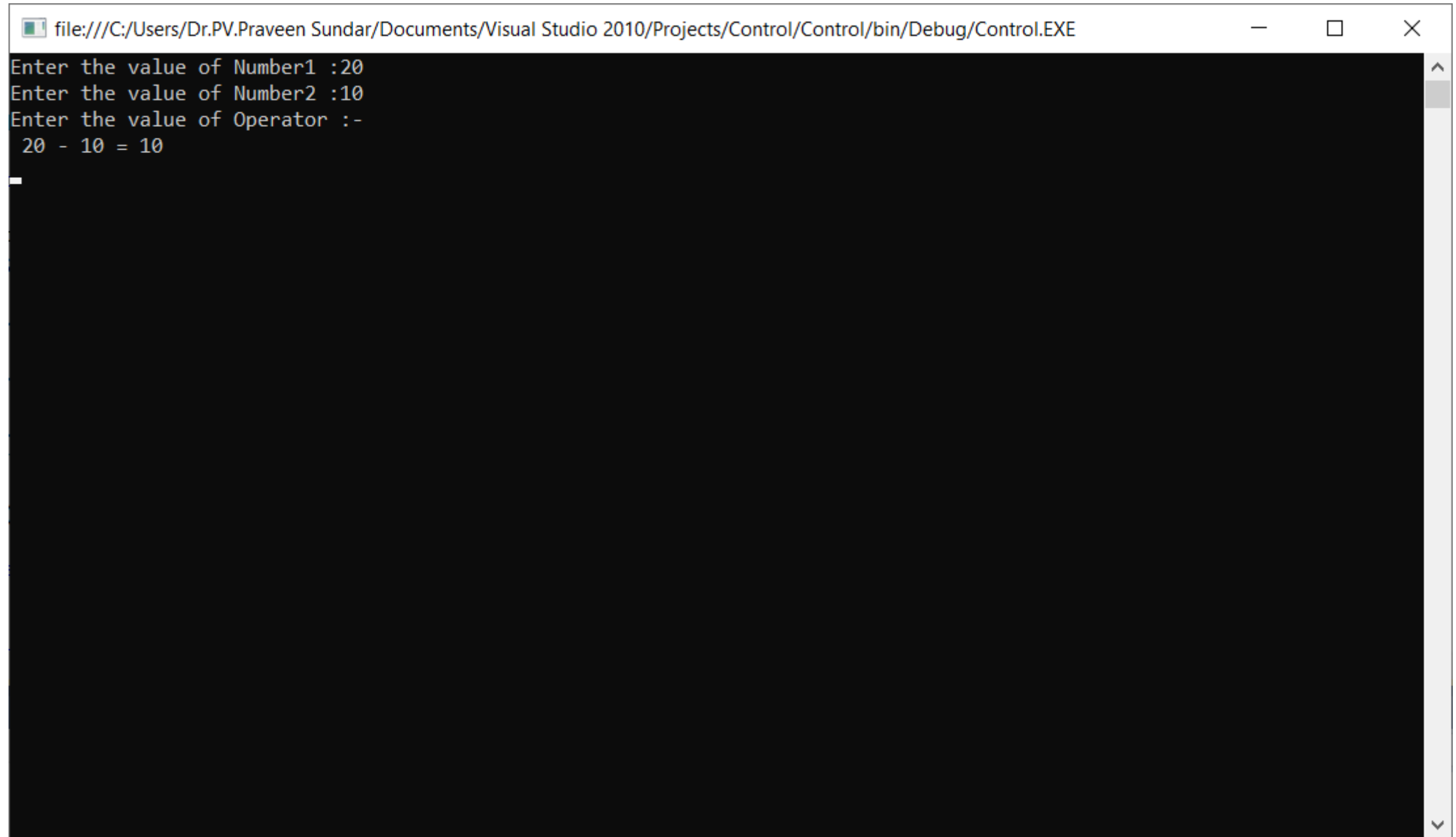
```
file:///C:/Users/Dr.PV.Praveen Sundar/Documents/Visual Studio 2010/Projects/Control/Control/bin/Debug/Control.EXE
Enter the Character :a
a is a vowel
_
```

```
using System;

namespace Control
{
    class Switch_calc
    {
        public static void Main()
        {
            int number1, number2;
            char op;

            Console.Write("Enter the value of Number1 :");
            number1 = Convert.ToInt32(Console.ReadLine());
            Console.Write("Enter the value of Number2 :");
            number2 = Convert.ToInt32(Console.ReadLine());
            Console.Write("Enter the value of Operator :");
            op = Convert.ToChar(Console.ReadLine());

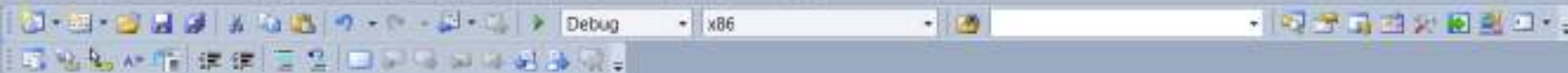
            switch (op)
            {
                case '+':
                    Console.WriteLine(" {0} + {1} = {2} ", number1, number2, number1+number2);
                    break;
                case '-':
                    Console.WriteLine(" {0} - {1} = {2} ", number1, number2, number1 - number2);
                    break;
                case '*':
                    Console.WriteLine(" {0} * {1} = {2} ", number1, number2, number1 * number2);
                    break;
                case '/':
                    Console.WriteLine(" {0} / {1} = {2} ", number1, number2, number1 / number2);
                    break;
                case '%':
                    Console.WriteLine(" {0} % {1} = {2} ", number1, number2, number1 % number2);
                    break;
                default:
                    Console.WriteLine(":Invalid Operation :");
                    break;
            }
            Console.ReadKey();
        }
    }
}
```



A screenshot of a Windows command prompt window. The title bar at the top shows the file path: `file:///C:/Users/Dr.PV.Praveen Sundar/Documents/Visual Studio 2010/Projects/Control/Control/bin/Debug/Control.EXE`. The window contains the following text:

```
Enter the value of Number1 :20
Enter the value of Number2 :10
Enter the value of Operator :-
20 - 10 = 10
```

A white cursor is visible on the line following the output.



switchbool.cs Switch_calc.cs Switch_char.cs switchcase.cs elseif.cs nestedif.cs ifelse.cs Program.cs

Control.switchbool Main()

```
using System;

namespace Control
{
    class switchbool
    {
        public static void Main()
        {
            bool value;

            value = int.MaxValue < long.MaxValue;

            switch (value)
            {
                case true:
                    Console.WriteLine(" Condition True ");
                    break;
                case false:
                    Console.WriteLine(" Condition False ");
                    break;
            }
            Console.ReadKey();
        }
    }
}
```

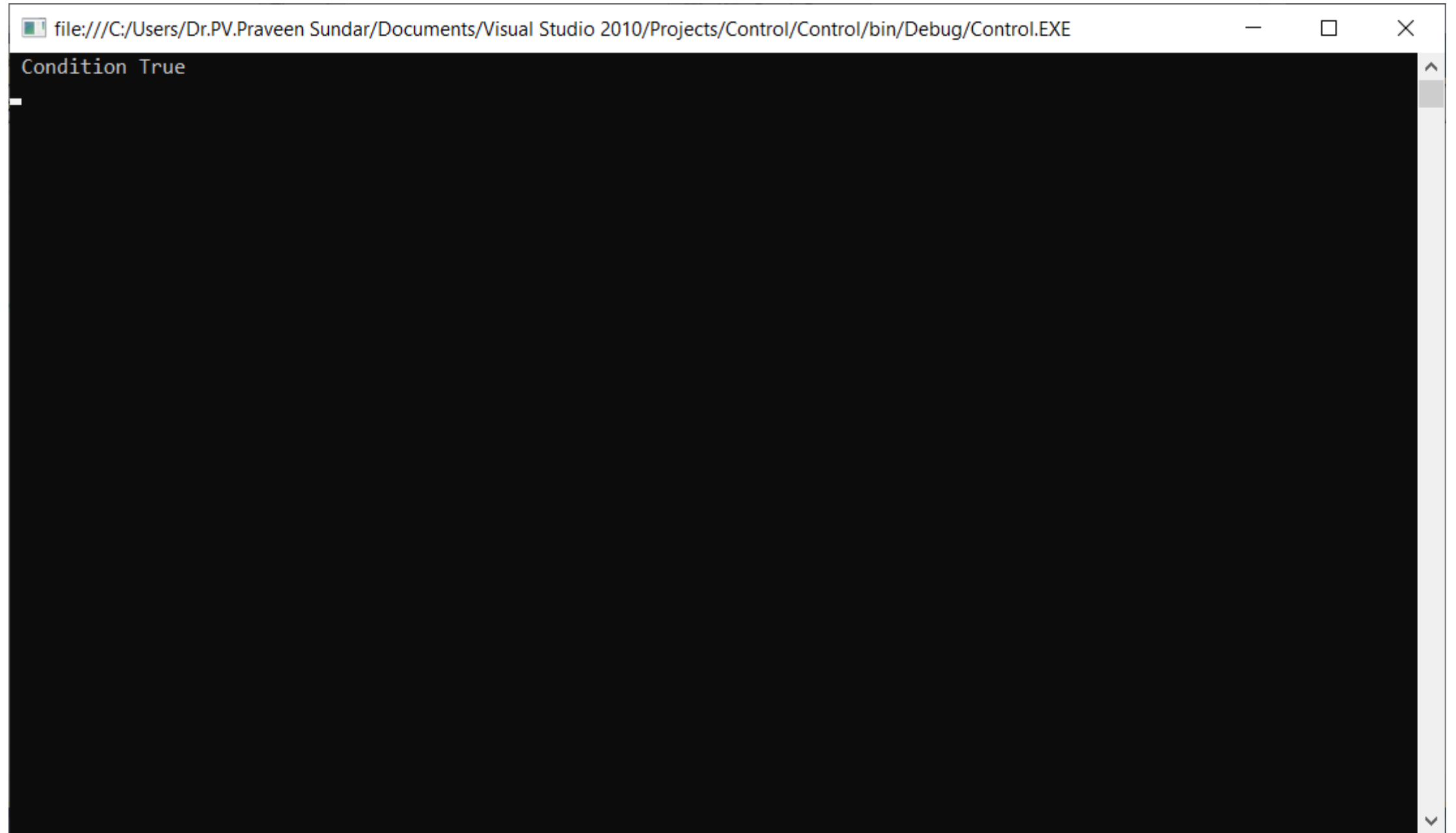
Solution Explorer

Solution 'Control' (1 project)

- Control
 - Properties
 - References
 - elseif.cs
 - ifelse.cs
 - nestedif.cs
 - Program.cs
 - Switch_calc.cs
 - Switch_char.cs
 - switchbool.cs
 - switchcase.cs

Solut... Team... Class...

Properties



Iteration Statements

56

- ❑ Iteration statements or Loops are used in programming to repeatedly execute a certain block of statements until some condition is met.
- ❑ The following statements repeatedly execute a statement or a block of statements:
 - ❑ The **for** statement: executes its body while a specified Boolean expression evaluates to true.
 - ❑ The **foreach** statement: enumerates the elements of a collection and executes its body for each element of the collection.
 - ❑ The **do** statement: conditionally executes its body one or more times.
 - ❑ The **while** statement: conditionally executes its body zero or more times.
- ❑ At any point within the body of an iteration statement, you can break out of the loop by using the **break** statement, or step to the next iteration in the loop by using the **continue** statement.

while loop

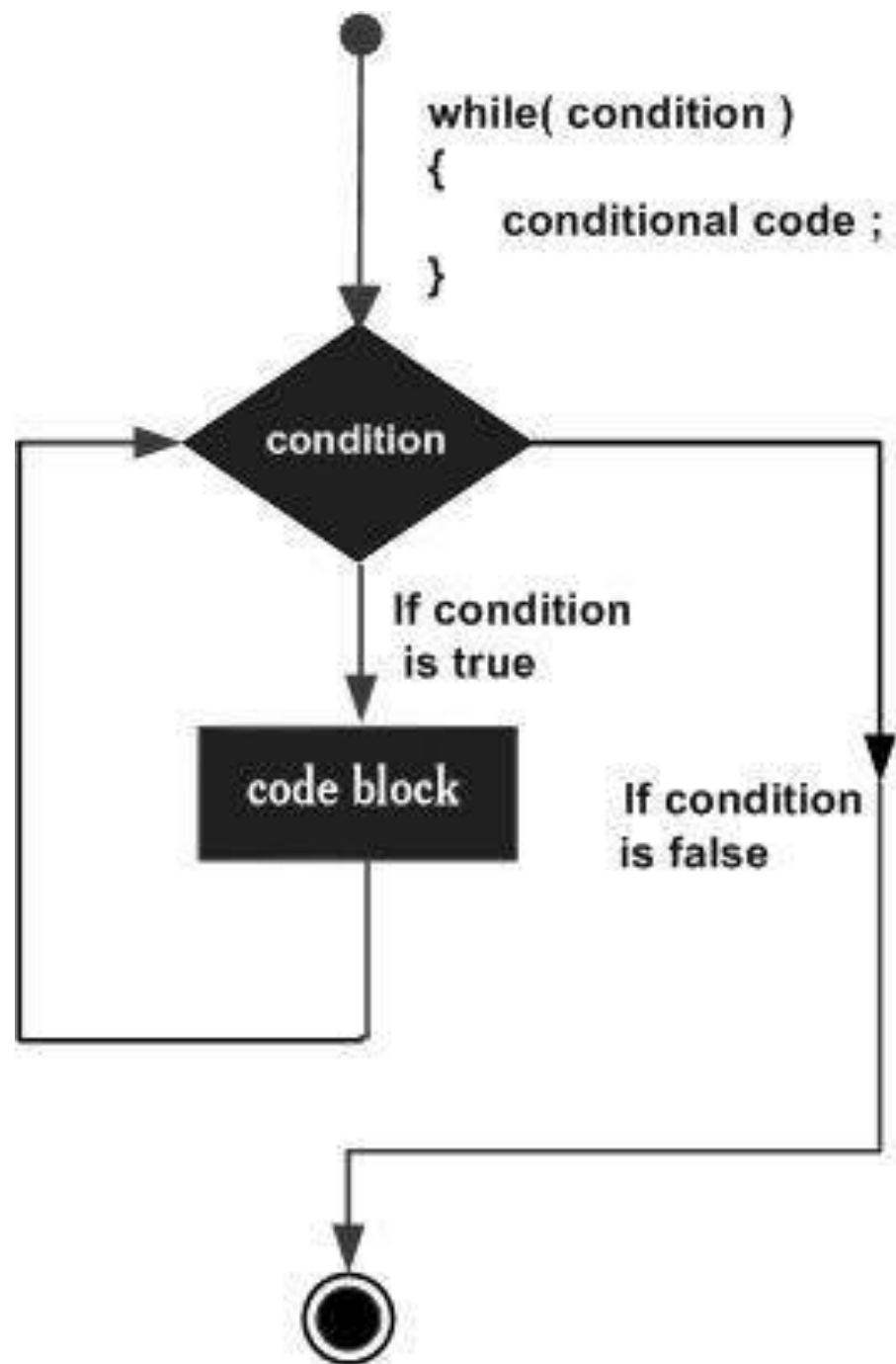
57

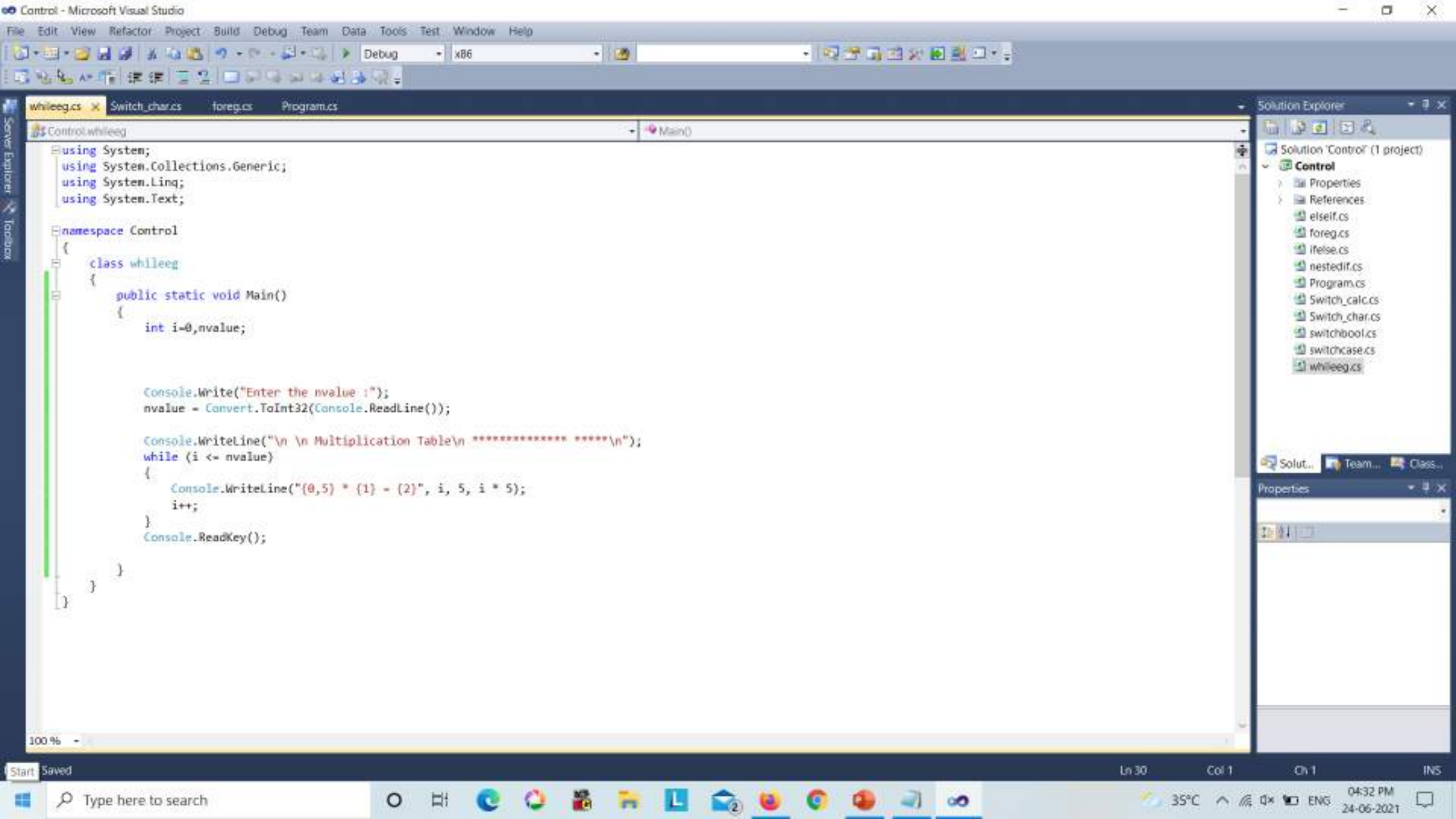
- ❑ C# provides the while loop to repeatedly execute a block of code as long as the specified condition returns false.
- ❑ The while loop starts with the while keyword, and it must include a Boolean conditional expression inside brackets that returns either true or false.
- ❑ It executes the code block until the specified conditional expression returns false.
- ❑ In a while loop, initialization should be done before the loop starts, and increment or decrement steps should be inside the loop.
- ❑ The statement(s) inside the while loop may be a single statement or a block of statements.
- ❑ The key point of the while loop is that the loop might not ever run. When the condition is tested and the result is false, the loop body is skipped and the first statement after the while loop is executed.

Syntax

The syntax of a **while** loop in C# is –

```
while(condition) {  
    statement(s);  
}
```

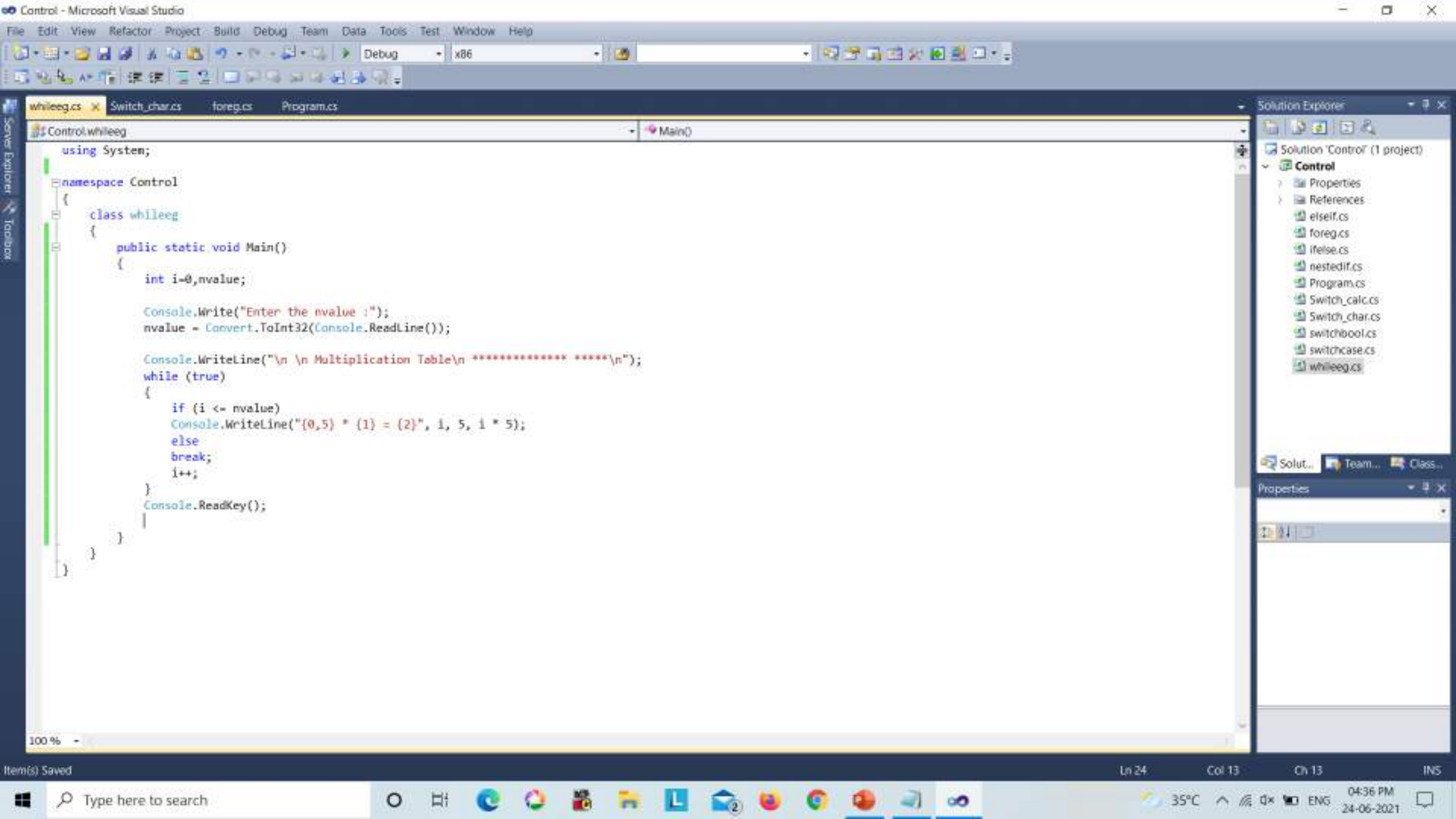




Enter the nvalue :10

Multiplication Table

```
0 * 5 = 0
1 * 5 = 5
2 * 5 = 10
3 * 5 = 15
4 * 5 = 20
5 * 5 = 25
6 * 5 = 30
7 * 5 = 35
8 * 5 = 40
9 * 5 = 45
10 * 5 = 50
```

Enter the nvalue :10

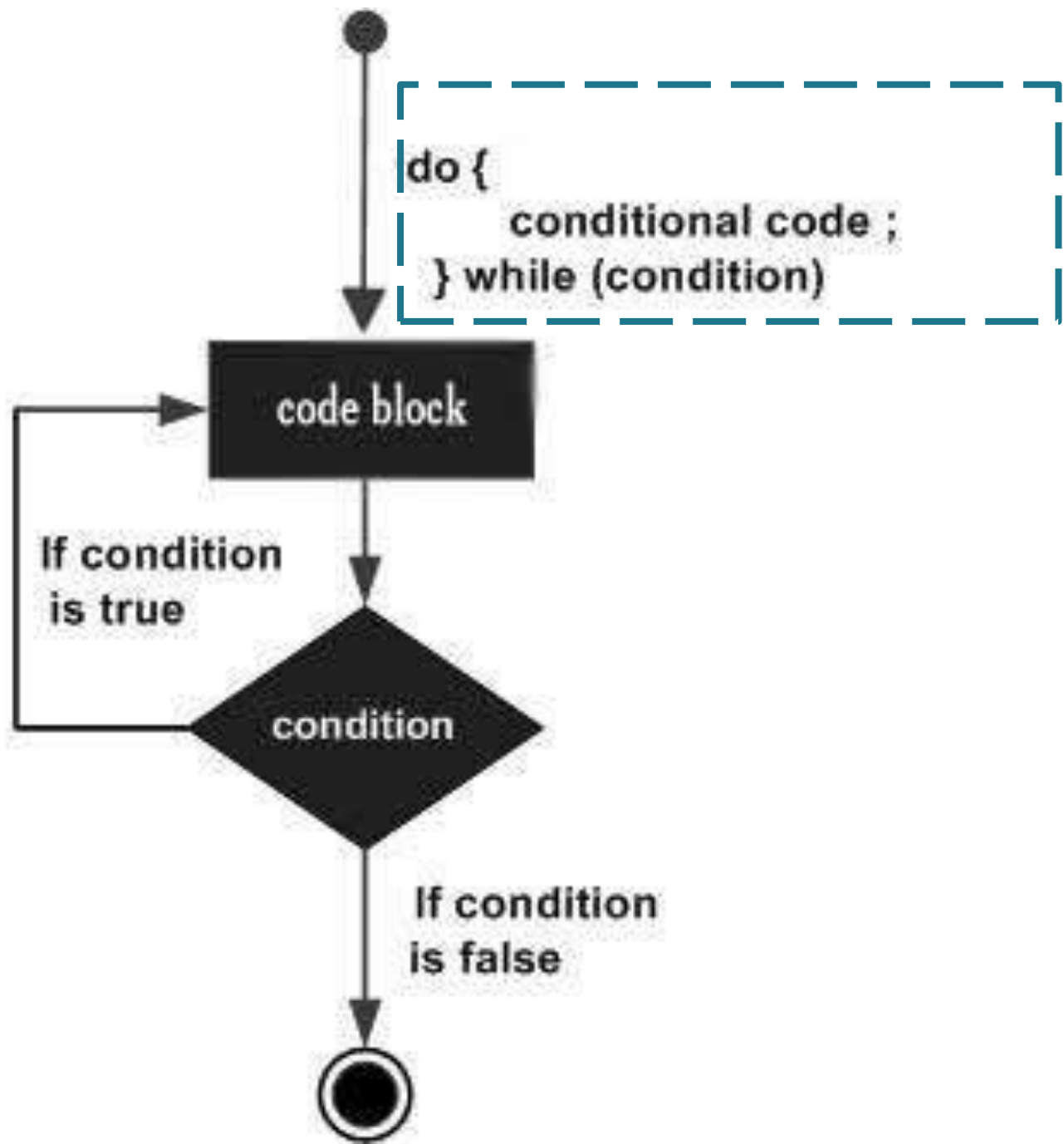
Multiplication Table

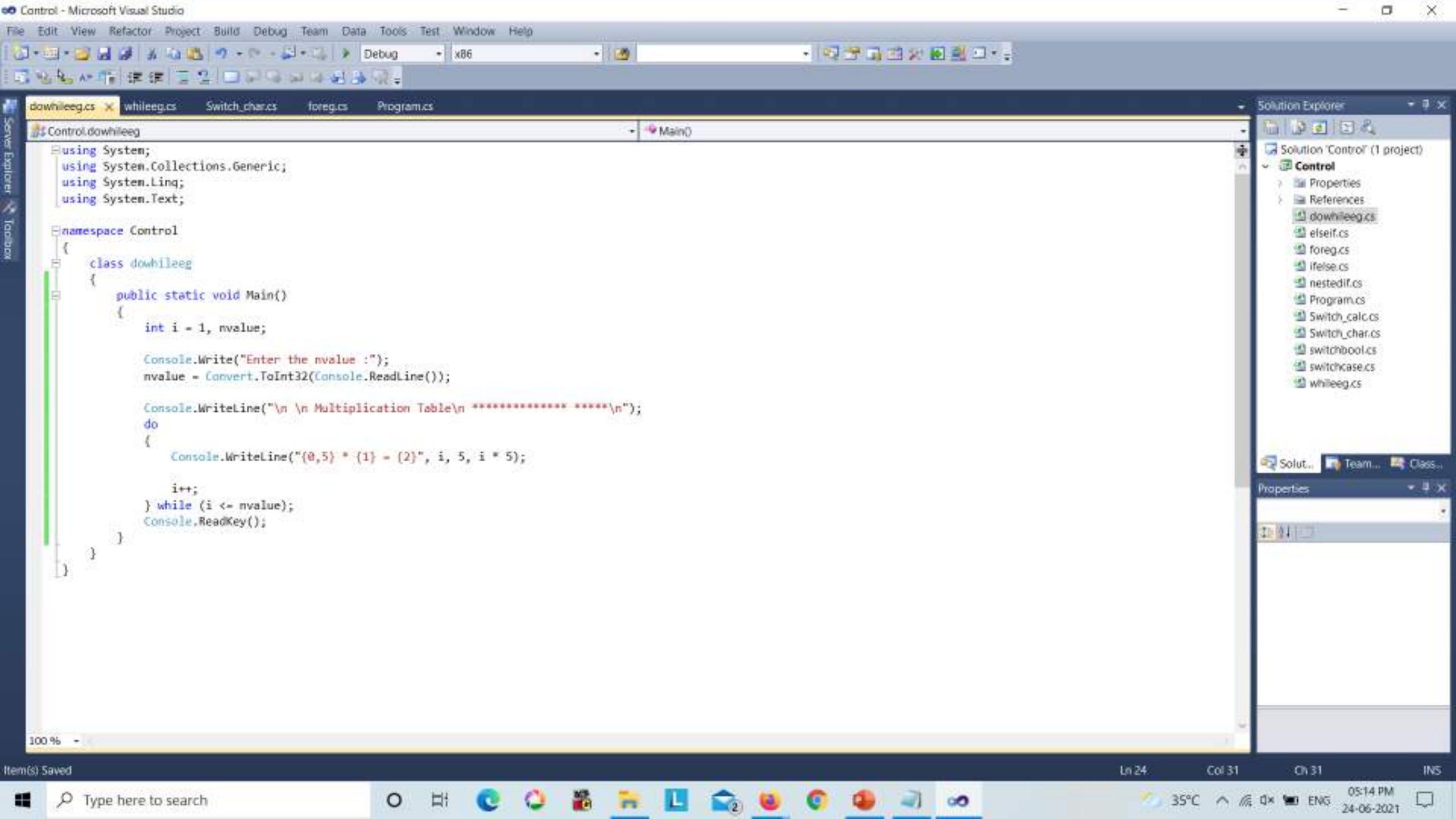
```
0 * 5 = 0
1 * 5 = 5
2 * 5 = 10
3 * 5 = 15
4 * 5 = 20
5 * 5 = 25
6 * 5 = 30
7 * 5 = 35
8 * 5 = 40
9 * 5 = 45
10 * 5 = 50
```

do while Statement

64

- ❑ The do while loop is the same as while loop except that it executes the code block at least once.
- ❑ The do-while loop starts with the do keyword followed by a code block and a boolean expression with the while keyword.
- ❑ The do while loop stops execution exits when a boolean condition evaluates to false. Because the while(condition) specified at the end of the block, it certainly executes the code block at least once.
- ❑ In a while loop, initialization should be done before the loop starts, and increment or decrement steps should be inside the loop.
- ❑ The statement(s) inside the while loop may be a single statement or a block of statements.
- ❑ The key point of the do while loop is that the loop will executed atleast once, even on the first time When the condition is tested and the result is false.
- ❑ This is the reason, do while is called as exit controlled loop.





Enter the nvalue :10

Multiplication Table

```
1 * 5 = 5
2 * 5 = 10
3 * 5 = 15
4 * 5 = 20
5 * 5 = 25
6 * 5 = 30
7 * 5 = 35
8 * 5 = 40
9 * 5 = 45
10 * 5 = 50
```

```
file:///C:/Users/Dr.PV.Praveen Sundar/documents/visual studio 2010/Projects/Control/Control/bin/Debug/Control.EXE
Enter the nvalue :-10

Multiplication Table
*****

1 * 5 = 5
```

for loop

69

- A for loop is a repetition control structure that allows you to efficiently write a loop that needs to execute a specific number of times.
 - ▣ C# for loop has three statements: initialization, condition and iterator.
 - ▣ The initialization statement is executed at first and only once. Here, the variable is usually declared and initialized.
 - ▣ Then, the condition is evaluated. The condition is a Boolean expression, i.e. it returns either true or false.
 - ▣ If the condition is evaluated to true:
 - The body of the loop, which must be a statement or a block of statements.
 - Then, the iterator statement is executed which usually changes the value of the initialized variable.
 - Again the condition is evaluated.
 - The process continues until the condition is evaluated to false.
 - ▣ If the condition is evaluated to false, the for loop terminates.

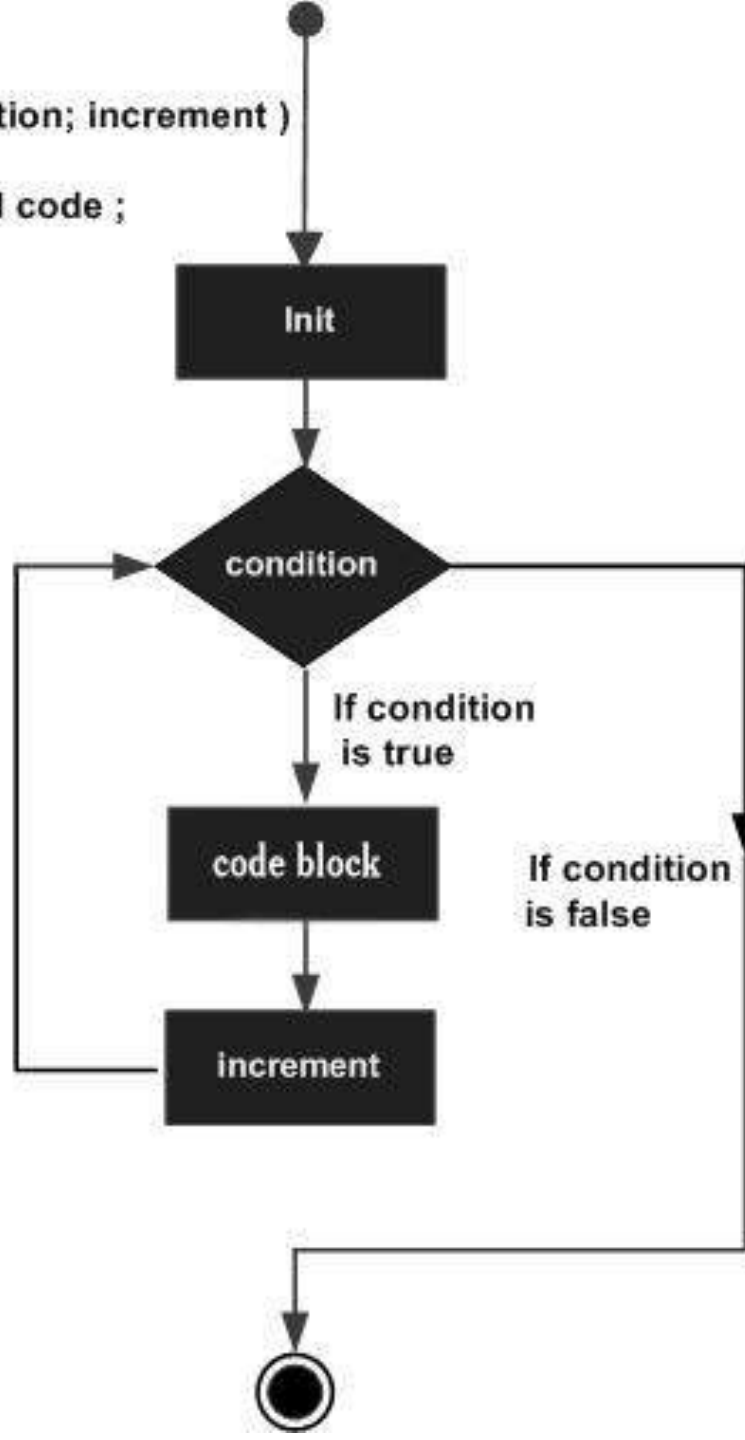
- The iterator section can contain zero or more of the following statement expressions, separated by commas:
 - ▣ prefix or postfix increment expression, such as ++i or i++
 - ▣ prefix or postfix decrement expression, such as --i or i--
 - ▣ assignment
 - ▣ invocation of a method
 - ▣ creation of an object by using the new operator.
- All the sections of the for statement are optional.

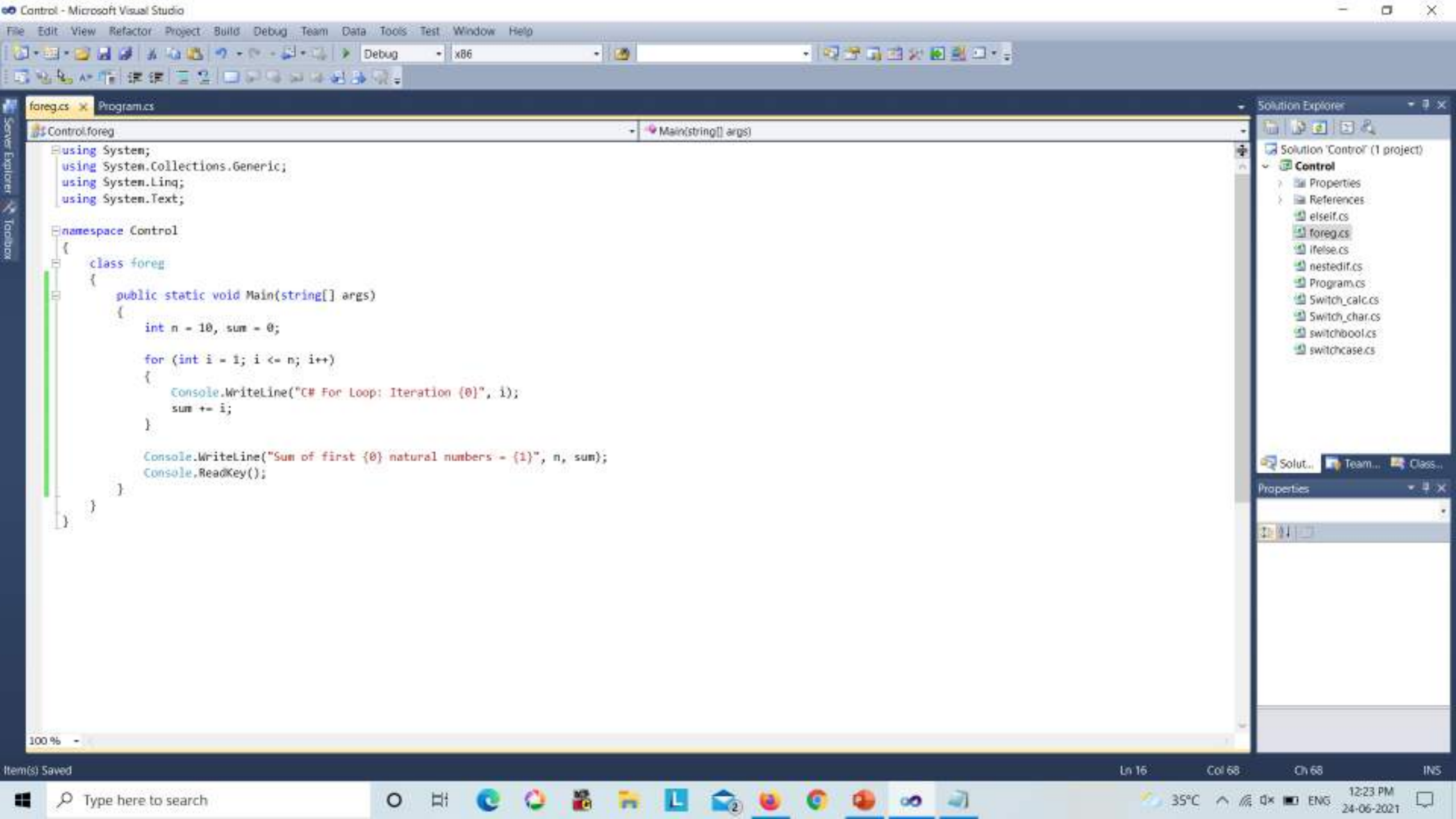
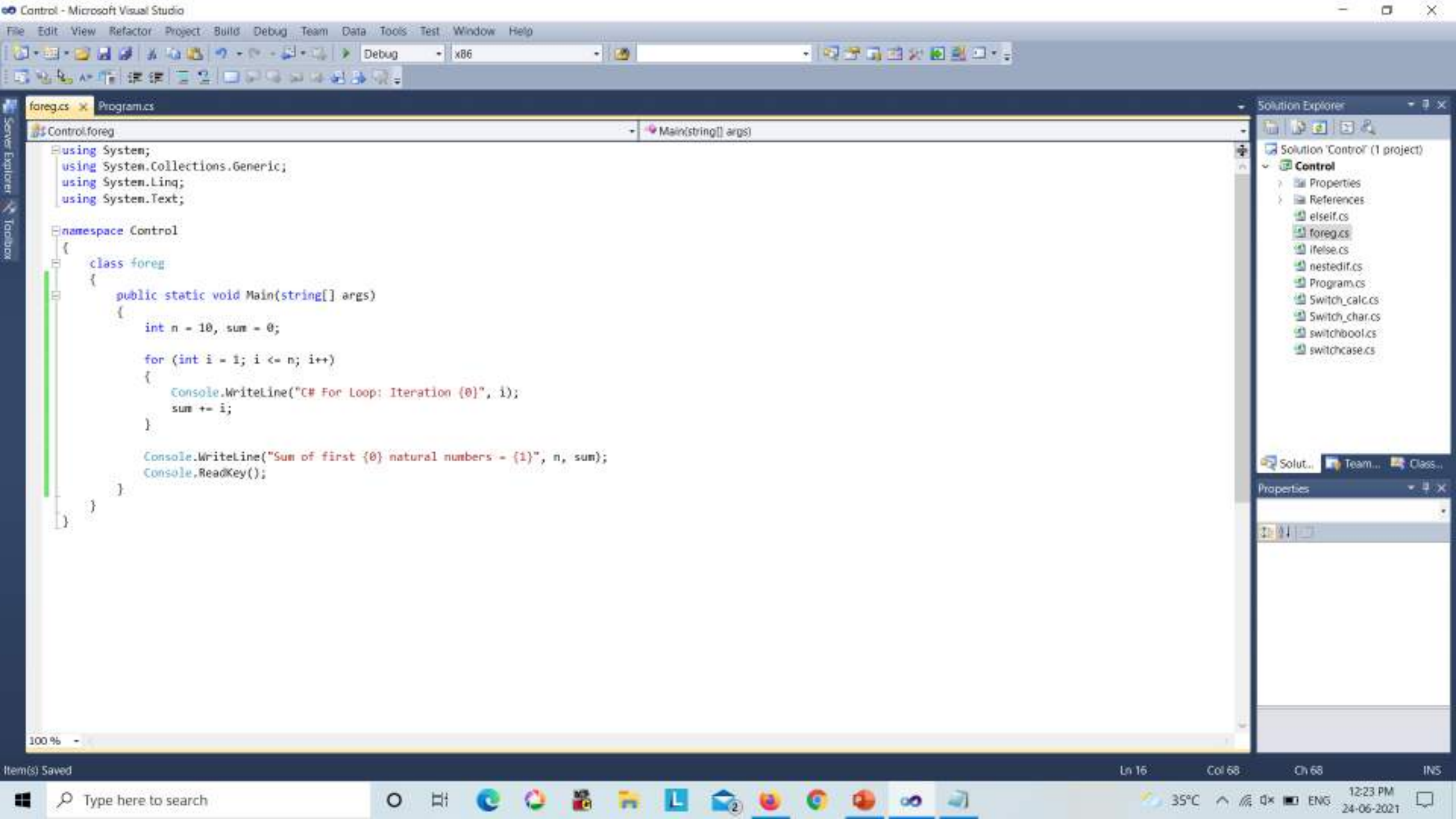
Syntax

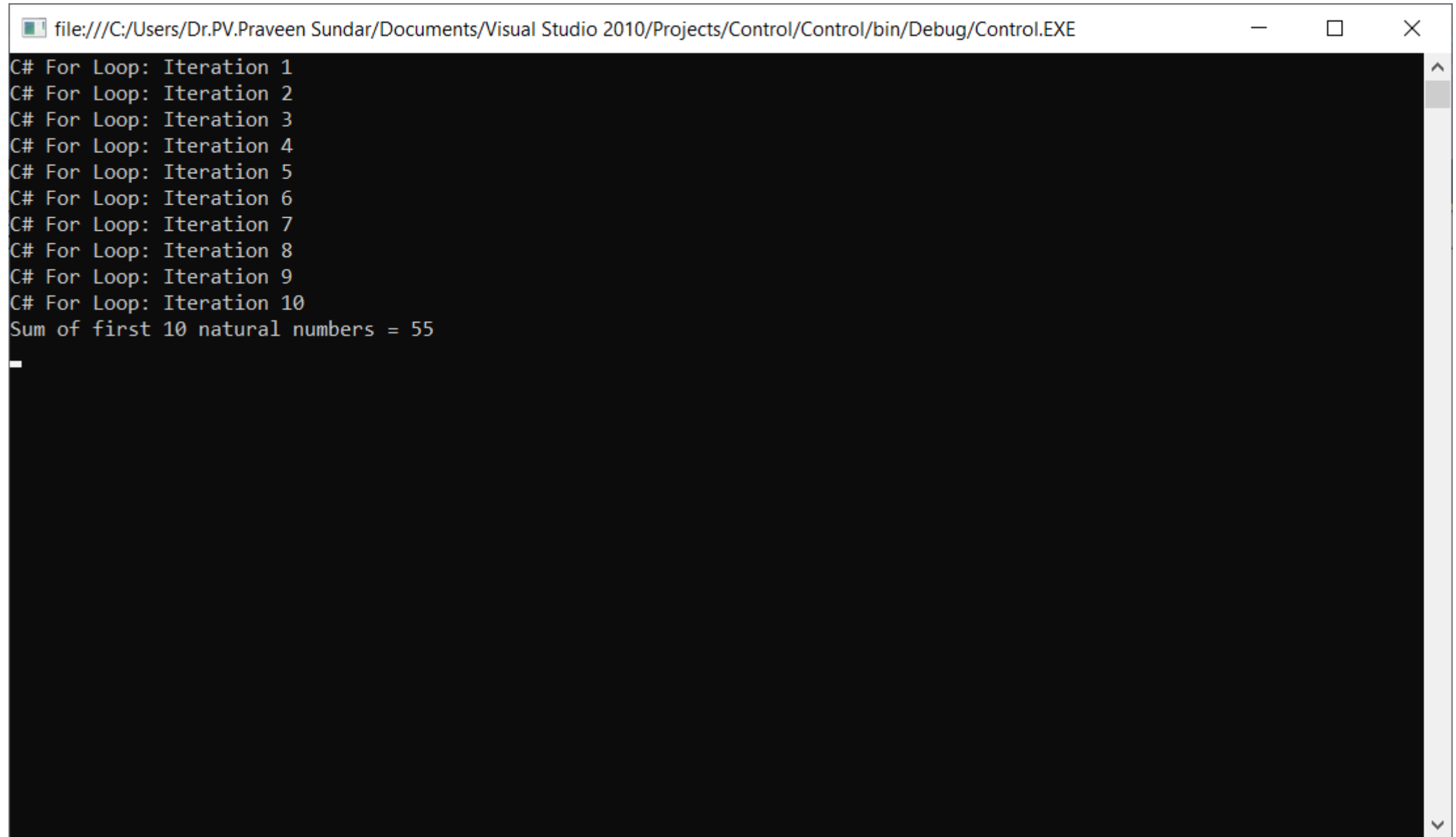
The syntax of a **for** loop in C# is –

```
for ( init; condition; increment ) {  
    statement(s);  
}
```

```
for( init; condition; increment )  
{  
    conditional code ;  
}
```







A screenshot of a Windows command prompt window. The title bar at the top shows the file path: `file:///C:/Users/Dr.PV.Praveen Sundar/Documents/Visual Studio 2010/Projects/Control/Control/bin/Debug/Control.EXE`. The window contains the following text output from a C# program:

```
C# For Loop: Iteration 1
C# For Loop: Iteration 2
C# For Loop: Iteration 3
C# For Loop: Iteration 4
C# For Loop: Iteration 5
C# For Loop: Iteration 6
C# For Loop: Iteration 7
C# For Loop: Iteration 8
C# For Loop: Iteration 9
C# For Loop: Iteration 10
Sum of first 10 natural numbers = 55
```

The text is displayed in a monospaced font on a black background. A small white cursor is visible on the line following the sum calculation.

foreach loop

75

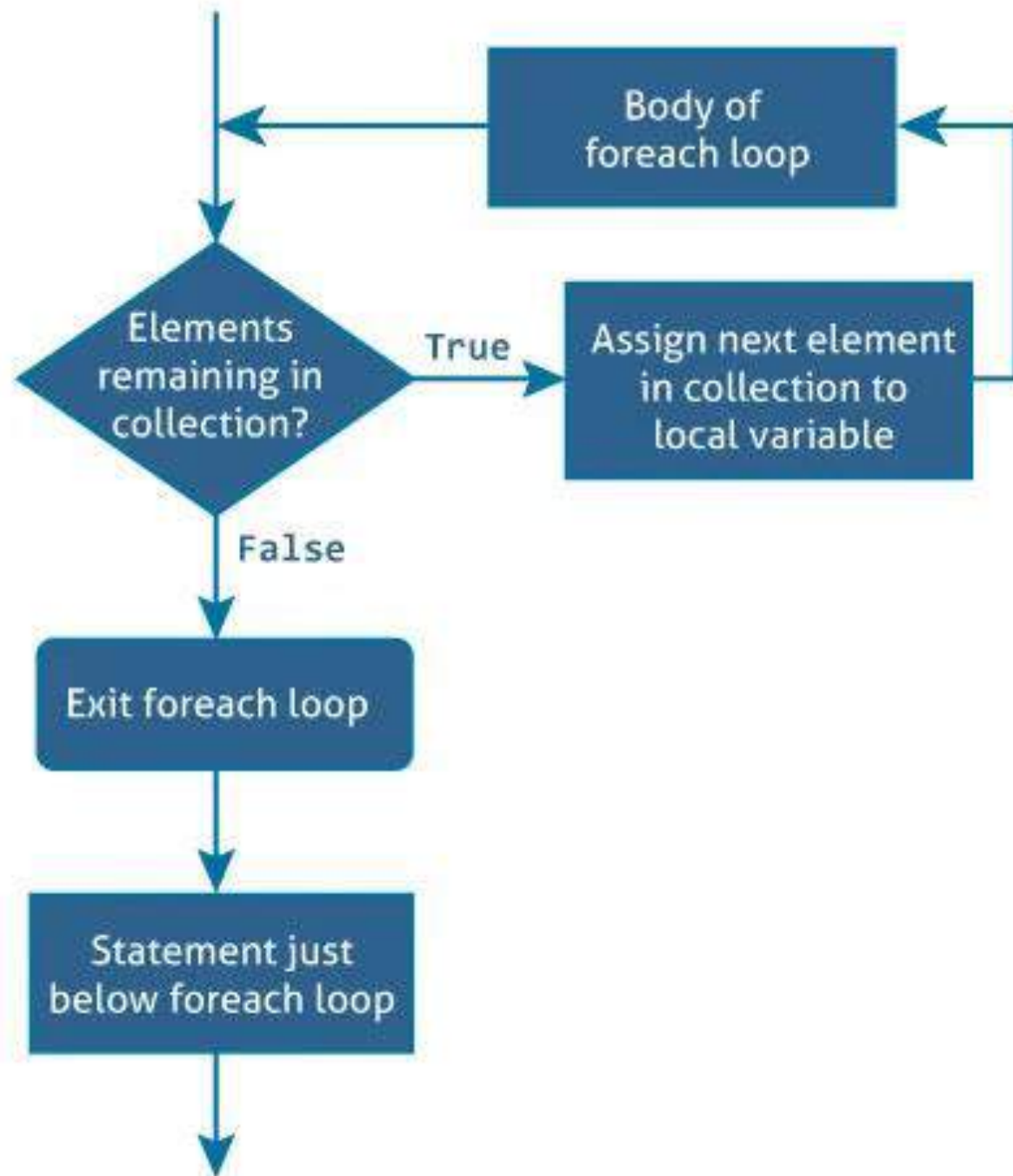
- ❑ foreach loop is a different kind of looping constructs in C# programming that doesn't includes initialization, termination and increment/decrement characteristics.
- ❑ The foreach loop in C# executes a block of code on each element in an array or a collection of items. When executing foreach loop it traversing items in a collection or an array .
- ❑ The foreach loop is useful for traversing each items in an array or a collection of items and displayed one by one.
- ❑ The iteration variable corresponds to a read-only local variable with a scope that extends over the embedded statement.
- ❑ During execution of a foreach statement, the iteration variable represents the collection element for which an iteration is currently being performed.
- ❑ A compile-time error occurs if the embedded statement attempts to modify the iteration variable (via assignment or the ++ and -- operators) or pass the iteration variable as a ref or out parameter.

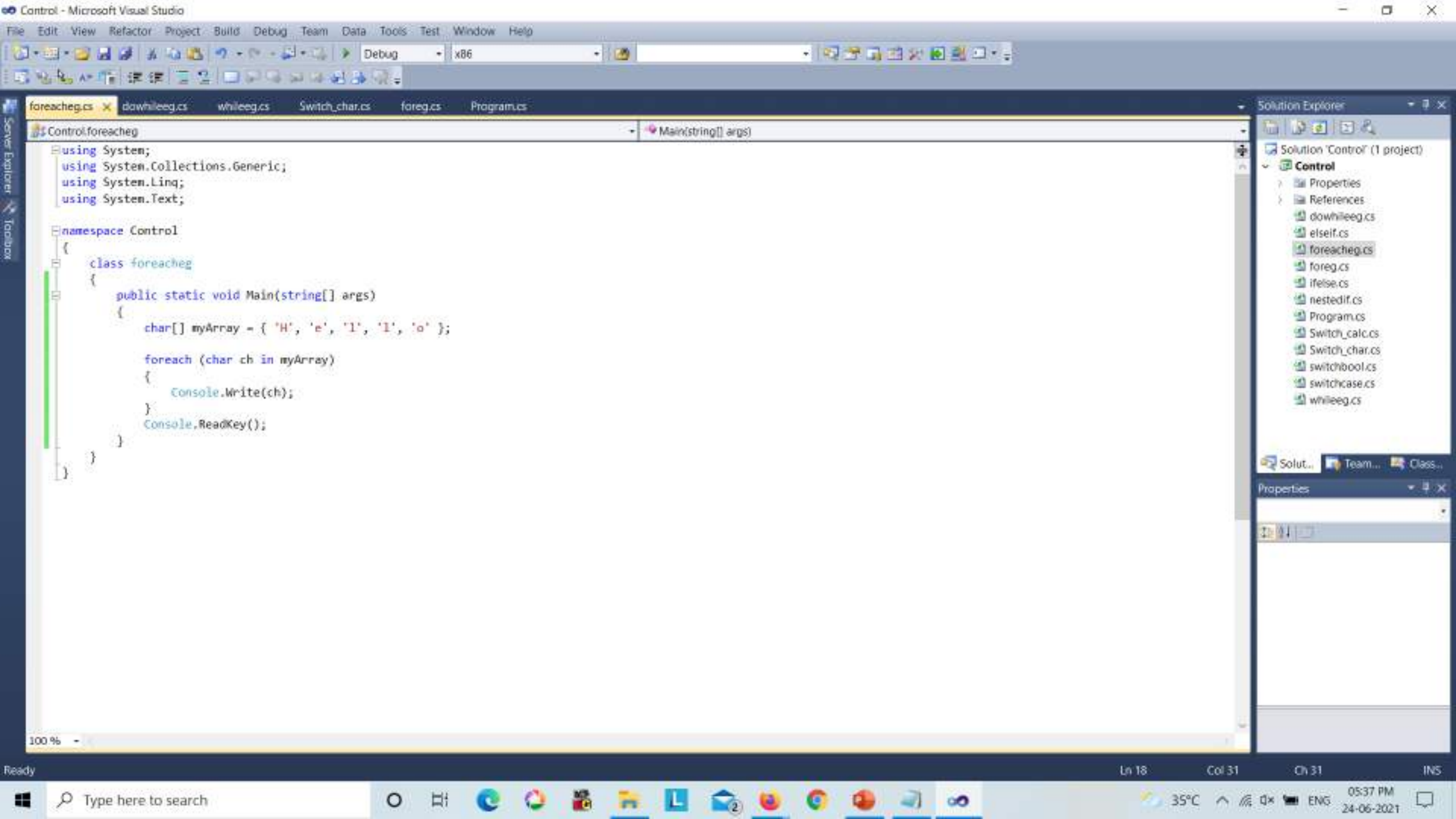
Syntax of foreach loop

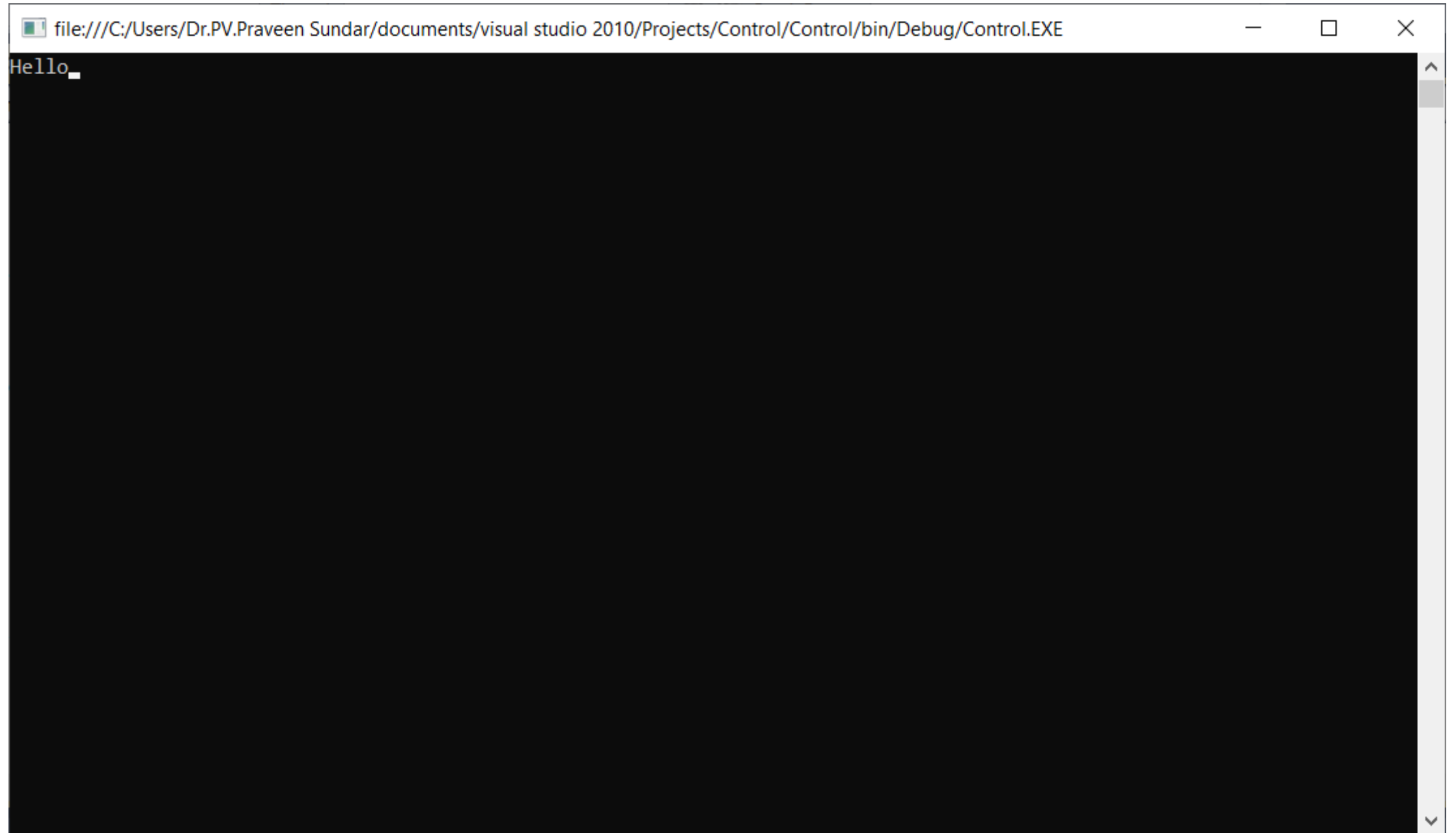
```
foreach (element in iterable-item)
{
    // body of foreach loop
}
```

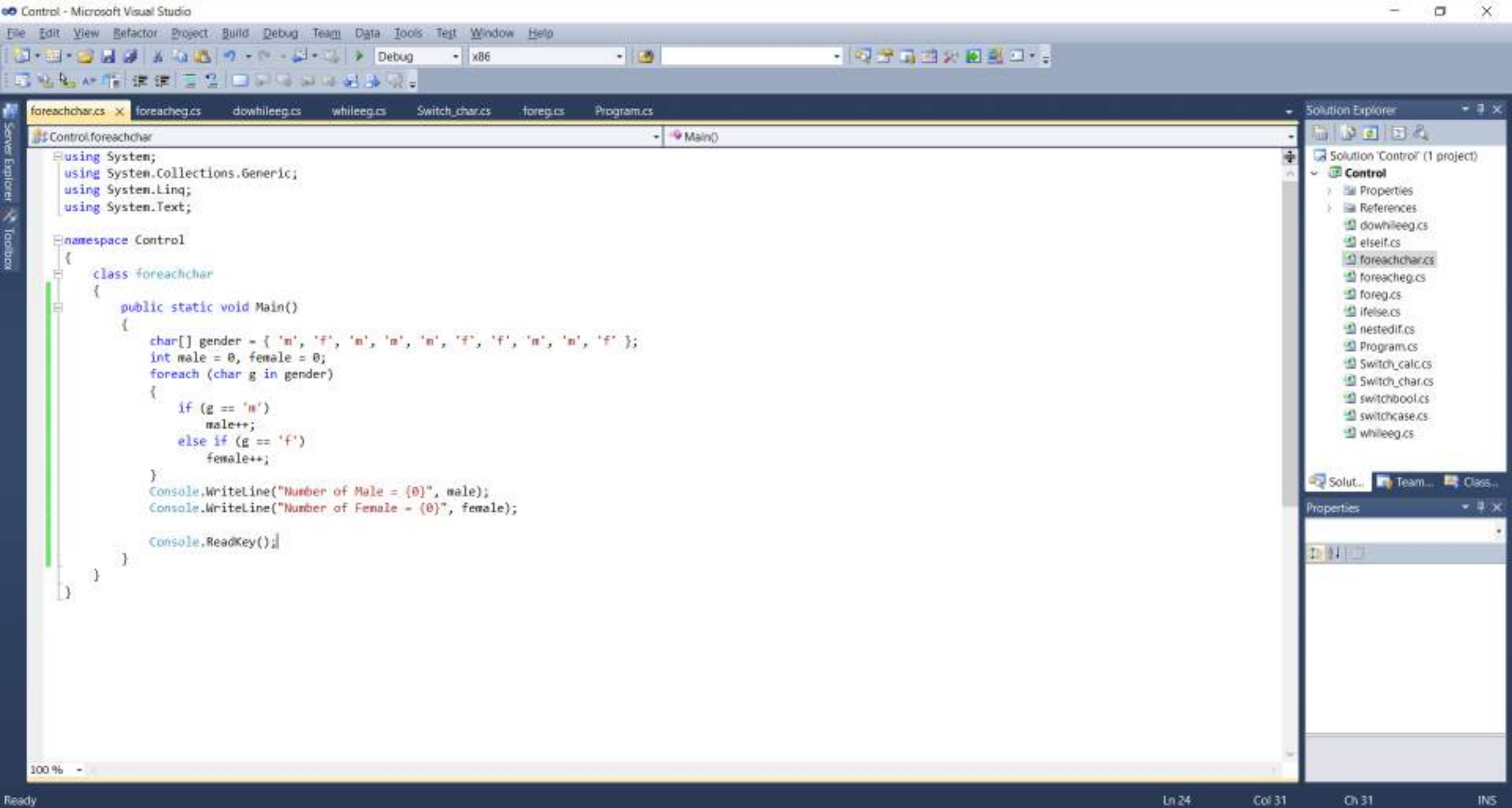
Here iterable-item can be an array or a class of collection.

