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

First Printed --------

ISBN:

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

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 5 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.

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#

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. 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

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.

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**
Assignments 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	**Fundamentals of C#**		
Unit 1	Introduction to C#	Week 1	Assignment 1
Unit 2	ECMA	Week 2	Assignment 2
Unit 3	C# and .NET Framework	Week 3	Assignment 3
Module 2	**C# Types**		
Unit 1	C# Type System	Week 4	Assignment 4
Unit 2	Essential Concept in C# Type System	Week 5	Assignment 5
Unit 3	C# Data Types	Week 6	Assignment 7
Module 3	**Language Basics**		
Unit 1	Naming Conventions	Week7	Assignment 7
Unit 2	C# Syntax	Week 8	Assignment 8
Unit 3	Getting Started with C#	Week 9	Assignment 9
Module 4	**C# Applications**		
Unit 1	Creating Console Assemblies	Week 10	Assignment 10
Unit 2	Creating GUI Assemblies	Week 11	Assignment 11
Unit 3	Creating Code Library Assemblies	Week 12	Assignment 12
Module 5	**Visual Studio.NET**		
Unit 1	Creating a Project	Week 13	Assignment 13
Unit 2	Language Concepts	Week 14	Assignment 14
Unit 3	C# Expressions and Operators	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!

MAIN COURSE

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

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

First Printed 2009

ISBN:

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

TABLE OF CONTENTS PAGE

MODULE 1 C# FUNDAMENTALS

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 combines some of the best features of modern programming languages such as Java, C++ or Visual Basic. 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 programming languages designed for the **Common Language Infrastructure**. 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.

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**

via the **ECMA**. 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

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.
- 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.
- 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

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#.

3.2.1 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. At the time of this writing the .NET Framework runs on Windows operating systems, 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.

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.

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 C# TYPES

Unit 1 C# Type System
Unit 2 Essential Concept in C# Type System
Unit 3 C# Datatypes

UNIT 1 C# Type System

CONTENTS

1.0 INTRODUCTION

Types are at the heart of C# programming. C# borrows a little from C++ and Java and adds some ingenuity to create elegant solutions to old problems. In this unit, we'll consider types in general and C# type system specifically.

2.0 OBJECTIVES

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

- Define a type
- Identify the categories of C# type system
- Distinguish between value types and reference types

3.0 MAIN CONTENT

3.1 What is a Type?

A **Type** is defined as a set of data and the operations performed on them. CSharp is a strongly typed language.

SELF ASSESSMENT EXERCISE

Give a concise definition of a Type.

3.2 C# Type System

The CSharp type system contains three Types, they are: ***Value Types, Reference Types and Pointer Types***.

3.2.1 Value Types

The Value Types store the data and are derived from ***System.ValueType***.

3.2.2 Reference Types

The Reference Types store references to the actual data and are derived from ***System.Object***.

3.2.3 Pointer Types

Pointer Types variable are used only in unsafe modes.

3.3 Difference between Value Types and Reference Types

The main difference between Value Types and Reference Types is how these Types store the values in memory. Common Language Runtime (CLR) allocates memory in Stack and the Heap. A Value Type holds its

actual value in memory allocated on the Stack and Reference Types referred to as objects, store references to the actual data.

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".

UNIT 2 ESSENTIAL CONCEPT IN C# TYPE SYSTEM

CONTENTS

1.0 INTRODUCTION

An essential concept in C# type system is the concept of Boxing and Unboxing. This unit sheds more light on this.

2.0 OBJECTIVES

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

- Explain the concept of ‘boxing’
- Describe the process of ‘unboxing’
- State the code for calling the object-defined
- Identify the code for converting an int value
- Discover the implication of a boxing conversion
- Outline the steps involved in unboxing conversion

3.0 MAIN CONTENT

3.1 Boxing and Unboxing

Boxing and unboxing is an essential concept in C#'s type system. *Boxing* is the operation of converting a value of a value type into a value of a corresponding reference type. Boxing in C# is implicit. *Unboxing* is the operation of converting a value of a reference type (previously boxed) into a value of a value type. Unboxing in C# requires an explicit type cast.

With Boxing and unboxing one can link between value-types and reference-types by allowing any value of a value-type to be converted to and from type object. Boxing and unboxing enables a unified view of the type system wherein a value of any type can ultimately be treated as an object. Converting a value type to reference type is called Boxing. Unboxing is an explicit operation.

SELF ASSESSMENT EXERCISE

State 2 applications of boxing and unboxing?

_____
_____
_____

3.2 Calling Object Methods

C# provides a unified type system. All types including value types derive from the type object. It is possible to call object methods on any value, even values of primitive types such as int.

3.2.1 Calling the Object-defined

Example:

```
using System;
class Test
{
   static void Main() {
      Console.WriteLine(3.ToString());
   }
}
```

calls the object-defined ToString method on an integer literal.

3.2.2 Converting an Int Value

Example:

```
class Test
{
   static void Main() {
      int i = 1;
      object o = i;        // boxing
      int j = (int) o;    // unboxing
   }
}
```

An int value can be converted to object and back again to int.

This example shows both boxing and unboxing. When a variable of a value type needs to be converted to a reference type, an object box is allocated to hold the value, and the value is copied into the box.

Unboxing is just the opposite. When an object box is cast back to its original value type, the value is copied out of the box and into the appropriate storage location.

3.3 Boxing Conversions

A boxing conversion permits any value-type to be implicitly converted to the type object or to any interface-type implemented by the value-type. Boxing a value of a value-type consists of allocating an object instance and copying the value-type value into that instance.

For example any value-type G, the boxing class would be declared as follows:

```
class vBox
{
   G value;
   G_Box(G g) {
      value = g;
   }
}
```

Boxing of a value v of type G now consists of executing the expression new G_Box(v), and returning the resulting instance as a value of type object. Thus, the statements:

```
int i = 12;
object box = i;
```

conceptually correspond to:

```
int i = 12;
object box = new int_Box(i);
```

Boxing classes like G_Box and int_Box above don't actually exist and the dynamic type of a boxed value isn't actually a class type. Instead, a boxed value of type G has the dynamic type G, and a dynamic type check using the is operator can simply reference type G. For example:

```
int i = 12;
object box = i;
if (box is int) {
   Console.Write("Box contains an int");
}
```

will output the string Box contains an int on the console.

