

VB.Net(18K4CS06)

Unit - I

Introduction to Programming - Converting source code - Converting Source Code to Machine Language Code - Explaining Program Development Cycle. Introducing .NET Framework4.5 and Visual Studio 2012. Exploring Visual Studio 2012 IDE - Developing a Console Application.

Unit - II

Visual Basic: Getting started with Visual Basic 2012 - Visual Basic 2012 Keywords - Operators - Variables - Constants - Arrays. WindowsForms: Adding Controls to a Forms - Resizing and Moving Forms and Controls at Runtime - Creating Input Boxes - Creating Dialog Boxes.

Unit - III

Windows Forms Controls I: Introducing the Control Classes - Using the Label Control - Using the TextBox control - Using the Button Control - Using the Radio Control - Using the Checkbox Control - Using the Panel Control - Using the PictureBox Control - Using the ProgressBar Control.

Unit - IV

Windows Form Controls II: Using the toolStrip Control - Using the MenuStrip Control - Using the CheckBox Control - Using the StatusStrip Control - Working with DialogBoxes - Using FolderBrowser Dialog Control - Using the OpenFileDialog Control - Using the SaveFile Dialog Control - Using the FontDialogControl - Using the ColorDialog Control.

Unit - V

Windows Presentation Foundation: Exploring the improvements in WPF4.5: The Ribbon Control - Support for Binding to Types that implement CustomTypeProvider - New Virtualizing Panel Features - Extensions for Events. Explaining WPF 4.5 Architecture : Windows Base - The Milecore Component - Exploring WPF4.5 Designer - Using XMAML in WPF - Working with WPF Controls.

Text : “.NET Programming” - Vikas Gupta - DreamTechPress - Edition - 2014.

E-Content Prepared by

1.N.Subha M.Sc., M.Phil.,

2.Ramya M.Sc., M.Phil.,

UNIT I

Introduction to Programming

A computer is a calculating and computing device that is used to perform calculations and manipulations on various types of data, such as integers, strings, and float numbers. A computer performs all these tasks with the help of numbers of software, such as word processors, spreadsheets, and database applications. Software can be defined as a set of one or more programs, which are organized sets of certain statements and instructions that direct a computer to perform a specific task. These programs are converted into executable files with the help of a compiler or an interpreter.

A programming language is a [formal language](#) comprising a [set of instructions](#) that produce various kinds of [output](#). Programming languages are used in [computer programming](#) to implement [algorithms](#).

Each programming language has its own set of rules, known as syntax, which governs the structure of the statements in a program.

Programming languages can be broadly divided into the following two categories:

§ High level language

§ Low level language

High level language

A high-level language (HLL) is a programming language such as C, FORTRAN, or Pascal that enables a programmer to write programs that are more or less independent of a particular type of computer. Such languages are considered high-level because they are closer to human languages and further from machine languages. Programs written in a high-level language must be translated into machine language by a [compiler](#) or [interpreter](#).

Program to add two numbers in BASIC

```
10 LET A =2
```

```
20 LET B=3
```

```
30 LET SUM = A+B
```

```
40 PRINT SUM
```

```
50 END
```

Low level language

A low-level language is a type of [programming language](#) that contains basic instructions recognized by a computer. Unlike [high-level languages](#) used by software [developers](#), low-level code is often cryptic and not human-readable. Two common types of low-level programming languages are

§ [assembly language](#)

§ [machine language](#).

Assembly language

Earlier than high level languages were not invented, the programmers used to write programs in the Assembly language. Similar to high level language, a program written in assembly language is also not understandable by a computer. To make it understandable to a computer it needs to be translated into machine language code with help of an assembler. A program written in the Assembly language consists of a set of instructions called mnemonics.

Machine language

The Machine language is the only language a computer can understand directly. A program written in machine language is a sequence of binary digits - 1s and 0s. The machine language is very difficult to understand and learn even for an advanced programmer.

Converting source code to machine language code

The programmers write programs by using a source language, which can be either a high-level language or an assembly language; however a computer cannot understand both these types of programming languages. Therefore, programs written in these types of languages need to be first converted into the machine language code. To convert the source code of a program into machine language code, a language converter is required.

HIGH LEVEL LANGUAGE TO MACHINE LANGUAGE CONVERSION

Conversion of code from high-level language to machine language is done with the help of the following two language converters:

Compiler:

Converts a complete program from a high-level language to machine language. If an error is caught by the compiler in the program during conversion, then the error is displayed to the user. On the other hand, if the compiler does not find any error in the program, then the compiler executes the program.

Interpreter :

Converts the code written in a high-level language into machine language code, line by line. If the interpreter finds an error in the first line of code during conversion, it displays the error to the user. On the other hand, if the interpreter does not find any error in the first line of code, then the second line of the code is checked. The same process is followed for the remaining lines of the code as well.

Assembly language to Machine language conversion

The conversion of assembly language code into machine language code is done through another language converter, called assembler.

EXPLAINING PROGRAM DEVELOPMENT CYCLE

The program development cycle is a series of tasks and activities that take place during the development of a program.

The various stages of the program development cycle are as follows

- 1 .Analyze the problem
2. Developing a solution
3. coding the solution
4. testing the program

ANALYZING THE PROBLEM

Analyzing the problem involves determining what task the program needs to perform and what you need to do to enable that program to perform that task. Once we have clearly visualized the problem, we can move on to design the solution of the problem while analyzing a problem , we need to consider:

- The input supplied
- The process to obtain the and required output from the given input

- The output desired

In addition to determine the required output on the given input for your program, We also need to determine the variables and their types to represent input and output. A variable acts as a Placeholder parts during values used in a program.

Input

As explained one or more inputs are needed to be taken by a program to perform a task and return an output. For example, following inputs are required to display their registration information of the students of an educational Institute:

- Course code
- A character specifying whether or not more students need to be registered

Let's assume that The Institute offers three courses with course code A, B and C. With this assumption come on the following variables are used to store the data used in this program:

- `course_code` for storing the value of course code
- `New_Regfor` Storing a character specifying whether or not more students need to be registered.
- `No_stud_A` for storing the number of students registered for course A
- `No_Stud_B` for storing the number of students registered for course B.
- `No_Stud_C` for storing the number of students registered for course C.

Process

To obtain the desired output from the given input, we need to process the data. the steps for processing the data in this program are as follows:

1. First, Check the course code for the first student.
2. If it is A, the entry will be made to the respective course, that is, course A. In this way, the course code for all the students is checked and the entries in the respective courses are made.
3. Finally, the total number of entries for the three courses are made.

Output

The output of the program includes:

- Total number of students registered for course A
- Total number of students registered for course B
- Total number of students registered for course C

DEVELOPING A SOLUTION

Once the problem has been analyzed, that is, the required output and the given input are determined, we can start developing a solution for the problem. The most important task in developing a solution is the development of logic that solves the problem. This requires creation of a step-by-step procedure to solve the problem. This type of procedure is commonly termed as an algorithm. Once we create an algorithm to solve a problem, we can convert it into either a flowchart or pseudocode. A flowchart is a graphical representation of an algorithm. Pseudocode refers to short, readable, and formally-styled natural language code that is used to explain specific tasks within a program's algorithm.

Algorithm

An algorithm, as stated earlier, is a series of instructions written in any language spoken by a human being, such as English. An algorithm describes a way to perform a particular programming task.

1. Initialize `New_Reg` to 'Y'
2. Initialize `No_Stud_A` to 0
3. Initialize `No_Stud_B` to 0
4. Initialize `No_Stud_C` to 0
5. Input `Course_Code`
6. Check the `Course_Code`. If it is A, then add 1 to `No_Stud_A`; otherwise, move to step 7.
7. If `Course_Code` is B, then add 1 to `No_Stud_B`; otherwise, move to step 8.
8. If `Course_Code` is C, then add 1 to `No_Stud_C`;
9. Input `New_Reg`
10. If `New_Reg` is 'Y', then repeat steps 5 to 9; otherwise, move to step 11.