- ❑ The in keyword used along with foreach loop is used to iterate over the iterable-item.
- ❑ The in keyword selects an item from the iterable-item on each iteration and store it in the variable element.
- ❑ On first iteration, the first item of iterable-item is stored in element. On second iteration, the second element is selected and so on.
- ❑ The number of times the foreach loop will execute is equal to the number of elements in the array or collection.









file:///C:/Users/Dr.PV.Praveen Sundar/documents/visual studio 2010/Projects/Control/Control/bin/Debug/Control.EXE



Number of Male = 6

Number of Female = 4

—



Control foreachlist.cs x foreachchar.cs foreacheg.cs dowhileeg.cs whileeg.cs Switch_char.cs foreg.cs Program.cs

Control.foreachlist Main(string[] args)

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Control
{
    class foreachlist
    {
        public static void Main(string[] args)
        {
            var numbers = new List<int>() { 5, -8, 3, 14, -9, 17, 0, 4 };
            int sum = 0;

            foreach (int number in numbers)
            {
                if(number>0)
                    sum += number;
            }
            Console.WriteLine("Sum = {0}", sum);

            Console.ReadKey();
        }
    }
}
```

Solution Explorer

Solution 'Control' (1 project)

Control

Properties

References

dowhileeg.cs

elseif.cs

foreachchar.cs

foreacheg.cs

foreachlist.cs

foreg.cs

ifelse.cs

nestedif.cs

Program.cs

Switch_calc.cs

Switch_char.cs

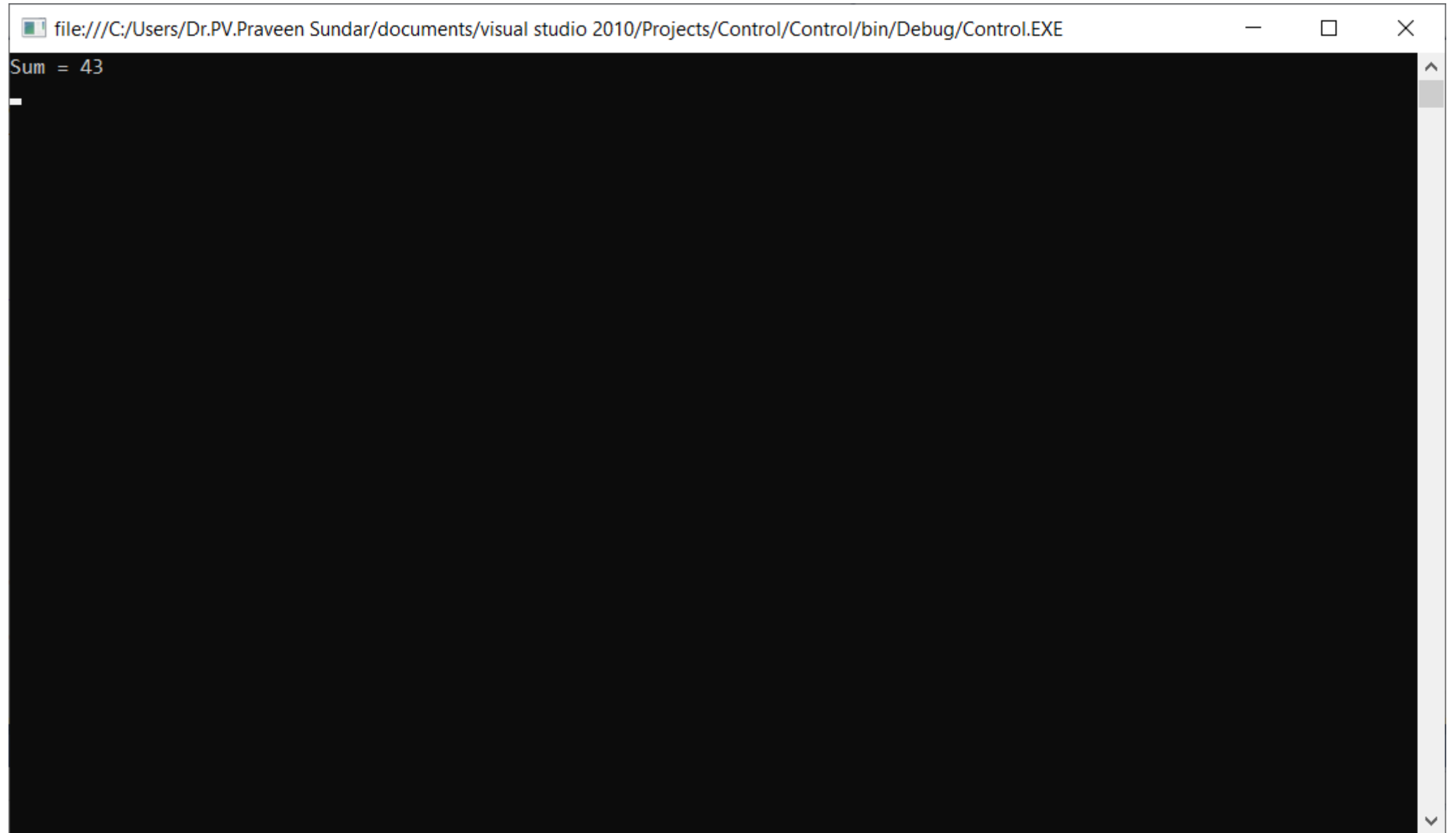
switchbool.cs

switchcase.cs

whileeg.cs

Solut... Team... Class...

Properties

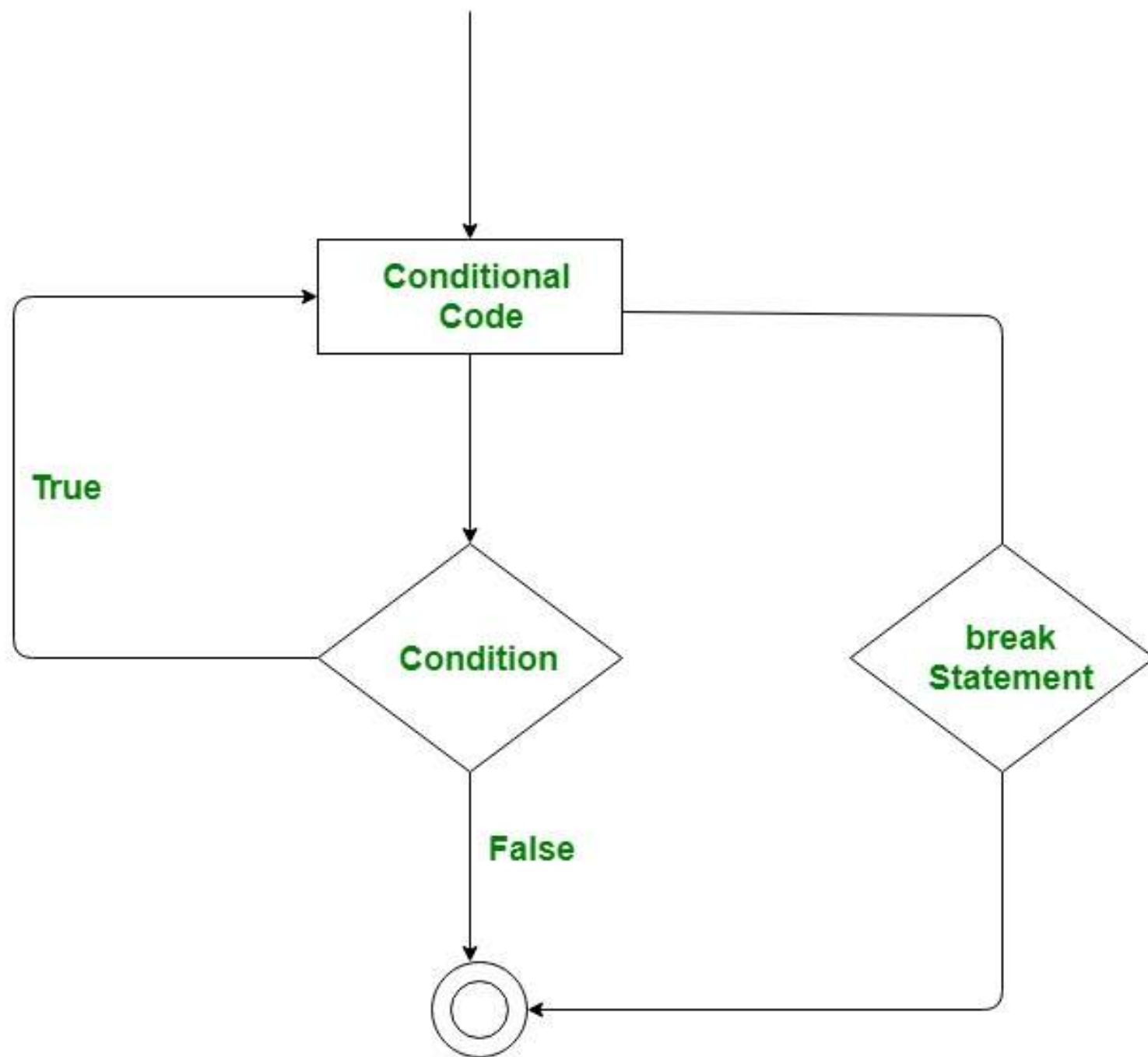


break Statements

84

Break (breaks the loop/switch)

- ❑ Break statement is used to terminate the current loop iteration or terminate the switch statement in which it appears.
- ❑ When the break statement is encountered inside a loop, the loop is immediately terminated and program control resumes at the next statement following the loop.
- ❑ Break statement can be used in the following scenarios:
 - ▣ for loop (For loop & nested for loop.
 - ▣ foreach loop (foreach loop & nested foreach loop.
 - ▣ While (while loop & nested while loop).
 - ▣ Do while (do while loop and nested while loop)
 - ▣ Switch case (Switch cases and nested switch cases)





breakeg.cs foreachlist.cs foreachchar.cs dowhileeg.cs whileeg.cs Switch_char.cs foreg.cs Program.cs

Control.breakeg Main()

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Control
{
    class breakeg
    {
        public static void Main()
        {
            for (int i = 0; i < 10; i++)
            {
                if (i == 5)
                    break;

                Console.WriteLine("Value of i: {0}", i);
            }
            Console.ReadKey();
        }
    }
}
```

Solution Explorer

Solution 'Control' (1 project)

- Control
 - Properties
 - References
 - breakeg.cs
 - dowhileeg.cs
 - elseif.cs
 - foreachchar.cs
 - foreacheg.cs
 - foreachlist.cs
 - foreg.cs
 - ifelse.cs
 - nestedif.cs
 - Program.cs
 - Switch_calc.cs
 - Switch_char.cs
 - switchbool.cs
 - switchcase.cs
 - whileeg.cs

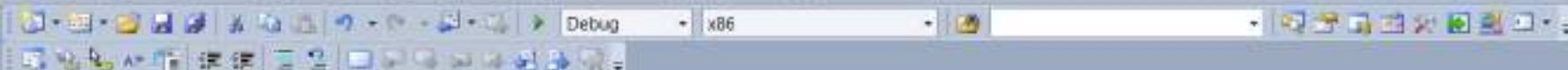
Solut... Team... Class...

Properties



file:///C:/Users/Dr.PV.Praveen Sundar/documents/visual studio 2010/Projects/Control/Control/bin/Debug/Control.EXE

```
Value of i: 0  
Value of i: 1  
Value of i: 2  
Value of i: 3  
Value of i: 4
```



breakeg.cs | foreachlist.cs | foreachchar.cs | dowhileeg.cs | whileeg.cs | Switch_char.cs | foreg.cs | Program.cs

Control.breakeg | Main()

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Control
{
    class breakeg
    {
        public static void Main()
        {
            for (int i = 0; i < 10; i++)
            {
                if (i == 5)
                {
                    break;
                }

                Console.WriteLine("Value of i: {0}", i);
            }
            Console.ReadKey();
        }
    }
}
```

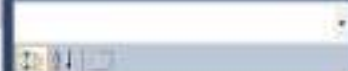
Solution Explorer

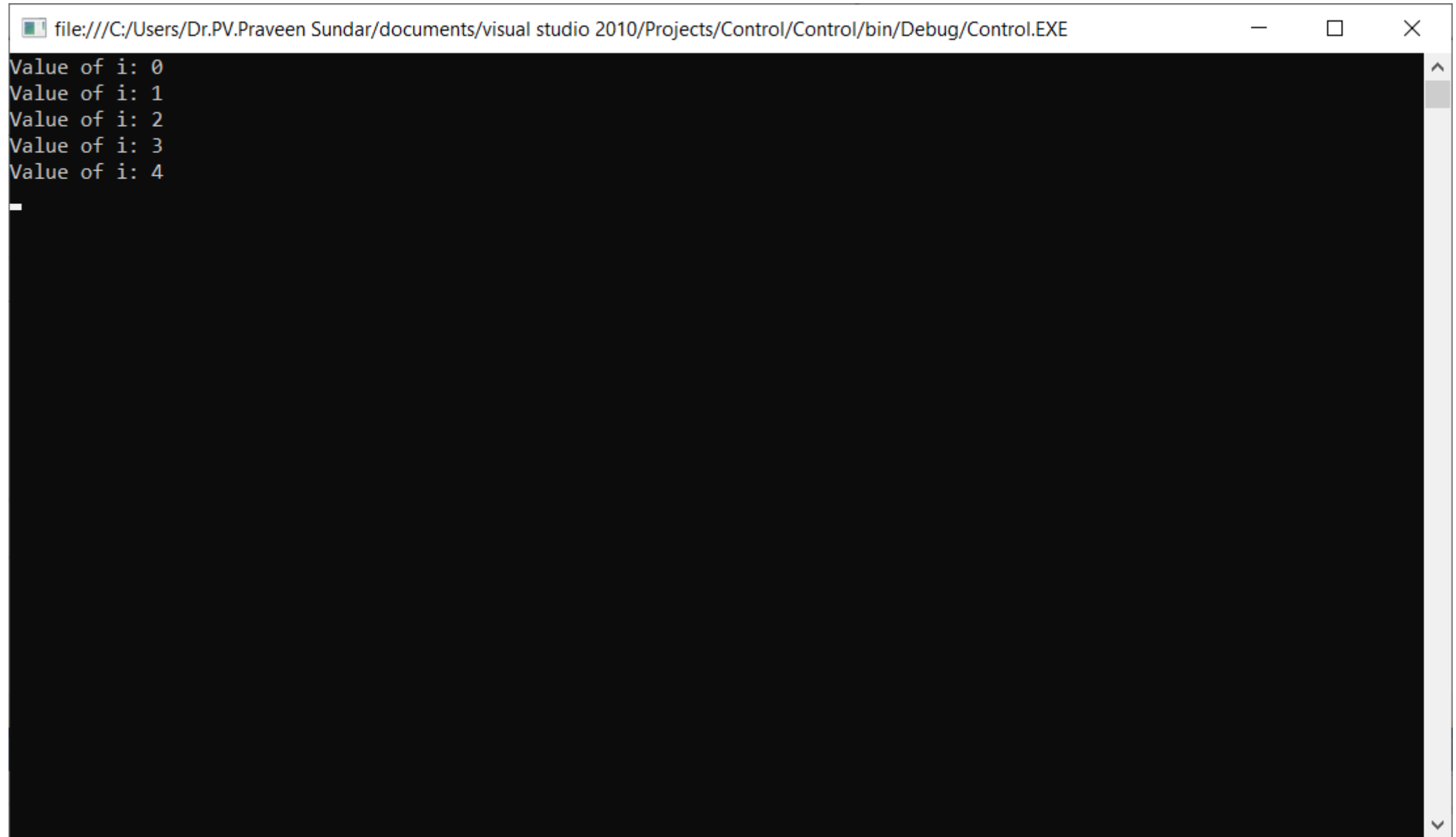
Solution 'Control' (1 project)

- Control
 - Properties
 - References
 - breakeg.cs
 - dowhileeg.cs
 - elseif.cs
 - foreachchar.cs
 - foreacheg.cs
 - foreachlist.cs
 - foreg.cs
 - ifelse.cs
 - nestedif.cs
 - Program.cs
 - Switch_calc.cs
 - Switch_char.cs
 - switchbool.cs
 - switchcase.cs
 - whileeg.cs

Solut... Team... Class...

Properties



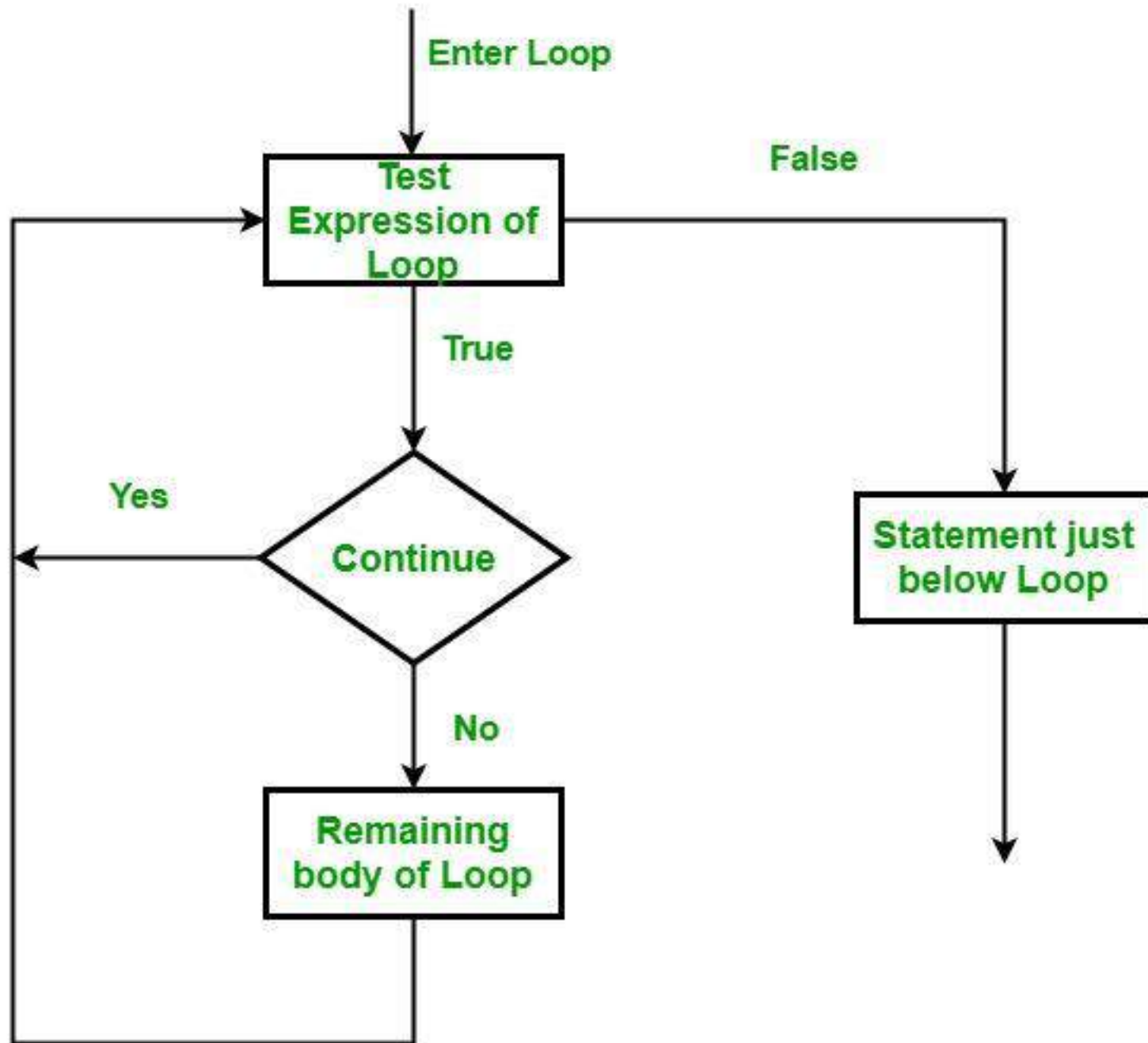


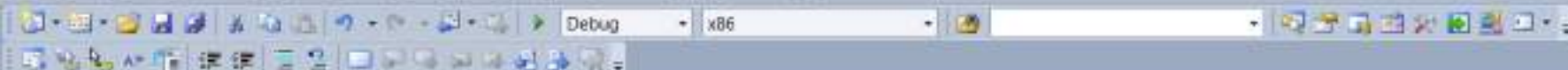
```
file:///C:/Users/Dr.PV.Praveen Sundar/documents/visual studio 2010/Projects/Control/Control/bin/Debug/Control.EXE
Value of i: 0
Value of i: 1
Value of i: 2
Value of i: 3
Value of i: 4
_
```

Continue Statements

90

- ❑ A Continue statement jumps out of the current loop condition and jumps back to the starting of the loop code.
- ❑ It is represented by continue;
- ❑ Continue statement can be used in the following scenarios:
 - ▣ for loop (For loop & nested for loop.
 - ▣ foreach loop (foreach loop & nested foreach loop.
 - ▣ While (while loop & nested while loop).
 - ▣ Do while (do while loop and nested while loop)
 - ▣ Switch case (Switch cases and nested switch cases)





Continueeeg.cs breakeg.cs foreachlist.cs foreachchar.cs dowhileeg.cs whileeg.cs Switch_char.cs foreg.cs Program.cs

ControlContinueeeg Main()

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Control
{
    class Continueeeg
    {
        public static void Main()
        {
            for (int i = 1; i <= 10; i++)
            {
                if (i % 2 == 0)
                    continue;

                Console.WriteLine("Value of i: {0}", i);
            }
            Console.ReadKey();
        }
    }
}
```

Solution Explorer

Solution 'Control' (1 project)

- Control
 - Properties
 - References
 - breakeg.cs
 - Continueeeg.cs
 - dowhileeg.cs
 - elseif.cs
 - foreachchar.cs
 - foreacheg.cs
 - foreachlist.cs
 - foreg.cs
 - ifelse.cs
 - nestedif.cs
 - Program.cs
 - Switch_calc.cs
 - Switch_char.cs
 - switchbool.cs
 - switchcase.cs

Solut... Team... Class...

Properties



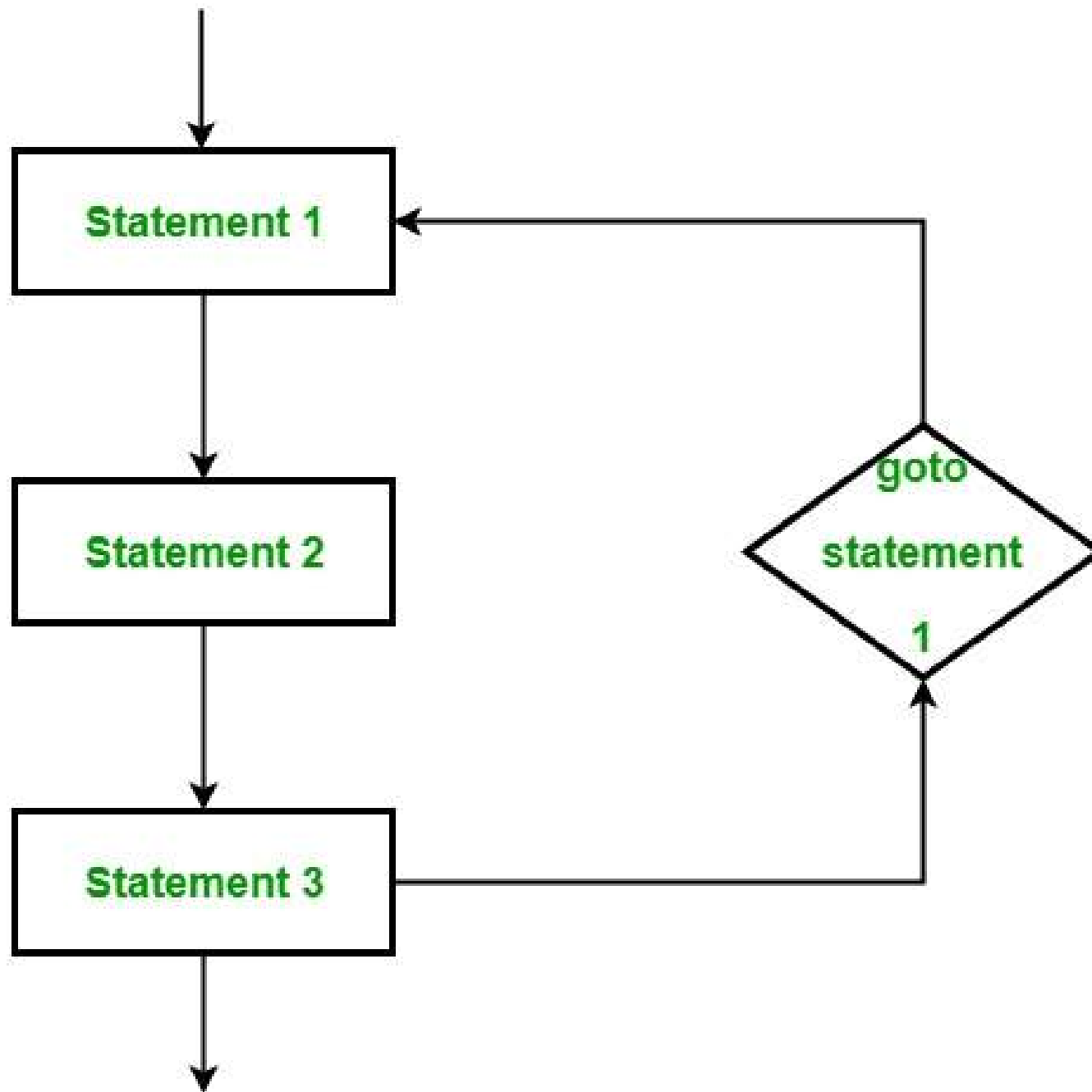
file:///C:/Users/Dr.PV.Praveen Sundar/documents/visual studio 2010/Projects/Control/Control/bin/Debug/Control.EXE

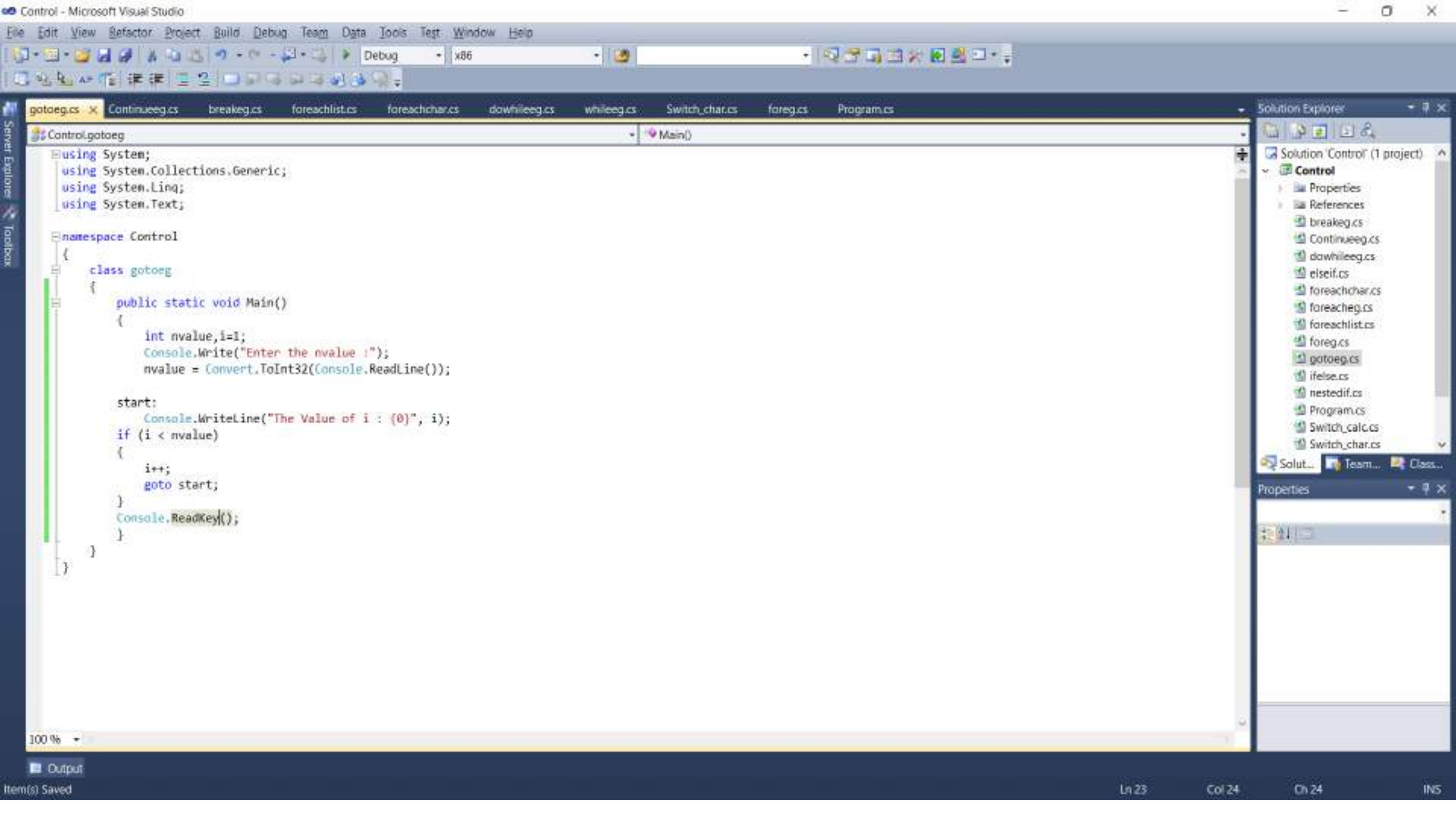
```
Value of i: 1  
Value of i: 3  
Value of i: 5  
Value of i: 7  
Value of i: 9  
_
```

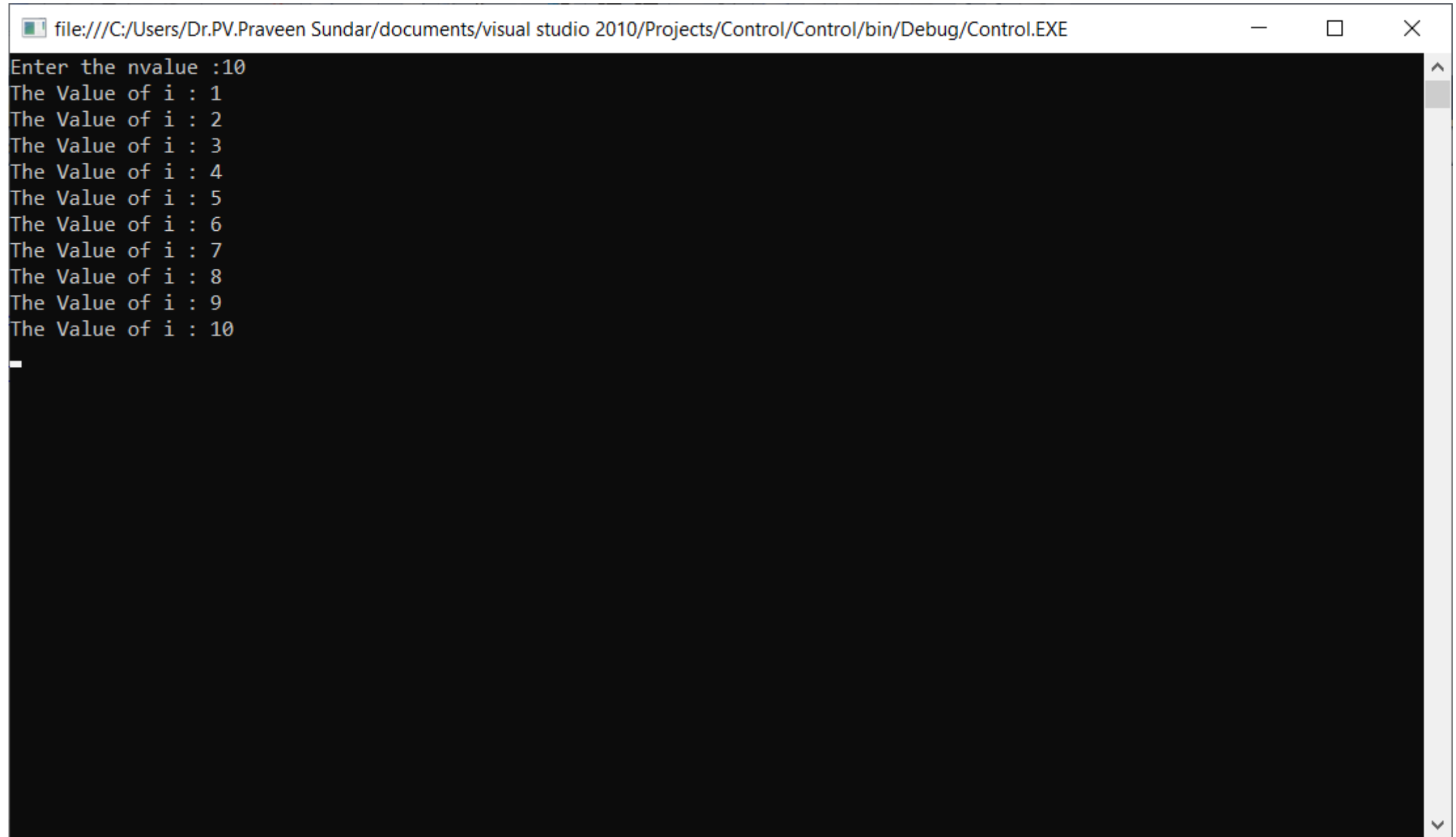

goto statement

94

- ❑ The goto statement transfers the program control directly to a labeled statement.
- ❑ The label is the valid identifier and placed just before the statement from where the control is transferred.
- ❑ 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.







The screenshot shows a Windows command prompt window with the title bar text: `file:///C:/Users/Dr.PV.Praveen Sundar/documents/visual studio 2010/Projects/Control/Control/bin/Debug/Control.EXE`. The window contains the following text:

```
Enter the nvalue :10
The Value of i : 1
The Value of i : 2
The Value of i : 3
The Value of i : 4
The Value of i : 5
The Value of i : 6
The Value of i : 7
The Value of i : 8
The Value of i : 9
The Value of i : 10
```

A small white cursor is visible on the line following the last output.

Program Exercise

98

//Question No-1: What is the Output of this Program

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

file:///D:/APCAS/2021- Odd Semester/II MCA/C#/Programs/Dotnet Programs/looping Statements/looping Statements/bin/Debug/lo... — □ ×

0
1
2
3
4
—

// Question No-2: What is the Output of this Program

```
int sum = 0;
for (int i, j, k = 0; sum <= 5; )
{
    sum = i + j + k;
    if (sum > 5)
        break;
    Console.WriteLine(k);
    k++;
}
```

// Question No-2: What is the Output of this Program

```
int sum = 0;
for (int i, j, k = 0; sum <= 5; )
{
    sum = i + j + k; // This produces Error because i and j are not initialized
    if (sum > 5)
        break;
    Console.WriteLine(k);
    k++;
}
```


// Question No-3: What is the Output of this Program?

```
int sum;  
for (int k = 0, j = 1; k <= 5; k++)  
{  
    sum = j + k;  
    if (sum > 5)  
        break;  
    Console.WriteLine(k);  
    k++;  
}  
Console.ReadKey();
```

file:///D:/APCAS/2021- Odd Semester/II MCA/C#/Programs/Dotnet Programs/looping Statements/looping Statements/bin/Debug/lo... — □ ×

0
2
4

// - Question No-4, What is the Output of this Program.

```
int i = 0;
float sum;
for (float f = 0; i <= 5; f = f + 0.10f)
{
    sum = f + i;
    Console.WriteLine(sum);
    i++;
}
Console.ReadKey();
```

file:///D:/APCAS/2021- Odd Semester/II MCA/C#/Programs/Dotnet Programs/looping Statements/looping Statements/bin/Debug/lo... — □ ×

```
0
1.1
2.2
3.3
4.4
5.5
```

```
// - Question No-5, What is the Output of this Program.  
char ch;  
for ( ; ; )  
{  
    ch = Char.Parse(Console.ReadLine());  
    if (ch == 'Y')  
        break;  
}  
Console.WriteLine("Program Ends");  
Console.ReadKey();
```

file:///D:/APCAS/2021- Odd Semester/II MCA/C#/Programs/Dotnet Programs/looping Statements/looping Statements/bin/Debug/lo... — □ ×

```
P  
R  
A  
V  
E  
E  
N  
S  
U  
N  
D  
A  
R  
P  
.  
V  
.  
Y  
Program Ends
```

```
// - Question No-6, What is the Output of this Program.  
char ch;  
for ( ; ; )  
{  
    ch = (char)Console.Read();  
    if (ch == 'Y')  
        break;  
}  
Console.WriteLine("Program Ends");
```

file:///D:/APCAS/2021- Odd Semester/II MCA/C#/Programs/Dotnet Programs/looping Statements/looping Statements/bin/Debug/lo... — □ ×

```
M
C
A
2
0
2
0
-
2
0
2
2
Y
Program Ends
-
```


Thank you



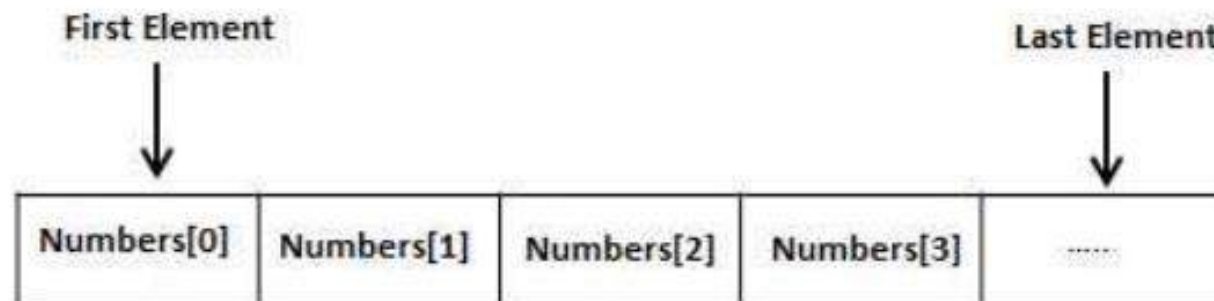
ARRAYS IN C#

Dr P.V. Praveen Sundar
Assistant Professor,
Department of Computer Science
Adhiparasakthi College of Arts & Science,
Kalavai.

Arrays

2

- ❑ A variable is used to store a literal value, whereas an array is used to store multiple literal values.
- ❑ An array is the data structure that stores a fixed number of literal values (elements) of the same data type.
- ❑ Array elements are stored contiguously in the memory.
- ❑ The lowest address corresponds to the first element and the highest address to the last element.



- ❑ In C#, an array can be of three types: Single-dimensional, Multidimensional, and Jagged array.
- ❑ In C#, array is an object of base type `System.Array`.
- ❑ In C#, array index starts from 0.
- ❑ In C#, arrays can be declared as fixed-length or dynamic.
- ❑ A fixed-length array can store a predefined number of items.
- ❑ A dynamic array does not have a predefined size. The size of a dynamic array increases as you add new items to the array.
- ❑ Its possible to change a dynamic array to static after it is defined.
- ❑ If we want the array to store elements of any type, we can specify object as its type. In the unified type system of C#, all types, predefined and user-defined, reference types and value types, inherit directly or indirectly from `Object`.

Declaring Arrays

To declare an array in C#, you can use the following syntax –

```
datatype[] arrayName;
```

where,

- *datatype* is used to specify the type of elements in the array.
- *[]* specifies the size of the array.
- *arrayName* specifies the name of the array.

For example,

```
double[] balance;
```

Initializing an Array

Declaring an array does not initialize the array in the memory. When the array variable is initialized, you can assign values to the array.

Array is a reference type, so you need to use the **new** keyword to create an instance of the array. For example,

```
double[] balance = new double[10];
```

```
class TestArraysClass
{
    static void Main()
    {
        // Declare a single-dimensional array of 5 integers.
        int[] array1 = new int[5];

        // Declare and set array element values.
        int[] array2 = new int[] { 1, 3, 5, 7, 9 };

        // Alternative syntax.
        int[] array3 = { 1, 2, 3, 4, 5, 6 };

        // Declare a two dimensional array.
        int[,] multiDimensionalArray1 = new int[2, 3];

        // Declare and set array element values.
        int[,] multiDimensionalArray2 = { { 1, 2, 3 }, { 4, 5, 6 } };

        // Declare a jagged array.
        int[][] jaggedArray = new int[6][];

        // Set the values of the first array in the jagged array structure.
        jaggedArray[0] = new int[4] { 1, 2, 3, 4 };
    }
}
```

The following example demonstrate invalid array declarations.

Example: Invalid Array Creation

```
//must specify the size
int[] evenNums = new int[];

//number of elements must be equal to the specified size
int[] evenNums = new int[5] { 2, 4 };

//cannot use var with array initializer
var evenNums = { 2, 4, 6, 8, 10};
```

Late Initialization

It is not necessary to declare and initialize an array in a single statement. You can first declare an array then initialize it later on using the new operator.

Example: Late Initialization

```
int[] evenNums;

evenNums = new int[5];
// or
evenNums = new int[] { 2, 4, 6, 8, 10 };
```

Assigning Values to an Array

You can assign values to individual array elements, by using the index number, like –

```
double[] balance = new double[10];  
balance[0] = 4500.0;
```

You can assign values to the array at the time of declaration, as shown –

```
double[] balance = { 2340.0, 4523.69, 3421.0};
```

You can also create and initialize an array, as shown –

```
int [] marks = new int[5] { 99, 98, 92, 97, 95};
```

You may also omit the size of the array, as shown –

```
int [] marks = new int[] { 99, 98, 92, 97, 95};
```

You can copy an array variable into another target array variable. In such case, both the target and source point to the same memory location –

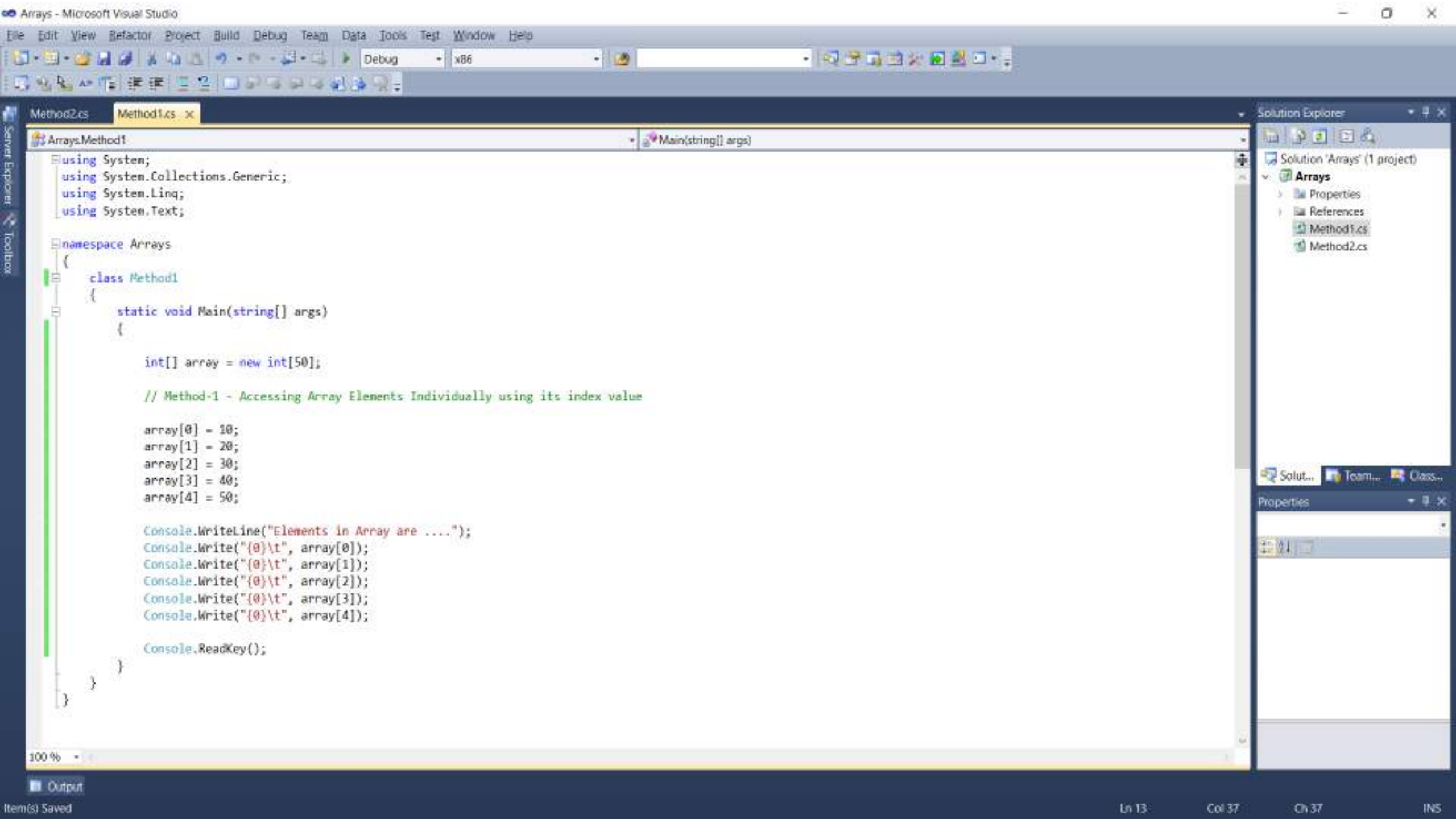
```
int [] marks = new int[] { 99, 98, 92, 97, 95};  
int[] score = marks;
```

When you create an array, C# compiler implicitly initializes each array element to a default value depending on the array type. For example, for an int array all elements are initialized to 0.

Accessing Array Elements

8

- ❑ Array elements can be accessed using an index.
- ❑ An index is a number associated with each array element, starting with index 0 and ending with array size - 1.
- ❑ There is a situation that we are trying to add more elements than its specified size will result in **IndexOutOfRangeException**.
- ❑ Arrays can be accessed in various methods like
 - ▣ Accessing individual elements.
 - ▣ Accessing using for loop.
 - ▣ Accessing using object.
 - ▣ Accessing using foreach statement.
 - ▣ Accessing using LINQ objects.



```
Arrays - Microsoft Visual Studio
File Edit View Refactor Project Build Debug Team Data Tools Test Window Help
Debug x86
Method2.cs Method1.cs x
Arrays.Method1 Main(string[] args)
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Arrays
{
    class Method1
    {
        static void Main(string[] args)
        {
            int[] array = new int[50];

            // Method-1 - Accessing Array Elements Individually using its index value

            array[0] = 10;
            array[1] = 20;
            array[2] = 30;
            array[3] = 40;
            array[4] = 50;

            Console.WriteLine("Elements in Array are ....");
            Console.Write("{0}\t", array[0]);
            Console.Write("{0}\t", array[1]);
            Console.Write("{0}\t", array[2]);
            Console.Write("{0}\t", array[3]);
            Console.Write("{0}\t", array[4]);

            Console.ReadKey();
        }
    }
}
Solution Explorer
Solution 'Arrays' (1 project)
  Arrays
    Properties
    References
    Method1.cs
    Method2.cs
Properties
100%
Output
Item(s) Saved
Ln 13 Col 37 Ch 37 INS
```

file:///c:/users/dr.pv.praveen sundar/documents/visual studio 2010/Projects/Arrays/Arrays/bin/Debug/Arrays.EXE

Elements in Array are

1020304050



Method2.cs Method1.cs

Arrays.Method1 Main(string[] args)

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Arrays
{
    class Method1
    {
        static void Main(string[] args)
        {
            int[] array = new int[5];

            // Method-1 - Accessing Array Elements Individually using its index value

            array[0] = 10;
            array[1] = 20;
            array[2] = 30;
            array[3] = 40;
            array[4] = 50;
            array[5] = 60;

            Console.WriteLine("Elements in Array are ....");
            Console.Write("{0}\t", array[0]);
            Console.Write("{0}\t", array[1]);
            Console.Write("{0}\t", array[2]);
            Console.Write("{0}\t", array[3]);
            Console.Write("{0}\t", array[4]);
            Console.Write("{0}\t", array[5]);
            Console.ReadKey();
        }
    }
}
```

Solution Explorer

Solution 'Arrays' (1 project)

- Arrays
 - Properties
 - References
 - Method1.cs
 - Method2.cs

Solution Team Class

Properties

Microsoft Visual Studio - Arrays (Debugging)

File Edit View Project Build Debug Team Data Tools Test Window Help

Process: [7572] Arrays.vshost.exe Thread: [4828] Main Thread Stack Frame: Arrays.exe!Arrays.Method1.Main(string[] a

Method1.cs

Arrays.Method1

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Arrays
{
    class Method1
    {
        static void Main(string[] args)
        {
            int[] array = new int[5];

            // Method-1 - Accessing Array Elements Individually using its index value

            array[0] = 10;
            array[1] = 20;
            array[2] = 30;
            array[3] = 40;
            array[4] = 50;
            array[5] = 60;

            Console.WriteLine("Array elements are:");
            Console.Write(" ");
            Console.WriteLine(" ");
            Console.WriteLine(" ");
            Console.WriteLine(" ");
            Console.WriteLine(" ");
            Console.WriteLine(" ");
            Console.ReadKey();
        }
    }
}
```

IndexOutOfRangeException was unhandled

Index was outside the bounds of the array.

Troubleshooting tips:

- Make sure that the maximum index on a list is less than the list size.
- Make sure the index is not a negative number.
- Make sure data column names are correct.
- Get general help for this exception.
- Search for more Help Online.

Actions:

- View Detail...
- Copy exception detail to the clipboard

Solution Explorer

Solution 'Arrays' (1 project)

- Arrays
 - Properties
 - References
 - Method1.cs

100 %

Ready

Ln 22 Col 13 Ch 13 INS



Method2.cs x Method1.cs

Arrays.Method2 Main()

```
namespace Arrays
{
    class Method2
    {
        public static void Main()
        {
            int[] array = new int[5];

            // Method-2 - Accessing Array Elements using For Loop

            Console.WriteLine(" Elements in an Array (Before Initialization are ....");
            for (int i = 0; i < 5; i++)
            {
                Console.Write(" \t {0} ", array[i]);
            }

            Console.WriteLine("\n\n \n Enter the elements of an array :");
            for (int i = 0; i < 5; i++)
            {
                Console.Write(" Enter the Value of array [{0}] :", i);
                array[i] = Convert.ToInt32(Console.ReadLine());
            }
            Console.WriteLine("\n\n \n Elements in an Array (After Initialization are ....");
            for (int i = 0; i < 5; i++)
            {
                Console.Write(" \t {0} ", array[i]);
            }
            Console.ReadKey();
        }
    }
}
```

100 %

Output

Elements in an Array (Before Initialization are

0 0 0 0 0

Enter the elements of an array :

Enter the Value of array [0] :10

Enter the Value of array [1] :20

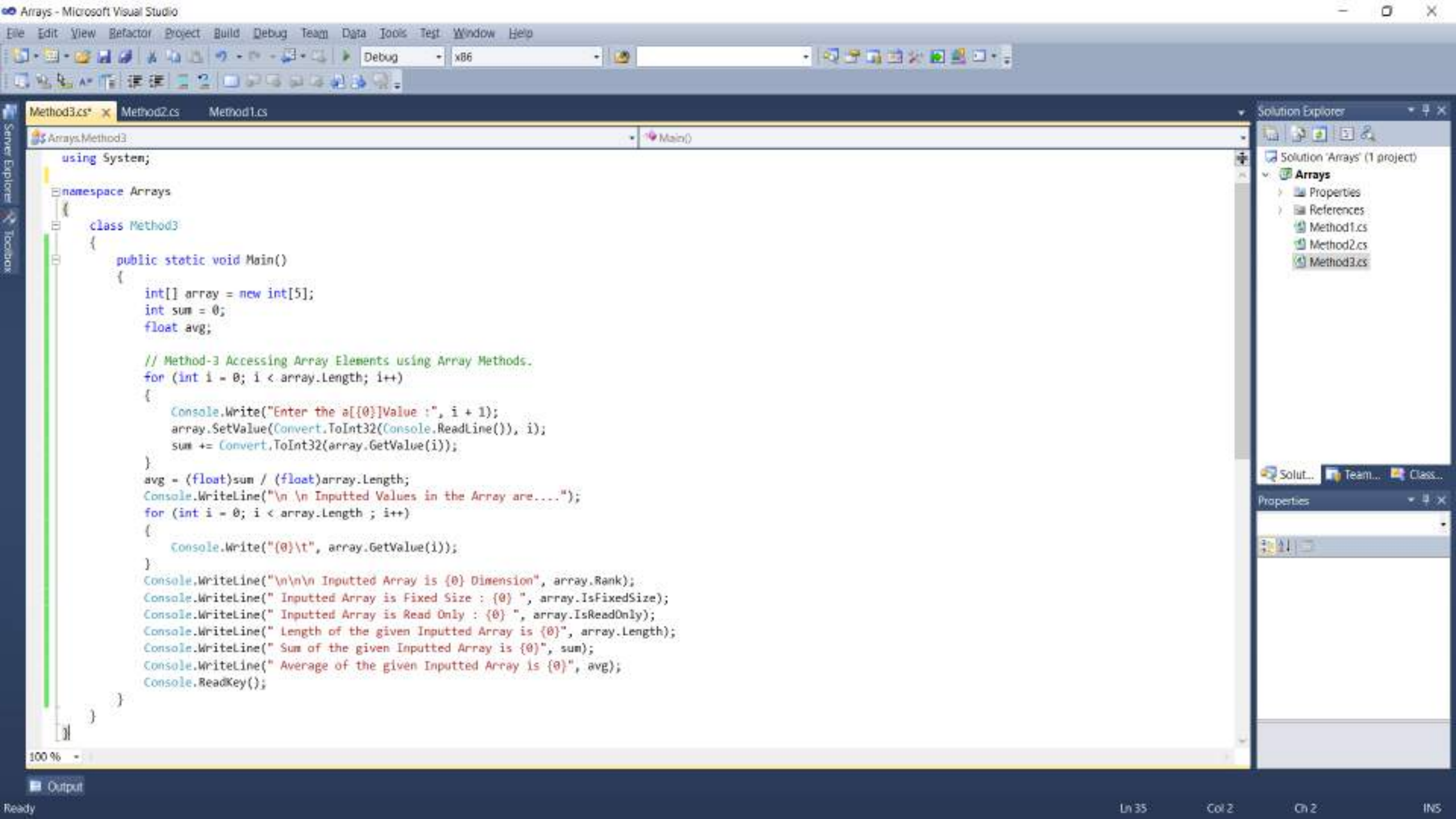
Enter the Value of array [2] :30

Enter the Value of array [3] :40

Enter the Value of array [4] :50

Elements in an Array (After Initialization are)

10 20 30 40 50




```
file:///c:/users/dr.pv.praveen sundar/documents/visual studio 2010/Projects/Arrays/Arrays/bin/Debug/Arrays.EXE
Enter the a[1]Value :10
Enter the a[2]Value :20
Enter the a[3]Value :30
Enter the a[4]Value :40
Enter the a[5]Value :50

Inputted Values in the Array are....
10      20      30      40      50

Inputted Array is 1 Dimension
Inputted Array is Fixed Size : True
Inputted Array is Read Only : False
Length of the given Inputted Array is 5
Sum of the given Inputted Array is 150
Average of the given Inputted Array is 30
```



Method4.cs Method3.cs Method2.cs Method1.cs

Arrays.Method4 Main()

```
class Method4
{
    public static void Main()
    {
        int[] array = new int[5];
        int sum = 0, i = 0;
        float avg;

        // Method-4 Accessing Array Elements using Foreach statement.
        foreach (int elements in array)
        {
            Console.WriteLine("Enter the a[{0}]Value :", i);
            array.SetValue(Convert.ToInt32(Console.ReadLine()), i);
            sum += Convert.ToInt32(array.GetValue(i));
            i++;
        }
        avg = (float)sum / (float)array.Length;
        Console.WriteLine("\n \n Inputted Values in the Array are....");
        foreach (int elements in array)
        {
            Console.WriteLine("{0}\t", elements);
        }
        Console.WriteLine("\n\n Inputted Array is {0} Dimension", array.Rank);
        Console.WriteLine(" Inputted Array is Fixed Size : {0} ", array.IsFixedSize);
        Console.WriteLine(" Inputted Array is Read Only : {0} ", array.IsReadOnly);
        Console.WriteLine(" Length of the given Inputted Array is {0}", array.Length);
        Console.WriteLine(" Sum of the given Inputted Array is {0}", sum);
        Console.WriteLine(" {0}", avg);
        Console.ReadKey();
    }
}
```

class System.String
Represents text as a series of Unicode characters.

Solution Explorer

Solution 'Arrays' (1 project)

- Arrays
 - Properties
 - References
 - Method1.cs
 - Method2.cs
 - Method3.cs
 - Method4.cs

Solut... Team... Class...

Properties

100 %

Output

Enter the a[0]Value :10
Enter the a[1]Value :20
Enter the a[2]Value :30
Enter the a[3]Value :49
Enter the a[4]Value :50

Inputted Values in the Array are....

10 20 30 49 50

Inputted Array is 1 Dimension

Inputted Array is Fixed Size : True

Inputted Array is Read Only : False

Length of the given Inputted Array is 5

Sum of the given Inputted Array is 159

Average of the given Inputted Array is 31.8

```
using System;
using System.Linq;

namespace Arrays
{
    class Method5
    {
        public static void Main()
        {
            int[] array = new int[5];
            int[] reverse = new int[5];
            int[] distarray = new int[5];
            // Method-5 Accessing Array Elements using LINQ Methods.
            for (int i = 0; i < array.Length; i++)
            {
                Console.WriteLine("Enter the a[{0}]Value :", i);
                array.SetValue(Convert.ToInt32(Console.ReadLine()), i);
            }
            Console.WriteLine("\n \n Inputted Values in the Array are....");
            foreach (int elements in array)
            {
                Console.WriteLine("{0}\t", elements);
            }
            reverse = array.Reverse().ToArray();
            distarray = array.Distinct().ToArray();
            Console.WriteLine("\n\n\n The Sum of the given Inputted Array is {0} Dimension", array.Sum());
            Console.WriteLine(" The Average of the given Inputted Array is : {0} ", array.Average());
            Console.WriteLine(" The Maximum value of the given Inputted Array is : {0} ", array.Max());
            Console.WriteLine(" The Minimum value of the given Inputted Array is : {0} ", array.Min());
            Console.WriteLine("\n \n Reverse of the given Array are....");
            foreach (int elements in reverse)
            {
                Console.WriteLine("{0}\t", elements);
            }
            Console.WriteLine("\n \n Distinct Values of the given Array are....");
            foreach (int elements in distarray)
            {
                Console.WriteLine("{0}\t", elements);
            }
            Console.ReadKey();
        }
    }
}
```

```
file:///c:/users/dr.pv.praveen sundar/documents/visual studio 2010/Projects/Arrays/Arrays/bin/Debug/Arrays.EXE
Enter the a[0]Value :10
Enter the a[1]Value :20
Enter the a[2]Value :25
Enter the a[3]Value :10
Enter the a[4]Value :20

Inputted Values in the Array are....
10      20      25      10      20

The Sum of the given Inputted Array is 85 Dimension
The Average of the given Inputted Array is : 17
The Maximum value of the given Inputted Array is : 25
The Minimum value of the given Inputted Array is : 10

Reverse of the given Array are....
20      10      25      20      10

Distinct Values of the given Array are....
10      20      25
```

Arrays.Method6
Main()

```
using System;

namespace Arrays
{
    class Method6
    {
        public static void Main()
        {
            // Method-6 : Copying from Another Array.
            int n;
            Console.WriteLine("\n Enter the N Value: ");
            n = Convert.ToByte(Console.ReadLine());

            int[] array = new int[n];
            int[] dup = new int[n];
            dup = array;

            for (int i = 0; i < n; i++)
            {
                Console.WriteLine("Enter the a[{0}]Value :", i + 1);
                array[i] = Convert.ToInt32(Console.ReadLine());
            }
            dup[2] = 60;
            Console.WriteLine("\n\nInputted Values in the Array are....");
            for (int i = 0; i < n; i++)
            {
                Console.WriteLine("{0}\t", array[i]);
                Console.WriteLine("{0}\t", dup[i]);
            }
            Console.ReadKey();
        }
    }
}
```

Solution Explorer

Solution 'Arrays' (1 project)

- Arrays
 - Properties
 - References
 - Method1.cs
 - Method2.cs
 - Method3.cs
 - Method4.cs
 - Method5.cs
 - Method6.cs

Solut... Team... Class...

