NATIONAL OPEN UNIVERSITY OF NIGERIA

COURSE CODE : CIT 351

COURSE TITLE: C# PROGRAMMING

COURSE GUIDE

CIT 351
C# PROGRAMMING

Course Developer/Writer	Vivian Nwaocha National Open University of Nigeria Lagos
Programme Leader	Professor Afolabi Adebanjo National Open University of Nigeria Lagos
Course Coordinator	Vivian Nwaocha National Open University of Nigeria, Lagos

NATIONAL OPEN UNIVERSITY OF NIGERIA

National Open University of Nigeria
Headquarters
14/16 Ahmadu Bello Way
Victoria Island
Lagos

Abuja Annex
245 Samuel Adesujo Ademulegun Street
Central Business District
Opposite Arewa Suites
Abuja

e-mail: @nou.edu.ng
URL: .nou.edu.ng

National Open University of Nigeria 2009
First Printed --------

ISBN:

Printed by ……………..

For
National Open University of Nigeria

CIT 351 C# PROGRAMMING

TABLE OF CONTENTS PAGE

Introduction

CIT 351: C# Programming is a 2 credit unit course for students studying towards acquiring a Bachelor of Science in Computer Science and other related disciplines.
The course is divided into 4 modules and 15 study units. It will introduce students to fundamental concepts of C# Programming, .NET framework and Visual Studio.NET. This course also provides information on C# data types, syntax, expressions and C# applications.

At the end of this course, it is expected that students should be able to understand, explain and be adequately equipped with fundamental notions of C# Programming. The course guide therefore gives you an overview of what the course; CIT 351 is all about, the textbooks and other materials to be referenced, what you are expected to know in each unit, and how to work through the course material. It suggests the general strategy to be adopted and also emphasizes the need for self-assessment and tutor marked assignment. There are also tutorial classes that are linked to this course and students are advised to attend.

Brief overview of the Course

C# is a general purpose, object-oriented, component-based programming language. As a general purpose language, you have a number of ways to apply C# to accomplish many different tasks. You can build web applications with ASP.NET, desktop applications with Windows Presentation Foundation (WPF), or build mobile applications for Windows Phone. Other applications include code that runs in the cloud via Windows Azure, and iOS, Android, and Windows Phone support with the Xamarin platform. However, from a general programming perspective, you can do a lot with C#.

What you will learn in this Course

The overall aim of this course, CIT 351, is to introduce you to basic concepts of C# programming in order to enable you create C# projects. This course provides hands-on, case study examples, and reference materials designed to enhance your programming skills. In the course of your studies, you will be equipped with definitions of common terms, characteristics and applications of object-oriented programming using C#. You will also learn about .NET framework and visualstudio.NET. Finally, you will learn about C# expressions and operators.

Course Aim

This course aims to introduce students to the basic concepts and features of C# programming. It is hoped that the knowledge would enhance the programming expertise of students to enable them develop C# based applications.

Course Objectives

It is important to note that each unit has specific objectives. Students should study them carefully before proceeding to subsequent units. Therefore, it may be useful to refer to these objectives in the course of your study of the unit to assess your progress. You should always look at the unit objectives after completing a unit. In this way, you can be sure that you have done what is required of you by the end of the unit.

However, below are overall objectives of this course. On completing this course, you should be able to:

• Explain the term C# (C Sharp)
• Clarify the origin of C#
• List the versions of C#
• Outline the basic features of C#
• Identify the aims of ECMA
• Outline the design goals
• List the categories of C# Type system
• Explain the concept of boxing and unboxing
• Declare a variable in C#
• Describe the naming conventions
• Identify common variables in C#
• Describe statements, statement blocks and comments
• Identify the 'Hello World' source code
• State the minimal requirement to use C#
• Outline the steps involved in building console applications
• State the procedure for building and running GUI applications
• Outline the steps required to build a code library
• Create a C# project in VisualStudio.NET
• Identify C# expressions
• List common operators used in C#
• Be exposed to a .NET and Object oriented programming language. Also
• Be able to design and implement a desktop application software.

- Be able to use all the features of object oriented programming language effectively.
- Be able to design and develop a web based application package.
- Be able to develop a database management system that will interface the C# programming language with database management system.
- Be able to implement simple algorithm in computer science processes

Working through this Course

To complete this course, you are required to study all the units, the recommended text books, and other relevant materials. Each unit contains some self-assessment exercises and tutor marked assignments, and at some point in this course, you are required to submit the tutor marked assignments. There is also a final examination at the end of this course. Stated below are the components of this course and what you have to do.

Course Materials

The major components of the course are:
1. Course Guide
2. Study Units
3. Text Books
4. Assignment File
5. Presentation Schedule

Study Units

There are 15 study units and 5 modules in this course. They are:

Recommended Texts

These texts will be of enormous benefit to you in learning this course:

1. Abelson, H and Gerald J. S. (1997). *Structure and Interpretation of Computer Programs*. The MIT Press.
2. Anggraini, A. R., & Oliver, J. (2019). Visual Studio. *Journal of Chemical Information and Modeling*, *53*(9), 1689–1699.
3. Armstrong, Deborah J. (2006). "The Quarks of Object-Oriented Development". *Communications of the ACM* **49** (2): 123–128. http://portal.acm.org/citation.cfm?id=1113040. Retrieved 2006-08-08.
4. Booch, Grady (1997). *Object-Oriented Analysis and Design with Applications*. Addison-Wesley.
5. Date, C. J and Hugh, D. (2006). Foundation for Future Database Systems: The Third Manifesto (2nd Edition)
6. Date, C. J and Hugh, D. (2007). Introduction to Database Systems: The Sixth Manifesto (6th Edition)
7. Eeles, P and Oliver, S. (1998). *Building Business Objects*. John Wiley & Sons.
8. Gamma, Erich; Richard Helm, Ralph Johnson, John Vlissides (1995). *Design Patterns: Elements of Reusable Object Oriented Software*. Addison-Wesley.
9. Harmon, Paul; William Morrissey (1996). *The Object Technology Casebook - Lessons from Award-Winning Business Applications*. John Wiley & Sons.
10. Jacobson, Ivar (1992). *Object-Oriented Software Engineering: A Use Case-Driven Approach*. Addison-Wesley.
11. John C. Mitchell, *Concepts in programming languages*, Cambridge University Press, 2003, p.278
12. Joyce, F. (2006). Microsoft Visual C#.NET with Visual Studio 2005
13. Kay, Alan. *The Early History of Smalltalk*. http://gagne.homedns.org/%7etgagne/contrib/EarlyHistoryST.html.
14. Martin, A and Luca, C. (2005). *A Theory of Objects*.
15. Meyer, Bertrand (1997). *Object-Oriented Software Construction*. Prentice Hall.
16. Michael Lee Scott (2006). *Programming language pragmatics*, (2nd Edition) p.

470
17. Perkins, B., Hammer, J. V., & Reid, J. D. (2018). Beginning C# 7 Programming with Visual Studio® 2017. In *Beginning C# 7 Programming with Visual Studio® 2017*.
18. Pierce, Benjamin (2002). *Types and Programming Languages*. MIT Press.
19. Rumbaugh, James; Michael Blaha, William Premerlani, Frederick Eddy, William Lorensen (1991). *Object-Oriented Modeling and Design*. Prentice Hall.
20. Schreiner, A. (1993). *Object oriented programming with ANSI-C*.
21. Studio, V. (2013). *Fundamentals of Computer Programming with C# (The Bulgarian C# Programming Book) by Svetlin Nakov & Co. http://www.introprogramming.info*.

22. Taylor, David A. (1992). *Object-Oriented Information Systems - Planning and Implementation*. John Wiley & Sons.
23. Trofimov, M. (1993) *OOOP - The Third "O" Solution: Open OOP*. First Class, OMG , Vol. 3, issue 3, p.14.

Assignment File

The assignment file will be given to you in due course. In this file, you will find all the details of the work you must submit to your tutor for marking. The marks you obtain for these assignments will count towards the final mark for the course. Altogether, there are 15 tutor marked assignments for this course.

Presentation Schedule

The presentation schedule included in this course guide provides you with important dates for completion of each tutor marked assignment. You should therefore endeavour to meet the deadlines.

Assessment

There are two aspects to the assessment of this course. First, there are tutor marked assignments; and second, the written examination. Therefore, you are expected to take note of the facts, information and problem solving gathered during the course. The tutor marked assignments must be submitted to your tutor for formal assessment, in accordance to the deadline given. The work submitted will count for 40% of your total course mark. At the end of the course, you will need to sit for a final written examination. This examination will account for 60% of your total score.

Tutor Marked Assignments (TMAs)

There are 15 TMAs in this course. You need to submit all the TMAs. The best 4 will therefore be counted. When you have completed each assignment, send them to your tutor as soon as possible and make certain that it gets to your tutor on or before the stipulated deadline. If for any reason you cannot complete your assignment on time, contact your tutor before the assignment is due to discuss the possibility of extension. Extension will not be granted after the deadline, unless on extraordinary cases.

Final Examination and Grading

The final examination for CIT 351 will be of last for a period of 2 hours and have a value of 60% of the total course grade. The examination will consist of questions which reflect the self assessment exercise and tutor marked assignments that you have previously encountered. Furthermore, all areas of the course will be examined. It would be better to use the time between finishing the last unit and sitting for the examination, to revise the entire course. You might find it useful to review your TMAs and comment on them before the examination. The final examination covers information from all parts of the course.

Course marking Scheme

The following table includes the course marking scheme

Table 1 Course Marking Scheme

Assessment	**Marks**
Assignment 1 – 15	15 assignments, 40% for the best 4 Total = 10% X 4 = 40%
Final Examination	60% of overall course marks
Total	100% of Course Marks

Course Overview

This table indicates the units, the number of weeks required to complete them and the assignments.

Table 2: Course Organizer

Unit	**Title of Work**	**Weeks Activity**	**Assessment (End of Unit)**
	Course Guide		Week 1
Module 1		**Introduction to C#**	
Unit 1	Introduction to C#	Week 1	Assignment 1
Unit 2	ECMA Standards	Week 2	Assignment 2
Unit 3	C# and .Net Framework	Week 3	Assignment 3
Module 2		**Fundamentals of C#**	
Unit 1	Introduction to C#.Net Programming	Week 4	Assignment 4
Unit 2	Variables, Data Types and Expressions	Week 5	Assignment 5
Unit 3	C#.Net Operators	Week 6	Assignment 6
Module 3		**Flow Control**	
Unit 1	Flow Control	Week 7	Assignment 7
Unit 2	C#.Net Looping and Iterations	Week 8	Assignment 8
Unit 3	C#.Net Type Conversion	Week 9	Assignment 9
Module 4		**Arrays, String Manipulations and Functions**	
Unit 1	C#.Net Arrays	Week 10	Assignment 10
Unit 2	C#.Net Strings manipulations	Week 11	Assignment 11
Unit 3	Functions	Week 12	Assignment 12
Module 5		**Object Oriented, Windows and database programming**	
Unit 1	Objects and Classes	Week 13	Assignment 13
Unit 2	C# Windows Application Development	Week 14	Assignment 14
Unit 3	Files and Databases	Week 15	Assignment 15

How to get the most out of this course

In distance learning, the study units replace the university lecturer. This is one of the huge advantages of distance learning mode; you can read and work through specially designed study materials at your own pace and at a time and place that is most convenient. Think of it as reading from the teacher, the study guide indicates what you ought to study, how to study it and the relevant texts to consult. You are provided with exercises at appropriate points, just as a lecturer might give you an in-class exercise.

Each of the study units follows a common format. The first item is an introduction to the subject matter of the unit and how a particular unit is integrated with the other units and the course as a whole. Next to this is a set of learning objectives. These learning objectives are meant to guide your studies. The moment a unit is finished, you must go back and check whether you have achieved the objectives. If this is made a habit, then you will increase your chances of passing the course. The main body of the units also guides you through the required readings from other sources. This will usually be either from a set book or from other sources. Self assessment exercises are provided throughout the unit, to aid personal studies and answers are provided at the end of the unit. Working through these self tests will help you to achieve the objectives of the unit and also prepare you for tutor marked assignments and examinations. You should attempt each self test as you encounter them in the units.

The following are practical strategies for working through this course

1. Read the course guide thoroughly
2. Organise a study schedule. Refer to the course overview for more details. Note the time you are expected to spend on each unit and how the assignment relates to the units. Important details, e.g. details of your tutorials and the date of the first day of the semester are available. You need to gather together all these information in one place such as a diary, a wall chart calendar or an organizer. Whatever method you choose, you should decide on and write in your own dates for working on each unit.
3. Once you have created your own study schedule, do everything you can to stick to it. The major reason that students fail is that they get behind with their course works. If you get into difficulties with your schedule, please let your tutor know before it is too late for help.
4. Turn to Unit 1 and read the introduction and the objectives for the unit.
5. Assemble the study materials. Information about what you need for a unit is given in the table of content at the beginning of each unit. You will almost always need both the study unit you are working on and one of the materials recommended for further

readings, on your desk at the same time.
6. Work through the unit, the content of the unit itself has been arranged to provide a sequence for you to follow. As you work through the unit, you will be encouraged to read from your set books.
7. Keep in mind that you will learn a lot by doing all your assignments carefully. They have been designed to help you meet the objectives of the course and will help you pass the examination.
8. Review the objectives of each study unit to confirm that you have achieved them. If you are not certain about any of the objectives, review the study material and consult your tutor.
9. When you are confident that you have achieved a unit's objectives, you can start on the next unit. Proceed unit by unit through the course and try to pace your study so that you can keep yourself on schedule.
10. When you have submitted an assignment to your tutor for marking, do not wait for its return before starting on the next unit. Keep to your schedule. When the assignment is returned, pay particular attention to your tutor's comments, both on the tutor marked assignment form and also written on the assignment. Consult you tutor as soon as possible if you have any questions or problems.
11. After completing the last unit, review the course and prepare yourself for the final examination. Check that you have achieved the unit objectives (listed at the beginning of each unit) and the course objectives (listed in this course guide).

Tutors and Tutorials

There are 8 hours of tutorial provided in support of this course. You will be notified of the dates, time and location together with the name and phone number of your tutor as soon as you are allocated a tutorial group. Your tutor will mark and comment on your assignments, keep a close watch on your progress and on any difficulties you might encounter and provide assistance to you during the course.

You must mail your tutor marked assignment to your tutor well before the due date. At least two working days are required for this purpose. They will be marked by your tutor and returned to you as soon as possible. Do not hesitate to contact your tutor by telephone, e-mail or discussion board if you need help. The following might be circumstances in which you would find help necessary: contact your tutor if:

• You do not understand any part of the study units or the assigned readings.
• You have difficulty with the self test or exercise.
• You have questions or problems with an assignment, with your tutor's comments on an

assignment or with the grading of an assignment.

You should endeavour to attend the tutorials. This is the only opportunity to have face-to-face contact with your tutor and ask questions which are answered instantly. You can raise any problem encountered in the course of your study. To gain the maximum benefit from the course tutorials, have some questions handy before attending them. You will learn a lot from participating actively in discussions. GOODLUCK!

Course Code	CIT 351
Course Title	C# Programming
Course Developer/Writer	Vivian Nwaocha National Open University of Nigeria Lagos
Programme Leader	Professor Afolabi Adebanjo National Open University of Nigeria Lagos
Course Coordinator	Vivian Nwaocha National Open University of Nigeria Lagos

NATIONAL OPEN UNIVERSITY OF NIGERIA

MAIN COURSE

National Open University of Nigeria
Headquarters
14/16 Ahmadu Bello Way
Victoria Island
Lagos

Abuja Annex
245 Samuel Adesujo Ademulegun Street
Central Business District
Opposite Arewa Suites
Abuja

e-mail: @nou.edu.ng
URL: .nou.edu.ng

National Open University of Nigeria 2009
First Printed 2009
ISBN: All Rights Reserved

Printed by …………….. For National Open University of Nigeria

TABLE OF CONTENTS PAGE

UNIT 1 INTRODUCTION TO C#

CONTENTS

1.0 INTRODUCTION

You are welcome to the world of C# programming language. This unit gives you a basic idea about C#. However there will be no difficulty in learning this language if you are a fresher, because this unit will elucidate the basic concepts and features of C# right from the beginning.

Even if you have gained previous programming experience with any conventional programming language, it is recommended that you go through the entire unit systematically to gain some insight of the course.

2.0 OBJECTIVES

At the end of this unit, you should be able to:

- Explain the term C#
- Give a summary of the origin of C#
- Identify the advancements in C#
- State the versions of C#
- Outline features of C#

3.0 MAIN CONTENT

3.1 What is C#?

C# (pronounced "C Sharp") is a **multi-paradigm programming language** encompassing **imperative**, **functional**, **generic**, **object-oriented** (**class-based**), and **component-oriented** programming disciplines.

It is a high-level language that combines some of the best features of modern programming languages such as Java, C++ VB.Net, Delphi, and C. C# is an object-oriented language with single inheritance but multiple interfaces per class. It supports component-based programming by properties (smart fields), events and delegates (enhanced function pointers).

C# is one of the most popular programming languages designed for the **Common Language Infrastructure** used by millions of developers worldwide. It supports all features of an Object Oriented language such as abstraction, encapsulation, inheritance, and polymorphism features. It is fully interoperable with other .NET languages such as VB.NET, Eiffel.NET or Oberon.NET and executed in a special environment called the **Common Language Runtime (CLR),** a part of the platform .Net Framework that contains bundles of standard libraries providing basic functionality, compilers, debuggers and other development tools.

SELF ASSESSMENT EXERCISE

Give a concise description of C#

____
____
____
____

3.2 Origin

Microsoft, with **Anders Hejlsberg** as Chief Engineer, created C# as part of their **.NET** initiative and subsequently opened its **specification** in 2002 via the **ECMA (Ecma-334)**. Thus, the language is open to implementation by other parties. Other implementations include **Mono** and **DotGNU**. Microsoft's original plan was to create a rival to Java, named J++ but this was abandoned to create C#, codenamed "Cool". Microsoft submitted

C# to the ECMA standards group mid-2000. C# is intended to be a simple, modern, general-purpose, object-oriented programming language. Its development team is led by **Anders Hejlsberg**, the designer of **Borland**'s **Turbo Pascal**. It has an object-oriented **syntax** based on **C++**.

3.3 Advancements in C#

During the development of .NET Framework, the **class libraries** were originally written in a language/compiler 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#.

C#'s principal designer and lead architect at Microsoft is **Anders Hejlsberg**, who was previously involved with the design of **Turbo Pascal**, **CodeGear Delphi** (formerly 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# programming language itself.

3.3.1 C# Versions

In the course of its development, C# has gone through several versions:

- **C# 1.0 -** introduced 2000 / released January 2002
- **C# 1.1 -** released April 2003
- **C# 2.0 -** released November 2005
- **C# 3.0 -** released November 2007
- **C# 4.0 -** in development
- C# 4.0 – released April 2010
- C# 5.0 – released August 2012
- C# 6.0 – released July 2015
- C# 7.0 – released March 2017
- C# 7.1 – released August 2017
- C# 7.2 – released November 2017
- C# 7.3 – released May 2018
- C# 8.0 – released September 2019

3.4 Language Name

The name C#, pronounced as "C sharp", was inspired from **musical notation** where a **sharp** indicates that the written note should be made a half-step higher in pitch. This is similar to the language name of **C++**, where "++" indicates that a variable should be incremented by 1. The sharp symbol also resembles a **ligature** of four "+" symbols (in a two-by-two grid), further implying that the language is an increment of **C++**.

Due to technical limitations of display (fonts, browsers, etc.) and the fact that the sharp symbol (♯ , U+266F, MUSIC SHARP SIGN) is not present on the standard keyboard, the **number sign** (#, U+0023, NUMBER SIGN) was chosen to represent the sharp symbol in the written name of the programming language. This convention is reflected in the ECMA-334 C# Language Specification. However, when it is practical to do so (for example, in advertising or in box art), Microsoft uses the intended musical symbol.

3.5 C# Features

Some notable distinguishing features of C# are:

• There are no global variables or functions. All methods and members must be declared within classes. Static members of public classes can substitute for global variables and functions.
• C# supports strongly type implicit variable declarations with the keyword (var), and implicitly typed arrays with the keyword (new[]) followed by a collection initializer
• Local variables cannot shadow variables of the enclosing block, unlike C and C++. **Variable shadowing** is often considered confusing by C++ texts.
• C# supports a strict **Boolean datatype**, bool. Statements that take conditions, such as while and if, require an expression of a boolean type. While C++ also has a boolean type, it can be freely converted to and from integers, and expressions such as if(a) require only that a is convertible to bool, allowing a to be an int, or a pointer. C# disallows this "integer meaning true or false" approach on the grounds that forcing programmers to use expressions that return exactly bool can prevent certain types of programming mistakes such as if (a = b) (use of = instead of ==).
• In C#, memory address pointers can only be used within blocks specifically marked as *unsafe*, and programs with unsafe code need appropriate permissions to run. Most object access is done through safe object references, which always either point to a "live" object or have the well-defined **null** value; it is impossible to obtain a reference to a "dead" object (one which has been garbage collected), or to a random block of memory. An

unsafe pointer can point to an instance of a value-type, array, string, or a block of memory allocated on a stack. Code that is not marked as unsafe can still store and manipulate pointers through the System. IntPtr type, but it cannot dereference them.

• Managed memory cannot be explicitly freed; instead, it is automatically garbage collected. Garbage collection addresses **memory leaks** by freeing the programmer of responsibility for releasing memory which is no longer needed. C# also provides direct support for deterministic finalization with the using statement (supporting the **Resource Acquisition Is Initialization** idiom).

• **Multiple inheritance** is not supported, although a class can implement any number of interfaces. This was a design decision by the language's lead architect to avoid complication, avoid **dependency hell** and simplify architectural requirements throughout CLI..

• **Checked exceptions** are not present in C# in contrast to Java which has been a conscious decision based on the issue of scalability and versionability.

• C# is more **typesafe** than C++. The only implicit conversions by default are those which are considered safe, such as widening of integers and conversion from a derived type to a base type. This is enforced at compile-time, during **JIT**, and, in some cases, at runtime. There are no implicit conversions between booleans and integers, nor between enumeration members and integers (except for literal 0, which can be implicitly converted to any enumerated type). Any user-defined conversion must be explicitly marked as explicit or implicit, unlike C++ **copy constructors** and conversion operators, which are both implicit by default.

• **Enumeration** members are placed in their own **scope**.

• C# provides **properties** as **syntactic sugar** for a common pattern in which a pair of methods, **accessor (getter) and mutator (setter)** encapsulate operations on a single **attribute** of a class.

• Full type **reflection** and discovery is available

• C# currently has 77 **reserved words**.

4.0 CONCLUSION

In this unit, we defined some basic concepts of C#. We also looked at the advancements in C# as well as the language name.

5.0 SUMMARY

We hope you enjoyed this unit. This unit provided an overview of C#: basic definition, features, versions and language name. Now, let us attempt the questions below.

6.0 TUTOR MARKED ASSIGNMENT

Outline at least 5 distinguishing features of C#.

7.0 REFERENCES/FURTHER READINGS

1. Archer, Tom (2001). *Inside C#*. Microsoft Press. ISBN 0-7356-1288-9.
2. Cornelius, Barry (December 1, 2005). "Java 5 catches up with C#". University of Oxford Computing Services.
3. Ecma International. June 2006.*C# Language Specification* (4th ed.).
4. Guthrie, Scott (November 28, 2006). "What language was ASP.Net originally written in?"
5. Hamilton, Naomi (October 1, 2008). "The A-Z of Programming Languages: C#". Computerworld.
6. Horton, Anson (2006-09-11). "C# XML documentation comments FAQ".
7. Kovacs, James (September 07, 2007). "C#/.NET History Lesson".
8. Microsoft. September 4, 2003."Visual C#.net Standard" (JPEG).
9. Microsoft (June 18, 2009) *C# Language Reference*.
10. O' Reilly. (2002).*C# Language Pocket Reference*. ISBN 0-596-00429-X.
11. Petzold, Charles (2002). *Programming Microsoft Windows with C#*. Microsoft Press. ISBN 0-7356-1370-2.
12. Steinman, Justin (November 7, 2006). "Novell Answers Questions from the Community".
13. Stallman, Richard (June 26, 2009). "Why free software shouldn't depend on Mono or C#".
14. Simon, Raphael; Stapf, Emmanuel; Meyer, Bertrand (June 2002). "Full Eiffel on the .NET Framework".
15. "The ECMA C# and CLI Standards". 2009 -07-06.
16. Zander, Jason (November 23, 2007). "Couple of Historical Facts".

UNIT 2 ECMA STANDARDS

CONTENTS

1.0 INTRODUCTION

In unit 1, we gave an overview of 'C# programming' as well as its basic features. This unit provides information about the ECMA international standards.

2.0 OBJECTIVES

• At the end of this unit, you should be able to:
• State what ECMA stands for
• Identify the role of ECMA International
• State the aims of ECMA
• Identify what ECMA specifies
• Discover what it does not specify
• State the design goals for C#

3.0 MAIN CONTENT

3.1 ECMA International Standards

ECMA stands for European Computer Manufacturers Association, an international association founded in 1961 that is dedicated to establishing standards in the information and communications fields. ECMA is a liaison organization to ISO. Ecma International is thus dedicated to the standardization of Information and Communication Technology (ICT) and Consumer Electronics (CE).

SELF ASSESSMENT EXERCISE

What does the acronym ECMA signify?

3.2 Aims of ECMA

The aims of ECMA are:

• To develop, in co-operation with the appropriate National, European and International organizations Standards and Technical Reports in order to facilitate and standardize the use of Information Communication Technology (ICT) and Consumer Electronics (CE).
• To encourage the correct use of Standards by influencing the environment in which they are applied.
• To publish these Standards and Technical Reports in electronic and printed form; the publications can be freely copied by all interested parties without restrictions.

This International Standard specifies the form and establishes the interpretation of programs written in the C# programming language.

3.2.1 What ECMA Specifies

ECMA specifies:

• The representation of C# programs;
• The syntax and constraints of the C# language;
• The semantic rules for interpreting C# programs;
• The restrictions and limits imposed by a conforming implementation of C#.

3.2.2 What ECMA Does Not Specify

This International Standard does not specify:
• The mechanism by which C# programs are transformed for use by a data-processing system;
• The mechanism by which C# applications are invoked for use by a data-processing system;
• The mechanism by which input data are transformed for use by a C# application;
• The mechanism by which output data are transformed after being produced by a C# application;
• The size or complexity of a program and its data that will exceed the capacity of any specific data-processing system or the capacity of a particular processor;
• All minimal requirements of a data-processing system that is capable of supporting a conforming implementation.

3.3 Design Goals

The Ecma standard lists these design goals for C#.

• C# is intended to be a simple, modern, general-purpose, object-oriented programming language.
• The language, and implementations thereof, should provide support for software engineering principles such as **strong type checking**, **array bounds checking**, detection of attempts to use uninitialized variables, and **automatic garbage collection**. Software robustness, durability, and programmer productivity are important.
• The language is intended for use in developing **software components** suitable for deployment in distributed environments.
• Source code portability is very important, as is programmer portability, especially for those programmers already familiar with C and C++.
• Support for **internationalization** is very important.
• C# is intended to be suitable for writing applications for both hosted and **embedded systems**, ranging from the very large that use sophisticated **operating systems**, down to the very small having dedicated functions.
• Although C# applications are intended to be economical with regard to memory and **processing power** requirements, the language was not intended to compete directly on performance and size with C or assembly language.

4.0 CONCLUSION

From our studies in this unit, it is vital to remember that the ECMA International Standard specifies the form and establishes the interpretation of programs written in the C# programming language. It is equally worth noting that ECMA has some specific design goals.

5.0 SUMMARY

In this unit, we looked at the ECMA international standards, its' aims; what it specifies and what it does not specify as well as it's design goals. We hope you found the unit enlightening. To assess your comprehension, attempt the questions below.

6.0 TUTOR MARKED ASSIGNMENT

What does ECMA specify?
Outline the aims of ECMA

7.0 REFERENCES/FURTHER READINGS

1. Archer, Tom (2001). *Inside C#*. Microsoft Press. ISBN 0-7356-1288-9.
2. Cornelius, Barry (December 1, 2005). "Java 5 catches up with C#". University of Oxford Computing Services.
3. Ecma International. June 2006.*C# Language Specification* (4th ed.).
4. Guthrie, Scott (November 28, 2006). "What language was ASP.Net originally written in?"
5. Hamilton, Naomi (October 1, 2008). "The A-Z of Programming Languages: C#". Computerworld.
6. Horton, Anson (2006-09-11). "C# XML documentation comments FAQ".
7. Kovacs, James (September 07, 2007). "C#/.NET History Lesson".
8. Microsoft. September 4, 2003."Visual C#.net Standard" (JPEG).
9. Microsoft (June 18, 2009) *C# Language Reference*.
10. O' Reilly. (2002).*C# Language Pocket Reference*. ISBN 0-596-00429-X.
11. Petzold, Charles (2002). *Programming Microsoft Windows with C#*. Microsoft Press. ISBN 0-7356-1370-2.
12. Steinman, Justin (November 7, 2006). "Novell Answers Questions from the Community".
13. Stallman, Richard (June 26, 2009). "Why free software shouldn't depend on Mono or

C#".
14. Simon, Raphael; Stapf, Emmanuel; Meyer, Bertrand (June 2002). "Full Eiffel on the .NET Framework".
15. "The ECMA C# and CLI Standards". 2009 -07-06.
16. Zander, Jason (November 23, 2007). "Couple of Historical Facts".

UNIT 3 C# AND .NET FRAMEWORK

CONTENTS

1.0 INTRODUCTION

The initial task we have in this unit is to identify the target platform of C#, so that you can see the relationship between C# and .NET framework. You might be tempted to think you're never going to get to grips with this. But don't worry - after a few lessons, things will start to feel familiar, and you will gain more confidence.

2.0 OBJECTIVES

By the end of this unit, you'll have learnt about the following:
• The target platform of C#
• The relationship between C# and .NET framework

3.0 MAIN CONTENT

3.1 C# Platform

In brief, C# (unlike C++, PERL, COBOL, Pascal, etc.) is a language that targets one and only one platform. This platform is the .NET Framework. However, the .NET Framework itself is a computing platform that is designed to be hosted by *any* operating system. It is the most common version of the .Net Framework. At the time of this writing, the .Net runs primarily on Microsoft Windows which led to a family of .NET platforms targeting mobile computing, embedded devices, rich internet applications, alternative operating systems, and web-browser plug-ins and I know of two other major operating systems for which a version of the .NET Framework is being developed. So you can see that although C# is designed to target only the Framework, the Framework itself is flexible enough to run your C# programs on many types of systems.

SELF ASSESSMENT EXERCISE

Give a brief description of the C# platform.

3.2 Relationship between C# and .NET Framework

The relationship between C# and the .NET Framework is somewhat unique. In a way it is similar to the relationship between Java and the Java Virtual Machine, however there are several major differences. First, C# is not the only language that can be used to write .NET Framework applications (called Managed Applications). Second, .NET or managed applications run in native machine-language and are not interpreted. Third, C# or managed applications do not run in a sandbox. Fourth, .NET has a Common Language Runtime (CLR) which is a virtual component of .NET framework. Fifth, C# is a part of .NET that has different features like Boolean Conditions. Standard Library, Indexers, Conditional Compilation, etc. Sixth, the basic usage of C# is essentially concentrated on desktop-based applications and in the case of .NET, it is used to develop Microsoft based applications.

What you should take away from this unit, as a programmer learning C#, is that C# is a programming language (that you will find simple to master); however, much of what you can do with C# is really more a part of the .NET Framework itself.

4.0 CONCLUSION

C# is a language that targets only one platform. This platform is the .NET Framework. C# is not the only language that can be used to write .NET Framework applications (called Managed Applications). The .NET or managed applications run in native machine-language and are not interpreted. C# or managed applications do not run in a sandbox. .

NET has a Common Language Runtime (CLR) which is a virtual component of .NET framework. C# is a part of .NET that has different features like Boolean Conditions. Standard Library, Indexers, Conditional Compilation, etc. The basic usage of C# is essentially concentrated on desktop-based applications and in the case of .NET, it is used to develop Microsoft based applications.

5.0 SUMMARY

In this unit, we considered the .NET framework which is the platform for C#. Hoping that you understood the topics discussed, you may now attempt the questions below.

6.0 TUTOR MARKED ASSIGNMENT

Outline 2 key differences between C# and .NET framework Give a brief description of the .NET framework

7.0 REFERENCES/FURTHER READINGS

1. Archer, Tom (2001). *Inside C#*. Microsoft Press. ISBN 0-7356-1288-9.
2. Cornelius, Barry (December 1, 2005). "Java 5 catches up with C#". University of Oxford Computing Services.
3. Ecma International. June 2006.*C# Language Specification* (4th ed.).
4. Guthrie, Scott (November 28, 2006). "What language was ASP.Net originally written in?"
5. Hamilton, Naomi (October 1, 2008). "The A-Z of Programming Languages: C#". Computerworld.
6. Horton, Anson (2006-09-11). "C# XML documentation comments FAQ".
7. Kovacs, James (September 07, 2007). "C#/.NET History Lesson".
8. Microsoft. September 4, 2003."Visual C#.net Standard" (JPEG).
9. Microsoft (June 18, 2009) *C# Language Reference*.
10. O' Reilly. (2002).*C# Language Pocket Reference*. ISBN 0-596-00429-X.
11. Petzold, Charles (2002). *Programming Microsoft Windows with C#*. Microsoft Press. ISBN 0-7356-1370-2.
12. Steinman, Justin (November 7, 2006). "Novell Answers Questions from the Community".
13. Stallman, Richard (June 26, 2009). "Why free software shouldn't depend on Mono or C#".
14. Simon, Raphael; Stapf, Emmanuel; Meyer, Bertrand (June 2002). "Full Eiffel on the .NET Framework".
15. "The ECMA C# and CLI Standards". 2009 -07-06.
16. Zander, Jason (November 23, 2007). "Couple of Historical Facts".

MODULE 2 FUNDAMENTALS OF C#.Net

UNIT 1: Introduction To C#.Net Programming

1.0 INTRODUCTION

C# is a general purpose programming language with a wide variety of application domains. It is object-oriented and organized around objects rather than "actions" and data rather than logic. It is also a component-based programming tool used for Design and development of computer-based systems with the help of reusable software components.