11. Print the total number of students registered in each course, that is, No_Stud_A, No_Stud_B and No_Stud_C.

Pseudocode

Pseudocode is structured set of English phrases that are used to describe the algorithms. It focuses on the logic of the algorithm and does not have any programming language specific keywords.

```
Initialize New_Reg to 'Y'
Initialize No_Stud_A to 0
Initialize No_Stud_B to 0
Initialize No_Stud_C to 0
While ( value of New_Reg is 'Y')
{
Course_code = get the course code from the user
    If Course_code= 'A' then
        No_Stud_A= No_Stud_A + 1
    If Course_code= 'B' then
        No_Stud_B= No_Stud_B + 1
    If Course_code= 'C' then
        No_Stud_C= No_Stud_C + 1
New_Reg = Get the character from the user
}
Print No_Stud_A
Print No_Stud_B
Print No_Stud_C
```

Flowchart

A flowchart is a graphical representation of the various steps involved in an algorithm. It makes the flow of the program easy to understand. A flowchart uses different symbols to indicate different operations. The various symbols used in a flowchart are shown in fig:

Symbol	Purpose	Description
	Flow line	Indicates the flow of logic by connecting symbols.
	Terminal(Stop/Start)	Represents the start and the end of a flowchart.
	Input/Output	Used for input and output operation.
	Processing	Used for arithmetic operations and data-manipulations.
	Decision	Used for decision making between two or more alternatives.
	On-page Connector	Used to join different flowline
	Off-page Connector	Used to connect the flowchart portion on a different page.
	Predefined Process/Function	Represents a group of statements performing one processing task.

CODING THE SOLUTION

The third stage is coding the solution or writing the program, based on the flowchart and pseudocode. Software used for programming may consist of one or more of the following tools.

Code Editor:

A code editor is optimized for programming. For example, in a code editor, executable statements might appear in one color and comments in another color to make the program easy to understand.

Compiler (or interpreter):

Converts the code in each source code file into machine language code and saves it in a file, known as object file.

Linker:

Combines all the object files into an executable program that can run directly on the target computer.

Debugger:

Helps us in finding errors in the programs.

Writing and executing a program is accomplished by performing the following steps:

1. Type the program
2. Compile the program
3. If there are any compile-time errors, correct them and again compile the program.
4. Run the program to obtain the desired results.
5. If there are any run-time errors, check the logic of the program and continue from step 2.

Compilers can only detect syntax errors. The errors that are detected at run-time are known as run-time errors. It is important to know the difference between compile-time and run-time errors.

- **compile-time error:** occurs if there is a mistake in the syntax of the program. A program will not compile successfully if there are syntax errors.

- Run-time error: Occurs if there is a mistake in the logic of the program. A program that was compiled successfully may have run-time errors. Run-time errors occur when we run the program.

TESTING THE PROGRAM

Testing is the last stage in the program development cycle. Once we have developed a program, we should test it to ensure that it is free of bugs and is capable of solving the given problem.

EXPLORING VISUAL STUDIO 2012 IDE

Visual studio 2012 IDE is a comprehensive environment for the development and execution of .NET applications. It consists of a menu bar, toolbar, and several windows that assist to design the UI of .NET applications as well as execute the applications.

Some of the important components of Visual Studio 2012 IDE are as follows:

- Start page
- Menu bar and Toolbar
- Toolbox
- Solution Explorer
- Properties window
- Designer and code editor
- Server Explorer
- Output window
- Object Browser
- Class View window

Start Page

As the name suggests, Start Page is the first page that appears whenever we open Visual Studio 2012.

Menu Bar and Tool Bar:

Various Menus in Visual Studio: A user can find a lot of menus on the top screen of Visual Studio as shown below

1. Create, Open and save projects commands are contained by File menu.
2. Searching, Modifying, Refactoring code commands are contained by the Edit menu.
3. View Menu is used to open the additional tool windows in Visual Studio.
4. Project menu is used to add some files and dependencies in the project.
5. To change the settings, add functionality to Visual Studio via extensions, and access various Visual Studio tools can be used by using Tools menu.

The below menu is known as the toolbar which provide the quick access to the most frequently used commands. You can add and remove the commands by going to View → Customize

Toolbox

Toolbox is a window that contains icons for various items and controls that we can add to a .NET application to design the UI of an application. The icons in Toolbox are logically grouped under different tabs, such as Stand, Containers, and Menus & Toolbars

Solution Explorer:

Visual Studio provides a Solution Explorer window that enables you to explore and manage your solutions and projects. To open the window select View > Solution Explorer.

Solution Explorer displays the projects that form your solution, the files and folders in a project as they appear on the physical hard drive, and any assemblies, COM objects or files the project references. The context menus within Solution Explorer provide a variety of commands that help you manage your projects.

Properties Window

Use this window to view and change the design-time properties and events of selected objects that are located in editors and designers. You can also use the Properties window to edit and view file, project, and solution properties. You can find Properties Window on the View menu. You can also open it by pressing F4 or by typing Properties in the search box.

The Properties window displays different types of editing fields, depending on the needs of a particular property. These edit fields include edit boxes, drop-down lists, and links to custom editor dialog boxes. Properties shown in gray are read-only.

Designer and Code Editor

The designer helps in designing the UI of application, while the code editor helps in adding the code or functionality for the application. The available designers and code editors in Visual Studio 2012 depend on the type of application and the file that we are working with.

Server Explorer

As said, the Server Explorer provides quick access to the connected database servers. With the Server Explorer, you can set up queries for use in your program. A default instance of the Server Explorer looks like the following:

- **Data Connections**
 - Servers
- **SharePoint Connections**

Developing a Console Application

1. Open Visual Studio 2012.
2. On the start window, choose Create a new project.
3. On the Create a new project window, enter or type *console* in the search box. Next, choose Visual Basic from the Language list, and then choose Windows from the Platform list.

After you apply the language and platform filters, choose the Console App (.NET Core) template, and then choose Next.

Then, in the Visual Studio Installer, choose the .NET Core cross-platform development workload.

After that, choose the Modify button in the Visual Studio Installer. You might be prompted to save your work; if so, do so. Next, choose Continue to install the workload. Then, return to step 2 in this "[Create a project](#)" procedure.

4. In the Configure your new project window, type or enter *WhatIsYourName* in the Project name box. Then, choose Create.

Visual Studio opens your new project.

Create the application

After you select your Visual Basic project template and name your project, Visual Studio creates a simple "Hello World" application for you. It calls the [WriteLine](#) method to display the literal string "Hello World!" in the console window.

If you click the HelloWorld button in the IDE, you can run the program in Debug mode.

When you do this, the console window is visible for only a moment before it closes. This happens because the `Main` method terminates after its single statement executes, and so the application ends.

Add some code

Let's add some code to pause the application and then ask for user input.

1. Add the following code immediately after the call to the [WriteLine](#) method:

```
Console.WriteLine("Press any key to continue...")  
Console.ReadKey(true)
```

This pauses the program until you press a key.

2. On the menu bar, select Build > Build Solution.

This compiles your program into an intermediate language (IL) that's converted into binary code by a just-in-time (JIT) compiler.

Run the application

1. Click the HelloWorld button on the toolbar.
2. Press any key to close the console window.

UNIT - II

New features of 2012

Async and Await:

- Async feature allows developers to write asynchronous code.
- Async method is used to execute a time consuming thread without forcing the caller's thread to wait.
- It helps them write the asynchronous code as easily as they write the synchronous code.
- It is a useful institution where the user interface is not responding or the server is not scalable.
- Await expression to suspend the execution of a thread until the awaited thread completes its task.

Iterators:

- An operator returns an element of a collection by using the yield statement.
- Whenever a yield statement is encountered the location in the code is retained.
- Next time, the execution is started from this location the iteration function is called by using the For Each...Next statement.

Call hierarchy

- The call hierarchy feature allows a developer to view all the calls made to and from a particular method or property or constructor.
- This feature provides a better understanding of how the code flows and the evaluation of the effects of modifications to code.

Caller information

- The caller information feature allows a developer to easily retrieve the information about the caller of a method by using the caller info attributes.
- This information is useful for tracing debugging and creating diagnostic tools.

Visual basic 2012 keywords

- Every programming language uses a set of predefined words in coding.
-
- Keyword has a specific predefined meaning for the compiler.
- Visual basic 2012 provides two types of keywords :
 - Reserved
 - Unreserved
- Reserved keywords for those keywords that cannot be used as names for programming elements such as variables methods and classes, while unreserved keywords for those keywords that can be used as names for programming elements.