Properties

file:///c:/users/dr.pv.praveen sundar/documents/visual studio 2010/Projects/Arrays/Arrays/bin/Debug/Arrays.EXE

Enter the N Value: 5
Enter the a[1]Value :10
Enter the a[2]Value :20
Enter the a[3]Value :30
Enter the a[4]Value :40
Enter the a[5]Value :50

Inputted Values in the Array are....
10 10 20 20 60 60 40 40 50 50 _



objectarray.cs Arrayinit.cs Method6.cs Method5.cs Method4.cs Method3.cs Method2.cs Method1.cs

Arrays.objectarray Main()

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Arrays
{
    class objectarray
    {
        public static void Main()
        {
            // Example for Object array...
            object[] obj = new object[5] { 10, 12.34, 'A', "String", 123456.45678 };
            for (int i = 0; i < 5; i++)
            {
                Console.WriteLine("{0}\n", obj[i]);
            }
            Console.ReadKey();
        }
    }
}
```

Solution Explorer

Solution 'Arrays' (1 project)

- Arrays
 - Properties
 - References
 - Arrayinit.cs
 - Method1.cs
 - Method2.cs
 - Method3.cs
 - Method4.cs
 - Method5.cs
 - Method6.cs
 - objectarray.cs

Solut... Team... Class...

Properties

100 %

Output

10
12.34
A
String
123456.4567

Multidimensional Arrays

25

- ❑ C# supports multidimensional arrays up to 32 dimensions.
- ❑ The multidimensional array can be declared by adding commas in the square brackets.
- ❑ For example, `[,]` declares two-dimensional array, `[, ,]` declares three-dimensional array, `[, , ,]` declares four-dimensional array, and so on.
- ❑ So, in a multidimensional array, no of commas = No of Dimensions + 1.

Example: Multidimensional Arrays

```
int[,] arr2d; // two-dimensional array
int[, ,] arr3d; // three-dimensional array
int[, , ,] arr4d ; // four-dimensional array
int[, , , ,] arr5d; // five-dimensional array
```



Arrays 2dMethod2.cs 2dMethod1.cs x objectarray.cs Arrayinit.cs Method6.cs Method5.cs Method4.cs Method3.cs Method2.cs Method1.cs

```
Arrays_2dMethod1
Main()

using System;

namespace Arrays
{
    class _2dMethod1
    {
        public static void Main()
        {
            // Accessing Two Dimensional Array using Individual elements

            int[,] matA = new int[5, 5];
            Console.WriteLine("\n Enter the Values");
            matA[0, 0] = Convert.ToInt32(Console.ReadLine());
            matA[0, 1] = Convert.ToInt32(Console.ReadLine());

            matA[1, 0] = Convert.ToInt32(Console.ReadLine());
            matA[1, 1] = Convert.ToInt32(Console.ReadLine());

            matA[2, 0] = Convert.ToInt32(Console.ReadLine());
            matA[2, 1] = Convert.ToInt32(Console.ReadLine());

            Console.WriteLine("\n\nInputted Values in the Array are....");
            Console.WriteLine("{0}\t", matA[0, 0]);
            Console.WriteLine("{0}\t\n", matA[0, 1]);

            Console.WriteLine("{0}\t", matA[1, 0]);
            Console.WriteLine("{0}\t\n", matA[1, 1]);

            Console.WriteLine("{0}\t", matA[2, 0]);
            Console.WriteLine("{0}\t\n", matA[2, 1]);

            Console.ReadKey();
        }
    }
}
```

Solution Explorer

Solution 'Arrays' (1 project)

- Arrays
 - Properties
 - References
 - 2dMethod1.cs
 - 2dMethod2.cs
 - Arrayinit.cs
 - Method1.cs
 - Method2.cs
 - Method3.cs
 - Method4.cs
 - Method5.cs
 - Method6.cs
 - objectarray.cs

Solut... Team... Class...

Properties

Enter the Values10

20

30

40

50

60

Inputted Values in the Array are....

10 20

30 40

50 60



2dMethod2.cs 2dMethod1.cs objectarray.cs Arrayinit.cs Method6.cs Method5.cs Method4.cs Method3.cs Method2.cs Method1.cs

Arrays_2dMethod2 Main()

```
using System;

namespace Arrays
{
    class _2dMethod2
    {
        public static void Main()
        {
            // Accessing Two Dimensional Array using For Loop
            int size;
            Console.WriteLine("Enter the Size of the array");
            size = Convert.ToInt32(Console.ReadLine());

            int[,] matA = new int[size, size];

            for (int i = 0; i < size; i++)
                for (int j = 0; j < size; j++)
                {
                    Console.WriteLine("Enter the a[{0},{1}]Value :", i + 1, j + 1);
                    matA[i, j] = Convert.ToInt32(Console.ReadLine());
                }

            Console.WriteLine("\n\nInputted Values in the Array are....");

            for (int i = 0; i < size; i++)
            {
                Console.WriteLine();
                for (int j = 0; j < size; j++)
                    Console.Write("\t {0}", matA[i, j]);
            }
            Console.ReadKey();
        }
    }
}
```

Solution Explorer

Solution 'Arrays' (1 project)

- Arrays
 - Properties
 - References
 - 2dMethod1.cs
 - 2dMethod2.cs
 - Arrayinit.cs
 - Method1.cs
 - Method2.cs
 - Method3.cs
 - Method4.cs
 - Method5.cs
 - Method6.cs
 - objectarray.cs

Solut... Team... Class...

Properties

100 %

Enter the Size of the array3

Enter the a[1,1]Value :10

Enter the a[1,2]Value :20

Enter the a[1,3]Value :30

Enter the a[2,1]Value :40

Enter the a[2,2]Value :50

Enter the a[2,3]Value :60

Enter the a[3,1]Value :70

Enter the a[3,2]Value :80

Enter the a[3,3]Value :90

Inputted Values in the Array are....

10	20	30
40	50	60
70	80	90

```
public static void Main()
{
    // Method-3 Accessing Array Elements using Array Methods.
    int rsize, csize, sum=0;
    Console.Write("Enter the row Size : ");
    rsize = Convert.ToInt32(Console.ReadLine());
    Console.Write("Enter the Column Size : ");
    csize = Convert.ToInt32(Console.ReadLine());

    int[,] matA = new int[rsize, csize];

    for (int i = 0; i < matA.GetLength(0); i++)
        for (int j = 0; j < matA.GetLength(1); j++)
        {
            Console.Write("Enter the a[{0},{1}]Value : ", i + 1, j + 1);
            matA.SetValue(Convert.ToInt32(Console.ReadLine()), i, j);
        }
    Console.WriteLine("\n\nInputted Values in the Array are....");

    for (int i = 0; i < matA.GetLength(0); i++)
    {
        sum = 0;
        Console.WriteLine();
        for (int j = 0; j < matA.GetLength(1); j++)
        {
            sum += matA[i, j];
            Console.Write("\t {0}", matA.GetValue(i, j));
        }
        Console.Write("\t = {0}", sum);
    }
    Console.WriteLine("\n");
    for (int i = 0; i < matA.GetLength(1); i++)
    {
        sum = 0;
        for (int j = 0; j < matA.GetLength(0); j++)
            sum += matA[j, i];
        Console.Write("\t {0}", sum);
    }
    Console.WriteLine(" \n The Inputted Matrix is {0} Dimension ", matA.Rank);
    Console.ReadKey();
}
```

Enter the row Size : 3
Enter the Column Size : 2
Enter the a[1,1]Value :10
Enter the a[1,2]Value :20
Enter the a[2,1]Value :30
Enter the a[2,2]Value :40
Enter the a[3,1]Value :50
Enter the a[3,2]Value :60

Inputted Values in the Array are....

10	20	= 30
30	40	= 70
50	60	= 110

90 120

The Inputted Matrix is 2 Dimension


```
namespace Arrays
{
    class _2dMethod4
    {
        public static void Main()
        {
            int rsize, csize;
            Console.Write("Enter the row Size : ");
            rsize = Convert.ToInt32(Console.ReadLine());
            Console.Write("Enter the Column Size : ");
            csize = Convert.ToInt32(Console.ReadLine());

            int[,] array = new int[rsize, csize];
            int sum = 0, i = 0, j = 0;
            float avg;

            // Method-4 Accessing Array Elements using Foreach statement.
            foreach (int elements in array)
            {
                if (j > array.GetUpperBound(1))
                {
                    i++; j = 0;
                }
                Console.Write("Enter the a[{0}][{1}] Value : ", i, j);
                array.SetValue(Convert.ToInt32(Console.ReadLine()), i, j);
                sum += Convert.ToInt32(array.GetValue(i, j));
                j++;
            }
            avg = (float)sum / (float)array.Length;
            Console.WriteLine("\n \n Inputted Values in the Array are....");
            foreach (int elements in array)
            {
                Console.Write("{0}\t", elements);
            }
            Console.WriteLine("\n\n\n Inputted Array is {0} Dimension", array.Rank);
            Console.WriteLine(" Inputted Array is Fixed Size : {0} ", array.IsFixedSize);
            Console.WriteLine(" Inputted Array is Read Only : {0} ", array.IsReadOnly);
            Console.WriteLine(" Length of the given Inputted Array is {0}", array.Length);
            Console.WriteLine(" Sum of the given Inputted Array is {0}", sum);
            Console.WriteLine(" Average of the given Inputted Array is {0}", avg);
            Console.ReadKey();
        }
    }
}
```

Enter the row Size : 2
Enter the Column Size : 3
Enter the a[0][0]]Value :10
Enter the a[0][1]]Value :20
Enter the a[0][2]]Value :30
Enter the a[1][0]]Value :40
Enter the a[1][1]]Value :50
Enter the a[1][2]]Value :60

Inputted Values in the Array are....
10 20 30 40 50 60

Inputted Array is 2 Dimension
Inputted Array is Fixed Size : True
Inputted Array is Read Only : False
Length of the given Inputted Array is 6
Sum of the given Inputted Array is 210
Average of the given Inputted Array is 35



2dMethod5.cs 2dMethod4.cs 2dMethod3.cs 2dMethod2.cs 2dMethod1.cs objectarray.cs Arrayinit.cs Method6.cs Method5.cs Method4.cs Method3.cs Method2.cs

Arrays_2dMethod5 Main()

```
{
    class _2dMethod5
    {
        public static void Main()
        {
            int rsize, csize;
            Console.Write("Enter the row Size : ");
            rsize = Convert.ToInt32(Console.ReadLine());
            Console.Write("Enter the Column Size : ");
            csize = Convert.ToInt32(Console.ReadLine());

            int[,] array = new int[rsize, csize];
            int[,] dup = new int[rsize, csize];
            dup = array;

            for (int i = 0; i < rsize; i++)
                for (int j = 0; j < csize; j++)
                {
                    Console.Write("Enter the a[{0}]Value :", i + 1);
                    array[i, j] = Convert.ToInt32(Console.ReadLine());
                }
            dup[0, 1] = 60;
            Console.WriteLine("\n\nInputted Values in the Array are....");
            for (int i = 0; i < rsize; i++)
            {
                Console.WriteLine();
                for (int j = 0; j < csize; j++)
                {
                    Console.Write("{0}\t", array[i, j]);
                    Console.Write("{0}\t", dup[i, j]);
                }
            }
            Console.ReadKey();
        }
    }
}
```

100 %

Solution Explorer

Solution 'Arrays' (1 project)

- Arrays
 - Properties
 - References
 - 2dMethod1.cs
 - 2dMethod2.cs
 - 2dMethod3.cs
 - 2dMethod4.cs
 - 2dMethod5.cs
 - Arrayinit.cs
 - Method1.cs
 - Method2.cs
 - Method3.cs
 - Method4.cs
 - Method5.cs
 - Method6.cs
 - objectarray.cs

Solut... Team... Class...

Properties

Enter the row Size : 2
Enter the Column Size : 3
Enter the a[1]Value :10
Enter the a[1]Value :20
Enter the a[1]Value :30
Enter the a[2]Value :40
Enter the a[2]Value :45
Enter the a[2]Value :50

Inputted Values in the Array are....

10	10	60	60	30	30
40	40	45	45	50	50

Jagged Arrays

36

- ❑ Jagged array is an array of arrays such that member arrays can be of different sizes.
- ❑ In other words, the length of each array index can differ.
- ❑ The elements of Jagged Array are reference types and initialized to null by default.
- ❑ Jagged Array can also be mixed with multidimensional arrays. Here, the number of rows will be fixed at the declaration time, but you can vary the number of columns.
- ❑ In Jagged arrays, user has to provide the number of rows only. If the user is also going to provide the number of columns, then this array will be no more Jagged Array.
- ❑ The elements of Jagged Array must be initialized before its use.

Syntax:

```
data_type[][] name_of_array = new data_type[rows][]
```

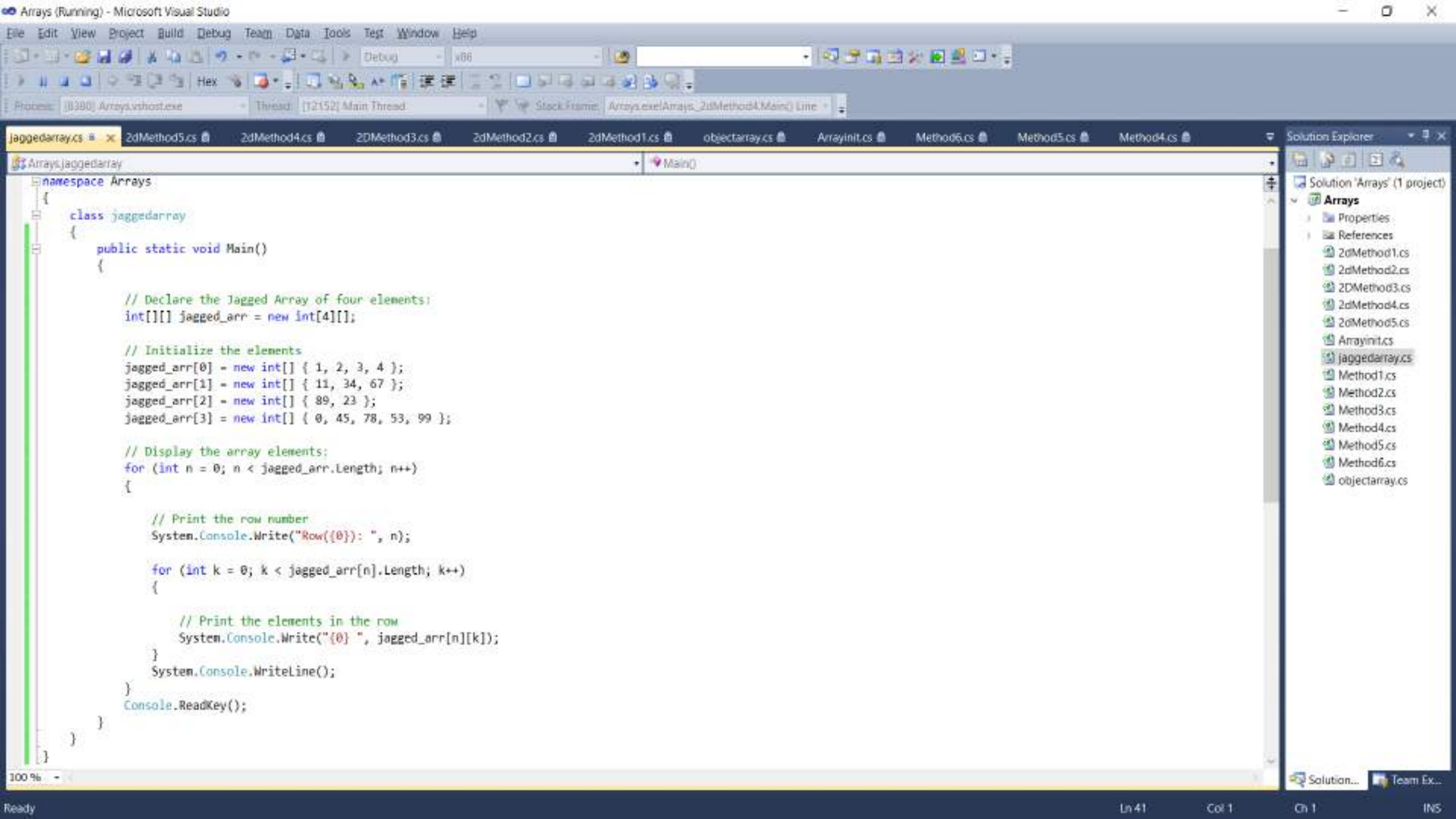
Example:

```
int[][] jagged_arr = new int[4][]
```

Declaration as well as Initialization

Example 1: Using the Direct Method

```
int[][] jagged_arr = new int[][]  
{  
    new int[] {1, 2, 3, 4},  
    new int[] {11, 34, 67},  
    new int[] {89, 23},  
    new int[] {0, 45, 78, 53, 99}  
};
```

file:///C:/Users/Dr.PV.Praveen Sundar/documents/visual studio 2010/Projects/Arrays/Arrays/bin/Debug/Arrays.EXE

Row(0): 1 2 3 4
Row(1): 11 34 67
Row(2): 89 23
Row(3): 0 45 78 53 99

Array Class

41

- ❑ Array Class provides methods for creating, manipulating, searching, and sorting arrays, thereby serving as the base class for all arrays in the common language runtime.
- ❑ Array class is defined in the System namespace, is the base class for arrays in C#.
- ❑ Array class is an abstract base class that means we cannot create an instance of the Array class.

Properties	Definition
IsFixedSize	Return a value indicating if an array has a fixed size or not.
IsReadOnly	Returns a value indicating if an array is read-only or not.
LongLength	Returns a 64-bit integer that represents a total number of items in all the dimensions of an array.
Length	Returns a 32-bit integer that represents the total number of items in all the dimensions of an array.
Rank	Returns the number of dimensions of an array.

BinarySearch()	Searches a one-dimensional sorted Array for a value, using a binary search algorithm.
Copy()	Copy array elements to another elements
Resize()	Changes the number of elements of a one-dimensional array to the specified new size.
Clear()	Sets a range of elements in the Array to zero, to false, or to null, depending on the element type.
IndexOf ()	Searches for the specified object and returns the index of its first occurrence in a one-dimensional array or in a range of elements in the array.
LastIndexOf ()	Returns the index of the last occurrence of a value in a one-dimensional Array or in a portion of the Array.
Sort()	Sorts the elements in a one-dimensional array.
Reverse()	Reverses the order of the elements in a one-dimensional Array or in a portion of the Array

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace ArrayClass
{
    class Method_1
    {
        static void Main(string[] args)
        {
            int[] array = new int[10];
            Array A= array;
            int value, index;
            for (int i = 0; i < 5; i++)
            {
                Console.Write(" Enter the Values :");
                A.SetValue(Convert.ToInt32(Console.ReadLine()), i);
            }
            Console.WriteLine("\n Entered Values are ...");
            for (int i = 0; i < 5; i++)
            {
                Console.Write("\t {0}", array.GetValue(i));
            }
            Console.WriteLine("\n");
            // Binary Search
            Console.Write(" Enter the Value to Search :");
            value = Convert.ToInt32(Console.ReadLine());
            index = Array.BinarySearch(array, value);

            if (index > 0)
            {
                Console.WriteLine("{0} is found at {1} Position ", value, index);
            }
            else
            {
                Console.WriteLine("Item Not Found");
            }
            Console.ReadKey();
        }
    }
}
```

```
file:///c:/users/dr.pv.praveen sundar/documents/visual studio 2010/Projects/ArrayClass/ArrayClass/bin/Debug/ArrayClass.EXE
Enter the Values :10
Enter the Values :20
Enter the Values :30
Enter the Values :40
Enter the Values :50

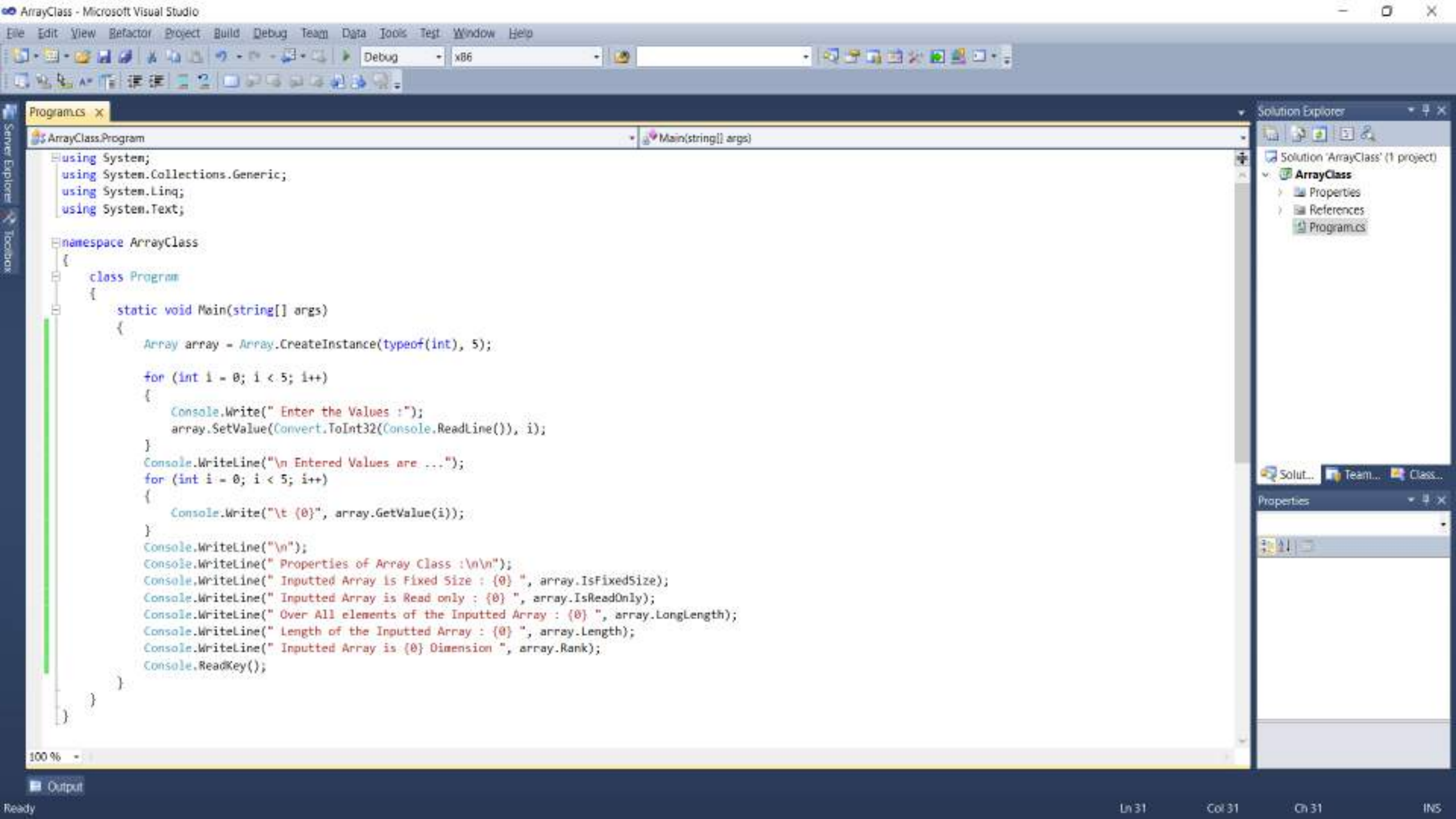
Entered Values are ...
    10    20    30    40    50

Enter the Value to Search :30
30 is found at 2 Position
```

Enter the Values :10
Enter the Values :20
Enter the Values :30
Enter the Values :40
Enter the Values :50

Entered Values are ...
10 20 30 40 50

Enter the Value to Search :60
Item Not Found



```
ArrayClass - Microsoft Visual Studio
File Edit View Refactor Project Build Debug Team Data Tools Test Window Help
Debug x86
Program.cs x
ArrayClass.Program Main(string[] args)
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace ArrayClass
{
    class Program
    {
        static void Main(string[] args)
        {
            Array array = Array.CreateInstance(typeof(int), 5);

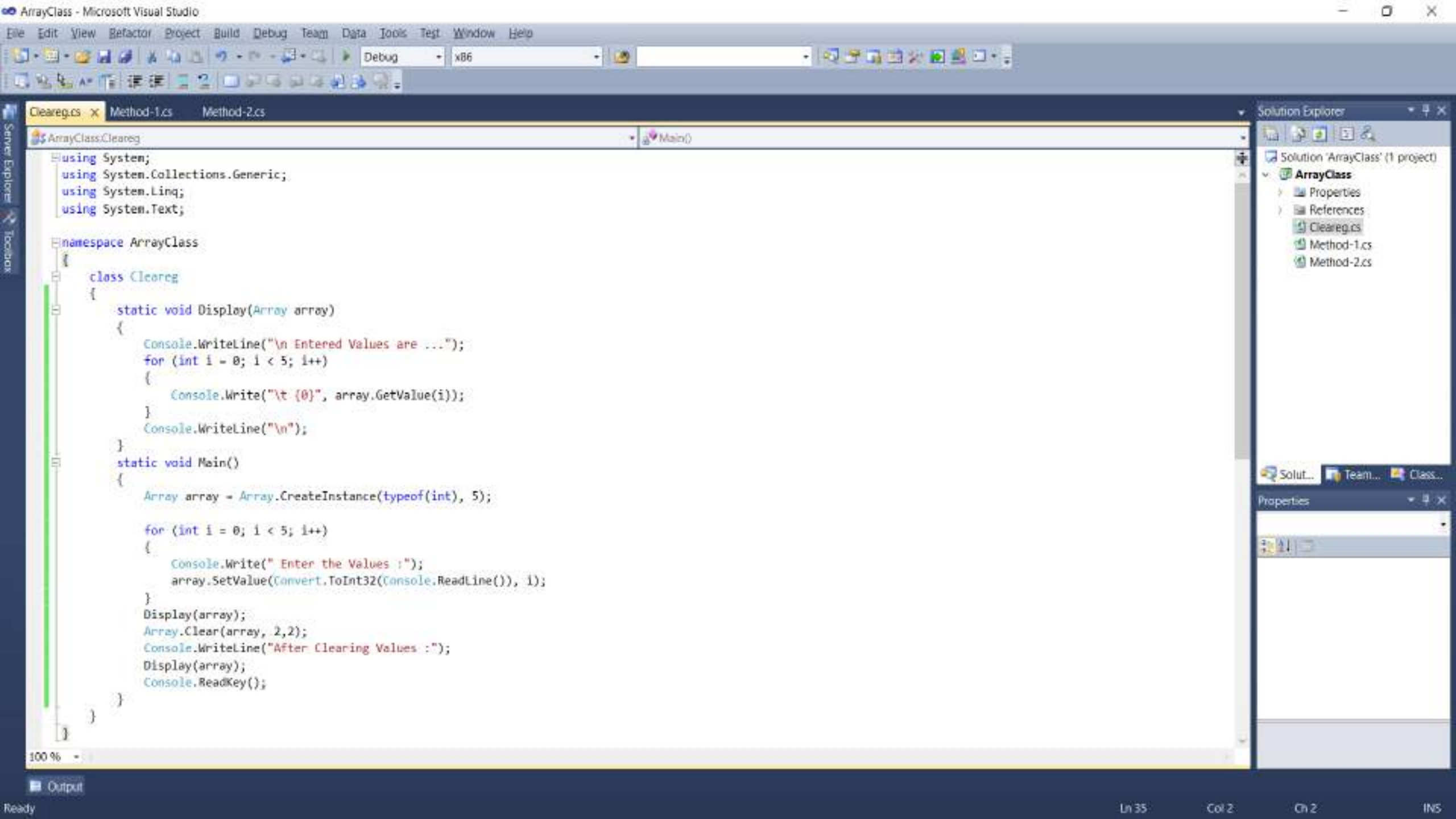
            for (int i = 0; i < 5; i++)
            {
                Console.Write(" Enter the Values :");
                array.SetValue(Convert.ToInt32(Console.ReadLine()), i);
            }
            Console.WriteLine("\n Entered Values are ...");
            for (int i = 0; i < 5; i++)
            {
                Console.Write("\t {0}", array.GetValue(i));
            }
            Console.WriteLine("\n");
            Console.WriteLine(" Properties of Array Class :\n\n");
            Console.WriteLine(" Inputted Array is Fixed Size : {0} ", array.IsFixedSize);
            Console.WriteLine(" Inputted Array is Read only : {0} ", array.IsReadOnly);
            Console.WriteLine(" Over All elements of the Inputted Array : {0} ", array.LongLength);
            Console.WriteLine(" Length of the Inputted Array : {0} ", array.Length);
            Console.WriteLine(" Inputted Array is {0} Dimension ", array.Rank);
            Console.ReadKey();
        }
    }
}
```

Enter the Values :10
Enter the Values :20
Enter the Values :30
Enter the Values :40
Enter the Values :50

Entered Values are ...
10 20 30 40 50

Properties of Array Class :

Inputted Array is Fixed Size : True
Inputted Array is Read only : False
Over All elements of the Inputted Array : 5
Length of the Inputted Array : 5
Inputted Array is 1 Dimension



```
file:///c:/users/dr.pv.praveen sundar/documents/visual studio 2010/Projects/ArrayClass/ArrayClass/bin/Debug/ArrayClass.EXE
Enter the Values :10
Enter the Values :20
Enter the Values :30
Enter the Values :40
Enter the Values :50

Entered Values are ...
    10    20    30    40    50

After Clearing Values :

Entered Values are ...
    10    20    0    0    50
```

```
ArrayClass.arraycopy Main()
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace ArrayClass
{
    class arraycopy
    {
        static void Display(Array array)
        {
            Console.WriteLine("\n Entered Values are ...");
            for (int i = 0; i < 5; i++)
            {
                Console.Write("\t {0}", array.GetValue(i));
            }
            Console.WriteLine("\n");
        }

        static void Main()
        {
            int[] array = new int[10];
            int[] duparray = new int[10];
            Array A = array;

            for (int i = 0; i < 5; i++)
            {
                Console.Write(" Enter the Values :");
                A.SetValue(Convert.ToInt32(Console.ReadLine()), i);
            }
            Console.WriteLine("\n Entered Values are ...");
            Display(array);
            Array.Copy(array, duparray, 2);
            Console.WriteLine("\n New Array;");
            Display(duparray);
            Console.WriteLine("\n");
            Console.ReadKey();
        }
    }
}
```

file:///c:/users/dr.pv.praveen sundar/documents/visual studio 2010/Projects/ArrayClass/ArrayClass/bin/Debug/ArrayClass.EXE

Enter the Values :10
Enter the Values :20
Enter the Values :30
Enter the Values :40
Enter the Values :50

Entered Values are ...

Entered Values are ...
10 20 30 40 50

New Array;

Entered Values are ...
10 20 0 0 0

```
namespace ArrayClass
{
    class sort_reverse
    {
        static void Display(Array array)
        {
            for (int i = 0; i < 5; i++)
            {
                Console.Write("\t {0}", array.GetValue(i));
            }
            Console.WriteLine("\n");
        }

        static void Main()
        {
            Array array = Array.CreateInstance(typeof(int), 5);

            for (int i = 0; i < 5; i++)
            {
                Console.Write(" Enter the Values :");
                array.SetValue(Convert.ToInt32(Console.ReadLine()), i);
            }
            Console.WriteLine("\n Entered Values are ...");
            Display(array);

            // Reversing an Array
            Array.Reverse(array);
            Console.WriteLine("\n Reverse Array;\n");
            Display(array);

            // Sorting an Array
            Array.Sort(array);
            Console.WriteLine("\n Sorted Array;\n");
            Display(array);
            Console.WriteLine("\n");
            Console.ReadKey();
        }
    }
}
```

Enter the Values :25
Enter the Values :35
Enter the Values :10
Enter the Values :20
Enter the Values :15

Entered Values are ...
25 35 10 20 15

Reverse Array;
15 20 10 35 25

Sorted Array;
10 15 20 25 35

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace ArrayClass
{
    class resizearray
    {
        static void Display(Array array)
        {
            Console.WriteLine("\n Entered Values are ...");
            for (int i = 0; i < array.Length; i++)
            {
                Console.Write("\t {0}", array.GetValue(i));
            }
            Console.WriteLine("\n");
        }

        static void Main()
        {
            int[] array = new int[5];
            Array A = array;

            for (int i = 0; i < 5; i++)
            {
                Console.Write(" Enter the Values :");
                A.SetValue(Convert.ToInt32(Console.ReadLine()), i);
            }
            Console.WriteLine("\n Entered Values are ...");
            Display(array);
            Console.WriteLine("The Length of the Array (Before Resize) is {0}", array.Length);
            Array.Resize(ref array, 10);
            Console.WriteLine("The Length of the Array (After Resize) is {0}", array.Length);
            Display(array);
            Console.WriteLine("\n");
            Console.ReadKey();
        }
    }
}
```

Enter the Values :10

Enter the Values :20

Enter the Values :30

Enter the Values :49

Enter the Values :50

Entered Values are ...

Entered Values are ...

10 20 30 49 50

The Length of the Array (Before Resize) is 5

The Length of the Array (After Resize) is 10

Entered Values are ...

10 20 30 49 50 0 0 0 0 0


```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace ArrayClass
{
    class findarray
    {
        static void Display(Array array)
        {
            Console.WriteLine("\n Entered Values are ...");
            for (int i = 0; i < array.Length; i++)
            {
                Console.Write("\t {0}", array.GetValue(i));
            }
            Console.WriteLine("\n");
        }
        static void Main()
        {
            Array array = Array.CreateInstance(typeof(int), 5);
            int index, value;
            for (int i = 0; i < 5; i++)
            {
                Console.Write(" Enter the Values :");
                array.SetValue(Convert.ToInt32(Console.ReadLine()), i);
            }
            Console.WriteLine("\n Entered Values are ...");
            Display(array);
            Console.WriteLine("Enter the Value to Search");
            value = Convert.ToInt32(Console.ReadLine());
            Console.WriteLine("Given Value is found at {0} Position", Array.IndexOf(array, value));
            Console.WriteLine("Last Index of Given Value is found at {0} Position", Array.LastIndexOf(array, value));
            Console.ReadKey();
        }
    }
}
```

Enter the Values :10

Enter the Values :20

Enter the Values :10

Enter the Values :20

Enter the Values :10

Entered Values are ...

Entered Values are ...

10 20 10 20 10

Enter the Value to Search

10

Given Value is found at 0 Position

Last Index of Given Value is found at 4 Position

```
file:///c:/users/dr.pv.praveen sundar/documents/visual studio 2010/Projects/ArrayClass/ArrayClass/bin/Debug/ArrayClass.EXE
Enter the Values :10
Enter the Values :20
Enter the Values :30
Enter the Values :40
Enter the Values :50

Entered Values are ...

Entered Values are ...
    10    20    30    40    50

Enter the Value to Search
60
Given Value is found at -1 Position
Last Index of Given Value is found at -1 Position
```

ArrayList

59

- ❑ In C#, the ArrayList represents an ordered collection of an object that can be indexed individually.
- ❑ It is basically an alternative to an array.
- ❑ However, unlike array you can add and remove items from a list at a specified position using an index and the array resizes itself automatically.
- ❑ It also allows dynamic memory allocation, adding, searching and sorting items in the list.

PROPERTIES	Definition
Capacity	Gets or sets the number of elements that the ArrayList can contain.
Count	Gets the number of elements actually contained in the ArrayList.
IsFixedSize	Gets a value indicating whether the ArrayList has a fixed size.
IsReadOnly	Gets a value indicating whether the ArrayList is read-only.
Item[Int32]	Gets or sets the element at the specified index.

Methods	Description
Add()	Add() method adds single elements at the end of ArrayList.
AddRange()	AddRange() method adds all the elements from the specified collection into ArrayList.
Insert()	Insert() method insert a single elements at the specified index in ArrayList.
InsertRange()	InsertRange() method insert all the elements of the specified collection starting from specified index in ArrayList.
Remove()	Remove() method removes the specified element from the ArrayList.
RemoveRange()	RemoveRange() method removes a range of elements from the ArrayList.
RemoveAt()	Removes the element at the specified index from the ArrayList.
Sort()	Sorts entire elements of the ArrayList.
Reverse()	Reverses the order of the elements in the entire ArrayList.
Contains	Checks whether specified element exists in the ArrayList or not. Returns true if exists otherwise false.
Clear	Removes all the elements in ArrayList.
CopyTo	Copies all the elements or range of elements to compitible Array.
GetRange	Returns specified number of elements from specified index from ArrayList.
IndexOf	Search specified element and returns zero based index if found. Returns -1 if element not found.
ToArray	Returns compitible array from an ArrayList.

Thank you



STRINGS IN C#

Dr P.V. Praveen Sundar
Assistant Professor,
Department of Computer Science
Adhiparasakthi College of Arts & Science,
Kalavai.

Character Handling

2

- The `System.Char` data type is used to hold a single, unicode character.
- C# has an alias for it, called `char`, which you can use when declaring your char variables:
`char ch;`
- The default value of the `char` type is `\0`.
- The `char` type supports comparison, equality, increment, and decrement operators. Moreover, for `char` operands, arithmetic and bitwise logical operators perform an operation on the corresponding character codes and produce the result of the `int` type.
- we can specify a `char` value with:
 - a character literal.
 - a Unicode escape sequence, which is `\u` followed by the four-symbol hexadecimal representation of a character code.
 - a hexadecimal escape sequence, which is `\x` followed by the hexadecimal representation of a character code.

- Since a string is basically just a range of characters, .NET actually uses a list of char's to represent a string. That also means that you can pull out a single char from a string, or iterate over a string and get each character as a char data type:

```
string helloWorld = "Hello, world!";  
foreach(char c in helloWorld)  
{  
    Console.WriteLine(c);  
}
```

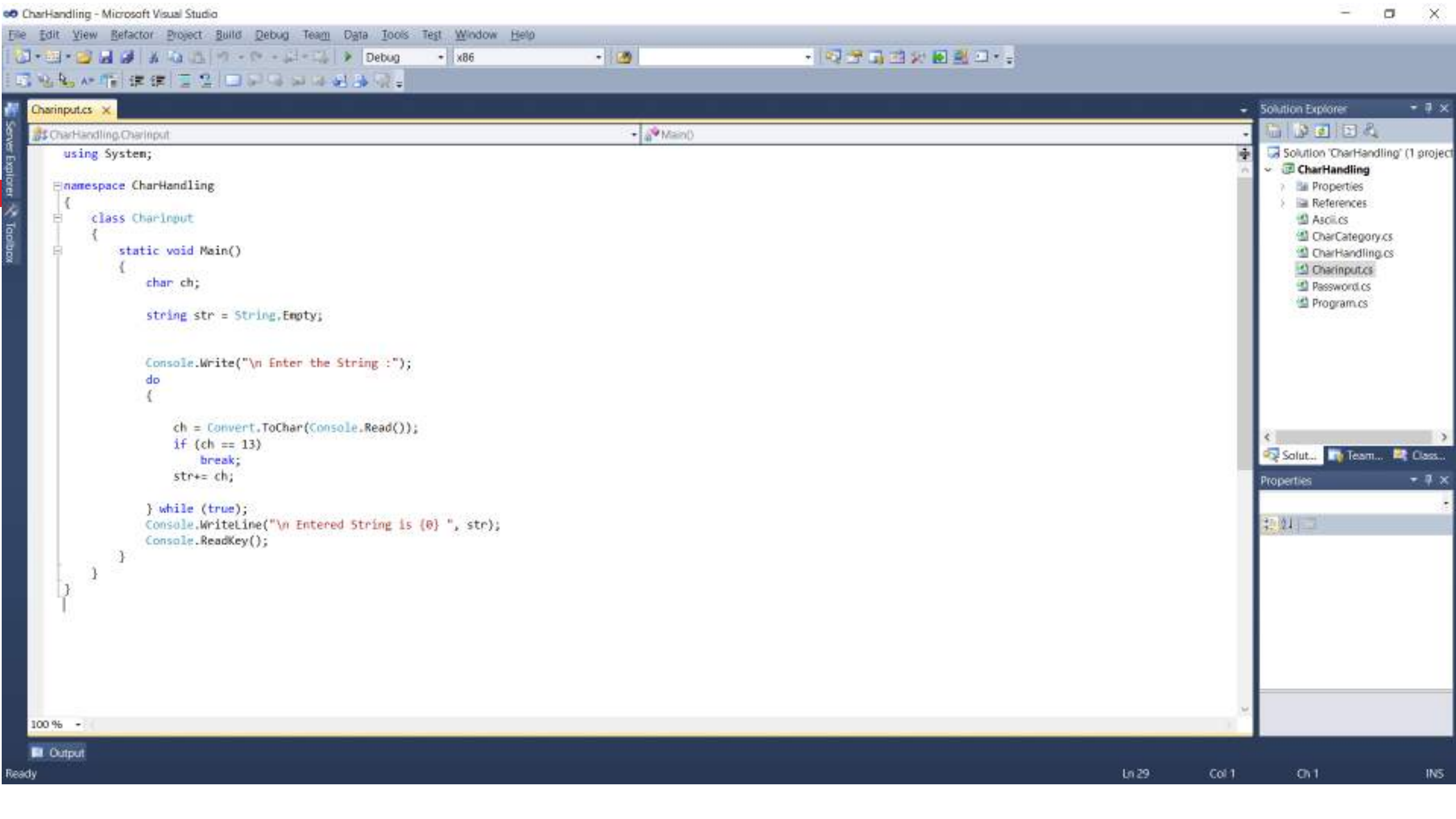
- Generally char is also considered as a numerical value, where each character has a specific number in the Unicode "alphabet".
- In C#, it is very easy to go from a char data type to its numeric representation.

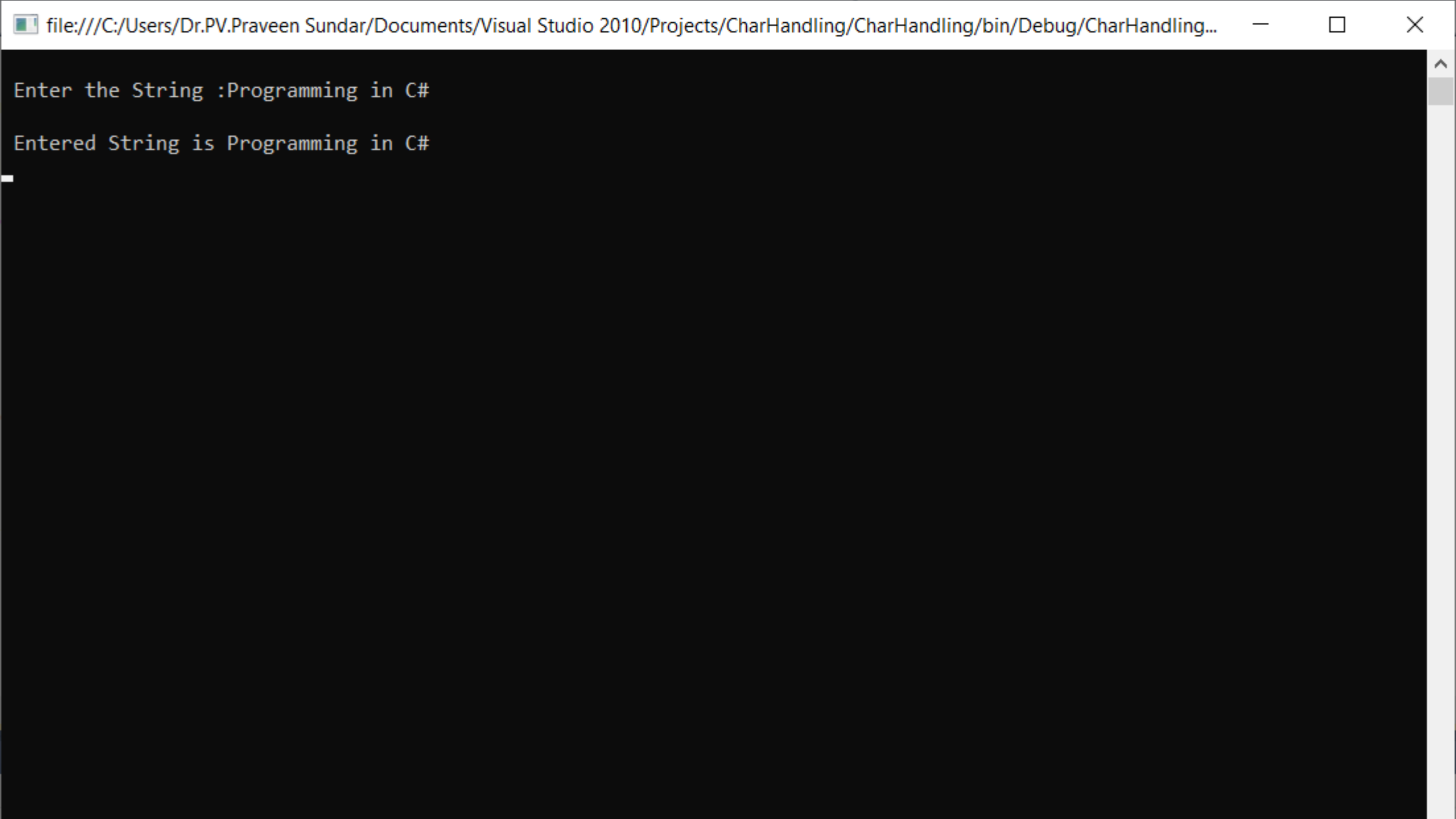
```
string helloWorld = "Hello, world!";  
foreach(char c in helloWorld)  
{  
    Console.WriteLine(c + ": " + (int)c);  
}
```

Properties	Description
MaxValue	Represents the largest possible value of a Char. This field is constant.
MinValue	Represents the smallest possible value of a Char. This field is constant

Methods	Description
CompareTo(Char)	Compares this instance to a specified Char object and indicates whether this instance precedes, follows, or appears in the same position in the sort order as the specified Char object.
Equals(Char)	Returns a value that indicates whether this instance is equal to the specified Char object.
GetNumericValue(Char)	Converts the specified numeric Unicode character to a double-precision floating point number.
GetUnicodeCategory(Char)	Categorizes a specified Unicode character into a group identified by one of the UnicodeCategory values.
IsControl(Char)	Indicates whether the specified Unicode character is categorized as a control character.
IsDigit(Char)	Indicates whether the specified Unicode character is categorized as a decimal digit.
IsLetter(Char)	Indicates whether the specified Unicode character is categorized as a Unicode letter.

Methods	Description
IsLetterOrDigit(Char)	Indicates whether the specified Unicode character is categorized as a letter or a decimal digit.
IsLower(Char)	Indicates whether the specified Unicode character is categorized as a lowercase letter.
IsNumber(Char)	Indicates whether the specified Unicode character is categorized as a number.
IsPunctuation(Char)	Indicates whether the specified Unicode character is categorized as a punctuation mark.
IsSeparator(Char)	Indicates whether the specified Unicode character is categorized as a separator character.
IsSymbol(Char)	Indicates whether the specified Unicode character is categorized as a symbol character.
IsUpper(Char)	Indicates whether the specified Unicode character is categorized as an uppercase letter.
IsWhiteSpace(Char)	Indicates whether the specified Unicode character is categorized as white space.
Parse(String)	Converts the value of the specified string to its equivalent Unicode character.
ToLower(Char)	Converts the value of a Unicode character to its lowercase equivalent.
ToString()	Converts the value of this instance to its equivalent string representation.
ToString(IFormatProvider)	Converts the value of this instance to its equivalent string representation using the specified culture-specific format information.
ToUpper(Char)	Converts the value of a Unicode character to its uppercase equivalent.
TryParse(String, Char)	Converts the value of the specified string to its equivalent Unicode character. A return code indicates whether the conversion succeeded or failed.





Enter the String :Programming in C#

Entered String is Programming in C#



Asciics

CharHandling.Ascci

```
using System;

namespace CharHandling
{
    class Ascci
    {
        static void Main()
        {
            for (int i = 1; i <= 255; i++)
            {
                Console.WriteLine("{0} \t", Convert.ToChar(i));
            }
            Console.ReadKey();
        }
    }
}
```

Solution Explorer

Solution 'CharHandling' (1 project)

- CharHandling
 - Properties
 - References
 - Asciics
 - CharCategory.cs
 - CharHandling.cs
 - CharInputs.cs
 - Password.cs
 - Program.cs

Properties

100 %

Output

Item(s) Saved

Ln 17

Col 1

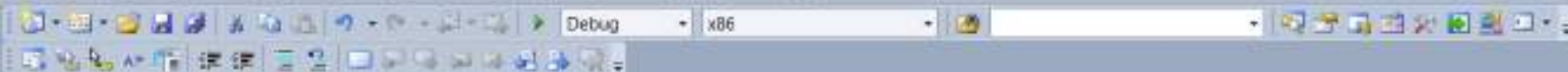
Ch 1

INS

```

+ ) 7 E S a o } ? ? S u A N B i 0
+ ( 6 D R ' n | ? ? ' A D _ 1 0
+ ' 5 C Q _ m { ? ? V 3 A I Y B U
+ & 4 B P ^ l z ? ? o 2 A I U e o
+ % 3 A O l k y ? ? E ± L I U e +
+ 5 2 @ N \ j x ? ? f o _ I U e 0
+ # 1 ? M [ i w ? ? i _ X E U c o
+ " 0 > L z h v ? ? r % E O e 0
+ / = K Y g u ? ? ? - > E x a 0
+ < J X f t ? ? ? - > E O a 0
+ < v - i I W e s ? ? ? u 1 C O a N y
+ > ^ i H V d r ? ? ? > > E O a d _
+ o + + 9 G U c d o ? ? ? c - A O a i y
+ 2 L * B F T b p - ? ? ? > A O a i 0

```

CharCategory.cs*

CharHandling.CharCategory

Main()

```
using System;

namespace CharHandling
{
    class CharCategory
    {
        public static void Main()
        {
            // Define a string with a variety of character categories.
            String s;
            Console.WriteLine("Enter the String:");
            s = Console.ReadLine();
            // Determine the category of each character.
            foreach (var ch in s)
            {
                Console.WriteLine("{0}: {1}", ch, Char.GetUnicodeCategory(ch));
            }
            Console.ReadKey();
        }
    }
}
```

Solution Explorer

Solution 'CharHandling' (1 project)

- CharHandling
 - Properties
 - References
 - Ascii.cs
 - CharCategory.cs
 - CharHandling.cs
 - CharInput.cs
 - Password.cs
 - Program.cs

Properties

Enter the String: C# supports many symbols !@#\$%^&*()_{}|'?:><

'C': UppercaseLetter
'#': OtherPunctuation
' ': SpaceSeparator
's': LowercaseLetter
'u': LowercaseLetter
'p': LowercaseLetter
'p': LowercaseLetter
'o': LowercaseLetter
'n': LowercaseLetter
't': LowercaseLetter
's': LowercaseLetter
' ': SpaceSeparator
'n': LowercaseLetter
'a': LowercaseLetter
'n': LowercaseLetter
'y': LowercaseLetter
' ': SpaceSeparator
's': LowercaseLetter
'y': LowercaseLetter
'n': LowercaseLetter
'b': LowercaseLetter
'o': LowercaseLetter
'l': LowercaseLetter
's': LowercaseLetter
' ': SpaceSeparator
'!': OtherPunctuation
'@': OtherPunctuation
'#': OtherPunctuation
'\$': CurrencySymbol
'%': OtherPunctuation
'^': ModifierSymbol
'&': OtherPunctuation
'*': OtherPunctuation
'(': OpenPunctuation
')': ClosePunctuation
'_': ConnectorPunctuation
'}': ClosePunctuation
'{': OpenPunctuation
'|': OtherPunctuation
' ': OtherPunctuation
'?': OtherPunctuation
'>': MathSymbol
'<': MathSymbol

```
using System;
namespace CharHandling
{
    class Program
    {
        static void Main()
        {
            string str;
            int digit, character, punct, spaces, upper, lower, others;
            digit = character = punct = spaces = others = upper = lower = 0;
            Console.WriteLine("\n Enter the String:");
            str = Console.ReadLine();
            Console.WriteLine("Given Input String is : {0}", str);
            for (int i = 0; i < str.Length; i++)
            {
                if (char.IsLetter(str[i]) == true)
                {
                    character++;
                    if (char.IsUpper(str[i]))
                        upper++;
                    if (char.IsLower(str[i]))
                        lower++;
                }
                else if (char.IsNumber(str[i]) == true)
                    digit++;
                else if (char.IsWhiteSpace(str[i]) == true)
                    spaces++;
                else if (char.IsPunctuation(str[i]) == true)
                    punct++;
                else
                    others++;
            }
            Console.WriteLine("\n Length of the String is :{0}", str.Length);
            Console.WriteLine(" Number of Characters : {0}", character);
            Console.WriteLine(" Number of Upper Characters : {0}", upper);
            Console.WriteLine(" Number of Lower Characters : {0}", lower);
            Console.WriteLine(" Number of Numbers : {0}", digit);
            Console.WriteLine(" Number of Spaces : {0}", spaces);
            Console.WriteLine(" Number of Symbols : {0}", punct);
            Console.WriteLine(" Number of Other Characters : {0}", others);
            Console.ReadKey();
        }
    }
}
```

```
Enter the String:C# (Programming) is very poupopular since 2001.  
Given Input String is : C# (Programming) is very poupopular since 2001.  
Length of the String is :45  
Number of Characters : 31  
Number of Upper Characters : 2  
Number of Lower Characters : 29  
Number of Numbers : 4  
Number of Spaces : 6  
Number of Symbols : 4  
Number of Other Characters : 0
```



CharHandling.cs

CharHandling.CharHandling

Main()

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace CharHandling
{
    class CharHandling
    {
        static void Main()
        {
            char chA;
            char ch='S';
            Console.Write("Enter the Character:");
            chA = Convert.ToChar(Console.Read());

            Console.WriteLine("Difference Between given character and 'C' is {0}",chA.CompareTo('C'));
            Console.WriteLine(chA.Equals('A'));
            Console.WriteLine("Numeric Value of Char is {0}",Char.GetNumericValue(chA));
            Console.WriteLine("Control Character :{0}",Char.IsControl('\t'));
            Console.ReadKey();
        }
    }
}
```

Solution Explorer

Solution 'CharHandling' (1 project)

- CharHandling
 - Properties
 - References
 - Ascii.cs
 - CharCategory.cs
 - CharHandling.cs
 - CharInput.cs
 - Password.cs
 - Program.cs

Properties

Enter the Character:P

Difference Between given character and 'C' is 13

False

Numeric Value of Char is -1

Control Character :True

file:///C:/Users/Dr.PV.Praveen Sundar/Documents/Visual Studio 2010/Projects/CharHandling/CharHandling/bin/Debug/CharHandling... — □ ×

Enter the Character:A
Difference Between given character and 'C' is -2
True
Numeric Value of Char is -1
Control Character :True

```
Enter the Character:1
Difference Between given character and 'C' is -18
False
Numeric Value of Char is 1
Control Character :True
```



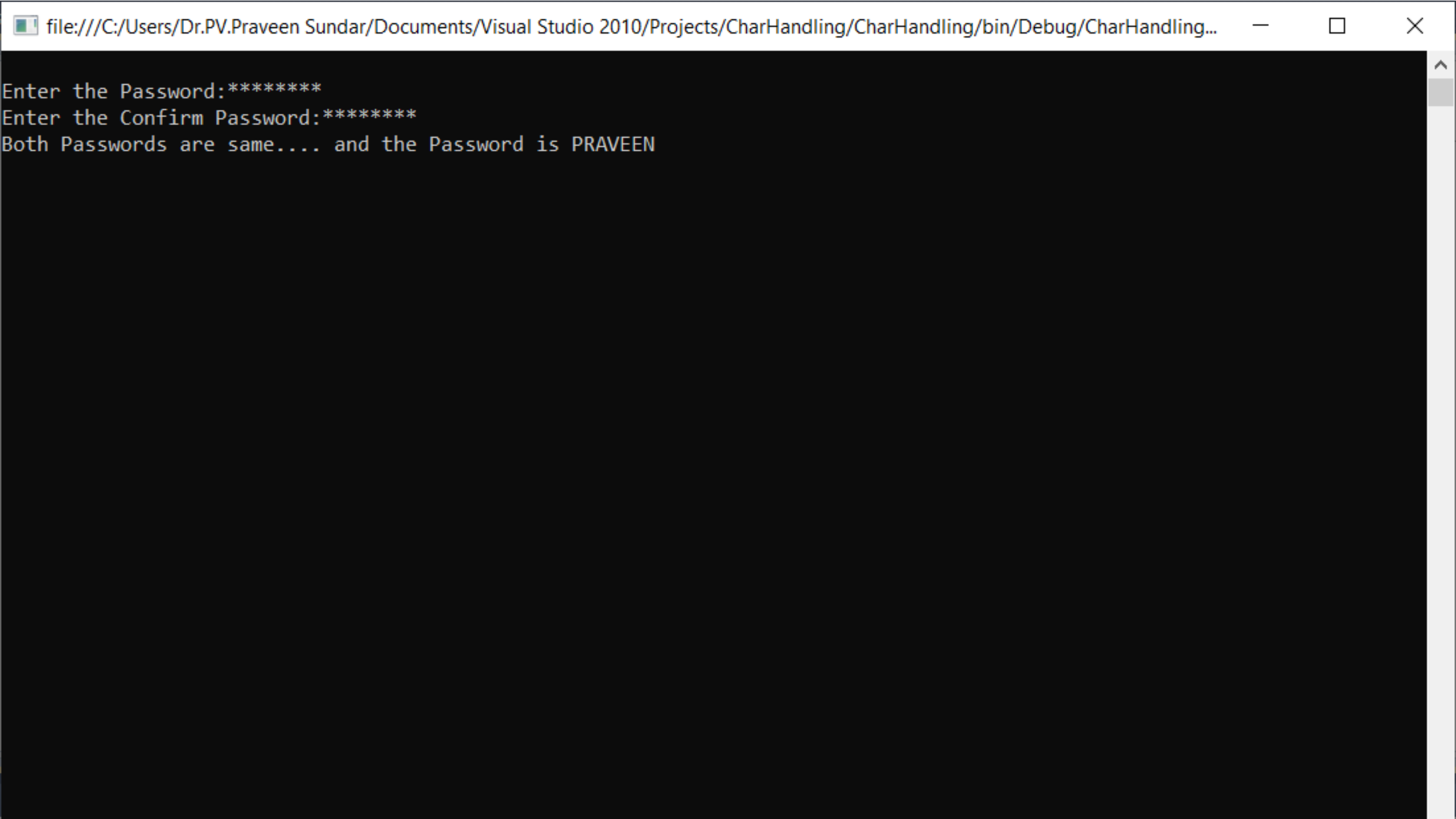
```
using System;

namespace CharHandling
{
    class Password
    {
        static void Main()
        {
            Console.WriteLine("\nEnter the Password:");
            string password = "";
            do
            {
                ConsoleKeyInfo key = Console.ReadKey(true);
                Console.Write("*");

                if (key.KeyChar == 13)
                    break;
                password += key.Key;
            } while (true);
            Console.WriteLine("\nEnter the Confirm Password:");
            string Confirm_password = "";
            do
            {
                ConsoleKeyInfo key = Console.ReadKey(true);
                Console.Write("*");

                if (key.KeyChar == 13)
                    break;
                Confirm_password += key.Key;
            } while (true);

            if (password == Confirm_password)
                Console.WriteLine("\nBoth Passwords are same.... and the Password is {0}", password);
            else
                Console.WriteLine("\nPasswords are not Same....");
            Console.ReadKey();
        }
    }
}
```



Enter the Password:*****

Enter the Confirm Password:*****

Both Passwords are same.... and the Password is PRAVEEN

Enter the Password:*****
Enter the Confirm Password:*****
Passwords are not Same...

Strings

21

- ❑ Strings are collections of characters that are grouped together to form words or sentences.
- ❑ There is no null-terminating character at the end of a C# string; therefore a C# string can contain any number of embedded null characters ('\0').
- ❑ The Length property of a string represents the number of Char objects it contains, not the number of Unicode characters.
- ❑ In C#, the string keyword is an alias for String. Therefore, String and string are equivalent, regardless it is recommended to use the provided alias string as it works even without using System;.
- ❑ The String class provides many methods for safely creating, manipulating, and comparing strings.
- ❑ Following are the ways to declare and initialize the strings.

```
// Declare without initializing.
string message1;

// Initialize to null.
string message2 = null;

// Initialize as an empty string.
// Use the Empty constant instead of the literal "".
string message3 = System.String.Empty;

// Initialize with a regular string literal.
string oldPath = "c:\\Program Files\\Microsoft Visual Studio 8.0";

// Initialize with a verbatim string literal.
string newPath = @"c:\Program Files\Microsoft Visual Studio 9.0";

// Use System.String if you prefer.
System.String greeting = "Hello World!";

// In local variables (i.e. within a method body)
// you can use implicit typing.
var temp = "I'm still a strongly-typed system.string!";

// Use a const string to prevent 'message4' from
// being used to store another string value.
const string message4 = "You can't get rid of me!";

// Use the String constructor only when creating
// a string from a char*, char[], or sbyte*. See
// System.String documentation for details.
char[] letters = { 'A', 'B', 'C' };
string alphabet = new string(letters);
```

Create a string using concatenation

String concatenation operator (+) can be used to combine more than one string to create a single string. The following code snippet creates two strings. The first string adds a text Date and current date value from the DateTime object. The second string adds three strings and some hard coded text to create a larger string.

```
01. string nowDateTime = "Date: " + DateTime.Now.ToString("D");
02. string firstName = "Mahesh";
03. string lastName = "Chand";
04. string age = "33";
05. string authorDetails = firstName + " " + lastName + " is " + age + " years old.";
06.
07. Console.WriteLine(nowDateTime);
08. Console.WriteLine(authorDetails);
```

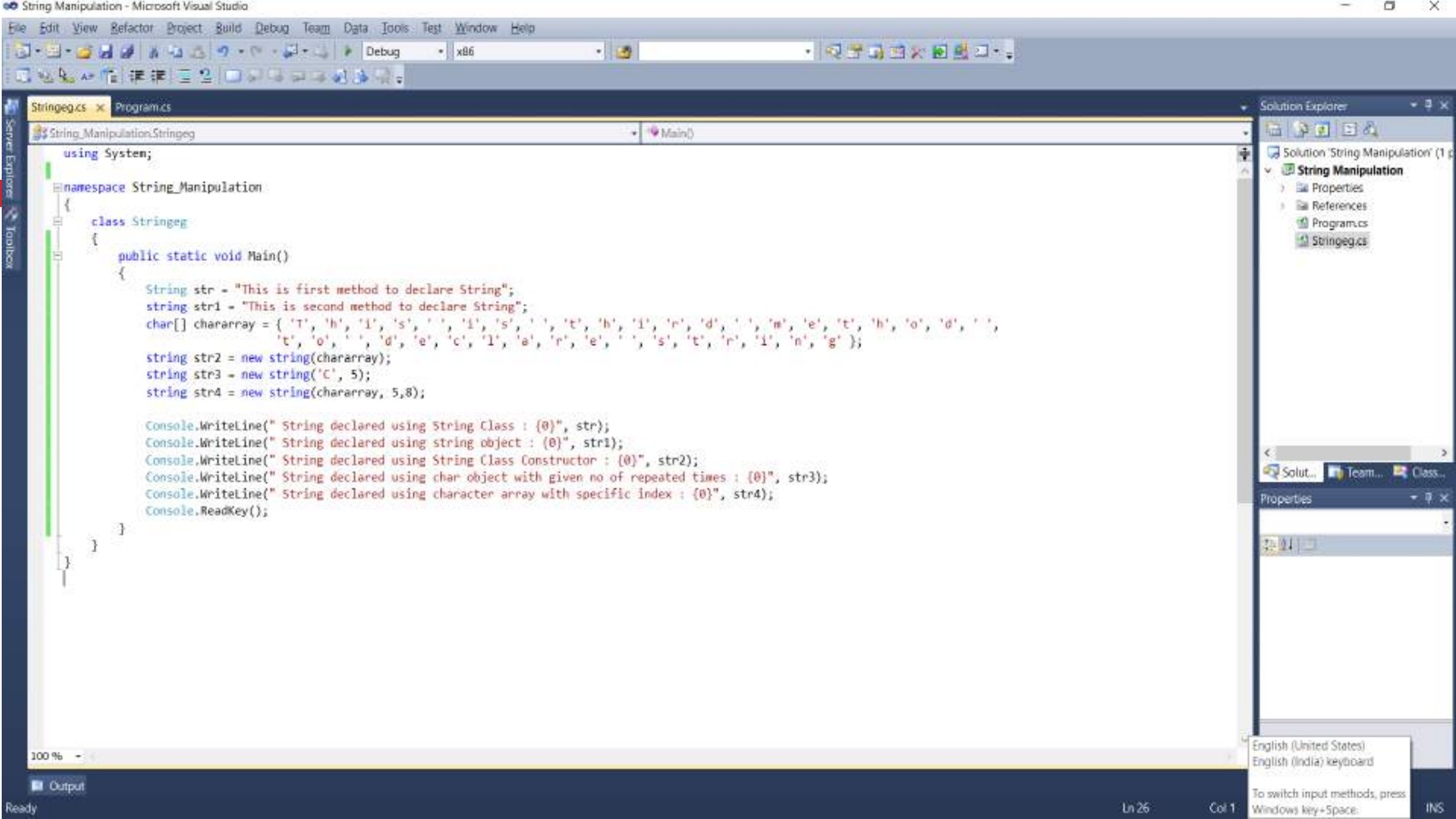
- ❑ In C#, the string is immutable, which means the string object cannot be modified once it is created.
- ❑ If any changes are made to the string object, like adding or modifying an existing value, it will simply discard the old instance in memory and create a new instance to hold the new value.
- ❑ For example, when we create a new string variable “msg” with the text “welcome”, a new instance will create a heap memory to hold this value.
- ❑ Now, if we make any changes to the msg variable, like changing the text from “welcome” to “welcome to C#”, then the old instance on heap memory will be discarded, and another instance will create on heap memory to hold the variable value instead of modifying the old instance in the memory.

- ❑ In C#, if we perform modifications like inserting, concatenating, removing, or replacing a value of the existing string multiple times, every time the new instance will create on heap memory to hold the new value, so automatically the performance of the application will be affected.
- ❑ In C#, Strings can be declared using following methods
 - ❑ Using Character array.
 - ❑ Using String Class
 - ❑ Using string object.
 - ❑ Using string object constructor.
 - ❑ Converting value to String type.
- ❑ In C#, String class can have only one property called as length. The Length property gets the number of characters in the current String object.

Method Name	Description
Clone()	It is used to return a reference to this instance of String.
Compare(String, String)	It is used to compares two specified String objects. It returns an integer that indicates their relative position in the sort order.
Concat(String, String)	It is used to concatenate two specified instances of String.
Contains(String)	It is used to return a value indicating whether a specified substring occurs within this string.
Copy(String)	It is used to create a new instance of String with the same value as a specified String.
CopyTo(Int32, Char[], Int32, Int32)	It is used to copy a specified number of characters from a specified position in this instance to a specified position in an array of Unicode characters.
EndsWith(String)	It is used to check that the end of this string instance matches the specified string.
Equals(String, String)	It is used to determine that two specified String objects have the same value.
Format(String, Object)	It is used to replace one or more format items in a specified string with the string representation of a specified object.
GetType()	It is used to get the Type of the current instance.
IndexOf(String)	It is used to report the zero-based index of the first occurrence of the specified string in this instance.
Insert(Int32, String)	It is used to return a new string in which a specified string is inserted at a specified index position.

Method Name	Description
Join(String, String[])	It is used to concatenate all the elements of a string array, using the specified separator between each element.
LastIndexOf(Char)	It is used to report the zero-based index position of the last occurrence of a specified character within String.
LastIndexOfAny(Char[])	It is used to report the zero-based index position of the last occurrence in this instance of one or more characters specified in a Unicode array.
PadLeft(Int32)	It is used to return a new string that right-aligns the characters in this instance by padding them with spaces on the left.
PadRight(Int32)	It is used to return a new string that left-aligns the characters in this string by padding them with spaces on the right.
Remove(Int32)	It is used to return a new string in which all the characters in the current instance, beginning at a specified position and continuing through the last position, have been deleted.
Replace(String, String)	It is used to return a new string in which all occurrences of a specified string in the current instance are replaced with another specified string.
Split(Char[])	It is used to split a string into substrings that are based on the characters in an array.
StartsWith(String)	It is used to check whether the beginning of this string instance matches the specified string.
Substring(Int32)	It is used to retrieve a substring from this instance. The substring starts at a specified character position and continues to the end of the string.

Method Name	Description
ToCharArray()	It is used to copy the characters in this instance to a Unicode character array.
ToLower()	It is used to convert String into lowercase.
ToString()	It is used to return instance of String.
ToUpper()	It is used to convert String into uppercase.
Trim()	It is used to remove all leading and trailing white-space characters from the current String object.
TrimEnd(Char[])	It Is used to remove all trailing occurrences of a set of characters specified in an array from the current String object.
TrimStart(Char[])	It is used to remove all leading occurrences of a set of characters specified in an array from the current String object.



String Manipulation - Microsoft Visual Studio

File Edit View Refactor Project Build Debug Team Data Tools Test Window Help

Debug x86

Stringeg.cs Program.cs

String_Manipulation.Stringeg Main()

```
using System;

namespace String_Manipulation
{
    class Stringeg
    {
        public static void Main()
        {
            String str = "This is first method to declare String";
            string str1 = "This is second method to declare String";
            char[] chararray = { 't', 'h', 'i', 's', ' ', 'i', 's', ' ', 't', 'h', 'i', 's', ' ', 'm', 'e', 't', 'h', 'o', 'd', ' ', 't', 'o', ' ', 'd', 'e', 'c', 'l', 'a', 'r', 'e', ' ', 's', 't', 'r', 'i', 'n', 'g' };
            string str2 = new string(chararray);
            string str3 = new string('C', 5);
            string str4 = new string(chararray, 5, 8);

            Console.WriteLine(" String declared using String Class : {0}", str);
            Console.WriteLine(" String declared using string object : {0}", str1);
            Console.WriteLine(" String declared using String Class Constructor : {0}", str2);
            Console.WriteLine(" String declared using char object with given no of repeated times : {0}", str3);
            Console.WriteLine(" String declared using character array with specific index : {0}", str4);
            Console.ReadKey();
        }
    }
}
```

100 %

Output

Ready

Ln 26 Col 1 INS

Solution Explorer

Solution 'String Manipulation' (1 d)

String Manipulation

- Properties
- References
- Program.cs
- Stringeg.cs

Properties

English (United States)
English (India) keyboard

To switch input methods, press
Windows key+Space.

```
String declared using String Class : This is first method to declare String  
String declared using string object : This is second method to declare String  
String declared using String Class Constructor : This is third method to declare string  
String declared using char object with given no of repeated times : CCCCC  
String declared using character array with specific index : is third
```

Obtaining the Length of a C# String

The length of a C# string may be obtained by accessing the *Length* property of the string object:

```
String myString = "Hello World";

System.Console.WriteLine ("myString length = " + myString.Length);
```

When executed, the output will read "myString length = 11".