2.0 OBJECTIVES

At the end of this unit, you should be able to:

- know the achievable tasks in C#,Net
- start a new project in C#.Net
- create a new Console application
- get to know how to use comments in C#

3.0 MAIN CONTENT

3.1 What is .Net?

- .NET is a platform that includes languages, a runtime, and framework libraries, allowing developers to create many types of applications.
- C# is one of the .NET languages, which also includes Visual Basic, F#, C++, and more.
  Tasks that can be achieved with C# includes:
- Desktop applications with Windows Presentation Foundation (WPF).
- Web applications with ASP.NET,
- Mobile applications for Windows Phone.

Other applications include code that runs in the cloud via Windows Azure, and iOS, Android, and Windows Phone support with the Xamarin platform.

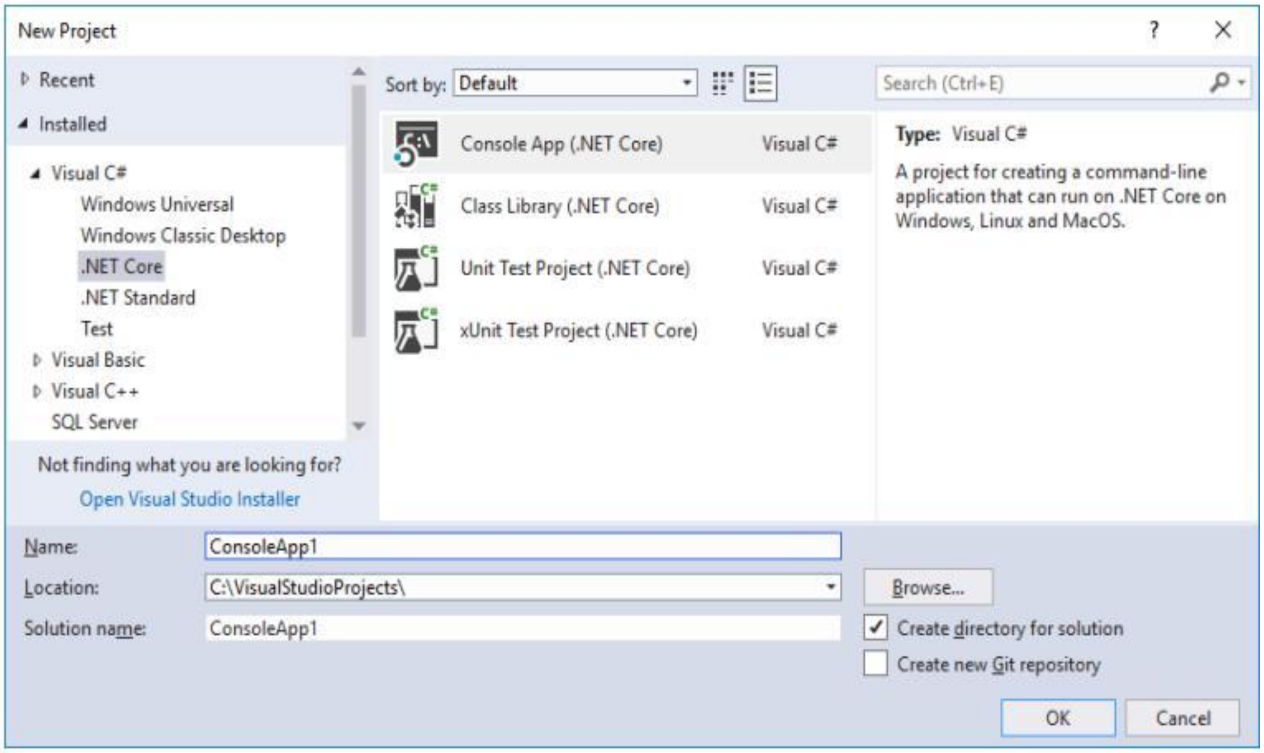
3.2 Starting a New Program in C#

You'll need the following to run C# on your computer:

- an editor or Integrated Development Environment (IDE) to write code.
- Microsoft Visual studio IDE , which can can be downloaded online from Google. Download the version 10 or above of Visual Studio express, and install it.

Creating a Project

When you select new project you are presented with a screen in Figure 1..


New Project

Recent
Installed
Visual C#
Windows Universal
Windows Classic Desktop
.NET Core
.NET Standard
Test
Visual Basic
Visual C++
SQL Server

Not finding what you are looking for?
Open Visual Studio Installer

Sort by: Default

Search (Ctrl+E)

Console App (.NET Core)	Visual C#
Class Library (.NET Core)	Visual C#
Unit Test Project (.NET Core)	Visual C#
xUnit Test Project (.NET Core)	Visual C#

Type: Visual C#
A project for creating a command-line application that can run on .NET Core on Windows, Linux and MacOS.

Name: ConsoleApp1
Location: C:\VisualStudioProjects\
Solution name: ConsoleApp1

Browse...
[✓] Create directory for solution
[ ] Create new Git repository

OK Cancel

Figure 1 New Project in VS.NET

The type of project that you will select from the pane on the left is *Visual C# Projects* (unless you don't want to create a C# project). Then you will build your project from a template by scrolling down by the right.

Working with Projects

Once you have created a C# project to work with in VS.NET you can do your regular programming tasks. This includes editing source code files, compiling, and debugging.

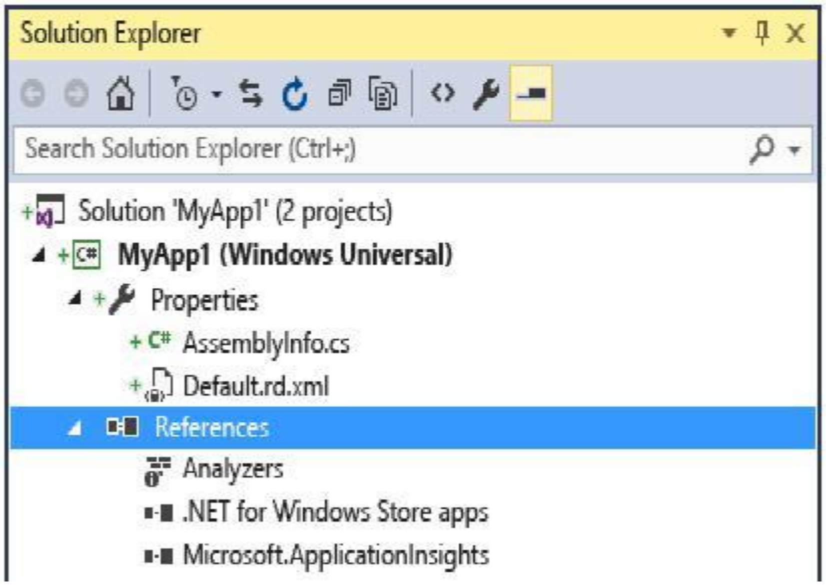

Solution Explorer
Search Solution Explorer (Ctrl+;)
Solution 'MyApp1' (2 projects)
MyApp1 (Windows Universal)
Properties
AssemblyInfo.cs
Default.rd.xml
References
Analyzers
.NET for Windows Store apps
Microsoft.ApplicationInsights

Figure2 Solution Explorer

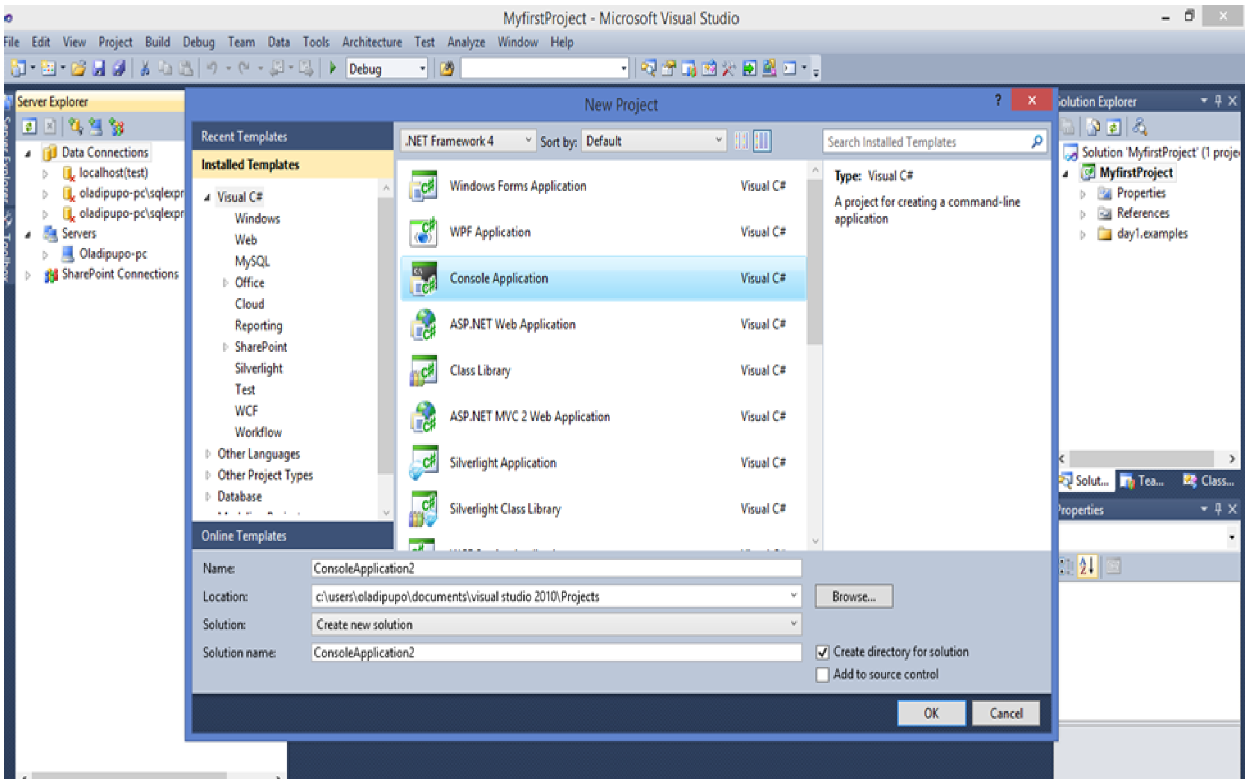
The solution explorer (see Figure 2) is your command central for dealing with solution (which is what contains one or more related projects). In Figure2 the Solution Explorer shows a solution named *LearningC#* which contains four projects.

From the Solution Explorer you can bring up source modules. Add source modules to a project. Debug a project or add and remover projects from a solution. The general rule is that you right-click your mouse on the item you want to affect, and then select the action from the menu that pops up.

3.3 Console Applications

- Console applications are those that don’t make use of the graphical windows environment. Instead, you run the application in a command prompt window, and interact with it in a much simpler way.

Figure 1 shows a screenshot of using Visual Studio to create a new project in C#.



Type: Visual C#

A project for creating a command-line application

Name: ConsoleApplication2

Location: c:\users\oladipupo\documents\visual studio 2010\Projects

Solution: Create new solution

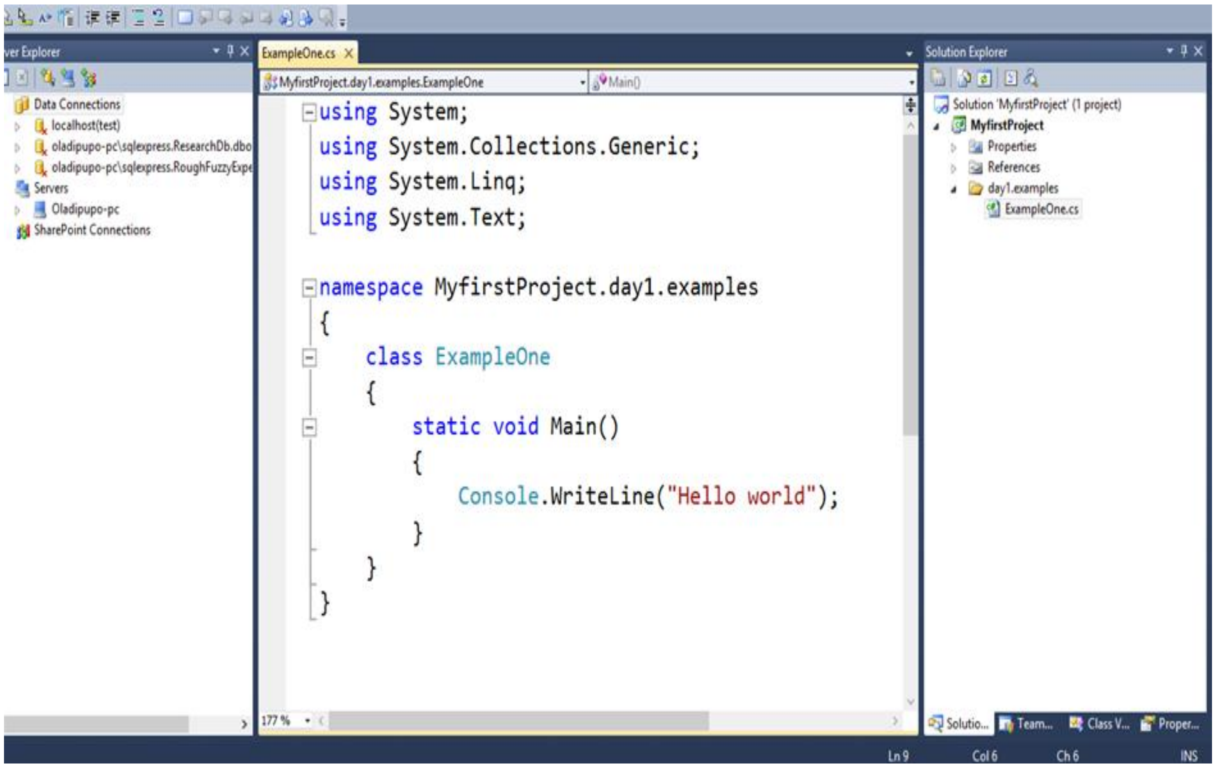
Solution name: ConsoleApplication2

[✓] Create directory for solution

[ ] Add to source control

Figure 1*: Visual Studio used to create New Project in C#*

Figure 2 shows a screenshot of Hello World program in C#.



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

namespace MyfirstProject.day1.examples
{
    class ExampleOne
    {
        static void Main()
        {
            Console.WriteLine("Hello world");
        }
    }
}
```

***Figure 2**: Hello World Program in C#*

3.4 Basic C#.Net Syntax

- C# compilers ignore additional spacing in code, known as **white space characters.**
- C# code is made up of a series of statements, each of which is terminated with **a semicolon**.
- C# is a block - structured language, meaning statements are part of a block of code.
- Use of indentation is a standard practice to clarify the sequence of code in C#.
- A very important point to note about C# code is that it is case sensitive .

  Console.WriteLine(“C# Programming!”);

- **The following codes written in C# are wrong:**

  console.WriteLine(“C# Programming!”);
  CONSOLE.WRITELINE(“C# Programming!”);

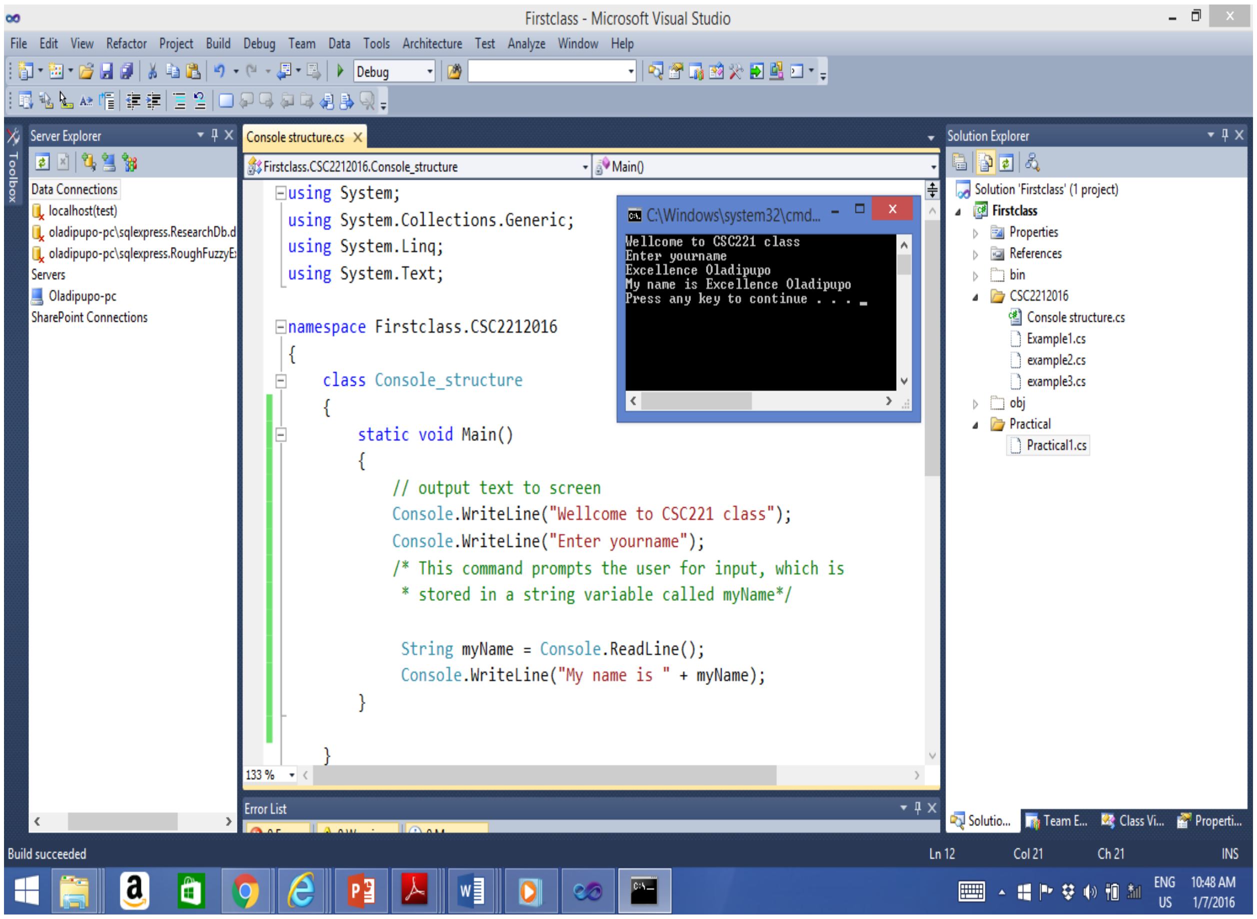
Console.Writeline(“C# Programming!”);

3.5 C#.Net Comments

- Comments are non executable statements.
- They are used for documentation.
- It can be just a single line comment
  // This is a different sort of comment.
  // Explanation of statement
- It can be multiple lines comment
  /* This is a comment
  C# programming language */

3.6 Structure of Console Application

Figure 3 shows the structure of Console Application in C#.



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

namespace Firstclass.CSC2212016
{
    class Console_structure
    {
        static void Main()
        {
            // output text to screen
            Console.WriteLine("Wellcome to CSC221 class");
            Console.WriteLine("Enter yourname");
            /* This command prompts the user for input, which is
             * stored in a string variable called myName*/

             String myName = Console.ReadLine();
             Console.WriteLine("My name is " + myName);
        }

    }
```

```
C:\Windows\system32\cmd...
Wellcome to CSC221 class
Enter yourname
Excellence Oladipupo
My name is Excellence Oladipupo
Press any key to continue . . .
```

***Figure 3**: Structure of Console application in C#*

4.0 CONCLUSION

In this unit, we start a new project and demonstrated the use of comments. We also looked at the achievable tasks in C#.Net as well as Console application development.

5.0 SUMMARY

We hope you enjoyed this unit. This unit provided an overview of C#: basic definition, features, versions and language name. Now, let us attempt the questions below.

6.0 TUTOR MARKED ASSIGNMENT

Outline at least 5 distinguishing features of C#.

7.0 REFERENCES/FURTHER READINGS

1. Archer, Tom (2001). *Inside C#*. Microsoft Press. ISBN 0-7356-1288-9.
2. Cornelius, Barry (December 1, 2005). "Java 5 catches up with C#". University of Oxford Computing Services.
3. Ecma International. June 2006.*C# Language Specification* (4th ed.).
4. Guthrie, Scott (November 28, 2006). "What language was ASP.Net originally written in?"
5. Hamilton, Naomi (October 1, 2008). "The A-Z of Programming Languages: C#". Computerworld.
6. Horton, Anson (2006-09-11). "C# XML documentation comments FAQ".
7. Kovacs, James (September 07, 2007). "C#/.NET History Lesson".
8. Microsoft. September 4, 2003."Visual C#.net Standard" (JPEG).
9. Microsoft (June 18, 2009) *C# Language Reference*.
10. O' Reilly. (2002).*C# Language Pocket Reference*. ISBN 0-596-00429-X.
11. Petzold, Charles (2002). *Programming Microsoft Windows with C#*. Microsoft Press. ISBN 0-7356-1370-2.
12. Steinman, Justin (November 7, 2006). "Novell Answers Questions from the Community".
13. Stallman, Richard (June 26, 2009). "Why free software shouldn't depend on Mono or C#".
14. Simon, Raphael; Stapf, Emmanuel; Meyer, Bertrand (June 2002). "Full Eiffel on the .NET Framework".
15. "The ECMA C# and CLI Standards". 2009 -07-06.
16. Zander, Jason (November 23, 2007). "Couple of Historical Facts".

UNIT 2: VARIABLES, DATA TYPES AND EXPRESSIONS

1.0 Introduction

The initial task we have in this unit is to known the need for variable declaration, data types and expression in C#. and use that to create a new project in Console application environment.

2.0 Objectives

• At the end of this unit, you should be able to:
• Define and declare variable and use different data type
• Start a new project in C#.Net

3.0 Main Content

3.1 Variables and Expressions

- Variables are concerned with the storage of data.
- Variables come in different flavours, known as **types**
- To use variables, you have to declare them.
- This means that you have to assign them a *name* and a *type* .
- Once you have declared variables, you can use them as storage units for the types of data that you declared them to hold.

3.2 Variable Naming

- The first character of a variable name must be either a letter, an underscore character (_), or the *at* symbol (@).
- Subsequent characters may be letters, underscore characters, or numbers.
- A name must not contain spaces.
- Any of the keyword in C# language must not be used as a variable name.

3.3 Variable Declaration

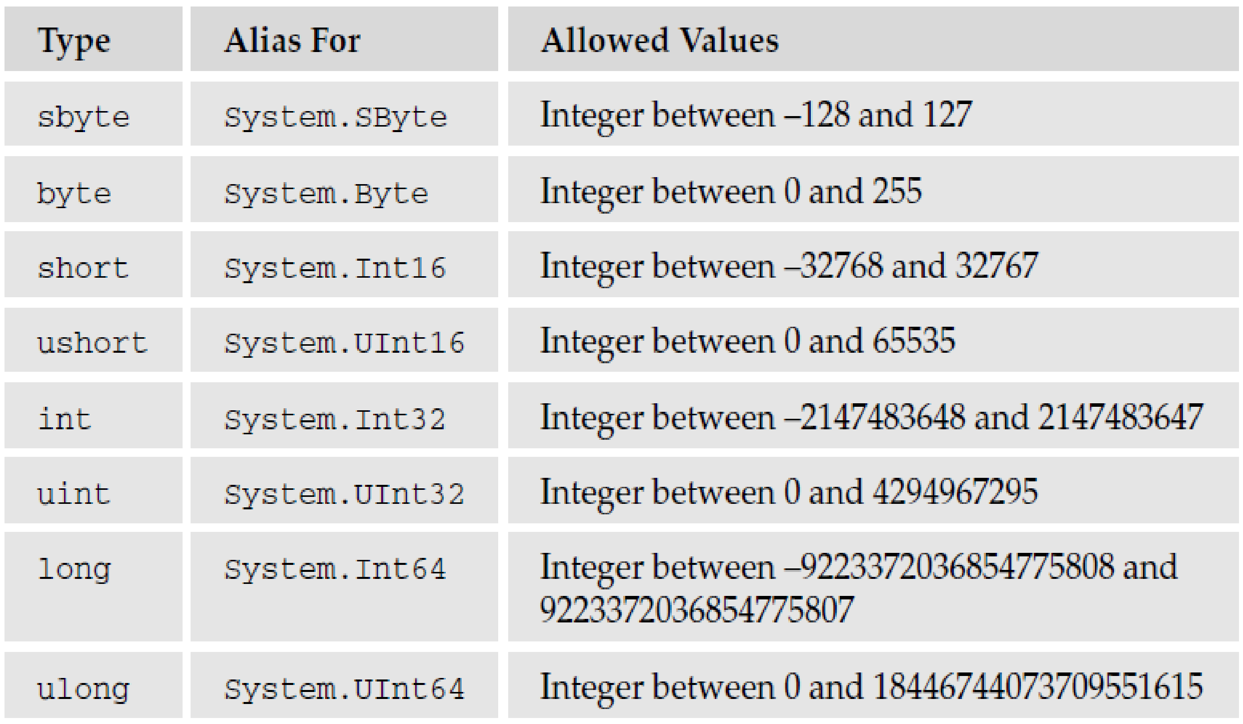
C# syntax for declaring variables merely specifies the type and variable name:

```
            < type > < name > ;
int a;      // to hold whole number
double  variableName;
double  VariableName;
float sum;
string  firstname
```

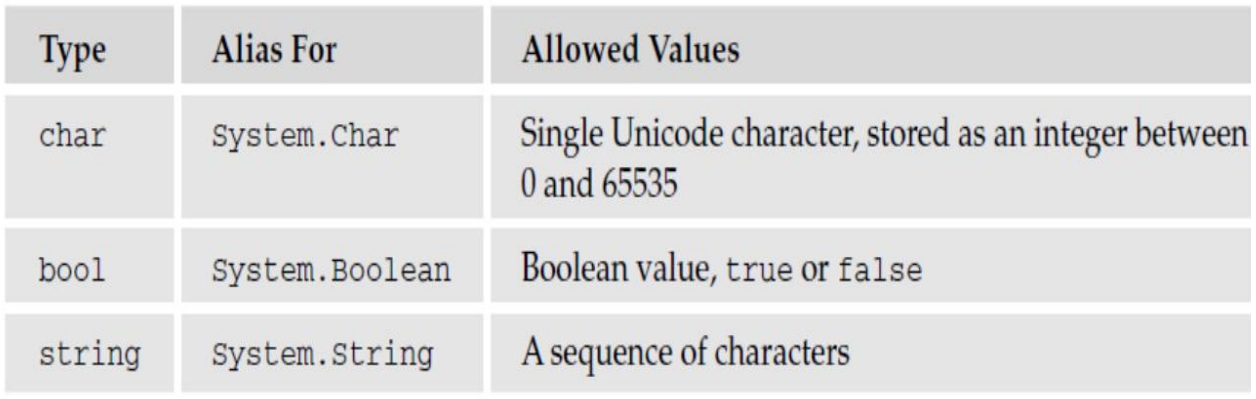
3.4 Simple Data Types and Allowed Values

Table 1 highlights a simple data types and their allowed values.

***Table 1**: Simple Data Types and their allowed values*



Type	Alias For	Allowed Values
`sbyte`	`System.SByte`	Integer between –128 and 127
`byte`	`System.Byte`	Integer between 0 and 255
`short`	`System.Int16`	Integer between –32768 and 32767
`ushort`	`System.UInt16`	Integer between 0 and 65535
`int`	`System.Int32`	Integer between –2147483648 and 2147483647
`uint`	`System.UInt32`	Integer between 0 and 4294967295
`long`	`System.Int64`	Integer between –9223372036854775808 and 9223372036854775807
`ulong`	`System.UInt64`	Integer between 0 and 18446744073709551615



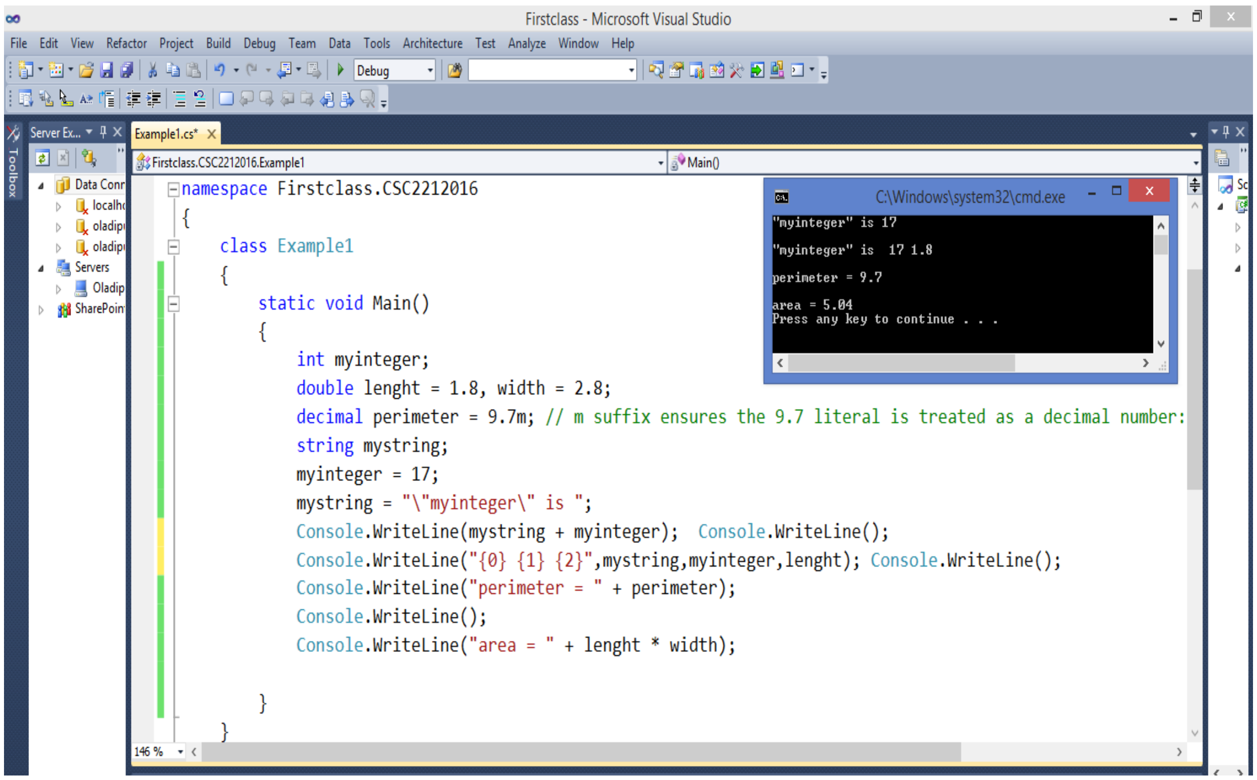
Type	Alias For	Allowed Values
`char`	`System.Char`	Single Unicode character, stored as an integer between 0 and 65535
`bool`	`System.Boolean`	Boolean value, `true` or `false`
`string`	`System.String`	A sequence of characters

3.5 Variable Declaration and Assignment

- Recall that you declare variables simply using their **type and name**:
  int age;
- You then assign values to variables using the "=' assignment operator:

```
age = 25;
int age, height;                    //Here, age and height are both declared as
                                       integer types.
int age = 25
double  length = 4.5, weight = 2.5;
decimal breadth = 7.58m;                    // declared as literal value of decimal
string myString;
myString = “ hello world”
```

Figure 4 shows a variable declaration sheet and assignments in C#.


Firstclass - Microsoft Visual Studio
File Edit View Refactor Project Build Debug Team Data Tools Architecture Test Analyze Window Help
Debug
Server Ex...
Example1.cs*
Firstclass.CSC2212016.Example1
Main()
C:\Windows\system32\cmd.exe
146 %

```
namespace Firstclass.CSC2212016
{
    class Example1
    {
        static void Main()
        {
            int myinteger;
            double lenght = 1.8, width = 2.8;
            decimal perimeter = 9.7m; // m suffix ensures the 9.7 literal is treated as a decimal number:
            string mystring;
            myinteger = 17;
            mystring = "\"myinteger\" is ";
            Console.WriteLine(mystring + myinteger);  Console.WriteLine();
            Console.WriteLine("{0} {1} {2}",mystring,myinteger,lenght); Console.WriteLine();
            Console.WriteLine("perimeter = " + perimeter);
            Console.WriteLine();
            Console.WriteLine("area = " + lenght * width);

        }
    }
```

```
"myinteger" is 17

"myinteger" is  17 1.8

perimeter = 9.7

area = 5.04
Press any key to continue . . .
```

***Figure 4**: Variable declaration and assignments in C#*

3.6 Expression

Now that you’ve learned how to declare and initialize variables, it’s time to look at manipulating them. C# contains a number of *operators* for this purpose, including the = assignment operator.

By combining operators with variables and literal values (together referred to as *operands* when used with operators), you can create *expressions*, which are the basic building blocks of computation.

4.0 CONCLUSION

In this unit, we defined some basic concepts of C#. We also looked at the advancements in C# as well as the language name.

5.0 SUMMARY

We hope you enjoyed this unit. This unit provided an overview of C#: basic definition, features, versions and language name. Now, let us attempt the questions below.

6.0 TUTOR MARKED ASSIGNMENT

Outline at least 5 distinguishing features of C#.

7.0 REFERENCES/FURTHER READINGS

1. Archer, Tom (2001). *Inside C#*. Microsoft Press. ISBN 0-7356-1288-9.
2. Cornelius, Barry (December 1, 2005). "Java 5 catches up with C#". University of Oxford Computing Services.
3. Ecma International. June 2006.*C# Language Specification* (4th ed.).
4. Guthrie, Scott (November 28, 2006). "What language was ASP.Net originally written in?"
5. Hamilton, Naomi (October 1, 2008). "The A-Z of Programming Languages: C#". Computerworld.
6. Horton, Anson (2006-09-11). "C# XML documentation comments FAQ".
7. Kovacs, James (September 07, 2007). "C#/.NET History Lesson".
8. Microsoft. September 4, 2003."Visual C#.net Standard" (JPEG).
9. Microsoft (June 18, 2009) *C# Language Reference*.
10. O' Reilly. (2002).*C# Language Pocket Reference*. ISBN 0-596-00429-X.
11. Petzold, Charles (2002). *Programming Microsoft Windows with C#*. Microsoft Press. ISBN 0-7356-1370-2.
12. Steinman, Justin (November 7, 2006). "Novell Answers Questions from the Community".
13. Stallman, Richard (June 26, 2009). "Why free software shouldn't depend on Mono or C#".