A boxing conversion implies making a copy of the value being boxed. This is different from a conversion of a reference-type to type object, in which the value continues to reference the same instance and simply is regarded as the less derived type object.

For example, given the declaration:

```
struct Point
{
   public int x, y;
   public Point(int x, int y) {
      this.x = x;
      this.y = y;
   }
}
```

the following statements:

```
Point p = new Point(10, 10);
object box = p;
p.x = 20;
Console.Write(((Point)box).x);
```

will output the value 10 on the console because the implicit boxing operation that occurs in the assignment of p to box causes the value of p to be copied. Had Point instead been declared a class, the value 20 would be output because p and box would reference the same instance.

3.4 Unboxing Conversions

An unboxing conversion permits an explicit conversion from type object to any value-type or from any interface-type to any value-type that implements the interface-type. An unboxing operation consists of first checking that the object instance is a boxed value of the given value-type, and then copying the value out of the instance. unboxing conversion of an object box to a value-type G consists of executing the expression ((G_Box)box).value.

Thus, the statements:
object box = 12;
int i = (int)box;

conceptually correspond to:
object box = new int_Box(12);
int i = ((int_Box)box).value;

For an unboxing conversion to a given value-type to succeed at run-time, the value of the source argument must be a reference to an object that was previously created by boxing a value of that value-type. If the source argument is null or a reference to an incompatible object, an InvalidCastException is thrown.

4.0 CONCLUSION

Boxing is the operation of converting a value of a value type into a value of a corresponding reference type. It is implicit. *Unboxing* is the operation of converting a value of a reference type (previously boxed) into a value of a value type. Unboxing in C# requires an explicit type cast.

5.0 SUMMARY

In summary, this unit looked an essential concept of C# type system-boxing and unboxing. We equally identified the relevance of boxing and unboxing conversions. We can now attempt the questions below.

6.0 TUTOR MARKED ASSIGNMENT

Explain the term ‘Unboxing’ with respect to C#
Describe the process of boxing conversion

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# DATA TYPES

CONTENTS

1.0 INTRODUCTION

In this unit, we'll be considering data types in general and typical data types in C#.

2.0 OBJECTIVES

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

- State what data types portray
- Discover how to declare a variable in C#
- List the common data types in C#
- Give specific examples of each data type in C#

3.0 MAIN CONTENT

3.1 Data Types

Data Types in a programming language describes what type of data a variable can hold. **CSharp** is a strongly typed language, therefore every variable and object must have a declared type.

SELF ASSESSMENT EXERCISE

We often refer to C# as being a strongly typed language. Give a brief explanation

_____
_____
_____

3.2 Declaring a Variable in C#

When we declare a variable, we have to tell the compiler about what type of the data the variable can hold or which data type the variable belongs to.

Syntax: DataType VariableName
DataType: The type of data that the variable can hold

VariableName: the variable we declare for hold the values.

Example:

```
int count;
```
int : is the data type

count : is the variable name

The above example shows, declare a variable 'count' for holding an integer values.

3.3 Common Datatypes in C#

The following are the commonly used datatypes in C#:

3.3.1 bool

The bool keyword is an alias of System.Boolean. It is used to declare variables to store the Boolean values, true and false. In C# , there is no conversion between the bool type and other types.

C# Runtime type : System.Boolean
CSharp declaration : bool flag;
CSharp Initialization : flag = true;

CSharp default initialization value : false

3.3.

int variables are stored signed 32 bit integer values in the range of -2,147,483,648 to +2,147,483,647

C# Runtime type : System.Int32
CSharp declaration : int count;
CSharp Initialization : count = 100;

CSharp default initialization value : 0

3.3.

The decimal keyword denotes a 128-bit data type.The approximate range and precision for the decimal type are -1.0 X 10-28 to 7.9 X 1028

C# Runtime type : System.Decimal
CSharp declaration : decimal val;
CSharp Initialization : val = 0.12;

CSharp default initialization value : 0.0M

3.3.4 string

The string type represents a string of Unicode characters. string variables are stored any number of alphabetic,

numerical, and special characters .

C# Runtime type : System.String
CSharp declaration : string str;

CSharp Initialization : str = "csharp string";

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 data types in C#
- State the syntax for declaring a variable 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".

MODULE 3 LANGUAGE BASICS

UNIT 1 NAMING CONVENTIONS

CONTENTS

1.0 INTRODUCTION

This unit describes the fundamental naming conventions in C# programming. Enjoy your studies.

2.0 OBJECTIVES

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

- State the significance of naming standards
- Identify the conventions for naming in C#
- Give a classic example to demonstrate the naming conventions

3.0 MAIN CONTENT

3.1 Reasoning

Much of the naming standards are derived from Microsoft's .NET Framework libraries. These standards have proven to make names readable and understandable "at a glance". By using the correct conventions when naming objects, you ensure that other C# programmers who read your code will easily understand what objects are without having to search your code for their definition.

SELF ASSESSMENT EXERCISE

What is the relevance of using the correct conventions when naming objects?

_____
_____
_____

3.2 Conventions

3.2.1 Namespace

Namespaces are named using **Pascal Case** (also called UpperCamelCase) with no underscores. This means the first letter of every word in the name is capitalized. For example: `MyNewNamespace`. Also, note that Pascal Case also denotes that acronyms of three or more letters should only have the first letter capitalized (`MyXmlNamespace` instead of `MyXMLNamespace`)

3.2.2 Assemblies

If an assembly contains only one namespace, they should use the same name. Otherwise, Assembles should follow the normal Pascal Case format.

3.2.3 Classes and Structures

Pascal Case, no underscores or leading "C", "cls", or "I". Classes should not have the same name as the namespace in which they reside. Any acronyms of three or more letters should be pascal case, not all caps. Try to avoid abbreviations, and try to always use nouns.

3.2.4 Exception Classes

Follow class naming conventions, but add Exception to the end of the name. In .Net 2.0, all classes should inherit from the *System.Exception* base class, and not inherit from the *System.ApplicationException*.

3.2.5 Interfaces

Follow class naming conventions, but start the name with "I" and capitalize the letter following the "I". Example: `IFoo` The "I" prefix helps to differentiate between Interfaces and classes and also to avoid name collisions.

3.2.6 Functions

Pascal Case, no underscores except in the event handlers. Try to avoid abbreviations. Many programmers have a nasty habit of overly abbreviating everything. This should be discouraged.