Treating Strings as Arrays

It is possible to access individual characters in a string by treating the string as an array

By specifying the index into the array of the character, individual characters may be accessed. It is important to note that strings are immutable (in other words the value of a string cannot be modified unless an entirely new string literal is assigned to the object). This means that while it is possible to read the value of a character in a string it is not possible to change the value:

```
string myString = "Hello World";

System.Console.WriteLine(myString[1]); //Displays second character (e)

myString[0] = 'h';    // Illegal - string cannot be modified.
```

```
//-----  
// Checking Two Strings are Equal  
//-----  
  
if (str1 == str) //- Method-1 using = Operator  
{  
    Console.WriteLine("Both are Equal");  
}  
else  
{  
    Console.WriteLine("Both are Different");  
}  
////-----  
if (str.Equals(str1) == true)// - Method-2 using Equals()  
{  
    Console.WriteLine("Both are Equal");  
}  
else  
{  
    Console.WriteLine("Both are Different");  
}  
////-----  
if (String.Compare(str, str1) == 0)// - Method-3 using Equals()  
{  
    Console.WriteLine("Both are Equal");  
}  
else if (String.Compare(str, str1) == 1)  
{  
    Console.WriteLine("Str is greater than Str_copy");  
}  
else  
{  
    Console.WriteLine("Str is lesser than Str_copy");  
}
```

```
////-----  
// String Concatenation  
  
Console.Write("Enter the String to Concat :");  
app_str = Console.ReadLine();  
  
str = str + app_str; // Method -1 using + Operator  
Console.WriteLine("After Concatenation of New string to Str is {0} ", str);  
  
Console.Write("Enter the String to Concat :");  
app_str = Console.ReadLine();  
  
app_str += str; // This will append new string + Old Value .  
  
Console.WriteLine("After Concatenation of New string to Str is {0} ", app_str);  
  
Console.Write("Enter the String to Concat :");  
app_str = Console.ReadLine();  
app_str = String.Concat(str, app_str);  
Console.WriteLine("After Concatenation of New string to Str is {0} ", app_str);  
////-----
```



```
////-----  
//- Finding Substring  
  
String substr;  
int S_index, e_index;  
Console.Write("Enter the Location of Substring :");  
S_index = Convert.ToInt16(Console.ReadLine());  
substr = str.Substring(S_index); // After 7th Character Entire String is Copied to substr  
Console.WriteLine("Substring Str is {0}:", substr);  
  
Console.Write("Enter the end Location of Substring :");  
e_index = Convert.ToInt16(Console.ReadLine());  
substr = str.Substring(S_index, e_index); // After 7th Character it Copies only value of e_index Characters  
Console.WriteLine("Substring Str is {0}:", substr);
```

```
class Program
{
    static void display(string str)
    {
        Console.WriteLine(str);
    }
    static void Main()
    {

        string str = "C# Programming";
        Console.WriteLine(str.PadRight(25, '*'));

        string str1= str.Insert(2, "is an OOP");
        Console.WriteLine(str1);
        Console.ReadKey();
    }
}
```

```
//-----  
// To Check whether the given character is Occured or Not  
  
char c;  
int loc;  
Console.Write("Enter the Character to Search :");  
c = Convert.ToChar(Console.Read());  
  
loc = str.IndexOf(c);  
  
if (loc > 0)  
{  
    Console.WriteLine("Given Character is occurred at {0} Location :", loc);  
}  
else  
{  
    Console.WriteLine("Given Character Not Found ");  
}  
  
Console.Write("Enter the Character to Search :");  
string str_search = Console.ReadLine();  
  
loc = str.IndexOf(str_search);  
  
if (loc > 0)  
{  
    Console.WriteLine("Given Character is occurred at {0} Location :", loc);  
}  
else  
{  
    Console.WriteLine("Given Character Not Found ");  
}  
}
```

Changing String Case

The case of the characters in a string may be changed using the `ToUpper` and `ToLower` methods. Both of these methods return a modified string rather than changing the actual string. For example:

```
string myString = "Hello World";  
string newString;  
  
newString = myString.ToUpper();  
  
System.Console.WriteLine (newString); // Displays HELLO WORLD  
  
newString = myString.ToLower();  
  
System.Console.WriteLine (newString); // Displays hello world
```

C# String Replacement

Parts of a string may be replaced using the *Replace()* method. This method takes the part of the string to be replaced and the string with which it is to be replaced as arguments and returns a new string reflecting the change. The *Replace()* method will replace all instances of the string:

```
string myString = "Hello World";  
string newString;  
  
newString = myString.Replace("Hello", "Goodbye");  
  
System.Console.WriteLine (newString);
```

Splitting a C# String into Multiple Parts

A string may be separated into multiple parts using the *Split()* method. *Split()* takes as an argument the character to use as the delimiter to identify the points at which the string is to be split. Returned from the method call is an array containing the individual parts of the string. For example, the following code splits a string up using the comma character as the delimiter. The results are placed in an array called *myColors* and a *foreach* loop then reads each item from the array and displays it:

```
string myString = "Red, Green, Blue, Yellow, Pink, Purple";

string[] myColors = myString.Split(',');

foreach (string color in myColors)
{
    System.Console.WriteLine (color);
}
```

The resulting output will read:

```
Red
Green
Blue
Yellow
Pink
Purple
```

Trimming and Padding C# Strings

Unwanted leading and trailing spaces can be removed from a string using the *Trim()* method. When called, this method returns a modified version of the string with both leading and trailing spaces removed:

```
string myString = "    hello    ";

System.Console.WriteLine ("[" + myString + "]");
System.Console.WriteLine ("[" + myString.Trim() + "]");
```

The above code will result in the following output:

```
[    hello    ]
[hello]
```

To remove just the leading or trailing spaces use the *TrimStart()* or *TrimEnd()* method respectively.

In inverse of the *Trim()* method are the *PadLeft()* and *PadRight()* methods. These methods allow leading or trailing characters to be added to a string. The methods take as arguments the total number of characters to which the string is to be padded and the padding character:

```
string myString = "hello";
string newString;

newString = myString.PadLeft(10, ' ');

newString = newString.PadRight(20, '*');

System.Console.WriteLine ("[" + newString + "]"); // Outputs [    hello*****]
```

String.Format()

40

- ❑ The string class also provides the String.Format() method.
- ❑ The primary purpose of the C# String.Format() method is to provide a mechanism for inserting string, numerical or boolean values into a string.
- ❑ The general syntax of the String.Format() method is as follows:
`String.Format("format string", arg1, arg2,);`
- ❑ The format string is the string into which the values will be placed. Within this string are place holders which indicate the location of each value within the string.
- ❑ Place holders take the form of braces surrounding a number indicating the corresponding argument to be substituted for the place holder. Following on from the format string is a comma separated list of arguments. There must be an argument for each of the place holders.

A Simple C# String Format Example

The following code fragment demonstrates a very simple use of the `String.Format()` method:

```
string newString;  
  
newString = String.Format("There are {0} cats in my {1} and no {2}", 2, "house", "dogs");  
  
System.Console.WriteLine (newString);
```

When run, the above code will result in the following output:

```
There are 2 cats in my house and no dogs
```

Let's quickly review the `String.Format()` method call. The *format string* contains 3 place holders indicated by `{0}`, `{1}` and `{2}`. Following the format string are the arguments to be used in each place holder. For example, `{0}` is replaced by the first argument (the number 2), the `{1}` by the second argument (the string "house") and so on.

Using Format Controls

So far we have only substituted arguments for place holders, but we have not made any changes to the format of the arguments before they are displayed. This essentially instructs the `Format()` method to use the default formatting for each argument type when displaying the string. Perhaps the most powerful aspect of the `Format()` method is the ability to use *format controls* within the place holders to control the format of the output.

Format controls appear inside the braces (`{}`) of the place holders. The format of a place holder with a format control is as follows:

`{n:controlx}`

Where *n* is the place holder number and *control* is the special format control sequence to be applied to the argument. The *x* is an optional number which further formats the output for certain controls.

As Simple Format Control Example

The following example uses the *X* format control to display a number in Hexadecimal format:

```
newString = String.Format("The number {0} in Hexadecimal is {0:X}", 432);  
  
System.Console.WriteLine (newString);
```

The above example displays argument 0 (the number 432) in two formats. The first is the default decimal format and the second uses the *X* format control to display the argument as a hexadecimal number.

The C# String.Format() Format Controls

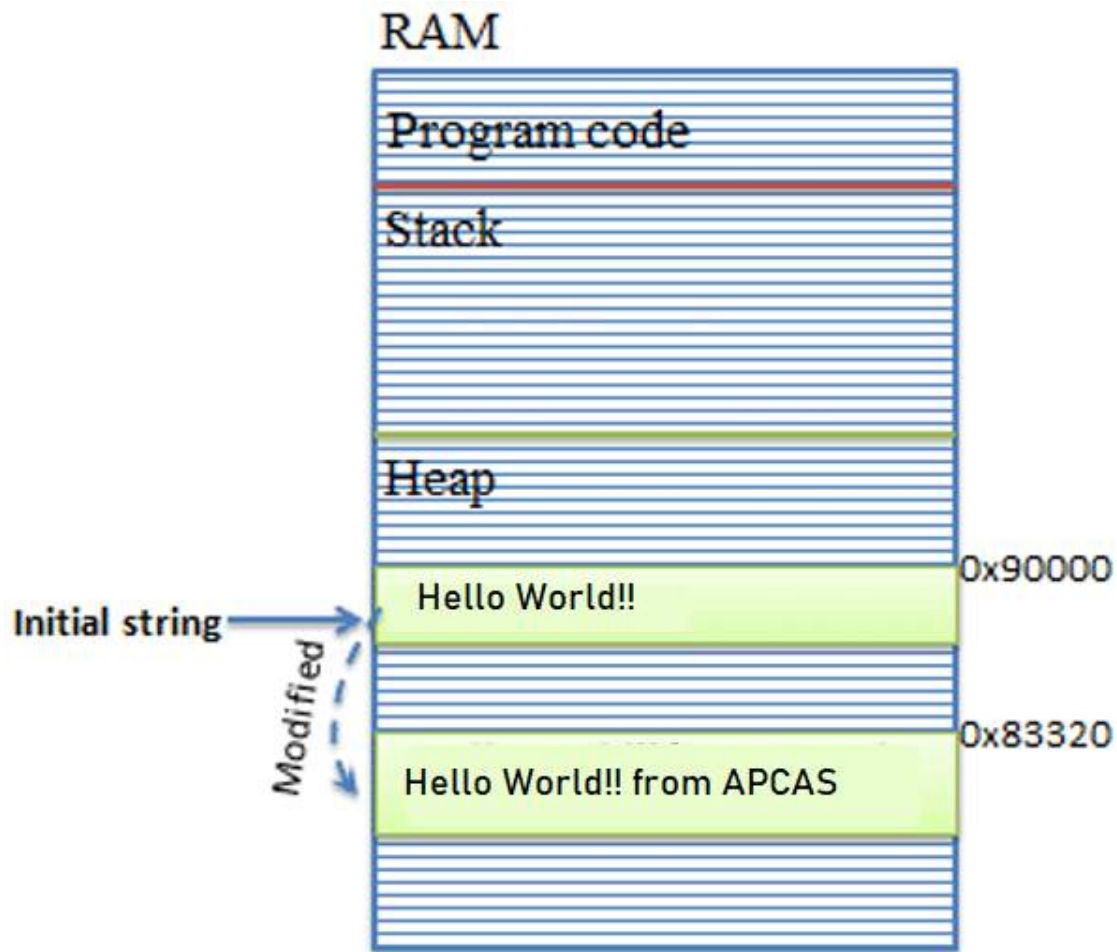
The following table lists format controls supported by the C# String.Format() method together with examples of each control:

Control	Type	Description	Example
C	Currency	Displays number prefixed with the currency symbol appropriate to the current locale	{0:C} of 432.00 outputs \$432.00
D	Decimal	Displays number in decimal form with optional padding	{0:D4} of 432 outputs 00432
E	Exponential	Displays number in scientific form with optional value for fractional part	{0:E5} of 432.32 outputs 4.32320E+002
E	Fixed	Displays the number including the specified number of decimal digits	{0:F3} of 432.324343 outputs 432.324
N	Number	Converts a number to a human friendly format by inserting commas and rounding to the nearest 100 th	{0:N} of 123432.324343 outputs 123,432.32
X	Hexadecimal	Converts a number to hexadecimal	{0:X} of 432 outputs 1B0
0:0...	Zero Padding	Adds zeros to pad argument	{0:0000.00} of 43.1 outputs 0043.10
0:0#...	Space Padding	Adds spaces to pad argument	{0:####.##} of 43.1 outputs 43.1
%	Percentage	Multiplies the argument by 100 and appends a percentage sign	{0:00.00%} of .432 outputs 43.20%

StringBuilder

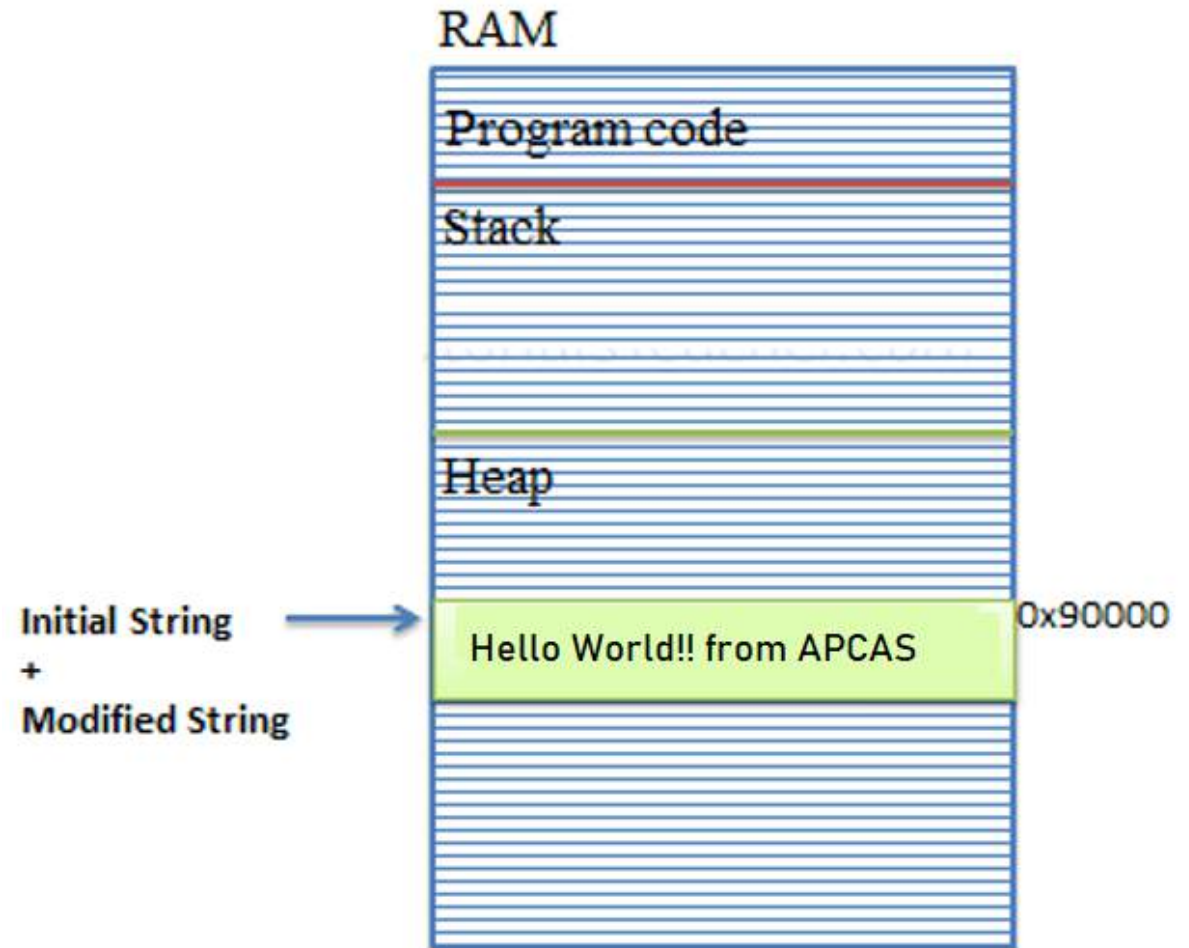
44

- ❑ In C#, StringBuilder is a class that is useful to represent a mutable string of characters, and it is an object of the **System.Text** namespace.
- ❑ Like string in C#, we can use a StringBuilder to create variables to hold any text, a sequential collection of characters based on our requirements.
- ❑ In C#, both String and StringBuilder will represent a sequence of characters and perform the same kind of operations, but the only difference is strings are immutable, and StringBuilder is mutable.
- ❑ Generally, in C# the string object cannot be modified once it is created. If any changes are made to the string object, like add or modify an existing value, it will simply discard the old instance in memory and create a new instance to hold the new value. If we are doing repeated modifications on the string object, it will affect the application's performance.



- For example, a new string, "Hello World!" will occupy a memory space on the heap. Now, by changing the initial string "Hello World!" to "Hello World! from APCAS" will create a new string object on the memory heap instead of modifying an original string at the same memory address.
- This behavior would delay the performance if the original string changed multiple times by replacing, appending, removing, or inserting new strings in the original string.

- To solve this problem, C# introduced an alternative called **StringBuilder**, which is a mutable string class.
- Mutability means once an instance of the class is created, then the same instance will be used to perform any operations like inserting, appending, removing, or replacing the characters instead of creating a new instance for every time.
- In C#, the **StringBuilder** is a dynamic object which will expand a memory dynamically to accommodate the modifications of string instead of creating a new instance in the memory.



How StringBuilder works

47

- The `StringBuilder.Length` property indicates the number of characters the StringBuilder object currently contains.
- Once we add characters to the StringBuilder object, its length increases until it equals the size of the `StringBuilder.Capacity` property, which defines the number of characters that the object can contain.
- If the number of added characters causes the length of the StringBuilder object to exceed its current capacity, new memory is allocated, the value of the Capacity property is doubled, new characters are added to the StringBuilder object, and its Length property is adjusted.
- Additional memory for the StringBuilder object is allocated dynamically until it reaches the value defined by the `StringBuilder.MaxCapacity` property.
- When the maximum capacity is reached, no further memory can be allocated for the StringBuilder object, and trying to add characters or expand it beyond its maximum capacity throws either an `ArgumentOutOfRangeException` or an `OutOfMemoryException` exception.

- ❑ The following example illustrates how a `StringBuilder` object allocates new memory and increases its capacity dynamically as the string assigned to the object expands.
- ❑ The code creates a `StringBuilder` object by calling its default (parameterless) constructor.
- ❑ The default capacity of this object is 16 characters, and its maximum capacity is more than 2 billion characters.
- ❑ Appending the string "This is a sentence." results in a new memory allocation because the string length (19 characters) exceeds the default capacity of the `StringBuilder` object. The capacity of the object doubles to 32 characters, the new string is added, and the length of the object now equals 19 characters. The code then appends the string "This is an additional sentence." to the value of the `StringBuilder` object 11 times. Whenever the append operation causes the length of the `StringBuilder` object to exceed its capacity, its existing capacity is doubled and the Append operation succeeds.

```

using System;
using System.Reflection;
using System.Text;

public class Example
{
    public static void Main()
    {
        StringBuilder sb = new StringBuilder();
        ShowSBInfo(sb);
        sb.Append("This is a sentence.");
        ShowSBInfo(sb);
        for (int ctr = 0; ctr <= 10; ctr++) {
            sb.Append("This is an additional sentence.");
            ShowSBInfo(sb);
        }
    }

    private static void ShowSBInfo(StringBuilder sb)
    {
        foreach (var prop in sb.GetType().GetProperties()) {
            if (prop.GetIndexParameters().Length == 0)
                Console.WriteLine("{0}: {1:NB} ", prop.Name, prop.GetValue(sb));
        }
        Console.WriteLine();
    }
}

// The example displays the following output:
// Capacity: 16 MaxCapacity: 2,147,483,647 Length: 0
// Capacity: 32 MaxCapacity: 2,147,483,647 Length: 19
// Capacity: 64 MaxCapacity: 2,147,483,647 Length: 50
// Capacity: 128 MaxCapacity: 2,147,483,647 Length: 81
// Capacity: 128 MaxCapacity: 2,147,483,647 Length: 112
// Capacity: 256 MaxCapacity: 2,147,483,647 Length: 143
// Capacity: 256 MaxCapacity: 2,147,483,647 Length: 174
// Capacity: 256 MaxCapacity: 2,147,483,647 Length: 205
// Capacity: 256 MaxCapacity: 2,147,483,647 Length: 236
// Capacity: 512 MaxCapacity: 2,147,483,647 Length: 267
// Capacity: 512 MaxCapacity: 2,147,483,647 Length: 298
// Capacity: 512 MaxCapacity: 2,147,483,647 Length: 329
// Capacity: 512 MaxCapacity: 2,147,483,647 Length: 360

```


Memory allocation

50

- ❑ The default capacity of a `StringBuilder` object is 16 characters, and its default maximum capacity is `Int32.MaxValue`. These default values are used if you call the `StringBuilder()` and `StringBuilder(String)` constructors.
- ❑ You can explicitly define the initial capacity of a `StringBuilder` object in the following ways:
 - ❑ By calling any of the `StringBuilder` constructors that includes a capacity parameter when you create the object.
 - ❑ By explicitly assigning a new value to the `StringBuilder.Capacity` property to expand an existing `StringBuilder` object. Note that the property throws an exception if the new capacity is less than the existing capacity or greater than the `StringBuilder` object's maximum capacity.
 - ❑ By calling the `StringBuilder.EnsureCapacity` method with the new capacity. The new capacity must not be greater than the `StringBuilder` object's maximum capacity. However, unlike an assignment to the `Capacity` property, `EnsureCapacity` does not throw an exception if the desired new capacity is less than the existing capacity; in this case, the method call has no effect.

- If the length of the string assigned to the `StringBuilder` object in the constructor call exceeds either the default capacity or the specified capacity, the `Capacity` property is set to the length of the string specified with the `value` parameter.
- we can explicitly define the maximum capacity of a `StringBuilder` object by calling the `StringBuilder(Int32, Int32)` constructor. we can't change the maximum capacity by assigning a new value to the `MaxCapacity` property, because it is read-only.

StringBuilder Constructors

52

Constructor	Description
<code>StringBuilder()</code>	Initializes a new instance of the <code>StringBuilder</code> class.
<code>StringBuilder(Int32)</code>	Initializes a new instance of the <code>StringBuilder</code> class using the specified capacity.
<code>StringBuilder(String)</code>	Initializes a new instance of the <code>StringBuilder</code> class using the specified string.
<code>StringBuilder(Int32, Int32)</code>	Initializes a new instance of the <code>StringBuilder</code> class that starts with a specified capacity and can grow to a specified maximum.
<code>StringBuilder(String, Int32)</code>	Initializes a new instance of the <code>StringBuilder</code> class using the specified string and capacity.
<code>StringBuilder(String, Int32, Int32, Int32)</code>	Initializes a new instance of the <code>StringBuilder</code> class from the specified substring and capacity.

Creating a StringBuilder Object

53

- We can create an object of the StringBuilder class using the new keyword and passing an initial string. The following example demonstrates creating StringBuilder objects.
- Optionally, you can also specify the maximum capacity of the StringBuilder object using overloaded constructors.

```
StringBuilder sb = new StringBuilder(); //string will be appended later
```

```
StringBuilder sb = new StringBuilder("Hello World!");
```

```
StringBuilder sb = new StringBuilder(50); //string will be appended later
```

```
StringBuilder sb = new StringBuilder("Hello World!", 50);
```

- Above, C# allocates a maximum of 50 spaces sequentially on the memory heap. This capacity will automatically be doubled once it reaches the specified capacity.
- We can also use the capacity or length property to set or retrieve the StringBuilder object's capacity.

- Using for loop, it is possible to iterate and get or set a character at the specified index.

Example: StringBuilder Iteration

```
StringBuilder sb = new StringBuilder("Hello World!");  
for(int i = 0; i < sb.Length; i++)  
    Console.Write(sb[i]); // output: Hello World!
```

Retrieve String from StringBuilder

- The StringBuilder is not the string. Use the ToString() method to retrieve a string from the StringBuilder object.

Example: Retrieve String from StringBuilder

```
StringBuilder sb = new StringBuilder("Hello World!");  
String greet = sb.ToString(); //returns "Hello World!"
```

StringBuilder Properties

55

Property	Description
StringBuilder.Capacity	Gets or sets the maximum number of characters that can be contained in the memory allocated by the current instance.
StringBuilder.Chars[Int32]	Gets or sets the character at the specified character position in this instance.
StringBuilder.Length	Gets or sets the length of the current StringBuilder object.
StringBuilder.MaxCapacity	Gets the maximum capacity of this instance.

StringBuilder Methods

56

Method	Description
StringBuilder.Append	This method will append the given string value to the end of the current StringBuilder.
StringBuilder.AppendFormat	It will replace a format specifier passed in a string with formatted text.
StringBuilder.Clear	Removes all characters from the current StringBuilder instance.
StringBuilder.CopyTo	Copies the characters from a specified segment of this instance to a destination Char span.
StringBuilder.EnsureCapacity(Int32)	Ensures that the capacity of this instance of StringBuilder is at least the specified value.
StringBuilder.Equals	Returns a value indicating whether the characters in this instance are equal to the characters in a specified read-only character span.
StringBuilder.Insert	It inserts a string at the specified index of the current StringBuilder.
StringBuilder.Remove	It removes a specified number of characters from the current StringBuilder.
StringBuilder.Replace	It replaces a specified character at a specified index.
StringBuilder.ToString	Converts the value of a StringBuilder to a String.

Add/Append String to StringBuilder

Use the `Append()` method to append a string at the end of the current `StringBuilder` object. If a `StringBuilder` does not contain any string yet, it will add it. The `AppendLine()` method append a string with the newline character at the end.

Example: Adding or Appending Strings in StringBuilder

```
StringBuilder sb = new StringBuilder();  
sb.Append("Hello ");  
sb.AppendLine("World!");  
sb.AppendLine("Hello C#");  
Console.WriteLine(sb);
```


Append Formated String to StringBuilder

Use the `AppendFormat()` method to format an input string into the specified format and append it.

Example: `AppendFormat()`

```
StringBuilder sbAmout = new StringBuilder("Your total amount is ");  
sbAmout.AppendFormat("{0:C} ", 25);  
  
Console.WriteLine(sbAmout); //output: Your total amount is $ 25.00
```

```
using System;
using System.Text;

public class Class1
{
    public static void Main()
    {
        StringBuilder sb = new StringBuilder("This is a string.");
        Console.WriteLine("{0} ({1} characters)", sb.ToString(), sb.Length);

        sb.Clear();
        Console.WriteLine("{0} ({1} characters)", sb.ToString(), sb.Length);

        sb.Append("This is a second string.");
        Console.WriteLine("{0} ({1} characters)", sb.ToString(), sb.Length);
    }
}

// The example displays the following output:
//      This is a string. (17 characters)
//      (0 characters)
//      This is a second string. (24 characters)
```

```

using System;
using System.Text;

class Sample
{
    protected static char[] dest = new char[6];
    public static void Main()
    {
        StringBuilder src = new StringBuilder("abcdefghijklmnopqrstuvwxyz!");
        dest[1] = ' ';
        dest[2] = ' ';

        // Copy the source to the destination in 9 pieces, 3 characters per piece.

        Console.WriteLine("\nPiece) Data:");
        for(int ix = 0; ix < 9; ix++)
        {
            dest[0] = ix.ToString()[0];
            src.CopyTo(ix * 3, dest, 3, 3);
            Console.Write("    ");
            Console.WriteLine(dest);
        }
    }
}

```

This example produces the following results:

```

Piece) Data:
0) abc
1) def
2) ghi
3) jkl
4) mno
5) pqr
6) stu
7) vwx
8) yz!
*/

```

```
public static void Main()
{
    StringBuilder sb1 = new StringBuilder("abc");
    StringBuilder sb2 = new StringBuilder("abc", 16);

    Console.WriteLine();
    Console.WriteLine("a1) sb1.Length = {0}, sb1.Capacity = {1}", sb1.Length, sb1.Capacity);
    Console.WriteLine("a2) sb2.Length = {0}, sb2.Capacity = {1}", sb2.Length, sb2.Capacity);
    Console.WriteLine("a3) sb1.ToString() = \"{0}\"", sb2.ToString() = \"{1}\"",
        sb1.ToString(), sb2.ToString());
    Console.WriteLine("a4) sb1 equals sb2: {0}", sb1.Equals(sb2));

    Console.WriteLine();
    Console.WriteLine("Ensure sb1 has a capacity of at least 50 characters.");
    sb1.EnsureCapacity(50);

    Console.WriteLine();
    Console.WriteLine("b1) sb1.Length = {0}, sb1.Capacity = {1}", sb1.Length, sb1.Capacity);
    Console.WriteLine("b2) sb2.Length = {0}, sb2.Capacity = {1}", sb2.Length, sb2.Capacity);
    Console.WriteLine("b3) sb1.ToString() = \"{0}\"", sb2.ToString() = \"{1}\"",
        sb1.ToString(), sb2.ToString());
    Console.WriteLine("b4) sb1 equals sb2: {0}", sb1.Equals(sb2));

    Console.WriteLine();
    Console.WriteLine("Set the length of sb1 to zero.");
    Console.WriteLine("Set the capacity of sb2 to 51 characters.");
    sb1.Length = 0;
    sb2.Capacity = 51;

    Console.WriteLine();
    Console.WriteLine("c1) sb1.Length = {0}, sb1.Capacity = {1}", sb1.Length, sb1.Capacity);
    Console.WriteLine("c2) sb2.Length = {0}, sb2.Capacity = {1}", sb2.Length, sb2.Capacity);
    Console.WriteLine("c3) sb1.ToString() = \"{0}\"", sb2.ToString() = \"{1}\"",
        sb1.ToString(), sb2.ToString());
    Console.WriteLine("c4) sb1 equals sb2: {0}", sb1.Equals(sb2));
}
```

```
StringBuilder sb_str;  
Console.Write("Enter the String :");  
sb_str = new StringBuilder(Console.ReadLine());  
  
StringBuilder sb_Substr;  
Console.Write("Enter the Sub String:");  
sb_Substr = new StringBuilder(Console.ReadLine());  
  
Console.WriteLine(" Both String Objects are Equal: {0}", sb_str.Equals(sb_Substr));  
|
```

Insert String into StringBuilder

Use the `Insert()` method inserts a string at the specified index in the `StringBuilder` object.

Example: `Insert()`

```
StringBuilder sb = new StringBuilder("Hello World!");  
sb.Insert(5, " C#");
```

```
Console.WriteLine(sb); //output: Hello C# World!
```

Remove String in StringBuilder

Use the `Remove()` method to remove a string from the specified index and up to the specified length.

Example: `Remove()`

```
StringBuilder sb = new StringBuilder("Hello World!",50);  
sb.Remove(6, 7);  
  
Console.WriteLine(sb); //output: Hello
```

Replace String in StringBuilder

Use the `Replace()` method to replace all the specified string occurrences with the specified replacement string.

Example: Replace()

```
StringBuilder sb = new StringBuilder("Hello World!");  
sb.Replace("World", "C#");  
  
Console.WriteLine(sb); //output: Hello C#!
```


Thank you



PRINCIPLES OF OBJECT ORIENTED PROGRAMMING

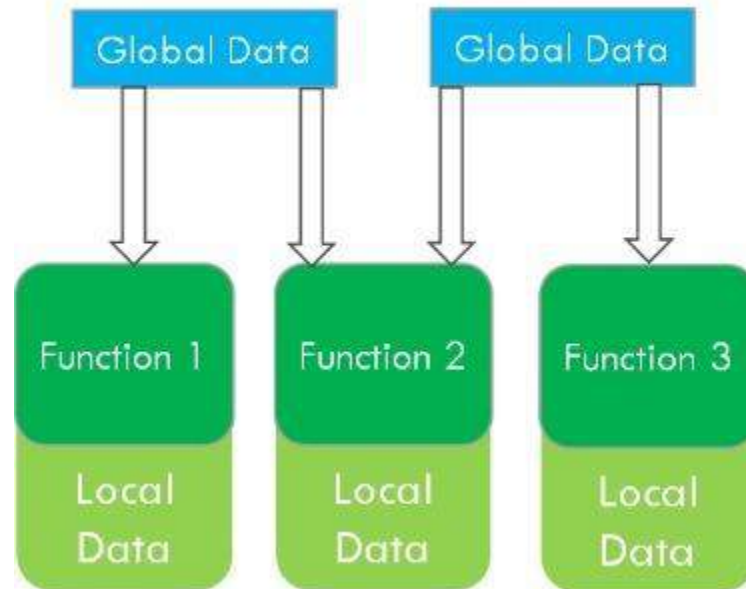
Dr P.V. Praveen Sundar
Assistant Professor,
Department of Computer Science
Adhiparasakthi College of Arts & Science,
Kalavai.

Object Oriented Programming (OOPS)

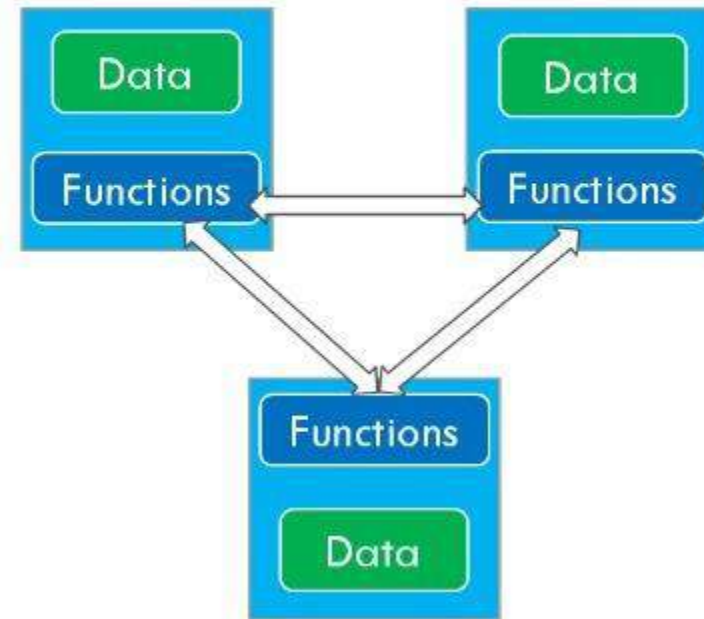
2

- ❑ Object-Oriented Programming or OOPs refers to languages that uses objects in programming.
- ❑ Object Oriented programming (OOP) is a programming paradigm that relies on the concept of **classes** and **objects**.
- ❑ OOPs ties data more closely to the functions that operate on it, and protects it from accidental modification from outside functions.
- ❑ OOP allows decomposition of a problem into a number of entities called objects and then builds data and functions around these objects.
- ❑ The data of an object can be accessed only by the functions associated with that object. However, functions of one object can access the function of other objects.

Procedural Oriented Programming



Object Oriented Programming



Features of OOPs

4

- Some of the striking features of programming are:
 - ▣ Emphasis is on data rather than procedure.
 - ▣ Programs are divided into what are known as objects.
 - ▣ Data structures are designed such that they characterize the objects.
 - ▣ Functions that operate on the data of an object are tied together in the data structure.
 - ▣ Data is hidden and cannot be accessed by external functions,
 - ▣ Objects may communicate with each other through functions,
 - ▣ Now data and functions can be easily added whenever necessary.
 - ▣ Follows bottom-up approach in program design,

- ❑ “Object Oriented Programming as an approach that provides a way of modularizing programs by creating partitioned memory area ,for both data and functions that can be used as templates for creating copies of such modules on demand.”
- ❑ Thus, an object is considered to be a partitioned area of computer memory that stores data and set of operations that can access that data Since the memory partitions are independent, the objects can be used in a variety of different programs without modifications.

Key concepts of OOPS

6

- ❑ The Object-Oriented Programming paradigm emphasizes the data rather than the algorithm.
- ❑ It implements programs using classes and objects.
- ❑ The Object Oriented Programing has been developed to overcome the drawbacks of Procedural and Structured programming.
- ❑ It is widely accepted that Object Oriented Programming is the most important and powerful way of creating software.
- ❑ The Object-Oriented Programming approach mainly encourages:
 - ▣ **Modularization:** where the program can be decomposed into modules.
 - ▣ **Software reuse:** where a program can be composed from existing and new modules.

Data Abstraction

7

- ❑ Abstraction refers to showing only the essential features without revealing background details.
- ❑ Classes use the concept of abstraction to define a list of abstract attributes and function which operate on these attributes.
- ❑ They encapsulate all the essential properties of the object that are to be created. The attributes are called as **Data Members** because they hold information.
- ❑ The functions that operate on these data called **Methods** or **Member Functions**.

Data Encapsulation

8

- ❑ The mechanism by which the data and functions are bound together into a single unit is known as Encapsulation.
- ❑ It implements abstraction.
- ❑ Encapsulation is about binding the data variables and functions together in class. It can also be called **Data Binding**.
- ❑ Encapsulation is the most striking feature of a class.
- ❑ The data is not accessible to the outside world, and only those functions which are wrapped in the class can access it. These functions provide the interface between the object's data and the program.
- ❑ This encapsulation of data from direct access by the program is called **Data Hiding or Information Hiding**.

Class

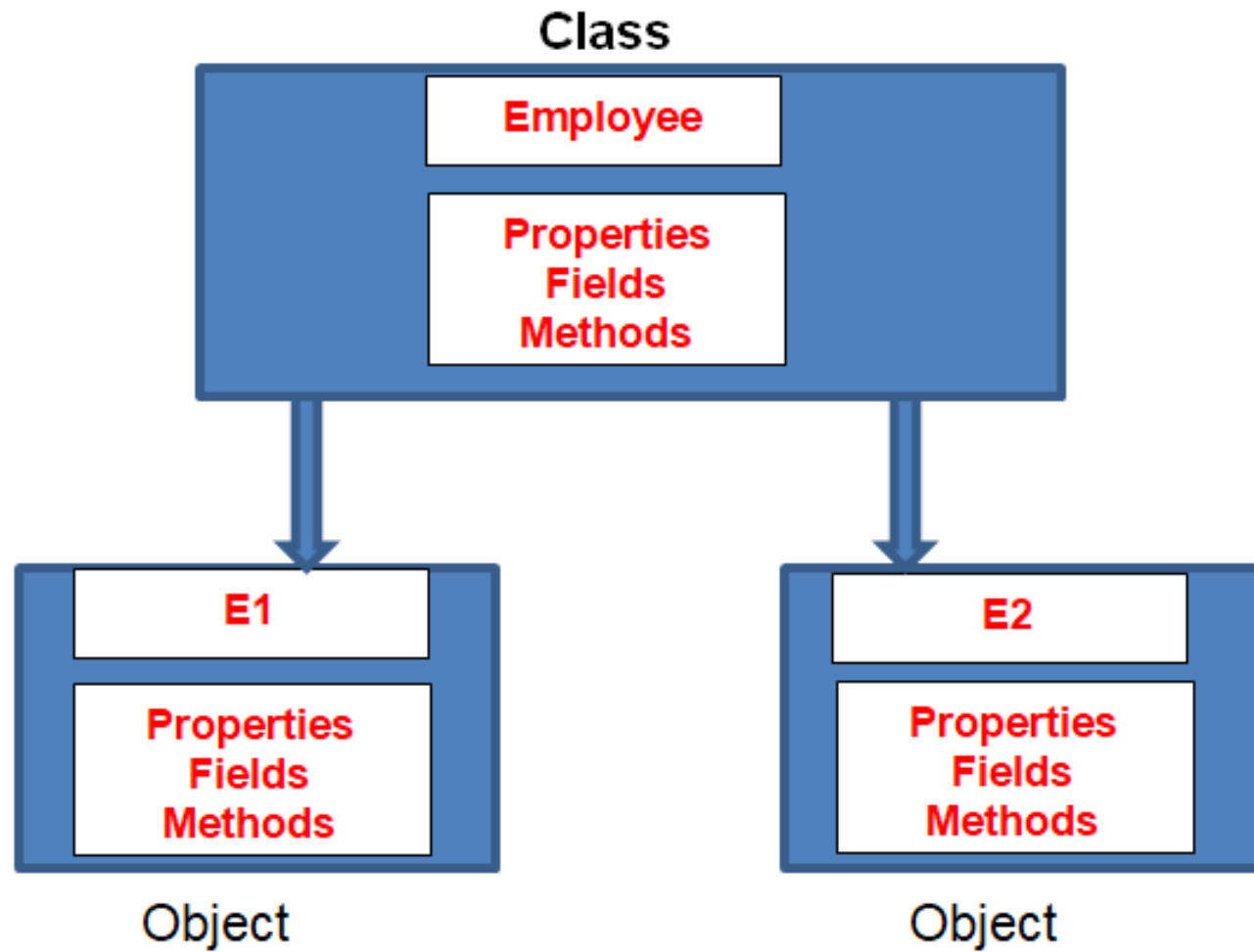
9

- ❑ It is similar to structures in C and C++ language.
- ❑ A Class is a construct in C# which is used to bind data and its associated function together into a single unit using the encapsulation concept.
- ❑ They may or may not be of similar data types. With the addition to data types, classes give you the provision to keep your data secure and add functions within the class, unlike what we saw in structures in C.
- ❑ Class is a reference data type, which holds its own data members and member functions, which can be accessed and used by creating an instance of that class..
- ❑ Classes can also be defined as a template or blueprint representing a group of objects that share common properties and relationship.

Objects

10

- ❑ Objects are the basic unit of OOP.
- ❑ Basically, an object is created from a class. They are instances of class also called class variables.
- ❑ An identifiable entity with some characteristics and behavior is called an object.
- ❑ An object is an entity that has state and behavior. Here, state means data and behavior means functionality.
- ❑ An Object is an instance of a Class. When a class is defined, no memory is allocated but when it is instantiated (i.e. an object is created) memory is allocated.



General form of Class

12

```
<access specifier> class  class_name {  
    // member variables  
    <access specifier> <data type> variable1;  
    <access specifier> <data type> variable2;  
    ...  
    <access specifier> <data type> variableN;  
    // member methods  
    <access specifier> <return type> method1(parameter_list) {  
        // method body  
    }  
    <access specifier> <return type> method2(parameter_list) {  
        // method body  
    }  
    ...  
    <access specifier> <return type> methodN(parameter_list) {  
        // method body  
    }  
}
```

Defining a Class

13

- A class definition starts with the keyword `class` followed by the class name; and the class body enclosed by a pair of curly braces.
- `Class` is a keyword and class name is any valid C# identifier. Everything inside the class is optional.
- Class members may have fields, methods, constructors, destructors, properties, indexers, delegates.

□ Eg:

```
class Program
{
    int a=10;
    void display();
    public static void Main()
    {

    }
}
```

Access Modifiers

14

- The Access Specifiers in C# are also called access modifiers which are used to define the scope of the type as well as the scope of their members. That is who can access them and who cannot access them are defined by the Access Specifiers.
- C# supports 5 access specifiers, they are as follows
 - ▣ Private
 - ▣ Internal
 - ▣ Protected
 - ▣ Protected Internal
 - ▣ Public
- Members that are defined in a type with any scope or specifiers are always accessible within that type; restriction comes into the picture only when they try to access them outside of the type.
- The default access specifier for a class type is **internal**. Default access for the members is **private**.

Modifier	Accessibility Control
private	member is accessible only within the class containing the member.
public	member is accessible from anywhere outside the class and it is accessible in derived classes.
protected	member is visible only to its own class and its derived class.
internal	member is available within the assembly or component that is being created, but not to the clients of the component.
protected internal	available in containing program or assembly and in the derived classes.

CLASS EXAMPLES





Program.cs

Classeg.Program Main()

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Classeg
{
    class Program
    {
        int a;

        public static void Main()
        {
            Program p = new Program();
            Console.Write("Enter the Value of A:");
            p.a = Convert.ToInt32(Console.ReadLine());

            Console.WriteLine("Entered Value is {0}", p.a);
            Console.ReadKey();
        }
    }
}
```

Solution Explorer

Solution 'Classeg' (1 project)

- Classeg
 - Properties
 - References
 - Program.cs

Solut... Team... Class...

Properties





Programus

Classeg.Program

Main()

```
namespace Classeg
{
    class Program
    {
        int a;
        void display()
        {
            Console.Write("Enter the Value of A:");
            a = Convert.ToInt32(Console.ReadLine());

            Console.WriteLine("Entered Value is {0}", a);
        }

        public static void Main()
        {
            Program p = new Program();
            p.display();
            Console.ReadKey();
        }
    }
}
```

Solution Explorer

Solution 'Classeg' (1 project)

- Classeg
 - Properties
 - References
 - Programus

Solut... Team... Class...

Properties



obj.cs Program.cs

Classeg.obj Main()

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Classeg
{
    class obj
    {
        int a;
        public void display()
        {
            Console.WriteLine(" The Value of A is {0}", a);
        }
        public static void Main()
        {
            obj o1 = new obj();
            Console.Write("Enter the Value of A:");
            o1.a = Convert.ToInt32(Console.ReadLine());
            o1.display();

            obj o2 = new obj();
            Console.Write("Enter the Value of A:");
            o2.a = Convert.ToInt32(Console.ReadLine());
            o2.display();
            if (o1 == o2)
                Console.WriteLine("Objects are Equal ");
            else
                Console.WriteLine("Objects are Not Equal");

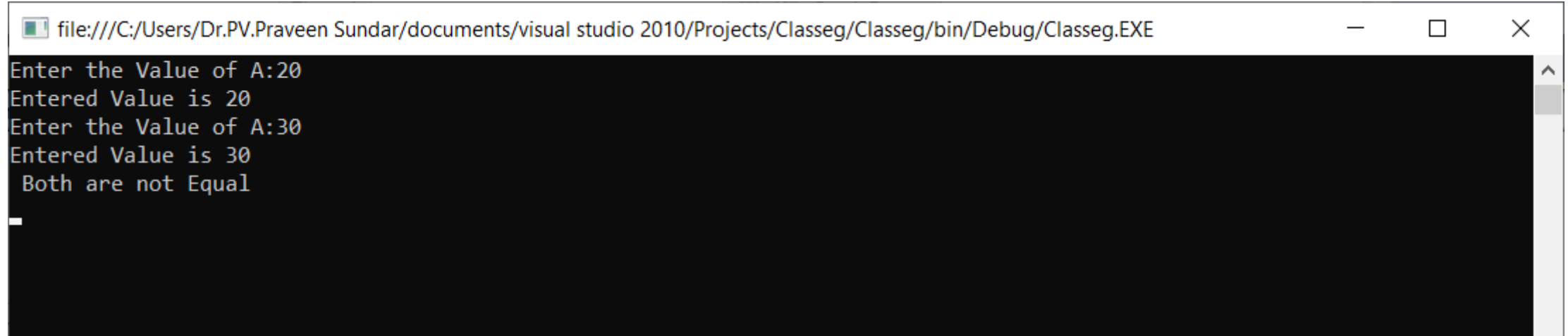
            Console.ReadKey();
        }
    }
}
```

Solution Explorer

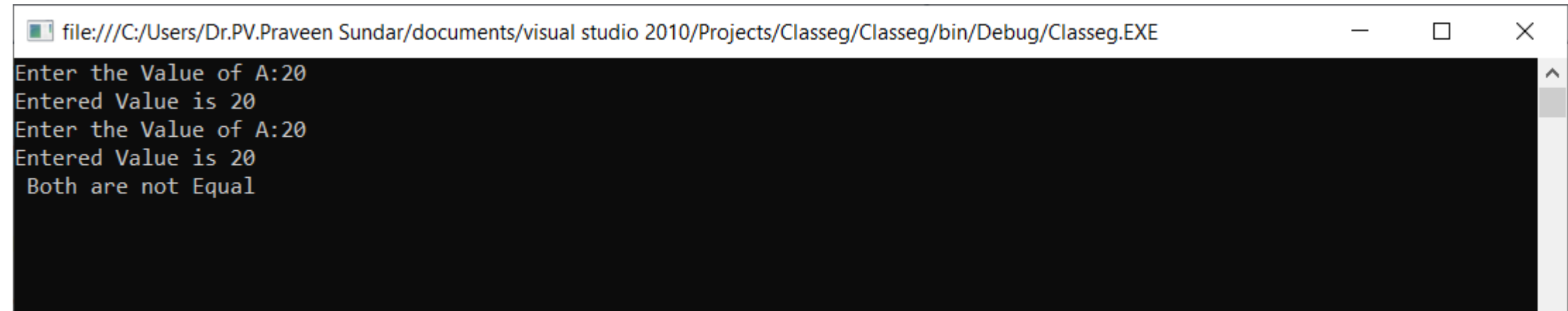
Solution 'Classeg' (1 project)

- Classeg
 - Properties
 - References
 - obj.cs
 - Program.cs

Properties



```
file:///C:/Users/Dr.PV.Praveen Sundar/documents/visual studio 2010/Projects/Classeg/Classeg/bin/Debug/Classeg.EXE
Enter the Value of A:20
Entered Value is 20
Enter the Value of A:30
Entered Value is 30
Both are not Equal
```



```
file:///C:/Users/Dr.PV.Praveen Sundar/documents/visual studio 2010/Projects/Classeg/Classeg/bin/Debug/Classeg.EXE
Enter the Value of A:20
Entered Value is 20
Enter the Value of A:20
Entered Value is 20
Both are not Equal
```



Program.cs

Classeg.Program Main()

```
namespace Classeg
{
    class Program
    {
        int a;
        void display()
        {
            Console.Write("Enter the Value of A:");
            a = Convert.ToInt32(Console.ReadLine());

            Console.WriteLine("Entered Value is {0}", a);
        }

        public static void Main()
        {
            Program p = new Program();
            p.display();
            Program p1 = new Program();
            p1.display();
            if ((p is Program) && (p1 is Program))
            {
                Console.WriteLine(" Both are of objects of the Same Classes ");
            }
            else
            {
                Console.WriteLine(" Both are not objects of the Same Classes ");
            }
            Console.ReadKey();
        }
    }
}
```

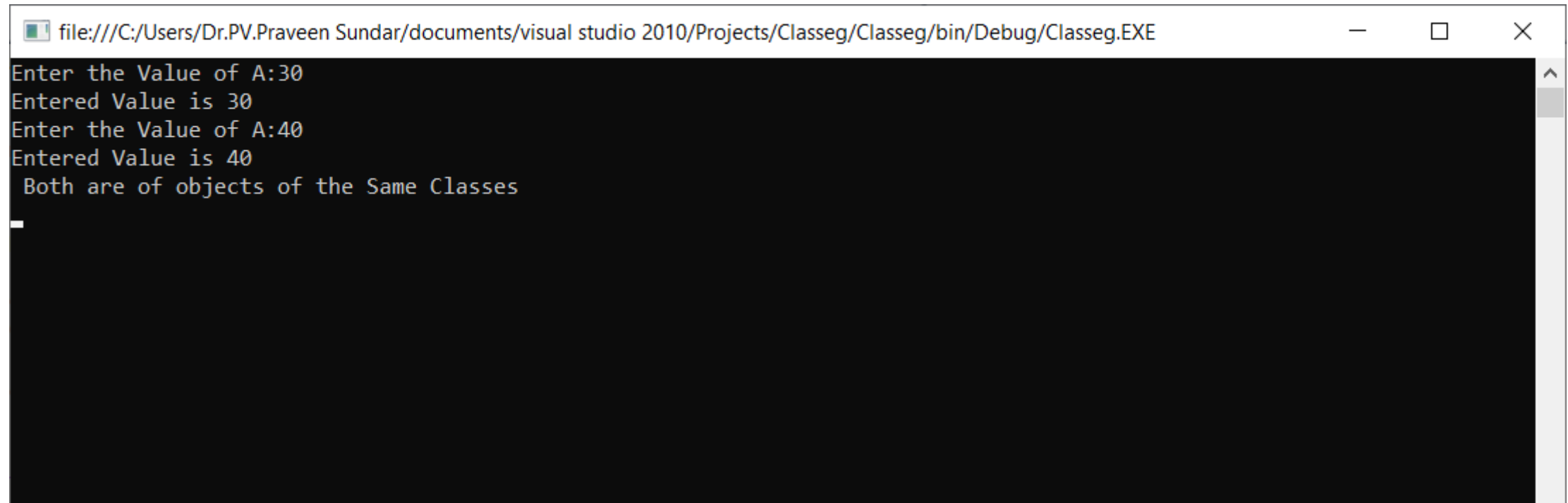
Solution Explorer

Solution 'Classeg' (1 project)

- Classeg
 - Properties
 - References
 - Program.cs

Solut... Team... Class...

Properties



A screenshot of a Windows command prompt window. The title bar shows the file path: `file:///C:/Users/Dr.PV.Praveen Sundar/documents/visual studio 2010/Projects/Classeg/Classeg/bin/Debug/Classeg.EXE`. The window contains the following text:

```
Enter the Value of A:30
Entered Value is 30
Enter the Value of A:40
Entered Value is 40
Both are of objects of the Same Classes
```

The text is displayed in a monospaced font on a black background. A small white cursor is visible on the line following the last line of output.



Program.cs

Classeg.Program Main()

```
namespace Classeg
{
    class Program
    {
        int a;
        void display()
        {
            Console.Write("Enter the Value of A:");
            a = Convert.ToInt32(Console.ReadLine());

            Console.WriteLine("Entered Value is {0}", a);
        }

        public static void Main()
        {
            Program p = new Program();
            p.display();
            Program p1 = new Program();
            p1.display();
            if (p.a == p1.a )
            {
                Console.WriteLine(" Both are Equal");
            }
            else
            {
                Console.WriteLine(" Both are not Equal ");
            }
            Console.ReadKey();
        }
    }
}
```

100 %

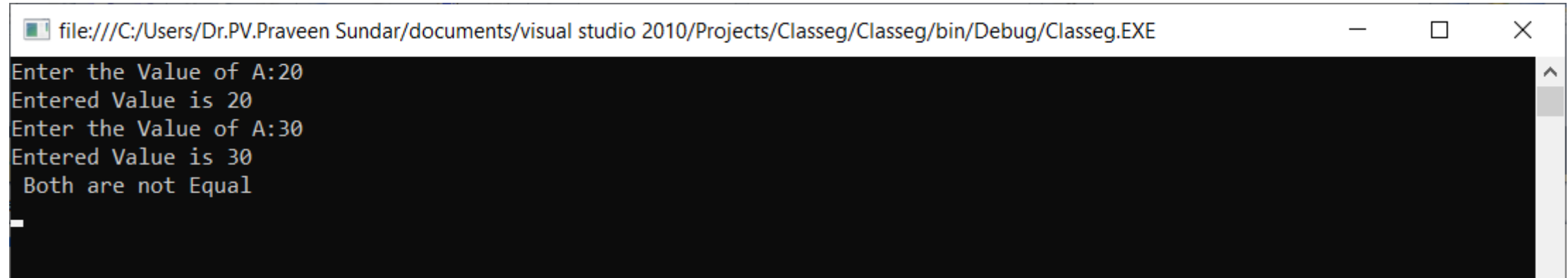
Solution Explorer

Solution 'Classeg' (1 project)

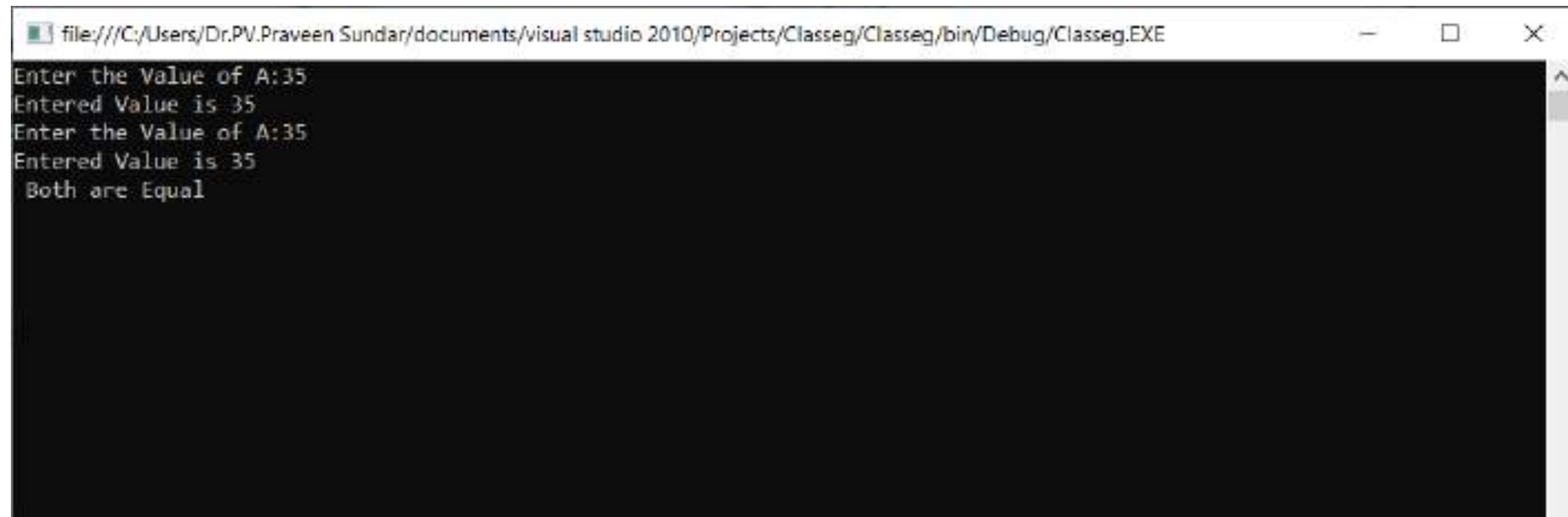
- Classeg
 - Properties
 - References
 - Program.cs

Solut... Team... Class...

Properties

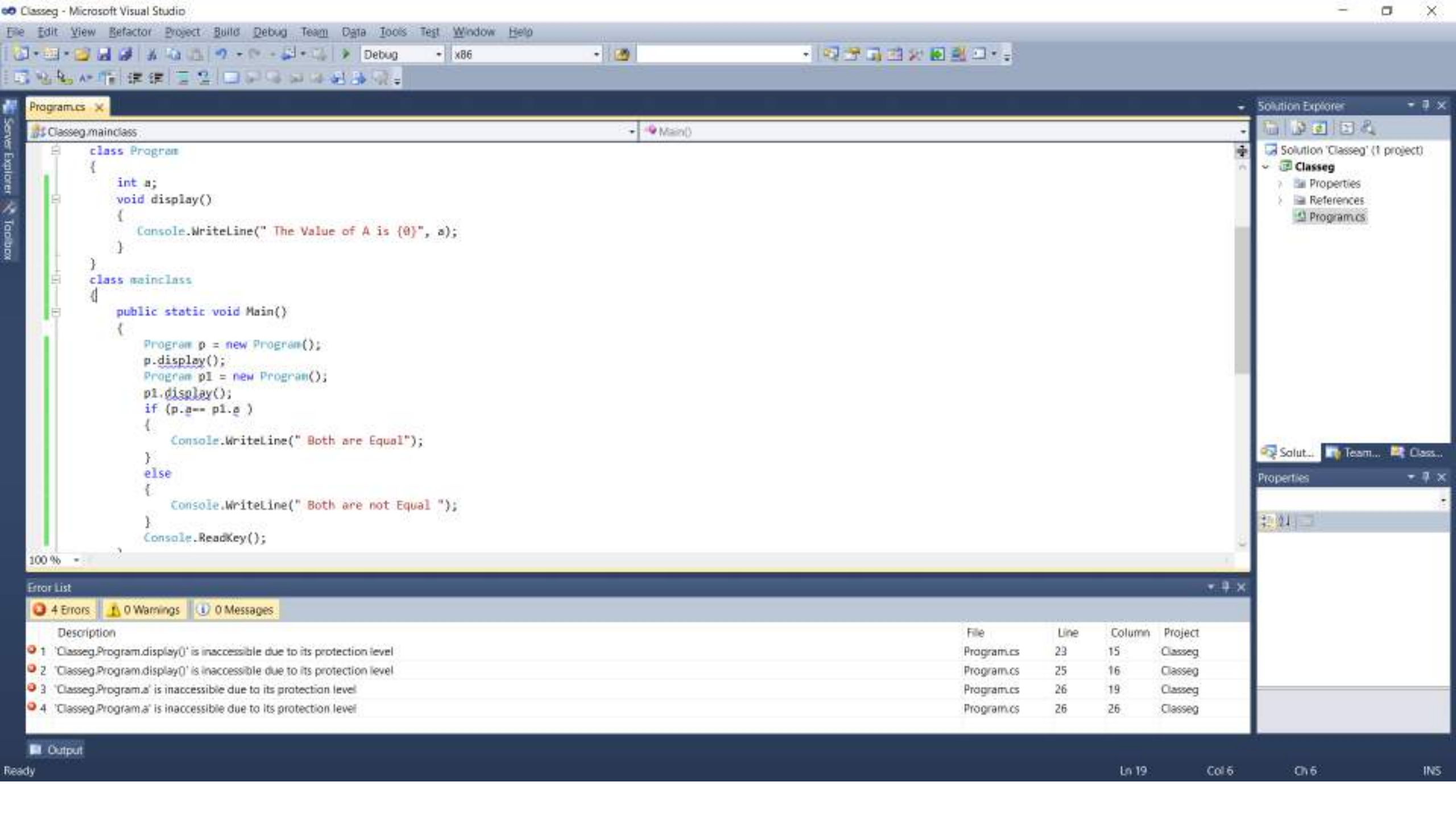
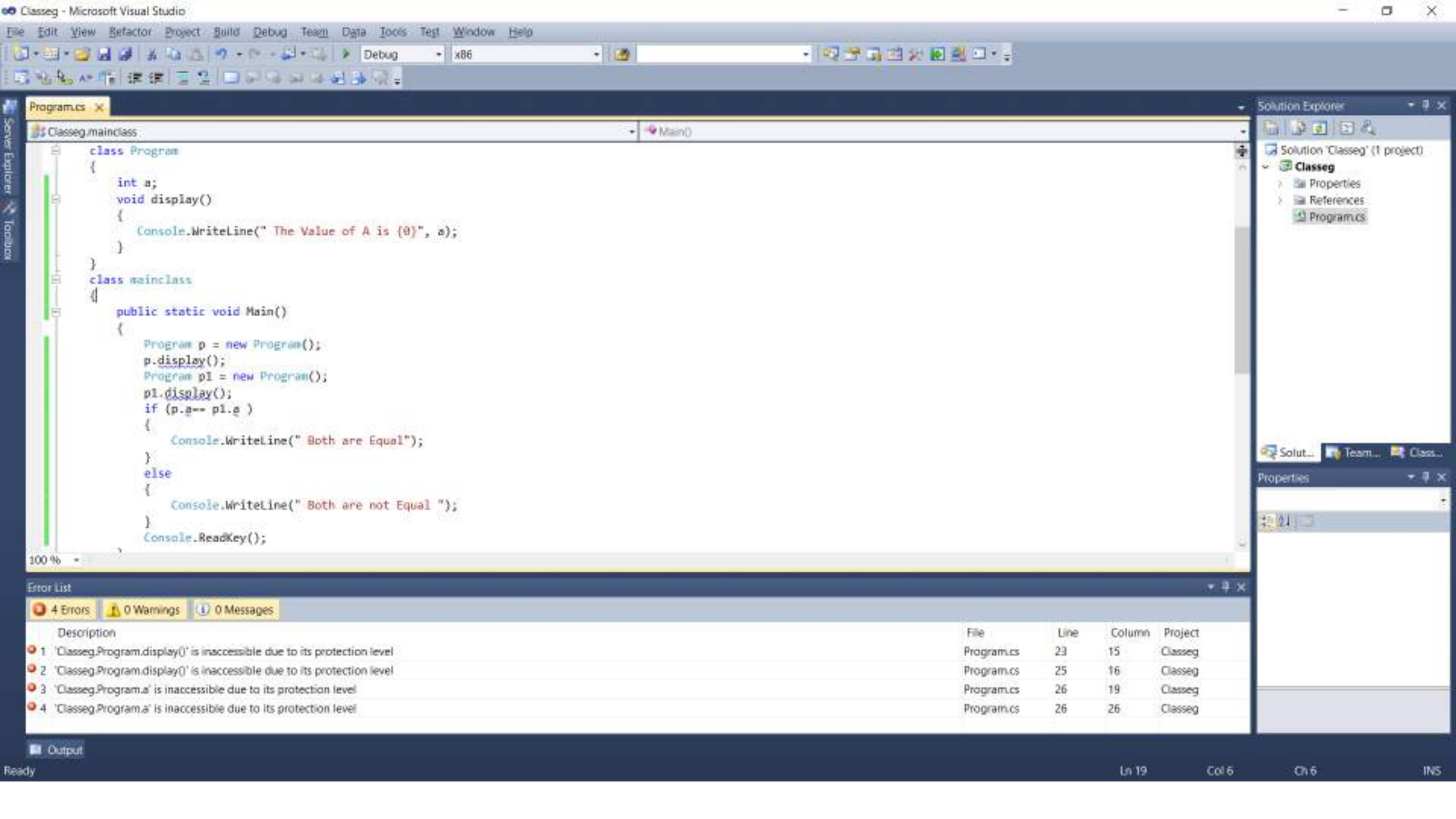


```
file:///C:/Users/Dr.PV.Praveen Sundar/documents/visual studio 2010/Projects/Classeg/Classeg/bin/Debug/Classeg.EXE
Enter the Value of A:20
Entered Value is 20
Enter the Value of A:30
Entered Value is 30
Both are not Equal
```



```
file:///C:/Users/Dr.PV.Praveen Sundar/documents/visual studio 2010/Projects/Classeg/Classeg/bin/Debug/Classeg.EXE
Enter the Value of A:35
Entered Value is 35
Enter the Value of A:35
Entered Value is 35
Both are Equal
```

EXAMPLES OF ACCESS SPECIFIERS





Program.cs

Classeg.mainclass Main()

```
class Program
{
    public int a;
    public void input()
    {
        Console.WriteLine("Enter the Value of A:");
        a = Convert.ToInt32(Console.ReadLine());
        display();
    }
    void display()
    {
        Console.WriteLine(" The Value of A is {0}", a);
    }
}
class mainclass
{
    public static void Main()
    {
        Program p = new Program();
        p.input();
        Program p1 = new Program();
        p1.input();
        if (p.a == p1.a )
        {
            Console.WriteLine(" Both are Equal");
        }
        else
        {
            Console.WriteLine(" Both are not Equal ");
        }
        Console.ReadKey();
    }
}
```

Solution Explorer

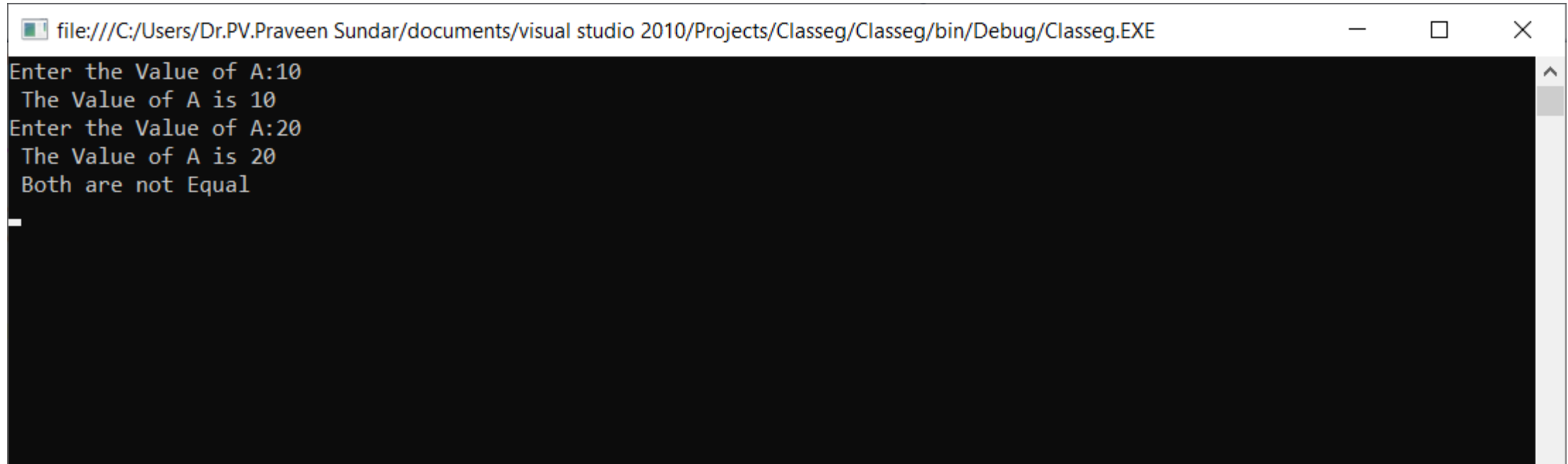
Solution 'Classeg' (1 project)

- Classeg
 - Properties
 - References
 - Program.cs

Solut... Team... Class...