14. Simon, Raphael; Stapf, Emmanuel; Meyer, Bertrand (June 2002). "Full Eiffel on the .NET Framework".
15. "The ECMA C# and CLI Standards". 2009 -07-06.
16. Zander, Jason (November 23, 2007). "Couple of Historical Facts".

UNIT 3: OPERATORS IN C#.Net

CONTENT

1.0 Introduction

This unit introduces students to various operators in C#

2.0 Objectives

• At the end of this unit, you should be able to:
• Differentiate the categories of operators in C#

3.0 Main Content

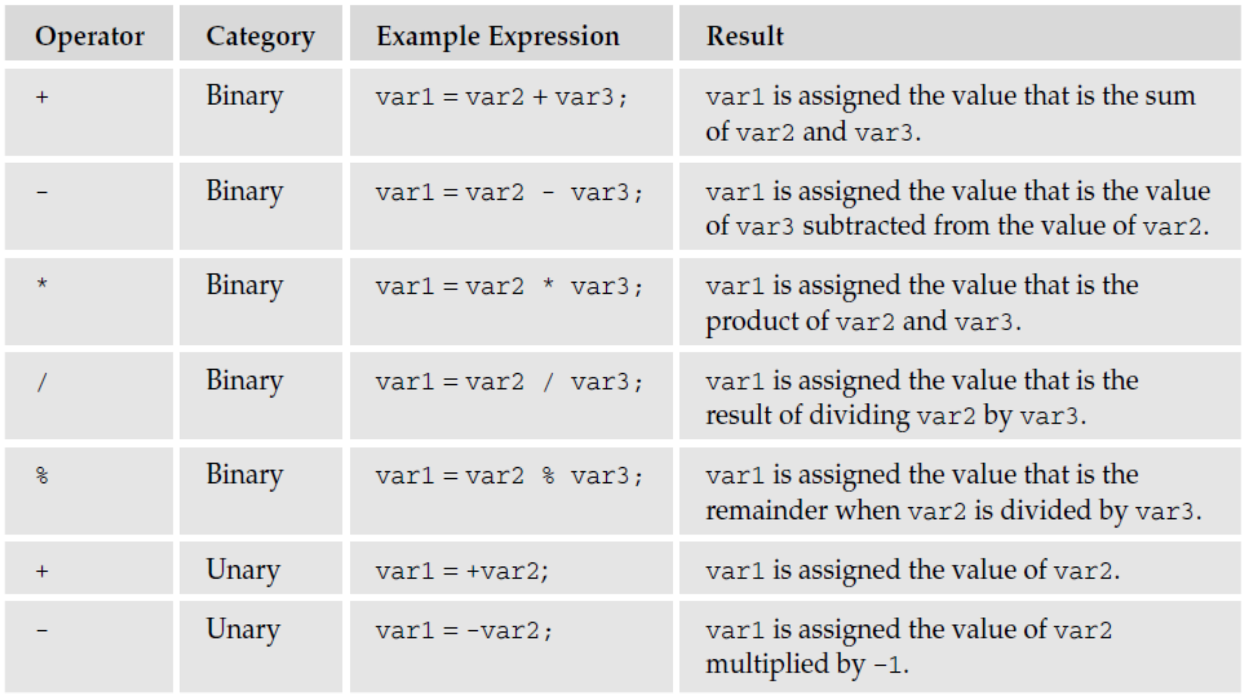
3.1 Operators Categories

Operators can be roughly classified into three categories:
Unary: Act on single operands
Binary: Act on two operands
Ternary: Act on three operands

3.2 Mathematical Operators

Table 2 contains mathematical operators in C#.

Table 2: *Mathematical Operators in C#*

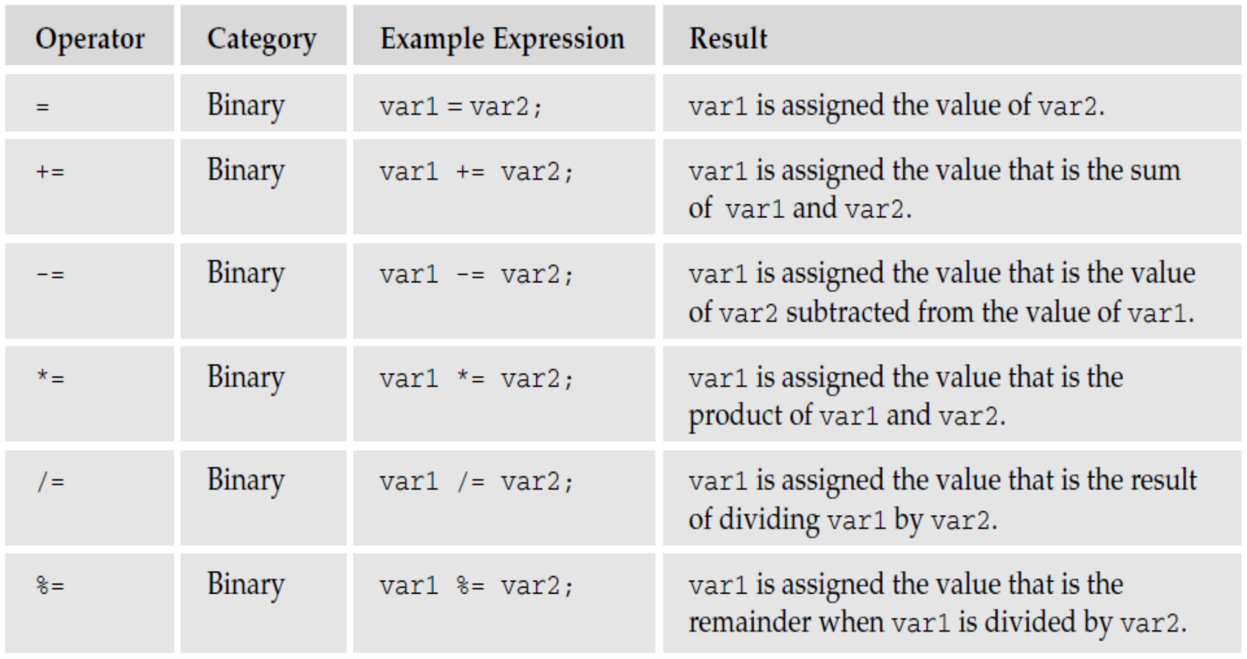


Operator	Category	Example Expression	Result
+	Binary	`var1 = var2 + var3;`	`var1` is assigned the value that is the sum of `var2` and `var3`.
-	Binary	`var1 = var2 - var3;`	`var1` is assigned the value that is the value of `var3` subtracted from the value of `var2`.
*	Binary	`var1 = var2 * var3;`	`var1` is assigned the value that is the product of `var2` and `var3`.
/	Binary	`var1 = var2 / var3;`	`var1` is assigned the value that is the result of dividing `var2` by `var3`.
%	Binary	`var1 = var2 % var3;`	`var1` is assigned the value that is the remainder when `var2` is divided by `var3`.
+	Unary	`var1 = +var2;`	`var1` is assigned the value of `var2`.
-	Unary	`var1 = -var2;`	`var1` is assigned the value of `var2` multiplied by `-1`.

Table 3 describes the assignment operators in C#.

3.3 Assignment Operators

Table 3: *Assignment Operators in C#*

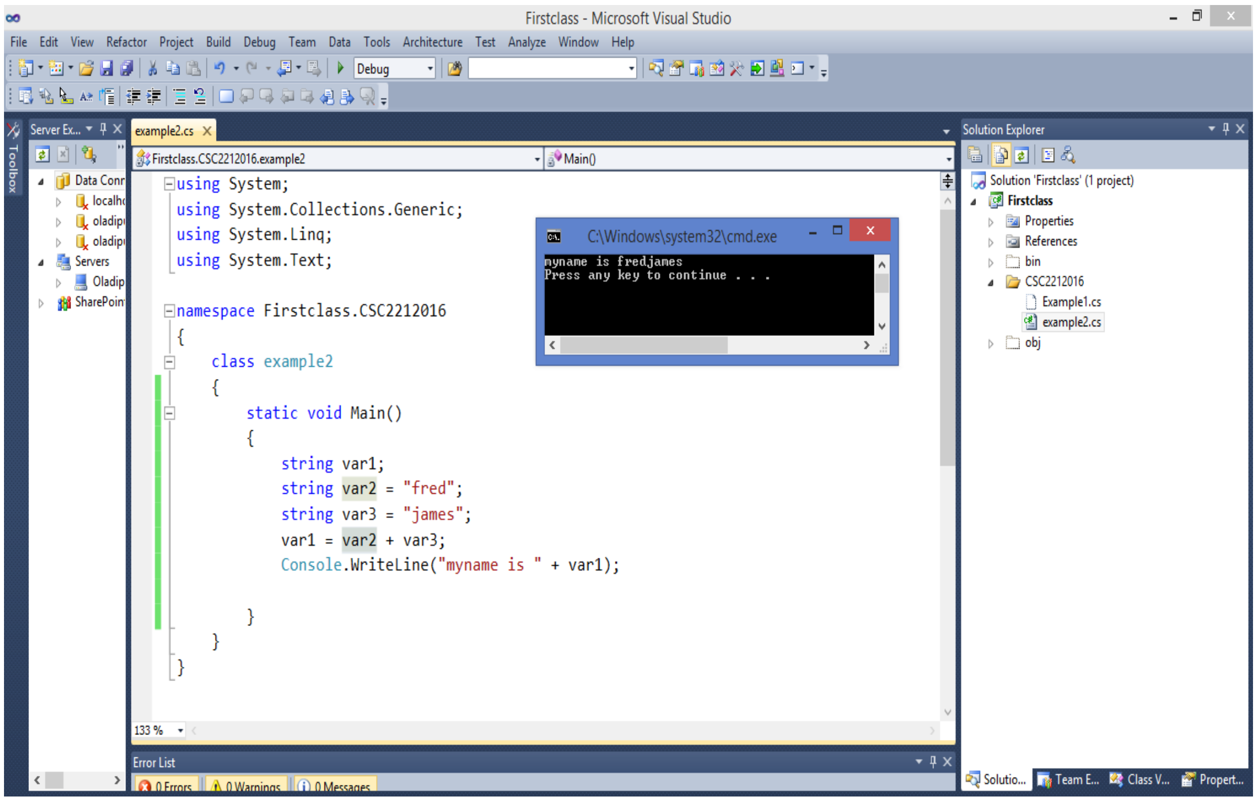


Operator	Category	Example Expression	Result
`=`	Binary	`var1 = var2;`	`var1` is assigned the value of `var2`.
`+=`	Binary	`var1 += var2;`	`var1` is assigned the value that is the sum of `var1` and `var2`.
`-=`	Binary	`var1 -= var2;`	`var1` is assigned the value that is the value of `var2` subtracted from the value of `var1`.
`*=`	Binary	`var1 *= var2;`	`var1` is assigned the value that is the product of `var1` and `var2`.
`/=`	Binary	`var1 /= var2;`	`var1` is assigned the value that is the result of dividing `var1` by `var2`.
`%=`	Binary	`var1 %= var2;`	`var1` is assigned the value that is the remainder when `var1` is divided by `var2`.

3.4 Binary Operator for string concatenation

- The binary + operator *does* make sense when used with string type variables.
  string var1,var2,var3;
  var1 = var2 + var3; /* string var2 and var3 are concatenated

Figure 5 shows strings concatenation in C#.


Firstclass - Microsoft Visual Studio
File Edit View Refactor Project Build Debug Team Data Tools Architecture Test Analyze Window Help
example2.cs
Firstclass.CSC2212016.example2
Main()
Solution Explorer
Solution 'Firstclass' (1 project)
Firstclass
Properties
References
bin
CSC2212016
Example1.cs
example2.cs
obj
Error List
0 Errors
0 Warnings
0 Messages
C:\Windows\system32\cmd.exe

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

namespace Firstclass.CSC2212016
{
    class example2
    {
        static void Main()
        {
            string var1;
            string var2 = "fred";
            string var3 = "james";
            var1 = var2 + var3;
            Console.WriteLine("myname is " + var1);

        }
    }
}
```

```
myname is fredjames
Press any key to continue . . .
```

***Figure 5**: Strings Concatenation in C#*

3.5 Incremental Operators

int var1 = ++ var1;

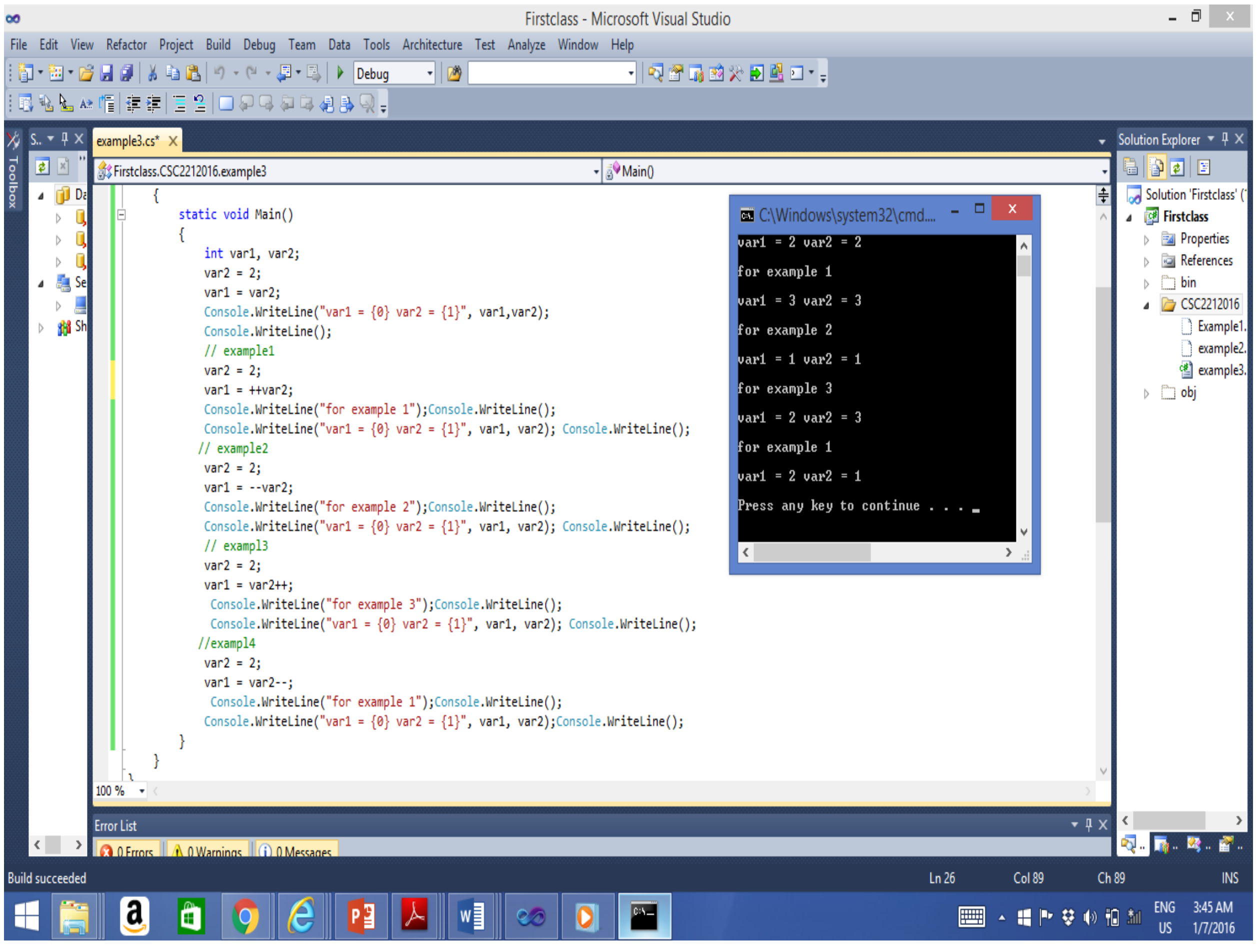
int var1 = var1+1;

Table 4 enumerates incremental operators in C#.

***Table 4**: Incremental Operators in C#*

Operator	Category	Example Expression	Result
`++`	Unary	`var1 = ++var2;`	`var1` is assigned the value of `var2 + 1`. `var2` is incremented by `1`.
`--`	Unary	`var1 = --var2;`	`var1` is assigned the value of `var2 - 1`. `var2` is decremented by `1`.
`++`	Unary	`var1 = var2++;`	`var1` is assigned the value of `var2`. `var2` is incremented by `1`.
`--`	Unary	`var1 = var2--;`	`var1` is assigned the value of `var2`. `var2` is decremented by `1`.

Figure 6 shows a screenshot of using operators in C#.

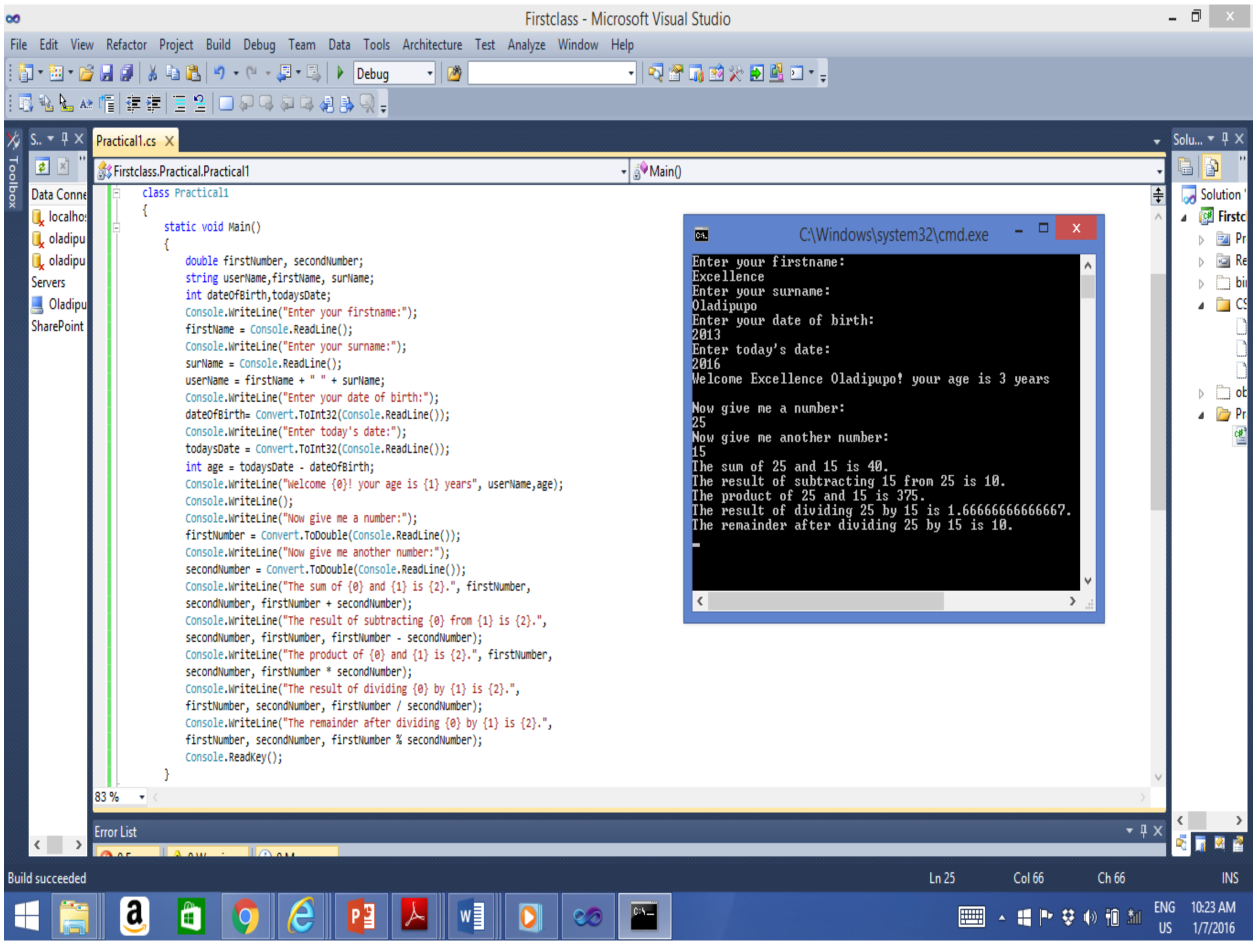


```
    {
        static void Main()
        {
            int var1, var2;
            var2 = 2;
            var1 = var2;
            Console.WriteLine("var1 = {0} var2 = {1}", var1,var2);
            Console.WriteLine();
            // example1
            var2 = 2;
            var1 = ++var2;
            Console.WriteLine("for example 1");Console.WriteLine();
            Console.WriteLine("var1 = {0} var2 = {1}", var1, var2); Console.WriteLine();
           // example2
            var2 = 2;
            var1 = --var2;
            Console.WriteLine("for example 2");Console.WriteLine();
            Console.WriteLine("var1 = {0} var2 = {1}", var1, var2); Console.WriteLine();
            // exampl3
            var2 = 2;
            var1 = var2++;
             Console.WriteLine("for example 3");Console.WriteLine();
             Console.WriteLine("var1 = {0} var2 = {1}", var1, var2); Console.WriteLine();
           //exampl4
            var2 = 2;
            var1 = var2--;
             Console.WriteLine("for example 1");Console.WriteLine();
            Console.WriteLine("var1 = {0} var2 = {1}", var1, var2);Console.WriteLine();
        }
    }
```

```
var1 = 2 var2 = 2

for example 1

var1 = 3 var2 = 3

for example 2

var1 = 1 var2 = 1

for example 3

var1 = 2 var2 = 3

for example 1

var1 = 2 var2 = 1

Press any key to continue . . .
```

***Figure 6**: Using Operators in C#*

Figure 7 shows a screenshot of operators and concatenation in C#.


Firstclass - Microsoft Visual Studio
File Edit View Refactor Project Build Debug Team Data Tools Architecture Test Analyze Window Help
Debug
Practical1.cs
Firstclass.Practical.Practical1
Main()
C:\Windows\system32\cmd.exe
83 %
Error List
Build succeeded
Ln 25 Col 66 Ch 66 INS
ENG US 10:23 AM 1/7/2016

```csharp
class Practical1
{
    static void Main()
    {
        double firstNumber, secondNumber;
        string userName,firstName, surName;
        int dateOfBirth,todaysDate;
        Console.WriteLine("Enter your firstname:");
        firstName = Console.ReadLine();
        Console.WriteLine("Enter your surname:");
        surName = Console.ReadLine();
        userName = firstName + " " + surName;
        Console.WriteLine("Enter your date of birth:");
        dateOfBirth= Convert.ToInt32(Console.ReadLine());
        Console.WriteLine("Enter today's date:");
        todaysDate = Convert.ToInt32(Console.ReadLine());
        int age = todaysDate - dateOfBirth;
        Console.WriteLine("Welcome {0}! your age is {1} years", userName,age);
        Console.WriteLine();
        Console.WriteLine("Now give me a number:");
        firstNumber = Convert.ToDouble(Console.ReadLine());
        Console.WriteLine("Now give me another number:");
        secondNumber = Convert.ToDouble(Console.ReadLine());
        Console.WriteLine("The sum of {0} and {1} is {2}.", firstNumber,
        secondNumber, firstNumber + secondNumber);
        Console.WriteLine("The result of subtracting {0} from {1} is {2}.",
        secondNumber, firstNumber, firstNumber - secondNumber);
        Console.WriteLine("The product of {0} and {1} is {2}.", firstNumber,
        secondNumber, firstNumber * secondNumber);
        Console.WriteLine("The result of dividing {0} by {1} is {2}.",
        firstNumber, secondNumber, firstNumber / secondNumber);
        Console.WriteLine("The remainder after dividing {0} by {1} is {2}.",
        firstNumber, secondNumber, firstNumber % secondNumber);
        Console.ReadKey();
    }
```

```
Enter your firstname:
Excellence
Enter your surname:
Oladipupo
Enter your date of birth:
2013
Enter today's date:
2016
Welcome Excellence Oladipupo! your age is 3 years

Now give me a number:
25
Now give me another number:
15
The sum of 25 and 15 is 40.
The result of subtracting 15 from 25 is 10.
The product of 25 and 15 is 375.
The result of dividing 25 by 15 is 1.66666666666667.
The remainder after dividing 25 by 15 is 10.
```

***Figure 7**: Operators and Strings Concatenation in C#*

3.6 Assessment Exercises

- Create a new console application called **Practical1** and save in a folder called **Practical** under the default VS directory that perform the following operations:
- Ask for your firstname and surname
- Concatenate the two names for username
- Ask for your year of birth and the present year to determine your age.
- Print a welcome message with the username and age.
- Prompt to enter two numbers successively to calculate
    - (i) X+Y, (ii) X-Y, (iii) X/Y and (iv) X%Y
- Print out the output for each expression on different lines
  E.g " sum of X and Y is ---"

Note: Even if you don't have a laptop, write down the solution
Solution Screenshot

4.0 CONCLUSION

In this unit, we defined some basic concepts of C#. We also looked at the advancements in C# as well as the language name.

5.0 SUMMARY

We hope you enjoyed this unit. This unit provided an overview of C#: basic definition, features, versions and language name. Now, let us attempt the questions below.

6.0 TUTOR MARKED ASSIGNMENT

Outline the three categories of C# Operators.

UNIT QUIZ

Which of the following is not a legal variable name?

❑ myVariableIsGood
❑ 99Flake
❑ floor
❑ time2GetJiggyWidIt
❑ wrox.com

Write a console application that obtains four int values from the user and displays the product.

7.0 REFERENCES/FURTHER READINGS

1. Archer, Tom (2001). *Inside C#*. Microsoft Press. ISBN 0-7356-1288-9.
2. Cornelius, Barry (December 1, 2005). "Java 5 catches up with C#". University of Oxford Computing Services.
3. Ecma International. June 2006.*C# Language Specification* (4th ed.).
4. Guthrie, Scott (November 28, 2006). "What language was ASP.Net originally written in?"
5. Hamilton, Naomi (October 1, 2008). "The A-Z of Programming Languages: C#".

Computerworld.
6. Horton, Anson (2006-09-11). "C# XML documentation comments FAQ".
7. Kovacs, James (September 07, 2007). "C#/.NET History Lesson".
8. Microsoft. September 4, 2003."Visual C#.net Standard" (JPEG).
9. Microsoft (June 18, 2009) *C# Language Reference*.
10. O' Reilly. (2002).*C# Language Pocket Reference*. ISBN 0-596-00429-X.
11. Petzold, Charles (2002). *Programming Microsoft Windows with C#*. Microsoft Press. ISBN 0-7356-1370-2.
12. Steinman, Justin (November 7, 2006). "Novell Answers Questions from the Community".
13. Stallman, Richard (June 26, 2009). "Why free software shouldn't depend on Mono or C#".
14. Simon, Raphael; Stapf, Emmanuel; Meyer, Bertrand (June 2002). "Full Eiffel on the .NET Framework".
15. "The ECMA C# and CLI Standards". 2009 -07-06.
16. Zander, Jason (November 23, 2007). "Couple of Historical Facts".

MODULE 3 FLOW CONTROL AND TYPE CONVERSION IN C#.Net

UNIT 1: Flow Control

1.0 Introduction

The initial task we have in this unit is to know the various flow controls in C#.

2.0 Objectives

At the end of this unit, you should be able to:
• Use Goto statement n a program
• Use If and Switch statement in C#.Net program

3.0 Main Content

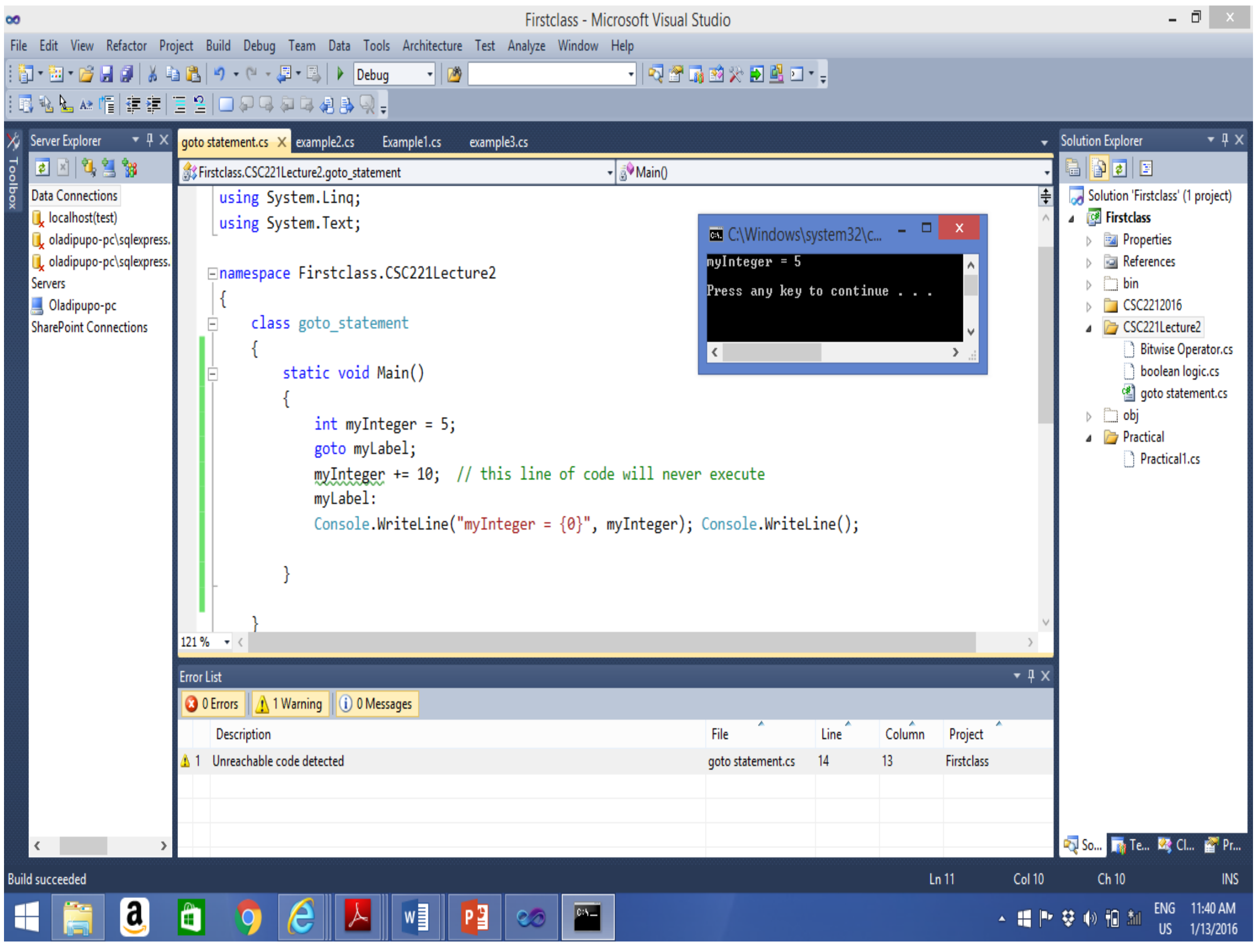
3.1 The goto Statement

- C# enables you to label lines of code and then jump straight to them using the ***goto statement***.
- The goto statement is used as follows:
  goto < *labelName* > ;
- Labels are defined as follows:
  < *labelName* > :

Program Sample

```
int myInteger = 5;
goto myLabel;
myInteger += 10;      // this line of code will never be executed
myLabel:
Console.WriteLine("myInteger = {0}", myInteger);
```

Figure 8 shows a screen shoot of using GOTO statement in C#.