Table 1.1: Reserved Keywords in Visual Basic 2012

AddHandler	AddressOf	Alias	And	AndAlso
Boolean	ByRef	Byte	ByVal	

Table 1.1: Reserved Keywords in Visual Basic 2012

Call	Case	Catch	CBool	CByte
CChar	CDate	CDbl	CDec	Char
CInt	Class	CLng	CObj	Const
Continue	CByte	CShort	CSng	CStr
CType	CUInt	CULng	CUShort	Date
Decimal	Declare	Default	Delegate	Dim
DirectCast	Do	Double	Each	Else
Elseif	End	EndIf	Enum	Erase
Error	Event	Exit	False	Finally
For	Friend	Function	Get	GetType
Global	GoSub	GoTo	Handles	If
Implements	Imports	In	Inherits	Integer
Interface	Is	IsNot	Let	Lib
Like	Long	Loop	Me	Mod
Module	MustInherit	MustOverride	MyBase	MyClass
Namespace	Narrowing	New	Next	Not
Nothing	NotInheritable	NotOverridable	Object	Of
On	Operator	Option	Optional	Or
OrElse	Out	Overloads	Overridable	Overrides
ParamArray	Partial	Private	Property	Protected
Public	RaiseEvent	ReadOnly	ReDim	REM
RemoveHandler	Resume	Return	SByte	Select
Set	Shadows	Shared	Short	Single
Static	Step	Stop	String	Structure
Sub	SyncLock	Then	Throw	To
True	Try	TryCast	TypeOf	UInteger
ULong	UShort	Using	Variant	Wend
When	While	Widening	With	WithEvents
WriteOnly	Xor	#Const	#Else	#Elseif
#End	#If	=	&	&=
*	*=	/	/=	\
\=	^	^=	+	+=
	=	>>	>>=	<<

Aggregate	Ansi	Assembly	Async	Await	Auto	Binary
Compare	Custom	Distinct	Equals	Explicit	From	Group By
Group Join	Into	IsFalse	IsTrue	iterator	Join	key
Mid	Off	Order By	Preserve	Skip	Skip While	Strict
Take	Take While	Text	Unicode	Until	Where	Yield
ExternalSource	Region					

Operators

An operator is a symbol that is used to perform an operation on one or more expressions called operands.

- ❖ Arithmetic operators
- ❖ Assignment operators
- ❖ Comparison operators
- ❖ Concatenation operators
- ❖ Logical and bitwise operators
- ❖ Miscellaneous operators

Arithmetic operators

The operators used to perform arithmetic operations such as subtraction multiplication and division are called arithmetic operators.

Operators	Meaning	Example
^	Raises one operand to the power of another	$x \wedge y$
+	Adds two operands	$x + y$
-	Subtracts second operand from the first	$x - y$
*	Multiplies both operands	$x * y$
/	Divides one operand by another and returns a floating-point result	x / y
\	Divides one operand by another and returns an integer result	$x \setminus y$
MOD	Modulus Operator and the remainder of a result after an integer division	$x \text{ MOD } y$

Assignment Operators

Assignment operators are used for assigning values to variables.

Operators	Example	Equivalent to
=	x = 4	x = 4
+=	x += 4	x = x + 4
-=	x -= 4	x = x - 4
*=	x *= 4	x = x * 4
/=	x /= 4	x = x / 4
\=	x \= 4	x = x \ 4
^=	x ^= 4	x = x ^ 4
<<=	x <<= 4	x = x << 4
>>=	x >>= 4	x = x >> 4
&=	x &= 4	x = x & 4

Comparison Operators

- Comparison operators are basically used to compare different values.
- These operators normally return Boolean values either True or False depending upon the condition.

Operators	Meaning	Example
=	Equality Check -Returns True if both values are the same	x == y
<>	Non-Equality Returns True if both values are unequal	x <> y
>	Greater than Check>Returns true if the first value specified is greater than the second	x > y
<	Less than>Returns true if the first value specified is less than second	x < y
>=	Checks for two conditions, If the first value is greater than or equal to the second value it returns true	>= y
<=	Checks for two conditions, If the first value is less than or equal to the second value it returns true	x <= y
Is	Compares two Object Variable for Reference, True If the same object reference	
IsNot	Compares two Object Variable for Reference, False If the same object reference	
Like	compares a string against a pattern.	

Concatenation Operators

The process of combining two text strings into one string is called string concatenation and operators used to perform string concatenation are called concatenation operators.

Operators	Meaning
&	String Concatenation
+	String Concatenation

Logical and Bitwise Operators

Logical operators can be defined as operators that are used with expressions and produce a Boolean value.

Operators	Meaning	Example
And	Logical as well as bitwise AND operator. Returns True If both the operands are true	x And y
	Does not perform short-circuiting, i.e., it evaluates both the expressions	
Or	Logical as well as bitwise OR operator. Returns True If any of the two operands is true. It does not perform short-circuiting.	x Or y
Not	Logical as well as bitwise NOT operator. If True, then this operator will make it false.	Not y
Xor	Logical as well as bitwise Logical Exclusive OR operator. Returns True if both expressions are the same; otherwise False.	x Xor y
AndAlso	Logical AND operator. Works only on Boolean data. Performs short-circuiting.	x AndAlso y
OrElse	Logical OR operator. Works only on Boolean data. Performs short-circuiting.	x OrElse y
IsFalse	Determines whether an expression is False	
IsTrue	Determines whether an expression is False	

Miscellaneous Operators

Operators	Example	Equivalent to
AddressOf	Returns the address of a procedure.	AddHandler Button1.Click, AddressOf Button1_Click
Await	It is applied to an operand in an asynchronous method or lambda expression to suspend execution of the method until the awaited task completes.	Dim result As res = Await AsyncMethodThatReturnsResult() Await AsyncMethod()
GetType	It returns a Type object for the specified type.	MsgBox(GetType(Integer).ToString())
Function Expression	It declares the parameters and code that define a function lambda expression.	Dim add5 = Function(num As Integer) num + 5 'prints 10 Console.WriteLine(add5(5))
If	It uses short-circuit evaluation to conditionally return one of two values.	Dim num = 5 Console.WriteLine(If(num >= 0, "Positive", "Negative"))

Operator Precedence

- When several operations occur in an expression, each part is evaluated and resolved in a predetermined order called *operator precedence*.
- The arithmetic and concatenation operators have an order of precedence that is described below, and all have higher precedence than the comparison and logical operators.
- Comparison operators have higher precedence than the logical operators, but lower precedence than the arithmetic and concatenation operators.
- All comparison operators have equal precedence; that is, they are evaluated in the order, left to right, in which they appear.

Arithmetic, concatenation and logical/bitwise Operators are evaluated in the following order of precedence:

Arithmetic/Concatenation Operators

Exponentiation (^)

Negation (−)

Multiplication and division (*, /)

Integer division (\)

Modulus arithmetic (**Mod**)

Addition and subtraction (+, −), String concatenation (+)

String concatenation (**&**)

Comparison Operators

Equality (=)

Inequality (<>)

Less than, greater than (<,>)

Greater than or equal to (>=)

Less than or equal to (<=)

Like

Is

TypeOf...Is

Logical/Bitwise Operators

Negation (**Not**)

Conjunction (**And, AndAlso**)

Disjunction (**Or, OrElse, Xor**)

Data types in Visual Basic 2012

- Data types refer to an extensive system used for declaring variables or functions of different types.
- The type of a variable determines how much space it occupies in storage and how the bit pattern stored is interpreted.

Data Types Available in VB.Net

VB.Net provides a wide range of data types.

The following table shows all the data types available –

Data Type	Storage Allocation	Value Range
Boolean	Depends on implementing platform	True or False
Byte	1 byte	0 through 255 (unsigned)
Char	2 bytes	0 through 65535 (unsigned)
Date	8 bytes	0:00:00 (midnight) on January 1, 0001 through 11:59:59 PM on December 31, 9999
Decimal	16 bytes	0 through +/-79,228,162,514,264,337,593,543,950,335 (+/-7.9...E+28) with no decimal point; 0 through +/-7.9228162514264337593543950335 with 28 places to the right of the decimal
Double	8 bytes	-1.79769313486231570E+308 through -4.94065645841246544E-324, for negative values 4.94065645841246544E-324 through 1.79769313486231570E+308, for positive values
Integer	4 bytes	-2,147,483,648 through 2,147,483,647 (signed)
Long	8 bytes	-9,223,372,036,854,775,808 through 9,223,372,036,854,775,807(signed)

Object	4 bytes on 32-bit platform 8 bytes on 64-bit platform	Any type can be stored in a variable of type Object
SByte	1 byte	-128 through 127 (signed)
Short	2 bytes	-32,768 through 32,767 (signed)
Single	4 bytes	-3.4028235E+38 through -1.401298E-45 for negative values; 1.401298E-45 through 3.4028235E+38 for positive values
String	Depends on implementing platform	0 to approximately 2 billion Unicode characters
UInteger	4 bytes	0 through 4,294,967,295 (unsigned)
ULong	8 bytes	0 through 18,446,744,073,709,551,615 (unsigned)
User-Defined	Depends on implementing platform	Each member of the structure has a range determined by its data type and independent of the ranges of the other members
UShort	2 bytes	0 through 65,535 (unsigned)

The Type Conversion Functions in VB.Net

S.No.	Functions & Description
1	CBool(expression) Converts the expression to Boolean data type.
2	CByte(expression) Converts the expression to Byte data type.
3	CChar(expression) Converts the expression to Char data type.
4	CDate(expression) Converts the expression to Date data type

5	CDBl(expression) Converts the expression to Double data type.
6	CDec(expression) Converts the expression to Decimal data type.
7	CInt(expression) Converts the expression to Integer data type.
8	CLng(expression) Converts the expression to Long data type.
9	CObj(expression) Converts the expression to Object type.
10	CSByte(expression) Converts the expression to SByte data type.
11	CShort(expression) Converts the expression to Short data type.
12	CSng(expression) Converts the expression to Single data type.
13	CStr(expression) Converts the expression to String data type.
14	CUInt(expression) Converts the expression to UInt data type.
15	CULng(expression) Converts the expression to ULng data type.
16	CUShort(expression) Converts the expression to UShort data type.