3.3 Classic Example

Here is an example of a class that uses all of these naming conventions combined.

```
using System;

namespace MyExampleNamespace
{
   public class Customer : IDisposable
   {
      private string _customerName;
      public string CustomerName
      {
         get
         {
            return _customerName;
         }
         set
         {
```

```
            _customerName = value;
            _lastUpdated = DateTime.Now;
        }
    }

    private DateTime _lastUpdated;

    public DateTime LastUpdated
    {
        get
        {
            return _lastUpdated;
        }
        private set
        {
            _lastUpdated = value;
        }
    }

    public void UpdateCustomer(string newName)
    {
        if( !newName.Equals(customerName))
        {
            CustomerName = newName;
        }
    }

    public void Dispose()
    {
        //Do nothing
    }
  }
}
```

4.0 CONCLUSION

Winding up, we can go over the key points of this unit. The naming standards are derived from Microsoft's .NET Framework libraries. By using the correct conventions when naming objects, you ensure that other C# programmers who read your code will easily understand what objects are without having to search your code for their definition.

5.0 SUMMARY

This unit provided an overview the significance of naming standards, identifying the conventions for naming in C#. We equally saw an example of a class that uses all the naming conventions. We hope you have found this unit interesting.

6.0 TUTOR MARKED ASSIGNMENT

- State the naming conventions for functions and interfaces

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# SYNTAX

CONTENTS

1.0 INTRODUCTION

This unit describes the basics of how the applications you write will be interpreted. C# syntax looks quite similar to the syntax of Java because both inherit much of their syntax from C and C++. **The object-oriented** nature of C# requires the high-level structure of a C# program to be defined in terms of **classes**, whose detailed behaviors are defined by their *statements*. Enjoy your studies.

2.0 OBJECTIVES

After going through this unit you would be able to:

- Explain the term ‘Statements’
- Identify the categories of statements
- State the role of code blocks
- Identify the styles of comments allowed in C#
- Discover the aspect of case-sensitivity in C#

3.0 MAIN CONTENT

3.1 Statements

The basic unit of execution in a C# program is the *statement*. A statement can declare a variable, define an expression, perform a simple action by calling a method, **control the flow of execution** of other statements, create an object, or assign a value to a variable, property, or field. Statements are usually terminated by a semicolon.

Statements can be grouped into ***comma-separated statement lists*** or ***brace-enclosed statement blocks***.

Examples:

```
int sampleVariable;                    // declaring a variable
sampleVariable = 5;                    // assigning a value
Method();                              // calling an instance method
SampleClass sampleObject = new SampleClass(); // creating a new
instance of an object
sampleObject.ObjectMethod();           // calling a member function
of an object

// executing a "for" loop with an embedded "if" statement
for(int i = 0; i < upperLimit; i++)
{
   if (SampleClass.SampleStaticMethodReturningBoolean(i))
   {
      sum += sampleObject.SampleMethodReturningInteger(i);
   }
}
```

SELF ASSESSMENT EXERCISE

What is the significance of statements?

_____
_____
_____
_____

3.2 Statement Blocks

A series of statements surrounded by curly braces form a *block* of code (i.e. statement blocks). Among other purposes, code blocks serve to limit the scope of variables defined within them. Code blocks can be nested and often appear as the bodies of methods.

```
private void MyMethod(int value)
{  // This block of code is the body of "MyMethod()"

   // The 'value' integer parameter is accessible to everything in the
method

   int methodLevelVariable; // This variable is accessible to everything in
the method

   if (value == 2)
   {
      // methodLevelVariable is still accessible here

      int limitedVariable; // This variable is only accessible to code in the
if block

      DoSomeWork(limitedVariable);
   }

   // limitedVariable is no longer accessible here

}  // Here ends the code block for the body of "MyMethod()".
```

3.3 Comments

Comments allow inline documentation of source code. The C# compiler ignores comments. Three styles of comments are allowed in C#:

3.3.1 Single-line Comments

The "//" character sequence marks the following text as a single-line comment. Single-line comments, as one would expect, end at the first end-of-line following the "//" comment marker.

3.3.2 Multiple-line Comments

Comments can span multiple lines by using the multiple-line comment style. Such comments start with "/*" and end with "*/". The text between those multi-line comment markers is the comment.

```
//This style of a comment is restricted to one line.
/*
  This is another style of a comment.
  It allows multiple lines.
*/
```

3.3.3 XML Documentation-line Comments

This comment is used to generate XML documentation. Each line of the comment begins with "///".

```
/// <summary> documentation here </summary>
```

This is the most recommended type. Avoid using butterfly style comments. For example:

```
//**************************
// Butterfly style documentation comments like this are not recommended.
//**************************
```

3.4 Case Sensitivity

C# is **case-sensitive**, including its variable and method names.

The variables `myInteger` and `MyInteger` below are distinct because C# is case-sensitive:

```
int myInteger = 3;
int MyInteger = 5;
```

For example, C# defines a class `Console` to handle most operations with the console window. Writing the following code would result in a compiler error unless an object named `console` had been previously defined.

```
 // Compiler error!
 console.writeline("Hello");
```

The following corrected code compiles as expected because it uses the correct case:

```
 Console.WriteLine("Hello");
```

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 3 GETTING STARTED WITH C#

CONTENTS

1.0 INTRODUCTION

In the previous unit we examined C# syntax. Here, we will be looking at Hello World a-la C# and the minimal requirement to use C#. Do make the most of your studies.

2.0 OBJECTIVES

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

- Explain the notion of managed applications
- Describe the parts of a Hello World source code
- State the minimal requirements for using C#
- Outline the procedure for building and running Hello GUI.cs

3.0 MAIN CONTENT

3.1 HelloWorld a-la C#

In any introductory book on programming, it is difficult to avoid the required *HelloWorld* application, and this is no exception. (But don't worry; there are some more application examples in this course material).

3.1.1 Managed Applications

```
using System.Windows.Forms;
using System.Drawing;

class MyForm:Form{
  public static void Main(){
    Application.Run(new MyForm());
  }

  protected override void OnPaint(PaintEventArgs e){
    e.Graphics.DrawString("Hello World!", new Font("Arial", 35),
      Brushes.Blue, 10, 100);
  }
}
```

Figure 1-1 HelloGUI.cs

This is an example of a simple C# or managed application. **Managed applications** are applications that run on the .NET Framework.