```
using System.Linq;
using System.Text;

namespace Firstclass.CSC221Lecture2
{
    class goto_statement
    {
        static void Main()
        {
            int myInteger = 5;
            goto myLabel;
            myInteger += 10;  // this line of code will never execute
            myLabel:
            Console.WriteLine("myInteger = {0}", myInteger); Console.WriteLine();

        }
    }
```

```
myInteger = 5
Press any key to continue . . .
```

Error List

0 Errors | 1 Warning | 0 Messages

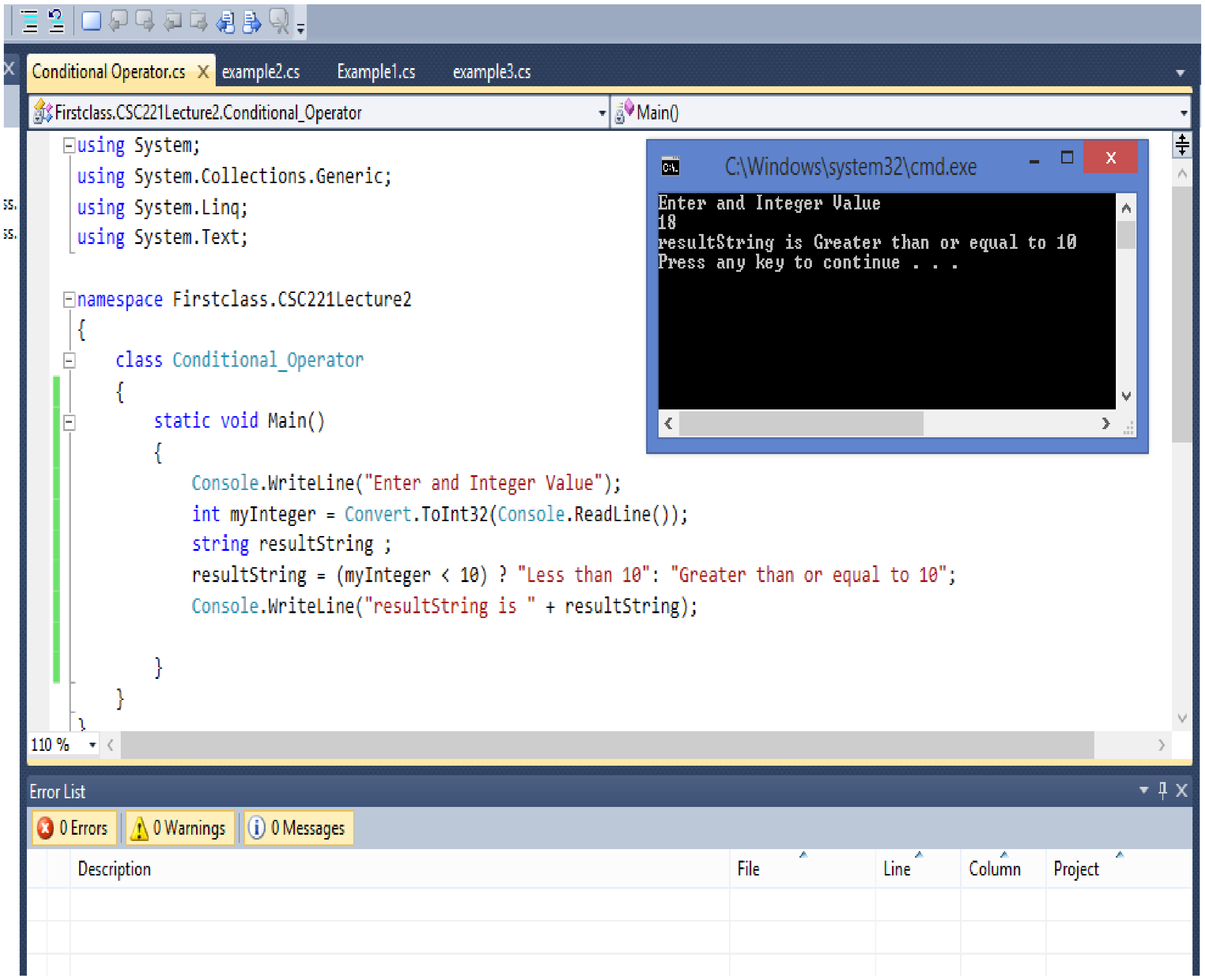
	Description	File	Line	Column	Project
1	Unreachable code detected	goto statement.cs	14	13	Firstclass

***Figure 8**: Using GOTO statement in C#*

3.2 Branching:

The ternary or conditional operator
The if statement
The switch statement

Figure 9 shows a screen shot of ternary operators and assignments in C#.



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

namespace Firstclass.CSC221Lecture2
{
    class Conditional_Operator
    {
        static void Main()
        {
            Console.WriteLine("Enter and Integer Value");
            int myInteger = Convert.ToInt32(Console.ReadLine());
            string resultString ;
            resultString = (myInteger < 10) ? "Less than 10": "Greater than or equal to 10";
            Console.WriteLine("resultString is " + resultString);

        }
    }
}
```

```
Enter and Integer Value
18
resultString is Greater than or equal to 10
Press any key to continue . . .
```

Figure 9: Ternary Operators and assignment in C#

3.3 The If statement

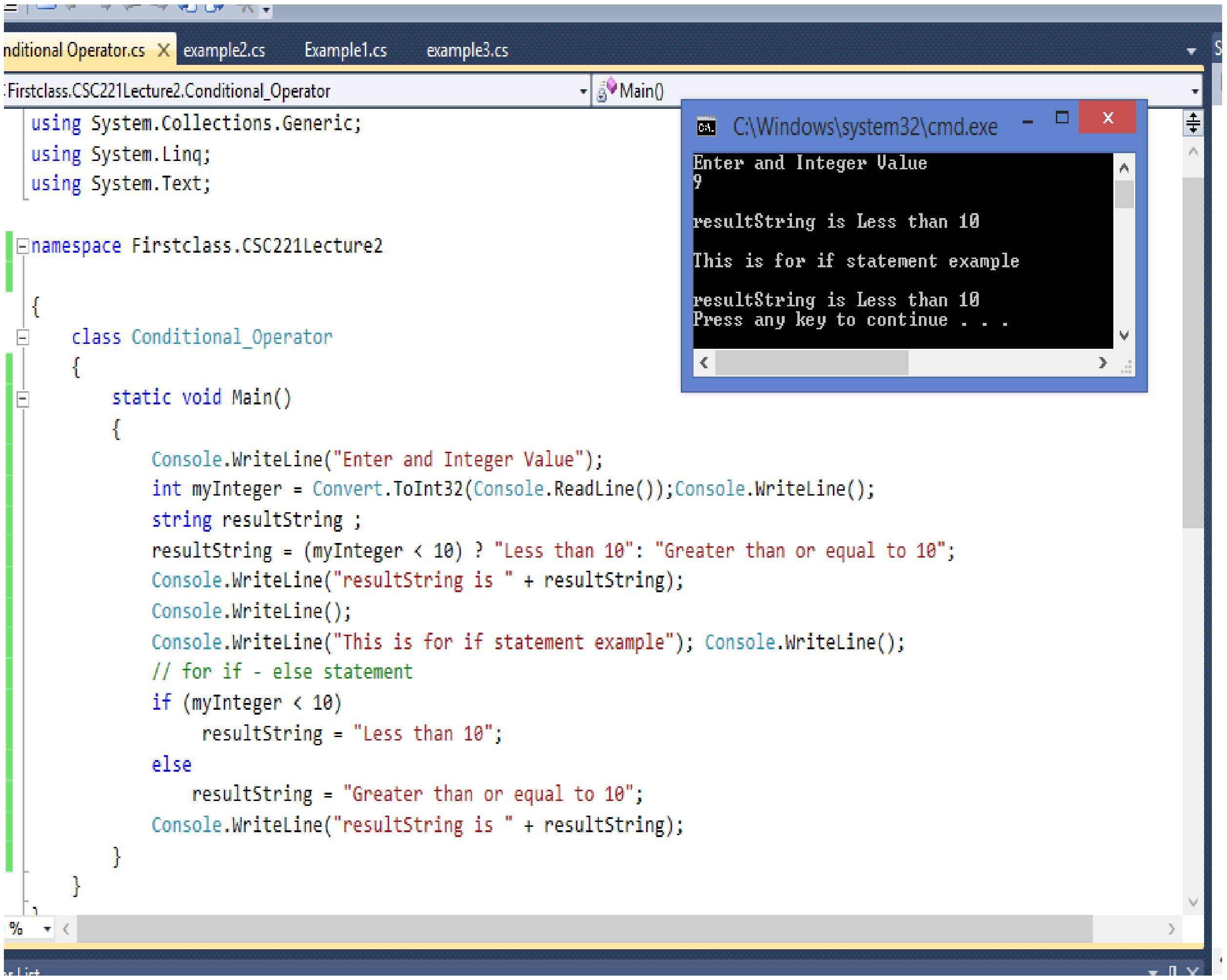
- if <test> is true, then *statement1* is executed and continue with other statements>
- if <test> is false, it will jump *statement1(not* executed) and continue with other statements>

```
if ( < test > )
        < statement1> ;
        <statement2>;
        <statement3>;
```

3.4 The If-else statement

```
if ( < test > )
                < code executed if < test > is true > ;
else
                < code executed if < test > is false > ;
if ( < test > )
                < code executed if < test > is true > ;
else
                < code executed if < test > is false > ;
```

Figure 10 shows a screenshot of if-else statement in C#


nditional Operator.cs
example2.cs
Example1.cs
example3.cs
Firstclass.CSC221Lecture2.Conditional_Operator
Main()
C:\Windows\system32\cmd.exe

```
Enter and Integer Value
9

resultString is Less than 10

This is for if statement example

resultString is Less than 10
Press any key to continue . . .
```

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

namespace Firstclass.CSC221Lecture2

{
    class Conditional_Operator
    {
        static void Main()
        {
            Console.WriteLine("Enter and Integer Value");
            int myInteger = Convert.ToInt32(Console.ReadLine());Console.WriteLine();
            string resultString ;
            resultString = (myInteger < 10) ? "Less than 10": "Greater than or equal to 10";
            Console.WriteLine("resultString is " + resultString);
            Console.WriteLine();
            Console.WriteLine("This is for if statement example"); Console.WriteLine();
            // for if - else statement
            if (myInteger < 10)
                resultString = "Less than 10";
            else
               resultString = "Greater than or equal to 10";
            Console.WriteLine("resultString is " + resultString);
        }
    }
```

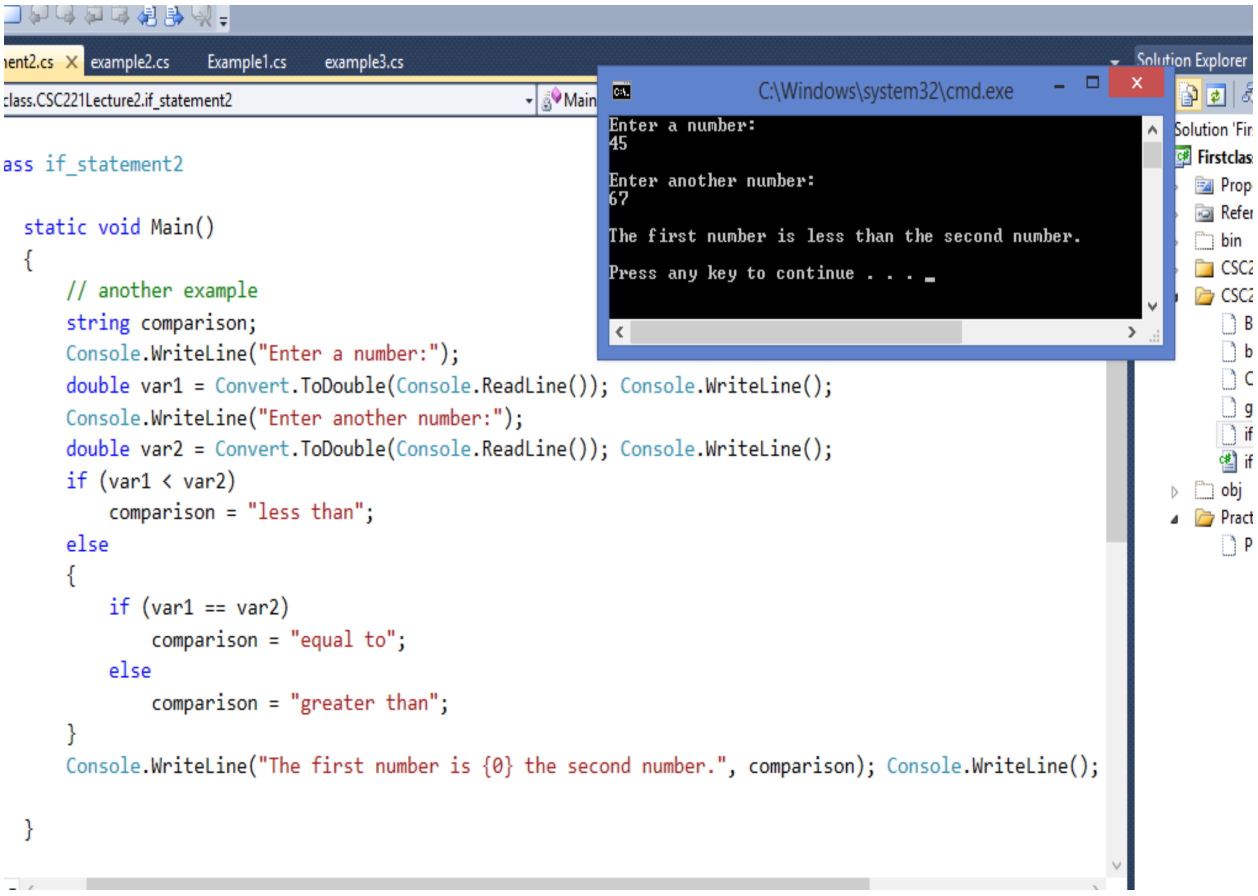
***Figure 10**: If-else statement in C#*

```
 if (var1 < var2)
          comparison = "less than";
else
      {
          if (var1 == var2)
             comparison = "equal to";
          else
             comparison = "greater than";
       }
```

- The nesting used here is just one method. You could equally have written

```
        this:
if (var1 < var2)
        comparison = “less than”;
if (var1 == var2)
        comparison = “equal to”;
if (var1 > var2)
        comparison = “greater than”;
```

Figure 11 shows a screenshot of if-else if in C#.



```
ass if_statement2

  static void Main()
  {
      // another example
      string comparison;
      Console.WriteLine("Enter a number:");
      double var1 = Convert.ToDouble(Console.ReadLine()); Console.WriteLine();
      Console.WriteLine("Enter another number:");
      double var2 = Convert.ToDouble(Console.ReadLine()); Console.WriteLine();
      if (var1 < var2)
          comparison = "less than";
      else
      {
          if (var1 == var2)
              comparison = "equal to";
          else
              comparison = "greater than";
      }
      Console.WriteLine("The first number is {0} the second number.", comparison); Console.WriteLine();

  }
```

```
Enter a number:
45

Enter another number:
67

The first number is less than the second number.

Press any key to continue . . .
```

***Figure 11**: If - else - if in C#*

3.5 The switch Statement

The switch statement is similar to the *if statement* in that it executes code conditionally based on the value of a test.
However, switch enables you to test for multiple values of a test variable in one go, rather than just a single condition.
This test is limited to discrete values, rather than clauses such as " greater than X, " so its use is slightly different.

But it can be a powerful technique.

```
switch ( < testVar > )
{
case < comparisonVal1 > :
    < code to execute if < testVar > == < comparisonVal1 > >
    break;
case < comparisonVal2 > :
    < code to execute if < testVar > == < comparisonVal2 > >
    break;
    ...
case < comparisonValN > :
    < code to execute if < testVar > == < comparisonValN > >
    break;
default:
    < code to execute if < testVar > != comparisonVals >
    break;
```

Explanation

- The value in < testVar > is compared to each of the < comparisonValX > values (specified with case statements), and if there is a match, then the code supplied for this match is executed.
- If there is no match, then the code in the default section is executed if this block exists.
- On completion of the code in each section, you have an additional command, break .
- With the break command, it is illegal for the flow of execution to reach a

second case statement after processing one case block.

- *This behaviour is one area in which C# differs from C++, where the processing of* case *statements is allowed to run from one to another.*

Class Exercise

- Write a program to determine the equipment for different weather conditions based on the table below:

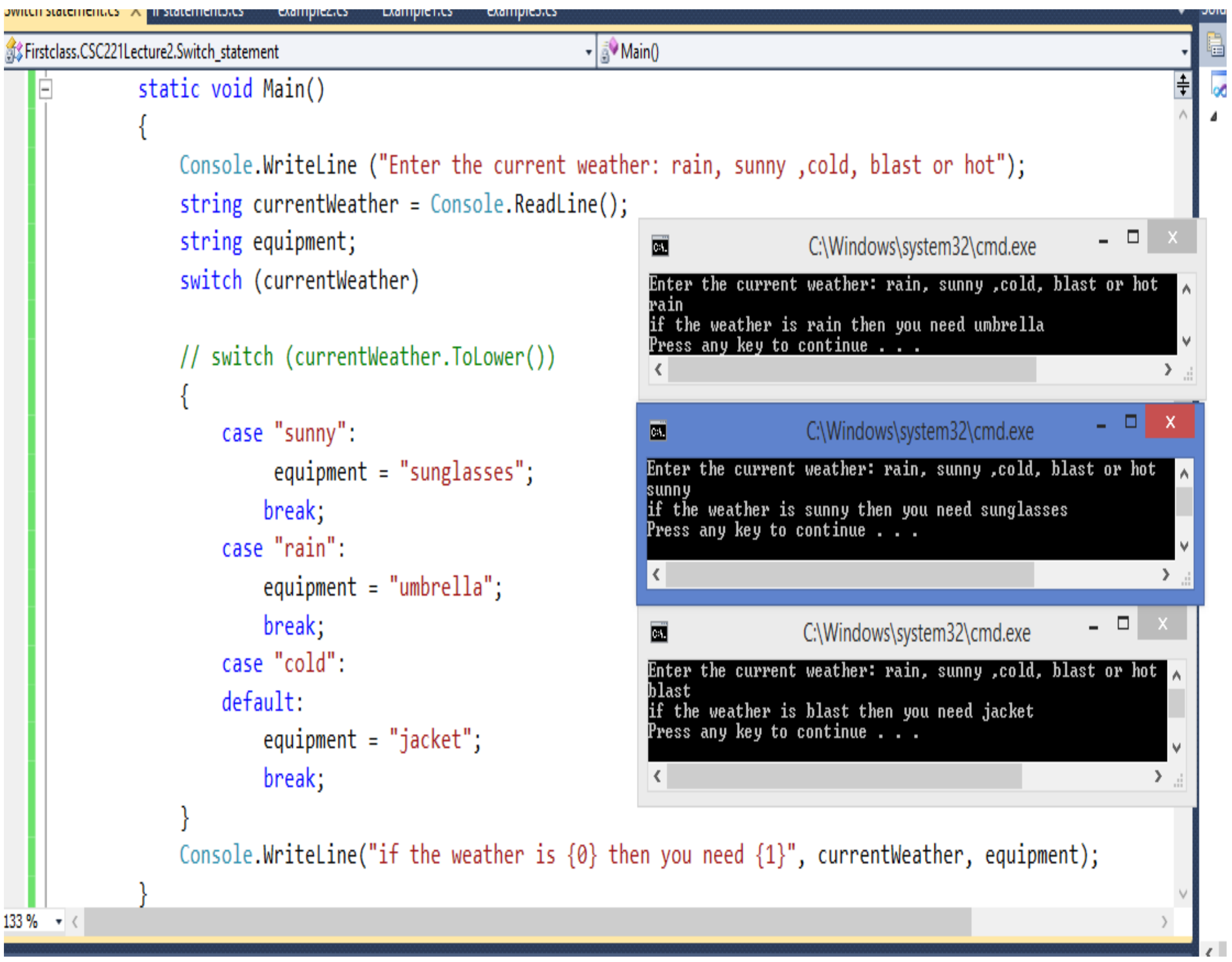
S/N	**Current weather**	**Equipment**
1.	**Sunny**	**Sunglasses**
2.	**Rain**	**Umbrella**
3.	**Cold**	**Jacket**
4	**Others**	**Jacket**

Solution program

```
{
        Console.WriteLine ("Enter the current weather: rain, sunny ,cold, blast or
hot");
         string currentWeather = Console.ReadLine();
         string equipment;
         switch (currentWeather)                    // this line will be sensitive to the
cases
            {
             case "sunny":
                 equipment = "sunglasses";
                break;
             case "rain":
                equipment = "umbrella";
                break;
             case "cold":
```

```
        default:
            equipment = "jacket";
            break;
}
```

Figure 12 shows switch statement in C#



```
Firstclass.CSC221Lecture2.Switch_statement        Main()

        static void Main()
        {
            Console.WriteLine ("Enter the current weather: rain, sunny ,cold, blast or hot");
            string currentWeather = Console.ReadLine();
            string equipment;
            switch (currentWeather)

            // switch (currentWeather.ToLower())
            {
                case "sunny":
                      equipment = "sunglasses";
                    break;
                case "rain":
                    equipment = "umbrella";
                    break;
                case "cold":
                default:
                    equipment = "jacket";
                    break;
            }
            Console.WriteLine("if the weather is {0} then you need {1}", currentWeather, equipment);
        }
```

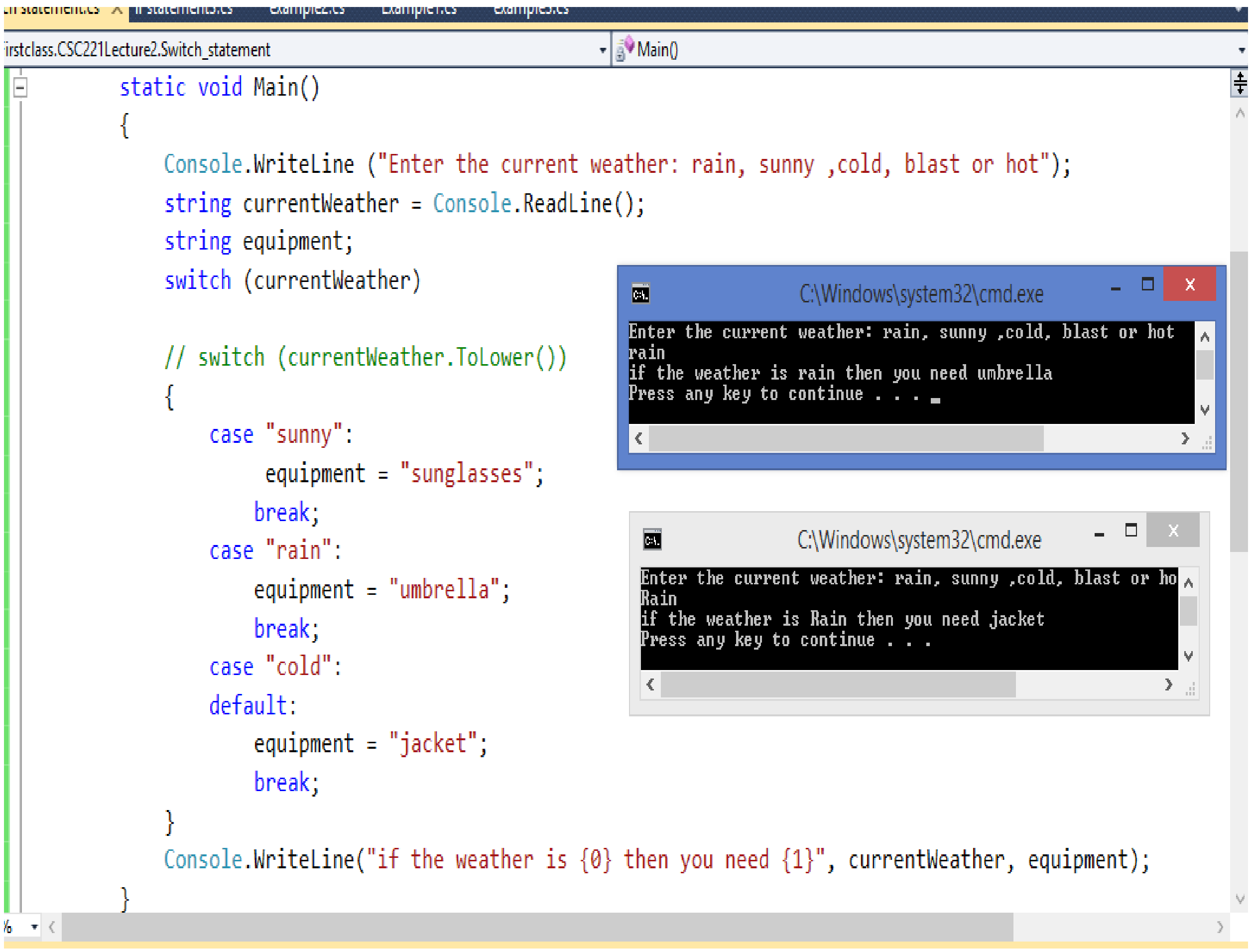
```
C:\Windows\system32\cmd.exe
Enter the current weather: rain, sunny ,cold, blast or hot
rain
if the weather is rain then you need umbrella
Press any key to continue . . .
```

```
C:\Windows\system32\cmd.exe
Enter the current weather: rain, sunny ,cold, blast or hot
sunny
if the weather is sunny then you need sunglasses
Press any key to continue . . .
```

```
C:\Windows\system32\cmd.exe
Enter the current weather: rain, sunny ,cold, blast or hot
blast
if the weather is blast then you need jacket
Press any key to continue . . .
```

***Figure 12**: Switch statement in C#*

Figure 13 shows a screenshot of applying switch statement in C#



```
static void Main()
{
    Console.WriteLine ("Enter the current weather: rain, sunny ,cold, blast or hot");
    string currentWeather = Console.ReadLine();
    string equipment;
    switch (currentWeather)

    // switch (currentWeather.ToLower())
    {
        case "sunny":
             equipment = "sunglasses";
            break;
        case "rain":
            equipment = "umbrella";
            break;
        case "cold":
        default:
            equipment = "jacket";
            break;
    }
    Console.WriteLine("if the weather is {0} then you need {1}", currentWeather, equipment);
}
```

```
Enter the current weather: rain, sunny ,cold, blast or hot
rain
if the weather is rain then you need umbrella
Press any key to continue . . .
```

```
Enter the current weather: rain, sunny ,cold, blast or ho
Rain
if the weather is Rain then you need jacket
Press any key to continue . . .
```

***Figure 13**: Applying Switch statement in C#*

Tutor Assessment

- Write a console application that will accept the students score in a course and print the letter grade as follows:
  - 70 and above: A
  - 60-69 B
  - 50-59 C
  - 45-49 D
  - Below 45 F

4.0 CONCLUSION

In this unit, we defined some basic concepts of C#. We also looked at the advancements in C# as well as the language name.

5.0 SUMMARY

We hope you enjoyed this unit. This unit provided an overview of C#: basic definition, features, versions and language name. Now, let us attempt the questions below.

6.0 TUTOR MARKED ASSIGNMENT

What is the different between the If statement and the Switch statement?.

UNIT QUIZ

Which of the following is not a legal variable name?

❑ myVariableIsGood
❑ 99Flake
❑ floor
❑ time2GetJiggyWidIt
❑ wrox.com

Write a console application that obtains four int values from the user and displays the product.

7.0 REFERENCES/FURTHER READINGS

1. Archer, Tom (2001). *Inside C#*. Microsoft Press. ISBN 0-7356-1288-9.
2. Cornelius, Barry (December 1, 2005). "Java 5 catches up with C#". University of Oxford Computing Services.
3. Ecma International. June 2006.*C# Language Specification* (4th ed.).
4. Guthrie, Scott (November 28, 2006). "What language was ASP.Net originally written in?"
5. Hamilton, Naomi (October 1, 2008). "The A-Z of Programming Languages: C#". Computerworld.
6. Horton, Anson (2006-09-11). "C# XML documentation comments FAQ".

7. Kovacs, James (September 07, 2007). "C#/.NET History Lesson".
8. Microsoft. September 4, 2003."Visual C#.net Standard" (JPEG).
9. Microsoft (June 18, 2009) *C# Language Reference*.
10. O' Reilly. (2002).*C# Language Pocket Reference*. ISBN 0-596-00429-X.
11. Petzold, Charles (2002). *Programming Microsoft Windows with C#*. Microsoft Press. ISBN 0-7356-1370-2.
12. Steinman, Justin (November 7, 2006). "Novell Answers Questions from the Community".
13. Stallman, Richard (June 26, 2009). "Why free software shouldn't depend on Mono or C#".
14. Simon, Raphael; Stapf, Emmanuel; Meyer, Bertrand (June 2002). "Full Eiffel on the .NET Framework".
15. "The ECMA C# and CLI Standards". 2009 -07-06.
16. Zander, Jason (November 23, 2007). "Couple of Historical Facts".

UNIT2 C#.NET LOOPING AND ITERATIONS

CONTENT

1.0 INTRODUCTION

Program control is how a program makes decisions or organizes its activities. Program control typically involves executing particular code based on the outcome of a prior operation or a user input.

2.0 OBJECTIVES

At the end of this unit, you should be able to:

- Understand how program control is structured
- use different types of program controls in C#
- Able use looping methods

3.0 MAIN CONTENT

3.1 What is Program control?

Is basically the ways computer handles the flow of

instruction execution in a program. Program control is achieved through the application of control structure which can be divided into three: simple sequence, selection and iteration/looping control structure.

SELF ASSESSMENT EXERCISE

Write short note on simple sequence, selection and iteration control structure.

3.2 C# Looping Methods

The following looping methods exists in C#.net
Looping:

- do Loops
- while Loops
- for Loops
- Interrupting Loops

Do loop

The structure of a do loop is as follows, where < Test > evaluates to a Boolean value:

```
do
{
        < code to be looped >
} while ( < Test > );
```

The semicolon after the while *statement is required.*

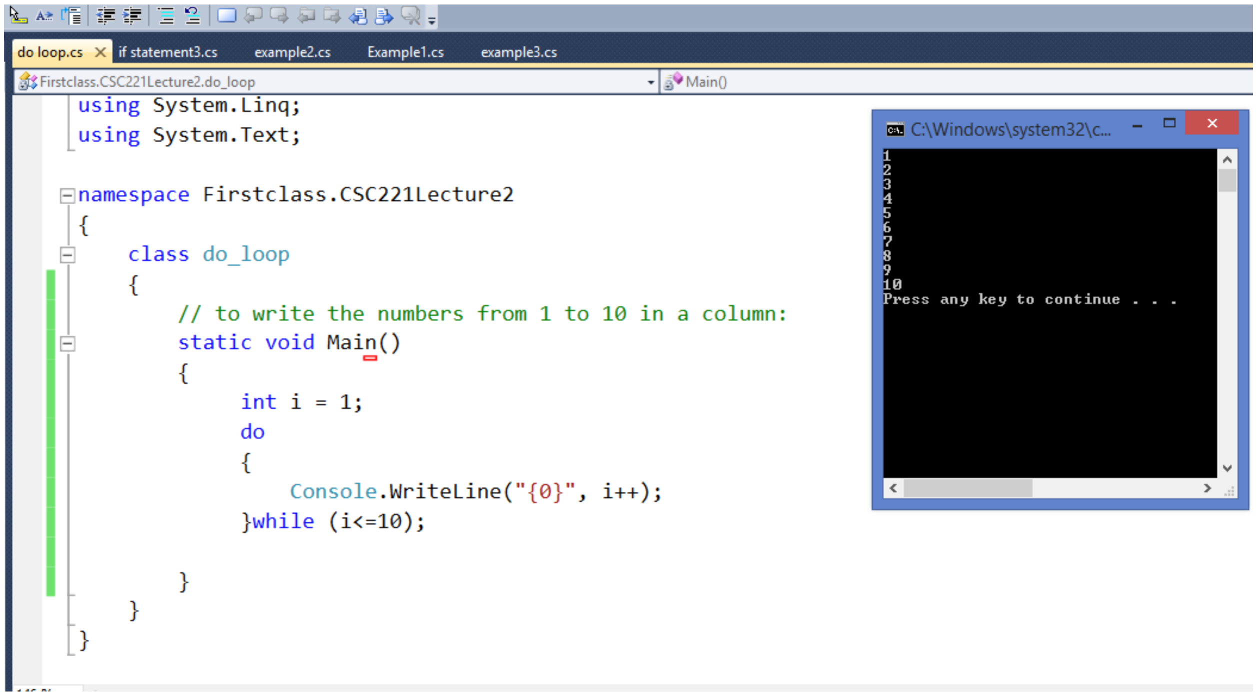
Example

For example, you could use the following to write the numbers from 1 to 10 in a column:

```
static void Main()
{
    int i = 1;
    do
    {
       Console.WriteLine("{0}", i++);
    }while (i<=10);
}
```

Figure 14 shows a screen shot of do while loop statement in C#.

Newly introduced diagram



```
using System.Linq;
using System.Text;

namespace Firstclass.CSC221Lecture2
{
    class do_loop
    {
        // to write the numbers from 1 to 10 in a column:
        static void Main()
        {
            int i = 1;
            do
            {
                Console.WriteLine("{0}", i++);
            }while (i<=10);

        }
    }
}
```

```
1
2
3
4
5
6
7
8
9
10
Press any key to continue . . .
```

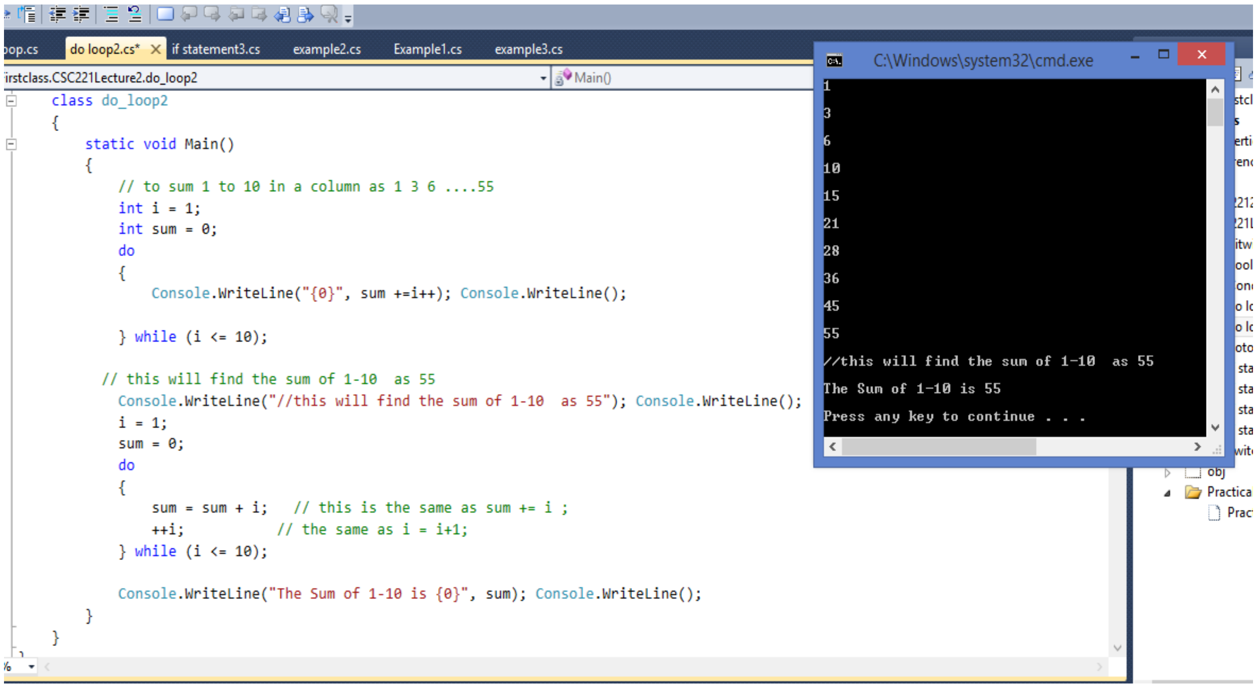
***Figure 14**: Using Do-While loop statement in C#*

- You can use the following to sum number 1 to 10: $\sum_{x=1}^{x=10} x$