Properties





A screenshot of a Windows command prompt window. The title bar at the top shows the file path: `file:///C:/Users/Dr.PV.Praveen Sundar/documents/visual studio 2010/Projects/Classeg/Classeg/bin/Debug/Classeg.EXE`. The window contains the following text:

```
Enter the Value of A:10
The Value of A is 10
Enter the Value of A:20
The Value of A is 20
Both are not Equal
```

A white cursor is visible on the line following the last output.

Program.cs

Classeg.Program

input()

```
class Program
{
    protected int a;
    public void input()
    {
        Console.WriteLine("Enter the Value of A:");
        a = Convert.ToInt32(Console.ReadLine());
        display();
    }
    void display()
    {
        Console.WriteLine(" The Value of A is {0}", a);
    }
}
class mainclass
{
    public static void Main()
    {
        Program p = new Program();
        p.input();
        Program p1 = new Program();
        p1.input();
        if (p.a == p1.a )
        {
            Console.WriteLine(" Both are Equal");
        }
        else
        {

```

100 %

Solution Explorer

Solution 'Classeg' (1 project)

- Classeg
 - Properties
 - References
 - Program.cs

Solut... Team... Clas...

Properties

Error List

2 Errors 0 Warnings 0 Messages

Description		File	Line	Column	Project
1	'Classeg.Program.a' is inaccessible due to its protection level	Program.cs	32	19	Classeg
2	'Classeg.Program.a' is inaccessible due to its protection level	Program.cs	32	26	Classeg

Output



Program.cs

Classeg/mainclass

Main()

```
class Program
{
    protected int a;
    public void input()
    {
        Console.WriteLine("Enter the Value of A:");
        a = Convert.ToInt32(Console.ReadLine());
        display();
    }
    void display()
    {
        Console.WriteLine(" The Value of A is {0}", a);
    }
}
class mainclass: Program
{
    public static void Main()
    {
        Program p = new Program();
        p.input();
        Program p1 = new Program();
        p1.input();
        if (p.a == p1.a )
        {
            Console.WriteLine(" Both are Equal");
        }
        else
        {
            Console.WriteLine(" Not Equal");
        }
    }
}
```

Error List

2 Errors 0 Warnings 0 Messages

Description					File	Line	Column	Project
1 Cannot access protected member 'Classeg.Program.a' via a qualifier of type 'Classeg.Program'; the qualifier must be of type 'Classeg.mainclass' (or derived from it)					Program.cs	32	19	Classeg
2 Cannot access protected member 'Classeg.Program.a' via a qualifier of type 'Classeg.Program'; the qualifier must be of type 'Classeg.mainclass' (or derived from it)					Program.cs	32	26	Classeg

Output

Solution Explorer

Solution 'Classeg' (1 project)

- Classeg
 - Properties
 - References
 - Program.cs

Solution Team Classeg

Properties



obj.cs

Program.cs

Classeg/mainclass

Main()

```
class Program
{
    protected int a;
    public void input()
    {
        Console.WriteLine("Enter the Value of A:");
        a = Convert.ToInt32(Console.ReadLine());
        display();
    }
    void display()
    {
        Console.WriteLine(" The Value of A is {0}", a);
    }
}
class mainclass: Program
{
    public static void Main()
    {
        mainclass m = new mainclass();
        m.input();

        mainclass m1 = new mainclass();
        m1.input();
        if (m.a == m1.a)
        {
            Console.WriteLine(" Both are Equal");
        }
        else
        {
            Console.WriteLine(" Both are not Equal ");
        }
        Console.ReadKey();
    }
}
```

Solution Explorer

Solution 'Classeg' (1 project)

- Classeg
 - Properties
 - References
 - obj.cs
 - Program.cs

Solut... Team... Class...

Properties

Output

Build succeeded

Ln 33

Col 13

Ch 13

INS



Program.cs

Classeg.mainclass Main()

```
class Program
{
    internal int a;
    internal void input()
    {
        Console.WriteLine("Enter the Value of A:");
        a = Convert.ToInt32(Console.ReadLine());
        display();
    }
    void display()
    {
        Console.WriteLine(" The Value of A is {0}", a);
    }
}
class mainclass: Program
{
    public static void Main()
    {
        mainclass m = new mainclass();
        m.input();

        mainclass m1 = new mainclass();
        m1.input();

        if (m.a == m1.a)
        {
            Console.WriteLine(" Both are Equal");
        }
        else
        {
            Console.WriteLine(" Both are not Equal ");
        }
        Console.ReadKey();
    }
}
```

100 %

Output

Solution Explorer

Solution 'Classeg' (1 project)

- Classeg
 - Properties
 - References
 - Program.cs

Solut... Team... Class...

Properties



Programus*

Classeg.mainclass Main()

```
class Program
{
    internal int a;
    internal void input()
    {
        Console.WriteLine("Enter the Value of A:");
        a = Convert.ToInt32(Console.ReadLine());
        display();
    }
    void display()
    {
        Console.WriteLine(" The Value of A is {0}", a);
    }
}
class mainclass
{
    public static void Main()
    {
        Program p = new Program();
        p.input();

        Program p1 = new Program();
        p1.input();

        if (p.a == p1.a)
        {
            Console.WriteLine(" Both are Equal");
        }
        else
        {
            Console.WriteLine(" Both are not Equal ");
        }
        Console.ReadKey();
    }
}
```

100 %

Output

Solution Explorer

Solution 'Classeg' (1 project)

- Classeg
 - Properties
 - References
 - Programus

Solut... Team... Class...

Properties

Constructors

34

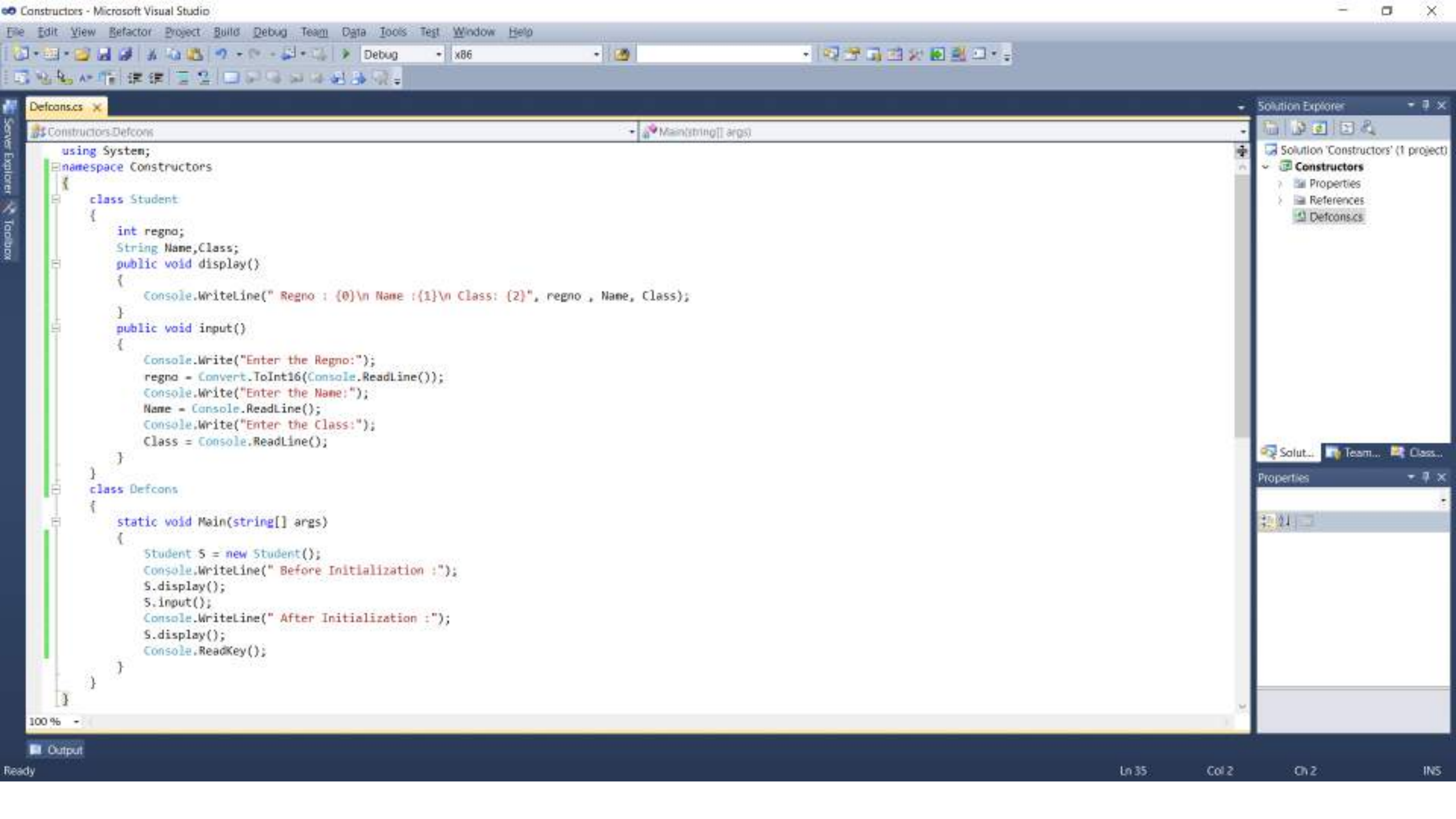
- ❑ A class constructor is a special member function of a class that is executed whenever we create new objects of that class.
- ❑ A constructor has exactly the same name as that of class and it does not have any return type.
- ❑ The Constructors are responsible for two things. One is the object initialization and the other one is memory allocation. The role of the new keyword is to create the object.
- ❑ **Rules to follow while creating the C# Constructors:**
 - ❑ The constructor name should be the same as the class name.
 - ❑ It should not contain return type even void also.
 - ❑ As part of the constructor body return statement with value is not allowed.

Types of Constructor

35

- There are five types of constructors available in C#, they are as follows
 - ▣ Default Constructor
 - ▣ Parameterized Constructor
 - ▣ Copy Constructor
 - ▣ Static Constructor
 - ▣ Private Constructor
- Default Constructor: The Constructor without parameter is called a default constructor. Again the default constructor is classified into two types.
 - ▣ System-defined default constructor
 - ▣ User-defined default constructor

- ❑ **System Defined Default Constructor:** As a programmer, if you are not defined any constructor explicitly in your program, then by default the system will provide one constructor at the time of compilation. That constructor is called a default constructor.
- ❑ The default constructor will assign default values to the data members (non-static variables).
- ❑ As this constructor is created by the system this is also called a system-defined default constructor.



```
file:///c:/users/dr.pv.praveen sundar/documents/visual studio 2010/Projects/Constructors/Constructors/bin/Debug/Constructors.EXE
Before Initialization :
Regno : 0
Name :
Class:
Enter the Regno:21312
Enter the Name:Praveen
Enter the Class:MCA
After Initialization :
Regno : 21312
Name :Praveen
Class: MCA
```

User-defined default constructor

39

- if we want to execute some logic at the time of object creation, that logic may be object initialization logic or some other useful logic, then as a developer, we must provide the constructor explicitly.
- The constructor which is defined by the user without any parameter is called a user-defined default constructor.
- This constructor does not accept any argument but as part of the constructor body, we can write our own logic.



UDCons.cs x Defcons.cs

Constructors\UDCons

Main(string[] args)

```
using System;
namespace Constructors
{
    class Stud
    {
        int regno;
        String Name, Class;
        public void display()
        {
            Console.WriteLine(" Regno : {0}\n Name :{1}\n Class: {2}", regno, Name, Class);
        }
        public Stud()
        {
            Console.Write("Enter the Regno:");
            regno = Convert.ToInt16(Console.ReadLine());
            Console.Write("Enter the Name:");
            Name = Console.ReadLine();
            Console.Write("Enter the Class:");
            Class = Console.ReadLine();
        }
    }
    class UDCons
    {
        static void Main(string[] args)
        {
            Stud S = new Stud();
            Console.WriteLine(" Before Initialization :");
            S.display();
            Console.WriteLine(" After Initialization :");
            S.display();
            Console.ReadKey();
        }
    }
}
```

Solution Explorer

Solution 'Constructors' (1 project)

- Constructors
 - Properties
 - References
 - Defcons.cs
 - UDCons.cs

Solut... Team... Class...

Properties

file:///c:/users/dr.pv.praveen sundar/documents/visual studio 2010/Projects/Constructors/Constructors/bin/Debug/Constructors.EXE

Enter the Regno:21312
Enter the Name:Sundar
Enter the Class:MCA
Before Initialization :
Regno : 21312
Name :Sundar
Class: MCA
After Initialization :
Regno : 21312
Name :Sundar
Class: MCA

Parameterized Constructor

42

- ❑ The drawback of the above user-defined default constructor is every instance (i.e. object) of the class will be initialized (assigned) with the same values. That means it is not possible to initialize each instance of the class with different values.
- ❑ If you want to initialize the object dynamically with the user-given values then you need to use the parameterized constructor. The advantage is that you can initialize each object with different values.
- ❑ The developer given constructor with parameters is called the parameterized constructor in C#. With the help of a Parameterized constructor, we can initialize each instance of the class with different values. That means using parameterized constructor we can store a different set of values into different objects created to the class.



Param_constructor.cs UDCons.cs Defcons.cs

Constructors.Param_constructor Main(string[] args)

```
using System;
namespace Constructors
{
    public class Stud_details
    {
        int regno;
        String Name, Class;
        public void display()
        {
            Console.WriteLine(" Regno : {0}\n Name :{1}\n Class: {2}", regno, Name, Class);
        }
        public Stud_details(int rno, String name, String course)
        {
            this.regno = rno;
            this.Name = name; this.Class = course;
        }
    }
    class Param_constructor
    {
        static void Main(string[] args)
        {
            int rno;
            String name, course;
            Console.Write("Enter the Regno:");
            rno = Convert.ToInt16(Console.ReadLine());
            Console.Write("Enter the Name:");
            name = Console.ReadLine();
            Console.Write("Enter the Class:");
            course = Console.ReadLine();
            Stud_details sd = new Stud_details(rno, name, course);
            sd.display();
            Console.ReadKey();
        }
    }
}
```

99 %

Output

Ready

Ln 35

Col 2

Ch 2

INS

Solution Explorer

Solution 'Constructors' (1 project)

- Constructors
 - Properties
 - References
 - Defcons.cs
 - Param_constructor.cs
 - UDCons.cs

Solut... Team... Class...

Properties

file:///c:/users/dr.pv.praveen sundar/documents/visual studio 2010/Projects/Constructors/Constructors/bin/Debug/Constructors.EXE

Enter the Regno:21312
Enter the Name:Praveen
Enter the Class:MCA
Regno : 21312
Name :Praveen
Class: MCA

- ❑ In C#, within a class, we can define any number of constructors. But the most important point that you need to remember is that each and every constructor must have a different signature.
- ❑ Different signature means the number, type, and parameter order should be different.
- ❑ So in a class, we can define one no-argument constructor plus 'n' number of parameterized constructors in C#.



Cons_ovrloading.cs* Param_constructor.cs UDCons.cs Defcons.cs

Constructors.Student_details Student_details(String name, String course)

```
using System;
namespace Constructors
{
    public class Student_details
    {
        int regno;
        String Name, Class;
        public void display()
        {
            Console.WriteLine(" Regno : {0}\n Name :{1}\n Class: {2}", regno, Name, Class);
        }
        public Student_details(int rno, String name, String course)
        {
            Console.WriteLine(" ;Parameterized Constructor with All Arguments :");
            this.regno = rno;
            this.Name = name;
            this.Class = course;
            display();
        }
        public Student_details(int rno)
        {
            Console.WriteLine(" ;Parameterized Constructor with one Argument :");
            this.regno = rno;
            display();
        }
        public Student_details(int rno, String name)
        {
            Console.WriteLine(" ;Parameterized Constructor with Two Arguments :");
            this.regno = rno;
            this.Name = name;
            display();
        }
    }
}
```

Solution Explorer

Solution 'Constructors' (1 project)

- Constructors
 - Properties
 - References
 - Cons_ovrloading.cs
 - Defcons.cs
 - Param_constructor.cs
 - UDCons.cs

Solut... Team... Class...

Properties



Cons_ovreloading.cs* Param_constructor.cs UDCons.cs Defcons.cs

Constructors.Cons_ovreloading Main()

```
public Student_details(String name, String course)
{
    Console.WriteLine(" ;Parameterized Constructor with Name and Course :");
    this.Name = name;
    this.Class = course;
    display();
}

public Student_details()
{
    Console.WriteLine(" User Defined Default Constructor :");
    Console.Write("Enter the Regno:");
    regno = Convert.ToInt16(Console.ReadLine());
    Console.Write("Enter the Name:");
    Name = Console.ReadLine();
    Console.Write("Enter the Class:");
    Class = Console.ReadLine();
    display();
}

}

class Cons_ovreloading
{
    public static void Main()
    {
        Student_details sd = new Student_details();
        Student_details sd1 = new Student_details(21312);
        Student_details sd2 = new Student_details(21312, "Praveen");
        Student_details sd3 = new Student_details("Praveen", "MCA");
        Console.ReadKey();
    }
}
```

Solution Explorer

Solution 'Constructors' (1 project)

- Constructors
 - Properties
 - References
 - Cons_ovreloading.cs
 - Defcons.cs
 - Param_constructor.cs
 - UDCons.cs

Solut... Team... Class...

Properties

User Defined Default Constructor :

Enter the Regno:100

Enter the Name:Sundar

Enter the Class:BCA

Regno : 100

Name :Sundar

Class: BCA

;Parameterized Constructor with one Argument :

Regno : 21312

Name :

Class:

;Parameterized Constructor with Two Arguments :

Regno : 21312

Name :Praveen

Class:

;Parameterized Constructor with Name and Course :

Regno : 0

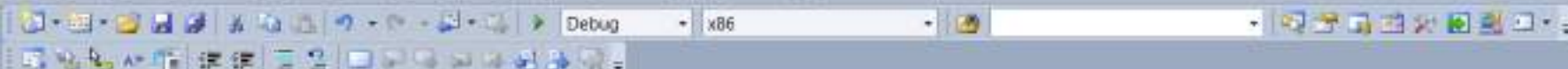
Name :Praveen

Class: MCA

Copy Constructor

49

- ❑ The constructor which takes a parameter of the class type is called a copy constructor.
- ❑ This constructor is used to copy one object's data into another object.
- ❑ The main purpose of the copy constructor is to initialize a new object (instance) with the values of an existing object (instance).



CopyCons.cs* Cons_oveloading.cs Param_constructor.cs UDCons.cs Defcons.cs

```
using System;
namespace Constructors
{
    class CopyCons
    {
        int regno;
        String Name, Class;
        public void display()
        {
            Console.WriteLine(" Regno : {0}\n Name :{1}\n Class: {2}", regno, Name, Class);
        }
        CopyCons(int rno, String name, String course)
        {
            Console.WriteLine(" :Parameterized Constructor with All Arguments :");
            this.regno = rno;
            this.Name = name;
            this.Class = course;
            display();
        }
        CopyCons(CopyCons C)
        {
            this.regno = C.regno;
            this.Name = C.Name;
            this.Class = C.Class;
        }
        public static void Main()
        {
            CopyCons obj1 = new CopyCons(21312, "Praveen", "MCA");
            CopyCons obj2 = new CopyCons(obj1);
            obj2.display();
            Console.ReadKey();
        }
    }
}
```

100 %

Output

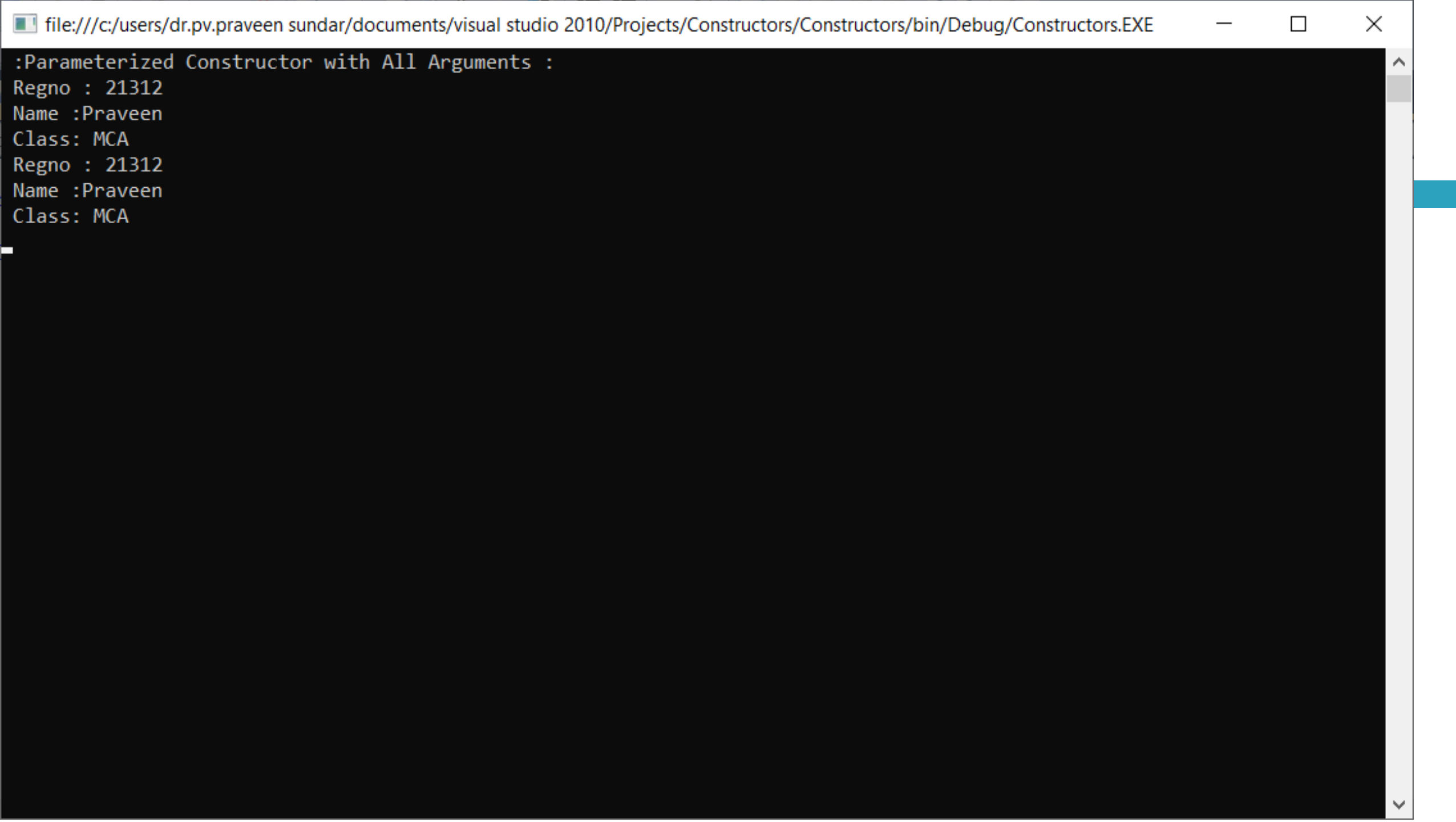
Solution Explorer

Solution 'Constructors' (1 project)

- Constructors
 - Properties
 - References
 - Cons_oveloading.cs
 - CopyCons.cs
 - Defcons.cs
 - Param_constructor.cs
 - UDCons.cs

Solut... Team... Class...

Properties





CopyCons.cs* Cons_oveloading.cs Param_constructor.cs UDCons.cs Defcons.cs

```
using System;
namespace Constructors
{
    class CopyCons
    {
        int regno;
        String Name, Class;
        public void display()
        {
            Console.WriteLine(" Regno : {0}\n Name :{1}\n Class: {2}", regno, Name, Class);
        }
        CopyCons(int rno, String name, String course)
        {
            Console.WriteLine(" :Parameterized Constructor with All Arguments :");
            this.regno = rno;
            this.Name = name;
            this.Class = course;
            display();
        }
        CopyCons(CopyCons C)
        {
            this.regno = C.regno;
            this.Name = C.Name;
            this.Class = C.Class;
        }
        public static void Main()
        {
            CopyCons obj1 = new CopyCons(21312, "Praveen", "MCA");
            CopyCons obj2 = new CopyCons(obj1);
            CopyCons obj3 = obj2;
            obj2.display();
            obj3.display();
        }
    }
}
```

100 %

Output

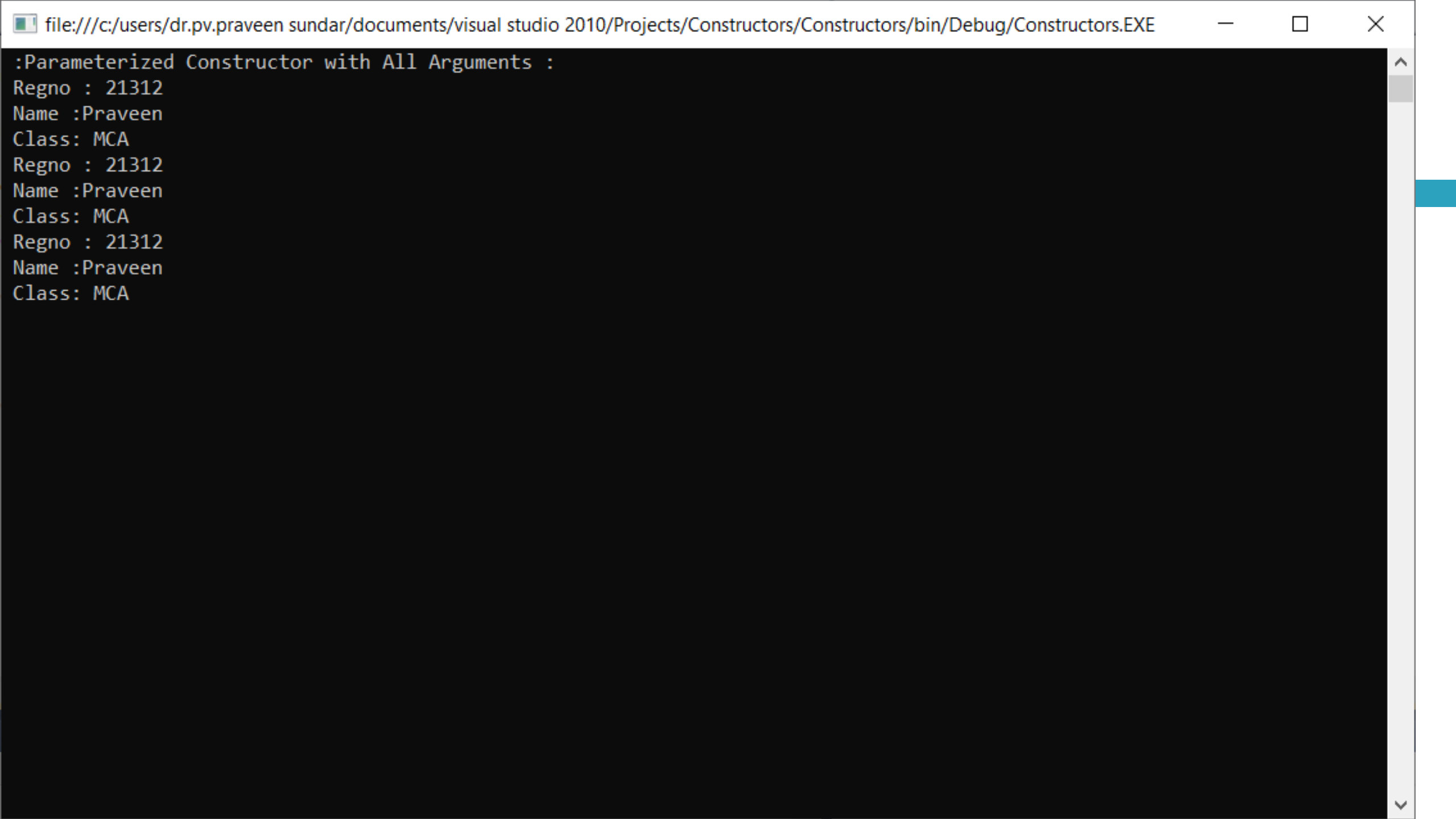
Solution Explorer

Solution 'Constructors' (1 project)

- Constructors
 - Properties
 - References
 - Cons_oveloading.cs
 - CopyCons.cs
 - Defcons.cs
 - Param_constructor.cs
 - UDCons.cs

Solut... Team... Class...

Properties



:Parameterized Constructor with All Arguments :

Regno : 21312
Name :Praveen
Class: MCA
Regno : 21312
Name :Praveen
Class: MCA
Regno : 21312
Name :Praveen
Class: MCA

Private Constructor

54

- ❑ In C#, it is also possible to create a constructor as private.
- ❑ The constructor whose accessibility is private is known as a private constructor.
- ❑ When a class contains a private constructor then we cannot create an object for the class outside of the class. So, private constructors are used to creating an object for the class within the same class.
- ❑ Generally, private constructors are used in the Remoting concept.



Privatecons.cs CopyCons.cs Cons_ovreloading.cs Param_constructor.cs UDCons.cs Defcons.cs

Constructors.Privatecons Main(string[] args)

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Constructors
{
    class Privatecons
    {
        int regno;
        String Name, Class;
        public void display()
        {
            Console.WriteLine(" Regno : {0}\n Name :{1}\n Class: {2}", regno, Name, Class);
        }
        Privatecons()
        {
            Console.Write("Enter the Regno:");
            regno = Convert.ToInt16(Console.ReadLine());
            Console.Write("Enter the Name:");
            Name = Console.ReadLine();
            Console.Write("Enter the Class:");
            Class = Console.ReadLine();
        }
        static void Main(string[] args)
        {
            Privatecons obj1 = new Privatecons();
            obj1.display();
            Console.ReadKey();
        }
    }
}
```

Solution Explorer

Solution 'Constructors' (1 project)

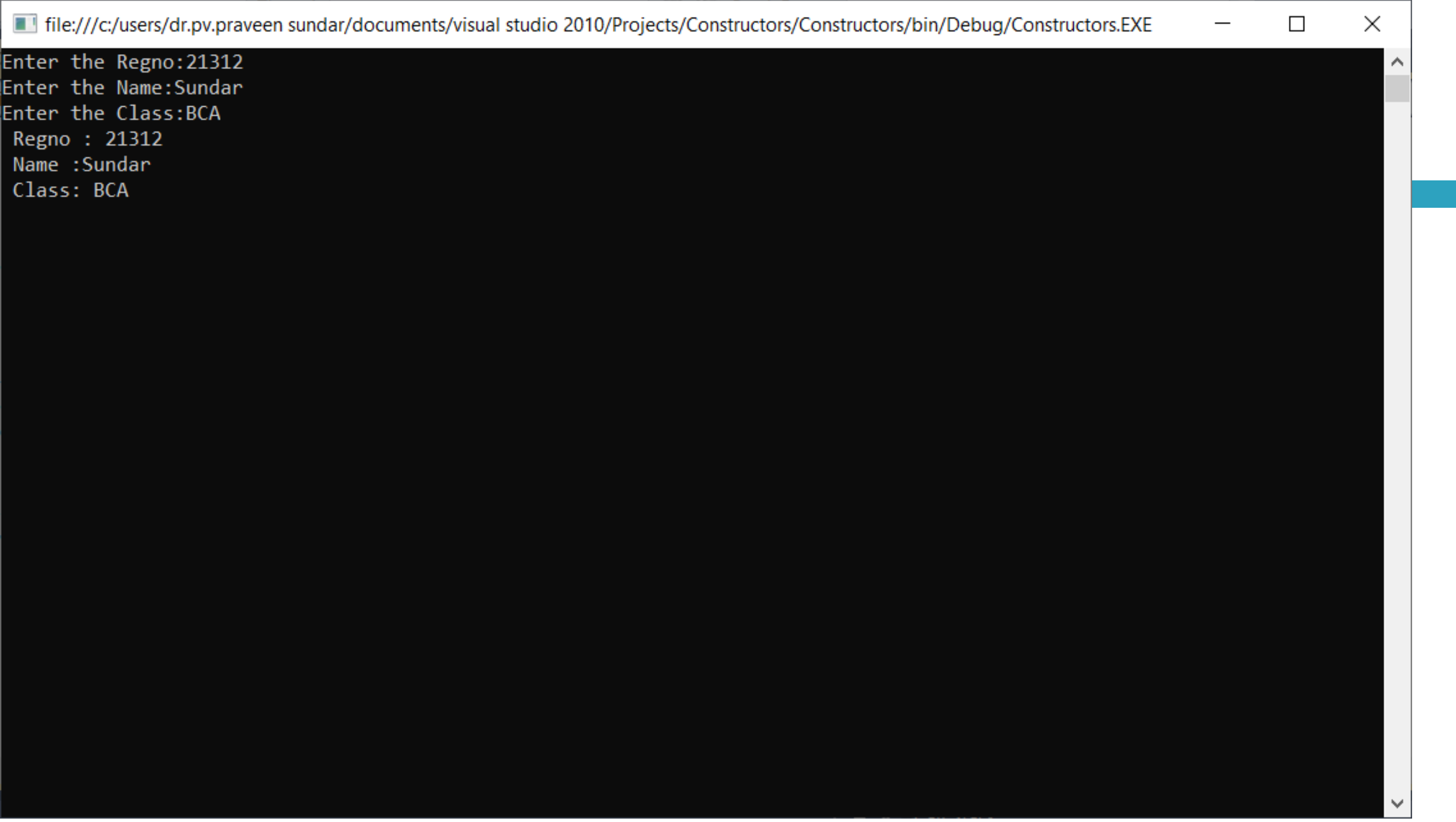
- Constructors
 - Properties
 - References
 - Cons_ovreloading.cs
 - CopyCons.cs
 - Defcons.cs
 - Param_constructor.cs
 - Privatecons.cs
 - staticcons.cs
 - UDCons.cs

Solut... Team... Class...

Properties

100 %

Output



Enter the Regno:21312

Enter the Name:Sundar

Enter the Class:BCA

Regno : 21312

Name :Sundar

Class: BCA

Static Constructor

57

- When a constructor is created using a static keyword, it will be invoked only once for all of the instances of the class and it is invoked during the creation of the first instance of the class or the first reference to a static member in the class.
- A static constructor is used to initialize static fields of the class and to write the code that needs to be executed only once.

- Points to Remember while creating Static Constructor in C#:
 - ▣ There can be only one static constructor in a class.
 - ▣ The static constructor should be without any parameter.
 - ▣ It can only access the static members of the class.
 - ▣ There should not be any access modifier in the static constructor definition.
 - ▣ If a class is static then we cannot create the object for the static class.
 - ▣ Static constructor will be invoked only once i.e. at the time of first object creation of the class, from 2nd object creation onwards static constructor will not be called.



staticcons.cs* Privatecons.cs CopyCons.cs Cons_ovreloading.cs Param_constructor.cs UDCons.cs Defcons.cs

Constructors.mainprogram Main()

```
using System;

namespace Constructors
{
    class staticcons
    {
        int i;
        static int j;
        public staticcons()
        {
            Console.WriteLine("Default Constructor");
            i = j++;
        }
        static staticcons()
        {
            j = 10;
            Console.WriteLine("Static Function");
        }
        public void Display()
        {
            Console.WriteLine(" The Value of i and j is {0},{1}", i, j);
        }
    }
    class mainprogram
    {
        public static void Main()
        {
            staticcons obj1 = new staticcons();
            obj1.Display();
            staticcons obj2 = new staticcons();
            obj2.Display();
            Console.ReadKey();
        }
    }
}
```

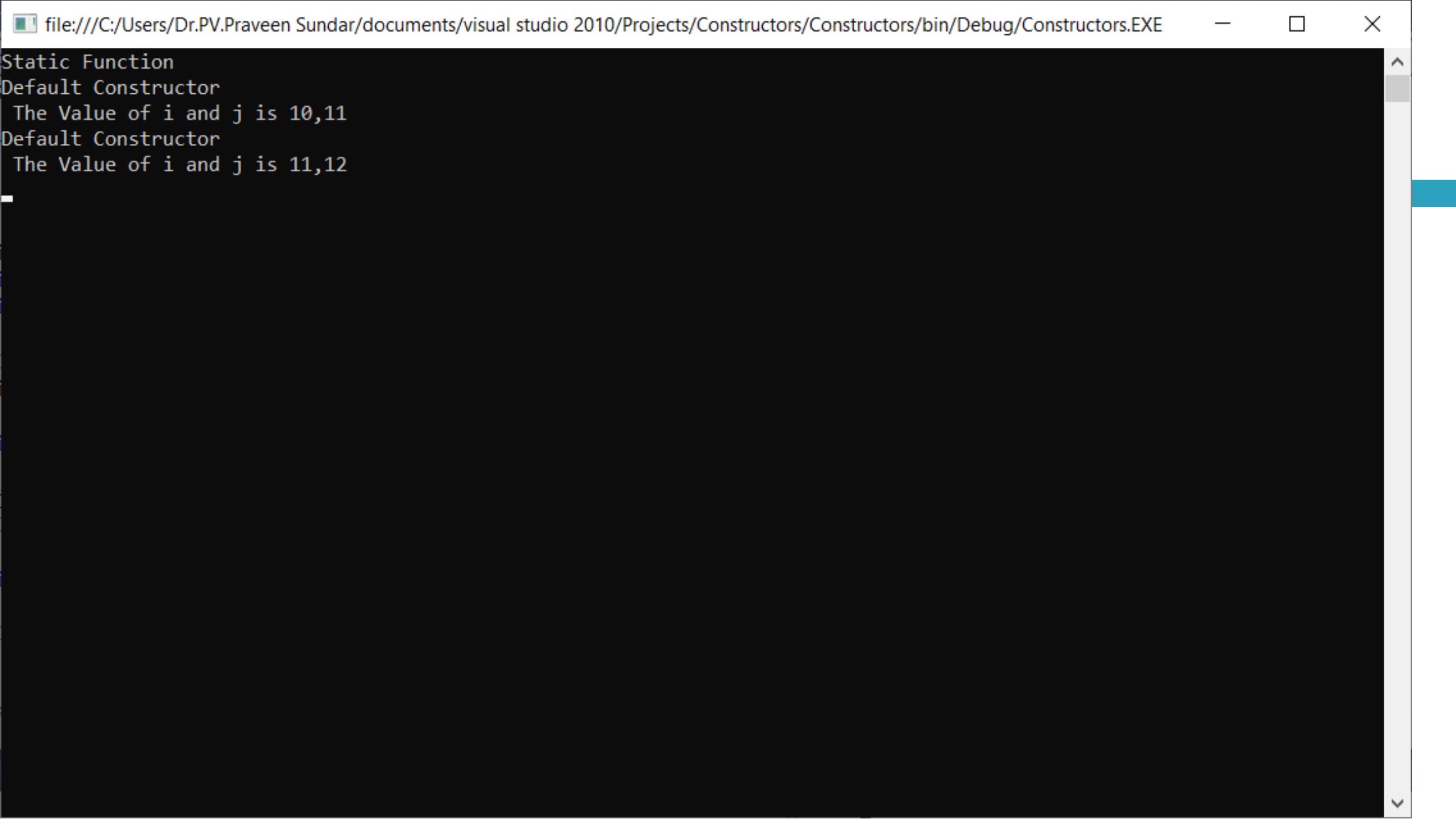
Solution Explorer

Solution 'Constructors' (1 project)

- Constructors
 - Properties
 - References
 - Cons_ovreloading.cs
 - CopyCons.cs
 - Defcons.cs
 - Param_constructor.cs
 - Privatecons.cs
 - staticcons.cs
 - UDCons.cs

Solut... Team... Class...

Properties



Static Function

Default Constructor

The Value of i and j is 10,11

Default Constructor

The Value of i and j is 11,12

Destructors

61

- The Destructor is also a special type of method present in a class, just like a constructor, having the same name as the class name but prefix with ~ tilde.
- The constructor in C# is called when the object of the class is created. On the other hand, the destructor in C# is gets executed when the object of the class is destroyed.
- The Constructor and destructor methods will exactly have the same name as the class to which they belong. So to differentiate between these two a tilde (~) operator is used before the destructor method.
- A destructor method cannot have any parameters as well as cannot be applied with any modifiers.

For Eg:

```
class Sample
{
    Sample()
    {
        //constructor
    }
    ~Sample()
    {
        //destructor
    }
}
```

- A destructor method gets called when the object of the class is destroyed.
- The object of a class in C# will be destroyed by the garbage collector in any of the following cases
 - ▣ **Case1:** At the end of a program execution each and every object that is associated with the program will be destroyed by the garbage collector.
 - ▣ **Case2:** The Implicit calling of the garbage collector occurs sometime in the middle of the program execution provided the memory is full so that the garbage collector will identify unused objects of the program and destroys them.
 - ▣ **Case 3:** The Explicit calling of the garbage collector can be done in the middle of program execution with the help of the “GC.Collect()” statement so that if there are any unused objects associated with the program will be destroyed in the middle of the program execution.



staticcons.cs Privatecons.cs CopyCons.cs Cons_ovrloading.cs* Param_constructor.cs UDCons.cs Defcons.cs

Constructors.Student_details Student_details(String name, String course)

```
using System;
namespace Constructors
{
    public class Student_details
    {
        public int regno;
        public String Name, Class;
        public void display()
        {
            Console.WriteLine(" Regno : {0}\n Name :{1}\n Class: {2}", regno, Name, Class);
        }
        public Student_details(int rno, String name, String course)
        {
            Console.WriteLine(" ;Parameterized Constructor with All Arguments :");
            this.regno = rno;
            this.Name = name;
            this.Class = course;
            display();
        }
        public Student_details(int rno)
        {
            Console.WriteLine(" ;Parameterized Constructor with one Argument :");
            this.regno = rno;
            display();
        }
        public Student_details(int rno, String name)
        {
            Console.WriteLine(" ;Parameterized Constructor with Two Arguments :");
            this.regno = rno;
            this.Name = name;
            display();
        }
    }
}
```

Solution Explorer

Solution 'Constructors' (1 project)

- Constructors
 - Properties
 - References
 - Cons_ovrloading.cs
 - CopyCons.cs
 - Defcons.cs
 - Param_constructor.cs
 - Privatecons.cs
 - staticcons.cs
 - UDCons.cs

Solut... Team... Class...

Properties



staticcons.cs Privatecons.cs CopyCons.cs Cons_ovreloading.cs* Param_constructor.cs UDCons.cs Defcons.cs

Constructors.Student_details Student_details(String name, String course)

```
public Student_details(String name, String course)
{
    Console.WriteLine("Parameterized Constructor with Name and Course :");
    this.Name = name;
    this.Class = course;
    display();
}

public Student_details()
{
    Console.WriteLine(" User Defined Default Constructor :");
    Console.Write("Enter the Regno:");
    regno = Convert.ToInt16(Console.ReadLine());
    Console.Write("Enter the Name:");
    Name = Console.ReadLine();
    Console.Write("Enter the Class:");
    Class = Console.ReadLine();
    display();
}

~Student_details()
{
    Console.WriteLine(" The memory of the Object is Released ");
}

class Cons_ovreloading
{
    public static void Main()
    {
        Student_details sd = new Student_details();
        Student_details sd1 = new Student_details(21312);
        Student_details sd2 = new Student_details(21312, "Praveen");
        Student_details sd3 = new Student_details("Praveen", "MCA");
        Console.ReadKey();
    }
}
```

Solution Explorer

Solution 'Constructors' (1 project)

Constructors

Properties

References

Cons_ovreloading.cs

CopyCons.cs

Defcons.cs

Param_constructor.cs

Privatecons.cs

staticcons.cs

UDCons.cs

Solut... Team... Class...

Properties

07-12-2019 02:40 PM 490 Constructors.vshost.exe.manifest

4 File(s) 57,146 bytes

2 Dir(s) 281,249,607,680 bytes free

C:\Users\Dr.PV.Praveen Sundar\Documents\Visual Studio 2010\Projects\Constructors\Constructors\bin\Debug>constructors

User Defined Default Constructor :

Enter the Regno:21312

Enter the Name:Praveen

Enter the Class:MCA

Regno : 21312

Name :Praveen

Class: MCA

;Parameterized Constructor with one Argument :

Regno : 21312

Name :

Class:

;Parameterized Constructor with Two Arguments :

Regno : 21312

Name :Praveen

Class:

;Parameterized Constructor with Name and Course :

Regno : 0

Name :Praveen

Class: MCA

The memory of the Object is Released

The memory of the Object is Released

The memory of the Object is Released

The memory of the Object is Released

C:\Users\Dr.PV.Praveen Sundar\Documents\Visual Studio 2010\Projects\Constructors\Constructors\bin\Debug>



staticcons.cs Privatecons.cs CopyCons.cs Cons_ovreloading.cs Param_constructor.cs UDCons.cs Defcons.cs

Constructors.Cons_ovreloading Main()

```
    Console.WriteLine("Enter the Name:");
    Name = Console.ReadLine();
    Console.WriteLine("Enter the Class:");
    Class = Console.ReadLine();
    display();
}
~Student_details()
{
    Console.WriteLine(" The memory of the Object is Released ");
}
}
class Cons_ovreloading
{
    public static void Main()
    {
        Student_details sd = new Student_details();
        Student_details sd1 = new Student_details(21312);
        Student_details sd2 = new Student_details(21312, "Praveen");
        Student_details sd3 = new Student_details("Praveen", "MCA");
        sd = sd1 = sd2 = sd3 = null;
        GC.Collect();
        Console.ReadKey();
    }
}
```

Solution Explorer

Solution 'Constructors' (1 project)

- Constructors
 - Properties
 - References
 - Cons_ovreloading.cs
 - CopyCons.cs
 - Defcons.cs
 - Param_constructor.cs
 - Privatecons.cs
 - staticcons.cs
 - UDCons.cs

Solut... Team... Class...

Properties

100 %

Output

User Defined Default Constructor :

Enter the Regno:21313

Enter the Name:Rajesh

Enter the Class:BCA

Regno : 21313

Name :Rajesh

Class: BCA

;Parameterized Constructor with one Argument :

Regno : 21312

Name :

Class:

;Parameterized Constructor with Two Arguments :

Regno : 21312

Name :Praveen

Class:

;Parameterized Constructor with Name and Course :

Regno : 0

Name :Praveen

Class: MCA

The memory of the Object is Released

The memory of the Object is Released

The memory of the Object is Released

The memory of the Object is Released

Thank you



INHERITANCE

Dr P.V. Praveen Sundar
Assistant Professor,
Department of Computer Science
Adhiparasakthi College of Arts & Science,
Kalavai.

Inheritance

2

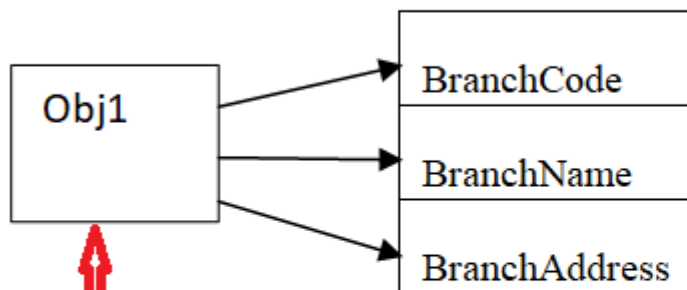
- ❑ The process of creating a new class from an existing class such that the new class acquires all the properties and behaviors of the existing class is called inheritance.
- ❑ The properties (or behaviors) are transferred from which class is called the superclass or parent class or base class whereas the class which derives the properties or behaviors from the superclass is known as a subclass or child class or derived class.
- ❑ Inheritance is the concept which is used for code reusability and changeability purpose. Here changeability means overriding the existed functionality or feature of the object or adding more functionality to the object.

- ❑ Following is the syntax of implementing an inheritance to define a derived class that inherits the base class's properties in the c# programming language.

```
<access_modifier> class <base_class_name>
{
    // Base class Implementation
}

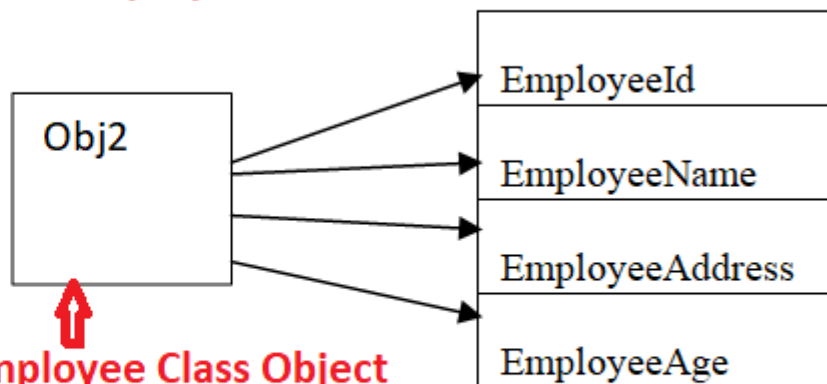
<access_modifier> class <derived_class_name> : <base_class_name>
{
    // Derived class implementation
}
```


Class Branch



Branch Class Object

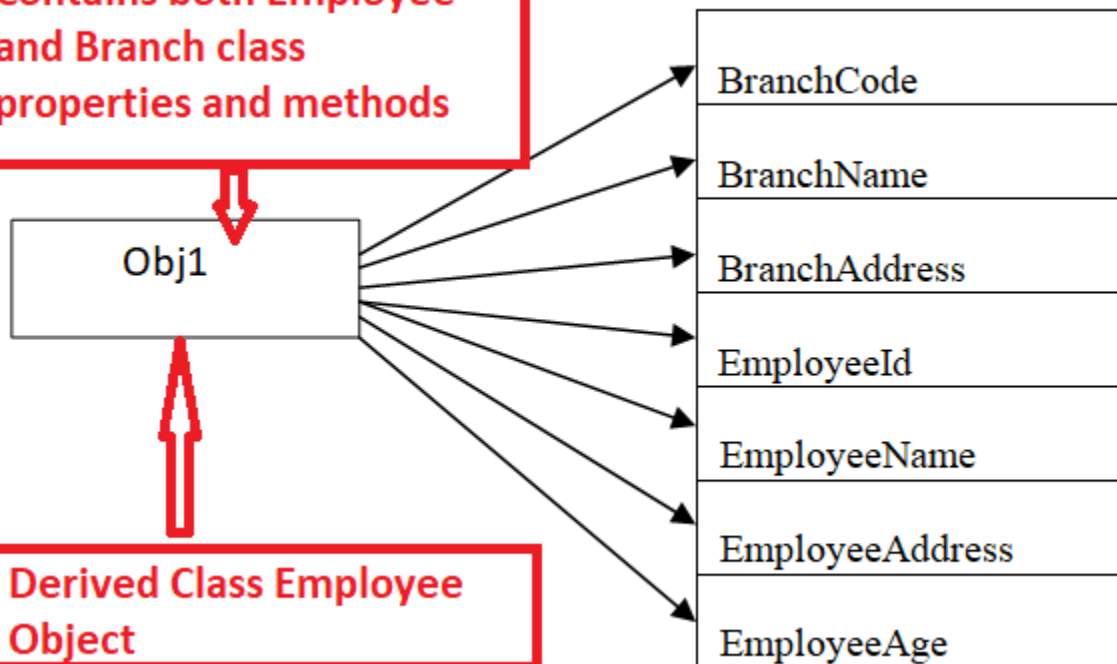
Class Employee



Employee Class Object

Without Inheritance

Contains both Employee and Branch class properties and methods



Derived Class Employee Object

Types of Inheritance

5

Inheritance is classified into 5 types. They are as follows.

- ▣ Single Inheritance
 - ▣ Hierarchical Inheritance
 - ▣ Multilevel Inheritance
 - ▣ Hybrid Inheritance
 - ▣ Multiple Inheritance
-
- **Single Inheritance:** When a class is derived from a single base class then the inheritance is called single inheritance.
 - **Hierarchical Inheritance:** Hierarchical inheritance is the inheritance where more than one derived class is created from a single base class.

- ❑ **Multilevel Inheritance:** When a derived class is created from another derived class, then that type of inheritance is called multilevel inheritance.
- ❑ **Hybrid Inheritance:** Hybrid Inheritance is the inheritance that is the combination of any single, hierarchical, and multilevel inheritances.
- ❑ **Multiple Inheritance:** When a derived class is created from more than one base class then such type of inheritance is called multiple inheritances. But multiple inheritances are not supported by .NET using classes and can be done using interfaces.
- ❑ Handling the complexity that causes due to multiple inheritances is very complex. Hence it was not supported in dot net with class and it can be done with interfaces.

- **Default Superclass:** Except Object class, which has no superclass, every class has one and only one direct superclass(single inheritance). In the absence of any other explicit superclass, every class is implicitly a subclass of Object class.
- **Superclass can only be one:** A superclass can have any number of subclasses. But a subclass can have only one superclass. This is because C# does not support multiple inheritance with classes. Although with interfaces, multiple inheritance is supported by C#.
- **Inheriting Constructors:** A subclass inherits all the members (fields, methods) from its superclass. Constructors are not members, so they are not inherited by subclasses, but the constructor of the superclass can be invoked from the subclass.
- **Private member inheritance:** A subclass does not inherit the private members of its parent class. However, if the superclass has properties(get and set methods) for accessing its private fields, then a subclass can inherit.

- ❑ In inheritance, the constructor of the parent class must be accessible to its child class otherwise the inheritance will not possible because when we create the child class object first it goes and calls the parent class constructor so that the parent class variable will be initialized and we can consume them under the child class.
- ❑ In inheritance, the child classes can consume the parent class members but the parent class does not consume child class members that are purely defined in the child class.

Class members Visibility

9

Table 13.1 *Visibility of class members*

<i>KEYWORD</i>	<i>VISIBILITY</i>			
	<i>CONTAINING CLASSES</i>	<i>DERIVED CLASSES</i>	<i>CONTAINING PROGRAM</i>	<i>ANYWHERE OUTSIDE THE CONTAINING PROGRAM</i>
Private	✓			
protected	✓	✓		
Internal	✓		✓	
protected internal	✓	✓	✓	
Public	✓	✓	✓	✓

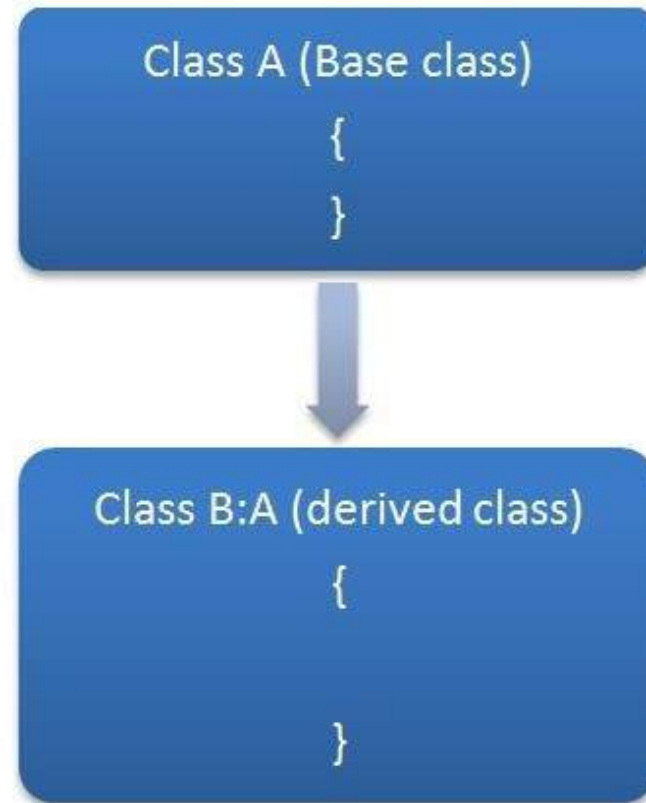
Table 13.2 *Accessibility domain of class members*

<i>MEMBER MODIFIER</i>	<i>MODIFIER OF THE CONTAINING CLASS</i>		
	<i>PUBLIC</i>	<i>INTERNAL</i>	<i>PRIVATE</i>
public	Everywhere	only program	only class
internal	only program	only program	only class
private	only class	only class	only class

Single Inheritance

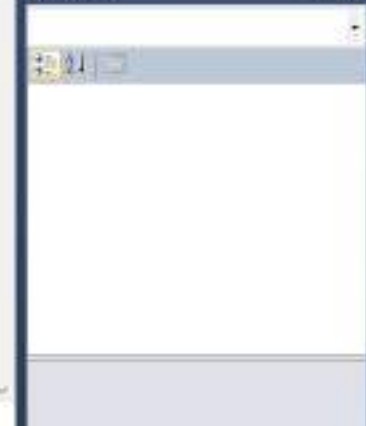
11

- ❑ Single inheritance enables a derived class to inherit properties and behavior from a single parent class.
- ❑ It allows a derived class to inherit the properties and behavior of a base class, thus enabling code reusability as well as adding new features to the existing code.
- ❑ This makes the code much more elegant and less repetitive.
- ❑ Single inheritance enables a derived class to call the parent class implementation for a specific method if this method is overridden in the derived class or the parent class constructor.