Note: Remember the terms *Managed Code* and *Managed Application*. They are often used when referring to .NET code generically, as opposed to referring to a specific programming language like C# or Visual Basic.

3.1.2 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). In addition it defines two new methods, Main() and OnPaint().

All C# (or .NET) applications must have a static method named Main() defined to be the entry point of the application.

Note: C# is a case-sensitive programming language. This means that it would see methods named Main() and main() as different methods. The entry point method for C# programs must be called Main().

The static Main() method can be defined in any class in the application, so long as its name is “Main” and it is declared to be static.

Note: Methods declared as static do not require an object instance to be called. They are similar to global functions, and are often referred to as type-methods (rather than instance methods).

The OnPaint() method is an override of a virtual method on the Form class. It is called when the window needs to paint itself. This sample uses this method to draw the text "Hello World!"

SELF ASSESSMENT EXERCISE

In what context are the terms *Managed Code* and *Managed Application* commonly used?

_____

_____

_____

3.2 C# and the .NET Framework Class Library (FCL)

We mentioned in the introduction that a big requirement of C# programming was learning the .NET Framework. This is true. To be more specific, the Framework Class Library (called the FCL) will be used extensively in almost every C# application.

We will make references to reusable object types defined in the FCL throughout this tutorial. The documentation for these types ships with the .NET Framework, and also installs with Visual Studio.NET.

To further emphasize the importance of the FCL, look back at Figure 1-1. While the code is written in C#, a great deal of the functionality comes from types defined in the FCL. For example, the window behaves as it does because MyForm is derived from the FCL class Form. Additionally, the PaintEventArgs, Graphics, Font, and Brushes classes play an important role in the drawing of the string on the form. And even the "Hello World!" string itself is of type String. All of these types (and many more) are defined by the FCL, and useable within your C# applications.

Two more point before we move on. First, types defined in the FCL

exist in a hierarchy of namespaces. For example the Form class is really the System.Windows.Forms.Form class. However, since we want to only type Form in our source code we enter a using statement at the beginning of our source code module that indicates that we are using the System.Windows.Forms namespace in this module. Notice that the code in Figure 1-1 has two using statements.

Second, the types in the FCL are published in files called *Assemblies*. When you compile your C# application you must include references to the assemblies that are used by your code. Visual Studio.NET lets you add references to your projects, and the command line C# compiler uses the /R switch to indicate referenced assemblies. We will consider more about building your code later.

3.3 Minimal Requirements to Use C#

In this section we are going to consider how to build a C# application. More specifically, we are going to see the minimal requirements to use C#. The most common way of programming with C# is by using the Visual Studio.NET developers integrated development environment (IDE). However, the C# command-line compiler ships with the .NET Framework, and we will use that first.

First, you should install the latest version of the .NET Framework to your PC running Windows. If you like you can install Visual Studio.NET which installs the .NET Framework automatically.

Once you have the .NET Framework installed, you also have the Framework SDK, which includes the command-line compiler for C#. The C# compiler is called csc.exe and exists in the following directory.

C:\WINDOWS\Microsoft.NET\Framework\v1.0.2914
The final directory named v1.0.2914 indicates the version of the .NET Framework that you have installed.

This the directory commonly set in the path, using the advanced tab of the *System* control panel applet. (The Visual Studio .NET installation optionally sets my path for me). This lets me type csc from the command line in any directory in my file-system. (I find that I use Visual Studio for my big projects, but I use the command line compiler for my little tests and scripts.)

This is all you need to use C#. You can use any editor you like to create C# source code modules. You should give your C# models a .cs extension, and you are good to go.

3.3.1 Compiling C# Code

If you were to cut and paste the source code from Figure 1-1 into a file (perhaps using Notepad), and save it as HelloGui.cs, you could then

compile it into an executable with the following command.

C:\>csc /Target:winexe HelloGui.cs
Its just that simple.

The /Target:winexe switch indicates to the compiler that you are creating a GUI application. The default is command-line or console applications.

3.3.2 Building and Running HelloGUI.cs

1. The source code in Figure 1-1 is a complete GUI application written in C#. It will display a window with some text on the Window.
2. Type in or copy the source code and save it in a .cs file.
3. Compile the source code using the command line compiler or the Visual Studio IDE.
    a. Hint: If you are using Visual Studio you will need to add references for the System.Drawing.dll, System.Windows.Forms.dll, and the System.dll.
4. Run the new executable.

4.0 CONCLUSION

In conclusion, a ***destructor*** also known as a ***finalizer*** is a special member that can be added to a class. It is executed automatically when the object is being removed from memory.

5.0 SUMMARY

We considered the notion of managed applications, described the parts of a Hello World source code, stated the minimal requirements for using C# and outlined the procedure for building and running Hello GUI.cs
To test your knowledge, let us attempt the exercise below.

6.0 TUTOR MARKED ASSIGNMENT

- Outline the steps involved in building and running Hello GUI.cs

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 C# APPLICATIONS

Unit 1 Creating Console Assemblies
Unit 2 Creating GUI Assemblies
Unit 3 Creating Code Library Assemblies

UNIT 1 CREATING CONSOLE ASSEMBLIES

CONTENTS

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 console application
- Specify the types of command line utility
- Give a classic example of a console application
- Outline the steps required to build and run a console application

3.0 MAIN CONTENT

3.1 Console Applications

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**. Actually many ad-hoc C# projects are console applications.

SELF ASSESSMENT EXERCISE

What are console applications?

_____
_____
_____

3.2 Application Sample

```
using System;

class App{
  public static void Main(String[] args){
    try{
      Int32 iterations = Convert.ToInt32(args[0]);
      if(iterations > 138){
        throw new Exception();
      }
      Decimal lastNum = 1;
      Decimal secondToLastNum = 0;
      while(iterations-- > 0){
        Decimal newNum = lastNum+secondToLastNum;
        Console.WriteLine(newNum);
        secondToLastNum = lastNum;
        lastNum = newNum;
      }
    }catch{
      Console.WriteLine(
        "Usage:  Rabbits [Fib Index]\n"+
        "\t[Fib Index] < 139");
    }
  }
}
```

Figure 1 Rabbits.cs

The C# source code shown in Figure 1 will calculate how many rabbits you get after a certain number of generations (assuming ideal circumstances, of course).