Visual Basic Statements

A statement is a complete instruction in Visual Basic programs.

It may contain keywords, operators, variables, literal values, constants and expressions.

Statements could be categorized as –

- Declaration statements – these are the statements where you name a variable, constant, or procedure, and can also specify a data type.
- Executable statements – these are the statements, which initiate actions. These statements can call a method or function, loop or branch through blocks of code or assign values or expression to a variable or constant. In the last case, it is called an Assignment statement.

Using the If...Else Statement:

Syntax

```
If boolean_expression Then  
  
// Statements to Execute if boolean expression is True  
  
Else  
  
// Statements to Execute if boolean expression is False  
  
End If
```

The statements inside of **If** condition will be executed only when the “**bool_expression**” returns **true** otherwise the statements inside of **Else** condition will be executed.

Example:

```
Dim x As Integer = 20  
  
If x >= 10 Then  
  
    Console.WriteLine("x is Greater than or equals 10")  
  
Else  
  
    Console.WriteLine("x is less than or equals to 10")End If
```

Using the Select...Case Statement:

Select...Case statement is useful to execute a single case statement from the group of multiple case statements based on the value of a defined expression.

By using **Select...Case** statement in Visual Basic, we can replace the functionality of if...else if statement to provide better readability for the code.

Generally, the Select...Case statement is a collection of multiple case statements and it will execute only one single case statement based on the matching value of the defined expression.

Syntax

```
Select Case variable/expression  
  
Case value1  
  
// Statements to Execute  
  
Case value2  
  
//Statements to Execute  
  
....  
  
....  
  
Case Else  
  
// Statements to Execute if No Case Matches  
  
End Select
```

Here, the **Select** statement will evaluate the **expression / variable** value by matching with **Case** statement values (value1, value2, etc.). If the variable/**expression** value matches with any of the **case** statements, then the statements inside of the particular **case** will be executed.

In case, if none of the **case** statements are matched with the defined **expression / variable** value, then the statements inside of the Else block will be executed and it's more like the Else block in if...else statement.

Example

```
Module Module1  
  
Sub Main()  
  
Dim x As Integer = 20  
  
Select Case x  
  
Case 10  
  
Console.WriteLine("x value is 10")
```

Case 15

```
Console.WriteLine("x value is 15")
```

Case 20

```
Console.WriteLine("x value is 20")
```

Case Else

```
Console.WriteLine("Not Known")
```

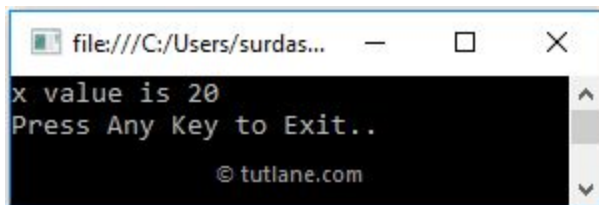
End Select

```
Console.WriteLine("Press Enter Key to Exit..")
```

```
Console.ReadLine()
```

End Sub

End Module



```
file:///C:/Users/surdas...  
x value is 20  
Press Any Key to Exit..  
© tutlane.com
```

Using the For...Next Statement:

For loop is useful to execute a statement or a group of statements repeatedly until the defined condition returns true.

Generally, For loop is useful to iterate and execute a certain block of statements repeatedly until the specified number of times.

Syntax

```
For variable As [Data Type] = start To end
```

```
// Statements to Execute
```

```
Next
```

Here, the variable parameter is required in the For statement and it must be numeric. The **Data Type** is optional and it is useful to define the data type for the variable. The **start** and **end** parameters are required to define the initial and final value of a variable.

Example

```
Module Module1
```

```
    Sub Main()
```

```
        For i As Integer = 1 To 4
```

```
            Console.WriteLine("i value: {0}", i)
```

```
        Next
```

```
        Console.WriteLine("Press Enter Key to Exit..")
```

```
        Console.ReadLine()
```

```
    End Sub
```

```
End Module
```



```
file:///C:/Users/surdasa...  
i value: 1  
i value: 2  
i value: 3 © tutlane.com  
i value: 4  
Press Any Key to Exit..
```

Using For Each...Next Statement:

Repeats a group of statements for each element in a collection.

Syntax

```
For Each element [ As datatype ] In group
```

```
    [ statements ]
```

```
    [ Continue For ]
```

```
    [ statements ]
```

```
    [ Exit For ]
```

```
    [ statements ]
```

```
Next [ element ]
```

Example

```
' Create a list of strings by using a  
' collection initializer.
```

```
Dim lst As New List(Of String) _  
    From {"abc", "def", "ghi"}
```

```
' Iterate through the list.
```

```
For Each item As String In lst  
    Debug.Write(item & " ")
```

```
Next
```

```
Debug.WriteLine("")
```

'Output: abc def ghi

Using the While... End While statement

While loop is useful to execute the block of statements as long as the specified condition is true.

Syntax

```
While boolean_expression
```

```
// Statements to Execute
```

```
End While
```

Example

```
Module Module1
```

```
    Sub Main()
```

```
        Dim i As Integer = 1
```

```
        While i <= 4
```

```
            Console.WriteLine("i value: {0}", i)
```

```
            i += 1
```

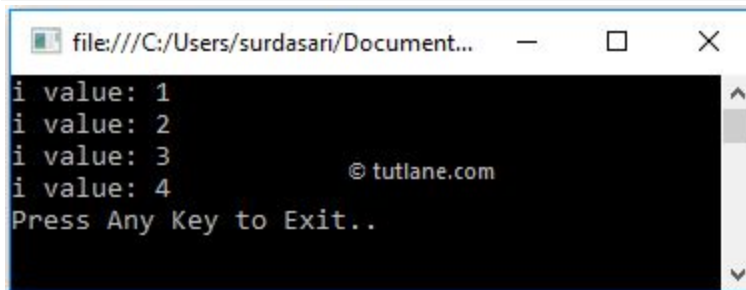
```
        End While
```

```
Console.WriteLine("Press Enter Key to Exit..")
```

```
Console.ReadLine()
```

```
End Sub
```

```
End Module
```



```
file:///C:/Users/surdasari/Document... - □ ×  
i value: 1  
i value: 2  
i value: 3 © tutlane.com  
i value: 4  
Press Any Key to Exit..
```

Using Do...Loop Statement

Generally, in Visual Basic the do-while loop is the same as while loop but only the difference is while loop will execute the statements only when the defined condition returns true but the do-while loop will execute the statements at least once because first it will execute the block of statements and then it will check the condition.

Syntax

Do

```
// Statements to Execute
```

```
Loop While boolean_expression
```

Example

```
Module Module1
```

```
Sub Main()
```

```
Dim i As Integer = 1
```

```
Do
```

```
Console.WriteLine("i value: {0}", i)
```

```
i += 1
```

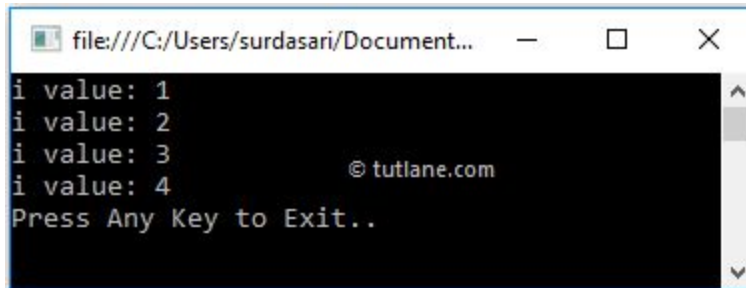
```
Loop While i <= 4
```

```
Console.WriteLine("Press Enter Key to Exit..")
```

```
Console.ReadLine()
```

```
End Sub
```

```
End Module
```



```
file:///C:/Users/surdasari/Document...  
i value: 1  
i value: 2  
i value: 3  
i value: 4  
Press Any Key to Exit..  
© tutlane.com
```

Variables

A variable is an identifier that denotes a storage area in the memory.

Syntax

```
Dim [Variable Name] As [Data Type]
```

```
Dim [Variable Name] As [Data Type] = [Value]
```

The variable name is to tell the compiler about the type of data the variable can hold.