```
// this will find the sum of 1-10  as 55
Console.WriteLine("this will find the sum of 1-10  as 55");
          Console.WriteLine();
          i = 1;
         sum = 0;
           do
           {
              sum = sum + i;              // this is the same as sum += i ;
              ++i;                        // the same as i = i+1;
           } while (i <= 10);
           Console.WriteLine("The Sum of 1-10 is {0}", sum); Console.WriteLine();
```

Figure 15 shows a screenshot of applying do while loop statement in C#.

Newly introduced diagram



```
class do_loop2
{
    static void Main()
    {
        // to sum 1 to 10 in a column as 1 3 6 ....55
        int i = 1;
        int sum = 0;
        do
        {
            Console.WriteLine("{0}", sum +=i++); Console.WriteLine();

        } while (i <= 10);

      // this will find the sum of 1-10  as 55
        Console.WriteLine("//this will find the sum of 1-10  as 55"); Console.WriteLine();
        i = 1;
        sum = 0;
        do
        {
            sum = sum + i;   // this is the same as sum += i ;
            ++i;           // the same as i = i+1;
        } while (i <= 10);

        Console.WriteLine("The Sum of 1-10 is {0}", sum); Console.WriteLine();
    }
}
```

```
1
3
6
10
15
21
28
36
45
55
//this will find the sum of 1-10  as 55
The Sum of 1-10 is 55
Press any key to continue . . .
```

***Figure 15**: Applying Do-While loop statement in C#*

while Loops

while loops are very similar to do loops, but they have one important difference:
The Boolean test in a while loop takes place at the start of the loop cycle, not the end.
If the test evaluates to false, then the loop cycle is never executed.
Instead, program execution jumps straight to the code following the loop.
The Syntax
while (< *Test* >)
{
    < *code to be looped* >
}

for Loops

for loop executes a set number of times and maintains its own counter.
To define a for loop you need the following information:
A starting value to initialize the counter variable .
A condition for continuing the loop, involving the counter variable .
An operation to perform on the counter variable at the end of each loop cycle

This information must be placed into the structure of a for loop as follows:

```
for ( < initialization > ; < condition > ; < operation > )
{
        < code to loop >
}
```

Interrupting Loops

- *break* — Causes the loop to end immediately .
- *continue* — Causes the current loop cycle to end immediately (execution continues with the next loop cycle) .
- *goto* — Allows jumping out of a loop to a labeled position (not recommended if you want your code to be easy to read and understand) .
- return — Jumps out of the loop and its containing function (this will be treated under function in the next chapter)
- The ***break*** command simply exits the loop, and execution continues at the first line of code after the loop, as shown in the following example:

  ```
  int i = 1;
  while (i < = 10)
  {
  if (i == 6) // when this is true stop the loop
  break;
  Console.WriteLine("{0}", i++);
  }
  ```

Continue

Continue only stops the current cycle, not the whole loop, as shown here:

```
int i;
for (i = 1; i < = 10; i++)
{
if ((i % 2) == 0)
continue;
Console.WriteLine(i);
}
```

Goto

The third method of interrupting a loop is to use ***goto*** , as shown earlier:

```
int i = 1;
while (i < = 10)
{
        if (i == 6)
        goto exitPoint;
        Console.WriteLine("{0}", i++);
}
Console.WriteLine("This code will never be reached.");
exitPoint:
Console.WriteLine("This code is run when the loop is exited using goto.");
```

4.0 CONCLUSION

In this unit, we defined some basic concepts of C#. We also looked at the advancements in C# as well as the language name.

5.0 SUMMARY

We hope you enjoyed this unit. This unit provided an overview of looping and iteration. Now, let us attempt the questions below.

6.0 TUTOR MARKED ASSIGNMENT

What is the different between the If statement and the Switch statement?.

UNIT QUIZ

Which of the following is not a legal variable name?

❑ myVariableIsGood
❑ 99Flake
❑ floor
❑ time2GetJiggyWidIt
❑ wrox.com

Write a console application that obtains four int values from the user and displays the product.

7.0 REFERENCES/FURTHER READINGS

1. Archer, Tom (2001). *Inside C#*. Microsoft Press. ISBN 0-7356-1288-9.
2. Cornelius, Barry (December 1, 2005). "Java 5 catches up with C#". University of Oxford Computing Services.
3. Ecma International. June 2006.*C# Language Specification* (4th ed.).
4. Guthrie, Scott (November 28, 2006). "What language was ASP.Net originally written in?"
5. Hamilton, Naomi (October 1, 2008). "The A-Z of Programming Languages: C#". Computerworld.
6. Horton, Anson (2006-09-11). "C# XML documentation comments FAQ".
7. Kovacs, James (September 07, 2007). "C#/.NET History Lesson".
8. Microsoft. September 4, 2003."Visual C#.net Standard" (JPEG).
9. Microsoft (June 18, 2009) *C# Language Reference*.
10. O' Reilly. (2002).*C# Language Pocket Reference*. ISBN 0-596-00429-X.
11. Petzold, Charles (2002). *Programming Microsoft Windows with C#*. Microsoft Press. ISBN 0-7356-1370-2.
12. Steinman, Justin (November 7, 2006). "Novell Answers Questions from the Community".
13. Stallman, Richard (June 26, 2009). "Why free software shouldn't depend on Mono or C#".
14. Simon, Raphael; Stapf, Emmanuel; Meyer, Bertrand (June 2002). "Full Eiffel on the .NET Framework".
15. "The ECMA C# and CLI Standards". 2009 -07-06.
16. Zander, Jason (November 23, 2007). "Couple of Historical Facts".

UNIT 3: C#.NET TYPE CONVERSION

CONTENT

1.0 INTRODUCTION

Program control is how a program makes decisions or organizes its activities. Program control typically involves executing particular code based on the outcome of a prior operation or a user input.

2.0 OBJECTIVES

At the end of this unit, you should be able to:

- Understand Various Data Types
- Able to do Type conversion
- Able to use complex variable types
- Able to debug and handle errors in C# program

3.0 MAIN CONTENT

3.1 Type Conversion

Type conversion is basically converting from one data type to another either implicitly or explicitly.

We have two types of type conversion in C#.net: Implicit conversion

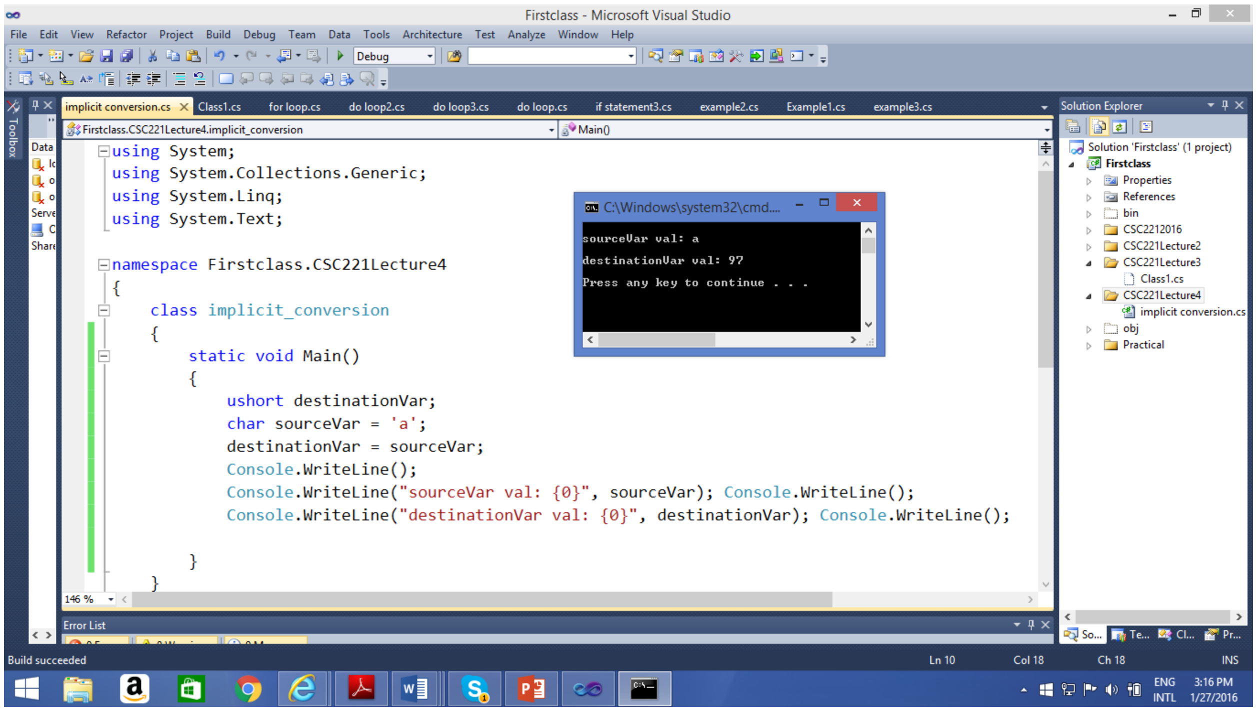
Explicit conversion

3.2: The Implicit Conversion

```
ushort  destinationVar;
char  sourceVar = ‘a’;
destinationVar = sourceVar;
Console.WriteLine(“sourceVar val: {0}”, sourceVar);
Console.WriteLine(“destinationVar val: {0}”, destinationVar);
```

Example of running program

Figure 16 shows a screen shot of implicit conversion in C#.



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

namespace Firstclass.CSC221Lecture4
{
    class implicit_conversion
    {
        static void Main()
        {
            ushort destinationVar;
            char sourceVar = 'a';
            destinationVar = sourceVar;
            Console.WriteLine();
            Console.WriteLine("sourceVar val: {0}", sourceVar); Console.WriteLine();
            Console.WriteLine("destinationVar val: {0}", destinationVar); Console.WriteLine();

        }
    }
```

```
sourceVar val: a
destinationVar val: 97
Press any key to continue . . .
```

***Figure 16**: Implicit Conversion in C#*

The implicit Conversion table for C# is highlighted in Table 4.

***Table 4**: Implicit Conversion in C#*

Type	Can Safely Be Converted To
Byte	short, ushort, int, uint, long, ulong, float, double, decimal
Sbyte	short, int, long, float, double, decimal
Short	int, long, float, double, decimal
Ushort	int, uint, long, ulong, float, double, decimal
Int	long, float, double, decimal
Uint	long, ulong, float, double, decimal
Long	float, double, decimal
Ulong	float, double, decimal
Float	double
Char	ushort, int, uint, long, ulong, float, double, decimal

3.3: Explicit Conversion

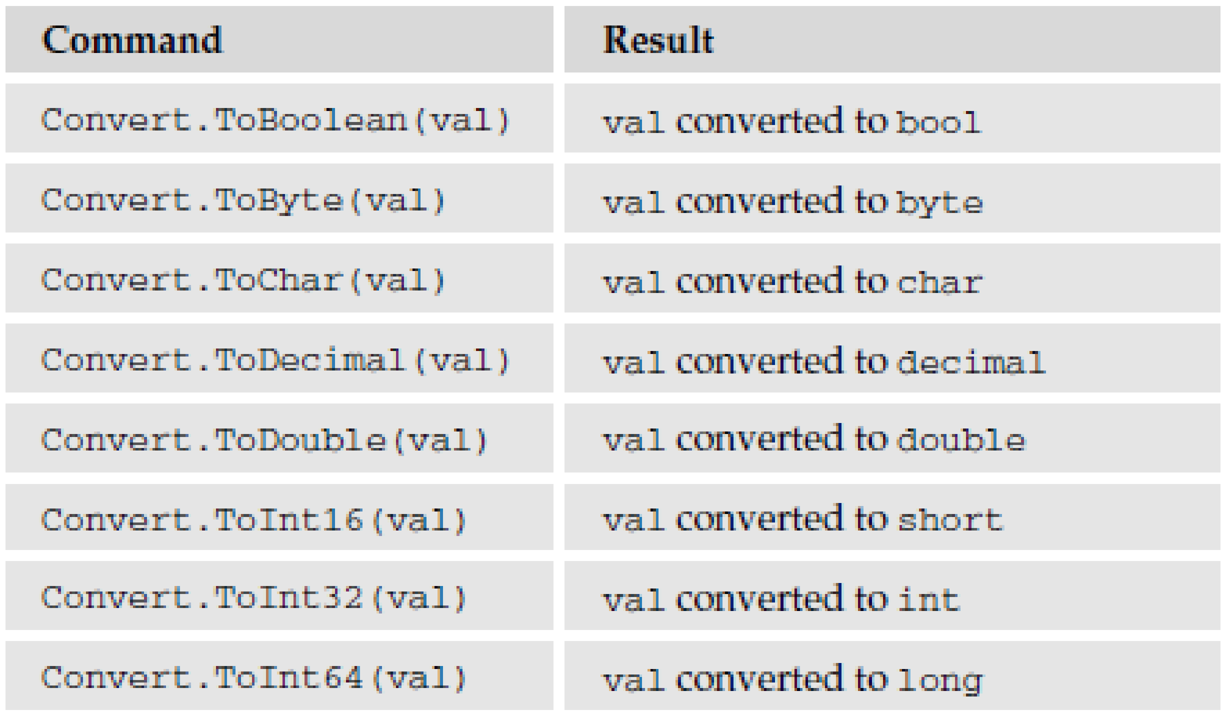
You can explicitly ask the compiler to convert a value from one data type to another if they are compatible and when precision will not be lost.

```
byte  destinationVar;
short  sourceVar = 7;
destinationVar = sourceVar;
Console.WriteLine("sourceVar val: {0}", sourceVar);
Console.WriteLine("destinationVar val: {0}", destinationVar);
```

Table 5 shows explicit conversion in C#

Explicit Conversion table

Table 5: *Explicit conversion in C#*



Command	Result
`Convert.ToBoolean(val)`	`val` converted to `bool`
`Convert.ToByte(val)`	`val` converted to `byte`
`Convert.ToChar(val)`	`val` converted to `char`
`Convert.ToDecimal(val)`	`val` converted to `decimal`
`Convert.ToDouble(val)`	`val` converted to `double`
`Convert.ToInt16(val)`	`val` converted to `short`
`Convert.ToInt32(val)`	`val` converted to `int`
`Convert.ToInt64(val)`	`val` converted to `long`

SELF ASSESSMENT EXERCISE

Write short note on different type conversions in C#.

3.4 Complex Variable Types

Enumeration is one key example of a complex variable type.
Enumerations allow the definition of a *type* that can take one of a finite set of values that you supply. The Syntax:
enum *typeName*
{
value1 ,
value2 ,

value3 ,
...
valueN
}

- you can declare variables of this new type as follows:
  typeName varName ;
- You can assign values using the following:
  varName = *typeName*.*value* ;
- Enumerations have an *underlying type* used for storage.
  Each of the values that an enumeration type can take is stored as a value of this underlying type, which by default is ***int*** .

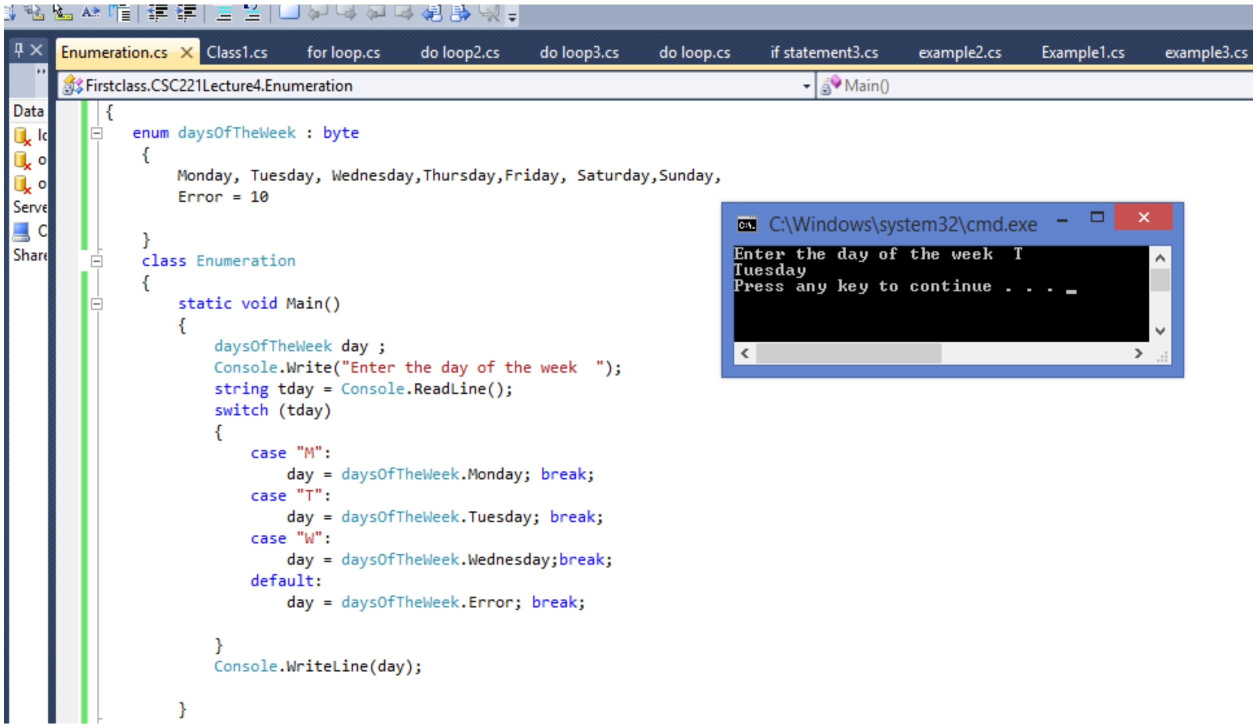
You can specify a different underlying type by adding the type to the enumeration declaration:

```
enum typeName : underlyingType
{
    value1 ,
    value2 ,
    value3 ,
    ...
    valueN
}
```

Enumerations can have underlying types of byte , sbyte , short , ushort , int , uint , long , and ulong .

Running program example

The Switch statement demonstration in C# is shown in Figure 17.



```
{
    enum daysOfTheWeek : byte
    {
        Monday, Tuesday, Wednesday,Thursday,Friday, Saturday,Sunday,
        Error = 10

    }
    class Enumeration
    {
        static void Main()
        {
            daysOfTheWeek day ;
            Console.Write("Enter the day of the week  ");
            string tday = Console.ReadLine();
            switch (tday)
            {
                case "M":
                    day = daysOfTheWeek.Monday; break;
                case "T":
                    day = daysOfTheWeek.Tuesday; break;
                case "W":
                    day = daysOfTheWeek.Wednesday;break;
                default:
                    day = daysOfTheWeek.Error; break;

            }
            Console.WriteLine(day);

        }
```

```
Enter the day of the week  T
Tuesday
Press any key to continue . . .
```

Figure 17: *Switch statement demonstration in C#*

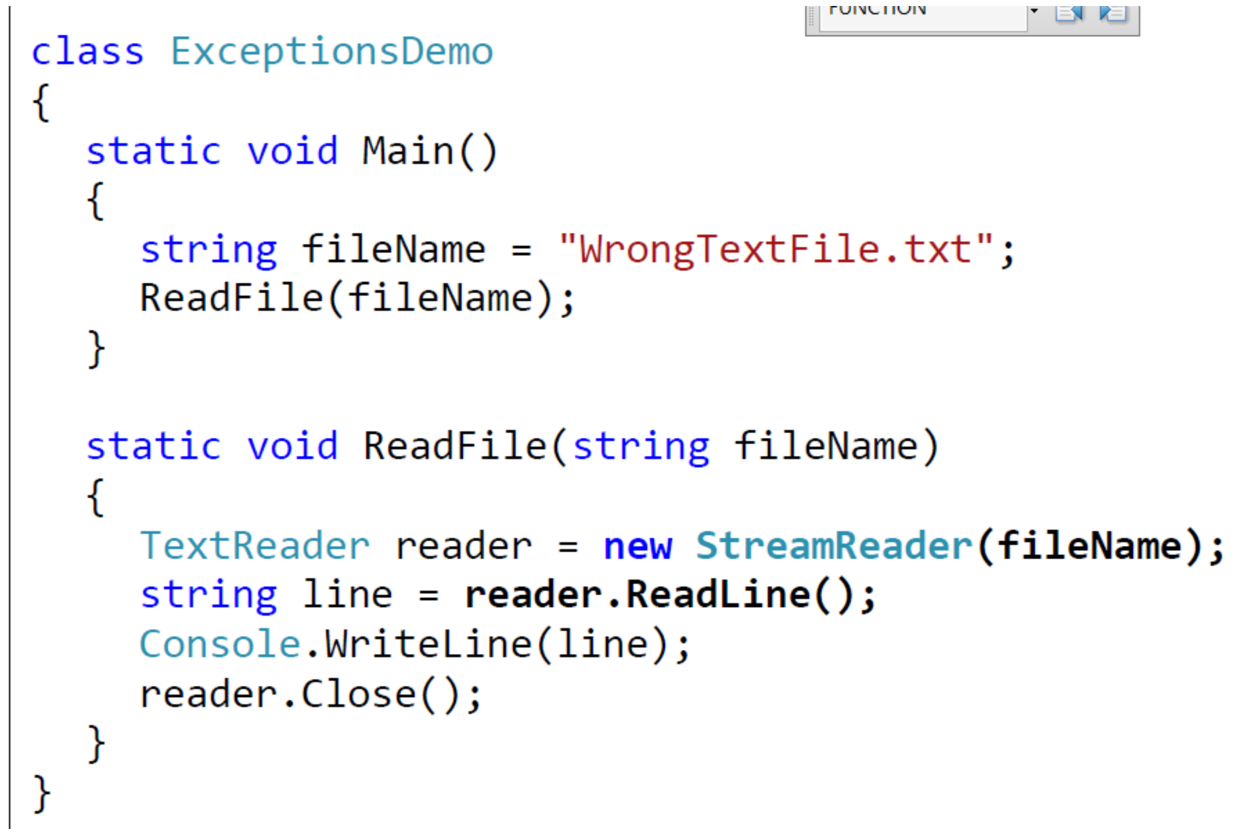
3.5 Errors and Exception Handling in C#

Exception is a notification that sometimes interrupts the normal program execution. Exceptions provide a programming paradigm for detecting and reacting to unexpected events. When an exception arises, the state of the program is saved, the normal flow is interrupted and the control is passed to an exception handler (if such exists in the current context). Exceptions are raised or thrown by programming code that must send a signal to the executing program about an error or an unusual situation. For example, if we try to open a file, which doesn’t exist, the code responsible for opening the file will detect this and will throw an exception with a proper error message.

Exception handling is a mechanism, which allows exceptions to be thrown and caught. Exceptions in the Object-Oriented Programming. In Object-Oriented Programming (OOP), exceptions are a powerful mechanism for centralized processing of errors and exceptional situations. Usually in OOP, a code executing some operation will cause an exception if there is a problem and the operation could not be successfully completed. The method causing the operation could catch the exception (and handle the error) or pass the exception through to the calling method.

Another fundamental concept is exceptions hierarchy. In OOP, exceptions are classes and they can be inherited to build hierarchies. When an exception is handled (caught), the handling mechanism could catch a whole class of exceptions and not just a particular error (as in the traditional procedural programming).
In OOP, it is recommended to use exceptions for managing error situations or unexpected events that may arise during a program execution.

A sample C# code demonstrating the use of exception is contains below:



```
class ExceptionsDemo
{
   static void Main()
   {
      string fileName = "WrongTextFile.txt";
      ReadFile(fileName);
   }

   static void ReadFile(string fileName)
   {
      TextReader reader = new StreamReader(fileName);
      string line = reader.ReadLine();
      Console.WriteLine(line);
      reader.Close();
   }
}
```

4.0 CONCLUSION

We discovered that The CSharp type system contains three Types, they are: ***Value Types, Reference Types and Pointer Types.*** We equally saw that the key difference between value types and reference types is the mode of storing values in memory.

5.0 SUMMARY

In this unit, we learnt about the general notion of type and the C# type system in particular. OK! Let us attempt the questions below.

6.0 TUTOR MARKED ASSIGNMENT

- Give a brief description of reference types
- State the application of pointer types

7.0 REFERENCES/FURTHER READINGS

1. Archer, Tom (2001). *Inside C#*. Microsoft Press. ISBN 0-7356-1288-9.
2. Cornelius, Barry (December 1, 2005). "Java 5 catches up with C#". University of Oxford Computing Services.
3. Ecma International. June 2006.*C# Language Specification* (4th ed.).
4. Guthrie, Scott (November 28, 2006). "What language was ASP.Net originally written in?"
5. Hamilton, Naomi (October 1, 2008). "The A-Z of Programming Languages: C#". Computerworld.
6. Horton, Anson (2006-09-11). "C# XML documentation comments FAQ".
7. Kovacs, James (September 07, 2007). "C#/.NET History Lesson".
8. Microsoft. September 4, 2003."Visual C#.net Standard" (JPEG).
9. Microsoft (June 18, 2009) *C# Language Reference*.
10. O' Reilly. (2002).*C# Language Pocket Reference*. ISBN 0-596-00429-X.
11. Petzold, Charles (2002). *Programming Microsoft Windows with C#*. Microsoft Press. ISBN 0-7356-1370-2.
12. Steinman, Justin (November 7, 2006). "Novell Answers Questions from the Community".
13. Stallman, Richard (June 26, 2009). "Why free software shouldn't depend on Mono or C#".
14. Simon, Raphael; Stapf, Emmanuel; Meyer, Bertrand (June 2002). "Full Eiffel on the .NET Framework".
15. "The ECMA C# and CLI Standards". 2009 -07-06.
16. Zander, Jason (November 23, 2007). "Couple of Historical Facts".

MODULE 4 ARRAYS, STRING MANIPULATIONS AND FUNCTIONS

UNIT 1: C#.NET ARRAYS

CONTENTS

1.0 INTRODUCTION

Array provides structured ways of storing data values. In this unit, concept of Array is demystified. Different types of Array is also illustrated..

1.0 **OBJECTIVES**

At the end of this unit, you should be able to:

- Explain the concept of ‘Array
- Declare and use Array in C# program
- Declare and use multi-dimensional Array in C# program

3.0 MAIN CONTENT

3.1 Arrays and Multidimensional arrays

Sometimes you want to store several values of the same type at the same time, without having to use a different variable for each value.
Example
string friendName1 = “Robert Barwell”;

```
string friendName2 = “Mike Parry”;
string friendName3 = “Jeremy Beacock”;
```

SELF ASSESSMENT EXERCISE

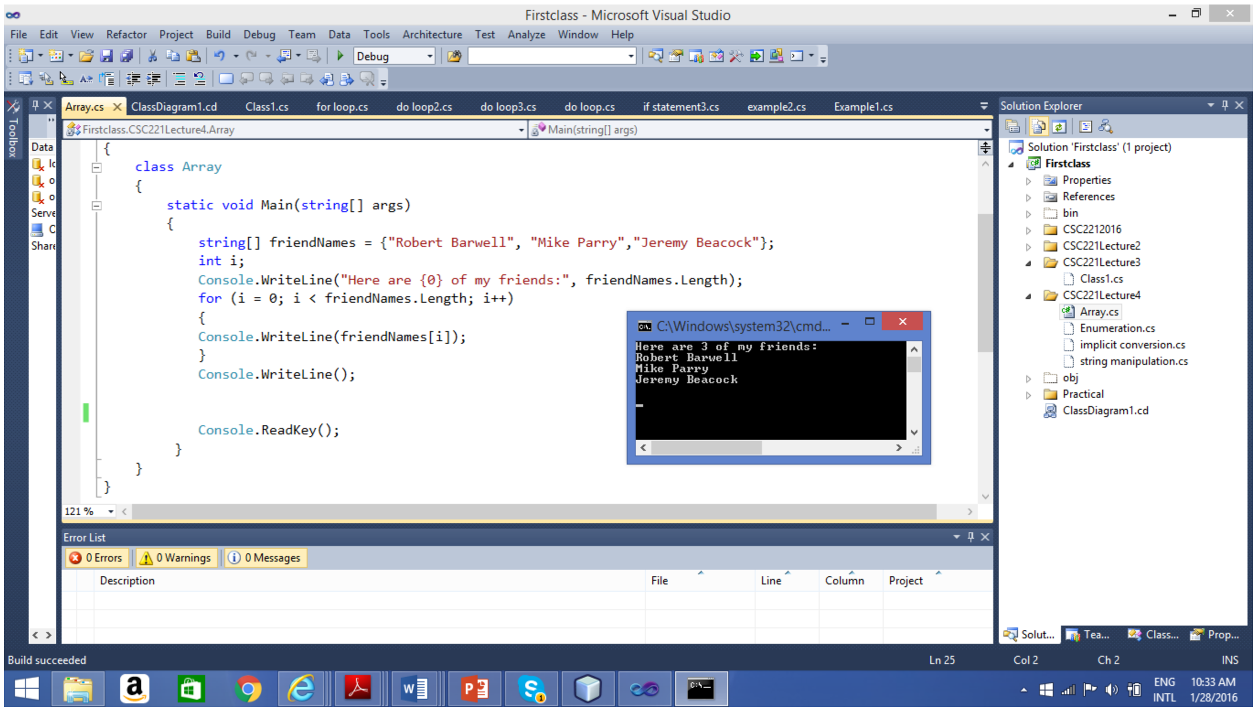
Arrays are indexed lists of variables stored in a single array type variable.
For example, you might have an array called friendNames that stores the three names shown in previous slide.
You can access individual members of this array by specifying their index in square brackets, as

```
friendNames[ < index > ]
```

This index is simply an integer, starting with 0 for the first entry, using 1 for the second, and so on.
This means that you can go through the entries using a loop:

```
int i;
for (i = 0; i < 3; i++)
{
Console.WriteLine(“Name with index of {0}: {1}”, i, friendNames[i]);
}
```

Figure 18 shows a screen shot of using loops and arrays in C~.


Firstclass - Microsoft Visual Studio
File Edit View Refactor Project Build Debug Team Data Tools Architecture Test Analyze Window Help
Debug
Array.cs ClassDiagram1.cd Class1.cs for loop.cs do loop2.cs do loop3.cs do loop.cs if statement3.cs example2.cs Example1.cs
Firstclass.CSC221Lecture4.Array
Main(string[] args)
Solution Explorer
Solution 'Firstclass' (1 project)
Firstclass
Properties
References
bin
CSC2212016
CSC221Lecture2
CSC221Lecture3
Class1.cs
CSC221Lecture4
Array.cs
Enumeration.cs
implicit conversion.cs
string manipulation.cs
obj
Practical
ClassDiagram1.cd
Error List
0 Errors 0 Warnings 0 Messages
Description File Line Column Project
121 %
Build succeeded
Ln 25 Col 2 Ch 2 INS
C:\Windows\system32\cmd...

```
{
    class Array
    {
        static void Main(string[] args)
        {
            string[] friendNames = {"Robert Barwell", "Mike Parry","Jeremy Beacock"};
            int i;
            Console.WriteLine("Here are {0} of my friends:", friendNames.Length);
            for (i = 0; i < friendNames.Length; i++)
            {
            Console.WriteLine(friendNames[i]);
            }
            Console.WriteLine();


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

```
Here are 3 of my friends:
Robert Barwell
Mike Parry
Jeremy Beacock
```

***Figure 18**: Using Loop and Array in C#*

3.2 Declaring and using Arrays in C# program

Arrays are declared in the following way:

< baseType > [ ] < name > ;

< baseType > may be any variable type.
Arrays must be initialized before you have access to them.
You can ’ t just access or assign values to the array elements like this:

```
int[ ]  myIntArray;
myIntArray[10]  = 5
```

Arrays can be initialized in two ways.
You can either specify the complete contents of the array in a literal form, or you can specify the size of the array and use the new keyword to initialize all array elements.

Specifying an array using literal values simply involves providing a comma - separated list of element values enclosed in curly braces:

```
int[] myIntArray = {5, 9, 10, 2, 99};
```

Here, myIntArray has five elements, each with an assigned integer value

The other method requires the following syntax:

```
int[ ] myIntArray = new int [5];
```

Here, you use the new keyword to explicitly initialize the array, and a constant value to define the size.

This method results in all the array members being assigned a default value, which is 0 for numeric types.

Class Exercise

Write a program in C# Sharp to calculate the sum of elements in an array. For example, if you enter 5,7,3,2,9 the program should display
The sum of the elements of the array is 26

3.3 Using two and multi-dimensional Array

A two - dimensional array such as this is declared as follows:

```
< baseType > [,] < name > ;
```

Arrays of more dimensions simply require more commas:

```
    < baseType > [,,,] < name > ;
double[,] hillHeight = new double [3,4];
```

Alternatively, you can use literal values for initial assignment. Here, you use nested blocks of curly braces, separated by commas:

```
double[,] hillHeight = {{1, 2, 3, 4}, {2, 3, 4, 5}, {3, 4, 5, 6}};
```

This array has the same dimensions as the previous one — that is, three rows and four columns.

By providing literal value these dimensions are defined implicitly.
To access individual elements of a multidimensional array, you simply specify the indices separated by commas:

hillHeight[2,1]

You can then manipulate this element just as you can other elements.
This expression will access the second element of the third nested array as defined previously (the value will be 4).

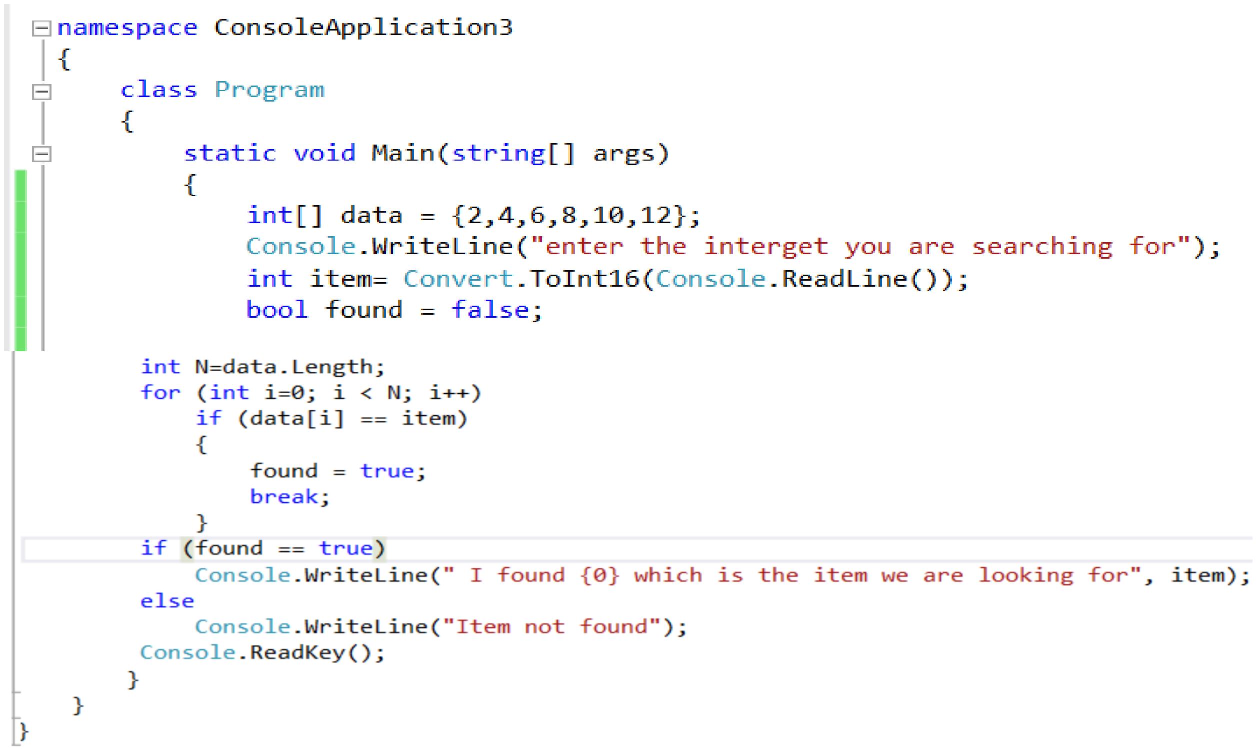
Class Work

Write a program to read in the following array and display it again looking exactly like this.