This is an example of a console application. If you build this application you can run it from the command line, and pass it an argument indicating the number of generations you would like to calculate.

3.3 Building a Console Application File

To build this file using the command line compiler you would use the following command.

C:\>csc Rabbit.cs

You will notice that it is not necessary to use the /Target switch like it was when we built the source code. This is because the default assembly built by the C# compiler is a console executable.

This is how you build a console application; simple stuff! However, the Rabbits.cs sample also shows some interesting things about C# in general, so we will highlight them here.

Like our last application, Rabbits.cs defines a class (arbitrarily named App) which defines an entry point function named Main().

The Main() method in Rabbits.cs does something a little different by taking a parameter defined as an array of String objects. These are the command line arguments passed to the program.

Rabbits.cs uses structured exception handling to handle errors. This is the try-catch code block that surrounds the bulk of the application.

You will notice from this sample that C# uses C syntax for its loop constructs. The **while loop** syntax in Rabbits.cs, is exactly what it would be in C, C++, or Java.

The sample code in Figure 1 uses the static WriteLine() method of the Console type defined in the Framework Class Library to output text to the console window. Notice that an instance of the Console type was not necessary to call the method. This is because WriteLine() is defined as a static method.

This sample uses a numeric type called Decimal which allows us to calculate many more generations than we would be able to do with a 32-bit integer value. The Decimal type is well suited for financial and scientific applications.

Rabbits.cs is an iterative Fibonacci calculator. Its' written this way just for fun. There is nothing about C# that would prohibit you from implementing this algorithm using recursion.

3.4 Building and Running a Console Application

1. The source code in Figure 1 Rabbits.cs is a console Fibonacci number generator written in C#.
2. Type in or copy the source code and save it in a .cs file.
3. Compile the source code using the command line compiler.
    a. Hint: The line that you would use to compile this application is as follows.
    c:\>csc Rabbit.cs
4. Run the new executable.

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".

UNIT 2 CREATING GUI ASSEMBLIES

CONTENTS

1.0 INTRODUCTION

In this unit, you'll gain knowledge of how to create GUI assemblies in C#. Enjoy your studies.

2.0 OBJECTIVES

What you would study in this unit, would equip you to do the following:

- Identify the steps involved in compiling the GUI application
- State the command for compiling a GUI application
- Discover the tool that facilitates GUI coding
- Outline the procedure required to build and run a GUI application

3.0 MAIN CONTENT

3.1 Creating GUI Assemblies in C#

Although we may have already seen how to build GUI applications in other Programming languages, we will do so again here. The following sample more

closely resembles GUI applications in the real world.

```
using System;
using System.Drawing;
using System.Windows.Forms;

class App{
  public static void Main(){
    Application.Run(new TribbleForm());
  }
}

class TribbleForm:Form{
  TextBox generationsTextBox;
  ListBox fibList;

  public TribbleForm() {
    generationsTextBox = new TextBox();
    generationsTextBox.Location = new Point(16, 16);

    Button tribbleButton = new Button();
    tribbleButton.Location = new Point(16, 48);
    tribbleButton.Size = new Size(100, 20);
    tribbleButton.Text = "Tribble Count";
    tribbleButton.Click += new EventHandler(OnClick);
    AcceptButton = tribbleButton;

    fibList = new ListBox();
    fibList.Location = new Point(16, 88);
    fibList.Size = new Size(192, 134);
    fibList.Anchor = AnchorStyles.Top|AnchorStyles.Bottom
      |AnchorStyles.Left|AnchorStyles.Right;

    ClientSize = new Size(226, 235);
    Controls.AddRange(new Control[]{ generationsTextBox,
      tribbleButton, fibList});
    Text = "Tribble Calculator";
  }

  void OnClick(Object sender, EventArgs e){
    try{
      Int32 iterations = Convert.ToInt32(generationsTextBox.Text);
      if(iterations > 138)
        throw new Exception();
```

```
        fibList.Items.Clear();
        Decimal lastNum = 1;
        Decimal secondToLastNum = 0;
        while(iterations-- > 0){
          Decimal newNum = lastNum+secondToLastNum;
          fibList.Items.Add(newNum);
          secondToLastNum = lastNum;
          lastNum = newNum;
        }
        fibList.SelectedIndex = fibList.Items.Count-1;
      }catch{
        MessageBox.Show("Enter a number from 1-138");
      }
    }
}
```

Figure 1 Tribbles.cs

Since we are moving from console applications to the more advanced GUI application, it would be appropriate to shift the theme from rabbits to tribbles. Tribbles.cs is a Fibonacci calculator with a graphical UI. The source code in Figure 1 seems complex. Don't worry if some of it is unclear; it's not the code that we are concerned with here, but rather the compiling of the code.

3.2 Compiling the GUI Application

To compile Tribbles.cs using the command line compiler use this command.

C:\>csc /Target:winexe Tribbles.cs

This command will create a GUI assembly executable. Like before, we need to use the /Target switch to indicate to the compiler that this is a GUI application.

Like the previous example, Tribbles.cs shows a lot more than just how to compile source code. It includes various usages of GUI types like Button and ListBox, as well as exception handling. Here are some highlights.

Tribbles.cs defines two classes. An App class which has a static Main() method to be used as the programs entry point, and the TribbleForm class which is derived from form and implements the programs window.

Most of the programs functionality exists in the TribbleForm class' constructor and the OnClick() handler method for the button.

Although you can write GUI code from scratch, it is far more common to use a forms designer to help with the task. The .NET Framework ships with a free forms designer called WinDes.exe. Of course Visual Studio.NET has forms designer as well. Both allow you to design your GUI using graphical tools, and the designer will create the code (much like the code you see in Figure 1).

C# GUI programs use a set of classes in the FCL known as the Windows Forms classes.

3.3 Building and Running a GUI Application

1. The source code in Figure 1 Tribbles.cs is a GUI Fibonacci number generator written in C#.
2. Type in or copy the source code and save it in a .cs file.
3. Compile the source code using the command line compiler.
    a. Hint: The line that you would use to compile this application is as follows.
    c:\>csc /Target:winexe Tribbles.cs
4. Run the new executable.

4.0 CONCLUSION

To wrap up, we discovered how to build, run and compile GUI applications.