```
baseclass.cs*  
Inheritance.baseclass  
using System;  
  
namespace Inheritance  
{  
    class baseclass  
    {  
        int regno;  
        String Name;  
        String Course;  
        public baseclass()  
        {  
            Console.WriteLine(" Enter the Reg no:");  
            regno = Convert.ToInt16(Console.ReadLine());  
            Console.WriteLine(" Enter the Name:");  
            Name = Console.ReadLine();  
            Console.WriteLine(" Enter the Course :");  
            Course = Console.ReadLine();  
        }  
  
        public void Display()  
        {  
            Console.WriteLine("\n Name : {0}", Name);  
            Console.WriteLine("\n Reg No : {0}", regno);  
            Console.WriteLine("\n Course : {0}", Course);  
        }  
    }  
}
```



baseclass.cs*

Inheritance.derivedclass

Main(string[] args)

```
class derivedclass : baseclass
{
    int mark1, mark2, mark3, mark4, mark5, tot;

    private void Output()
    {
        Console.WriteLine("\n Subject 1 Marks : {0}", mark1);
        Console.WriteLine("\n Subject 2 Marks : {0}", mark2);
        Console.WriteLine("\n Subject 3 Marks : {0}", mark3);
        Console.WriteLine("\n Subject 4 Marks : {0}", mark4);
        Console.WriteLine("\n Subject 5 Marks : {0}", mark5);
        Console.WriteLine("\n Total= :{0}", tot);
    }

    public derivedclass()
    {
        Console.Write("\n Enter the Subject 1 marks:");
        mark1 = Convert.ToInt16(Console.ReadLine());
        Console.Write("\n Enter the Subject 2 marks:");
        mark2 = Convert.ToInt16(Console.ReadLine());
        Console.Write("\n Enter the Subject 3 marks:");
        mark3 = Convert.ToInt16(Console.ReadLine());
        Console.Write("\n Enter the Subject 4 marks:");
        mark4 = Convert.ToInt16(Console.ReadLine());
        Console.Write("\n Enter the Subject 5 marks:");
        mark5 = Convert.ToInt16(Console.ReadLine());
        tot = mark1 + mark2 + mark3 + mark4 + mark5;
    }

    static void Main(string[] args)
    {
        derivedclass obj1 = new derivedclass();
        obj1.Display();
        obj1.Output();
        Console.ReadKey();
    }
}
```

Enter the Reg no:21312

Enter the Name:Praveen

Enter the Course :BCA

Enter the Subject 1 marks:45

Enter the Subject 2 marks:55

Enter the Subject 3 marks:50

Enter the Subject 4 marks:60

Enter the Subject 5 marks:65

Name : Praveen

Reg No : 21312

Course : BCA

Subject 1 Marks : 45

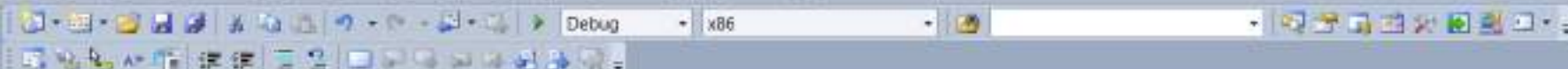
Subject 2 Marks : 55

Subject 3 Marks : 50

Subject 4 Marks : 60

Subject 5 Marks : 65

Total= :275



Singleinherit.cs baseclass.cs

Inheritance.Stud_class

regno

```
using System;

namespace Inheritance
{
    class Stud_class
    {
        int regno;
        String Name;
        String Course;
        public Stud_class(int no, String name, String course)
        {
            this.regno= no;
            this.Name= name;
            this.Course= course;
        }
        public void Display()
        {
            Console.WriteLine("\n Name : {0}", Name);
            Console.WriteLine("\n Reg No : {0}", regno);
            Console.WriteLine("\n Course : {0}", Course);
        }
    }
    class Singleinherit : Stud_class
    {
        int mark1, mark2, mark3, mark4, mark5, tot;

        public Singleinherit( int no, String name,String course, int m1,int m2,int m3,int m4, int m5): base(no, name, course)
        {
            this.mark1 = m1;
            this.mark2 = m2;
            this.mark3 = m3;
            this.mark4 = m4;
            this.mark5 = m5;
            this.tot = m1 + m2 + m3 + m4 + m5;
        }
    }
}
```

Solution Explorer

Solution 'Inheritance' (1 project)

- Inheritance
 - Properties
 - References
 - baseclass.cs
 - Singleinherit.cs

Solut... Team... Class...

Properties

Output

Item(s) Saved

Ln 1

Col 14

Ch 14

INS

```
}  
public void Output()  
{  
    Display();  
    Console.WriteLine("\n Subject 1 Marks : {0}", mark1);  
    Console.WriteLine("\n Subject 2 Marks : {0}", mark2);  
    Console.WriteLine("\n Subject 3 Marks : {0}", mark3);  
    Console.WriteLine("\n Subject 4 Marks : {0}", mark4);  
    Console.WriteLine("\n Subject 5 Marks : {0}", mark5);  
    Console.WriteLine("\n Total= :{0}", tot);  
}  
  
public static void Main()  
{  
    int rno, m1, m2, m3, m4, m5;  
    String name, course;  
    Console.Write(" Enter the Reg no:");  
    rno = Convert.ToInt16(Console.ReadLine());  
    Console.Write(" Enter the Name:");  
    name = Console.ReadLine();  
    Console.Write(" Enter the Course :");  
    course = Console.ReadLine();  
    Console.Write("\n Enter the Subject 1 marks:");  
    m1 = Convert.ToInt16(Console.ReadLine());  
    Console.Write("\n Enter the Subject 2 marks:");  
    m2 = Convert.ToInt16(Console.ReadLine());  
    Console.Write("\n Enter the Subject 3 marks:");  
    m3 = Convert.ToInt16(Console.ReadLine());  
    Console.Write("\n Enter the Subject 4 marks:");  
    m4 = Convert.ToInt16(Console.ReadLine());  
    Console.Write("\n Enter the Subject 5 marks:");  
    m5 = Convert.ToInt16(Console.ReadLine());  
  
    SingleInherit obj1 = new SingleInherit(rno, name, course, m1, m2, m3, m4, m5);  
    obj1.Output();  
    Console.ReadKey();  
}
```

Enter the Course :BCA

Enter the Subject 1 marks:25

Enter the Subject 2 marks:35

Enter the Subject 3 marks:45

Enter the Subject 4 marks:55

Enter the Subject 5 marks:65

Name : Rajesh

Reg No : 21312

Course : BCA

Subject 1 Marks : 25

Subject 2 Marks : 35

Subject 3 Marks : 45

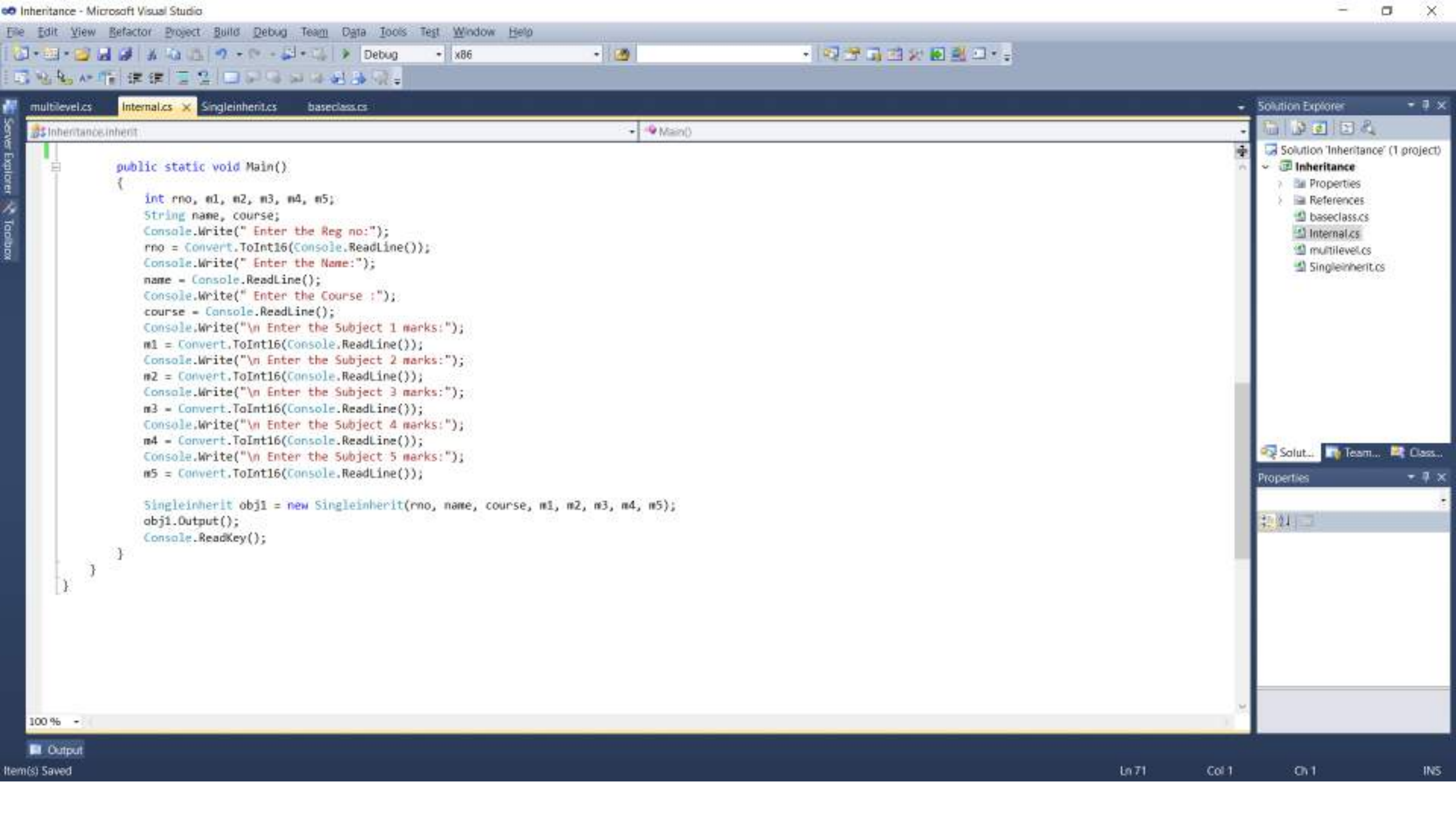
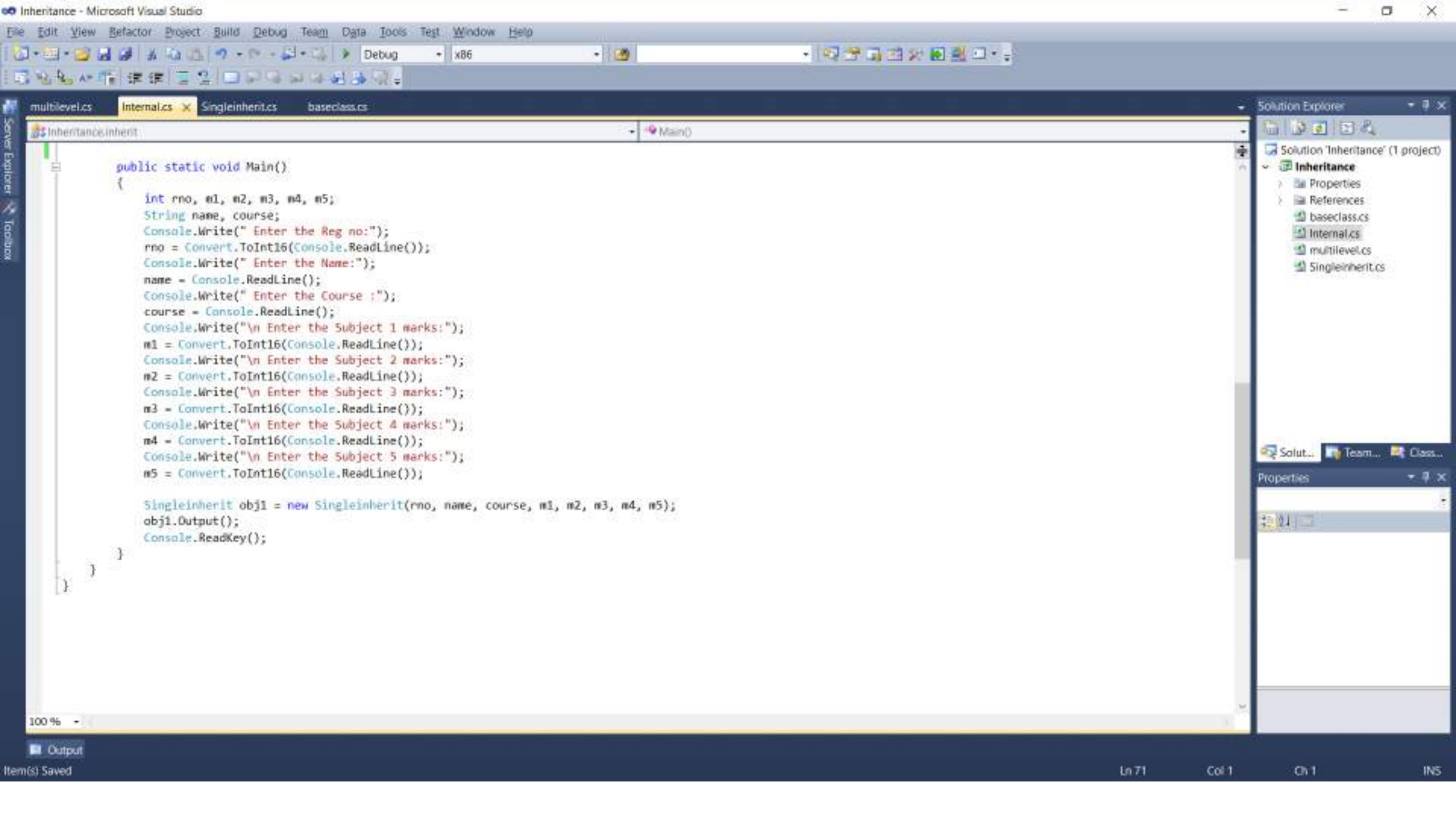
Subject 4 Marks : 55

Subject 5 Marks : 65

Total= :225


```
using System;
namespace Inheritance
{
    class Internal
    {
        internal int regno;
        internal String Name;
        internal String Course;
        public Internal(int no, String name, String course)
        {
            this.regno = no;
            this.Name = name;
            this.Course = course;
        }
    }
    class inherit : Internal
    {
        int mark1, mark2, mark3, mark4, mark5, tot;

        public inherit(int no, String name, String course, int n1, int n2, int n3, int n4, int n5)
            : base(no, name, course)
        {
            this.mark1 = n1;
            this.mark2 = n2;
            this.mark3 = n3;
            this.mark4 = n4;
            this.mark5 = n5;
            this.tot = n1 + n2 + n3 + n4 + n5;
        }
        public void Output()
        {
            Console.WriteLine("\n Name : {0}", Name);
            Console.WriteLine("\n Reg No : {0}", regno);
            Console.WriteLine("\n Course : {0}", Course);
            Console.WriteLine("\n Subject 1 Marks : {0}", mark1);
            Console.WriteLine("\n Subject 2 Marks : {0}", mark2);
            Console.WriteLine("\n Subject 3 Marks : {0}", mark3);
            Console.WriteLine("\n Subject 4 Marks : {0}", mark4);
            Console.WriteLine("\n Subject 5 Marks : {0}", mark5);
            Console.WriteLine("\n Total= :{0}", tot);
        }
    }
}
```

Enter the Course :BCA

Enter the Subject 1 marks:56

Enter the Subject 2 marks:67

Enter the Subject 3 marks:78

Enter the Subject 4 marks:89

Enter the Subject 5 marks:90

Name : Rajesh

Reg No : 21312

Course : BCA

Subject 1 Marks : 56

Subject 2 Marks : 67

Subject 3 Marks : 78

Subject 4 Marks : 89

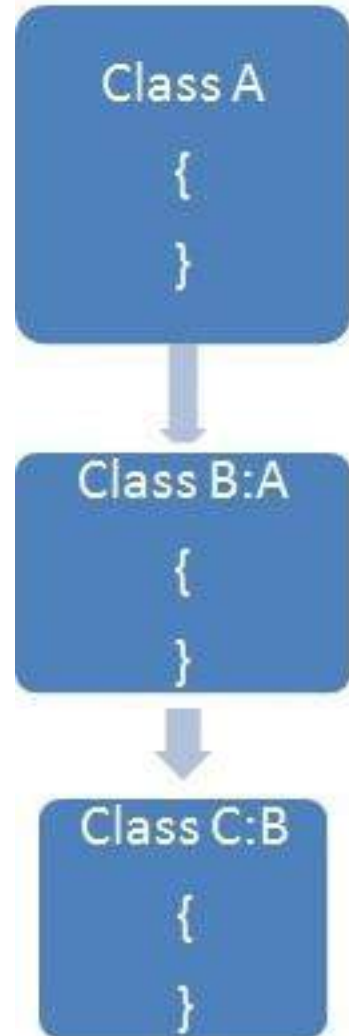
Subject 5 Marks : 90

Total= :380

Multilevel Inheritance

22

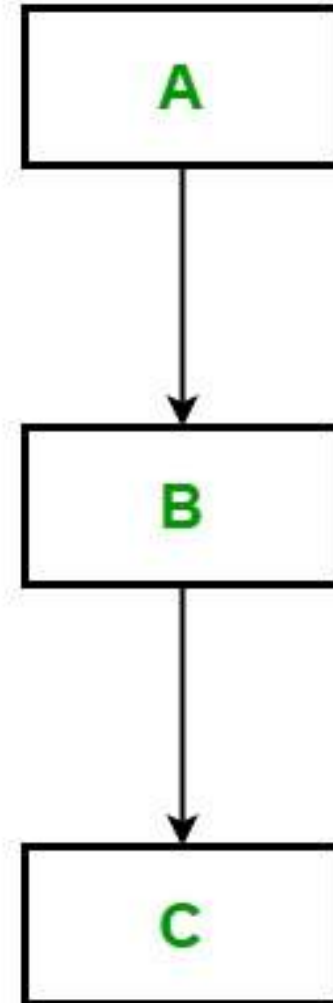
- ❑ In the Multilevel inheritance, a derived class will inherit a base class and as well as the derived class also act as the base class to other class.
- ❑ For example, three classes called A, B, and C, as shown in the below image, where class C is derived from class B and class B, is derived from class A.
- ❑ In this situation, each derived class inherit all the characteristics of its base classes. So class C inherits all the features of class A and B.
- ❑ In multilevel inheritance the level of inheritance can be extended to any number of level depending upon the relation.
- ❑ Multilevel inheritance is similar to relation between grandfather, father and child.



Base class

Intermediary
class

Derived class



```
using System;
namespace Inheritance
{
    class multilevel : Singleinherit
    {
        String Result;
        public multilevel(int no, String name, String course, int m1, int m2, int m3, int m4, int m5)
            : base(no, name, course, m1, m2, m3, m4, m5)
        {
            Output();
            if (m1 >= 35 && m2 >= 35 && m3 >= 35 && m4 >= 35 && m5 > 35)
                Result = "Pass";
            else
                Result = "Fail";
            Console.WriteLine("Result :{0}", Result);
        }
        public static void Main()
        {
            int rno, m1, m2, m3, m4, m5;
            String name, course;
            Console.Write(" Enter the Reg no:");
            rno = Convert.ToInt16(Console.ReadLine());
            Console.Write(" Enter the Name:");
            name = Console.ReadLine();
            Console.Write(" Enter the Course :");
            course = Console.ReadLine();
            Console.Write("\n Enter the Subject 1 marks:");
            m1 = Convert.ToInt16(Console.ReadLine());
            Console.Write("\n Enter the Subject 2 marks:");
            m2 = Convert.ToInt16(Console.ReadLine());
            Console.Write("\n Enter the Subject 3 marks:");
            m3 = Convert.ToInt16(Console.ReadLine());
            Console.Write("\n Enter the Subject 4 marks:");
            m4 = Convert.ToInt16(Console.ReadLine());
            Console.Write("\n Enter the Subject 5 marks:");
            m5 = Convert.ToInt16(Console.ReadLine());

            multilevel obj1 = new multilevel(rno, name, course, m1, m2, m3, m4, m5);
            Console.ReadKey();
        }
    }
}
```

Enter the Reg no:21312

Enter the Name:Vijay

Enter the Course :BCA

Enter the Subject 1 marks:87

Enter the Subject 2 marks:76

Enter the Subject 3 marks:75

Enter the Subject 4 marks:83

Enter the Subject 5 marks:64

Name : Vijay

Reg No : 21312

Course : BCA

Subject 1 Marks : 87

Subject 2 Marks : 76

Subject 3 Marks : 75

Subject 4 Marks : 83

Subject 5 Marks : 64

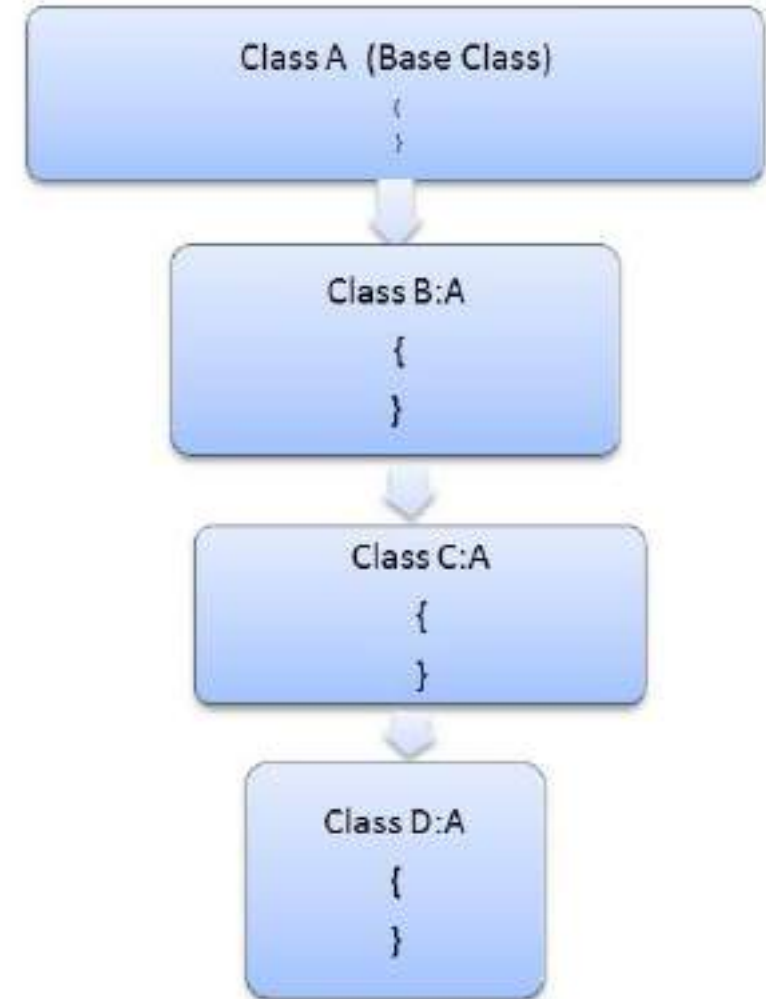
Total= :385

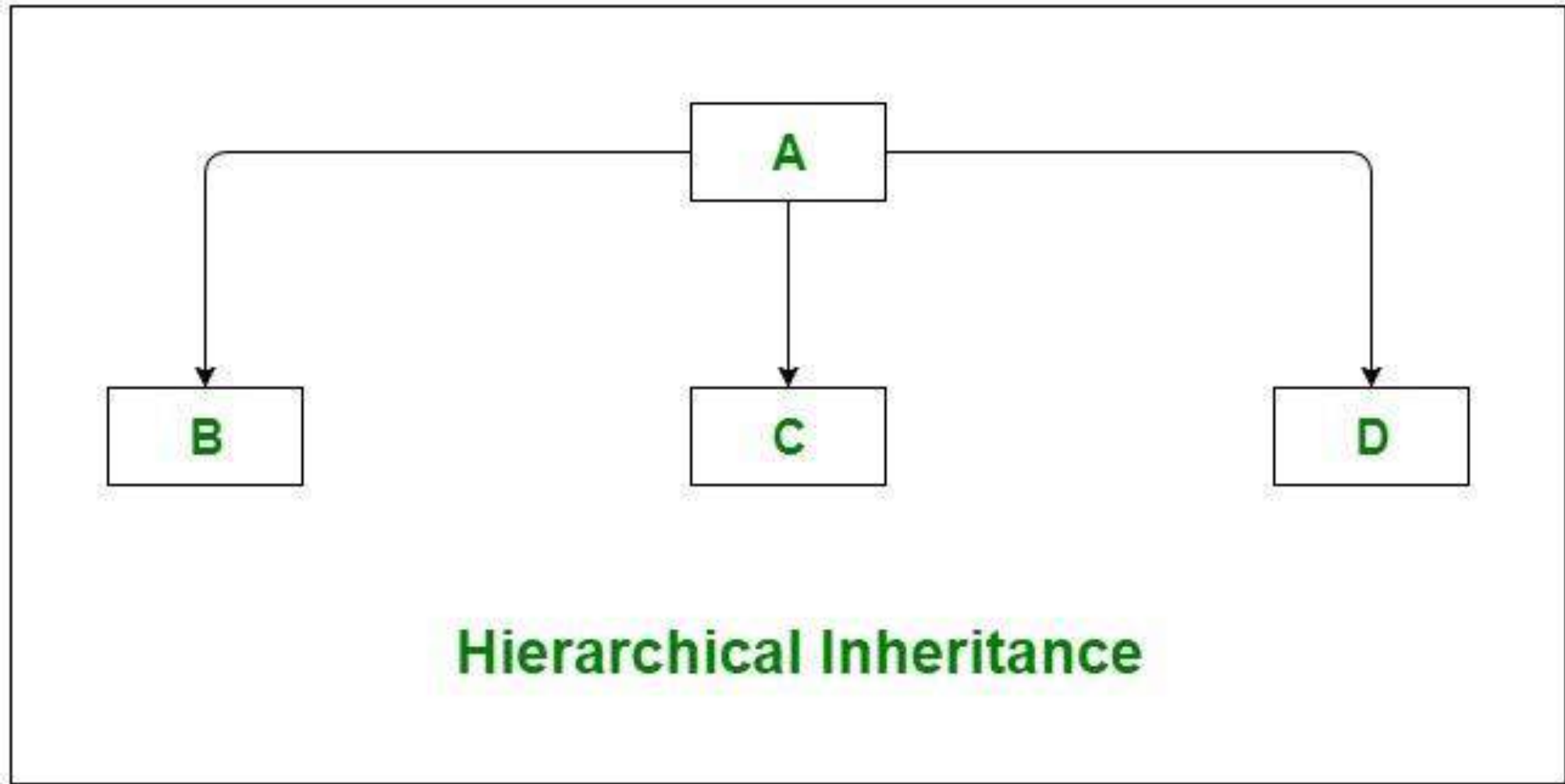
Result :Pass

Hierarchical inheritance

26

- ❑ Hierarchical inheritance involves multiple classes inheriting from a single base class.
- ❑ This is quite useful if the features of the base class are required in multiple classes.






```
using System;
namespace Inheritance
{
    class Studentdetails: baseclass
    {
        long aadharNo;
        String fathername;
        String mothername;
        String Address;

        public Studentdetails()
        {
            Console.Write(" Enter the Aadhar no");
            aadharNo = Convert.ToInt64(Console.ReadLine());
            Console.Write(" Enter the Father Name:");
            fathername = Console.ReadLine();
            Console.Write(" Enter the Mother Name:");
            mothername = Console.ReadLine();
            Console.Write(" Enter the Address:");
            Address = Console.ReadLine();
        }

        public void Output()
        {
            Display();
            Console.WriteLine("\n Aadhar no : {0}", aadharNo);
            Console.WriteLine("\n Father Name : {0}", fathername);
            Console.WriteLine("\n Mother Name : {0}", mothername);
        }
    }
}
```

```
class marks : baseclass
{
    int mark1, mark2, mark3, mark4, mark5, tot;
    public marks()
    {
        Console.WriteLine("\n Enter the Subject 1 marks:");
        mark1 = Convert.ToInt16(Console.ReadLine());
        Console.WriteLine("\n Enter the Subject 2 marks:");
        mark2 = Convert.ToInt16(Console.ReadLine());
        Console.WriteLine("\n Enter the Subject 3 marks:");
        mark3 = Convert.ToInt16(Console.ReadLine());
        Console.WriteLine("\n Enter the Subject 4 marks:");
        mark4 = Convert.ToInt16(Console.ReadLine());
        Console.WriteLine("\n Enter the Subject 5 marks:");
        mark5 = Convert.ToInt16(Console.ReadLine());
        tot = mark1 + mark2 + mark3 + mark4 + mark5;
    }
    public void Output()
    {
        Display();
        Console.WriteLine("\n Subject 1 Marks : {0}", mark1);
        Console.WriteLine("\n Subject 2 Marks : {0}", mark2);
        Console.WriteLine("\n Subject 3 Marks : {0}", mark3);
        Console.WriteLine("\n Subject 4 Marks : {0}", mark4);
        Console.WriteLine("\n Subject 5 Marks : {0}", mark5);
        tot = mark1 + mark2 + mark3 + mark4 + mark5;
        Console.WriteLine("\n Total= :{0}", tot);
    }
}

class Hierarchical
{
    public static void Main()
    {
        marks M = new marks();
        M.Output();
        Studentdetails SD = new Studentdetails();
        SD.Output();
        Console.ReadKey();
    }
}
```

Enter the Reg no:21312

Enter the Name:Kumar

Enter the Course :BCA

Enter the Subject 1 marks:87

Enter the Subject 2 marks:76

Enter the Subject 3 marks:65

Enter the Subject 4 marks:54

Enter the Subject 5 marks:43

Name : Kumar

Reg No : 21312

Course : BCA

Subject 1 Marks : 87

Subject 2 Marks : 76

Subject 3 Marks : 65

Subject 4 Marks : 54

Subject 5 Marks : 43

Total= :325

Enter the Reg no:21312
Enter the Name:Kumar
Enter the Course :BCA
Enter the Aadhar no:98867657
Enter the Father Name:Suresh
Enter the Mother Name:baneerji
Enter the Address:Kolkatta

Name : Kumar

Reg No : 21312

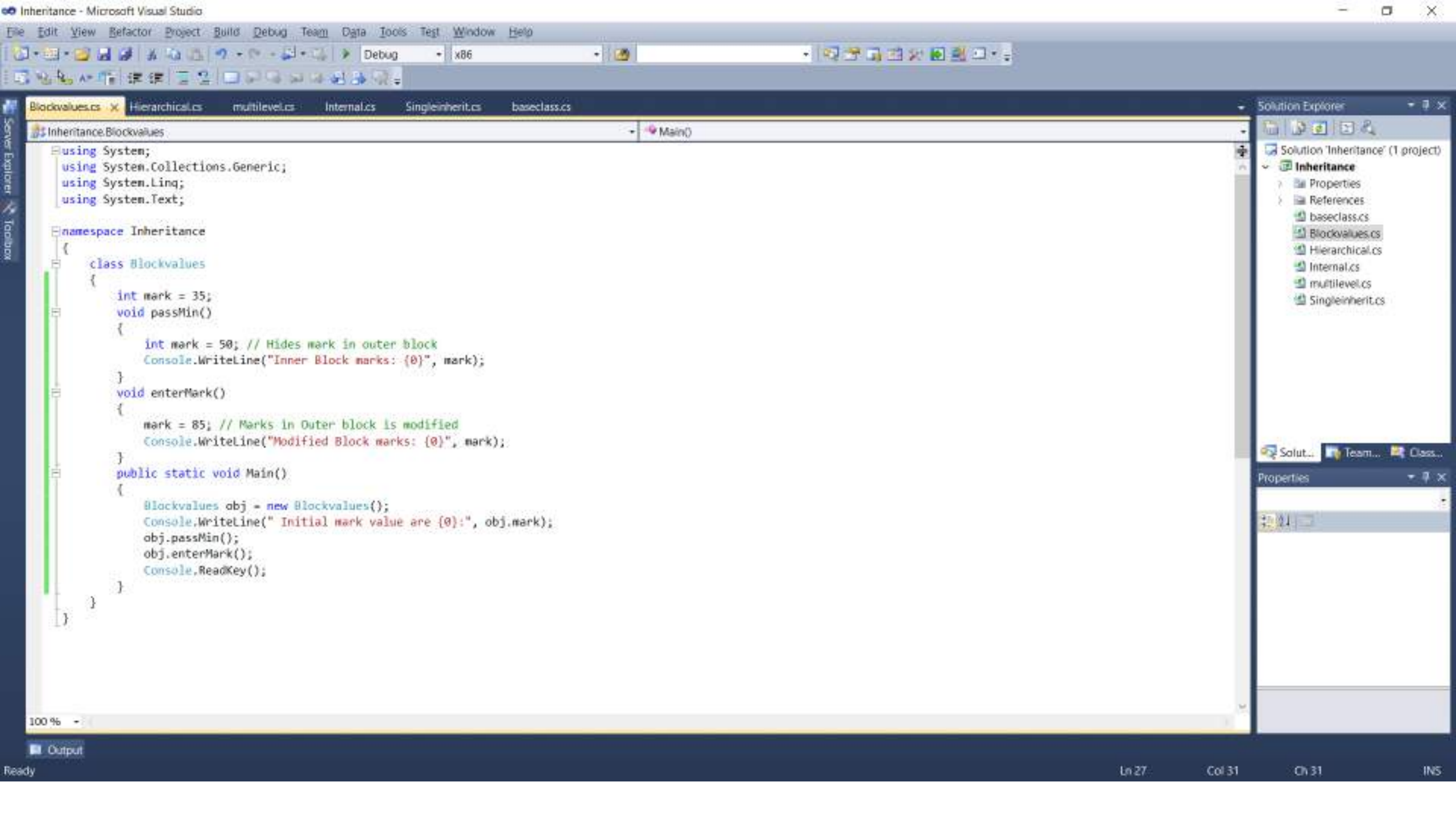
Course : BCA

Aadhar no : 98867657

Father Name : Suresh

Mother Name : baneerji

Address : Kolkatta



file:///C:/Users/Dr.PV.Praveen Sundar/documents/visual studio 2010/Projects/Inheritance/Inheritance/bin/Debug/Inheritance.EXE

Initial mark value are 35:
Inner Block marks: 50
Modified Block marks: 85

Nested Class

34

- ❑ When a class is declared within another class, the inner class is called as Nested class (i.e. the inner class) and the outer class is known as Enclosing class.
- ❑ Nested class can be defined in private as well as in the public section of the Enclosing class.
- ❑ The inner class can act as a helper class to serve the outer class.
- ❑ A method in the inner class can access all members including private members of its outer class.
- ❑ A public inner class is accessed within the scope of the outer class.
- ❑ The members in an inner class hide the members having the same name in its outer class. Thus name hiding is possible by nesting blocks or nesting classes.

- ❑ A nested class can be declared as a private, public, protected, internal, protected internal, or private protected.
- ❑ Outer class is not allowed to access inner class members directly.
- ❑ It is possible to create objects of inner class in outer class.
- ❑ Inner class can access static member declared in outer class.
- ❑ Inner class can access non-static member declared in outer class


```
{
    class Outerclass {
        int empid;
        String Name;
        String Designation;
        Outerclass(int empid,String name, String Designation)
        {
            this.empid = empid;
            this.Name = name;
            this.Designation = Designation;
        }
        class InnerClass {
            long salary;
            public InnerClass()
            {
                Console.WriteLine(" Enter the Salary:");
                salary = Convert.ToInt64(Console.ReadLine());
            }
            public void Display(Outerclass obj)
            {
                Console.WriteLine(" Employee id : {0}",obj.empid);
                Console.WriteLine(" Employee Name : {0}", obj.Name);
                Console.WriteLine(" Designation : {0}", obj.Designation);
                Console.WriteLine("Salary : {0}", salary);
            }
        }
    }
    public static void Main()
    {
        int empid;
        String Name, Designation;
        Console.WriteLine(" Enter the Employee id:");
        empid = Convert.ToInt32(Console.ReadLine());
        Console.WriteLine(" Enter the Employee Name");
        Name = Console.ReadLine();
        Console.WriteLine(" Enter the Employee Designation:");
        Designation = Console.ReadLine();
        Outerclass obj = new Outerclass(empid, Name, Designation);
        InnerClass obj1 = new InnerClass();
        obj1.Display(obj);
        Console.ReadKey();
    }
}
```

```
Enter the Employee id:21312
Enter the Employee NameSuresh
Enter the Employee Designation:Doctor
Enter the Salary:20000
Employee id : 21312
Employee Name : Suresh
Designation : Doctor
Salary : 20000
```

```
using System;

namespace Inheritance
{
    class NestedClass
    {
        static int empid;
        static String Name;
        static String Designation;

        public class InnerClass
        {
            long salary;
            public InnerClass()
            {
                Console.Write(" Enter the Employee id:");
                empid = Convert.ToInt32(Console.ReadLine());
                Console.Write(" Enter the Employee Name");
                Name = Console.ReadLine();
                Console.Write(" Enter the Employee Designation:");
                Designation = Console.ReadLine();
                Console.Write(" Enter the Salary:");
                salary = Convert.ToInt64(Console.ReadLine());
            }
            public void Display()
            {
                Console.WriteLine(" Employee id : {0}", NestedClass.empid);
                Console.WriteLine(" Employee Name : {0}", NestedClass.Name);
                Console.WriteLine(" Designation : {0}", NestedClass.Designation);
                Console.WriteLine("Salary : {0}", salary);
            }
        }
        public static void Main()
        {
            InnerClass obj1 = new InnerClass();
            obj1.Display();
            Console.ReadKey();
        }
    }
}
```

file:///C:/Users/Dr.PV.Praveen Sundar/documents/visual studio 2010/Projects/Inheritance/Inheritance/bin/Debug/Inheritance.EXE

Enter the Employee id:21312
Enter the Employee NameRamesh
Enter the Employee Designation:Police
Enter the Salary:50000
Employee id : 21312
Employee Name : Ramesh
Designation : Police
Salary : 50000

Constant Members

40

- ❑ Constants are immutable values which are known at compile time and do not change for the life of the program.
- ❑ Constants are declared with the `const` modifier.
- ❑ Only the C# built-in types can be declared as `const`.
- ❑ User-defined types, including classes, structs, and arrays, cannot be declared as `const`.
- ❑ C# does not support `const` methods, properties, or events.
- ❑ It's mandatory to initialize constant fields with required values during the declaration itself; otherwise, we will get compile-time errors in our C # application.
- ❑ Following is the syntax of defining constant fields using `const` keyword in c# programming language.

`const data_type field_name = "value";`

- The following are the different ways of declaring and initializing constant variables in the c# programming language.

// Constant variables

```
const string name = "Praveen Sundar";
```

```
const string location = "Vellore";
```

```
const int age = 18;
```

```
public const int Months = 12, Weeks = 52, Days = 365;
```

```
public const int Months = 12;
```

```
public const int Weeks = 52;
```

```
public const int Days = 365;
```

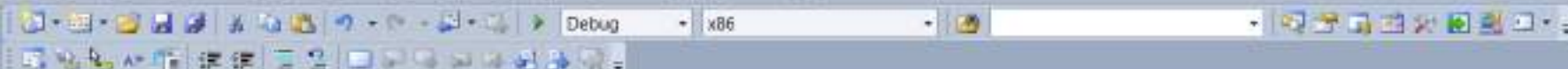
```
public const double DaysPerWeek = (double) Days / (double) Weeks;
```

```
public const double DaysPerMonth = (double) Days / (double) Months;
```

Read Only Members

42

- ❑ The variable which is declared by using the readonly keyword is known as a read-only variable.
- ❑ The read-only variable's value cannot be modified once after its initialization.
- ❑ It is not mandatory or required to initialize the read-only variable at the time of its declaration like a constant. You can initialize the read-only variables under a constructor but the most important point is that once after initialization, you cannot modify the value.
- ❑ The behavior of a read-only variable is similar to the behavior of a non-static variable. That is, it maintains a separate copy for each object. The only difference between these two is non-static variables can be modified while the read-only variables cannot be modified.
- ❑ A constant variable is a fixed value for the complete class whereas a read-only variable is a fixed value but specific to one object of the class.



Constantegcs.cs

InheritanceConstantegcs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Inheritance
{
    class Constantegcs
    {
        const float PI = 3.1414f;
        readonly public float ValuePI = 300.1414f;
        Constantegcs()
        {
            ValuePI = 3.1414f;
        }
        public static void Main()
        {
            Constantegcs obj1 = new Constantegcs();
            float radius, result;
            Console.WriteLine("Enter the radius value :");
            radius = Convert.ToSingle(Console.ReadLine());
            result = PI * radius * radius;
            Console.WriteLine("Result= {0}", result);
            result = obj1.ValuePI * radius * radius;

            Console.WriteLine("Result= {0}", result);
            Console.ReadKey();
        }
    }
}
```

Solution Explorer

- abstractclass.cs
- Autoproperty.cs
- baseclass.cs
- Blockvalues.cs
- Constantegcs.cs
- Genericclass.cs
- genericindexer.cs
- Hierarchical.cs
- Indexereg.cs
- Internal.cs
- multilevel.cs
- NestedClass.cs
- Outerclass.cs
- OverloadedIndexer.cs
- Propertieseg.cs
- Propertiesvalid.cs
- Singleinherit.cs

Properties

file:///C:/Users/Dr.PV.Praveen Sundar/documents/visual studio 2010/Projects/Inheritance/Inheritance/bin/Debug/Inheritance.EXE

Enter the radius value :5
Result= 78.535
Result= 78.535

Properties

45

- ❑ In order to encapsulate and protect the data members (i.e. fields), we use properties in C#.
- ❑ The Properties in C# are used as a mechanism to set and get the values of a class outside of that class.
- ❑ If a class contains any value in it and if we want to access those values outside of that class, then you can provide access to those values in two different ways
 - ❑ By storing the value under a public variable we can give access to the value outside of the class.
 - ❑ By storing that value in a private variable we can also give access to that value outside of the class by defining a property for that variable.
- ❑ A property in C# is a member of a class which is used to set and get the data from a data field of a class.

- ❑ Whenever we create a property, the data type of the property must be the same as the data type of the data field for which we create the property.
- ❑ A property can never accept any arguments.
- ❑ The most important point that you need to remember is. a property in C# is never used to store data, it just acts as an interface to transfer the data.
- ❑ We use the Properties as they are the public data members of a class, but they are actually special methods called accessors.
- ❑ The Assessors are nothing but special methods which are used to set and get the values from the underlying data member. Assessors are of two types such as
 - ▣ set accessor
 - ▣ get accessor
- ❑ The set accessor is used to set the data (i.e. value) into a data field. This set accessor contains a fixed variable named “value”.

- Whenever we call the property to set the data, whatever data (value) we are supplying that will come and store in the variable “value” by default.

Syntax:

`set { Data Field Name = value; }`

- The get accessor is used to get the data from the data field. Using this get accessor you cannot set the data.

Syntax:

`get { return Data Field Name; }`

- The default accessibility modifier of the accessor is same as the accessibility modifier of property.
- If the accessibility modifier of the accessors (both get and set) are the same within a property then the accessors are known as Symmetric accessors.

- On the other hand, if the accessibility modifier of the accessors is not the same within a property then the accessors are known as Asymmetric accessors.

For example:

```
public int empid
{
    protected set { _empid = value; }
    get { return _empid; }
}
```

- The C#.NET supports four types of properties. They are as follows
 - ▣ Read-only property
 - ▣ Write only property
 - ▣ Read Write property
 - ▣ Auto-implemented property

- The **Read-only property** is used to read the data from the data field.
- Using this property you cannot set the data into the data field. This property will contain only one accessor i.e. “get” accessor.

Syntax:

```
AccessModifier Datatype PropertyName { get { return DataFieldName; } }
```

- The **Write-only property** is used to write the data into the data field of a class.
- Using this property you cannot read the data from the data field. This property will contain only one accessor i.e. set accessor.

Syntax:

```
AccessModifier Datatype PropertyName { set { DataFieldName = value; } }
```

- The **Read-Write property** is used for both read the data from the data field as well as write the data into the data field. This property will contain two accessor i.e. set and get.

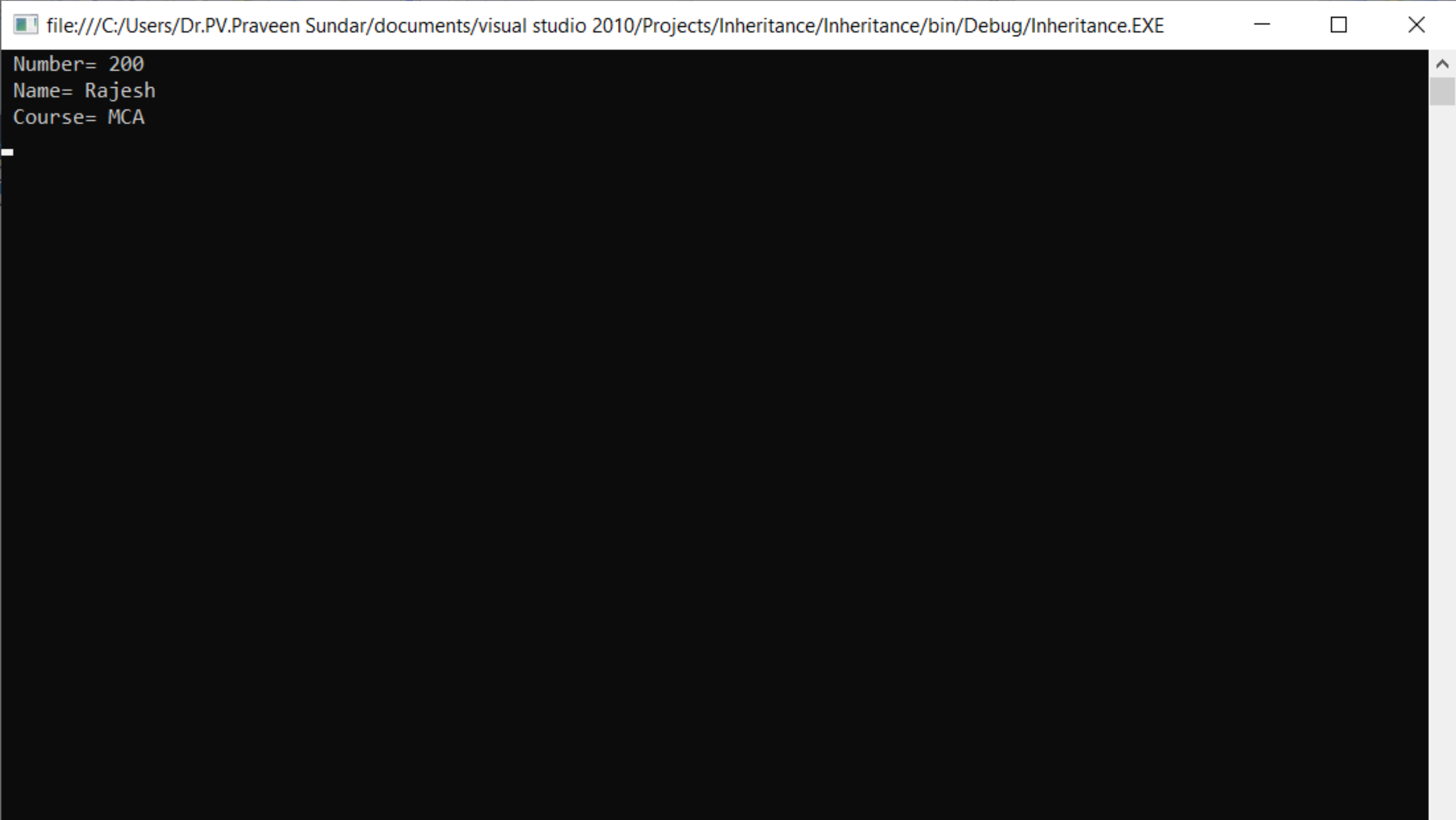
Syntax:

AccessModifier DataType PropertyName

```
{  
    set  
        {  
            DataFieldName = value;  
        }  
    get  
        {  
            Return DataFieldName;  
        }  
}
```

```
using System;

namespace Inheritance
{
    class Propertieseg
    {
        private int regnumber;
        private String name;
        private String course;
        public int Number
        {
            get
            { return regnumber; }
            set
            { regnumber= value; }
        }
        public String Name
        {
            get
            { return name; }
        }
        public String Degree
        {
            set
            { course = value; }
        }
        public static void Main()
        {
            Propertieseg obj1 = new Propertieseg();
            obj1.Number = 200;
            obj1.name = "Rajesh";
            obj1.Degree = "MCA"; // Write only Property
            Console.WriteLine(" Number= {0}", obj1.Number); // Read-Write Property
            Console.WriteLine(" Name= {0}", obj1.Name); // Read only Property
            Console.WriteLine(" Course= {0}", obj1.course);
            Console.ReadKey();
        }
    }
}
```

Number= 200
Name= Rajesh
Course= MCA

Auto-Implemented Properties in C#

53

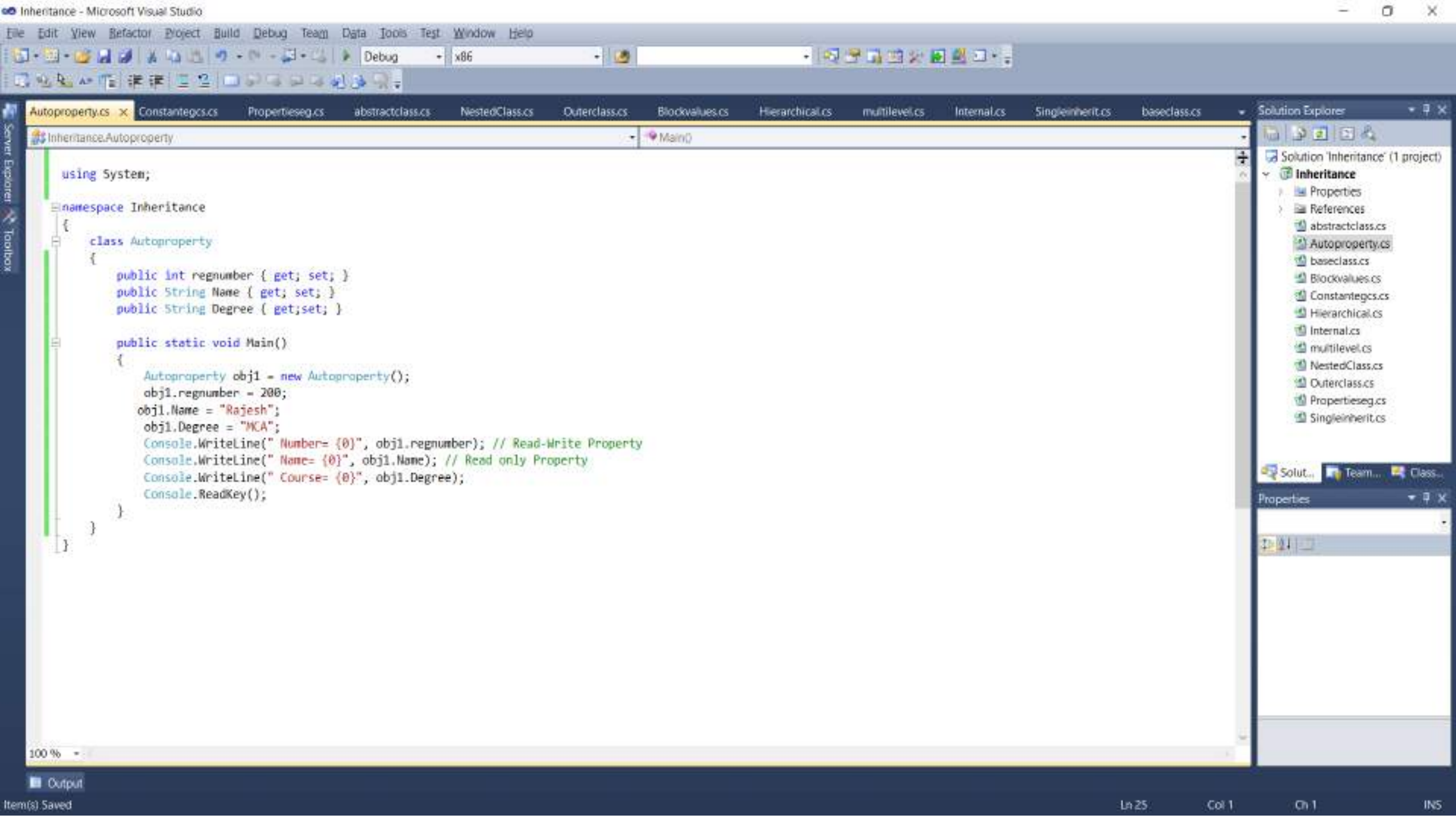
- If we do not have any additional logic while setting and getting the data from a data field then we can make use of the auto-implemented properties which was introduced in C# 3.0
- The Auto-implemented property reduces the amount of code that we have to write.
- When we use auto-implemented properties, the C# compiler implicitly creates a private, anonymous field behind the scene which is going to hold the data.

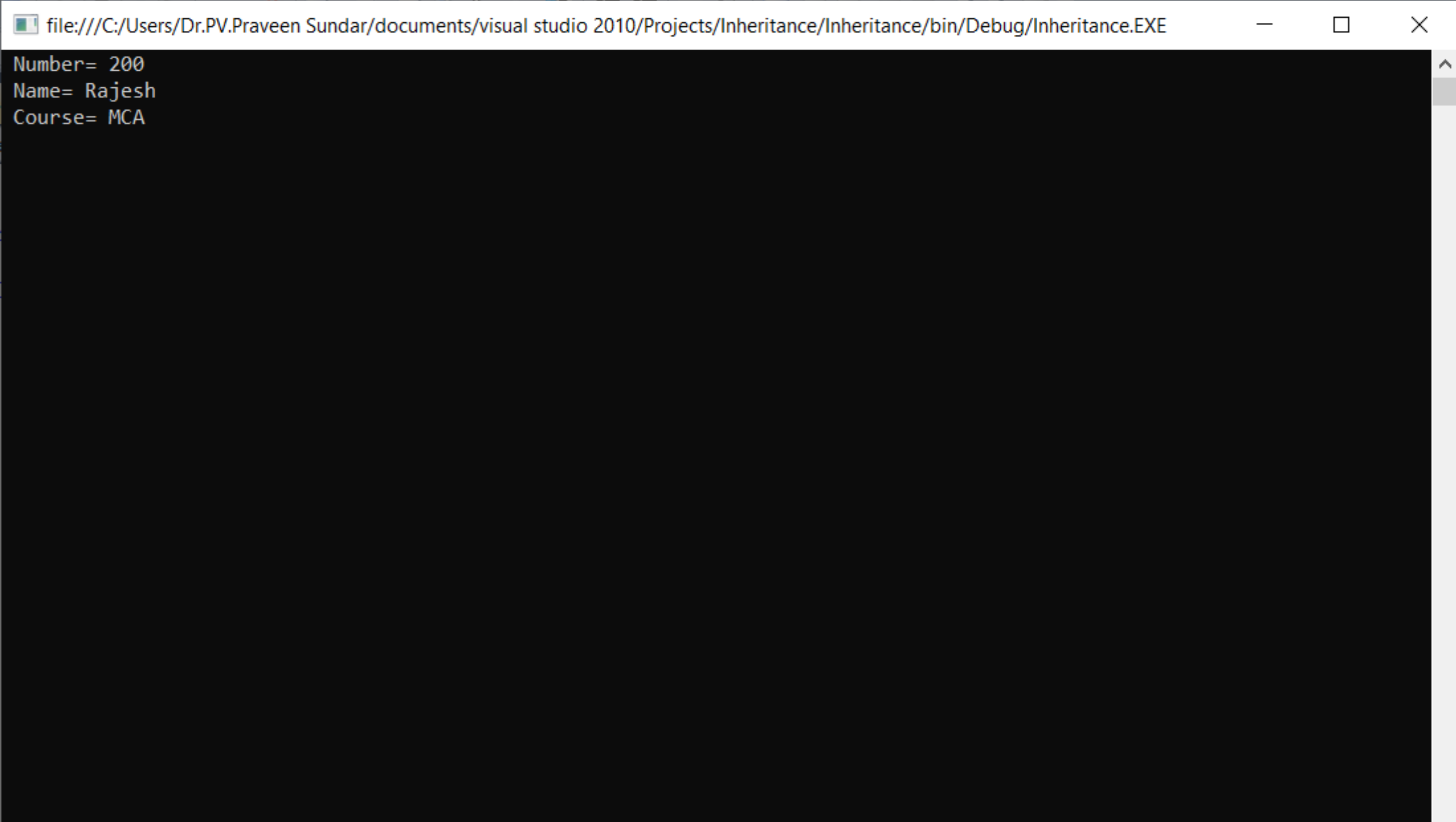
Syntax:

Access specifier Datatype Property name

```
{  
    get;  
    set;  
}
```

- Example: `public int A { Get; Set; }`





Number= 200
Name= Rajesh
Course= MCA

Need of properties in real-time applications

56

- A good design of a class is achieved by hiding implementation details of the methods and preventing direct access to data fields. Abstract classes and interfaces help in hiding implementations. Preventing direct access to the fields is achieved by making the fields private.
- Only by making the fields private the benefits of data integrity do not come automatically.
- The programmer must provide validity checking. Methods that set the values of private data should verify whether the input values are valid or not. If the values are not proper, the set method may provide an appropriate value.
- Properties appear to the outside world as fields, but allow processing when their values are read or modified. They are usually used to modify the behavior at runtime.
- Even though a public set accessor seems to allow other methods to read the data at will, it is possible to control a new value appropriate for the typical application. The access is restricted by proper implementation of the accessors by the programmers. Thus the benefits of data integrity is obtained by providing validity checking in the accessor methods of properties.

```
using System;
namespace Inheritance
{
    class Student
    {
        private int id;
        private string _name;
        private int _passMark = 35;
        public int ID
        {
            set
            {
                if (value < 0)
                {
                    value = 0;
                    Console.WriteLine("ID value should be greater than zero");
                }
                this.id = value;
            }
            get
            {
                return this.id;
            }
        }
        public string Name
        {
            set
            {
                if (string.IsNullOrEmpty(value))
                {
                    Console.WriteLine("Name should not be empty");
                }
                this._name = value;
            }
            get
            {
                return string.IsNullOrEmpty(this._name) ? "No Name" : this._name;
            }
        }
    }
}
```

```
public int PassMark
{
    get
    {
        return this._passMark;
    }
    set
    {
        if (value < 0 || value > 100)
        {
            value = 0;
            Console.WriteLine(" Invalid Marks");
        }
        this._passMark = value;
    }
}

public void Display()
{
    Console.WriteLine("Student ID = {0}", ID);
    Console.WriteLine("Student Name = {0}", Name);
    Console.WriteLine("Student Pass Mark = {0}", PassMark);
}

public void Input()
{
    Console.Write(" Enter the Valid Student ID:");
    ID = Convert.ToInt32(Console.ReadLine());
    Console.Write(" Enter the Valid Name :");
    Name = Console.ReadLine();
    Console.Write(" Enter the Valid Marks :");
    PassMark = Convert.ToInt32(Console.ReadLine());
}

public static void Main(string[] args)
{
    Student S = new Student();
    S.Input();
    S.Display();
    Console.ReadKey();
}
```

file:///C:/Users/Dr.PV.Praveen Sundar/documents/visual studio 2010/Projects/Inheritance/Inheritance/bin/Debug/Inheritance.EXE

Enter the Valid Student ID:-10
ID value should be greater than zero
Enter the Valid Name :
Name should not be empty
Enter the Valid Marks :125
Invalid Marks
Student ID = 0
Student Name = No Name
Student Pass Mark = 35

Enter the Valid Student ID:21312

Enter the Valid Name :Praveen

Enter the Valid Marks :75

Student ID = 75

Student Name = Praveen

Student Pass Mark = 35

- Advantages of using Properties in C#?
 - ▣ Properties will provide the abstraction to the data fields.
 - ▣ They also provide security to the data fields.
 - ▣ Properties can also validate the data before storing into the data fields.

Indexers

62

- ❑ C# indexers are usually known as smart arrays.
- ❑ A C# indexer is a class property that allows you to access a member variable of a class or struct using the features of an array.
- ❑ In C#, indexers are created using **this** keyword.
- ❑ Indexers in C# are applicable on both classes and structs.
- ❑ Defining an indexer allows you to create a class like that can allows its items to be accessed an array.
- ❑ Instances of that class can be accessed using the [] array access operator.
- ❑ C# allows us to define custom indexers, generic indexers, and also overload indexers.

- ❑ An indexer can be defined the same way as property with this keyword and square brackets [].
- ❑ Actually an indexer is a special kind of property and includes get and set accessors to specify its behaviour. Hence, indexers and properties share the same syntax.
- ❑ An indexer is defined in the same way as a property is defined with following differences:
 - ▣ The Indexer takes an index argument as its subscript.
 - ▣ The class itself is being treated as an array and the keyword **this** is used as the name of the indexer in the indexer definition.

Syntax

64

```
<modifier> <return type> this [argument list]
{
  get
  {
    // your get block code
  }
  set
  {
    // your set block code
  }
}
```

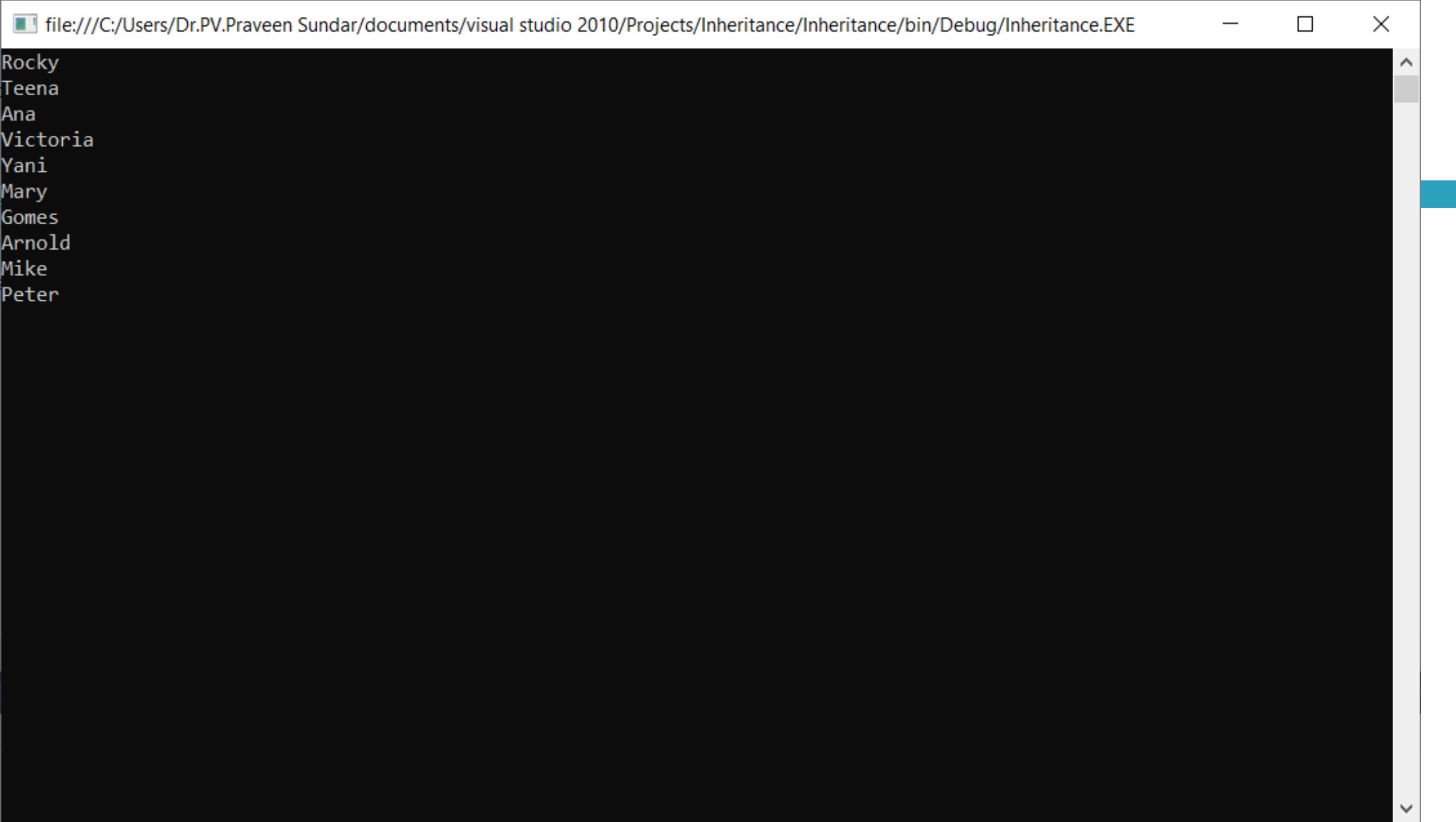
- ❑ Indexers are always created with **this** keyword.
- ❑ Indexers are implemented through get and set accessors for the [] operator.
- ❑ The formal parameter list of an indexer corresponds to that of a method and at least one parameter should be specified.
- ❑ Indexer is an instance member so can't be static but property can be static.
- ❑ Indexer is identified by its signature where as a property is identified it's name.
- ❑ Indexers are accessed using indexes where as properties are accessed by names.
- ❑ Indexer can be overloaded.
- ❑ Indexer are defined in pretty much same way as properties, with get and set functions. The main difference is that the name of the indexer is the keyword this.

```
Inheritance.Indexereg
Main(string[] args)

using System;

namespace Inheritance
{
    class Indexereg
    {
        class IndexerClass
        {
            private string[] names = new string[10];
            public string this[int i]
            {
                get
                {
                    return names[i];
                }
                set
                {
                    names[i] = value;
                }
            }
        }
    }

    static void Main(string[] args)
    {
        IndexerClass Team = new IndexerClass();
        Team[0] = "Rocky";
        Team[1] = "Teena";
        Team[2] = "Ana";
        Team[3] = "Victoria";
        Team[4] = "Yani";
        Team[5] = "Mary";
        Team[6] = "Gomes";
        Team[7] = "Arnold";
        Team[8] = "Mike";
        Team[9] = "Peter";
        for (int i = 0; i < 10; i++)
        {
            Console.WriteLine(Team[i]);
        }
        Console.ReadKey();
    }
}
```



Rocky
Teena
Ana
Victoria
Yani
Mary
Gomes
Arnold
Mike
Peter

Indexers	Properties
Indexers are created with this keyword.	Properties don't require this keyword.
Indexers are identified by signature.	Properties are identified by their names.
Indexers are accessed using indexes.	Properties are accessed by their names.
Indexer are instance member, so can't be static.	Properties can be static as well as instance members.
A get accessor of an indexer has the same formal parameter list as the indexer.	A get accessor of a property has no parameters.
A set accessor of an indexer has the same formal parameter list as the indexer, in addition to the value parameter.	A set accessor of a property contains the implicit value parameter.

Generics

69

- ❑ Generic means the general form, not specific. In C#, generic means not specific to a particular data type. In other words, Generics allow you to write a class or method that can work with any data type. We always write the specifications for the class or the method, with substitute parameters for data types.
- ❑ When the compiler encounters a constructor for the class or a function call for the method, it generates code to handle the specific data type.
- ❑ A generic type is declared by specifying a type parameter in angle brackets after a type name, e.g. `TypeName<T>` where T is a type parameter.
- ❑ A type parameter is a placeholder for a particular type specified when creating an instance of the generic type.

Generic Class

70

- Generic classes are defined using a type parameter in an angle brackets after the class name.
- The following defines a generic class.
- Example: Define Generic Class

```
class DataStore<T>
{
    public T Data { get; set; }
}
```

- Above, the DataStore is a generic class. T is called type parameter, which can be used as a type of fields, properties, method parameters, return types, and delegates in the DataStore class.
- For example, Data is generic property because we have used a type parameter T as its type instead of the specific data type.
- It is not required to use T as a type parameter. You can give any name to a type parameter. Generally, T is used when there is only one type parameter.

Instantiating Generic Class

71


- You can also define multiple type parameters separated by a comma.
- Example: Generic Class with Multiple Type Parameters

```
class KeyValuePair<TKey, TValue>
{
    public TKey Key { get; set; }
    public TValue Value { get; set; }
}
```

- You can create an instance of generic classes by specifying an actual type in angle brackets. The following creates an instance of the generic class DataStore.
- `DataStore<string> store = new DataStore<string>();`
- Above, we specified the string type in the angle brackets while creating an instance. So, T will be replaced with a string type wherever T is used in the entire class at compile-time. Therefore, the type of Data property would be a string.

```
DataStore<string> store = new DataStore<string>();
```

```
class DataStore<T>  
{  
    public T Data { get; set; }  
}
```



You can assign a string value to the Data property. Trying to assign values other than string will result in a compile-time error.

```
DataStore<string> store = new DataStore<string>();  
store.Data = "Hello World!";  
//store.Data = 123; //compile-time error
```

```
DataStore<string> strStore = new DataStore<string>();  
strStore.Data = "Hello World!";  
//strStore.Data = 123; // compile-time error
```

```
DataStore<int> intStore = new DataStore<int>();  
intStore.Data = 100;  
//intStore.Data = "Hello World!"; // compile-time error
```

```
KeyValuePair<int, string> kvp1 = new KeyValuePair<int, string>();  
kvp1.Key = 100;  
kvp1.Value = "Hundred";
```

```
KeyValuePair<string, string> kvp2 = new KeyValuePair<string, string>();  
kvp2.Key = "IT";  
kvp2.Value = "Information Technology";
```

```
public class MyGenericArray<T>
{
    private T[] array;

    public MyGenericArray(int size)
    {
        array = new T[size + 1];
    }
    public T getItem(int index)
    {
        return array[index];
    }
    public void setItem(int index, T value)
    {
        array[index] = value;
    }
}
```



```
class Tester
{
    static void Main(string[] args)
    {
        //declaring an int array
        MyGenericArray<int> intArray = new MyGenericArray<int>(5);

        //setting values
        for (int c = 0; c < 5; c++)
        {
            intArray.setItem(c, c * 5);
        }

        //retrieving the values
        for (int c = 0; c < 5; c++)
        {
            Console.Write(intArray.getItem(c) + " ");
        }

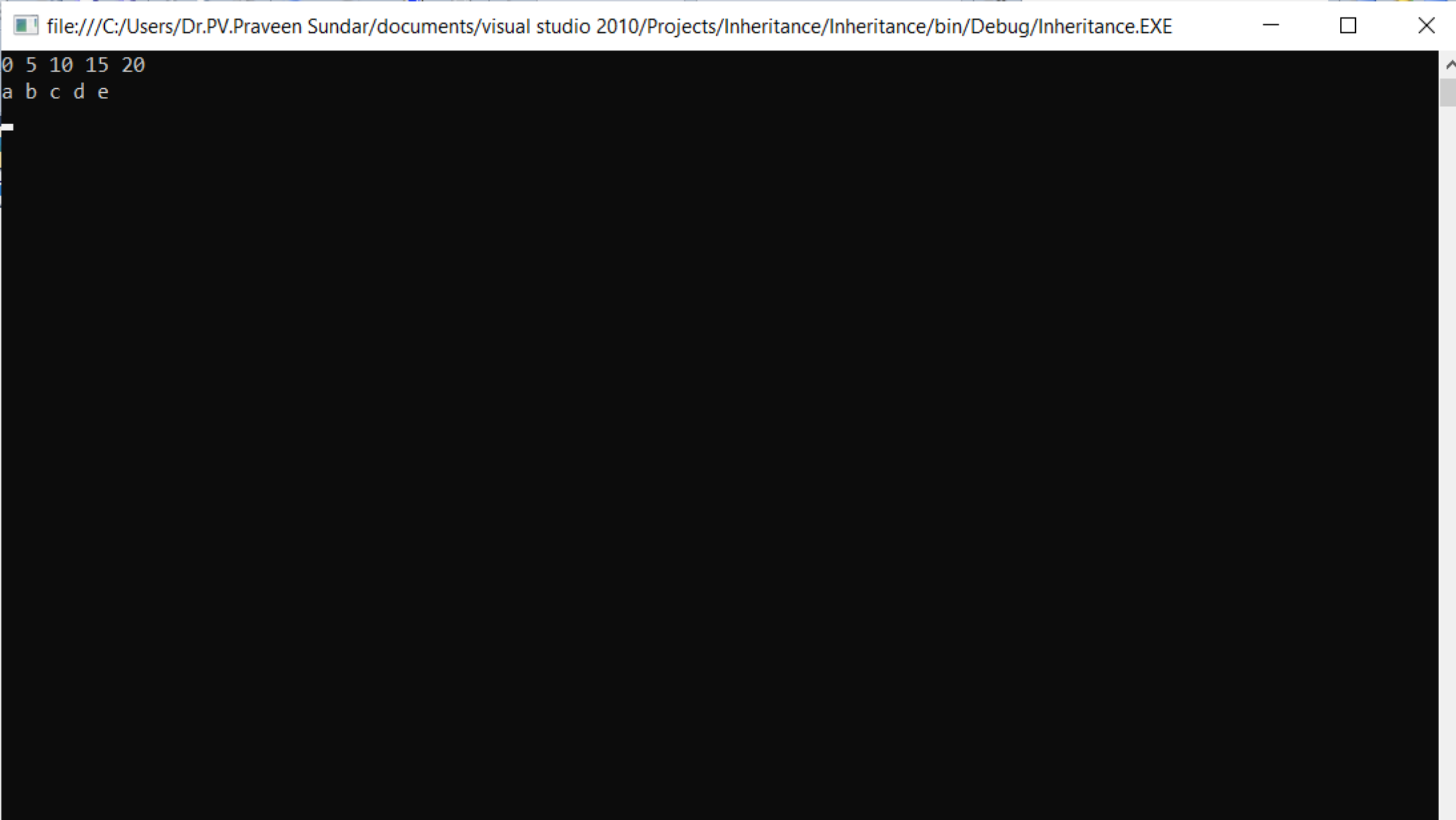
        Console.WriteLine();

        //declaring a character array
        MyGenericArray<char> charArray = new MyGenericArray<char>(5);

        //setting values
        for (int c = 0; c < 5; c++)
        {
            charArray.setItem(c, (char)(c + 97));
        }

        //retrieving the values
        for (int c = 0; c < 5; c++)
        {
            Console.Write(charArray.getItem(c) + " ");
        }
        Console.WriteLine();

        Console.ReadKey();
    }
}
```



Generic Class Characteristics

78

- ❑ A generic class increases the reusability. The more type parameters mean more reusable it becomes. However, too much generalization makes code difficult to understand and maintain.
- ❑ A generic class can be a base class to other generic or non-generic classes or abstract classes.
- ❑ A generic class can be derived from other generic or non-generic interfaces, classes, or abstract classes.
- ❑ we can create your own generic interfaces, classes, methods, events, and delegates.
- ❑ we may create generic classes constrained to enable access to methods on particular data types.
- ❑ we may get information on the types used in a generic data type at run-time by means of reflection.

Generic Methods

79

- A method declared with the type parameters for its return type or parameters is called a generic method.

```
class DataStore<T>
{
    private T[] _data = new T[10];

    public void AddOrUpdate(int index, T item)
    {
        if(index >= 0 && index < 10)
            _data[index] = item;
    }

    public T GetData(int index)
    {
        if(index >= 0 && index < 10)
            return _data[index];
        else
            return default(T);
    }
}
```

- Above, the AddOrUpdate() and the GetData() methods are generic methods. The actual data type of the item parameter will be specified at the time of instantiating the DataStore<T> class. as shown below.