```
1       2       3       4
5       6       7       8
9       10      11      12
```

Searching an Array – Linear Search algorithm

A sample code for searching an array using linear search algorithm is shown below:



```
namespace ConsoleApplication3
{
    class Program
    {
        static void Main(string[] args)
        {
            int[] data = {2,4,6,8,10,12};
            Console.WriteLine("enter the interget you are searching for");
            int item= Convert.ToInt16(Console.ReadLine());
            bool found = false;

        int N=data.Length;
        for (int i=0; i < N; i++)
            if (data[i] == item)
            {
                found = true;
                break;
            }
        if (found == true)
            Console.WriteLine(" I found {0} which is the item we are looking for", item);
        else
            Console.WriteLine("Item not found");
        Console.ReadKey();
       }
    }
}
```

Sorting an Array (Selection Sort Algorithm)

The selection sort algorithm is based on the idea of selecting the largest array element and swapping it with the last array element. The result is an unsorted list whose size is 1 less than the size of the original list, which is followed by a sorted list of size 1 (remember that a single value is a sorted list of size 1.) If we do this step again on the unsorted list, we will have an ordered list of size 2 at the end, and the size of the unordered list will be reduced by 1. Repetitive application of this process will give us a sorted list when the size of the unsorted list becomes one. This technique is known as simple selection sort. Presented in the following codes:

```
for (int i = 0; i < array_to_sort.Length - 1; i++)
        {
            int minValue = i;
            for (int j = i + 1; j < array_to_sort.Length; j++)
            {
                if (array_to_sort[j] < array_to_sort[minValue])
                {
                    minValue = j;
                }
            }
            /*Swap Code*/
            int tempData = array_to_sort[minValue];
            array_to_sort[minValue] = array_to_sort[i];
            array_to_sort[i] = tempData;
        }
```

Assignment

Write and implement a C#.net program to arrange the ages of ten students in a class in ascending order of youngest to the oldest in the class.

4.0 CONCLUSION

Concept of array is discussed and illustrated in this unit.

5.0 SUMMARY In summary, this unit looked an essential concept of C# array and different types. We can now attempt the questions below.

6.0 TUTOR MARKED ASSIGNMENT

Explain the term Multi-dimensional array?

7.0 REFERENCES/FURTHER READINGS

1. Archer, Tom (2001). *Inside C#*. Microsoft Press. ISBN 0-7356-1288-9.
2. Cornelius, Barry (December 1, 2005). "Java 5 catches up with C#". University of Oxford Computing Services.
3. Ecma International. June 2006.*C# Language Specification* (4th ed.).
4. Guthrie, Scott (November 28, 2006). "What language was ASP.Net originally written in?"
5. Hamilton, Naomi (October 1, 2008). "The A-Z of Programming Languages: C#". Computerworld.
6. Horton, Anson (2006-09-11). "C# XML documentation comments FAQ".
7. Kovacs, James (September 07, 2007). "C#/.NET History Lesson".
8. Microsoft. September 4, 2003."Visual C#.net Standard" (JPEG).
9. Microsoft (June 18, 2009) *C# Language Reference*.
10. O' Reilly. (2002).*C# Language Pocket Reference*. ISBN 0-596-00429-X.
11. Petzold, Charles (2002). *Programming Microsoft Windows with C#*. Microsoft Press. ISBN 0-7356-1370-2.
12. Steinman, Justin (November 7, 2006). "Novell Answers Questions from the Community".
13. Stallman, Richard (June 26, 2009). "Why free software shouldn't depend on Mono or C#".
14. Simon, Raphael; Stapf, Emmanuel; Meyer, Bertrand (June 2002). "Full Eiffel on the .NET Framework".
15. "The ECMA C# and CLI Standards". 2009 -07-06.
16. Zander, Jason (November 23, 2007). "Couple of Historical Facts".

UNIT 2 C#. Net Strings Manipulation

CONTENTS

1.0 INTRODUCTION

In this unit, we'll be considering string manipulations and functions in C#.

2.0 OBJECTIVES

At the end of this unit, you should be able to:

• State what string manipulations are
• Discover how to declare and use string variable in C#
• various string manipulation inbuilt functions in C#
• Give specific examples of each string manipulation and functions in C#

3.0 MAIN CONTENT

3.1 String manipulation

in a programming language describes how characters that forms a string can be manipulated in C#.

To start with, note that a string type variable can be treated as a read - only array of char variables.
This means that you can access individual characters using syntax like the following:
string myString = “A string”;
char myChar = myString[1];

- You can't assign individual characters in this way.

- To get a char array that you can write to, you can use the following code.
- This uses the ToCharArray() command of the array variable:

```
string myString = “A string”;
char[] myChars = myString.ToCharArray();
```

As with arrays, you can also get the number of elements using myString.Length .
This gives you the number of characters in the string:

```
string myString = Console.ReadLine();
Console.WriteLine(“You typed {0} characters.”, myString.Length);
string userResponse = Console.ReadLine();
userResponse = userResponse.Trim();
if (userResponse.ToLower() == “yes”)
{
                    // Act on response.
}
```

Another example

```
char[] trimChars = {‘ ‘, ‘e’, ‘s’};
string userResponse = Console.ReadLine();
userResponse = userResponse.ToLower();
userResponse = userResponse.Trim(trimChars);
if (userResponse == “y”)
{
// Act on response.
}
```

SELF ASSESSMENT EXERCISE

Write a string processing program in C# to search for the length of surname of a student named: Jonathan DANIEL, assuming DANIEL is the surname.

4.0 CONCLUSION

In conclusion, **CSharp** is a strongly typed language, therefore every variable and object must have a declared type. The following are the commonly used datatypes in C#: bool, int, decimal and string.

5.0 SUMMARY

In this unit, we discovered what data types portray and identified how to declare a variable in C#. We equally considered the common data types in C# and gave specific examples of each data type in C# Hope you grasped the key points. Now, let us attempt the questions below.

6.0 TUTOR MARKED ASSIGNMENT

• List common string manipulation functions in C#

7.0 REFERENCES/FURTHER READINGS

1. Archer, Tom (2001). *Inside C#*. Microsoft Press. ISBN 0-7356-1288-9.
2. Cornelius, Barry (December 1, 2005). "Java 5 catches up with C#". University of Oxford Computing Services.
3. Ecma International. June 2006.*C# Language Specification* (4th ed.).
4. Guthrie, Scott (November 28, 2006). "What language was ASP.Net originally written in?"
5. Hamilton, Naomi (October 1, 2008). "The A-Z of Programming Languages: C#". Computerworld.
6. Horton, Anson (2006-09-11). "C# XML documentation comments FAQ".
7. Kovacs, James (September 07, 2007). "C#/.NET History Lesson".
8. Microsoft. September 4, 2003."Visual C#.net Standard" (JPEG).
9. Microsoft (June 18, 2009) *C# Language Reference*.
10. O' Reilly. (2002).*C# Language Pocket Reference*. ISBN 0-596-00429-X.
11. Petzold, Charles (2002). *Programming Microsoft Windows with C#*. Microsoft Press. ISBN 0-7356-1370-2.
12. Steinman, Justin (November 7, 2006). "Novell Answers Questions from the Community".
13. Stallman, Richard (June 26, 2009). "Why free software shouldn't depend on Mono or C#".
14. Simon, Raphael; Stapf, Emmanuel; Meyer, Bertrand (June 2002). "Full Eiffel on the .NET Framework".
15. "The ECMA C# and CLI Standards". 2009 -07-06.
16. Zander, Jason (November 23, 2007). "Couple of Historical Facts".

UNIT 3: FUNCTIONS

CONTENT

1.0 Introduction

2.0 Objectives

3.0 Main Content

3.1 Function

A function is a basic part of a program.

Functions help to solve a certain problem and eventually take parameters and return a result. For example, if you have a function that calculates the maximum value in an array, you can use this function from any point in your code, and use the same lines of code in each case. This concept is referred to as reusability.

A function is a basic part of a program.

Functions help to solve a certain problem and eventually take parameters and return a result.

For example, if you have a function that calculates the maximum value in an array, you can use this function from any point in your code, and use the same lines of code in each case.

This concept is referred to as reusability

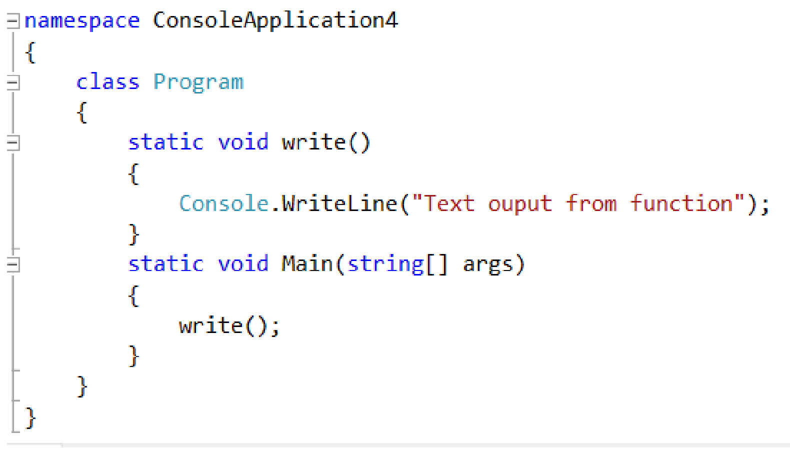
3.2 Advantages of Function

Functions make your code more readable, as you can use them to group related code together, thereby reducing the length of your code.
Functions can also be used to create multipurpose code, enabling them to perform the same operations on varying data.
For example, you could supply an array to search as a parameter and obtain the maximum value in the array as a return value. This means that you can use the same function to work with a different array each time.
Reusability - supply code once.

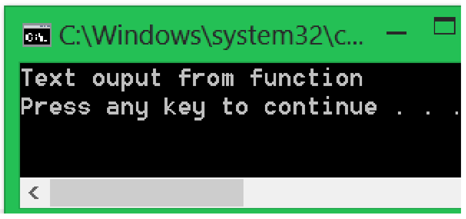
Declaring and using functions

Function definition consists of the following:
Two keywords: static and void
A function name followed by parentheses, e.g write()

A block of code to execute enclosed in curly braces is shown below:



```
namespace ConsoleApplication4
{
    class Program
    {
        static void write()
        {
            Console.WriteLine("Text ouput from function");
        }
        static void Main(string[] args)
        {
            write();
        }
    }
}
```



```
Text ouput from function
Press any key to continue . . .
```

Local Variables

Local variables are variables declared inside the body of a method.
The area where a local variable exists, and can be used, begins from the line where the variable is declared and ends at the closing curly bracket "}" of the method body.
This is the so-called area of visibility of the variable (variable scope).
If we try to declare variable, after we have already declared a variable with the same name, the code will not compile due to an error.

Return Values

The simplest way to exchange data with a function is to use a return value.
Functions that have return values evaluate to that value, in exactly the same way that variables evaluate to the values they contain when you use them in expressions.
Just like variables, return values have a type. When a function returns a value, you have to modify your function in two ways: Specify the type of the return value in the function declaration instead of using the void keyword.

Use the return keyword to end the function execution and transfer the return value to the calling code.

Invoking or calling a method is actually the process of execution of the method's code, placed into its body.
To invoke a method, write the method's name <method_name>, followed by the round brackets and semicolon ";" at the end: <method_name>();
When a function is to accept parameters, you must specify the following:
A list of the parameters accepted by the function in its definition, along with the types of those parameters .
A matching list of parameters in each function call .
This involves the following code, where you can have any number of parameters, each with a type and a name:

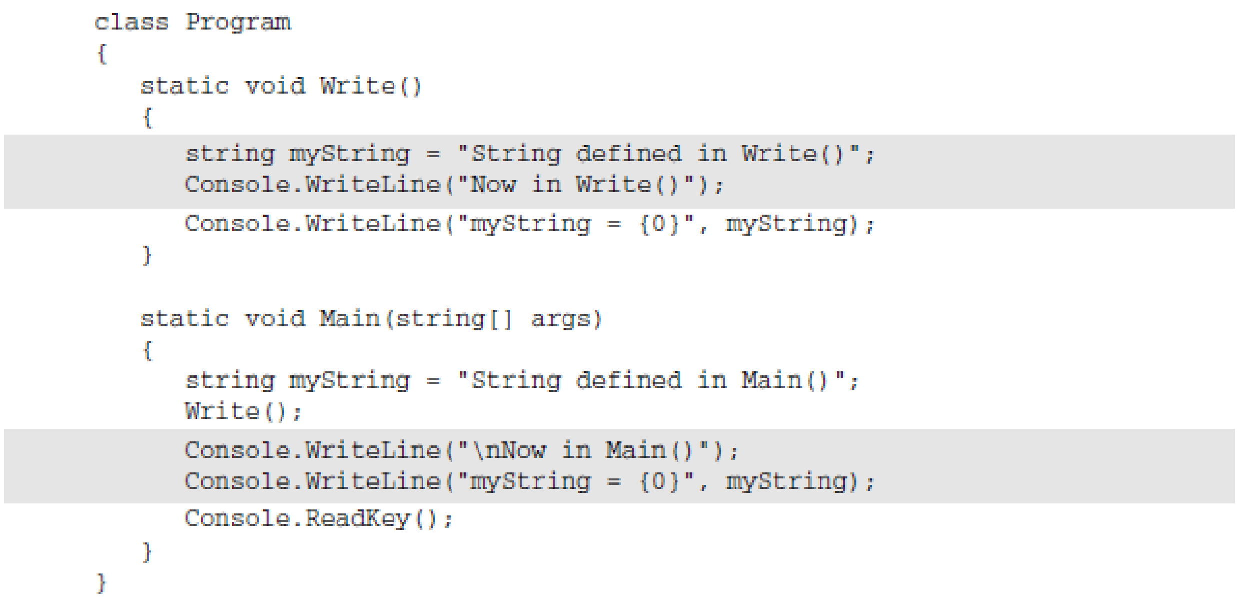
```
static < returnType > < functionName > ( < paramType > < paramName > , ...)
        {
        ...
        return < returnValue > ;
        }
```

The parameters are separated using commas, and each of these parameters is accessible

from code within the function as a variable.
For example, a simple function might take two double parameters and return their product:

```
static double Product(double param1, double param2)
{
return param1 * param2;
}
```

Program sample to demonstrate function usage is shown below:



```
class Program
{
   static void Write()
   {
      string myString = "String defined in Write()";
      Console.WriteLine("Now in Write()");
      Console.WriteLine("myString = {0}", myString);
   }

   static void Main(string[] args)
   {
      string myString = "String defined in Main()";
      Write();
      Console.WriteLine("\nNow in Main()");
      Console.WriteLine("myString = {0}", myString);
      Console.ReadKey();
   }
}
```

SELF ASSESSMENT EXERCISE

Method to Calculate the Sum of Prices of Books

Imagine we are in a bookstore and we want to calculate the amount of money we must pay for all the books we bought.
We will create a method that gets the prices of all the books as an array of type decimal[], and then returns the total amount we must pay:

Solution to the above problem is shown below:

```
static void PrintTotalAmountForBooks(decimal[] prices)
{
  decimal totalAmount = 0;
  foreach (decimal singleBookPrice in prices)
  {
    totalAmount += singleBookPrice;
  }
  Console.WriteLine("The total amount for all books is:" +
    totalAmount);
}
```

3.4 Rules for naming a function

The name of a method must start with capital letter.
The PascalCase rule must be applied, i.e. each new word, that concatenates to form the method name, must start with capital letter.
It is recommended that the method name must consist of verb, or verb and noun.

3.5 Return Values and local variable

The simplest way to exchange data with a function is to use a return value.
Functions that have return values evaluate to that value, in exactly the same way that variables evaluate to the values they contain when you use them in expressions.
Just like variables, return values have a type.
When a function returns a value, you have to modify your function in two ways:
Specify the type of the return value in the function declaration instead of using the void keyword.

Use the return keyword to end the function execution and transfer the return value to the calling code (see sample code below).

```
static <return_type> <method_name>(<parameters_list>)
{
  // … code goes here - in the method's body …
}
```

Local Variables

Local variables are variables declared inside the body of a method.
The area where a local variable exists, and can be used, begins from the line where the variable is declared and ends at the closing curly bracket "}" of the method body.
This is the so-called area of visibility of the variable (variable scope).
If we try to declare variable, after we have already declared a variable with the same name, the code will not compile due to an error.

3.6 Invoking a function and Parameter passing

Invoking or calling a method is actually the process of execution of the method's code, placed into its body.
To invoke a method, write the method's name <method_name>, followed by the round brackets and semicolon ";" at the end: <method_name>();

Parameters

When a function is to accept parameters, you must specify the following:
A list of the parameters accepted by the function in its definition, along with the types of those parameters .
A matching list of parameters in each function call .

This involves the following code, where you can have any number of parameters, each with a type and a name:

```
static < returnType > < functionName > ( < paramType > < paramName > , ...)
        {
        ...
        return < returnValue > ;
        }
```

The parameters are separated using commas, and each of these parameters is accessible from code within the function as a variable.
For example, a simple function might take two double parameters and return their product:

```
        static double Product(double param1, double param2)
        {
        return param1 * param2;
        }
```

3.7 Program Example and Class exercise

Some examples of program codes and class exercise are described in this session.

```
class Program
{
   static void Write()
   {
      string myString = "String defined in Write()";
      Console.WriteLine("Now in Write()");
      Console.WriteLine("myString = {0}", myString);
   }

   static void Main(string[] args)
   {
      string myString = "String defined in Main()";
      Write();
      Console.WriteLine("\nNow in Main()");
      Console.WriteLine("myString = {0}", myString);
      Console.ReadKey();
   }
}
```

Another Program Example

```
class Program
{
   static int MaxValue(int[] intArray)
   {
      int maxVal = intArray[0];
      for (int i = 1; i < intArray.Length; i++)
      {
         if (intArray[i] > maxVal)
            maxVal = intArray[i];
      }
      return maxVal;
   }

   static void Main(string[] args)
   {
      int[] myArray = {1, 8, 3, 6, 2, 5, 9, 3, 0, 2};
      int maxVal = MaxValue(myArray);
      Console.WriteLine("The maximum value in myArray is {0}", maxVal);
      Console.ReadKey();
   }
}
```

Divide the above to subtasks as follow:

Function main() should
Take the temperature measured in Fahrenheit degrees as an input from the console (the user must enter it)

Print a message for the converted temperature in Celsius.
. If the temperature is found to be higher than 37 ºC, print a message that the user is ill.
Function ConvertFahrenheitToCelsius() should
Convert the entered number to its corresponding value, for temperature measured in Celsius.

4.0 CONCLUSION

To end this session, the basic unit of execution in a C# program is the *statement*. A series of statements surrounded by curly braces form a *block* of code (i.e. statement blocks).*Comments* allow inline documentation of source code. The C# compiler ignores comments. Three styles of comments are allowed in C#: single-line, multiple-line and XML documentation-line comments.

5.0 SUMMARY

This unit provided an overview of C# syntax. It equally outlined the styles of comments allowed in C#. We hope you found this unit enlightening.

6.0 TUTOR MARKED ASSIGNMENT

• State 2 categories of statements

7.0 REFERENCES/FURTHER READINGS

1. Archer, Tom (2001). *Inside C#*. Microsoft Press. ISBN 0-7356-1288-9.
2. Cornelius, Barry (December 1, 2005). "Java 5 catches up with C#". University of Oxford Computing Services.
3. Ecma International. June 2006.*C# Language Specification* (4th ed.).
4. Guthrie, Scott (November 28, 2006). "What language was ASP.Net originally written in?"
5. Hamilton, Naomi (October 1, 2008). "The A-Z of Programming Languages: C#". Computerworld.

6. Horton, Anson (2006-09-11). "C# XML documentation comments FAQ".
7. Kovacs, James (September 07, 2007). "C#/.NET History Lesson".
8. Microsoft. September 4, 2003."Visual C#.net Standard" (JPEG).
9. Microsoft (June 18, 2009) *C# Language Reference*.
10. O' Reilly. (2002).*C# Language Pocket Reference*. ISBN 0-596-00429-X.
11. Petzold, Charles (2002). *Programming Microsoft Windows with C#*. Microsoft Press. ISBN 0-7356-1370-2.
12. Steinman, Justin (November 7, 2006). "Novell Answers Questions from the Community".
13. Stallman, Richard (June 26, 2009). "Why free software shouldn't depend on Mono or C#".
14. Simon, Raphael; Stapf, Emmanuel; Meyer, Bertrand (June 2002). "Full Eiffel on the .NET Framework".
15. "The ECMA C# and CLI Standards". 2009 -07-06.
16. Zander, Jason (November 23, 2007). "Couple of Historical Facts".

MODULE 5 OBJECT ORIENTED, WINDOWS AND DATABASE PROGRAMMING

UNIT 1: Objects and Classes

7.0 CONTENT

1.0 INTRODUCTION

An object is a building block of an OOP application. This building block encapsulates part of the application, which may be a process, a chunk of data, or a more abstract entity. Objects in C# are created from types. The type of an object is known by a special name in OOP, its class

2.0 **OBJECTIVES**

-After going through this unit you would be able to:
• Explain the term 'Object Oriented Programming (OOP) principle'
• Identify the various components of OOP
• understand the concept of Inheritance
• understand the concept of Polymorphism in C#

3.0 MAIN CONTENT

3.1 Class example

The class keyword is preceded by the access level. Because public is used in this case, anyone can create objects from this class.
The name of the class follows the class keyword.
The remainder of the definition is the class body, where the behaviour and data are defined.
Fields, properties, methods, and events on a class are collectively referred to as class members.

UML representation of a class

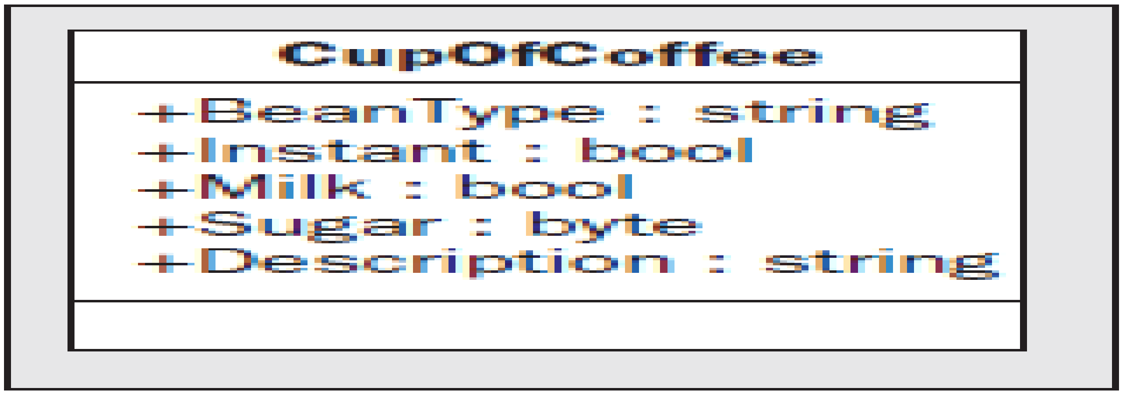


Figure 22*: UML representation of a class in C# Screenshot*

Figure 22 is a representation of the CupOfCoffee class, with five members (properties or fields, because no distinction is made in UML.
Each of the entries contains the following information:
Accessibility: A + symbol is used for a public member, a - symbol is used for a private member.
The member name .
The type of the member .

Properties and Fields

Properties and fields provide access to the data contained in an object.
Object data is what differentiates separate objects.
The various pieces of data contained in an object together make up the state of that

object.
Both fields and properties have their types.
Unlike fields, properties are not classified as variables.
You cannot pass a property as a parameter.
Properties are declared in the class block by specifying the access level of the property, followed by the type of the property, followed by the name of the property, and followed by a code block that declares a get-accessor and/or a set accessor.
Accessibility determines which code can access these members — that is, whether they are available to all code (public), only to code within the class (private), or use a more complex scheme.
One common practice is to make fields private and provide access to them via public properties.

Methods

Method is the term used to refer to functions exposed by objects.
They may be called in the same way as any other function and may use return values and parameters.
Methods provide access to the object's functionality.
Like fields and properties, they can be public or private.
They have access to private members such as private fields if required

In UML, class boxes show methods in the third section, as shown below:

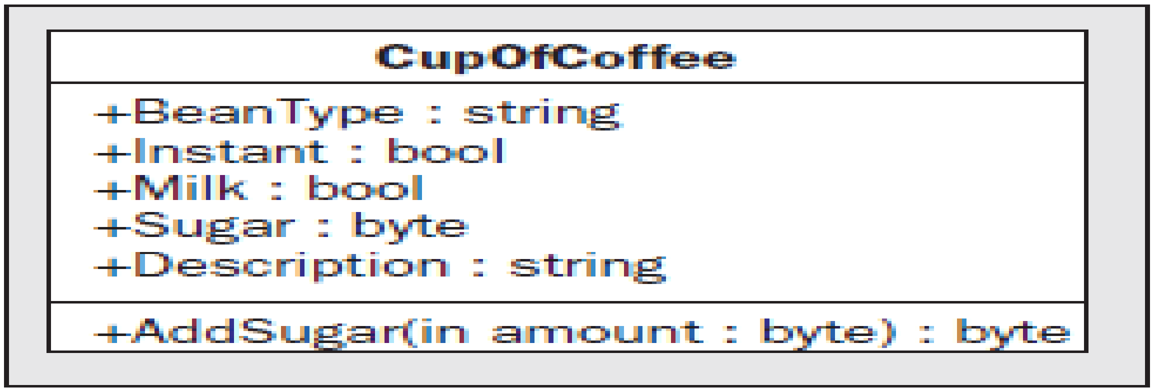


***Figure 23**: Class method representation of a class in C# Screenshot*

The type shown at the end is the return type and method parameters are shown.
Each parameter is displayed in UML with one of the following identifiers: in or out
These are used to signify the direction of data flow.
Everything is an object in C#
Every command you have used so far has been a property or a method, such as:

```
< String > .Length ,
< String >.ToUpper() , etc.
```

The period character separates the object instance ' s name from the property or method name.

3.2 Lifecycle of an object

Apart from the normal state of " being in use, " every object includes two important stages:

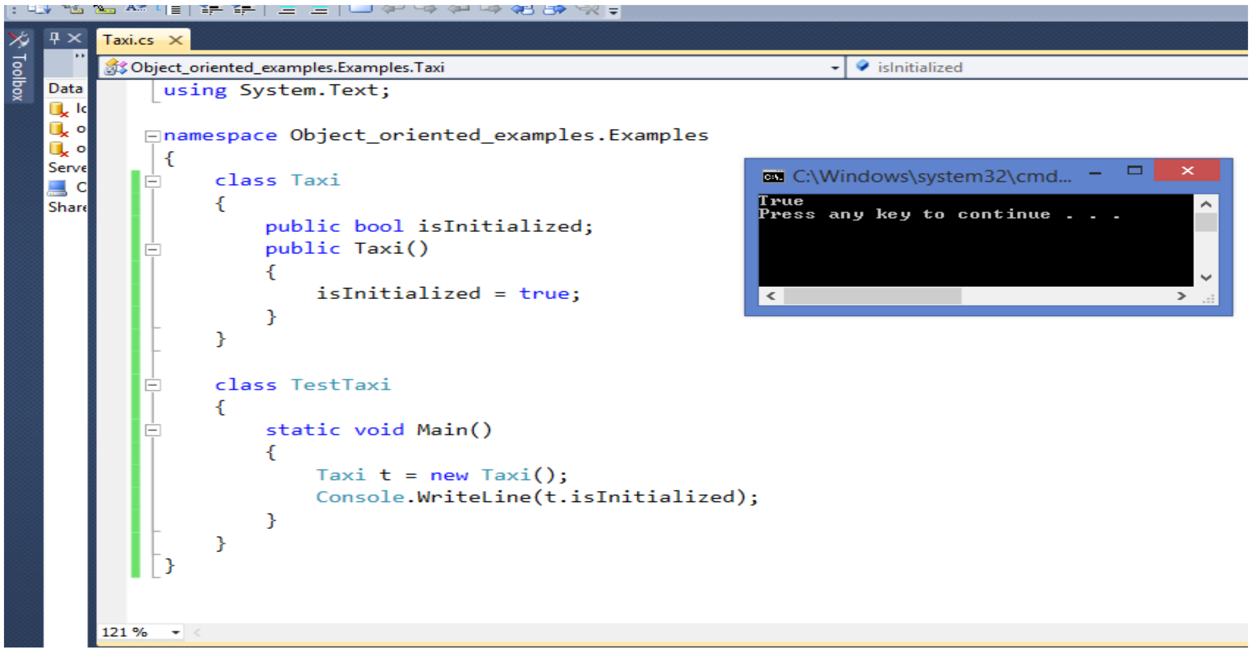
Construction: When an object is first instantiated it needs to be initialized.
This initialization is known as construction and is carried out by a constructor function.
Destruction: When an object is destroyed, there are often some clean - up tasks to perform, such as freeing memory.
This is the job of a destructor function.
A constructor is used to initialize the data by an object.
All objects have a default constructor , which is a parameterless method with the same name as the class itself.
A class definition might include several constructor methods with parameters, known as nondefault constructors.
Constructors are called using the new keyword:

```
CupOfCoffee myCup = new CupOfCoffee();
```

The line of code above instantiates a CupOfCoffee object using its default constructor.

3.3 Constructor example

Figure 24 shows an object oriented programming in C#.