5.0 SUMMARY

In this unit, we discovered the steps involved in compiling the GUI application. We equally identified the command for compiling a GUI application and the tool that facilitates GUI coding. Finally we learnt about the steps required to build and run a GUI application. Let us now attempt the question below.

6.0 TUTOR MARKED ASSIGNMENT

State the procedure for building and running a GUI 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".

Online Materials

1. C# Language (MSDN)
2. C# Programming Guide (MSDN)
3. C# Specification (MSDN)
4. ISO C# Language Specification.

UNIT 3 CREATING CODE LIBRARY ASSEMBLIES

CONTENTS

1.0 INTRODUCTION

In this unit, you have a chance to learn another aspect of C# programming. We will study the procedure for building a code library. We'll equally look at the syntax for compiling objects.

2.0 OBJECTIVES

What you would study in this unit, would enable you do the following:

- Identify the source code for defining an object
- Describe the steps involved in compiling an object
- Identify parts of the syntax for compiling an object
- Outline the steps involved in building a code library
- Outline the procedure for building an app to test a code library

3.0 MAIN CONTENT

3.1 Creating Code Library Assemblies in C#

Knowing how to create executables with C# is important. But component development is going to become increasingly more important as the software industry evolves. More and more application code is going to be housed in reusable objects. These reusable types will often exist in **binary modules** or **assemblies** external of the main executable.

In fact, when you design web pages using C#, the entire code that implements the web page exists in a library assembly, with one or more objects derived from the Page type defined in the FCL. But we should start with a more mundane object library.

At the risk of absolutely hammering the Fibonacci series into the ground, we are going to make a Fib object that can be reused from other assemblies. We admit that the Fib object is perhaps one of the strangest implementations of a Fibonacci generator in existence but it does demonstrate reusable object libraries in C# nicely.

SELF ASSESSMENT EXERCISE

List 3 types of static properties

_____
_____
_____
_____

3.2 Source Code for Defining Objects

```
using System;

public class Fib{
  Decimal current;
  Decimal last;
  public Fib(){
    current = 1;
    last = 0;
  }

  private Fib(Decimal last, Decimal secondToLast){
    current = last+secondToLast;
    this.last = last;
  }

  public Fib GetNext(){
    return new Fib(current, last);
  }

  public Decimal Value{
    get{return current;}
  }
}
```

Figure 1 FibObj.cs

The source code in Figure defines an object named Fib that can be used from other assemblies.

3.3 Compiling the Object

To compile this object into a library (using the command line compiler) do the following.

```
C:\>csc /Target:library FibObj.cs
```

Note: In this case we used the /Target switch to indicate that the assembly we are building is a library. This will create an assembly named FibObj.dll. FibObj.dll does not have a static Main() entry method defined, and it can not be executed directly. If you try to build a non-library assembly without an entry point method defined, the compiler will give you an error message.

We have now built a binary library containing an object with executable code. However, we need a regular executable assembly just to try the code.

```
using System;

class App{
  public static void Main(){
    Int32 index = 50;
    Fib obj = new Fib();
    do{
      Console.WriteLine(obj.Value);
      obj = obj.GetNext();
    }while(index-- != 0);
  }
}
```

Figure 2 FibTest.cs

The sources for FibTest.cs can be used to test the FibObj.dll. However, special measures must be taken when compiling the sources in Figure 2. This is because the source code refers to an object type called Fib, and the compiler needs to be told where the Fib type is implemented. That is done at compile time as follows.

C:\>csc /r:FibObj.dll FibTest.cs
The /r switch indicates to the C# compiler that the source code in FibTest.cs references objects implemented in FibObj.dll. There are a couple of points worthy of note here. First, the referenced file is the binary assembly file FibObj.dll, not the source code file FibObj.cs. Second, the lack of /Target switch indicates to the compiler that FibTest.cs should be compiled into the default assembly type, which is a console application.

Note: Remember the concept of assembly references. It comes up all of the time. Each time you use third party objects you will need to reference the assembly that implements the object. Every time you use a type defined by the FCL you are referencing an assembly that ships with the Framework Class Library. (You might have noticed that the command line compiler automatically references the assemblies in the FCL for you. The Visual Studio.NET environment requires you to explicitly set assembly references).

3.4 Creating Console Assemblies

What is happening here between FibTest.exe and FibObj.dll is the simple foundation of a very important ability. That is the ability for objects to use each other without being compiled together (or even at the same time, or company, or content). This is a major improvement over the way C++ deals with object types, and is much more similar to the

way Java deals with types. (However, it would take a lot of text to explain it, but C# and the .NET Framework's ability to compose your application of binary libraries of objects is far advanced of Java or any other mainstream platform.

3.5 Building a Code Library

1. The source code in Figure 1 FibObj.cs is a C# reusable object to be compiled into a library.
2. Type in or copy the source code and save it in a .cs file.
3. Compile the source code using the command line compiler.
    a. Hint: The line that you would use to compile this application is as follows.
    c:\csc /Target:library FibObj.cs
4. Do a directory to confirm that a file named FibOjb.dll was created.

3.6Building an App to Test a Code Library

1. The source code in Figure 2 FibTest.cs is a brief console application designed to consume or test the object in FibObj.dll.
2. Type in or copy the source code and save it in a .cs file.
3. Compile the source code using the command line compiler.
    a. Hint: The line that you would use to compile this application is as follows.
    c:\>csc /r:FibObj.dll FibTest.cs
4. Run the new executable.

4.0 CONCLUSION

In conclusion, we have seen that static properties are often thought of as global variables, they can be ***read/write, read-only or write-only*.** The only condition for adding a static property to a class is that the declaration includes the 'static' keyword to modify the property's behaviour.

5.0 SUMMARY

In sum, we discovered the source code for defining an object, the steps involved in compiling an object. We equally identified parts of the syntax for compiling an object as well as the steps involved in building a code library. You can now attempt the question below.

6.0 TUTOR MARKED ASSIGNMENT

Outline the procedure for building a code library

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".

Online Materials

5. C# Language (MSDN)
6. C# Programming Guide (MSDN)
7. C# Specification (MSDN)
8. ISO C# Language Specification.
9. Microsoft Visual C#.NET

MODULE 5 VISUAL STUDIO.NET

Unit 1 Creating a Project
Unit 2 Language Concepts
Unit 3 C# Expressions and Operators

UNIT 1 CREATING A PROJECT

CONTENTS

1.0 INTRODUCTION

This unit is a preamble to Visual Studio.NET. If you are going to be doing a lot of C# programming, then it is most likely that you will find yourself using Visual Studio.NET. Visual Studio.NET manages a lot of the details of project management, building, debugging, and GUI development. Most programmers actually spend about 50% of their C# programming time using Visual Studio.NET and the other 50% just writing quick programs with notepad (to solve a quick problem or test something). Do take note of the key points.