Item	Description
Dim	It is useful to declare and allocate the storage space for one or more variables.
[Variable Name]	It's the name of the variable to hold the values in our application.
As	The As clause in the declaration statement allows you to define the data type.
[Data Type]	It's a type of data the variable can hold such as integer, string, decimal, etc.
[Value]	Assigning a required value to the variable.

Constants

- Generally, in visual basic the constant field values are set at compile-time and those values will never be changed.
- In visual basic, `Const` keyword is to declare the constant field, then that field value cannot be changed throughout the application.
- It's mandatory to initialize constant fields with required values during the declaration itself otherwise compile-time errors in visual basic application.

Syntax

```
Const field_name As data_type = "value"
```

Arrays

- In visual basic, **Arrays** are useful to store multiple elements of the same data type at contiguous memory locations and arrays will allow us to store the fixed number of elements sequentially based on the predefined number of items.
- In visual basic, **Arrays** can be declared by specifying the type of elements followed by the brackets `()` like as shown below.

```
Dim array_name As [Data_Type]();
```

Enumerations

- In visual basic, **Enum** is a keyword and it is useful to declare an enumeration.
- In visual basic, the enumeration is a type and that will contain a set of named constants as a list.
- By using enumeration, we can group constants that are logically related to each other.
- For example, the days of week can be grouped together by using enumeration in visual basic.

Syntax

```
Enum enum_name
```

```
' enumeration list
```

```
End Enum
```


Example

```
Enum Week
```

```
    Sunday
```

```
    Monday
```

```
    Tuesday
```

```
    Wednesday
```

```
    Thursday
```

```
    Friday
```

```
    Saturday
```

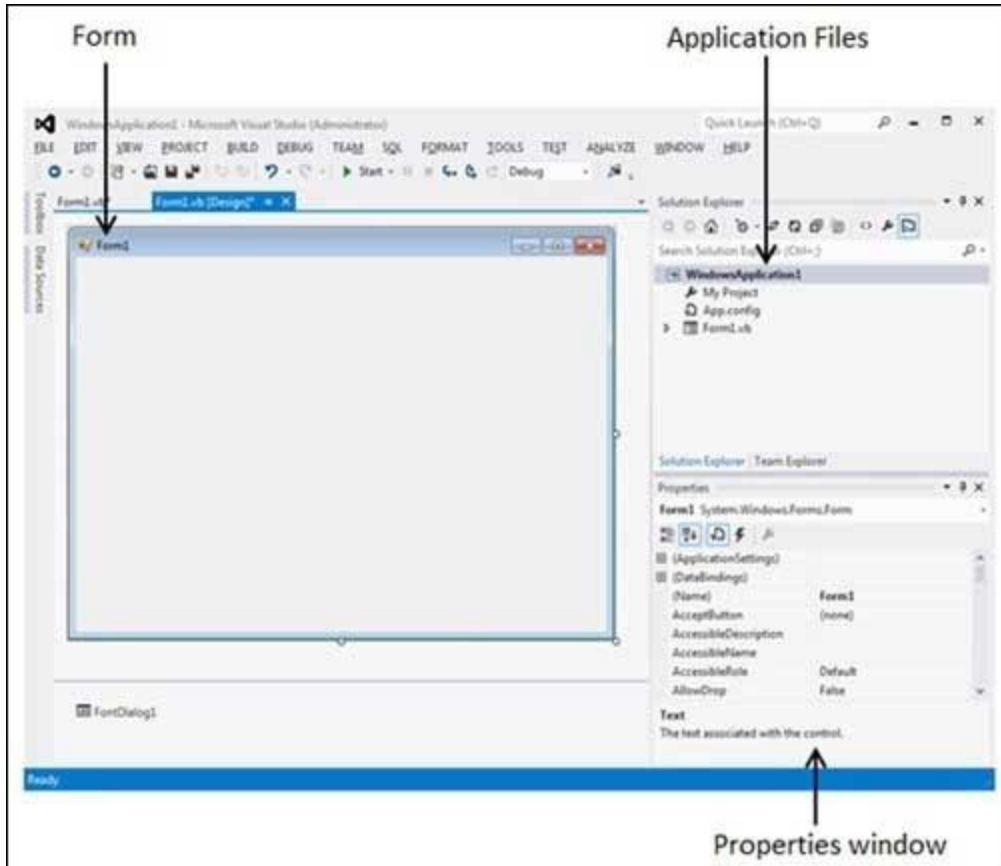
```
End Enum
```

In visual basic, by default the first named constant in the enumerator has a value of **0**, and the value of each successive item in the enumerator will be increased by **1**. For example, in the above enumeration, Sunday value is **0**, Monday is **1**, Tuesday is **2**, and so forth.

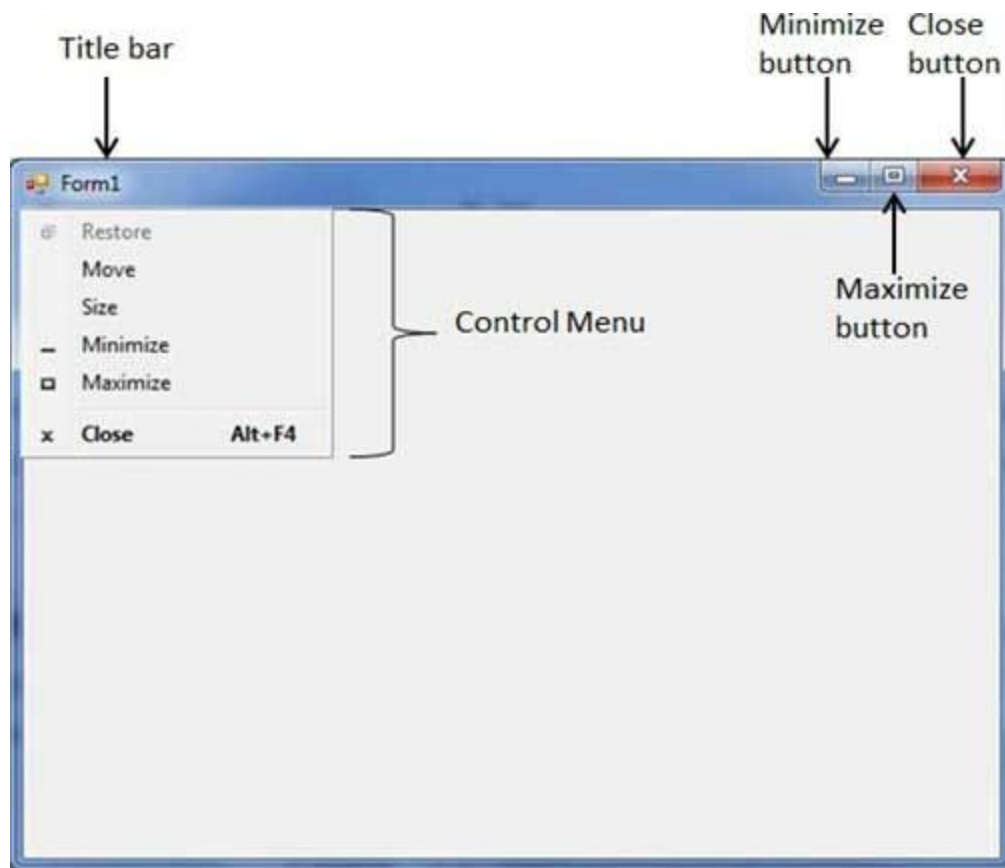
Properties of Form Class

Let's start with creating a Window Forms Application by following the following steps in **Microsoft Visual Studio - File → New Project → Windows Forms Applications**

Finally, select OK, Microsoft Visual Studio creates your project and displays the following window Form with the name Form1.



- Visual Basic Form is the container for all the controls that make up the user interface.
- Every window you see in a running visual basic application is a form, thus the terms form and window describe the same entity.
- Visual Studio creates a default form for you when you create a Windows Forms Application.
- Every form will have title bar on which the form's caption is displayed and there will be buttons to close, maximize and minimize the form shown below –



If you click the icon on the top left corner, it opens the control menu, which contains the various commands to control the form like to move control from one place to another place, to maximize or minimize the form or to close the form.

Form Properties

Following table lists down various important properties related to a form. These properties can be set or read during application execution.

S.No.	Properties	Description
1	AcceptButton	The button that's automatically activated when you press Enter, no matter which control has the focus at the time. Usually the OK button on a form is set as AcceptButton for a form.
2	CancelButton	The button that's automatically activated when you hit the Esc key. Usually, the Cancel button on a form is set as CancelButton for a form.