```
using System.Text;

namespace Object_oriented_examples.Examples
{
    class Taxi
    {
        public bool isInitialized;
        public Taxi()
        {
            isInitialized = true;
        }
    }

    class TestTaxi
    {
        static void Main()
        {
            Taxi t = new Taxi();
            Console.WriteLine(t.isInitialized);
        }
    }
}
```

```
True
Press any key to continue . . .
```

***Figure 24**: Objects Oriented Programming in C#*

3.4 Destructors Example

Destructors are used by the .NET Framework to clean up after objects are no longer used. Generally, you don ’ t have to provide code for a destructor method; instead, the default operation does the work for you.
However, you can provide specific instructions if anything important needs to be done before the object instance is deleted. An example

```
class Car
{
   ~Car()  // destructor
   {
      // cleanup statements...
   }
}
```

3.5 Inheritance

Inheritance enables you to extend or create more specific classes from a single, more generic base class (see Figure 25).

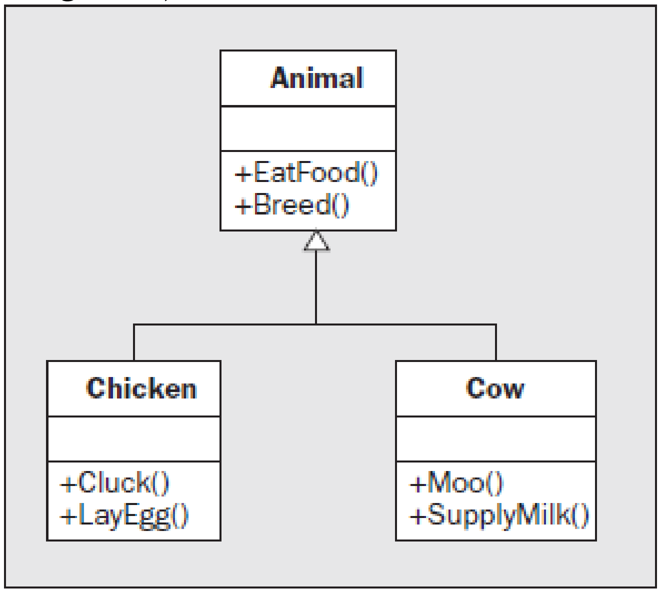


Figure 25: *Inheritance illustration in C#*

For example, from an Animal class that possesses methods such as: EatFood() or Breed(), you could create a derived class called Cow , which would support all of these methods and include others like Moo() and SupplyMilk() and others.
The parent class is called the base class, and the class that inherits from the base class is called the derived class.
C# does not support multiple inheritance though the base class can also be a derived class.

Accessibility of Members:

Private members of the base class are not accessible from a derived class, but public members are.protected keyword allows only derived classes to have access to a member. For external code, this is identical to a private member — it doesn ' t have access in either case.

The syntax used in C# for creating derived classes is as follows:

```
<acess-specifier> class <base_class>
{
   ...
}
class <derived_class> : <base_class>
{
   ...
}
```

Example of inheritance

```
class Shape
      {
         public void setWidth(int w)
         {
            width = w;
         }
         public void setHeight(int h)
         {
            height = h;
         }
         protected int width;
         protected int height;
      }
```

3.6 Polymorphism

The ability to treat an instance of derived class as an instance of the base class and being able to call the operations of the base class using an instance of the derived class is called Polymorphism.

Polymorphism means one name many forms. Polymorphism means one object behaving as multiple forms. One function behaves in different forms

Polymorphism concept illustration in C# is contained in Figure 26.

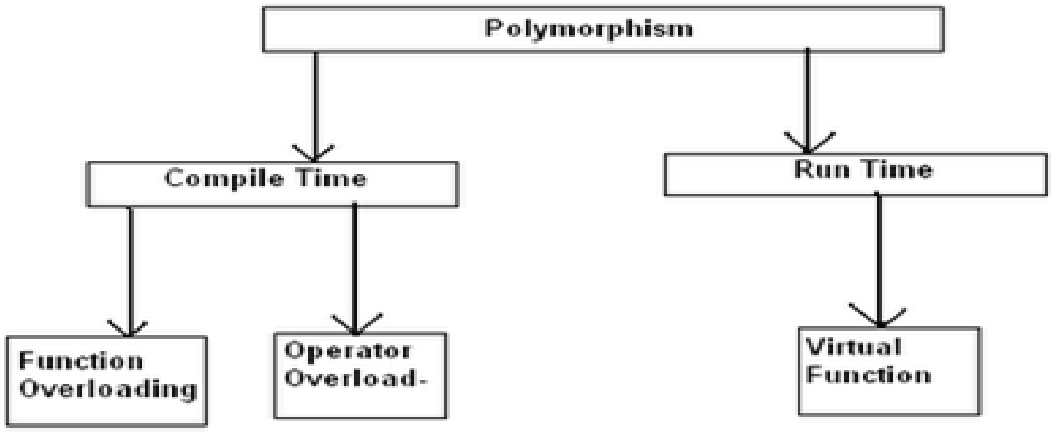


Figure 26: *Polymorphism concept illustration in C#*

Static or Compile Time Polymorphism

In static polymorphism, the decision is made at compile time.
Which method is to be called is decided at compile-time only.
Method overloading is an example of this.
Compile time polymorphism is method overloading, where the compiler knows which overloaded method it is going to call.

Dynamic or Runtime Polymorphism

Run-time polymorphism is achieved by method overriding.
Method overriding allows us to have methods in the base and derived classes with the same name and the same parameters.
By runtime polymorphism, we can point to any derived class from the object of the base class at runtime that shows the ability of runtime binding.

3.7 SELF ASSESSMENT EXERCISE

Define a class Car that contains a field make that stores the make of the car. Define a non-default constructor that initializes the make of the car when an object of the class is created. Also define a destructor that prints the message “Inside the Destructor”.
Other features of OOP are:

- Inheritance
- Polymorphism
- Relationships between objects

- Operator overloading
- Events
- Reference versus value types

4.0 CONCLUSION

To end, the basic unit of execution in a C# program is the *statement*. A series of statements surrounded by curly braces form a *block* of code (i.e. statement blocks).*Comments* allow inline documentation of source code. The C# compiler ignores comments. Three styles of comments are allowed in C#: single-line, multiple-line and XML documentation-line comments.

5.0 SUMMARY

This unit provided an overview of C# syntax. It equally outlined the styles of comments allowed in C#. We hope you found this unit enlightening.

6.0 TUTOR MARKED ASSIGNMENT

• State 2 categories of statements

7.0 REFERENCES/FURTHER READINGS

1. Archer, Tom (2001). *Inside C#*. Microsoft Press. ISBN 0-7356-1288-9.
2. Cornelius, Barry (December 1, 2005). "Java 5 catches up with C#". University of Oxford Computing Services.
3. Ecma International. June 2006.*C# Language Specification* (4th ed.).
4. Guthrie, Scott (November 28, 2006). "What language was ASP.Net originally written in?"
5. Hamilton, Naomi (October 1, 2008). "The A-Z of Programming Languages: C#". Computerworld.
6. Horton, Anson (2006-09-11). "C# XML documentation comments FAQ".
7. Kovacs, James (September 07, 2007). "C#/.NET History Lesson".
8. Microsoft. September 4, 2003."Visual C#.net Standard" (JPEG).
9. Microsoft (June 18, 2009) *C# Language Reference*.
10. O' Reilly. (2002).*C# Language Pocket Reference*. ISBN 0-596-00429-X.
11. Petzold, Charles (2002). *Programming Microsoft Windows with C#*. Microsoft Press. ISBN 0-7356-1370-2.
12. Steinman, Justin (November 7, 2006). "Novell Answers Questions from the Community".

13. Stallman, Richard (June 26, 2009). "Why free software shouldn't depend on Mono or C#".
14. Simon, Raphael; Stapf, Emmanuel; Meyer, Bertrand (June 2002). "Full Eiffel on the .NET Framework".
15. "The ECMA C# and CLI Standards". 2009 -07-06.
16. Zander, Jason (November 23, 2007). "Couple of Historical Facts".

UNIT 2: Windows Programming

1.0 INTRODUCTION

In the previous unit we examined C# syntax. Here, we will be looking at designing windows forms and developing windows applications in C#. Do make the most of your studies.

2.0 OBJECTIVES

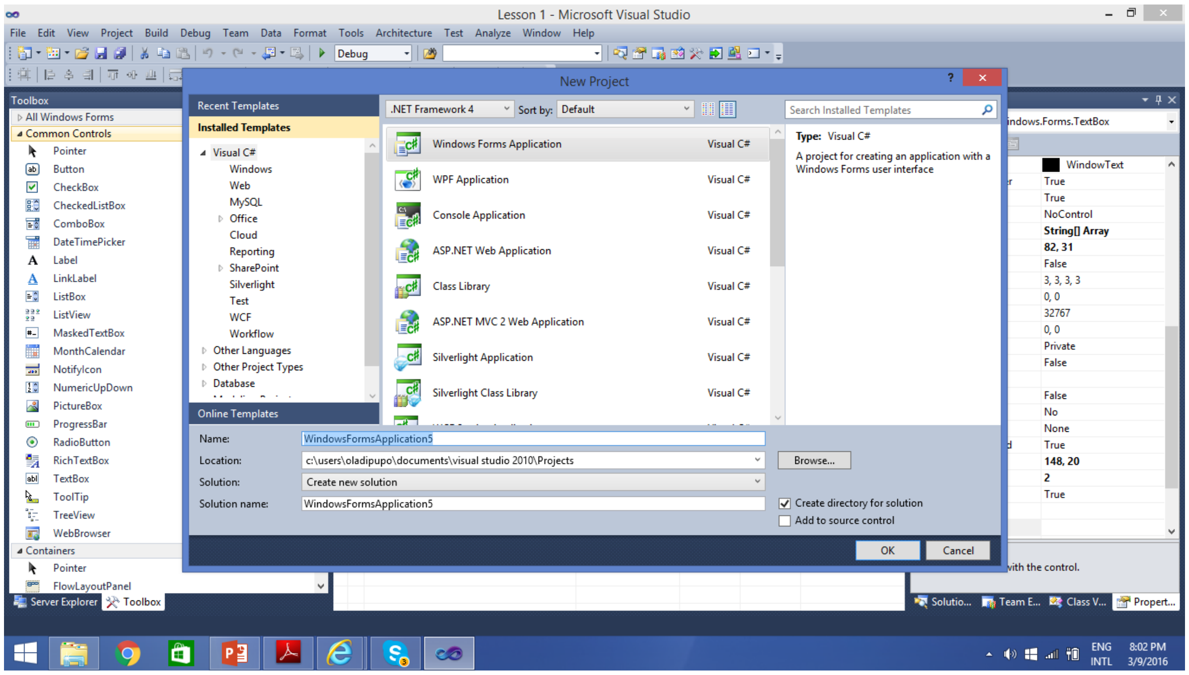
At the end of this unit, you should be able to:
• Add forms and design the GUI of application
• Add codes behind the click events of buttons
• Running the windows application

3.0 MAIN CONTENT

3.1 How to create a new Window form Application

- Click new from File
- Click project
- Click window form application
- Type the filename
- And select location or use the default (browse)
- Click ok

Figure 27 shows a sample screenshot of how to create a Window form application in C#



Type: Visual C#
A project for creating an application with a Windows Forms user interface

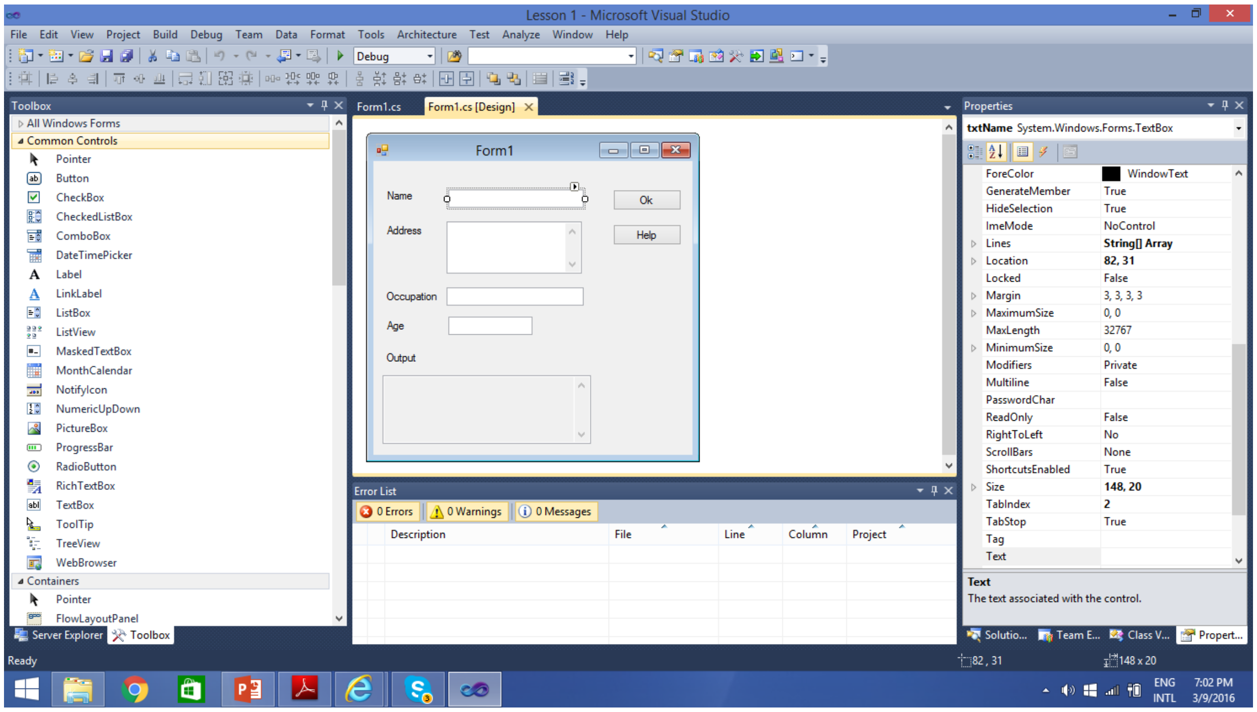
Name: WindowsFormsApplication5
Location: c:\users\oladipupo\documents\visual studio 2010\Projects
Solution: Create new solution
Solution name: WindowsFormsApplication5
[✓] Create directory for solution
[ ] Add to source control

Figure 27: *How to create a Window form application in C#*

3.2 Label and link controls

Label and link control is a control to display information to the user.
Most controls in .NET derive from the System.Windows.Forms.Control class.
This class defines the basic functionality of the controls, which is why many properties and events in the controls you ' ll see are identical.
Many of these classes are themselves base classes for other controls, as is the case with the Label and TextBox Base classes

Figure 28 shows how a screenshot of adding controls to form in C#



txtName System.Windows.Forms.TextBox

ForeColor	■ WindowText
GenerateMember	True
HideSelection	True
ImeMode	NoControl
Lines	**String[] Array**
Location	**82, 31**
Locked	False
Margin	3, 3, 3, 3
MaximumSize	0, 0
MaxLength	32767
MinimumSize	0, 0
Modifiers	Private
Multiline	False
PasswordChar	
ReadOnly	False
RightToLeft	No
ScrollBars	None
ShortcutsEnabled	True
Size	**148, 20**
TabIndex	**2**
TabStop	True
Tag	
Text	

Text
The text associated with the control.

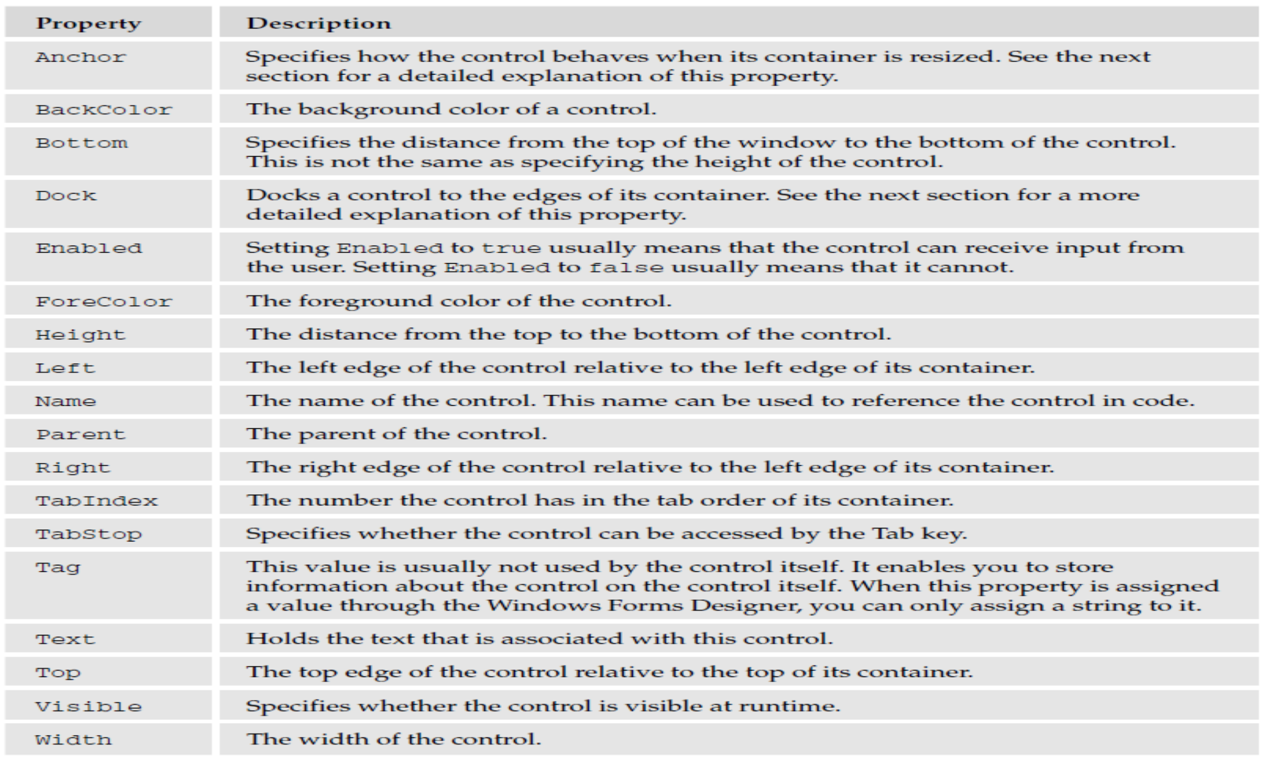
Figure 28: *Adding Controls to form in C# screenshot*

Properties

- All controls have a number of properties that are used to manipulate the behavior of the control.
- The base class of most controls, System.Windows.Forms.Control , has several properties that other
- controls either inherit directly or override to provide some kind of custom behavior.

The property and descriptions in C# are showcased in Table 6.

***Table 6**: Property and Descriptions in C#*



Property	Description
`Anchor`	Specifies how the control behaves when its container is resized. See the next section for a detailed explanation of this property.
`BackColor`	The background color of a control.
`Bottom`	Specifies the distance from the top of the window to the bottom of the control. This is not the same as specifying the height of the control.
`Dock`	Docks a control to the edges of its container. See the next section for a more detailed explanation of this property.
`Enabled`	Setting `Enabled` to `true` usually means that the control can receive input from the user. Setting `Enabled` to `false` usually means that it cannot.
`ForeColor`	The foreground color of the control.
`Height`	The distance from the top to the bottom of the control.
`Left`	The left edge of the control relative to the left edge of its container.
`Name`	The name of the control. This name can be used to reference the control in code.
`Parent`	The parent of the control.
`Right`	The right edge of the control relative to the left edge of its container.
`TabIndex`	The number the control has in the tab order of its container.
`TabStop`	Specifies whether the control can be accessed by the Tab key.
`Tag`	This value is usually not used by the control itself. It enables you to store information about the control on the control itself. When this property is assigned a value through the Windows Forms Designer, you can only assign a string to it.
`Text`	Holds the text that is associated with this control.
`Top`	The top edge of the control relative to the top of its container.
`Visible`	Specifies whether the control is visible at runtime.
`Width`	The width of the control.

Anchoring, Docking, and Snapping Controls

What is snapping control?
Create a Button control on the form (drag or double click on the control)
Move the Button to the middle of the form
Create another Button control on the form
Try to drag the 2nd Button under the 1st Button
The blue line that is controlling the alignment is called snapping control.

Anchoring and Docking

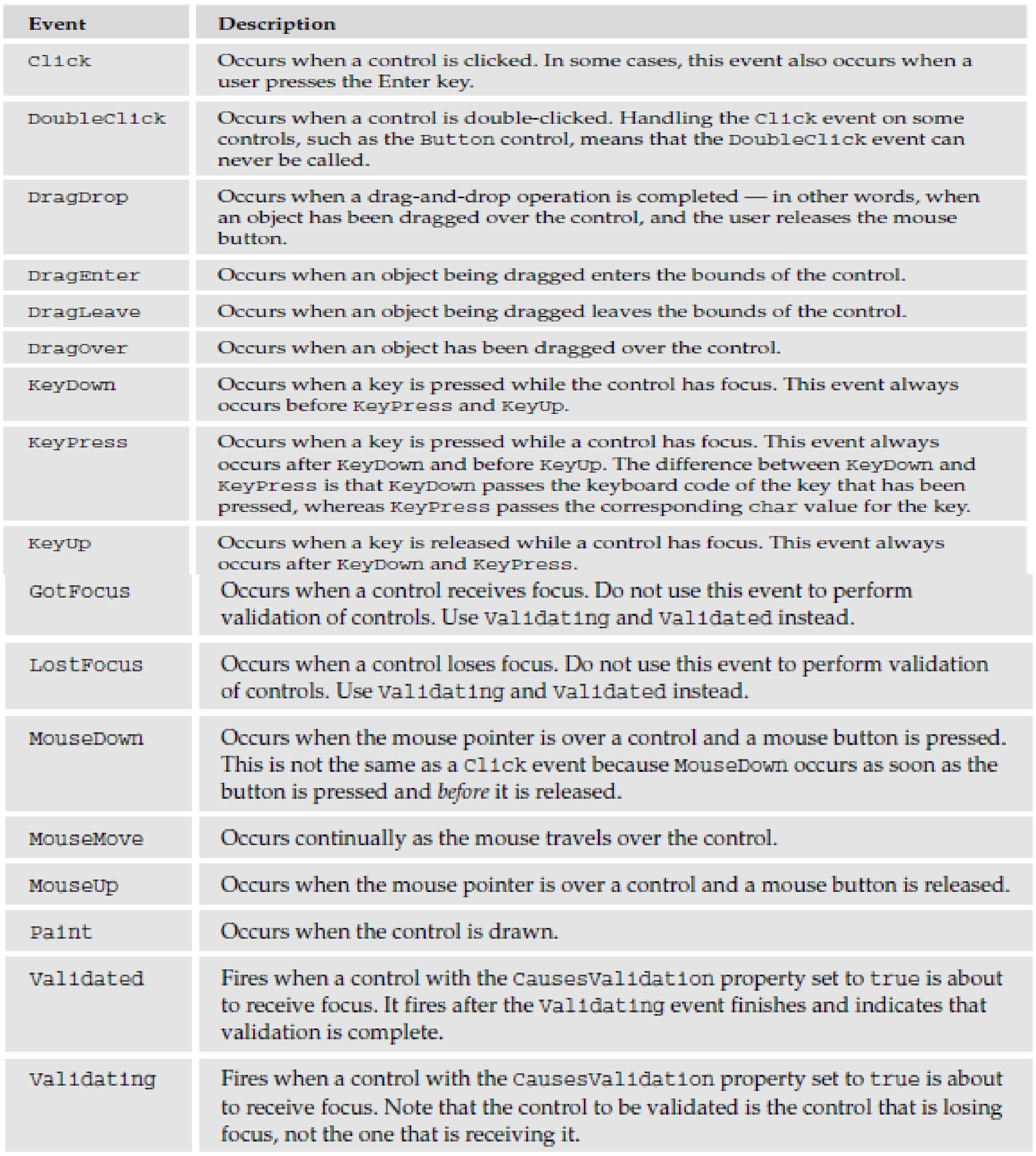
These two properties are especially useful when you are designing your form.
Ensuring appropriate adjustment of controls when a window form is resized
The Anchor property specifies how the control behaves when a user resizes the window. You can specify that the control should resize itself, anchoring itself in proportion to its own edges, or stay the same size, anchoring its position relative to the window ' s edges.

Events

These events are usually associated with user actions.
For example, when the user clicks a button, that button generates an event indicating what just happened to it.
Handling the event is the means by which the programmer can provide some functionality for that button.

Table 7 *shows a screenshot of Toolbox items and descriptions in C#.*

***Table 7**: Toolbox items and Descriptions in C#*



Event	Description
Click	Occurs when a control is clicked. In some cases, this event also occurs when a user presses the Enter key.
DoubleClick	Occurs when a control is double-clicked. Handling the Click event on some controls, such as the Button control, means that the DoubleClick event can never be called.
DragDrop	Occurs when a drag-and-drop operation is completed — in other words, when an object has been dragged over the control, and the user releases the mouse button.
DragEnter	Occurs when an object being dragged enters the bounds of the control.
DragLeave	Occurs when an object being dragged leaves the bounds of the control.
DragOver	Occurs when an object has been dragged over the control.
KeyDown	Occurs when a key is pressed while the control has focus. This event always occurs before KeyPress and KeyUp.
KeyPress	Occurs when a key is pressed while a control has focus. This event always occurs after KeyDown and before KeyUp. The difference between KeyDown and KeyPress is that KeyDown passes the keyboard code of the key that has been pressed, whereas KeyPress passes the corresponding char value for the key.
KeyUp	Occurs when a key is released while a control has focus. This event always occurs after KeyDown and KeyPress.
GotFocus	Occurs when a control receives focus. Do not use this event to perform validation of controls. Use Validating and Validated instead.
LostFocus	Occurs when a control loses focus. Do not use this event to perform validation of controls. Use Validating and Validated instead.
MouseDown	Occurs when the mouse pointer is over a control and a mouse button is pressed. This is not the same as a Click event because MouseDown occurs as soon as the button is pressed and *before* it is released.
MouseMove	Occurs continually as the mouse travels over the control.
MouseUp	Occurs when the mouse pointer is over a control and a mouse button is released.
Paint	Occurs when the control is drawn.
Validated	Fires when a control with the CausesValidation property set to true is about to receive focus. It fires after the Validating event finishes and indicates that validation is complete.
Validating	Fires when a control with the CausesValidation property set to true is about to receive focus. Note that the control to be validated is the control that is losing focus, not the one that is receiving it.

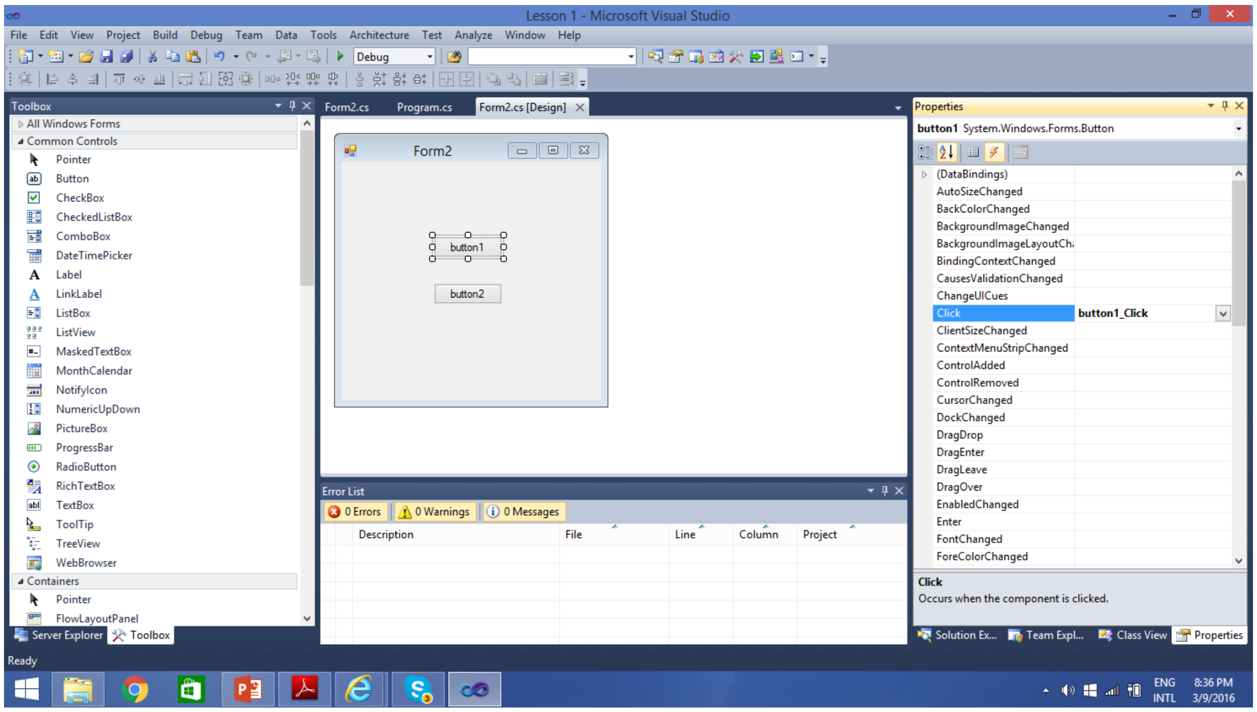
3.3 The Button Control

The Button control exists on just about any Windows dialog you can think of.
A button is primarily used to perform three kinds of tasks:
To close a dialog with a state (e.g., the OK and Cancel buttons) .

To perform an action on data entered on a dialog (e.g., clicking Search after entering some search criteria) .

To open another dialog or application (e.g., Help buttons)

The screenshot in Figure 29 shows the screenshot of adding even handler in C#


Lesson 1 - Microsoft Visual Studio
File Edit View Project Build Debug Team Data Tools Architecture Test Analyze Window Help
Toolbox
Form2.cs Program.cs Form2.cs [Design]
Form2
button1
button2
Properties
button1 System.Windows.Forms.Button
Click button1_Click
Error List
0 Errors 0 Warnings 0 Messages
Description File Line Column Project
Ready

Click
Occurs when the component is clicked.

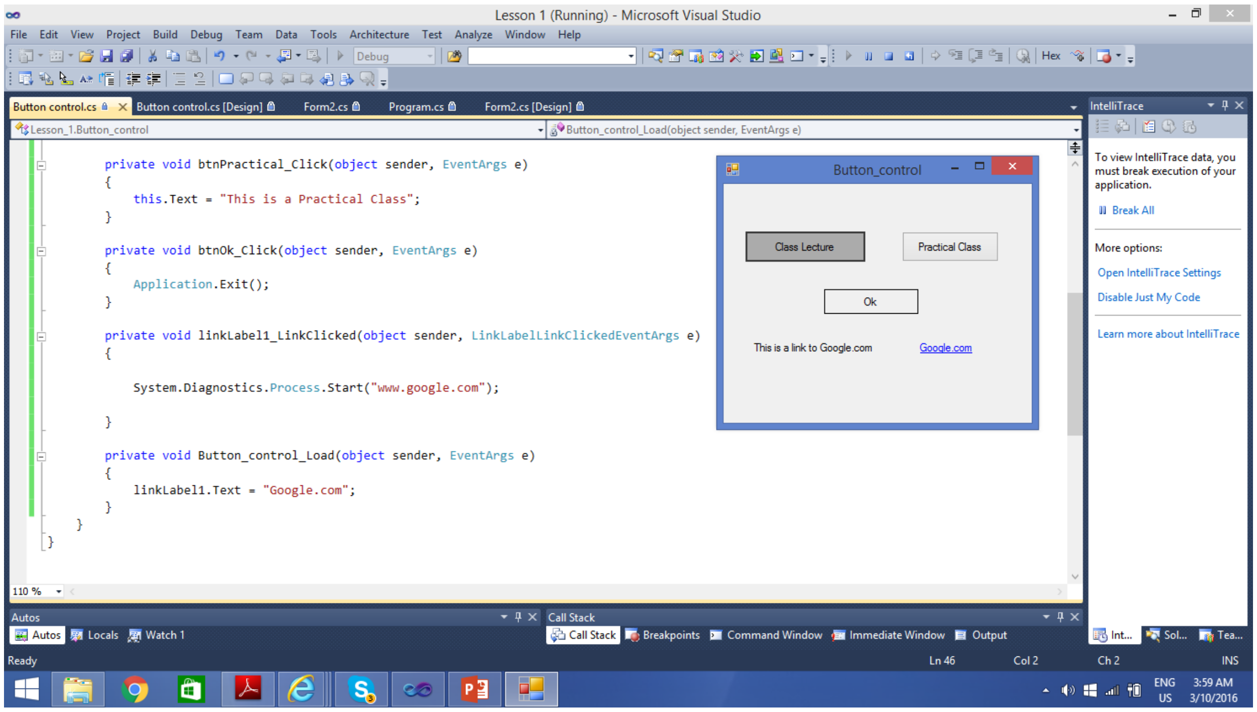
Figure 29: *Adding Even Handler in C# screenshot*

Adding the Event handlers

Try out
Double click Class Lecture button, add the following code

```
 private void btnLecture_Click(object sender, EventArgs e)
        {
            this.Text = "This is a lecture class";
        }
```

The sample running your windows application in C# screenshot is contained in Figure 30.


Lesson 1 (Running) - Microsoft Visual Studio
File Edit View Project Build Debug Team Data Tools Architecture Test Analyze Window Help
Button control.cs
Button control.cs [Design]
Form2.cs
Program.cs
Form2.cs [Design]
Lesson_1.Button_control
Button_control_Load(object sender, EventArgs e)
Button_control
Class Lecture
Practical Class
Ok
This is a link to Google.com
Google.com
IntelliTrace
Break All
More options:
Open IntelliTrace Settings
Disable Just My Code
Learn more about IntelliTrace
Autos Locals Watch 1
Call Stack Breakpoints Command Window Immediate Window Output
Ready Ln 46 Col 2 Ch 2 INS

```
        private void btnPractical_Click(object sender, EventArgs e)
        {
            this.Text = "This is a Practical Class";
        }

        private void btnOk_Click(object sender, EventArgs e)
        {
            Application.Exit();
        }

        private void linkLabel1_LinkClicked(object sender, LinkLabelLinkClickedEventArgs e)
        {

            System.Diagnostics.Process.Start("www.google.com");