2.0 OBJECTIVES

What you would study in this unit, would equip you to:

- Give an overview of VS.NET
- List and describe the common template types in VS.NET
- Identify the role of a solution explorer
- Outline the procedure for adding a source file to a project

3.0 MAIN CONTENT

3.1 Overview of VisualStudio.NET

VisualStudio.NET is an Integrated Development Environment or IDE. IDE's commonly include source code editors (usually pretty fancy ones), as well as project managers, online-help and debuggers, all in one software package. The idea is to keep all of your programming related tasks in one place. VS.NET does a good job of this.

When you run VS.NET you are presented with a *Start Page*. This page has links that help you create a new project or open an existing one. VS.NET projects are arranged in a hierarchy of projects and solutions. Solutions are sort of a collection of projects. Most often you will have one project per solution, but if you are creating a program with an executable and a library of reusable classes, then you might make a solution with two projects, for example.

SELF ASSESSMENT EXERCISE

What is the main role of an IDE?

_____
_____
_____

3.2 Creating a Project

When you select new project you are presented with a screen as follows.

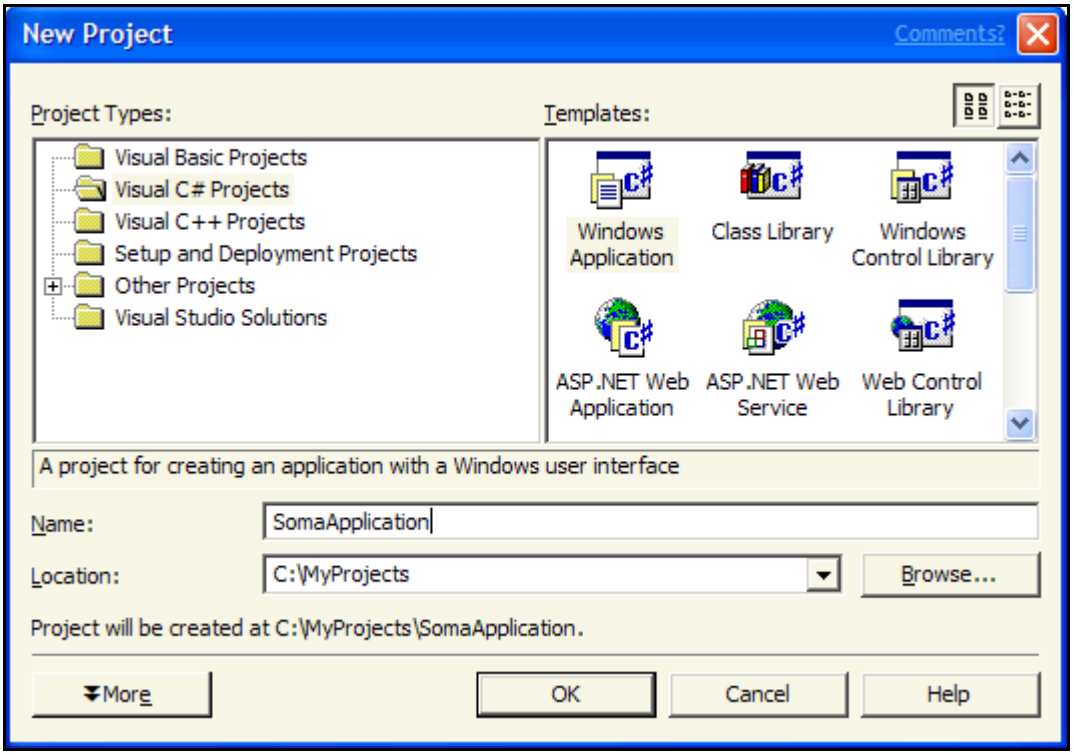

Figure 1 New Project in VS.NET

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 on the right.

3.2.1 Common Template Types

The most common template types are:

a. Empty Project – This one is hidden down near the bottom of the templates, but it can be the most useful. This is because with the *Empty Project* template VS.NET lets you add all of the source code modules and it doesn't set anything up for you. Sometimes, this is what you want.

b. Windows Application – This template sets up a project for building a GUI application. It will create a couple of source code modules for you, one of which already includes a class derived from Form to be your main window. When you create a project from this template, it loads up automatically in the *Forms Designer* tool of VS.NET which lets you drag and drop buttons and other GUI elements onto your window.

c. Console Application – If you want to create a console or command line application you should use this template.

d. Class Library – If you want to make a library of reusable types, you should use the Class Library template. The assembly that you build from this project will not be executable, but you can use the types in this file from other executables.

e. ASP.NET Web Application – This template creates a web forms application for creating web-pages with C#. This is an advanced project type, and you should consider writing a few console or GUI applications in C# before attempting a web-form. I will discuss web forms in detail in a tutorial later in this series.

f. ASP.NET Web Service -- This template creates a web service with C#. This is an advanced project type. I will discuss web services in detail in a tutorial later in this series.

3.3 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.

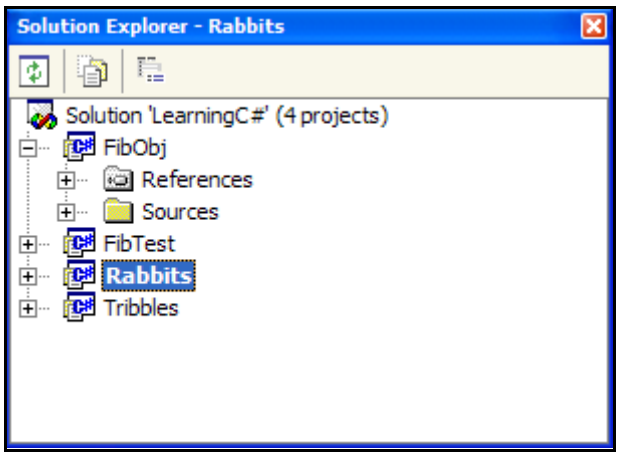

Figure2 Solution Explorer

The Solution Explorer is your command central for dealing with a 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.