3	AutoScale	This Boolean property determines whether the controls you place on the form are automatically scaled to the height of the current font. The default value of this property is True. This is a property of the form, but it affects the controls on the form.
4	AutoScroll	This Boolean property indicates whether scroll bars will be automatically attached to the form if it is resized to a point that not all its controls are visible.
5	AutoScrollMinSize	This property lets you specify the minimum size of the form, before the scroll bars are attached.
6	AutoScrollPosition	The AutoScrollPosition is the number of pixels by which the two scroll bars were displaced from their initial locations.
7	BackColor	Sets the form background color.
8	BorderStyle	The BorderStyle property determines the style of the form's border and the appearance of the form – <ul style="list-style-type: none"> ● None – Borderless window that can't be resized. ● Sizable – This is default value and will be used for a resizable window that's used for displaying regular forms. ● Fixed3D – Window with a visible border, "raised" relative to the main area. In this case, windows can't be resized. ● FixedDialog – A fixed window, used to create dialog boxes. ● FixedSingle – A fixed window with a single line border. ● FixedToolWindow – A fixed window with a Close button only. It looks like the toolbar displayed by the drawing and imaging applications. ● SizableToolWindow – Same as the FixedToolWindow but resizable. In addition, its caption font is smaller than the usual.
9	ControlBox	By default, this property is True and you can set it to False to hide the icon and disable the Control menu.
10	Enabled	If True, allows the form to respond to mouse and keyboard events; if False, disables form.
11	Font	This property specify font type, style, size

12	HelpButton	Determines whether a Help button should be displayed in the caption box of the form.
13	Height	This is the height of the Form in pixels.
14	MinimizeBox	By default, this property is True and you can set it to False to hide the Minimize button on the title bar.
15	MaximizeBox	By default, this property is True and you can set it to False to hide the Maximize button on the title bar.
16	MinimumSize	This specifies the minimum height and width of the window you can minimize.
17	MaximumSize	This specifies the maximum height and width of the window you maximize.
18	Name	This is the actual name of the form.
19	StartPosition	<p>This property determines the initial position of the form when it's first displayed. It will have any of the following values –</p> <ul style="list-style-type: none"> ● CenterParent – The form is centered in the area of its parent form. ● CenterScreen – The form is centered on the monitor. ● Manual – The location and size of the form will determine its starting position. ● WindowsDefaultBounds – The form is positioned at the default location and size determined by Windows. ● WindowsDefaultLocation – The form is positioned at the Windows default location and has the dimensions you've set at design time.
20	Text	The text, which will appear at the title bar of the form.
21	Top, Left	These two properties set or return the coordinates of the form's top-left corner in pixels.
22	TopMost	This property is a True/False value that lets you specify whether the form will remain on top of all other forms in your application. Its default property is False.
23	Width	This is the width of the form in pixels.

Form Methods

The following are some of the commonly used methods of the Form class.

S.No.	Method Name & Description
1	Activate Activates the form and gives it focus.
2	ActivateMdiChild Activates the MDI child of a form.
3	AddOwnedForm Adds an owned form to this form.
4	BringToFront Brings the control to the front of the z-order.
5	CenterToParent Centers the position of the form within the bounds of the parent form.
6	CenterToScreen Centers the form on the current screen.
7	Close Closes the form.
8	Contains Retrieves a value indicating whether the specified control is a child of the control.
9	Focus Sets input focus to the control.
10	Hide Conceals the control from the user.
11	Refresh Forces the control to invalidate its client area and immediately redraw itself and any child controls.
12	Scale(SizeF) Scales the control and all child controls by the specified scaling factor.
13	ScaleControl Scales the location, size, padding, and margin of a control.

14	ScaleCore Performs scaling of the form.
15	Select Activates the control.
16	SendToBack Sends the control to the back of the z-order.
17	SetAutoScrollMargin Sets the size of the auto-scroll margins.
18	SetDesktopBounds Sets the bounds of the form in desktop coordinates.
19	SetDesktopLocation Sets the location of the form in desktop coordinates.
20	SetDisplayRectLocation Positions the display window to the specified value.
21	Show Displays the control to the user.
22	ShowDialog Shows the form as a modal dialog box.

Form Events

Following table lists down various important events related to a form.

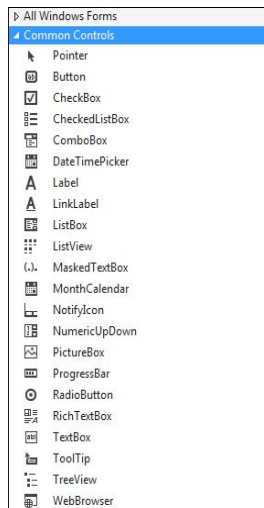
S.No.	Event	Description
1	Activated	Occurs when the form is activated in code or by the user.
2	Click	Occurs when the form is clicked.
3	Closed	Occurs before the form is closed.
4	Closing	Occurs when the form is closing.
5	DoubleClick	Occurs when the form control is double-clicked.
6	DragDrop	Occurs when a drag-and-drop operation is completed.

7	Enter	Occurs when the form is entered.
8	GotFocus	Occurs when the form control receives focus.
9	HelpButtonClicked	Occurs when the Help button is clicked.
10	KeyDown	Occurs when a key is pressed while the form has focus.
11	KeyPress	Occurs when a key is pressed while the form has focus.
12	KeyUp	Occurs when a key is released while the form has focus.
13	Load	Occurs before a form is displayed for the first time.
14	LostFocus	Occurs when the form loses focus.
15	MouseDown	Occurs when the mouse pointer is over the form and a mouse button is pressed.
16	MouseEnter	Occurs when the mouse pointer enters the form.
17	MouseHover	Occurs when the mouse pointer rests on the form.
18	MouseLeave	Occurs when the mouse pointer leaves the form.
19	MouseMove	Occurs when the mouse pointer is moved over the form.
20	MouseUp	Occurs when the mouse pointer is over the form and a mouse button is released.
21	MouseWheel	Occurs when the mouse wheel moves while the control has focus.
22	Move	Occurs when the form is moved.
23	Resize	Occurs when the control is resized.
24	Scroll	Occurs when the user or code scrolls through the client area.
25	Shown	Occurs whenever the form is first displayed.
26	VisibleChanged	Occurs when the Visible property value changes.

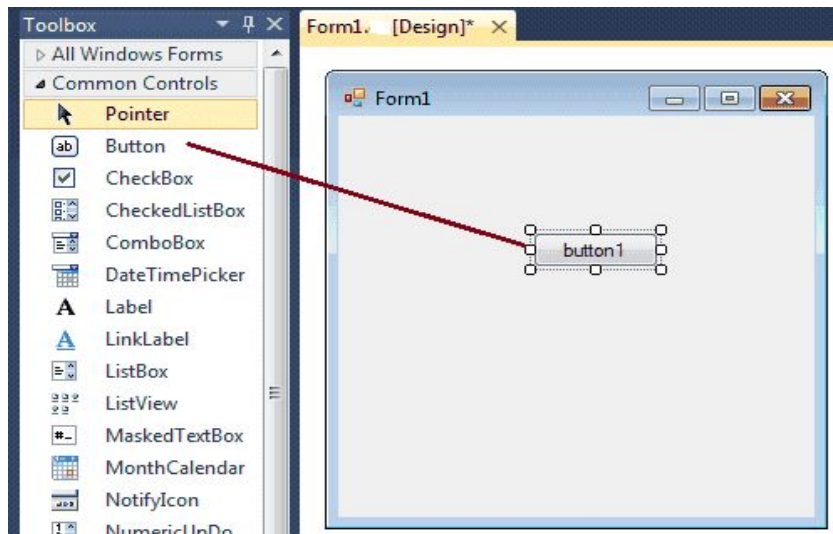
FormStartPosition Enumeration

Member Name	Description
CenterParent	The form is centered within the bounds of its parent form.
CenterScreen	The form is centered on the current display, and has the dimensions specified in the form's size.
Manual	The position of the form is determined by the <u>Location</u> property.
WindowsDefaultBounds	The form is positioned at the Windows default location and has the bounds determined by Windows default.
WindowsDefaultLocation	The form is positioned at the Windows default location and has the dimensions specified in the form's size.