Example: Generic Methods

```
DataStore<string> cities = new DataStore<string>();  
cities.AddOrUpdate(0, "Mumbai");  
cities.AddOrUpdate(1, "Chicago");  
cities.AddOrUpdate(2, "London");  
  
DataStore<int> empIds = new DataStore<int>();  
empIds.AddOrUpdate(0, 50);  
empIds.AddOrUpdate(1, 65);  
empIds.AddOrUpdate(2, 89);
```

- The generic parameter type can be used with multiple parameters with or without non-generic parameters and return type. The followings are valid generic method overloading.

Example: Generic Method Overloading

```
public void AddOrUpdate(int index, T data) { }  
public void AddOrUpdate(T data1, T data2) { }  
public void AddOrUpdate<U>(T data1, U data2) { }  
public void AddOrUpdate(T data) { }
```

A non-generic class can include generic methods by specifying a type parameter in angle brackets with the method name, as shown below.

Example: Generic Method in Non-generic Class

```
class Printer  
{  
    public void Print<T>(T data)  
    {  
        Console.WriteLine(data);  
    }  
}  
  
Printer printer = new Printer();  
printer.Print<int>(100);  
printer.Print(200); // type infer from the specified value  
printer.Print<string>("Hello");  
printer.Print("World!"); // type infer from the specified value
```

```

namespace Inheritance
{
    public class Program
    {
        public static void Main()
        {
            DataStore<int> grades = new DataStore<int>();
            grades[0] = 100;
            grades[1] = 25;
            grades[2] = 34;
            grades[3] = 42;
            grades[4] = 12;
            grades[5] = 18;
            grades[6] = 2;
            grades[7] = 95;
            grades[8] = 75;
            grades[9] = 53;

            for (int i = 0; i < grades.Length; i++)
                Console.WriteLine(grades[i]);

            DataStore<string> names = new DataStore<string>(5);
            names[0] = "Steve";
            names[1] = "Bill";
            names[2] = "James";
            names[3] = "Ram";
            names[4] = "Andy";

            for (int i = 0; i < names.Length; i++)
                Console.WriteLine(names[i]);
            Console.ReadKey();
        }
    }
}

```

- ❑ Indexer can also be generic.
- ❑ The following generic class includes generic indexer and properties.

```
class DataStore<T>
{
    private T[] store;

    public DataStore()
    {
        store = new T[10];
    }

    public DataStore(int length)
    {
        store = new T[length];
    }

    public T this[int index]
    {
        get
        {
            if (index < 0 && index >= store.Length)
                throw new IndexOutOfRangeException("Index out of range");

            return store[index];
        }
        set
        {
            if (index < 0 || index >= store.Length)
                throw new IndexOutOfRangeException("Index out of range");

            store[index] = value;
        }
    }

    public int Length
    {
        get
        {
            return store.Length;
        }
    }
}
```


file:///C:/Users/Dr.PV.Praveen Sundar/documents/visual studio 2010/Projects/Inheritance/Inheritance/bin/Debug/Inheritance.EXE

100
25
34
42
12
18
2
95
75
53
Steve
Bill
James
Ram
Andy
_

Index overloading

85

- ❑ Like functions, Indexers can also be overloaded.
- ❑ In C#, we can have multiple indexers in a single class.
- ❑ Multiple types allow you to build in flexibility and further increase the fault tolerance and robustness of the class and application.
- ❑ To overload an indexer, declare it with multiple parameters and each parameter should have a different data type.
- ❑ Indexers are overloaded by passing 2 different types of parameters.
- ❑ It is quite similar to method overloading.

```
using System;
using System.Collections.Generic;

namespace Inheritance
{
    class OverloadedIndexer
    {
        private string[] guideNames = new string[10];

        public string this[int index]
        {
            get { return guideNames [index]; }
            set { guideNames[index] = value; }
        }

        public string this[string name]
        {
            get
            {
                return name.ToUpper();
            }
        }

        public static void Main()
        {
            OverloadedIndexer obj1 = new OverloadedIndexer();
            for (int i = 0; i < 5; i++)
            {
                Console.WriteLine("Enter the Guide Name :");
                obj1[i] = Console.ReadLine();
            }
            for (int i = 0; i < 5; i++)
            {
                Console.WriteLine(" Guide Name: {0}", obj1.guideNames[i]);
                Console.WriteLine(" Guide Name in Upper Case : {0}", obj1[obj1[i]]);
            }
            Console.ReadKey();
        }
    }
}
```

```
Enter the Guide Name :Praveen
Enter the Guide Name :Sundar
Enter the Guide Name :Vamsy
Enter the Guide Name :Suresh
Enter the Guide Name :Raina
Guide Name: Praveen
Guide Name in Upper Case : PRAVEEN
Guide Name: Sundar
Guide Name in Upper Case : SUNDAR
Guide Name: Vamsy
Guide Name in Upper Case : VAMSY
Guide Name: Suresh
Guide Name in Upper Case : SURESH
Guide Name: Raina
Guide Name in Upper Case : RAINA
```

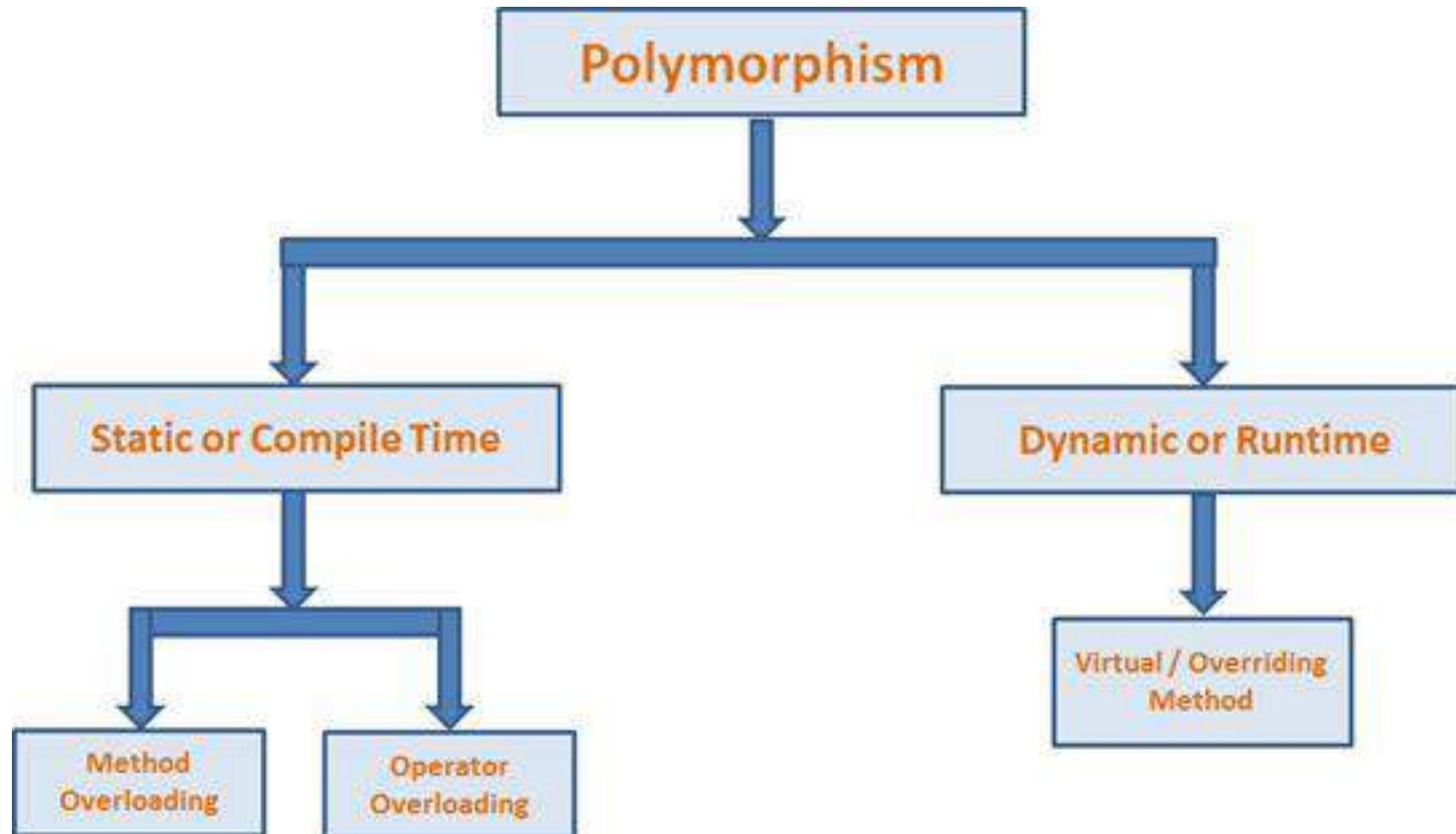
Thank you



POLYMORPHISM

Dr P.V. Praveen Sundar
Assistant Professor,
Department of Computer Science
Adhiparasakthi College of Arts & Science,
Kalavai.

- ❑ Polymorphism is often referred to as the third pillar of object-oriented programming, after encapsulation and inheritance.
- ❑ Polymorphism is a Greek word, meaning "one name many forms".
- ❑ In other words, one object has many forms or has one name with multiple functionalities. "Poly" means many and "morph" means forms/Behaviour.
- ❑ Polymorphism provides the ability to a class to have multiple implementations with the same name.
- ❑ Polymorphism can be static or dynamic.
- ❑ In static polymorphism, the response to a function is determined at the compile time.
- ❑ In dynamic polymorphism, it is decided at run-time.



Static Polymorphism

4

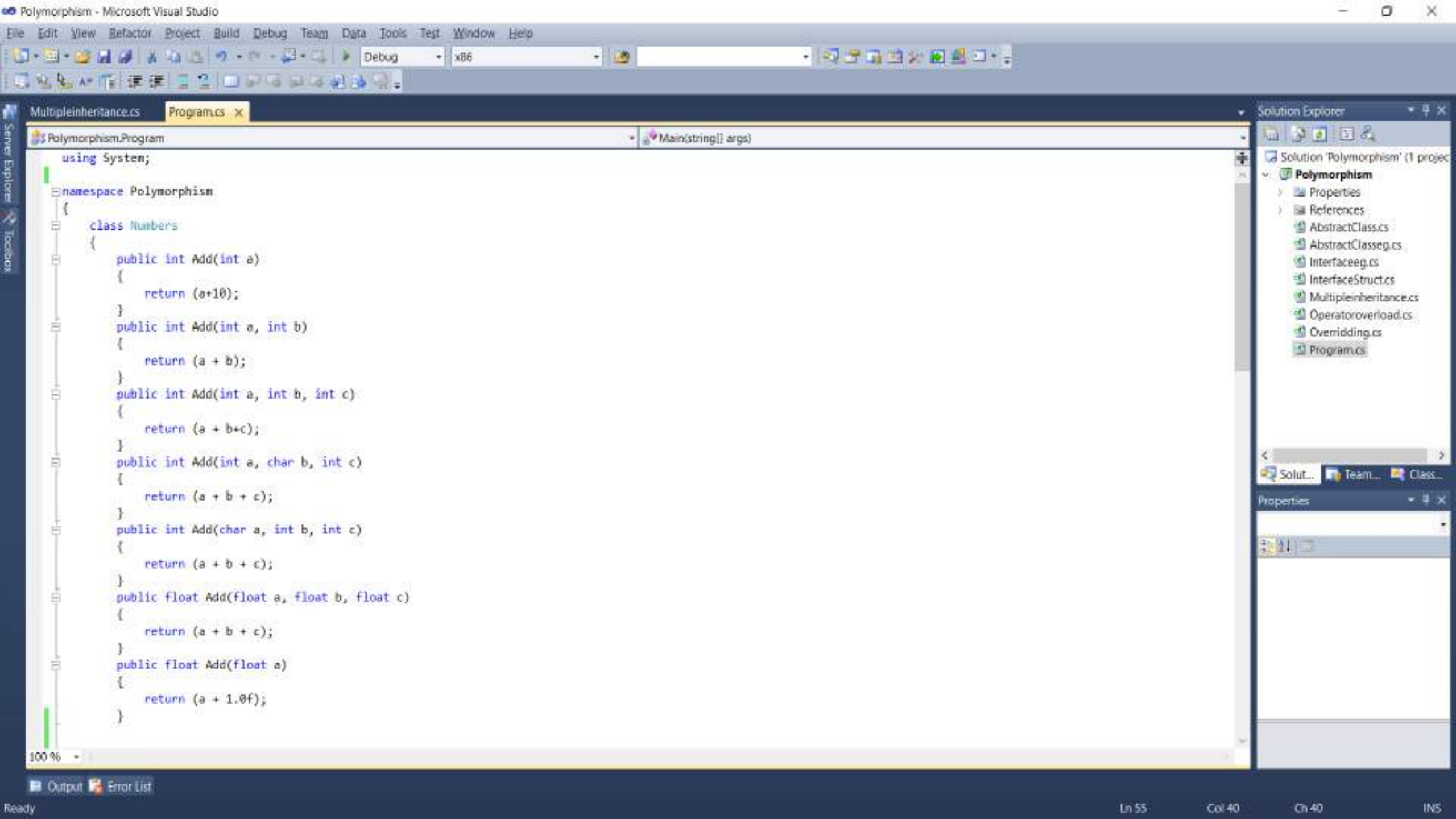
- ❑ It is also known as **Early Binding** or **Compile time polymorphism** or **Static polymorphism**.
- ❑ In the case of compile-time polymorphism, the object of class recognizes which method to be executed for a particular method call at the time of program compilation and binds the method call with method definition.
- ❑ In case of overloading each method will have a different signature and based on the method call, the compiler can easily recognize the method which matches the method signature.
- ❑ Static polymorphism is achieved by using Function overloading and Operator overloading.

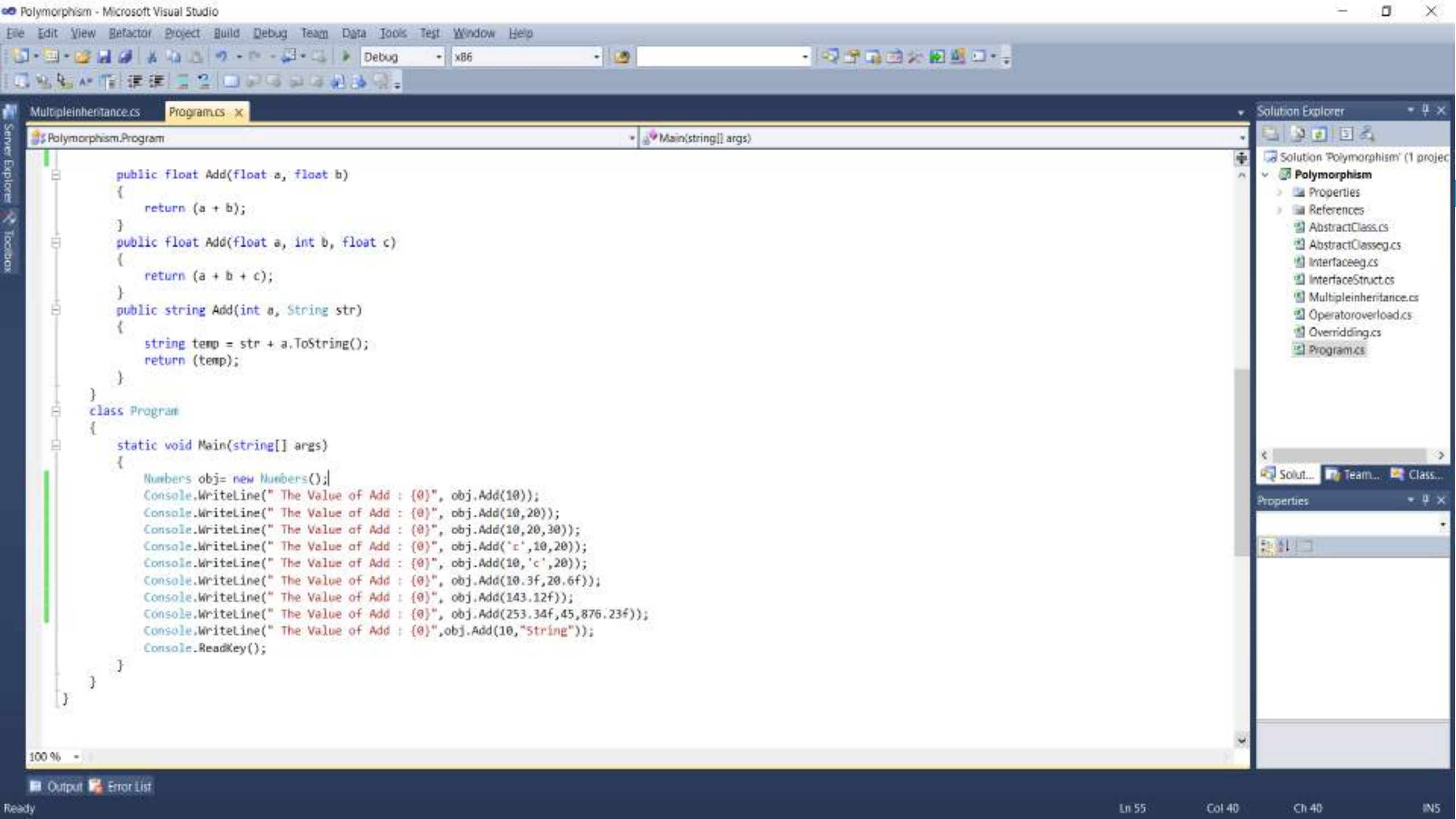
Function Overloading

5

- ❑ It is a process of creating multiple methods in a class with the same name but with a different signature.
- ❑ In C#, It is also possible to overload the methods in the derived classes, which means, it allows us to create a method in the derived class with the same name as the method name defined in the base class.
- ❑ In simple words, we can say that the Function Overloading in C# allows a class to have multiple methods with the same name but with a different signature.
- ❑ So in C# functions or methods can be overloaded based on the number, type (int, float, etc.), order, and kind (Value, Ref or Out) of parameters.
- ❑ The signature of a method consists of the name of the method and the data type, number, order, and kind (Value, Ref or Out) of parameters.

- ❑ If two methods have the same method name those methods are considered overloaded methods.
- ❑ Then the rule we should check is both methods must have different parameter types/number/order. But there is no rule on return type, non-accessibility modifier and accessibility modifier means overloading methods can have their own return type, non-accessibility modifier, and accessibility modifier because overloading methods are different methods.
- ❑ Methods can be overloaded in the same or in super and sub classes because overloaded methods are different methods.
- ❑ A method that is defined in a class can also be overloaded under its child class. It is called inheritance-based overloading.





```
The Value of Add : 20  
The Value of Add : 30  
The Value of Add : 60  
The Value of Add : 129  
The Value of Add : 129  
The Value of Add : 30.9  
The Value of Add : 144.12  
The Value of Add : 1174.57  
The Value of Add : String10
```

Operator Overloading

10

- ❑ C# supports a number of operators, all these operators have predefined implementations.
- ❑ The operands of most of these operators are of primitive datatypes.
- ❑ If an operator has a user defined implementation, the operator is said to be overloaded.
- ❑ The existing C# operator redefined for use with user-defined types is called as Operator Overloading.
- ❑ In fact, the overloaded operator is simply another means of calling a method. It does not add anything new to the language. It helps in operator abstraction.
- ❑ Overloaded operators are functions with special names the keyword operator followed by the symbol for the operator being defined.

Operator Overloading

11

- ❑ Similar to any other function, an overloaded operator has a return type and a parameter list.
- ❑ Operator is the keyword which is used to implement operator overloading. The return type of operator overload can never be void.
- ❑ In operator overloading preference is always given to user-defined implementations rather than predefined implementations.
- ❑ In user-defined implementations, syntax and precedence cannot be modified.
 - We can overload all the binary operators i.e $+$, $-$, $*$, $/$, $\%$, $\&$, $|$, $<<$, $>>$.
 - We can overload all the unary operators i.e. $++$, $-$, $!$, $++$, $--$, $+$, $-$, \sim .
 - Some operators like $\&\&$, $||$, $[]$, $()$ cannot be overloaded.
 - We can overload relational operators in pairs. These are $==$, $=$, $<$, $>$, $<=$, $>=$ etc.
 - When binary operators are overloaded, the left hand object must be an object of the relevant class

Below is the syntax of implementing operator overloading:

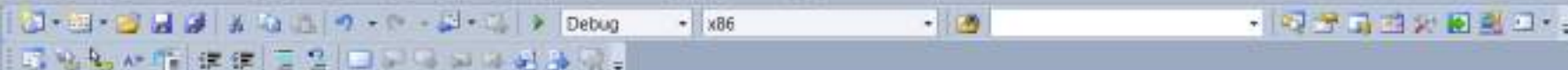
```
public static classname operator op (parameters)
{
    // Code
}
```

For Unary Operator

```
public static classname operator op (t)
{
    // Code
}
```

For Binary Operator

```
public static classname operator op (t1, t2)
{
    // Code
}
```



OperatorOverload.cs* MultipleInheritance.cs Program.cs

Polymorphism.OperatorOverload

operator <=(OperatorOverload obj1, OperatorOverload obj2)

using System;

namespace Polymorphism

{
class OperatorOverload

{

int x, y;

public void input()

{

Console.WriteLine("Enter the value of X:");

x = Convert.ToInt32(Console.ReadLine());

Console.WriteLine("Enter the value of Y:");

y = Convert.ToInt32(Console.ReadLine());

}

public static OperatorOverload operator ++(OperatorOverload obj)

{

obj.x++;

obj.y++;

return (obj);

}

public static OperatorOverload operator --(OperatorOverload obj)

{

obj.x--;

obj.y--;

return (obj);

}

public static OperatorOverload operator -(OperatorOverload obj)

{

obj.x = -obj.x;

obj.y = -obj.y;

return (obj);

}

100 %

Output Error List

```
public static OperatorOverload operator + (OperatorOverload obj1, OperatorOverload obj2)
{
    OperatorOverload obj3 = new OperatorOverload();
    obj3.x = obj1.x + obj2.x;
    obj3.y = obj1.y + obj2.y;
    return (obj3);
}

public static OperatorOverload operator -(OperatorOverload obj1, OperatorOverload obj2)
{
    OperatorOverload obj3 = new OperatorOverload();
    obj3.x = obj1.x - obj2.x;
    obj3.y = obj1.y - obj2.y;
    return (obj3);
}

public static OperatorOverload operator *(OperatorOverload obj1, OperatorOverload obj2)
{
    OperatorOverload obj3 = new OperatorOverload();
    obj3.x = obj1.x * obj2.x;
    obj3.y = obj1.y * obj2.y;
    return (obj3);
}

public static OperatorOverload operator / (OperatorOverload obj1, OperatorOverload obj2)
{
    OperatorOverload obj3 = new OperatorOverload();
    obj3.x = obj1.x / obj2.x;
    obj3.y = obj1.y / obj2.y;
    return (obj3);
}

public static bool operator <(OperatorOverload obj1, OperatorOverload obj2)
{
    if ((obj1.x < obj2.x) && (obj1.y < obj2.y))
        return (true);
    else
        return(false);
}

public static bool operator >(OperatorOverload obj1, OperatorOverload obj2)
{
    if ((obj1.x > obj2.x) && (obj1.y > obj2.y))
        return (true);
    else
        return (false);
}
```

```
public static bool operator <=(Operatoroverload obj1, Operatoroverload obj2)
{
    if ((obj1.x <= obj2.x) && (obj1.y <= obj2.y))
        return (true);
    else
        return (false);
}

public static bool operator >=(Operatoroverload obj1, Operatoroverload obj2)
{
    if ((obj1.x >= obj2.x) && (obj1.y >= obj2.y))
        return (true);
    else
        return (false);
}

public static bool operator ==(Operatoroverload obj1, Operatoroverload obj2)
{
    if ((obj1.x == obj2.x) && (obj1.y == obj2.y))
        return (true);
    else
        return (false);
}

public static bool operator !=(Operatoroverload obj1, Operatoroverload obj2)
{
    return (!(obj1==obj2));
}

public void Display()
{
    Console.WriteLine(" The value of X : {0} and Y: {1}", x, y);
}

public static void Main()
{
    int ch;
    Operatoroverload obj1, obj2, obj3;
    obj1 = new Operatoroverload();
    obj1.input();
    obj1.Display();
    obj2 = new Operatoroverload();
    obj2.input();
    obj2.Display();
    do
    {
```

```
Console.WriteLine("1- Addition ");
Console.WriteLine("2- Subtraction ");
Console.WriteLine("3- Multiplication ");
Console.WriteLine("4- Division ");
Console.WriteLine("5- Comparision ");
ch = Convert.ToInt32(Console.ReadLine());
switch (ch)
{
    case 1:
        obj3 = obj1 + obj2;
        obj3.Display();
        break;
    case 2:
        obj3 = obj1 - obj2;
        obj3.Display();
        break;
    case 3:
        obj3 = obj1 * obj2;
        obj3.Display();
        break;
    case 4:
        obj3 = obj1 / obj2;
        obj3.Display();
        break;
    case 5:
        Console.WriteLine(" Object 1 is Less than Object 2 : {0} ",obj1 < obj2);
        Console.WriteLine(" Object 1 is greater than Object 2 : {0} ", obj1 > obj2);
        Console.WriteLine(" Object 1 is Less than or Equal to Object 2 : {0} ",obj1 <= obj2);
        Console.WriteLine(" Object 1 is greater than or Equal to Object 2 : {0} ", obj1 >= obj2);
        Console.WriteLine(" Object 1 is Equal to Object 2 :{0} ", obj1 == obj2);
        Console.WriteLine(" Object 1 is Not equal to Object 2 :{0} ", obj1 != obj2);
        break;
    default:
        break;
}
} while (ch<6);
Console.ReadKey();
```



```
Enter the value of X:10
Enter the value of Y:20
The value of X : 10 and Y: 20
Enter the value of X:30
Enter the value of Y:40
The value of X : 30 and Y: 40
1- Addition
2- Subtraction
3- Multiplication
4- Division
5- Comparision
1
The value of X : 40 and Y: 60
1- Addition
2- Subtraction
3- Multiplication
4- Division
5- Comparision
2
The value of X : -20 and Y: -20
1- Addition
2- Subtraction
3- Multiplication
4- Division
5- Comparision
3
The value of X : 300 and Y: 800
1- Addition
2- Subtraction
3- Multiplication
4- Division
5- Comparision
4
The value of X : 0 and Y: 0
1- Addition
2- Subtraction
3- Multiplication
4- Division
5- Comparision
5
Object 1 is Less than Object 2 : True
Object 1 is greater than Object 2 : False
Object 1 is Less than or Equal to Object 2 : True
Object 1 is greater than or Equal to Object 2 : False
Object 1 is Equal to Object 2 :False
Object 1 is Not equal to Object 2 :True
1- Addition
2- Subtraction
3- Multiplication
4- Division
```

1. The **virtual** keyword is used to modify a method, property, indexer, or event declared in the base class and allow it to be overridden in the derived class.
2. The **override** keyword is used to extend or modify a virtual/abstract method, property, indexer, or event of base class into derived class.
3. The **new** keyword is used to hide a method, property, indexer, or event of base class into derived class.

Method Overriding

19

- ❑ In C#, The process of re-implementing the base class non-static method in the subclass with the same prototype (same signature defined in the superclass) is called Function Overriding or Method Overriding.
- ❑ In C#, the Method Overriding is also called run time polymorphism or late binding.
- ❑ The Method Overriding in C# can be achieved using **Override** & **Virtual** keywords and the inheritance principle.
- ❑ The implementation of the subclass overrides (i.e. replaces) the implementation of base class methods.
- ❑ The overriding method is always going to be executed from the current class object.
- ❑ If a method in sub-class contains the same signature as the base class non-private non-static method, then the subclass method is treated as the overriding method and the superclass method is treated as the overridden method.

- ❑ To override a parent class method in its child class, first the method in the parent class must be declared as virtual by using the keyword **virtual**, then only the child classes get the permission for overriding that method.
- ❑ Declaring the method as virtual is marking the method as overridable. If the child class wants to override the parent class virtual method then the child class can do it with the help of the override modifier. But overriding the method under child class is not mandatory for the child classes.

Syntax:

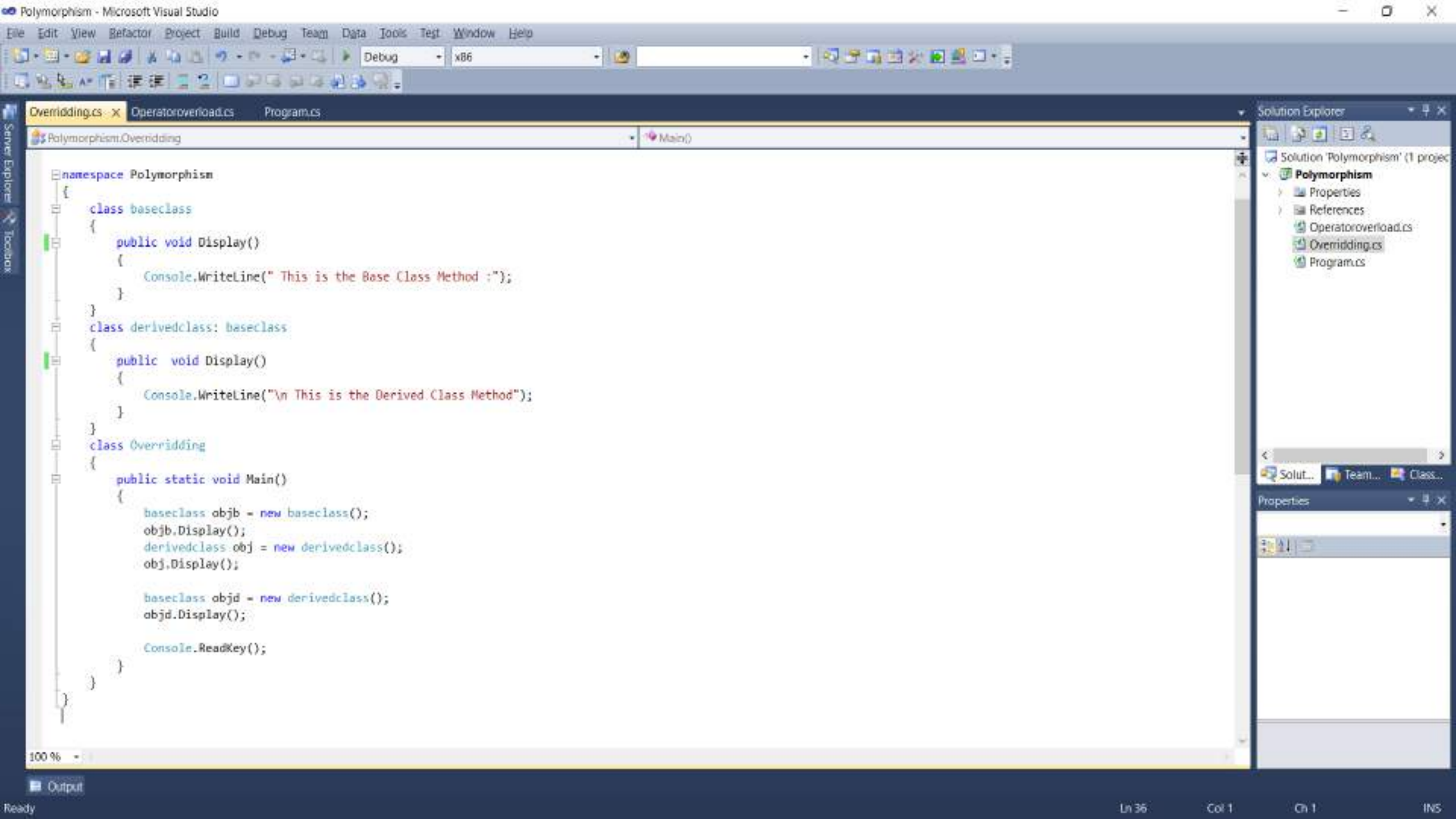
Class1:

```
Public virtual void show(){} //virtual function (overridable)
```

Class2: Class1

```
Public override void show(){} //overriding
```

- Once we re-implement the parent class methods under the child class, then the object of the child class calls its own methods but not its parent class method.
- if you want to still consume or call the parent class's methods from the child class, then it can be done in two different ways.
- By creating the parent class object under the child class, we can call the parent class methods from the child class, or by using the base keyword, we can call parent class methods from the child class.



```
Polymorphism - Microsoft Visual Studio
File Edit View Refactor Project Build Debug Team Data Tools Test Window Help
[Icons] Debug x86 [Icons]

Overriding.cs Operatoroverload.cs Program.cs
Polymorphism\Overriding Main()

namespace Polymorphism
{
    class baseclass
    {
        public void Display()
        {
            Console.WriteLine(" This is the Base Class Method :");
        }
    }
    class derivedclass: baseclass
    {
        public void Display()
        {
            Console.WriteLine("\n This is the Derived Class Method");
        }
    }
    class Overriding
    {
        public static void Main()
        {
            baseclass objb = new baseclass();
            objb.Display();
            derivedclass obj = new derivedclass();
            obj.Display();

            baseclass objd = new derivedclass();
            objd.Display();

            Console.ReadKey();
        }
    }
}
```

Solution Explorer
Solution 'Polymorphism' (1 project)
Polymorphism
Properties
References
Operatoroverload.cs
Overriding.cs
Program.cs

Properties
[Icons]

Output
Ready
Ln 36 Col 1 Ch 1 INS

This is the Base Class Method :

This is the Derived Class Method

This is the Base Class Method :



Overriding.cs OperatorOverload.cs Program.cs

Polymorphism.Overriding Main()

```
namespace Polymorphism
{
    class baseclass
    {
        public virtual void Display()
        {
            Console.WriteLine(" This is the Base Class Method :");
        }
    }
    class derivedclass: baseclass
    {
        public override void Display()
        {
            Console.WriteLine("\n This is the Derived Class Method");
        }
    }
    class Overriding
    {
        public static void Main()
        {
            baseclass objb = new baseclass();
            objb.Display();
            derivedclass obj = new derivedclass();
            obj.Display();

            baseclass objd = new derivedclass();
            objd.Display();

            Console.ReadKey();
        }
    }
}
```

Solution Explorer

Solution 'Polymorphism' (1 project)

- Polymorphism
 - Properties
 - References
 - OperatorOverload.cs
 - Overriding.cs
 - Program.cs

Properties

file:///C:/Users/Dr.PV.Praveen Sundar/documents/visual studio 2010/Projects/Polymorphism/Polymorphism/bin/Debug/Polymorphis...

This is the Base Class Method :

This is the Derived Class Method

This is the Derived Class Method



Overriding.cs* Program.cs

Polymorphism>Overriding

Main()

using System;

namespace Polymorphism

{

class baseclass

{

public virtual void Display()

{

Console.WriteLine(" This is the Base Class Method :");

}

}

class derivedclass: baseclass

{

public override void Display()

{

base.Display();

Console.WriteLine("\n This is the Derived Class Method");

}

}

class Overriding

{

public static void Main()

{

baseclass objb = new baseclass();

objb.Display();

derivedclass obj = new derivedclass();

obj.Display();

baseclass objd = new derivedclass();

objd.Display();

Console.ReadKey();

}

}

}

Solution Explorer

Solution 'Polymorphism' (1 project)

Polymorphism

Properties

References

OperatorOverload.cs

Overriding.cs

Program.cs

Properties

Solution Explorer

Solution 'Polymorphism' (1 project)

Polymorphism

Properties

References

OperatorOverload.cs

Overriding.cs

Program.cs

Properties

Solution Explorer

Solution 'Polymorphism' (1 project)

Polymorphism

Properties

References

OperatorOverload.cs

Overriding.cs

Program.cs

Properties

Solution Explorer

Solution 'Polymorphism' (1 project)

Polymorphism

Properties

References

OperatorOverload.cs

Overriding.cs

Program.cs

Properties

Output

This is the Base Class Method :

This is the Base Class Method :

This is the Derived Class Method

This is the Base Class Method :

This is the Derived Class Method

Method Overloading	Method Overriding
It is an approach of defining multiple methods with the same name but with a different signature.	It is an approach of defining multiple methods with the same name and with the same signature.
Overloading a method can be performed within a class or within the child classes also.	Overriding of methods is not possible within the same class it must be performed under the child classes.
To overload a parent class method under the child class, the child class does not require permission from the parent.	To override a parent class method under the child class, first, the child class requires explicit permission from its parent.
This is all about defining multiple behaviors to a method.	This is all about changing the behavior of a method.
Used to implement static polymorphism.	Used to implement dynamic polymorphism.
This is a code refinement technique.	This is a code replacement technique.
No separate keywords are used to implement function overloading.	Use the virtual keyword for the base class function and override keyword in the derived class function to implement function overriding.

Method Hiding

29

- ❑ C# also provides a concept to hide the methods of the base class from derived class, this concept is known as Method Hiding. It is also known as Method Shadowing. In method hiding, you can hide the implementation of the methods of a base class from the derived class using the new keyword.
- ❑ Usually, we will get a compiler warning if we miss the new keyword. This is also used for re-implementing a parent class method under child class.
- ❑ Reimplementing parent class methods under child classes can be done using two different approaches, such as
 - ❑ Method overriding
 - ❑ Method hiding
- ❑ In the first case, we re-implement the parent class methods under child classes with the permission of parent class because here in parent class the method is declared as virtual giving permission to the child classes for overriding the methods.
- ❑ In the 2nd approach, we re-implement the method of parent class even if those methods are not declared as virtual that is without parent permission we are reimplementing the methods.



Overriding.cs* Program.cs

Polymorphism>Overriding

Main()

```
using System;

namespace Polymorphism
{
    class baseclass
    {
        public void Display()
        {
            Console.WriteLine(" This is the Base Class Method :");
        }
    }
    class derivedclass: baseclass
    {
        public new void Display()
        {
            Console.WriteLine("\n This is the Derived Class Method");
        }
    }
    class Overriding
    {
        public static void Main()
        {
            baseclass objb = new baseclass();
            objb.Display();
            derivedclass obj = new derivedclass();

            baseclass objd = new derivedclass();
            objd.Display();

            Console.ReadKey();
        }
    }
}
```

Solution Explorer

Solution 'Polymorphism' (1 project)

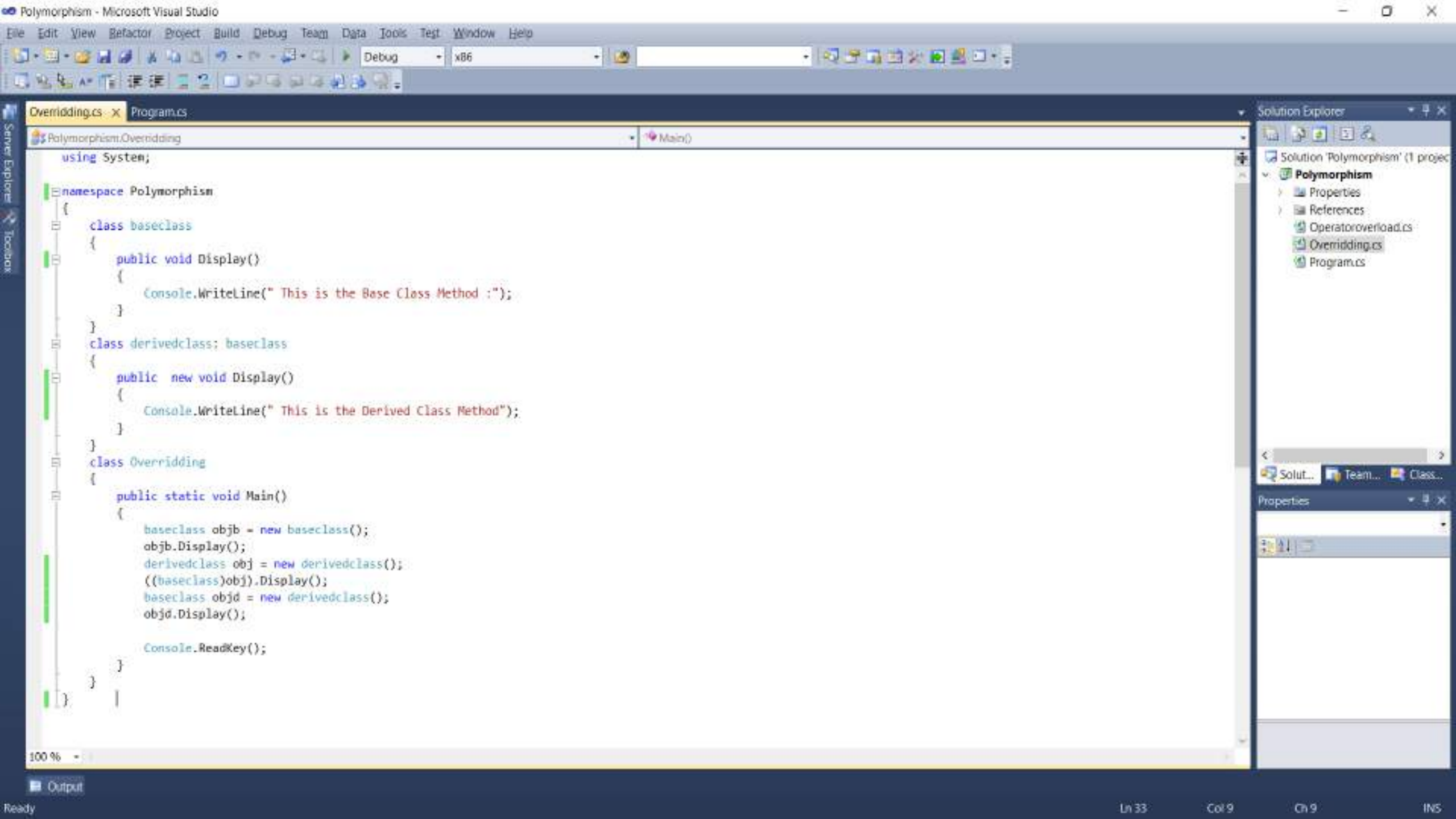
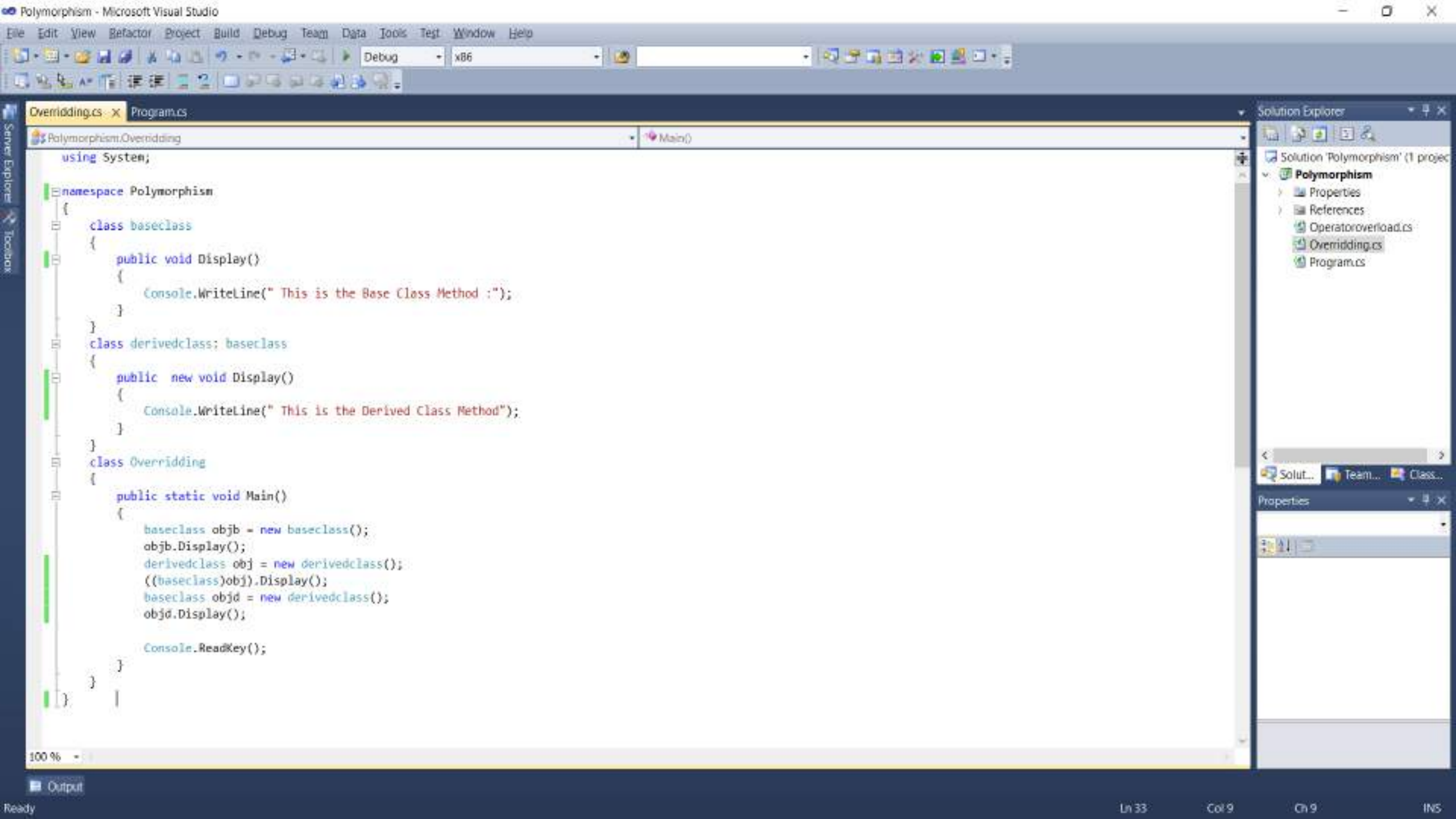
- Polymorphism
 - Properties
 - References
 - Operatoroverload.cs
 - Overriding.cs
 - Program.cs

Properties

Output

file:///C:/Users/Dr.PV.Praveen Sundar/documents/visual studio 2010/Projects/Polymorphism/Polymorphism/bin/Debug/Polymorphis... — □ ×

```
This is the Base Class Method :  
This is the Derived Class Method  
This is the Base Class Method :
```



file:///C:/Users/Dr.PV.Praveen Sundar/documents/visual studio 2010/Projects/Polymorphism/Polymorphism/bin/Debug/Polymorphis...

This is the Base Class Method :
This is the Base Class Method :
This is the Base Class Method :

Abstract Class

34

- ❑ A class that is declared by using the keyword **abstract** is called an abstract class.
- ❑ An abstract class is a partially implemented class used for developing some of the operations of an object which are common for all next level subclasses. So it contains both abstract methods, concrete methods including variables, properties, and indexers.
- ❑ An abstract class may or may not have abstract methods. But if a class contains an abstract method then it must be declared as abstract.
- ❑ An abstract class cannot be instantiated directly. It's compulsory to create/derive a subclass from the abstract class in order to provide the functionality to its abstract functions.

- ❑ The purpose of an abstract class is to provide a common definition of a base class that multiple derived classes can share. For example, a class library may define an abstract class that is used as a parameter to many of its functions, and require programmers using that library to provide their own implementation of the class by creating a derived class.
- ❑ Abstract classes may also define abstract methods. This is accomplished by adding the keyword `abstract` before the return type of the method.

```
public abstract class A
{
    // Class members here.
}
```


- ❑ Abstract methods are usually declared where two or more subclasses are expected to fulfill a similar role in a different manner.
- ❑ The subclasses are required to fulfill an interface, so the abstract superclass might provide several of the interface methods, but leave the subclasses to implement their own variations of the abstract methods.
- ❑ A method that does not have a body is called an abstract method. It is declared with the modifier `abstract`. It contains only a Declaration/signature and does not contain the implementation/body/definition of the method.
- ❑ An abstract function should be terminated with a semicolon. Overriding an abstract function is compulsory.

Rules of Abstract Method and Abstract Class

37

- **Rule1:** If a method does not have the body, then it should be declared as abstract using the abstract modifier else it leads to a compile-time error: “must declare a body because it is not marked abstract, extern, or partial”.
- **Rule2:** If a class has an abstract method it should be declared as abstract by using the keyword abstract else it leads to a compile-time error:
- **Rule3:** If a class is declared as abstract it cannot be instantiated violation leads to compile-time Error.
- **Rule4:** The sub-classes of an abstract class should override all the abstract methods or it should be declared as abstract else it leads to the compile-time error:



AbstractClass.cs* OperatorOverload.cs MultipleInheritance.cs Program.cs

Polymorphism.Motorcycle display()

```
using System;
namespace Polymorphism
{
    public abstract class Vehicle
    {
        public void show()
        {
            Console.WriteLine("This is an Abstract Method");
        }
        public abstract void display();
    }

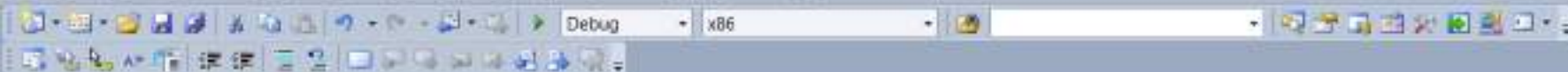
    public class Bus : Vehicle
    {
        public override void display()
        {
            Console.WriteLine("Bus");
        }
    }

    public class Car : Vehicle
    {
        public override void display()
        {
            Console.WriteLine("Car");
        }
    }

    public class Motorcycle : Vehicle
    {
        public override void display()
        {
            Console.WriteLine("Motorcycle");
        }
    }
}
```

100 %

Output Error List



AbstractClass.cs* OperatorOverload.cs MultipleInheritance.cs Program.cs

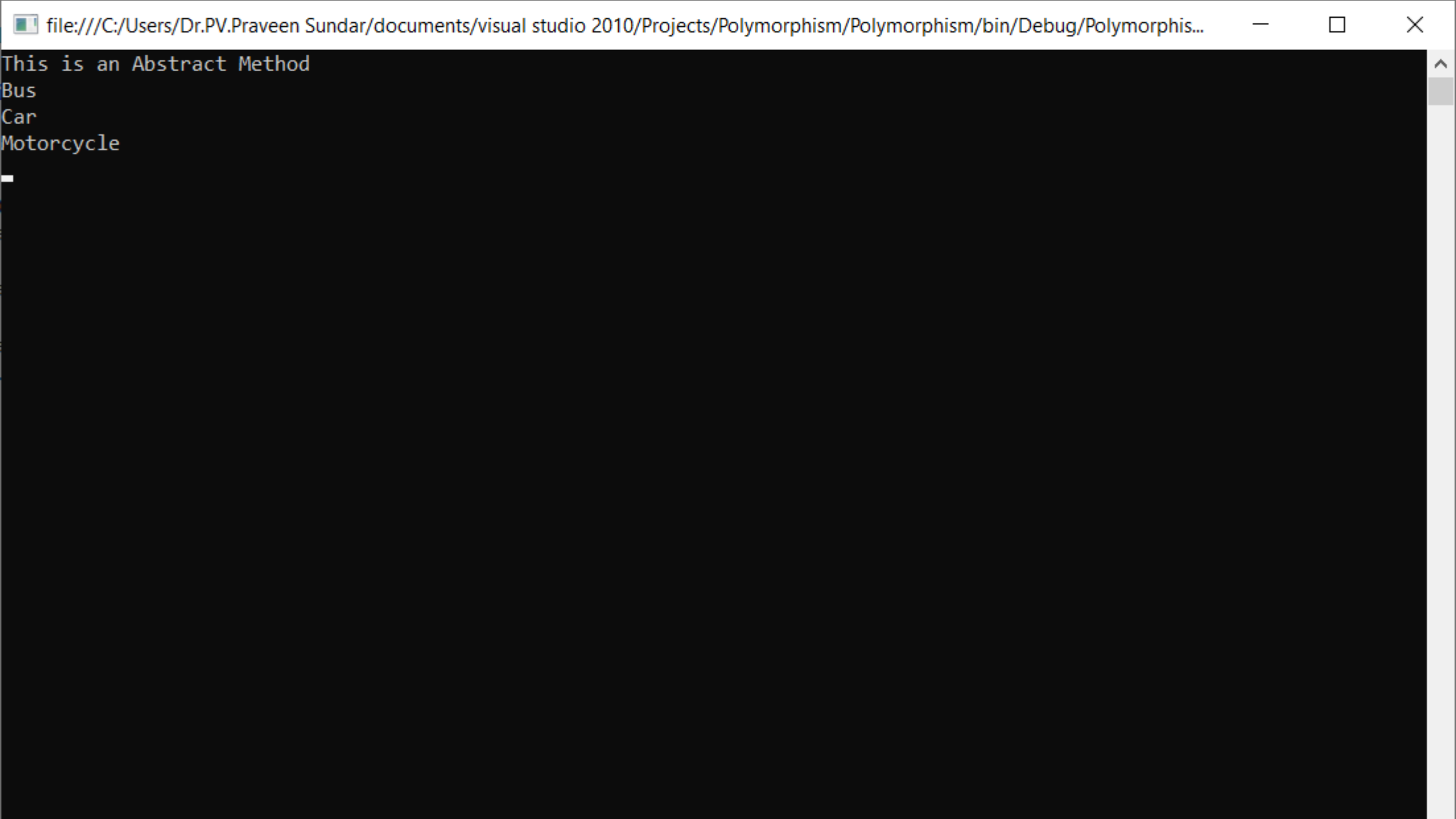
Polymorphism>AbstractClass

Main()

```
class AbstractClass
{
    public static void Main()
    {
        Vehicle v;
        v = new Bus();
        v.show();
        v.display();
        v = new Car();
        v.display();
        v = new Motorcycle();
        v.display();
        Console.ReadKey();
    }
}
```

100 %

Output Error List



This is an Abstract Method

Bus

Car

Motorcycle

AbstractClass.cs · x

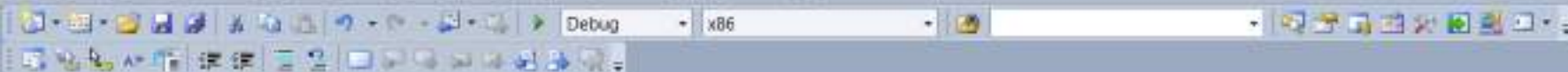
Polymorphism.Cylinder

Volume()

```
using System;
namespace Polymorphism
{
    abstract class Solid
    {
        internal double radius;
        public const double PI = 31.14159;
        public abstract void SurfaceArea();
        public abstract void Volume();
        public abstract void readRadius();

        public double baseArea(double r)
        {
            double p;
            p = PI * r * r;
            return (p);
        }
    }
    class Cylinder : Solid
    {
        double height;
        public Cylinder(double h)
        {
            height = h;
        }
        public override void readRadius()
        {
            Console.Write(" Enter the radius (Cylinder) : ");
            radius = Convert.ToDouble(Console.ReadLine());
        }
        public override void SurfaceArea()
        {
            double bArea;
            double totalArea;
            bArea = baseArea(radius);
            totalArea = 2 * PI * radius * height + 2 * bArea;
            Console.WriteLine(" Surface Area of Cylinder is {0}", totalArea);
        }
        public override void Volume()
        {
            Console.WriteLine(" Volume of Cylinder is {0}", (baseArea(radius)) * height);
        }
    }
}
```

100 %



AbstractClasses.cs*

Polymorphism.Core

Volume()

```
class Cone : Solid
{
    double height;
    public Cone(double h)
    {
        height = h;
    }
    public override void readRadius()
    {
        Console.Write(" Enter the radius (Cone) : ");
        radius = Convert.ToDouble(Console.ReadLine());
    }
    public override void SurfaceArea()
    {
        double area;
        double slantHeight;
        slantHeight = Math.Sqrt(radius * radius + height * height);
        area = PI * radius * slantHeight;
        Console.WriteLine(" Surface Area of Cone is {0}", area);
    }
    public override void Volume()
    {
        double volume;
        volume = (1.0 / 3) * (baseArea(radius)) * height;
        Console.WriteLine(" Volume of Cone is {0}", volume);
    }
}
```

AbstractClasse.cs*

Polymorphism.Cone

Volume()

```
class Sphere : Solid
{
    public override void readRadius()
    {
        Console.Write(" Enter the radius (Sphere) : ");
        radius = Convert.ToDouble(Console.ReadLine());
    }
    public override void SurfaceArea()
    {
        Console.WriteLine(" Surface Area of Cone is {0}", 4* baseArea(radius));
    }
    public override void Volume()
    {
        double volume;
        volume = (4.0 / 3) * (baseArea(radius)*radius );
        Console.WriteLine(" Volume of Sphere is {0}", volume);
    }
}

class AbstractClasseg
{
    public static void Main()
    {
        Solid s = new Cylinder(7);
        s.readRadius();
        s.SurfaceArea();
        s.Volume();
        s = new Cone(7);
        s.readRadius();
        s.SurfaceArea();
        s.Volume();
        s = new Sphere();
        s.readRadius();
        s.SurfaceArea();
        s.Volume();
        Console.ReadKey();
    }
}
```



```
file:///C:/Users/Dr.PV.Praveen Sundar/documents/visual studio 2010/Projects/Polymorphism/Polymorphism/bin/Debug/Polymorphis...
Enter the radius (Cylinder) : 5
Surface Area of Cylinder is 3736.9908
Volume of Cylinder is 5449.77825
Enter the radius (Cone) : 5
Surface Area of Cone is 1339.45043256441
Volume of Cone is 1816.59275
Enter the radius (Sphere) : 5
Surface Area of Cone is 3114.159
Volume of Sphere is 5190.265
```

Differences between overriding methods and abstract methods

45

- ❑ The concept of the abstract method is near similar to the concept of method overriding because in method overriding if a Parent class contains any virtual methods in it, then those methods can be re-implemented under the child class by using the override modifier.
- ❑ In a similar way, if a parent class contains any abstract methods in it, those abstract methods must be implemented under the child class by using the same override modifier.
- ❑ The main difference between method overriding and abstract method is in the case of method overriding the child class re-implementing the method is optional but in the case of the abstract method, the child class implementing the method is mandatory.

Sealed Class & Methods

46

- ❑ In C#, sealed is a keyword used to stop inheriting the particular class from other classes.
- ❑ Based on our requirements, it is possible to prevent overriding the particular properties or methods.
- ❑ Generally, while creating a particular class it is possible to inherit all the properties and methods in any class.
- ❑ If we want to restrict access to a defined class and its members, then by using a sealed keyword, we can prevent other classes from inheriting the defined class.
- ❑ In C#, a sealed class can define by using a **sealed** keyword.
- ❑ In C#, if we define a class with the sealed keyword, then we don't have a chance to inherit that particular class.

Points to be considered

47

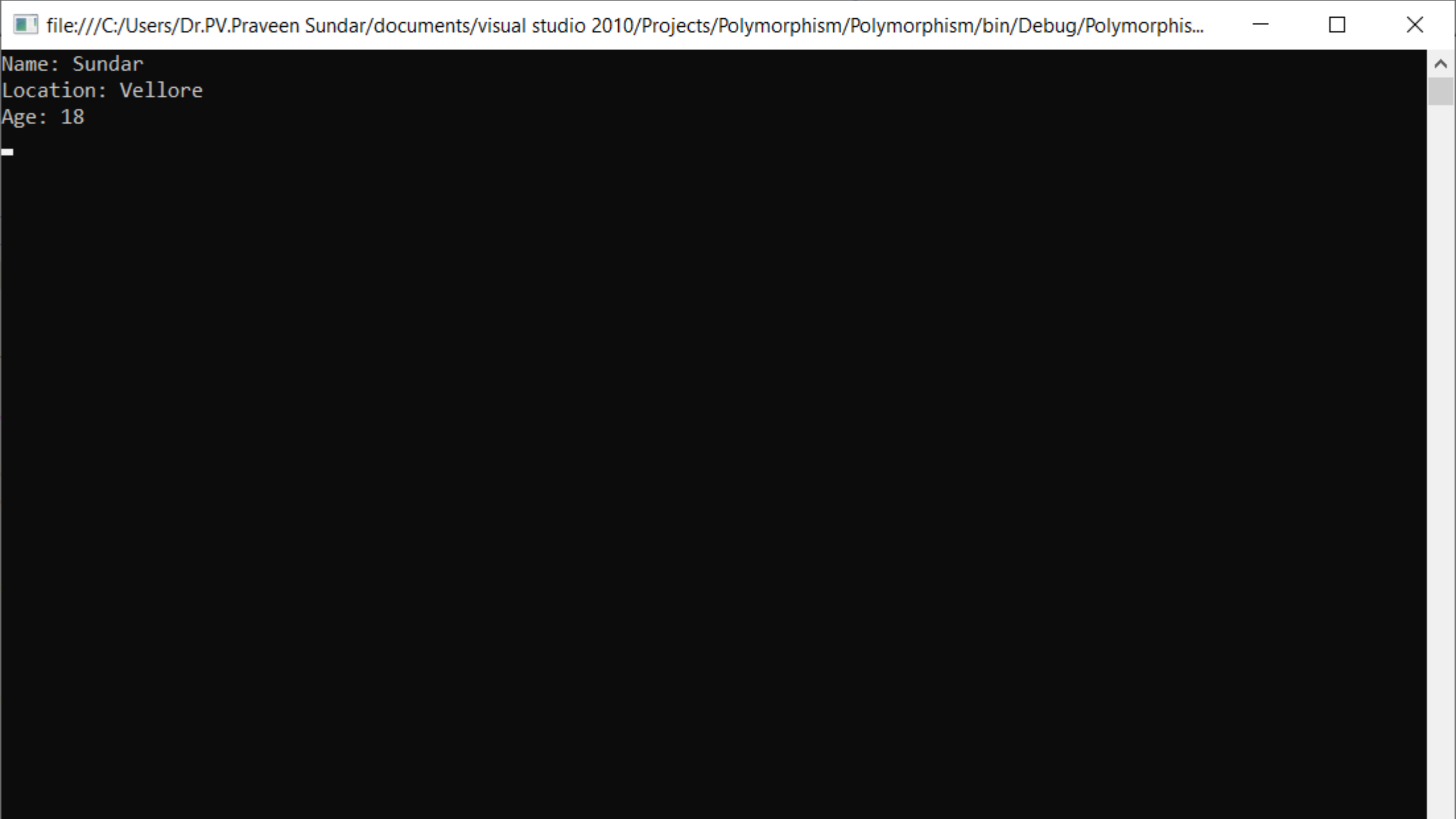
- ❑ A sealed class is completely opposite to an abstract class.
- ❑ This sealed class cannot contain abstract methods.
- ❑ It should be the bottom-most class within the inheritance hierarchy.
- ❑ A sealed class can never be used as a base class.
- ❑ The sealed class is specially used to avoid further inheritance.
- ❑ The keyword `sealed` can be used with classes, instance methods, and properties.
- ❑ Even if a sealed class cannot be inherited we can still consume the class members from any other class by creating the object of the class.