        }

        private void Button_control_Load(object sender, EventArgs e)
        {
            linkLabel1.Text = "Google.com";
        }
    }
}
```

To view IntelliTrace data, you must break execution of your application.

***Figure 30:** Running your Windows application in C# screenshot*

3.4 Textbox control

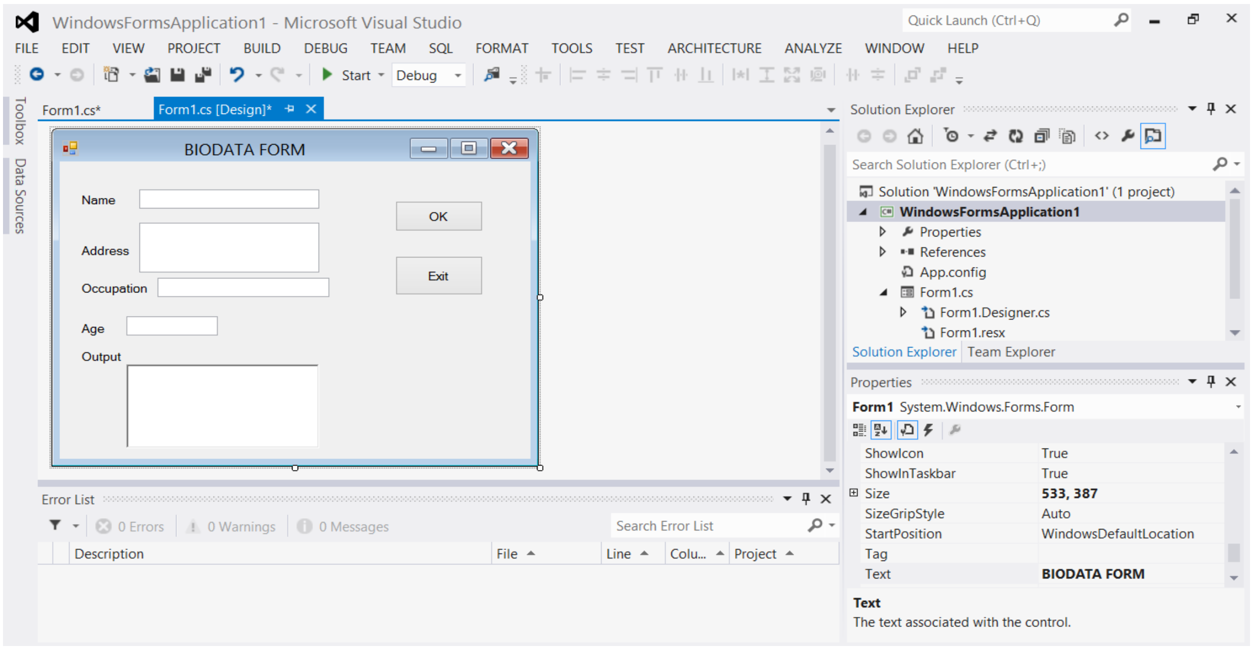
Textbox control enable users of your application to enter text, such as the TextBox control

SELF ASSESSMENT EXERCISE

1. Add a new form to the project
2. Drag 5label, 4TextBox , 1RichTextBox and 2button controls on the design form surface
3. Label1 (Text: Name); Label2(Text Address); label 3 (Name:Occupation)
4. Label4 (Text: Age) ; Label5 (Text: Output
5. Text1 (Name: txtName; Multiline :true; Anchor: Top,Left,Right)
6. Text2 (Name: txtAddress; Multiline: true; scroll: vertical ; Anchor: Top,Left,Right)
7. Text3 (Name: txtoccupation ; Anchor: Top,Left,Right)
8. Text 4 (Name: txtAge; Anchor: Top,Left,Right)

9. Text5 (Name:richTextBoxOutput)
10. Button1 (Name: btnOk; Text : OK)
11. Button2 (Name : btnExit ; Text Exit)

***Figure 31 describes the** setting properties of Windows application in C#*


WindowsFormsApplication1 - Microsoft Visual Studio
Quick Launch (Ctrl+Q)
FILE EDIT VIEW PROJECT BUILD DEBUG TEAM SQL FORMAT TOOLS TEST ARCHITECTURE ANALYZE WINDOW HELP
Start Debug
Toolbox Data Sources
Form1.cs* Form1.cs [Design]*
BIODATA FORM
Name
Address
Occupation
Age
Output
OK
Exit
Solution Explorer
Search Solution Explorer (Ctrl+;)
Solution 'WindowsFormsApplication1' (1 project)
WindowsFormsApplication1
Properties
References
App.config
Form1.cs
Form1.Designer.cs
Form1.resx
Solution Explorer Team Explorer
Error List
0 Errors 0 Warnings 0 Messages
Search Error List
Description File Line Colu... Project

Properties

Form1 System.Windows.Forms.Form

ShowIcon	True
ShowInTaskbar	True
Size	533, 387
SizeGripStyle	Auto
StartPosition	WindowsDefaultLocation
Tag	
Text	BIODATA FORM

Text
The text associated with the control.

***Figure 31:** Setting Properties of Windows application in C# screenshot*

3.5 Building and running GUI application

The screenshot in Figure 32 shows the designing Biodata Windows application in C#

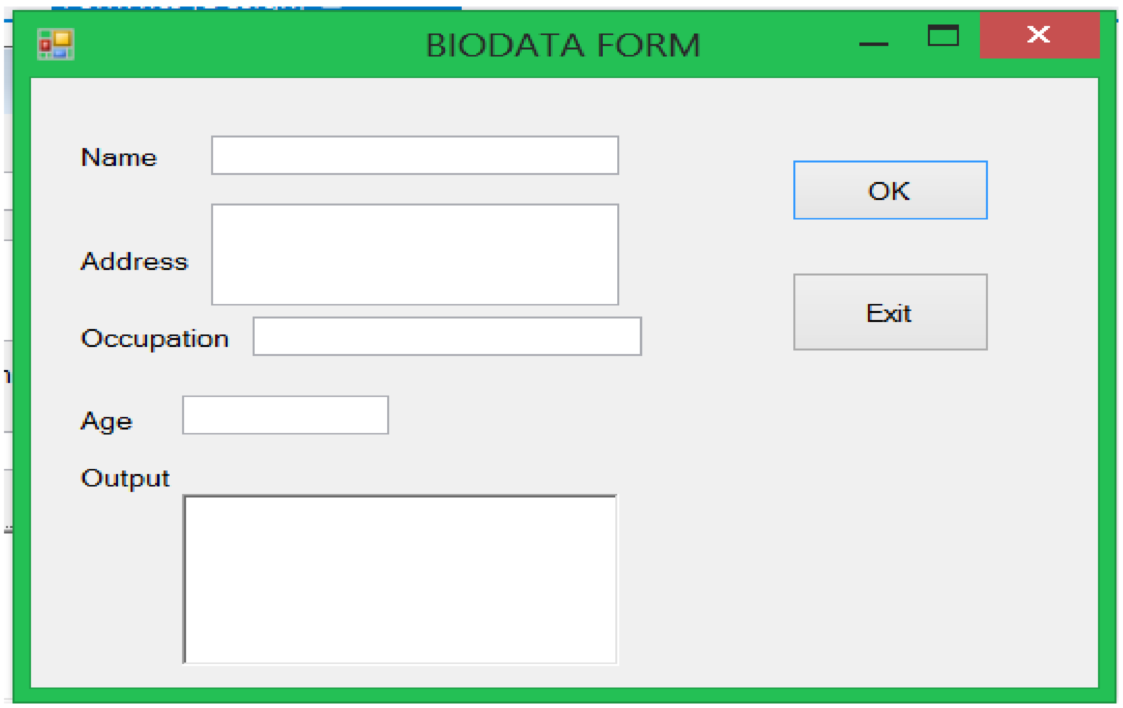


Figure 32: *Designing Biodata Windows application in C# screenshot*

Type in the code for the Ok click event handler (double click on the Ok control)
Type in the code for the Exit button
After this RUN the form and fill in the details

Click Ok observe the effect

Click Exit

The sample codes are shown below:

```
private void buttonOK_Click(object sender, EventArgs e)
{
    String name = textBoxName.Text;
    String Address = textBoxAddress.Text;
    String Occupation = textBoxOccupation.Text;
    int Age = Int32.Parse(textBoxAge.Text);
    string AgeS = Age.ToString();
    String Output;
    Output = "Name: " + name + " \r\n " + "Address: "+ Address +" \r\n " +
        "Occupation: "+ Occupation + " \r\n " + "Age: "+ AgeS;
    richTextBoxOutput.Text= Output;
}
```

Hello World Source Code

The source code in Figure 1-1 displays the text "Hello World!" in a window. (The C# version of a command line hello-world application would be a one-liner). As you can see from the code, C# has a C-based syntax, but with objects like C++ or Java. Every function in C# is a method of a type. In this example, the MyForm class is defined to Derive its functionality from the Form class (part of the .NET Framework Class Library).

Class Activity 2

```
private void buttonExit_Click(object sender, EventArgs e)
{
    this.Close();
}
```

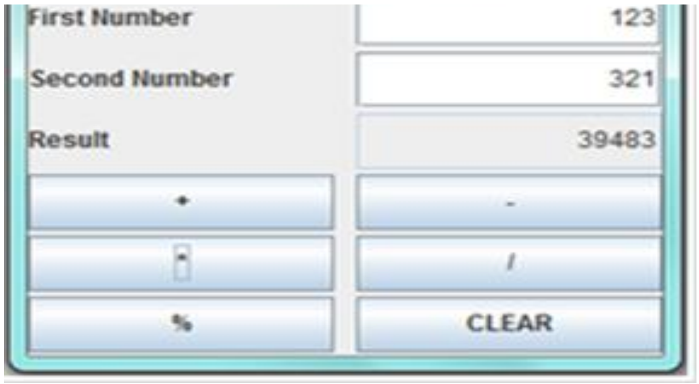


Figure 33: *Designing Calculator application in C#*

4.0 CONCLUSION

In conclusion, we able to learn how to create a new windows application , design forms and add codes behind the click events of some buttons.

5.0 SUMMARY

We considered building a GUI based windows application in this unit with the knowledge gained.

6.0 TUTOR MARKED ASSIGNMENT

▪ Outline the steps involved in building and running GUI application.

7.0 REFERENCES/FURTHER READINGS

These texts will be of enormous benefit to you in learning this course:

1. Abelson, H and Gerald J. S. (1997). Structure and Interpretation of Computer Programs. The MIT Press.
2. Armstrong, Deborah J. (2006). "The Quarks of Object-Oriented Development". Communications of the ACM 49 (2): 123–128.
http://portal.acm.org/citation.cfm?id=1113040. Retrieved 2006-08-08.
3. Booch, Grady (1997). Object-Oriented Analysis and Design with Applications. Addison-Wesley.
4. Date, C. J and Hugh, D. (2006). Foundation for Future Database Systems: The Third Manifesto (2nd Edition)
5. Date, C. J and Hugh, D. (2007). Introduction to Database Systems: The Sixth Manifesto (6th Edition)
6. Eeles, P and Oliver, S. (1998). Building Business Objects. John Wiley & Sons.
7. Gamma, Erich; Richard Helm, Ralph Johnson, John Vlissides (1995). Design Patterns: Elements of Reusable Object Oriented Software. Addison-Wesley.
8. Harmon, Paul; William Morrissey (1996). The Object Technology Casebook - Lessons from Award-Winning Business Applications. John Wiley & Sons.
9. Jacobson, Ivar (1992). Object-Oriented Software Engineering: A Use Case-Driven Approach. Addison-Wesley.
10. John C. Mitchell, Concepts in programming languages, Cambridge University Press, 2003, p.278
11. Joyce, F. (2006). Microsoft Visual C#.NET with Visual Studio 2005
12. Kay, Alan. The Early History of Smalltalk.
http://gagne.homedns.org/%7etgagne/contrib/EarlyHistoryST.html.
13. Martin, A and Luca, C. (2005). A Theory of Objects.
14. Meyer, Bertrand (1997). Object-Oriented Software Construction. Prentice Hall.
15. Michael Lee Scott (2006). Programming language pragmatics, (2nd Edition) p. 470
CIT 351 C# PROGRAMMING
ix
16.Pierce, Benjamin (2002). Types and Programming Languages. MIT Press.
17.Rumbaugh, James; Michael Blaha, William Premerlani, Frederick Eddy, William Lorensen (1991). Object-Oriented Modeling and Design. Prentice Hall.

18.Schreiner, A. (1993). Object oriented programming with ANSI-C.
19. Taylor, David A. (1992). Object-Oriented Information Systems - Planning and Implementation. John Wiley & Sons.
20. Trofimov, M. (1993) OOOP - The Third "O" Solution: Open OOP. First Class, OMG, Vol. 3, issue 3, p.14.

UNIT 3: FILE AND DATABASE CONNECTIONS

1.0 INTRODUCTION

In the previous modules, we discussed mainly the theoretical aspects of C# programming. This unit describes the process of creating console assemblies.

2.0 **OBJECTIVES**

What you would study in this unit, would enable you to:
- Explain the concept of database driven application
- Demonstrate the use of storing information into an external text file
- Give a classic example of a database driven application
- Outline the steps required to build and run a database drivene application

3.0 MAIN CONTENT

3.1 File Connections

Files can be a great way to store data between instances of your application, or they can be used to transfer data between applications.
All input and output in the .NET Framework involves the use of streams
A stream is an abstract representation of a serial device A serial device is something that stores data in a linear manner and is accessed the same way: one byte at a time.

Types of Streams

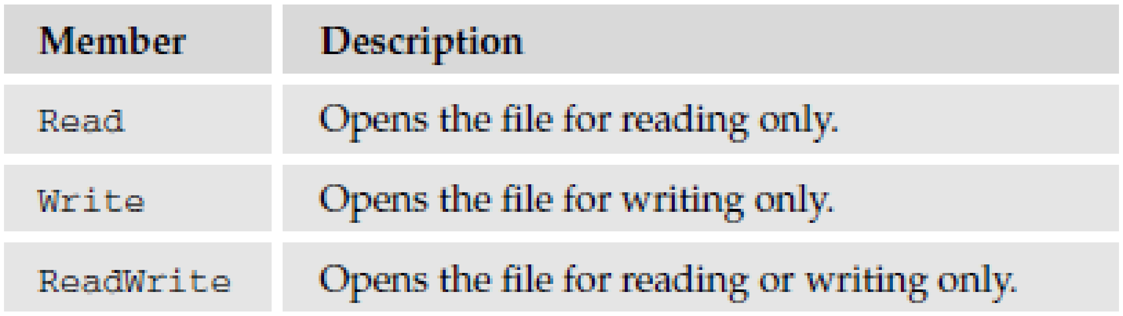
Output:

Output streams are used when data is written to some external destination, which can be a physical disk file, a network location, a printer, or another program.
Input:
Input streams are used to read data into memory or variables that your program can access. The most common form of input stream you have worked with so far is the keyboard.
An input stream can come from almost any source eg reading disk files.

The FileStream Object

The members of the FileAccess enumeration are shown in the Table 8:

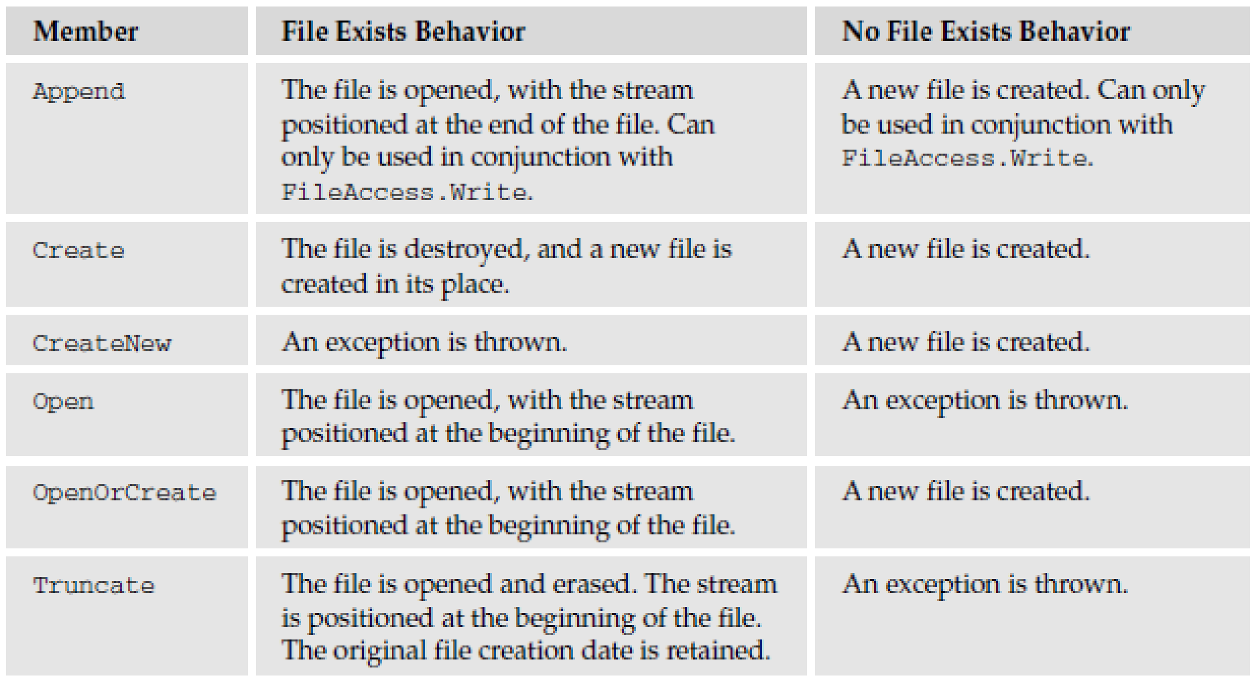
Table 8*: The FileMode enumeration members in C#*



Member	Description
`Read`	Opens the file for reading only.
`Write`	Opens the file for writing only.
`ReadWrite`	Opens the file for reading or writing only.

Table 9 contains files members and descriptions in C#.

***Table 9**: File Members and Descriptions in C#*



Member	File Exists Behavior	No File Exists Behavior
`Append`	The file is opened, with the stream positioned at the end of the file. Can only be used in conjunction with `FileAccess.Write`.	A new file is created. Can only be used in conjunction with `FileAccess.Write`.
`Create`	The file is destroyed, and a new file is created in its place.	A new file is created.
`CreateNew`	An exception is thrown.	A new file is created.
`Open`	The file is opened, with the stream positioned at the beginning of the file.	An exception is thrown.
`OpenOrCreate`	The file is opened, with the stream positioned at the beginning of the file.	A new file is created.
`Truncate`	The file is opened and erased. The stream is positioned at the beginning of the file. The original file creation date is retained.	An exception is thrown.

Class Assignments

- Write a program in C# Sharp to create a blank file in the disk newly.
- Write a program in C# Sharp to create a file and add some text.
- Write a program in C# Sharp to read a file and display the contents on the screen.

3.2 Database Connections

MySQL offers a easy connector that will allow you to execute queries to phpmyadmin from your windows form.
Requirements
MySQL .NET Connector Extension.
Appserv(phpmyadmin)

3.3 Building and running a database application

Steps

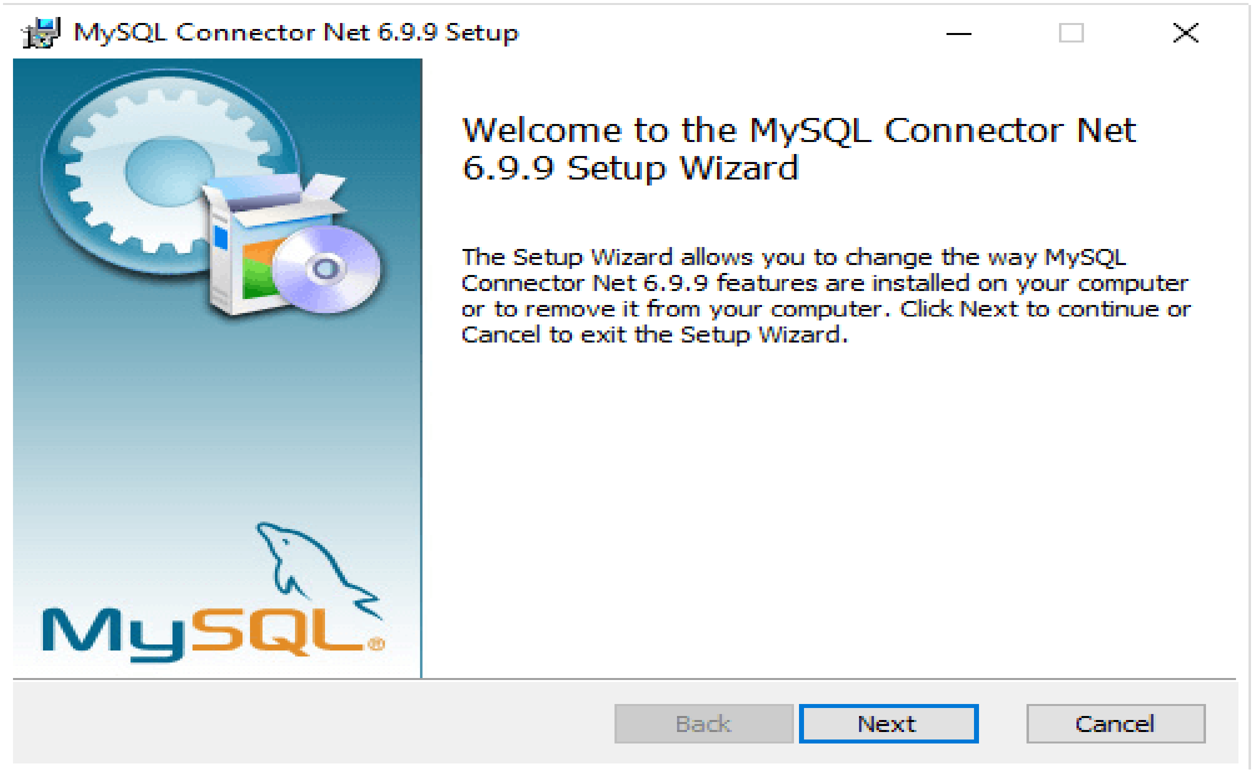
- Install mysql "using appserv-win32-2.5.10"
- Server name should be "localhost"
- Take note of the username "root"

- Take note of the password.
- Create a table in mySql looking like this
- Open mySQL through the browser using http://localhost/phpmyadmin/
- Create a database &Give the database name "csharpapp"
- Create a table & give the name "verification" like this;
- Insert a record into the table by clicking on insert and browse to view it.

Install MySql Connector

mysql-connector-net-6.3.0

The screenshot is shown in Figure 34 including the MySQL Connector installation in C#



Welcome to the MySQL Connector Net 6.9.9 Setup Wizard

The Setup Wizard allows you to change the way MySQL Connector Net 6.9.9 features are installed on your computer or to remove it from your computer. Click Next to continue or Cancel to exit the Setup Wizard.

Figure 34: *MySQL Connector installation in C# screenshot*

Designing the Form

Create a new project in windows application named "csharpapp".
Create a form looking like the Figure 34.

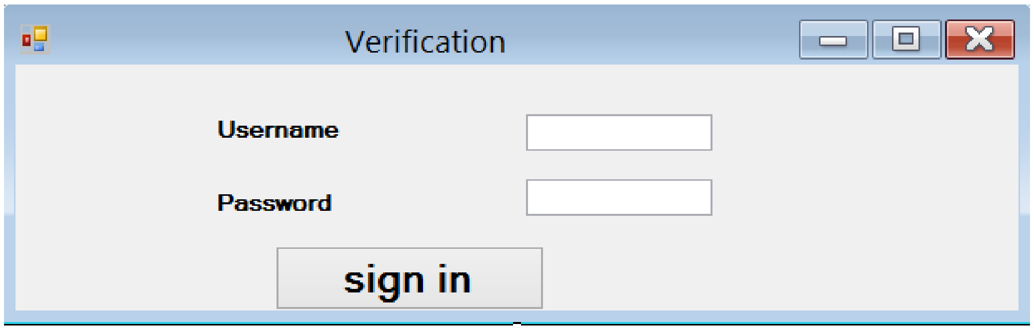


Figure 34: *Designing Login application in C# screenshot*

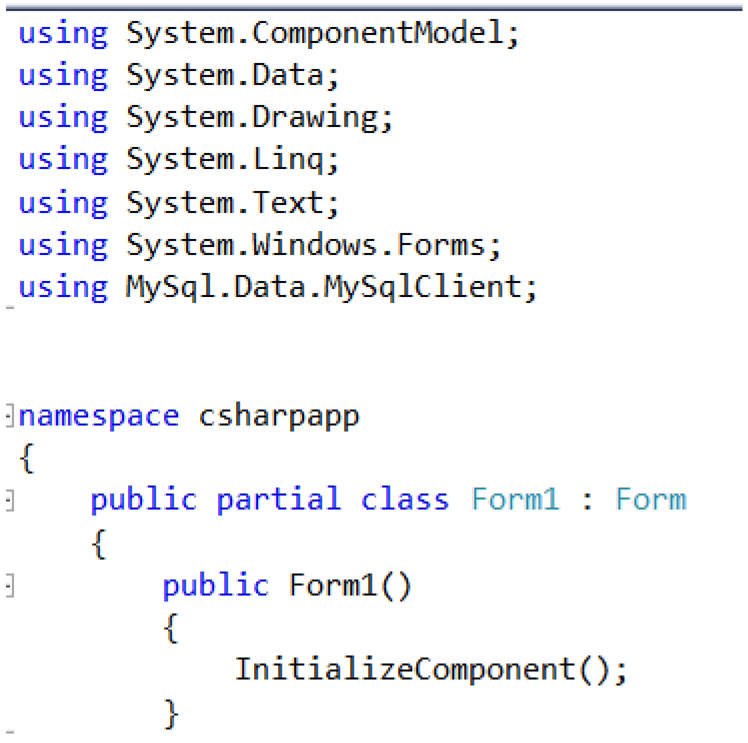
Add mysql-connector-net-6.3.0

Locate the solution explorer in the right top corner of Visual Studio when your project is open, use right click on References and then select Add Reference from the context menu.
In the shown menu, navigate to Extensions and select the checkbox from the list the MySql.Data (MySql.Data.dll) and then click on OK.

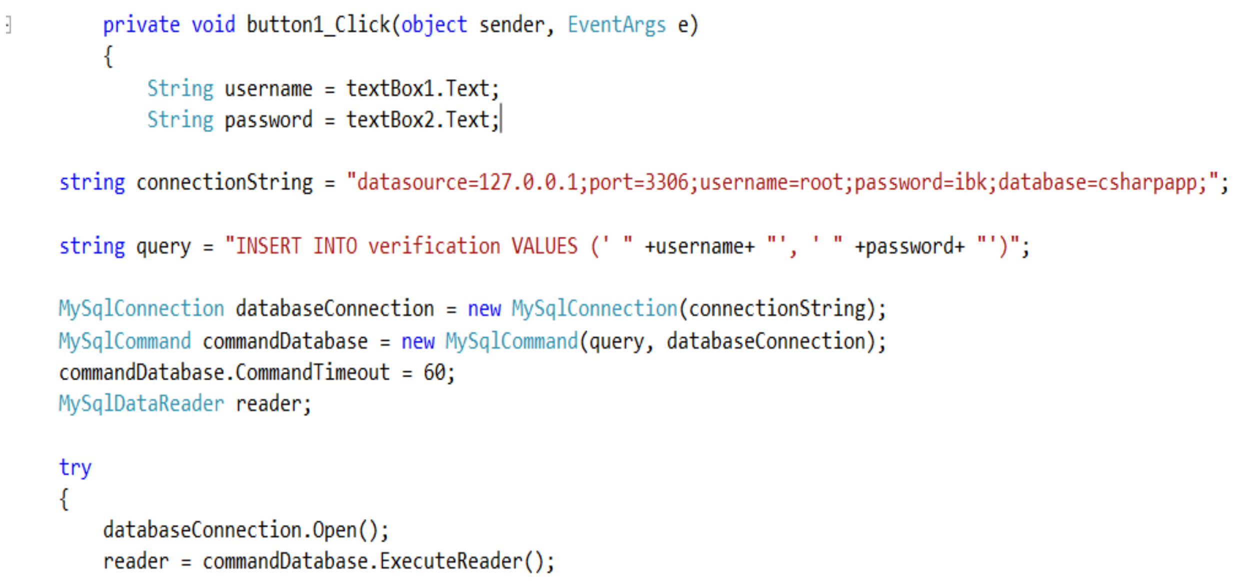
Using C# to connect and execute queries
Add the using statement of the reference in your class :

```
using MySql.Data.MySqlClient;
```

Attach the following code to the button (sign in)



```
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using MySql.Data.MySqlClient;


namespace csharpapp
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
```



```
        private void button1_Click(object sender, EventArgs e)
        {
            String username = textBox1.Text;
            String password = textBox2.Text;

    string connectionString = "datasource=127.0.0.1;port=3306;username=root;password=ibk;database=csharpapp;";

    string query = "INSERT INTO verification VALUES (' " +username+ "', ' " +password+ "')";

    MySqlConnection databaseConnection = new MySqlConnection(connectionString);
    MySqlCommand commandDatabase = new MySqlCommand(query, databaseConnection);
    commandDatabase.CommandTimeout = 60;
    MySqlDataReader reader;

    try
    {
        databaseConnection.Open();
        reader = commandDatabase.ExecuteReader();
```

```
                // Succesfully updated

                databaseConnection.Close();
            }
            catch (Exception ex)
            {
                // Ops, maybe the id doesn't exists
                MessageBox.Show(ex.Message);
            }

                }
            }
    }
```

Testing the application (see Figure 35).

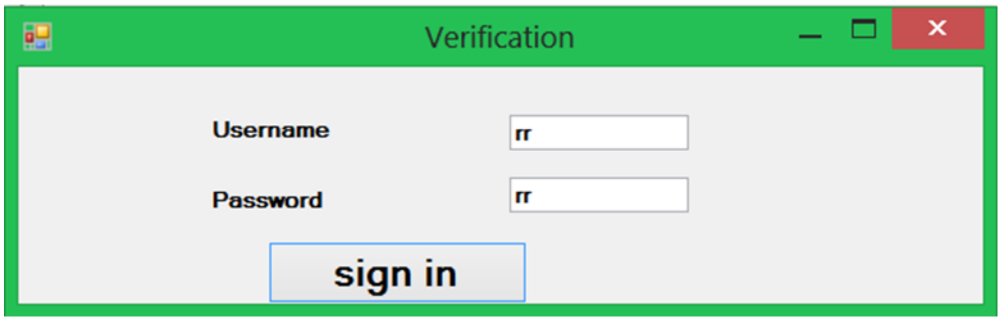
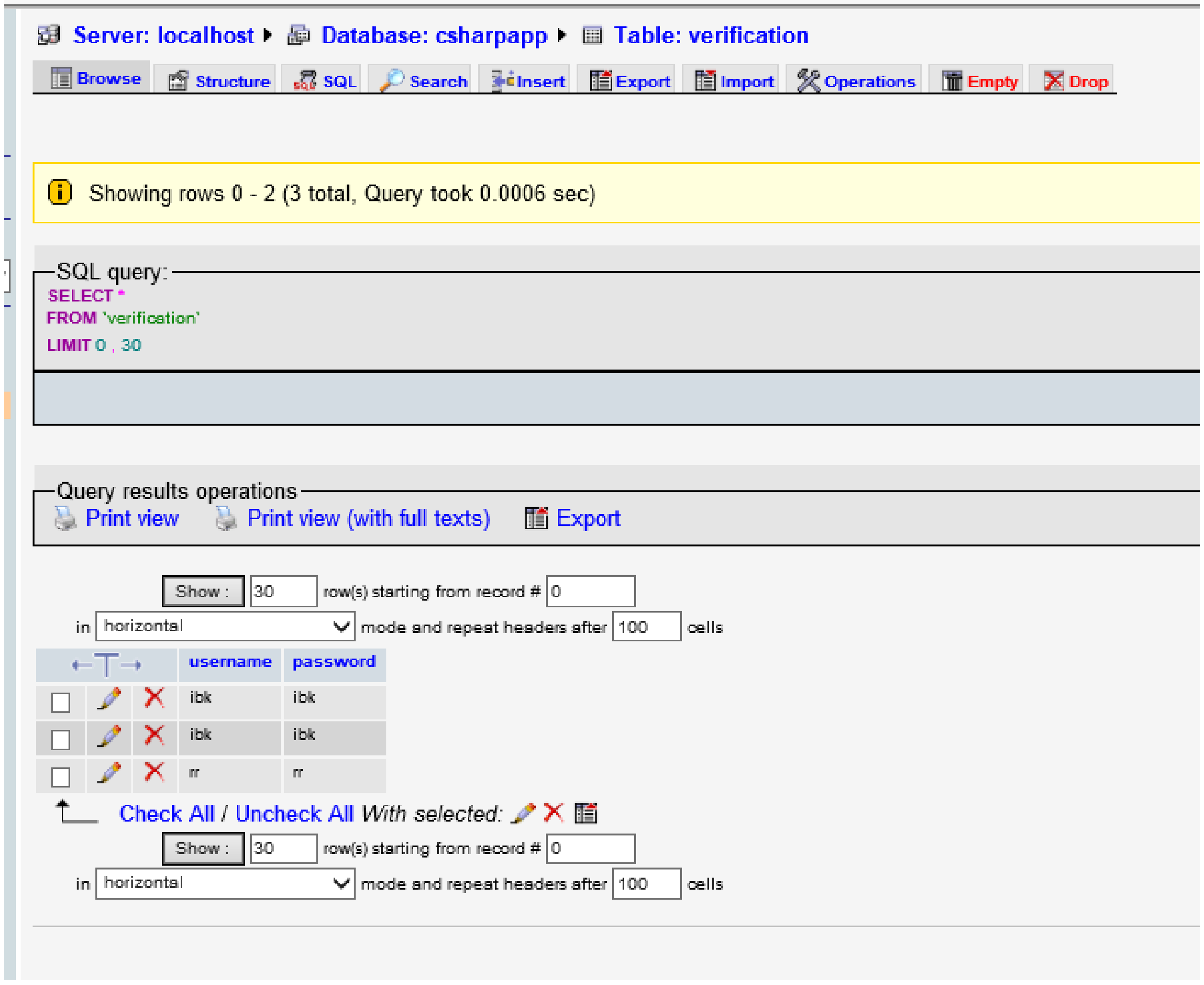


Figure 35: *Running Login application in C# screenshot*

MySql Database localhost screenshot is highlighted in Figure 36.



Showing rows 0 - 2 (3 total, Query took 0.0006 sec)

SQL query:

```
SELECT *
FROM 'verification'
LIMIT 0 , 30
```

Show : 30 row(s) starting from record # 0

in horizontal mode and repeat headers after 100 cells

	username	password
	ibk	ibk
	ibk	ibk
	rr	rr

Show : 30 row(s) starting from record # 0

in horizontal mode and repeat headers after 100 cells

Figure 36: *MySql Database localhost screenshot*

4.0 CONCLUSION

In this unit, we learnt that Console applications are the kind of executables that you would use to create a command line utility. The command line utility could be **dir** or **xcopy**. We equally identified the steps required to build and run a console application.

5.0 SUMMARY

In this unit, we considered the concept of console application, the types of command line utility and a classic example of a console application. We equally identified the steps required to build and run a console application. Let us attempt the question below.

6.0 TUTOR MARKED ASSIGNMENT

Outline the procedure for building and running a console application.

7.0 REFERENCES/FURTHER READINGS

1. Archer, Tom (2001). *Inside C#*. Microsoft Press. ISBN 0-7356-1288-9.
2. Cornelius, Barry (December 1, 2005). "Java 5 catches up with C#". University of Oxford Computing Services.
3. Ecma International. June 2006.*C# Language Specification* (4th ed.).
4. Guthrie, Scott (November 28, 2006). "What language was ASP.Net originally written in?"
5. Hamilton, Naomi (October 1, 2008). "The A-Z of Programming Languages: C#". Computerworld.
6. Horton, Anson (2006-09-11). "C# XML documentation comments FAQ".
7. Kovacs, James (September 07, 2007). "C#/.NET History Lesson".
8. Microsoft. September 4, 2003."Visual C#.net Standard" (JPEG).
9. Microsoft (June 18, 2009) *C# Language Reference*.
10. O' Reilly. (2002).*C# Language Pocket Reference*. ISBN 0-596-00429-X.
11. Petzold, Charles (2002). *Programming Microsoft Windows with C#*. Microsoft Press. ISBN 0-7356-1370-2.
12. Steinman, Justin (November 7, 2006). "Novell Answers Questions from the Community".
13. Stallman, Richard (June 26, 2009). "Why free software shouldn't depend on Mono or C#".
14. Simon, Raphael; Stapf, Emmanuel; Meyer, Bertrand (June 2002). "Full Eiffel on the .NET Framework".
15. "The ECMA C# and CLI Standards". 2009 -07-06.
16. Zander, Jason (November 23, 2007). "Couple of Historical Facts".