Adding Controls to a Form

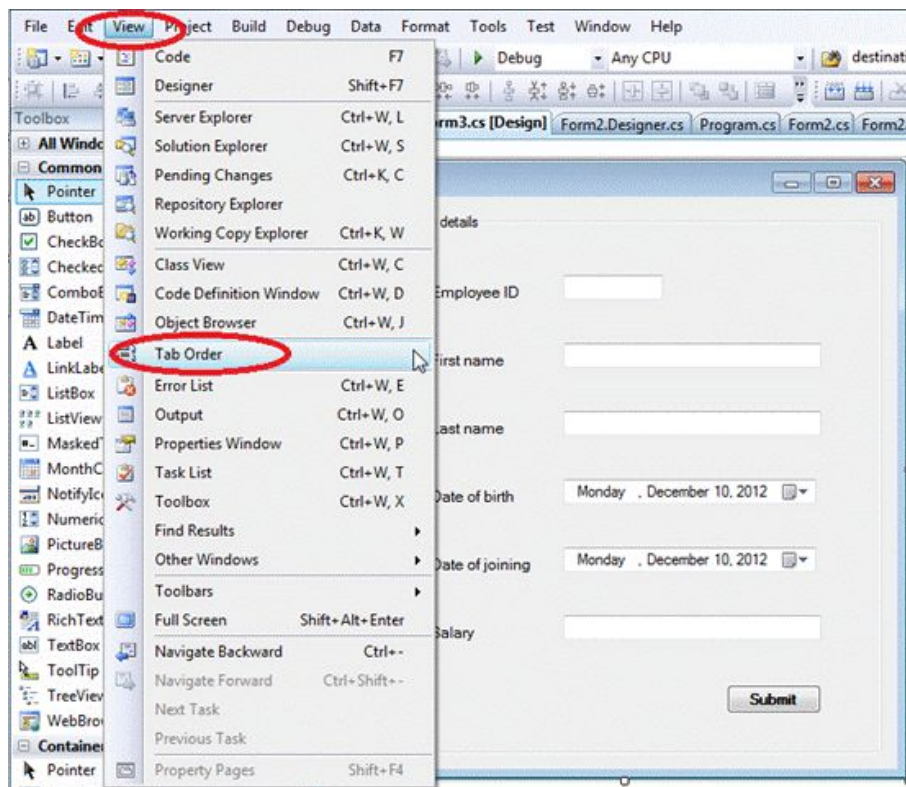


Each time you create a Windows application, Visual Studio will display a default blank form, onto which you can drag and drop controls from the Visual Studio Toolbox window.



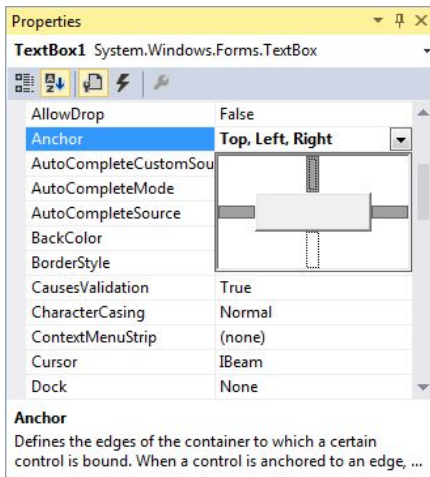
Setting TabOrder of Controls

Add your controls on the and form and go to design view of the form and select **View->Tab order**

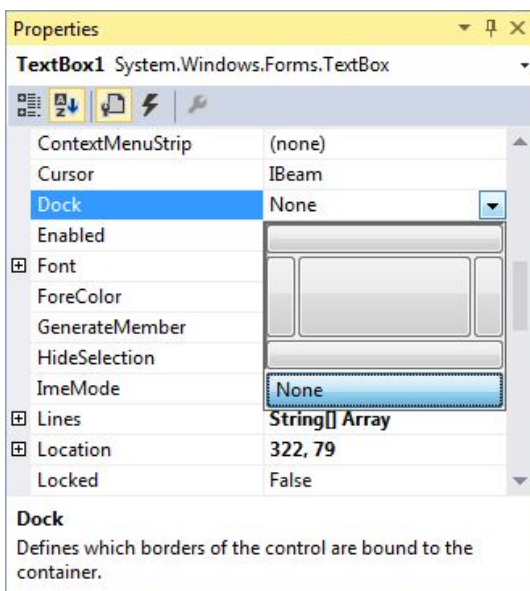


Docking and Anchoring Controls

- The Anchor and Dock properties of a form are two separate properties. Anchor refers to the position a control has relative to the edges of the form.
- A text box, for example, that is anchored to the left edge of a form will stay in the same position as the form is resized. Docking refers to how much space you want the control to take up on the form.
- If you dock a control to the left of the form, it will stretch itself to the height of the form, but its width will stay the same.



Docking is similar to Anchoring, but this time the control fills a certain area of the form.



Adding Forms to Windows Form Application

The first step is to start a new project and build a form.

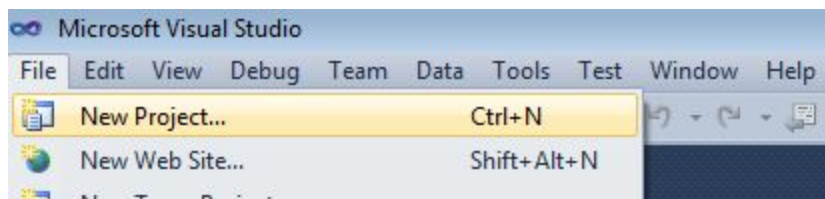
Open your Visual Studio and select **File->NewProject** and

select **Visual Basic** from the **New project dialog box** and

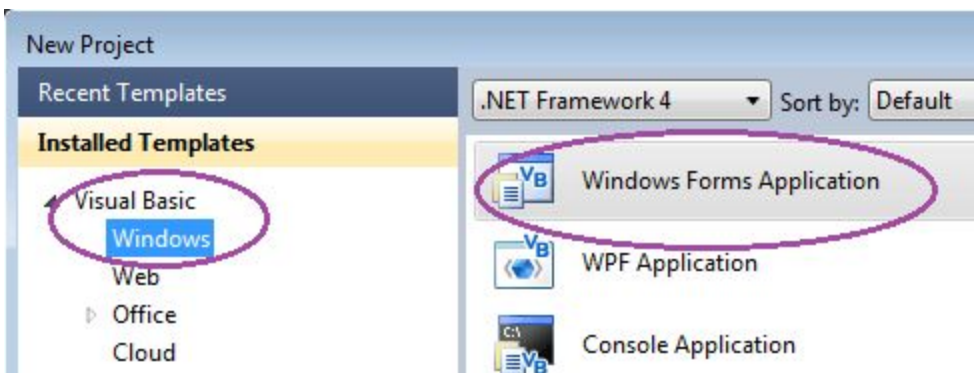
select **Windows Forms Application**

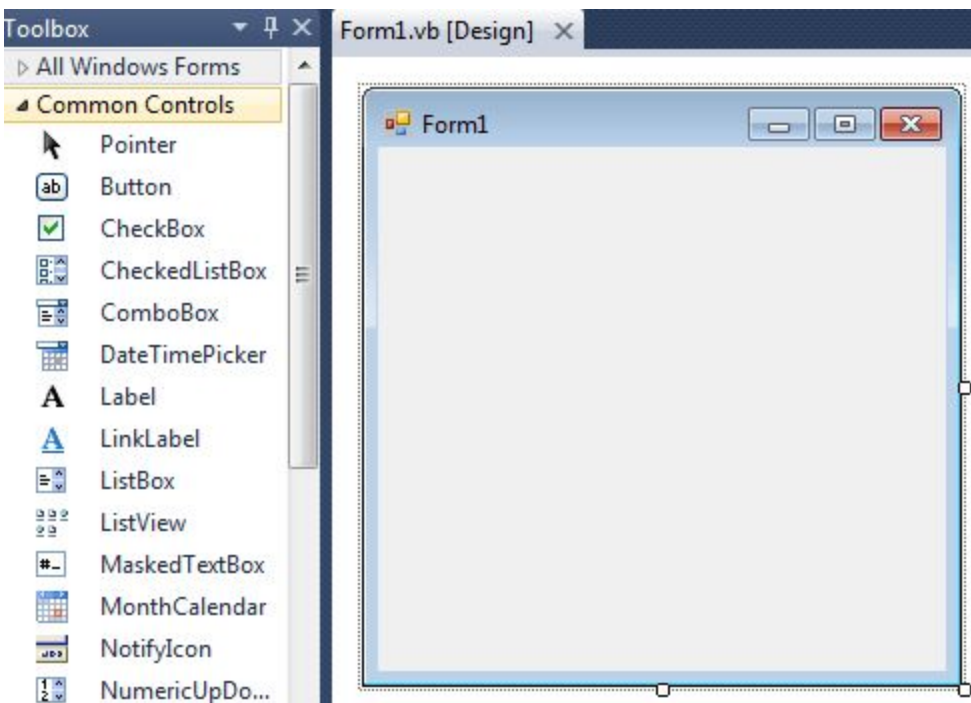
Enter your project name instead of WindowsApplication1 in the bottom of the dialog box and click the OK button.

The following picture shows how to create a new Form in Visual Studio.



Select project type from New project dialog Box.









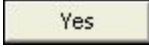
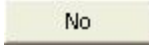

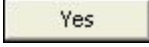
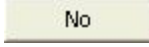




To work on multiple forms, create the new project and add a new form.