SealedClass.cs*

Polymorphism.Details

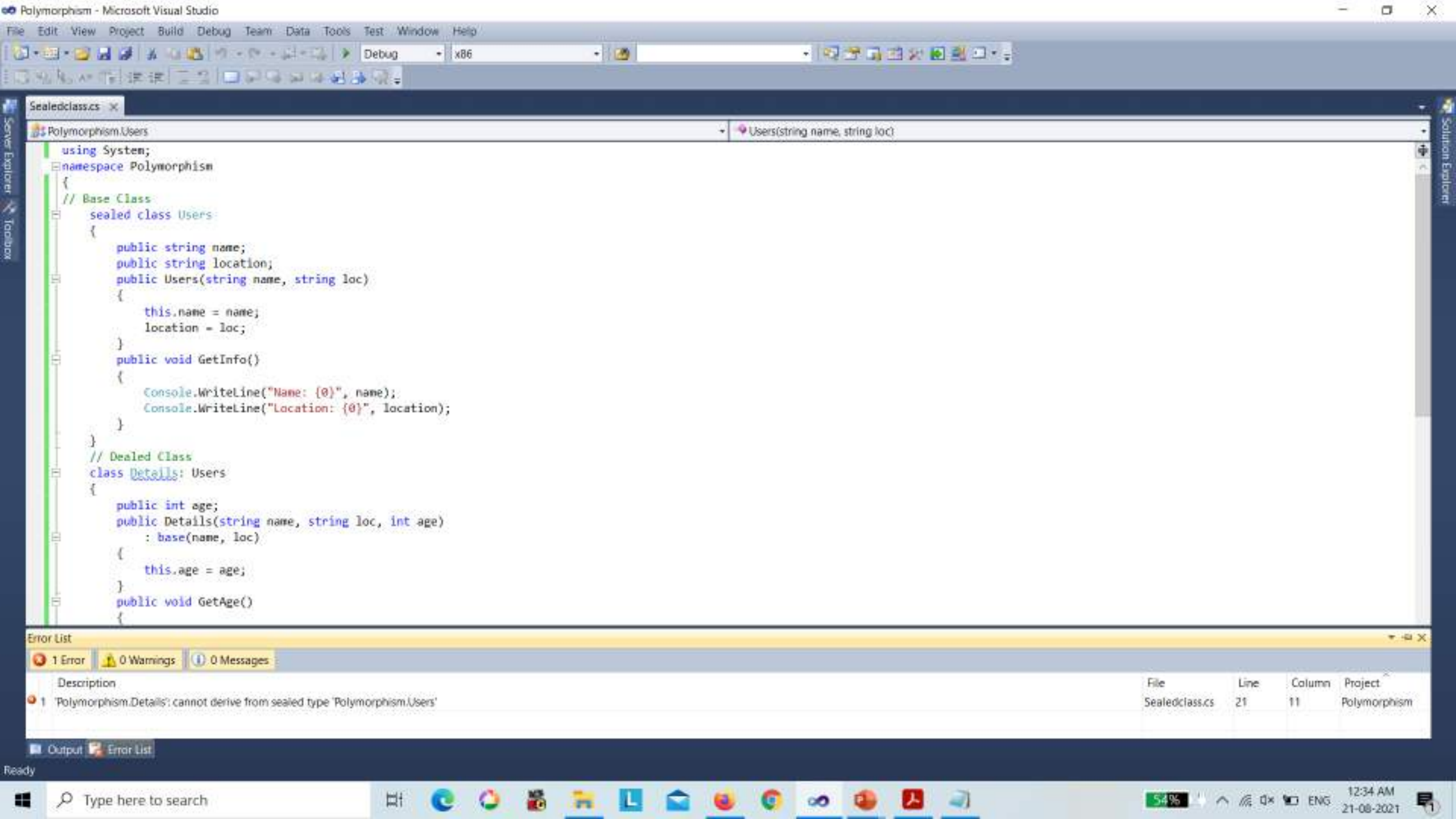
Main()

```
using System;
namespace Polymorphism
{
    // Base Class
    public class Users
    {
        public string name;
        public string location;
        public Users(string name, string loc)
        {
            this.name = name;
            location = loc;
        }
        public void GetInfo()
        {
            Console.WriteLine("Name: {0}", name);
            Console.WriteLine("Location: {0}", location);
        }
    }
    // Dealed Class
    class Details: Users
    {
        public int age;
        public Details(string name, string loc, int age)
            : base(name, loc)
        {
            this.age = age;
        }
        public void GetAge()
        {
            Console.WriteLine("Age: {0}", age);
        }
        public static void Main()
        {
            Details d = new Details("Sundar", "Vellore", 18);
            d.GetInfo();
            d.GetAge();
            Console.ReadKey();
        }
    }
}
```



Name: Sundar
Location: Vellore
Age: 18

```
using System;
namespace Polymorphism
{
    // Base Class
    sealed class Users
    {
        public string name;
        public string location;
        public Users(string name, string loc)
        {
            this.name = name;
            location = loc;
        }
        public void GetInfo()
        {
            Console.WriteLine("Name: {0}", name);
            Console.WriteLine("Location: {0}", location);
        }
    }
    // Dealed Class
    class Details: Users
    {
        public int age;
        public Details(string name, string loc, int age)
            : base(name, loc)
        {
            this.age = age;
        }
        public void GetAge()
        {
            Console.WriteLine("Age: {0}", age);
        }
        public static void Main()
        {
            Details d = new Details("Sundar", "Vellore", 18);
            d.GetInfo();
            d.GetAge();
            Console.ReadKey();
        }
    }
}
```



SealedClasses.cs

Polymorphism.Details

Main()

```
using System;
namespace Polymorphism
{
    // Base Class
    sealed class Users
    {
        public string name;
        public string location;
        public Users(string name, string loc)
        {
            this.name = name;
            location = loc;
        }
        public void GetInfo()
        {
            Console.WriteLine("Name: {0}", name);
            Console.WriteLine("Location: {0}", location);
        }
    }
    // Dealed Class
    class Details
    {
        public int age;
        public Details(string name, string loc, int age)
        {
            this.age = age;
        }
        public void GetAge()
        {
            Console.WriteLine("Age: {0}", age);
        }
        public static void Main()
        {
            Users u = new Users("Sundar", "Vellore");
            Details d = new Details("Sundar", "Vellore", 18);
            u.GetInfo();
            d.GetAge();
            Console.ReadKey();
        }
    }
}
```

100 %

Output

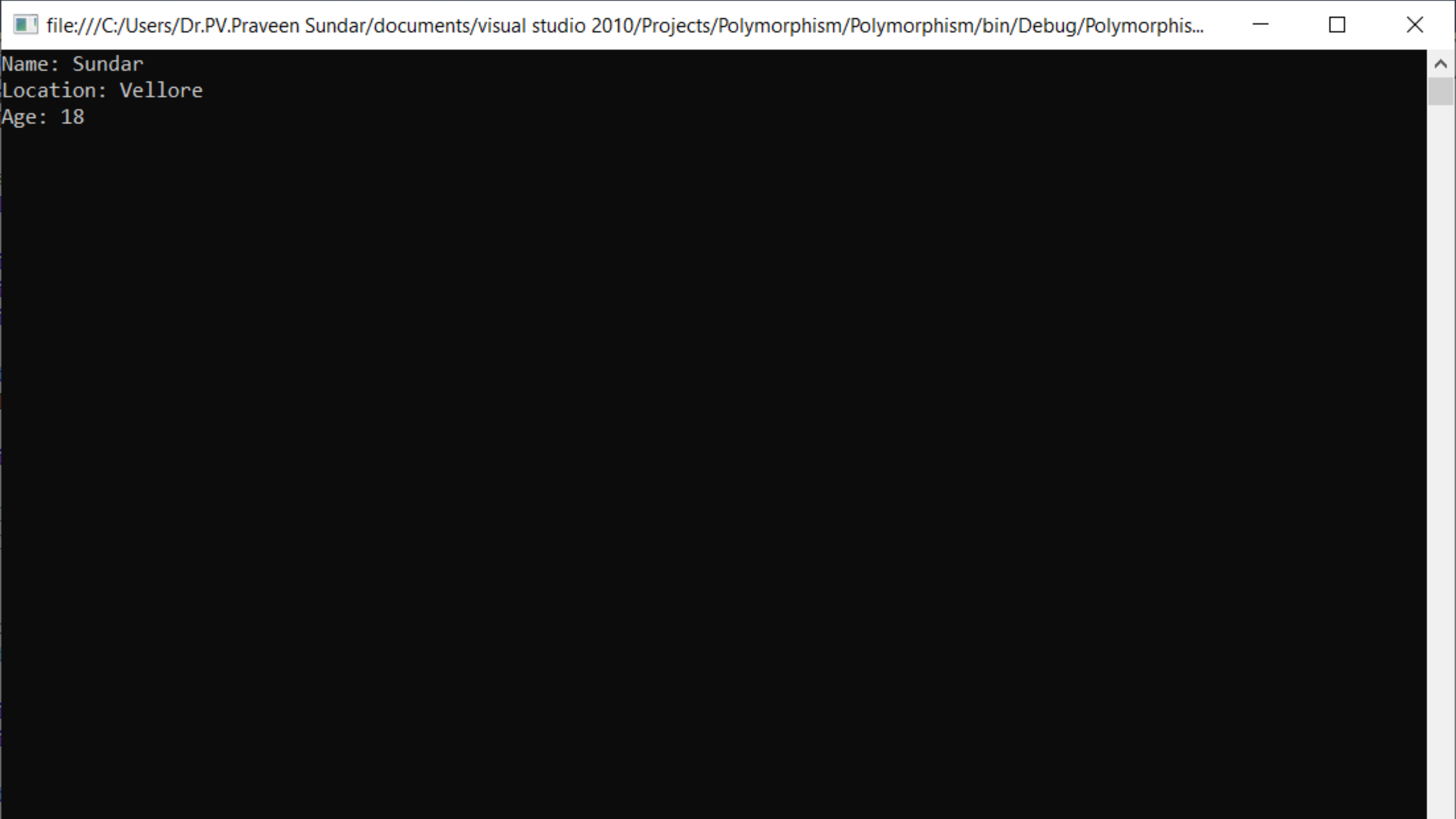
Build succeeded

Ln 42

Col 1

Ch 1

INS



Name: Sundar
Location: Vellore
Age: 18

Sealed Methods in C#

- ❑ In C#, we can also use the sealed keyword on a method or property that overrides a virtual method or property in a base class to allow other classes to derive from the base class and prevent them from overriding specific virtual methods or properties.
- ❑ If we don't want to allow subclasses to override the base class method and to ensure that all sub-classes use the same base class method logic then that method should be declared as sealed.
- ❑ The sealed method cannot be overridden in sub-classes violation leads to a compile-time error.
- ❑ The private method is not inherited whereas the sealed method is inherited but cannot be overridden in C#. So, a private method cannot be called from sub-classes whereas a sealed method can be called from sub-classes.

```
1 using System;
2 namespace Sealedclasseg
3 {
4     public class A
5     {
6         public virtual void GetInfo()
7         {
8             Console.WriteLine("Base Class A Method");
9         }
10        public virtual void Test()
11        {
12            Console.WriteLine("Base Class A Test Method");
13        }
14    }
15    public class B: A
16    {
17        public sealed override void GetInfo()
18        {
19            Console.WriteLine("Derived Class B Method");
20        }
21        public override void Test()
22        {
23            Console.WriteLine("Derived Class B Test Method");
24        }
25    }
```

```
26 public class C: B
27 {
28     // Compile time error
29     // public override void GetInfo()
30     // {
31     //     Console.WriteLine("Age: {0}", base.age);
32     // }
33     public override void Test()
34     {
35         Console.WriteLine("Derived Class C Test Method");
36     }
37 }
38 public class Program
39 {
40     public static void Main(string[] args)
41     {
42         C c = new C();
43         c.GetInfo();
44         c.Test();
45         Console.WriteLine("\nPress Enter Key to Exit..");
46         Console.ReadLine();
47     }
48 }
49 }
50
```

Derived Class B Method

Derived Class C Test Method

Press Enter Key to Exit..

...Program finished with exit code 0

Press ENTER to exit console.

ABSTRACT CLASS	SEALED CLASS
A class that contains one or more abstract methods is known as an abstract class.	A class from which it is not possible to derive a new class is known as a sealed class.
The abstract class can contain abstract and non-abstract methods.	The sealed class can contain non-abstract methods; it cannot contain abstract and virtual methods.
Creating a new class from an abstract class is compulsory to consume.	It is not possible to create a new class from a sealed class.
An abstract class cannot be instantiated directly; we need to create the object for its child classes to consume an abstract class.	We should create an object for a sealed class to consume its members.
We need to use the keyword abstract to make any class abstract.	We need to use the keyword sealed to make any class as sealed.
An abstract class cannot be the bottom-most class within the inheritance hierarchy.	The sealed class should be the bottom-most class within the inheritance hierarchy.

Partial Class and Methods

- ❑ Partial Classes are the new feature that has been added in C# 2.0 which allows us to define a class on multiple files i.e. we can physically split the content of the class into different files but even physically they are divided but logically it is one single unit only.
- ❑ A class in which code can be written in two or more files is known as a partial class.
- ❑ To make any class partial we need to use the keyword `partial`.
- ❑ Partial classes allow us to split a class definition into 2 or more files.
- ❑ It is also possible to split the definition of a struct or an interface over two or more source files.
- ❑ Each source file will contain a section of the class definition, and all parts are combined into a single class when the application is compiled.

Rules for Partial Classes

- ❑ All the partial class definitions must be in the same assembly and namespace.
- ❑ All the parts must have the same accessibility like public or private, etc.
- ❑ If any part is declared abstract, then the whole type is considered abstract.
- ❑ If any part is declared sealed, then the whole type is considered sealed.
- ❑ If any part declares a base type, then the whole type inherits that class.
- ❑ Any class member declared in a partial definition are available to all other parts.
- ❑ Different parts can have different base types and so the final class will inherit all the base types.
- ❑ The Partial modifier can only appear immediately before the keywords class, struct, or interface.

```
namespace HeightWeightInfo
{
    class File1
    {
    }
    public partial class Record
    {
        private int h;
        private int w;
        public Record(int h, int w)
        {
            this.h = h;
            this.w = w;
        }
    }
}
```

```
namespace HeightWeightInfo
{
    class File2
    {
    }
    public partial class Record
    {
        public void PrintRecord()
        {
            Console.WriteLine("Height:" + h);
            Console.WriteLine("Weight:" + w);
        }
    }
}
```

```
namespace HeightWeightInfo
{
    class Program
    {
        static void Main(string[] args)
        {
            Record myRecord = new Record(10, 15);
            myRecord.PrintRecord();
            Console.ReadLine();
        }
    }
}
```

Partial Methods

- ❑ A partial class may contain a partial method.
- ❑ One part of the class contains the signature of the method.
- ❑ An optional implementation may be defined in the same part or another part.
- ❑ If the implementation is not supplied, then the method and all calls are removed at compile time.
- ❑ The implementation can be provided in the same physical file or in another physical file that contains the partial class.

```
public partial class Car
{
    partial void InitializeCar();
    public void BuildRim() { }
    public void BuildWheels() { }
}
```

```
public partial class Car
{
    public void BuildEngine() { }
    partial void InitializeCar()
    {
        string str = "Car";
    }
}
```

Interface

66

- ❑ The Interface in C# is a fully un-implemented class used for declaring a set of operations of an object. So, we can define an interface as a pure abstract class which allows us to define only abstract methods. The abstract method means a method without body or implementation.
- ❑ An interface is defined as a syntactical contract that all the classes inheriting the interface should follow. The interface defines the 'what' part of the syntactical contract and the deriving classes define the 'how' part of the syntactical contract.
- ❑ In C#, the interface is same as a class, but the only difference is class can contain both declarations and implementation of methods, properties, and events, but the interface will contain only the declarations of methods, properties, and events that a class or struct can implement.

- ❑ The class or struct that implements an interface must provide an implementation for all the members specified in the interface definition.
- ❑ Generally, C# will not support multiple inheritance of classes, but that can achieve by using an interface.
- ❑ Also, a structure in C# cannot be inherited from another structure or class, but that can inherit by using interfaces.
- ❑ If we define an abstract class in place of an interface, a service provider cannot implement multiple specifications so that the service provider cannot have multiple businesses.
- ❑ In C#, we can define an interface by using **interface** keyword.

- ❑ Following is the example of defining an interface using interface keyword.

```
// SYNTAX:  
public interface InterfaceName  
{  
    //only abstract methods  
}  
  
// For example  
public interface Example  
{  
    void show();  
}
```

- ❑ Here the keyword `interface` tells that `Example` is an interface containing one abstract method i.e. `show()`.
- ❑ By default, the members of an interface are public and abstract.
- ❑ An interface can contain
 - ❑ **Abstract methods**
 - ❑ **Properties**
 - ❑ **Indexes**
 - ❑ **Events**
- ❑ **An interface cannot contain**
 - ❑ **Non-abstract functions**
 - ❑ **Data fields**
 - ❑ **Constructors**
 - ❑ **Destructors**

While working with an interface, we must follow the below rules.

- ❑ The interface cannot have concrete methods, violation leads to Compilation Error: interface methods cannot have a body.
- ❑ We cannot declare interface members as private or protected members violation leads to Compilation Error :”
- ❑ An interface cannot be instantiated but its reference variable can be created for storing its subclass object reference.
- ❑ We cannot declare the interface as sealed it leads to Compilation Error : “illegal combination of modifier interface and final”.
- ❑ The class derived from the interface should implement all abstract methods of the interface otherwise it should be declared as abstract else it leads to a compile-time error.
- ❑ The subclass should implement the interface method with public keyword because interface methods default accessibility modifier is public.
- ❑ In an interface, we cannot create fields, variable violation leads to a compile-time error.

- An interface is different from a class in the following ways:
 - ▣ We cannot instantiate an interface.
 - ▣ An interface does not contain any constructor or data fields or destructor, etc.
 - ▣ All of the methods of an interface are abstract and public by default.
 - ▣ An interface is not extended by a class; it is implemented by a class.
 - ▣ An interface can extend multiple interfaces.
- An interface is similar to an abstract class in the following ways
 - ▣ Both interface and the abstract class cannot be instantiated means we cannot create the object.
 - ▣ But we can create a reference variable for both interface and abstract class.
 - ▣ The subclass should implement all abstract methods.
 - ▣ Both cannot be declared as sealed.



Polymorphism Interfaceeg.cs* AbstractClass.cs AbstractClass.cs Overriding.cs Program.cs

Polymorphism.Interfaceeg Div(int a, int b)

```
using System;

namespace Polymorphism
{
    interface Addition
    {
        int Add(int a, int b);
    }
    interface Subtraction
    {
        int Sub(int a, int b);
    }
    interface Multiplication
    {
        int Mul(int a, int b);
    }
    interface Division
    {
        float Div(int a, int b);
    }
    class Interfaceeg : Addition, Subtraction, Multiplication, Division
    {
        public int Add(int a, int b)
        {
            return (a + b);
        }
        public int Sub(int a, int b)
        {
            return (a - b);
        }
        public int Mul(int a, int b)
        {
            return (a * b);
        }
    }
}
```

Solution Explorer

Solution 'Polymorphism' (1 project)

- Polymorphism
 - Properties
 - References
 - AbstractClass.cs
 - AbstractClass.cs
 - Interfaceeg.cs
 - OperatorOverload.cs
 - Overriding.cs
 - Program.cs

Properties



Polymorphism Interfaceeg.cs* AbstractClass.cs AbstractClass.cs Overriding.cs Program.cs

```
Polymorphism.Interfaceeg
Div(int a, int b)
{
}

public float Div(int a, int b)
{
    float temp = (float)a / (float)b;
    return (temp);
}

public static void Main(string[] args)
{
    int val1, val2;
    Console.Write("Enter the Value of A:");
    val1 = Convert.ToInt32(Console.ReadLine());
    Console.Write("Enter the Value of B:");
    val2 = Convert.ToInt32(Console.ReadLine());
    Interfaceeg obj1 = new Interfaceeg();
    Console.WriteLine(" Addition of {0},{1} is {2}", val1, val2, obj1.Add(val1, val2));
    Console.WriteLine(" Subtraction of {0},{1} is {2}", val1, val2, obj1.Sub(val1, val2));
    Console.WriteLine(" Multiplication of {0},{1} is {2}", val1, val2, obj1.Mul(val1, val2));
    Console.WriteLine(" Division of {0},{1} is {2}", val1, val2, obj1.Div(val1, val2));
    Console.ReadKey();
}
```

Solution Explorer

Solution 'Polymorphism' (1 project)

- Polymorphism
 - Properties
 - References
 - AbstractClass.cs
 - AbstractClass.cs
 - Interfaceeg.cs
 - OperatorOverload.cs
 - Overriding.cs
 - Program.cs

Properties

file:///C:/Users/Dr.PV.Praveen Sundar/Documents/Visual Studio 2010/Projects/Polymorphism/Polymorphism/bin/Debug/Polymorphis...

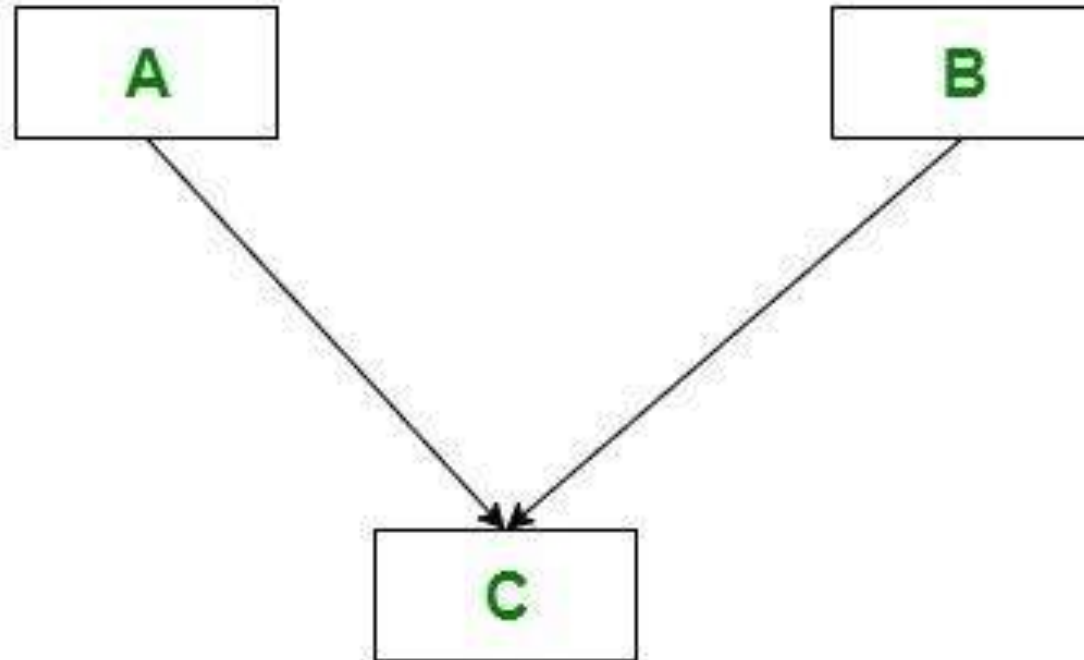
Enter the Value of A:30
Enter the Value of B:25
Addition of 30,25 is 55
Subtraction of 30,25 is 5
Multiplication of 30,25 is 750
Division of 30,25 is 1.2

Abstract class	Interface
It is a partially implemented class. It allows us to define both concrete and abstract methods.	It is a fully un-implemented class. It allows us to define only abstract methods.
It provides both reusability and forcibility	It provides the only forcibility
It should be declared as abstract by using the abstract keyword, abstract methods should also contain the abstract keyword.	It should be created by using the keyword interface. Declaring its methods as abstract is optional because by default the methods of an interface are abstract.
A class that contains one or more abstract functions is called abstract class.	The class which contains all the abstract functions is known as an interface.
Its member's default accessibility modifier is private and can be changed to any of the other accessibility modifiers.	Its member's default accessibility modifier is public and cannot be changed.
It is possible to declare data fields in an abstract class.	But it is not possible to declare any data fields in an interface.
An abstract class can contain the non-abstract function.	An interface cannot contain non-abstract functions.
An abstract class can inherit from another abstract class or from an interface.	An interface can inherit from only other interfaces but cannot inherits from the abstract class.
An abstract class cannot be used to implement multiple inheritances.	An interface can be used to implement multiple inheritances.
Abstract class members can have access modifiers.	Interface members cannot have access modifiers.

Multiple Inheritance

76

- In Multiple inheritance, one class can have more than one superclass and inherit features from all its parent classes.
- C# doesn't allow multiple inheritance with classes but it can be implemented using interface. The reason behind is:
 - ▣ Multiple inheritance add too much complexity with little benefit.
 - ▣ There are huge chances of conflicting base class member. For example, if there is a method calculate() in two base class and both are doing different calculation. What happened if user call calculate() in child class? Which method will work?
 - ▣ Inheritance with Interface provides same job of multiple inheritance.
 - ▣ Multiple Inheritance inject a lots of burden into implementation and it cause slow program execution.



Multiple Inheritance

- ❑ An interface can inherit from one or more other interfaces.
- ❑ An interface inherits all members of its base interfaces.
- ❑ A class that implements an interface also implicitly all of the interface's base interfaces.
- ❑ The Syntax for multiple inheritance of an interface varies only in the interface implementation as given below.

```
accessModifier class ClassName: baselist
{
    Method implementation
    Member declarations of the class
    Public static void Main()
    {.....}
}
```

- ❑ The list of interfaces following the colon(:) in the class declaration is known as baselist.
- ❑ It is possible to include only one class in the baselist.
- ❑ When the baselist of a class contains a base class and interfaces, the base class must be written first in the base list.
- ❑ Maximum of one class and any number of interfaces are allowed in the baselist.

```
using System;
namespace Polymorphism
{
    interface IKitchen
    {
        void Store();
    }
    interface IDrawingHall
    {
        void ShowCase();
    }
    interface IBedRoom
    {
        void acProvision();
    }
    class MultipleInheritance: IKitchen, IDrawingHall , IBedRoom
    {
        public void Store()
        {
            Console.WriteLine("Place the Groceries on the Shelf");
        }
        public void ShowCase()
        {
            Console.WriteLine("Keep the prizes, Momento and medals in the showcase");
        }
        public void acProvision()
        {
            Console.WriteLine("Provide an Airconditioner in the bed room");
        }
        public static void Main()
        {
            MultipleInheritance obj = new MultipleInheritance();
            obj.Store();
            obj.ShowCase();
            obj.acProvision();
            IKitchen objk = new MultipleInheritance();
            objk.Store();
            IDrawingHall objD = obj;
            objD.ShowCase();
            IBedRoom objB = new MultipleInheritance();
            objB.acProvision();
            Console.ReadKey();
        }
    }
}
```

Place the Groceries on the Shelf
Keep the prizes, Memento and medals in the showcase
Provide an Airconditioner in the bed room
Place the Groceries on the Shelf
Keep the prizes, Memento and medals in the showcase
Provide an Airconditioner in the bed room

DELEGATES

Dr P.V. Praveen Sundar
Assistant Professor,
Department of Computer Science
Adhiparasakthi College of Arts & Science,
Kalavai.

- ❑ The dictionary of delegate is "a person acting for another person". In C#, it really means a method acting for another method.
- ❑ A delegate is another reference type introduced in C#.
- ❑ An instance of a delegate encapsulates a reference to a method.
- ❑ A delegate refers to either an instance method or a static method.
- ❑ Delegates always refer to methods. They are defined at run-time.
- ❑ By using a delegate, a method may be passed as an argument to another method. A method that is passed to another method as an argument is called a callback method or callback function.
- ❑ Callback functions are used to pass a function as an argument in other languages.
- ❑ A delegate can be declared in a class or an interface
- ❑ All delegates are implicitly derived from the `System.Delegate` class.

- ❑ Creating and using delegates involve four steps. They include:
 - ❑ Delegate Declaration
 - ❑ Delegate Method Definition
 - ❑ Delegate Instantiation
 - ❑ Delegate Invocation
- ❑ Delegates methods are any functions(defined in a class) whose signatures matches the delegate signature exactly.
- ❑ The delegate instance holds the reference to the delegate methods.
- ❑ The instance is used to invoke the methods indirectly.
- ❑ An important feature if a delegate is that it can be used to hold reference to a method of any class. The only requirement is that its signature must match the signature of the method.

Rules of using Delegates in C#

85

- ❑ A delegate in C# is a user-defined type and hence before invoking a method using a delegate, we must have to define that delegate first.
- ❑ The signature of the delegate must match the signature of the method, the delegate points to otherwise we will get a compiler error. This is the reason why delegates are called type-safe function pointers.
- ❑ A Delegate is similar to a class. This means we can create an instance of a delegate and when we do so, we need to pass the method name as a parameter to the delegate constructor, and it is the function the delegate will point to
- ❑ Delegates syntax looks very much similar to a method with a delegate keyword.

Declaration of Delegate

86

- A delegate can be declared using the delegate keyword followed by a function signature, as shown below.

Delegate Syntax

```
[access modifier] delegate [return type] [delegate name]([parameters])
```

- Where
 - modifier** represents public, protected, internal, private and new.
 - delegate** is the keyword representing the declaration as a delegate.
 - return type** indicates the return type of the delegate
 - Delegate name** is any valid identifier and is the name of the delegate that will be used to instantiate delegate objects.
 - Parameters** identifies the signature of the delegate

- ❑ The new modifier is only permitted on delegates declared within another type, in which case it specifies that such a delegate hides an inherited member by the same name.
- ❑ The method to be passed must be declared as a normal method with the same return type and parameter list. It can be either static or instance method.
- ❑ Since it is a class type, it can be defined in any place where a class definition is permitted. That is, a delegate may be defined in the following places:
 - ❑ Inside a class
 - ❑ Outside all classes
 - ❑ As the top level object in a namespace

Depending on how visible we want the delegate to be, we can apply any of the visibility modifiers to the delegate definition.

Instantiation & Invocation of Delegates

88

- Once a delegate type is declared, a delegate object must be created with the new keyword and be associated with a particular method.
- When creating a delegate, the argument passed to the new expression is written similar to a method call, but without the arguments to the method. This is known as invoking the delegate.
- Syntax:
`[delegate_name] [instance_name] = new [delegate_name](expression);`
- The delegate-name is the name of the delegate declared earlier whose object is to be created .
- The expression must be a method name or a value of a delegate-type.
- If it is a method name its signature and return type must be the same as those of the delegate.
- If no matching method exists, or more than one matching method exists, an error occurs.

- ❑ The matching method may be either an instance method or a static method.
- ❑ If it is an instance method, we need to specify the instance as well as the name of the method.
- ❑ If it is a static one, then it is enough to specify the class name and the method name.
- ❑ The method and the object to which a delegate refers are determined when the delegate is instantiated and then remain constant for the entire lifetime of the delegate. It is, therefore, not possible to change them, once the delegate is created.
- ❑ It is also not possible to create a delegate that would refer to a constructor, indexer, or user-defined operator.

Invoking a delegate

90

- ❑ C# uses a special syntax for invoking a delegate.
- ❑ A delegate can be invoked using the Invoke() method or using the () operator.
- ❑ When a delegate is invoked, it in turn invokes the method whose reference has been encapsulated into the delegate, (only if their signatures match).
- ❑ Invocation takes the following form:
`delegate_object (parameters list)`
- ❑ The optional parameters list provides values for the parameters of the method to be used.
- ❑ If the invocation invokes a method that returns void, the result is nothing and therefore it cannot be used as an operand of any operator. It can be simply a statement_expression.

```
public delegate void MyDelegate(string msg); // declare a delegate

// set target method
MyDelegate del = new MyDelegate(MethodA);
// or
MyDelegate del = MethodA;
// or set lambda expression
MyDelegate del = (string msg) => Console.WriteLine(msg);

// target method
static void MethodA(string message)
{
    Console.WriteLine(message);
}
```



```
using System;

public delegate void MyDelegate(string msg); // Step-1 Declaring Delegate
public delegate int NumberDelegate(int n);
public delegate int StaticDelegate();

namespace Delegateeg
{
    class ClassA
    {
        static int num = 10;
        public void MethodA(string message) // Step-2 - Giving The body of the functions
        {
            Console.WriteLine(message+ " Called ClassA.MethodA() : ");
        }
        public static string MethodC(string message)
        {
            return ("Welcome" + message);
        }
        public static int AddNum(int p)
        {
            num += p;
            return num;
        }
        public static int SubNum(int p)
        {
            num -= p;
            return num;
        }
        public static int MultNum(int q)
        {
            num *= q;
            return num;
        }
        public static int getNum()
        {
            return num;
        }
    }
}
```

```
class ClassB
{
    static int num;
    public static void MethodB(string message)
    {
        Console.WriteLine(message+ " Called ClassB.MethodB()  : ");
    }
    public static int AddNum()
    {
        num ++;
        return num;
    }
    public static int SubNum()
    {
        num --;
        return num;
    }
    public static int MultNum(int q)
    {
        num *= q+20;
        return num;
    }
}
```

```
class Program
{
    static void Main(string[] args)
    {
        ClassA objA = new ClassA();
        MyDelegate obj = new MyDelegate(objA.MethodA); // Step-3 Creating an Object for Delegate
        obj("Object 1"); // Step-4 Invoking a Delegate
        obj = ClassB.MethodB;
        obj.Invoke("Object 2");
        obj = (string msg) => Console.WriteLine(msg + "MCA II Year Students");
        obj.Invoke("Object 3");
        NumberDelegate obj1 = new NumberDelegate(ClassA.AddNum);
        Console.WriteLine("The value = {0}", obj1.Invoke(25));
        NumberDelegate obj2 = new NumberDelegate(ClassA.MultNum);
        Console.WriteLine("The value = {0}", obj1.Invoke(25));
        obj2 = ClassA.SubNum;
        Console.WriteLine("The value = {0}", obj2.Invoke(25));
        obj2 = ClassB.MultNum;
        Console.WriteLine("The value = {0}", obj2.Invoke(25));
        StaticDelegate obj3 = new StaticDelegate(ClassB.AddNum);
        Console.WriteLine("The value = {0}", obj3.Invoke());
        Console.WriteLine("The value = {0}", obj3.Invoke());
        Console.WriteLine("The value = {0}", obj3.Invoke());
        Console.ReadKey();
    }
}
```

```
Object 1 Called ClassA.MethodA() :  
Object 2 Called ClassB.MethodB() :  
Object 3MCA II Year Students  
The value = 35  
The value = 60  
The value = 35  
The value = 0  
The value = 1  
The value = 2  
The value = 3
```

Types of Delegates

96

- The Delegates in C# are classified into two types such as
- **Single Cast Delegate:** If a delegate is used for invoking a single method then it is called a single cast delegate or unicast delegate. In other words, we can say that the delegates that represent only a single function are known as single cast delegates.
- **Multicast Delegate:** If a delegate is used for invoking multiple methods then it is known as the multicast delegate. Or the delegates that represent more than one function are called Multicast delegates.
- Delegate objects can be composed using the "+" operator. A composed delegate calls the two delegates it was composed from. Only delegates of the same type can be composed. The "-" operator can be used to remove a component delegate from a composed delegate.
- Using this property of delegates you can create an invocation list of methods that will be called when a delegate is invoked. This is called multicasting of a delegate.



Eventseg.cs Multidelegate.cs Program.cs

Delegateeg.ClassC SubNum()

```
using System;
public delegate int MultiDelegateeg();
public delegate void strDelegate(string msg);
namespace Delegateeg
{
    class ClassC
    {
        static int num;
        public static void MethodA(string message) // Step-2 - Giving The body of the functions
        {
            Console.WriteLine(message + " Called ClassA.MethodA() : ");
        }
        public static void MethodB(string message) // Step-2 - Giving The body of the functions
        {
            Console.WriteLine(message + " Called ClassA.MethodB() : ");
        }

        public static int AddNum()
        {
            num+=10;
            return num;
        }
        public static int IncNum()
        {
            num++;
            return num;
        }
        public static int SubNum()
        {
            num--;
            return num;
        }
    }
}
```

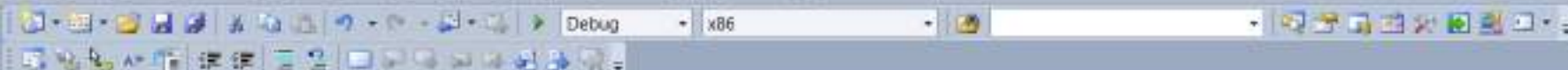
Solution Explorer

Solution 'Delegateeg' (1 project)

- Delegateeg
 - Properties
 - References
 - Eventseg.cs
 - Multidelegate.cs
 - Program.cs

Solut... Team... Class...

Properties



Eventseg.cs Multidelegate.cs* Program.cs

Delegateeg.Multidelegate Main(string[] args)

```
class Multidelegate
{
    static void Main(string[] args)
    {
        strDelegate obj1 = ClassC.MethodA;
        strDelegate obj2 = ClassC.MethodB;
        strDelegate obj3, obj4;
        obj3 = obj1 + obj2;
        obj3("Hello World");
        obj4 = obj3 - obj1;
        obj4("Hello World");

        MultiDelegateeg obj5 = ClassC.AddNum;
        MultiDelegateeg obj6 = ClassC.IncNum;
        MultiDelegateeg obj7, obj8;
        obj7 = obj5 + obj6;
        Console.WriteLine("The value = {0}", obj7.Invoke());
        obj8 = obj5 - obj6;
        Console.WriteLine("The value = {0}", obj8.Invoke());
        Console.ReadKey();
    }
}
```

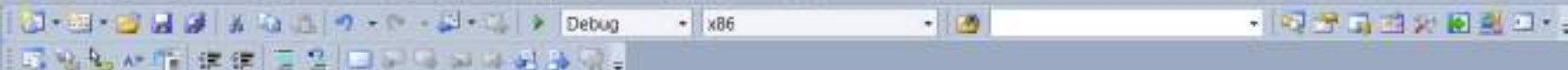
Solution Explorer

Solution 'Delegateeg' (1 project)

- Delegateeg
 - Properties
 - References
 - Eventseg.cs
 - Multidelegate.cs
 - Program.cs

Solut... Team... Class...

Properties



Eventseg.cs Multidelegate.cs* Program.cs

Delegateeg.Multidelegate Main(string[] args)

```
class Multidelegate
{
    static void Main(string[] args)
    {
        strDelegate obj1 = ClassC.MethodA;
        strDelegate obj2 = ClassC.MethodB;
        strDelegate obj3, obj4;
        obj3 = obj1 + obj2;
        obj3("Hello World");
        obj4 = obj3 - obj1;
        obj4("Hello World");

        MultiDelegateeg obj5 = ClassC.AddNum;
        MultiDelegateeg obj6 = ClassC.IncNum;
        MultiDelegateeg obj7, obj8;
        obj7 = obj5 + obj6;
        Console.WriteLine("The value = {0}", obj7.Invoke());
        obj8 = obj5 - obj6;
        Console.WriteLine("The value = {0}", obj8.Invoke());
        Console.ReadKey();
    }
}
```

Solution Explorer

Solution 'Delegateeg' (1 project)

- Delegateeg
 - Properties
 - References
 - Eventseg.cs
 - Multidelegate.cs
 - Program.cs

Solut... Team... Class...

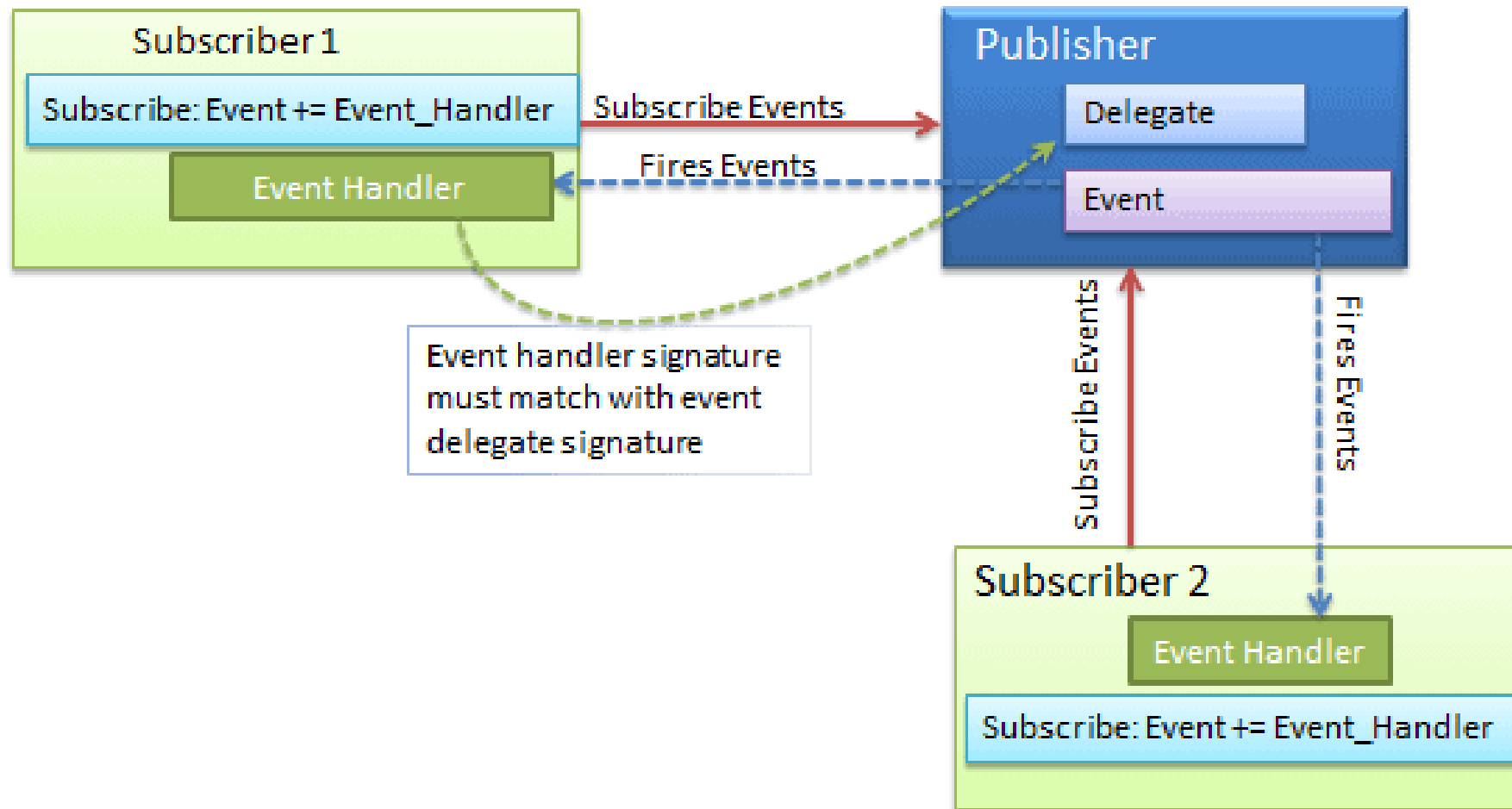
Properties

Events

100

- ❑ Events are nothing just a user action. For example
 - ❑ When you click with the mouse – It is mouse click events.
 - ❑ When you press any key in keyboard – It is KeyPress events
 - ❑ When you refresh your webpage – It is page load events
 - ❑ When you move mouse cursor – It is mouse hover events etc.
- ❑ So when you take any action like a key press, mouse movements, clicks etc an event raised. Let me clear more about it. For example, you filled an online form and click on submit button.
 - ❑ In the background `button_click()` event raised.
 - ❑ This event calls and execute an associated function `Submit_Click()`.
 - ❑ This function processes your request and submits page information to database.

- ❑ In C#, the event is a message sent by an object to indicate that particular action will happen. The action could be caused either by a button click, mouse movements, or other programming logic. The object that raises an event is called an event sender.
- ❑ In simple words, we can say that events are used to signal user actions such as button click, mouse over, menu selection, etc., to the user interface.
- ❑ The class who raises events is called **Publisher**, and the class who receives the notification is called **Subscriber**. There can be multiple subscribers of a single event. Typically, a publisher raises an event when some action occurred. The subscribers, who are interested in getting a notification when an action occurred, should register with an event and handle it.
- ❑ In C#, an event is an encapsulated delegate. It is dependent on the delegate. The delegate defines the signature for the event handler method of the subscriber class.



- ❑ In C#, events are used to enable a class or object to notify other classes or objects about the action that is going to happen.
- ❑ To declare an event, we need to use **event** keyword with delegate type.
- ❑ An Event has no return type and it is always void.
- ❑ All events are based on delegates.
- ❑ All the published events must have a listening object.
- ❑ Before raising an event, we need to check whether an event is subscribed or not.
- ❑ By using += operator, we can subscribe to an event, and by using -= operator, we can unsubscribe from an event.
- ❑ To raise an event, we need to invoke the event delegate.

- ❑ To respond to an event, we can define an event handler method in the event receiver, and the handler method must have the same signature of delegate in the event.
- ❑ To raise events, there must be subscribers; otherwise, they won't be raised.
- ❑ In C#, an event can have multiple subscribers, and a subscriber can handle multiple events from multiple publishers.
- ❑ In case an event has multiple subscribers, then event handlers are invoked synchronously when an event is raised.
- ❑ In C#, the publisher determines when an event is raised, and the subscriber determines what action is taken in response to the event.
- ❑ In .NET Framework, events are based on EventHandler delegate and an EventArgs base class.

Events Declaration

105

- In C#, events are the encapsulated delegates, so first, we need to define a delegate before we declare an event inside of a class by using event keyword.

Modifier event delegateType eventVariable

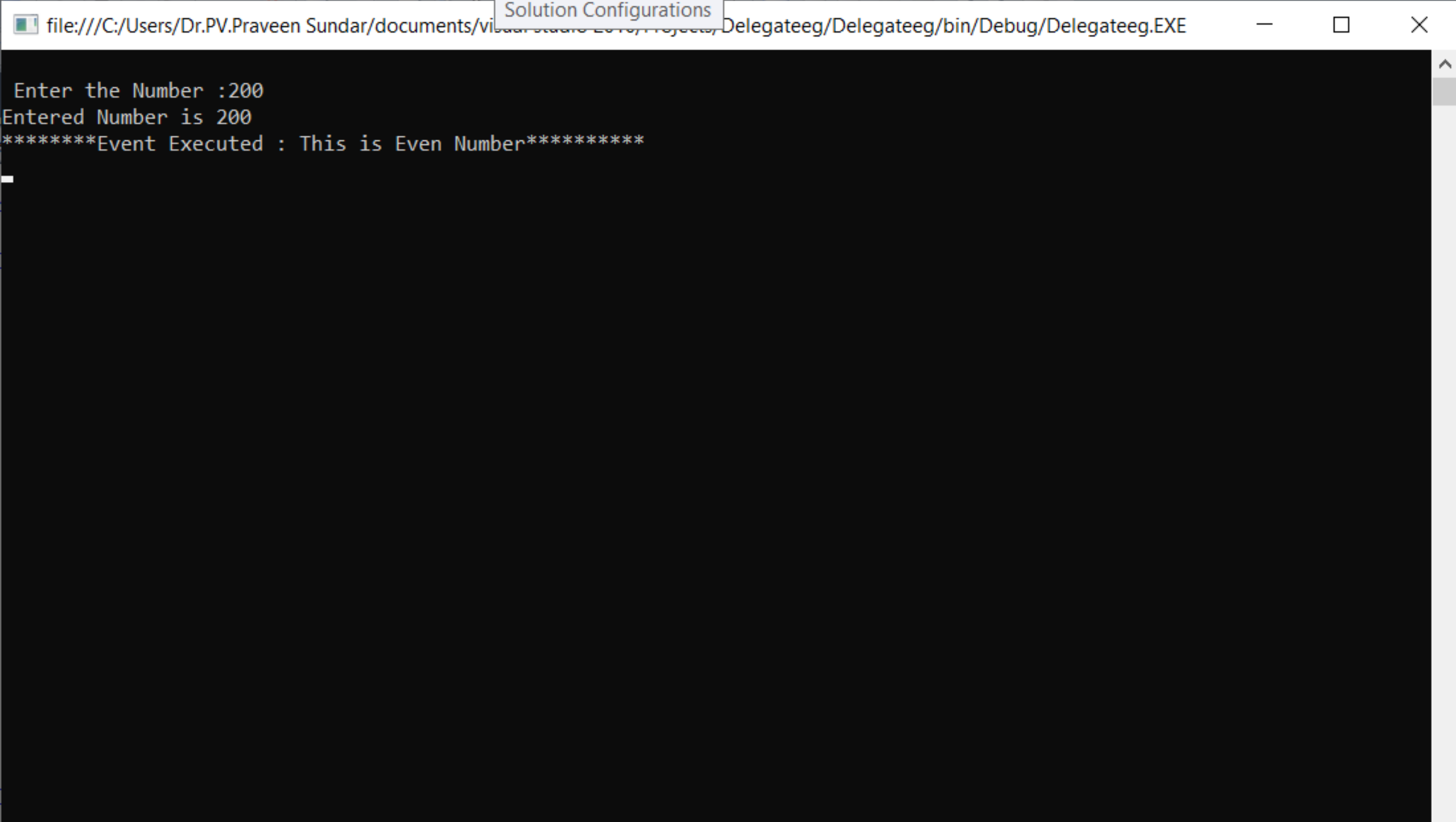
Where

- ▣ **Modifier** maybe new public, protected, private, internal, static, virtual, sealed, override, abstract.
- ▣ **Event** is a keyword representing an event declaration
- ▣ **delegateType** represents an event declaration
- ▣ **eventVariable** represents an identifier to the name of the event.

Keyword	Description
static	Makes the event available to callers at any time, even if no instance of the class exists.
virtual	Allows derived classes to override the event behavior by using the override keyword.
sealed	Specifies that for derived classes it is no longer virtual.
abstract	The compiler will not generate the add and remove event accessor blocks and therefore derived classes must provide their own implementation.

- ❑ To respond to an event, we need to define an event handler method in the event receiver, and this method signature must match with the event delegate's signature.
- ❑ In the event handler, you can perform actions that are required whenever the event is raised, such as getting the user input after a click on the button.


```
using System;
namespace Delegateeg
{
    class AddTwoNumbers
    {
        public delegate void dg_OddNumber(); //Declared Delegate
        public event dg_OddNumber ev_OddNumber; //Declared Events
        public event dg_OddNumber ev_EventNumber;
        public void Add()
        {
            int number;
            Console.WriteLine("\n Enter the Number :");
            number = Convert.ToInt32(Console.ReadLine());
            Console.WriteLine("Entered Number is {0}", number.ToString());
            if ((number % 2 != 0) && (ev_OddNumber != null))
            {
                ev_OddNumber(); //Raised Event
            }
            else
            {
                ev_EventNumber();
            }
        }
    }
    class Eventseg
    {
        static void Main(string[] args)
        {
            AddTwoNumbers a = new AddTwoNumbers();
            a.ev_OddNumber += new AddTwoNumbers.dg_OddNumber(OddMessage); //Event gets binded with delegates
            a.ev_EventNumber += new AddTwoNumbers.dg_OddNumber(EvenMessage);
            a.Add();
            Console.ReadLine();
        }
        //Delegates calls this method when event raised.
        static void OddMessage()
        {
            Console.WriteLine("*****Event Executed : This is Odd Number*****");
        }
        static void EvenMessage()
        {
            Console.WriteLine("*****Event Executed : This is Even Number*****");
        }
    }
}
```



Enter the Number :200

Entered Number is 200

*****Event Executed : This is Even Number*****

file:///C:/Users/Dr.PV.Praveen Sundar/documents/visual studio 2010/Projects/Delegateeg/Delegateeg/bin/Debug/Delegateeg.EXE

Enter the Number :13
Entered Number is 13
*****Event Executed : This is Odd Number*****

Built in Delegates in C#

111

- Microsoft provides two built in delegates to work with.
`public delegate void EventHandler(object sender, EventArgs e);`
`public delegate void EventHandler<TEventArgs>(object sender, TEventArgs e);`
- These built in delegates helps you to write event handling code with ease.
- With these delegates you can pass one or more than one value to event handler.
- When you raise event you must follow the discipline and pass required parameters to delegates.

```
using System;
namespace Delegateeg
{
    class Numbers
    {
        public event EventHandler<EventArgs> bev_OddNumber; //Declared Events
        public event EventHandler<EventArgs> bev_EvenNumber;
        public void Add()
        {
            int number;
            Console.WriteLine("\n Enter the Number :");
            number = Convert.ToInt32(Console.ReadLine());
            Console.WriteLine("Entered Number is {0}", number.ToString());
            if ((number % 2 != 0) && (bev_OddNumber != null))
            {
                bev_OddNumber(this, EventArgs.Empty); //Raised Event
            }
            else
            {
                bev_EvenNumber(this, EventArgs.Empty);
            }
        }
    }
    class BuiltInEvents
    {
        static void Main(string[] args)
        {
            Numbers a = new Numbers();
            a.bev_OddNumber += OddMessage; //Event gets binded with delegates
            a.bev_EvenNumber += EvenMessage;
            a.Add();
            Console.ReadKey();
        }
        //Delegates calls this method when event raised.
        static void OddMessage(object sender, EventArgs e)
        {
            Console.WriteLine("*****Event Executed : This is Odd Number*****");
        }
        static void EvenMessage(object sender, EventArgs e)
        {
            Console.WriteLine("*****Event Executed : This is Even Number*****");
        }
    }
}
```

EXCEPTION HANDLING

Dr P.V. Praveen Sundar
Assistant Professor,
Department of Computer Science
Adhiparasakthi College of Arts & Science,
Kalavai.

Exceptions

114

- ❑ When we write and execute our code in the .NET framework then there is a possibility of two types of error occurrences they are
 - ❑ Compilation errors
 - ❑ Runtime errors
- ❑ The error that occurs in a program at the time of compilation is known as compilation error (compile-time error).
- ❑ These errors occur due to the syntactical mistakes under the program. That means these errors occur by typing the wrong syntax like missing double quotes and terminators, typing wrong spelling for keywords, assigning wrong data to a variable, trying to create an object for abstract class and interface, etc.
- ❑ Usually, this type of error occurs due to a poor understanding of the programming language. These errors can be identified by the programmer and can be rectified before the execution of the program only. So these errors do not cause any harm to the program execution.

- ❑ The errors which are occurred at the time of program execution are called the runtime error.
- ❑ These errors occurred when we are entering wrong data into a variable, trying to open a file for which there is no permission, trying to connect to the database with the wrong user id and password, the wrong implementation of logic, missing required resources, etc.
- ❑ A runtime error is known as an exception in C#. The exception will cause the abnormal termination of the program execution.
- ❑ Runtime errors are dangerous because whenever they occur in the program, the program terminates abnormally on the same line where the error gets occurred without executing the next line of code.

- ❑ Objects of exception classes are responsible for abnormal termination of the program whenever runtime errors (exceptions) occur.
- ❑ These exception classes are predefined under BCL (Base Class Libraries) where a separate class is provided for each and every different type of exception.
- ❑ After abnormal termination of the program exception classes will be displaying an error message which specifies the reason for abnormal termination. So, whenever a runtime error (exception) occurs in a program, first the exception manager under the CLR (Common Language Runtime) identifies the type of error that occurs in the program, then creates an object of the exception class related to that error and throws that object which will immediately terminate the program abnormally on the line where error got occur and display the error message related to that class.

Table 5.3 Examples of predefined Exception classes

<i>S.No.</i>	<i>Exception Type</i>	<i>Remarks</i>
1	Exception	Base class for all exception objects.
2	SystemException	Base class for all run time errors.
3	ApplicationException	Error in application program.
4	ArgumentException	Invalid argument; base class of all argument exceptions.
5	ArgumentNullException	A null argument passed to a method is not valid.
6	ArithmeticException	Arithmetic overflow or underflow error.
7	IndexOutOfRangeException	An array index is out of range.
8	NullReferenceException	Runtime error because a null object is referenced.
9	InvalidOperationException	Invalid operation in a method.
10	OutOfMemoryException	Error due to lack of memory for execution.
11	InvalidCastException	Error due to cast to an invalid class.
12	StackOverflowException	Stack overflow error.
13	FormatException	Error in the format of an argument.
14	DivideByZeroException	Error by dividing a value by zero.
15	NotFiniteNumberException	Error due to invalid number.

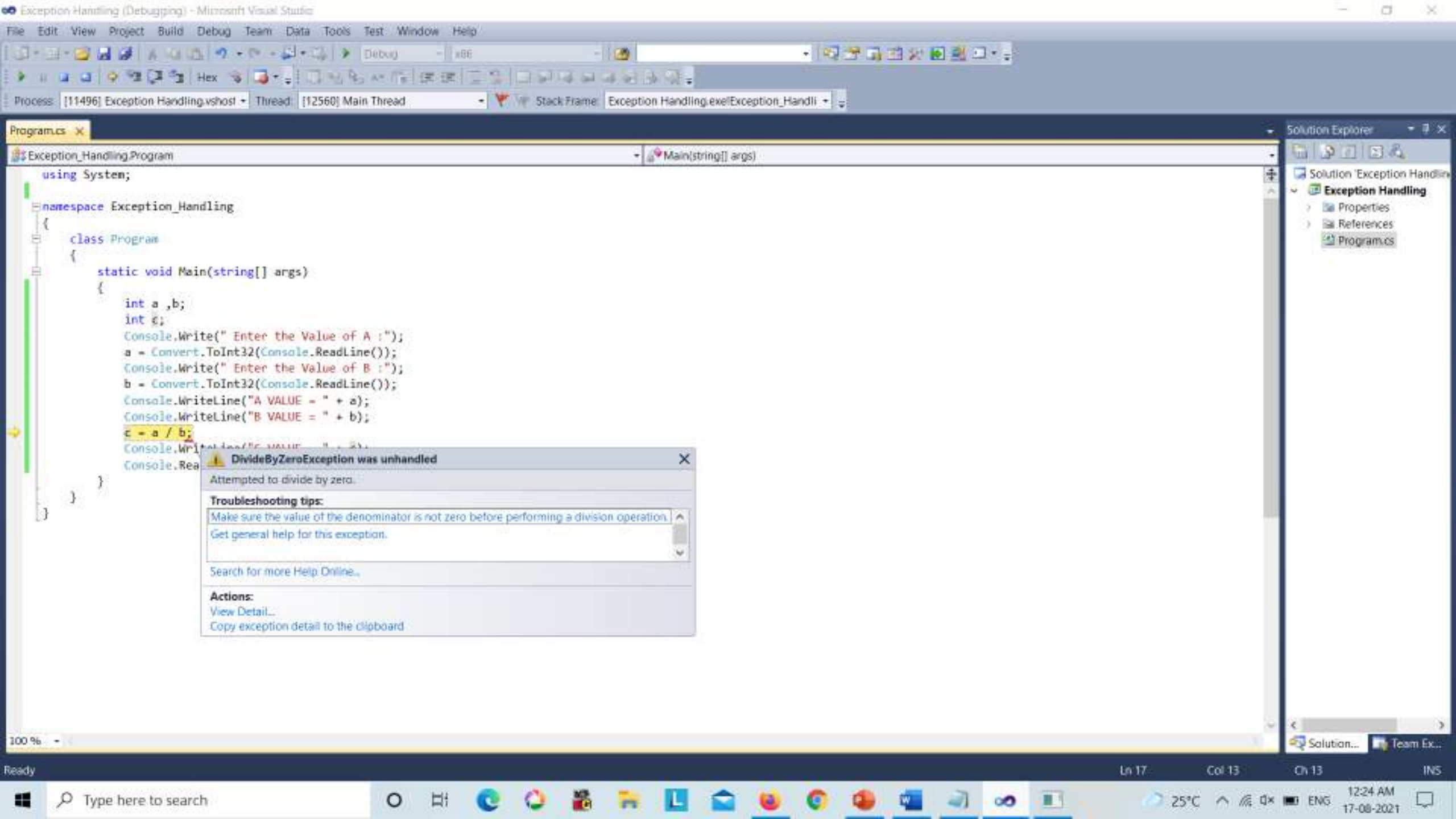
Exception Handling

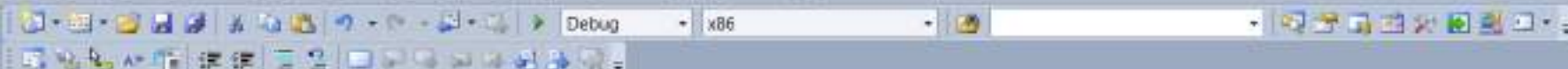
118

- Exception handling is a mechanism to detect and handle run time errors.
- It is achieved by using try-catch-finally blocks and throw keyword.
- Once we handle an exception under a program we will be getting the following advantages
 - ▣ We can stop the abnormal termination
 - ▣ We can perform any corrective action that may resolve the problem.
 - ▣ Displaying a user-friendly error message, so that the user can resolve the problem provided if it is under his control.
- **try:** The try keyword establishes a block in which we need to write the exception causing and its related statements. That means exception-causing statements must be placed in the try block so that we can handle and catch that exception for stopping abnormal termination and to display end-user understandable messages.
- **Catch:** The catch block is used to catch the exception that is thrown from its corresponding try block. It has the logic to take necessary actions on that caught exception. The Catch block syntax in C# looks like a constructor. It does not take accessibility modifier, normal modifier, return type. It takes the only single parameter of type Exception. Inside catch block, we can write any statement which is legal in .NET including raising an exception.

- ❑ **Finally:** The keyword finally establishes a block that definitely executes statements placed in it. Statements that are placed in finally block are always going to be executed irrespective of the way the control is coming out from the try block either by completing normally or throwing an exception by catching or not catching.
- ❑ A try block must be followed by catch or finally or both blocks. The try block without a catch or finally block will give a compile-time error.
- ❑ A catch block should include a parameter of a built-in or custom exception class to get an error detail. The following includes the Exception type parameter that catches all types of exceptions.
- ❑ You can use multiple catch blocks with the different exception type parameters. This is called exception filters. Exception filters are useful when you want to handle different types of exceptions in different ways.

- Once we use the try and catch blocks in our code the execution takes place as follows:
 - If all the statements under try block are executed successfully, from the last statement of the try block, the control directly jumps to the first statement that is present after the catch block (after all catch blocks) without executing catch block (it means there is no runtime error in the code at all).
 - Then if any of the statements in the try block causes an error, from that statement without executing any other statements in the try block, the control directly jumps to the catch blocks which can handle that exception.
 - If a proper catch block is found that handles the exception thrown by the try block, then the abnormal termination stops there, executes the code under the catch block, and from there again it jumps to the first statement after all the catch blocks.
 - If a matching catch is not found then abnormal termination occurs.





logical.cs Program.cs

Exception_Handling.logical Main(string[] args)

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Exception_Handling
{
    class logical
    {
        static void Main(string[] args)
        {
            int a, b;
            int c;
            Console.Write(" Enter the Value of A :");
            a = Convert.ToInt32(Console.ReadLine());
            Console.Write(" Enter the Value of B :");
            b = Convert.ToInt32(Console.ReadLine());
            Console.WriteLine("A VALUE = " + a);
            Console.WriteLine("B VALUE = " + b);

            if (b == 0)
            {
                Console.WriteLine("Second number should not be zero");
            }
            else
            {
                c = a / b;
                Console.WriteLine("C VALUE = " + c);
            }
            Console.ReadKey();
        }
    }
}
```

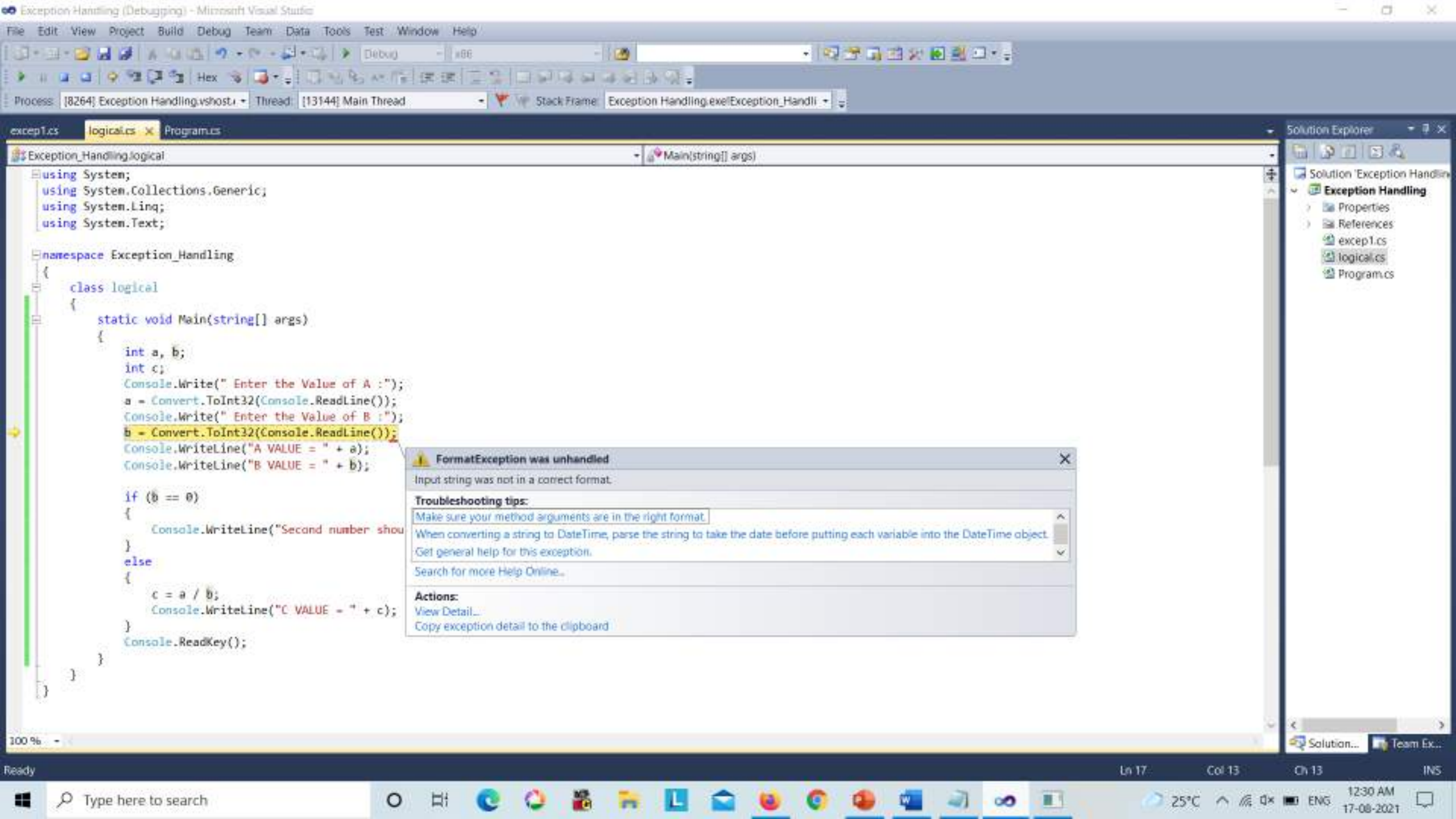
Solution Explorer

Solution 'Exception Handling' (1 p

- Exception Handling
 - Properties
 - References
 - logical.cs
 - Program.cs

Properties

```
Enter the Value of A :20
Enter the Value of B :0
A VALUE = 20
B VALUE = 0
Second number should not be zero
```

file:///c:/users/dr.pv.praveen sundar/documents/visual studio 2010/Projects/Exception Handling/Exception Handling/bin/Debug/Exc...

Enter the Value of A :20
Enter the Value of B :0
A VALUE = 20
B VALUE = 0
Second value is not equal to zero- Attempted to divide by zero.

```
file:///c:/users/dr.pv.praveen sundar/documents/visual studio 2010/Projects/Exception Handling/Exception Handling/bin/Debug/Exc...
Enter the Value of A :12345678900987654321
Value was either too large or too small for an Int32.
Program Executed Finally block
Enter the Value of A :10
Enter the Value of B :20
A VALUE = 10
B VALUE = 20
C VALUE = 0
Program Executed Finally block
```

Properties of Exception Class in C#

127

- The C# Exception Class has 3 properties are as follows:
 - **Message:** This property will store the reason why an exception has occurred.
 - **Source:** This property will store the name of the application from which the exception has been raised.
 - **Help link:** This is used to provide a link to any file /URL to give helpful information to the user when an exception is raised.

User-defined Exceptions

- An exception that is raised explicitly under a program based on our own condition (i.e. user-defined condition) is known as an application exception. As a programmer, we can raise application exception at any given point of time.
- In C#, it is possible to create our own exception class. But Exception must be the ultimate base class for all exceptions in C#. So the user-defined exception classes must inherit from either Exception class or one of its standard derived classes.

- The throw statement throws an exception. The syntax is given below

Throw expression

Where,

- ▣ Throw is a keyword
- ▣ Expression denotes a value of a class type System.Exception or a class that is derived from System.Exception.
- ▣ An expression arises through statement in the called method.

```
using System;
namespace Exception_Handling
{
    class except1
    {
        static void Main(string[] args)
        {
            int a, b, c;
            aa:
            try
            {
                Console.WriteLine(" Enter the Value of A and B:");
                a = Convert.ToInt32(Console.ReadLine());
                b = Convert.ToInt32(Console.ReadLine());
                Console.WriteLine(" A VALUE = " + a + "\n B VALUE = " + b);
                c = a / b;
                if (a + b > 100)
                    throw new Exception(" Out of Range Exception");
                Console.WriteLine("C VALUE = " + c);
            }
            catch (FormatException ex)
            {
                Console.WriteLine(" Inputted values are not an integer value - " + ex.Message);
                goto aa;
            }
            catch (DivideByZeroException ex)
            {
                Console.WriteLine(" Second value is not equal to zero- " + ex.Message);
                goto aa;
            }
            catch (Exception ex)
            {
                Console.WriteLine(ex.Message.ToString());
                goto aa;
            }
            finally
            {
                Console.WriteLine(" Program Executed Finally block");
            }
            Console.ReadKey();
        }
    }
}
```

Thank you