Creating Message Boxes

- A message box is a special dialog box used to display a piece of information to the user.
- As opposed to a regular form, the user cannot type anything in the dialog box.
- To support message boxes, the Visual Basic language provides a function named **MsgBox**.
- To support message boxes, the .NET Framework provides a class named.
- To display a simple message box, you can use the MsgBox() function with the following formula:

```
Private Sub btnMessage_Click(ByVal sender As System.Object, _  
    ByVal e As System.EventArgs) _  
    Handles btnMessage.Click  
    MsgBox("Welcome to Microsoft Visual Basic")  
End Sub
```

To Display	MsgBoxStyle	Integral Value
	OKOnly	0
 	OKCancel	1
  	AbortRetryIgnore	2
  	YesNoCancel	3
 	YesNo	4
 	RetryCancel	5

MessageBoxButtons Enumeration

AbortRetryIgnore	The message box contains Abort ,Retry,and Ignore buttons
OK	The message box contains an OK button
OkCancel	The message box contains OK and Cancel buttons
RetryCancel	The message box contains Retry and Cancel buttons
YesNo	The message box contains Yes and No buttonx
YesNoCancel	The message box with Yes, No and Cancel buttons

The Members of MessageBoxIcon Enumeraion

Member	Description
Asterisk	The message box contains a symbol consisting of a lowercase letter i in a circle.
Error	The message box contains a symbol consisting of white X in a circle with a red background.
Exclamation	The message box contains a symbol consisting of an exclamation point in a triangle with a yellow background
Hand	The message box contains a symbol consisting of a white X in a circle with a red background.
Information	The message box contains a symbol consisting of a lowercase letter i in a circle.
None	The message box contains no symbols
Question	The message box contains a symbol consisting of a question mark in a circle. The question mark message icon is no longer recommended because it does not clearly represent a specific type of message and because the phrasing of a message as a question could apply to any message type. In addition, users can confuse the question mark symbol with a help information symbol. Therefore, do not use this question mark symbol in your message boxes. The system continues to support its inclusion only for backward compatibility.
Stop	The message box contains a symbol consisting of white X in a circle with a red background.
Warning	The message box contains a symbol consisting of an exclamation point in a triangle with a yellow background

The Members of MessageDefaultButton Enumeration

Member	Description
Button1	The first button on the message box is the default button.
Button2	The second button on the message box is the default button.
Button3	The third button on the message box is the default button.

The Members of MessageBoxOptions Enumeration

Member	Description
RightAllign	The message box text is right-aligned.
RtlReading	Specifies that the message box text is displayed with right to left reading order.
ServiceNotification	The message box is displayed on the active desktop. The caller is a service notifying the user of an event. Show displays a message box on the current active desktop, even if there is no user logged on to the computer
DefaultDesktopOnly	The message box is displayed on the active desktop. This constant is similar to ServiceNotification, except that the system displays the message box only on the default desktop of the interactive window station. The application that displayed the message box loses focus, and the message box is displayed without using visual styles

The Members of DialogResult Enumeration

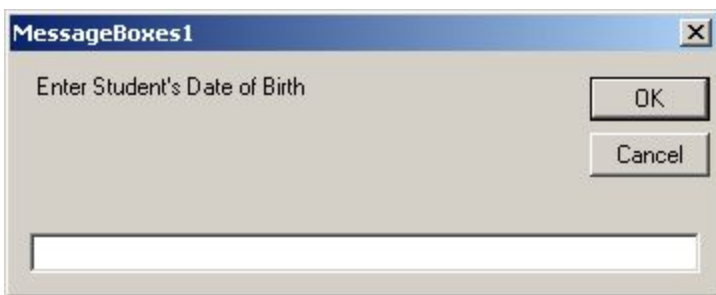
Member	Description
Abort	Returns Abort
Cancel	Returns Cancel
Ignore	Returns Ignore
No	Returns No
None	Returns nothing from the dialog box

The Members of DialogResult Enumeration

Member	Description
Ok	Return Ok
Retry	Return Retry
Yes	Return Yes

Creating Input Boxes

The **Input Box**. When an **input box** displays, it presents a request to the user who can then provide a value. After using the **input box**, the user can change his or her mind and press Esc or click Cancel. If the user provided a value and want to acknowledge it, he or she can click OK or press Enter.



Creating Dialog Boxes

- Dialog boxes are used to interact with the user and retrieve information.
- In simple terms, a dialog box is a form with its FormBorderStyle enumeration property set to FixedDialog.
- You can construct your own custom dialog boxes by using the Windows Forms Designer in Visual Studio.
- Add controls such as Label, Textbox, and Button to customize dialog boxes to your specific needs.
- The .NET Framework also includes predefined dialog boxes, such as File Open and message boxes, which you can adapt for your own applications.

Handling Events

VB.Net is an event-driven language. There are mainly two types of events –

- Mouse events
- Keyboard events

Handling Mouse Events

Mouse events occur with mouse movements in forms and controls. Following are the various mouse events related with a Control class –

- MouseDown – it occurs when a mouse button is pressed
- MouseEnter – it occurs when the mouse pointer enters the control
- MouseHover – it occurs when the mouse pointer hovers over the control
- MouseLeave – it occurs when the mouse pointer leaves the control
- MouseMove – it occurs when the mouse pointer moves over the control
- MouseUp – it occurs when the mouse pointer is over the control and the mouse button is released
- MouseWheel – it occurs when the mouse wheel moves and the control has focus

The event handlers of the mouse events get an argument of type MouseEventArgs. The MouseEventArgs object is used for handling mouse events. It has the following properties –

- Buttons – indicates the mouse button pressed
- Clicks – indicates the number of clicks
- Delta – indicates the number of detents the mouse wheel rotated
- X – indicates the x-coordinate of mouse click
- Y – indicates the y-coordinate of mouse click

Handling Keyboard Events

Following are the various keyboard events related with a Control class –

- KeyDown – occurs when a key is pressed down and the control has focus
- KeyPress – occurs when a key is pressed and the control has focus
- KeyUp – occurs when a key is released while the control has focus

The event handlers of the KeyDown and KeyUp events get an argument of type KeyEventArgs. This object has the following properties –

- Alt – it indicates whether the ALT key is pressed
- Control – it indicates whether the CTRL key is pressed
- Handled – it indicates whether the event is handled
- KeyCode – stores the keyboard code for the event

- KeyData – stores the keyboard data for the event
- KeyValue – stores the keyboard value for the event
- Modifiers – it indicates which modifier keys (Ctrl, Shift, and/or Alt) are pressed
- Shift – it indicates if the Shift key is pressed

The event handlers of the KeyDown and KeyUp events get an argument of type KeyEventArgs. This object has the following properties –

- Handled – indicates if the KeyPress event is handled
- KeyChar – stores the character corresponding to the key pressed

UNIT - III

Windows Form Controls

All the windows Forms controls inherit the common properties, methods and events of the Control class.

Properties of Control Class

Property	Description
Anchor	Obtains or sets the edges of the parent control to which the child control is bound and determines how the control is resized with respect to the size of its parent control
BackColor	Obtains or sets the background color of the control
BackgroundImage	Obtains or sets the background

Text Box

Text Box is used to accept textual input from the user. The user can add strings, numerical values and a combination of those, but Images and other multimedia content are not supported.

Label

It is used to show any text to the user, typically the text in a label does not change while the application is running.

Button

It is used as a standard Windows Button. In most cases, the Button Control is used to generate a click event, its name, size and appearance are not changed in the runtime.

ListBox

As the name suggests, this control works as a way to display a list of items on the application.

Users can select any options from the list.

Combo Box

It is similar to the list but it works as a dropdown for the user. A user can enter both text in the box or he can click on the downwards arrow on the right side and select any item.

Radio Button

Radio Button is one of the popular ways of limiting the user to pick just one option. The programmer can set any of the buttons as default if needed. These buttons are grouped together.

Checkbox

Checkboxes are similar to radio buttons in the way that they are also used in groups, however, a user can select more than one item in the group.

PictureBox

This VB.Net control is used to show images and graphics inside a form. The image can be of any supported format and we can select the size of the object in the form too.

Progress Bar

This is used to show a Windows Progress bar, this bar can represent an ongoing process such as moving a file or exporting a document.

Panel Control

The **Panel control** is a container of other **controls**. The **Panel control** is displayed by default without any borders at run time. Drag and drop **Panel control** from the toolbox on the window Form.

Timer Control

The **timer control** is a looping **control** used to repeat any task in a given time interval. Once the **timer** is enabled, it generates a tick event handler to perform any defined task in its time interval property.

GroupBox Control

A **GroupBox control** is a container **control** that is used to place Windows Forms child **controls** in a **group**. The purpose of a **GroupBox** is to define user interfaces where we can categories related **controls** in a **group**.