3.4 Adding a Source File to the Project

To add a .cs source file to the project *FibObj* shown in Figure2, you could simply right click on the icon for *FibObj*, and then select *Add->Add New Item.*

If you choose to program in C# without using Visual Studio.NET, you can write all the same kinds of software that you can write with VS.Net.

If you do choose to use Visual Studio.NET, I would suggest setting aside a couple of hours to just create several trial projects. Build them, debug them, and try making minor modifications to them. This will introduce you to the look and feel of VS.NET and will make it more productive for you when you have to do real work with the tool.

4.0 CONCLUSION

To end, we learnt that VisualStudio.NET is an Integrated Development Environment or IDE. IDE's commonly include source code editors (usually pretty fancy ones), as well as project managers, online-help and debuggers, all in one software package. 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.

5.0 SUMMARY

In this unit, we looked at: the notion of VS.NET, the common template types in VS.NET and the role of a solution explorer. You may now proceed to the tutor marked assignment below.

6.0 TUTOR MARKED ASSIGNMENT

- Give an overview of VS.NET
- List and describe the common template types in VS.NET

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".

Online Materials

1. C# Language (MSDN)
2. C# Programming Guide (MSDN)
3. C# Specification (MSDN)
4. ISO C# Language Specification.
5. Microsoft Visual C# .NET

UNIT 2 LANGUAGE CONCEPTS

CONTENTS

1.0 INTRODUCTION

In this unit, we'll learn about the syntax ground rules and the primitive types in C#.

2.0 OBJECTIVES

What you would study in this unit, would enable you do the following:

- State the basic rules that apply to C# code syntax
- Explain the concept of primitive types in C#
- List the common C# primitive types and their corresponding aliases

3.0 MAIN CONTENT

3.1 Syntax Ground Rules

The following are some basic rules that apply to C# code syntax.

- C# is a case-sensitive language. This means that method and variable names are distinguished by their case. For example, a method named Hello() and a method named hello() would be seen as different methods in C#.

- White space means nothing in C#. White space includes spaces, carriage returns, and tabs. The following two code blocks are seen as exactly the same by the C# compiler.
  ```
  if(x==y){DoSomething()}
  ```
  or
  ```
  if(x==y){
    DoSomething();
  }
  ```
  You should use white space to increase the readability of your source code.
- The semicolon (;) is used to indicate the end of a statement. The only exception is the end of a code block (}), which does not require a terminating semicolon (;). It is because of this rule that the compiler can safely ignore carriage returns (where other languages use carriage returns to indicate the end of a statement).
- Statements can be grouped into a single unit called a code block. This is done by putting the statements inside of a pair of curly braces ({}). Code blocks can be nested.

SELF ASSESSMENT EXERCISE

Outline at least three basic rules that apply to C# code syntax

_____

_____

_____

3.2 Primitive Types

The table shown later in this section shows the types in the Framework Class Library that are treated as primitive types by C#. Primitive types are special because the C# language will allow direct assignment, math operations, and comparison of primitive types. For example, the following line of code would be legal in C# because both variables are of type Int32

```
x = x + y;
```

Meanwhile, the very same line of code would not be legal of the x and y variables were of some non-primitive type such as Form or FileStream.

One other interesting point about the primitive types in C# is that the C# language provides *Aliases* for the type names. For example, whether you declare a variable of type Int32 or of type int, you are always really declaring an Int32 variable. Choosing which name to use in your source code is a matter of style. The most important thing is that you recognize that the FCL name for a primitive type and the alias name for a primitive type both refer to the same underlying type.

C# Primitive	C# Alias	Description
Boolean	Bool	Indicates a true or false value. The if, while, and do-while constructs require expressions of type Boolean.
Byte	Byte	Numeric type indicating an unsigned 8-bit value.
Char	Char	Character type that holds a 16-bit Unicode character.
Decimal	Decimal	High-precession numerical type for financial or scientific applications.
Double	Double	Double precession floating point numerical value.
Single	Float	Single precession floating point numerical value.
Int32	Int	Numerical type indicating a 32-bit signed value.
Int64	Long	Numerical type indicating a 64-bit signed value.
SByte	Sbyte	Numerical type indicating an 8-bit signed value.
Int16	Short	Numerical type indicating a 16-bit signed value.
UInt32	Uint	Numerical type indicating a 32-bit unsigned value.
UInt64	Ulong	Numerical type indicating a 64-bit unsigned value.
UInt16	Ushort	Numerical type indicating a 16-bit unsigned value.
String	String	Immutable string of character values
Object	Object	The base type of all type in any managed code.

Figure 1 Primitive Types

3.3 Primitive Types in C#

Here are a couple of points worthy of note about the primitive types in C#.

C# has a String type, rather than requiring that strings be represented as an array of Char variables like C or C++.

C# has a Boolean type, and it is not valid to use a numerical type where a Boolean expression is required. C# also includes two keywords for indicating true and false.

Since C# uses types defined in the FCL as its primitive types, C#'s primitive types are Object's just like all other types in the system. Said another way, all C# primitive types are derived from Object, and as objects do object-like things such-as implement instance methods, etc.

The following C# method shows the use of some of the primitive types in C#. The primitive variables are shown in red.

```
public static void Primitives(){
  Int32 x = 10;
  Int32 y = x/3;
  Console.WriteLine(y);

  Decimal d = 10;
  Decimal e = d/3;
  Console.WriteLine(e);

  String s = "Hello";
  String t = s+" C#";
  Console.WriteLine(t);

  Boolean f = (x==y);
  Console.WriteLine(f);
}
```

Figure 2 Primitive Code Snippit (Excerpted from Language.cs)

The code in Figure 2 is the first in this tutorial to not be a compile compileable C# module. If you would like to try compiling or running the code in Figure 2 you can cut and paste the function into a console application, complete with a Main() method, or you can build and run the Languages.cs file.

4.0 CONCLUSION

To wrap up, we learnt that: C# has a String type, rather than requiring that strings be represented as an array of Char variables like C or C++. C# has a Boolean type, and it is not valid to use a numerical type where a Boolean expression is required. C# also includes two keywords for indicating true and false. C# uses types defined in the FCL as its primitive types, C#'s primitive types are Object's just like all other types in the system.

5.0 SUMMARY

In this unit, we identified the basic rules that apply to C# code syntax. We also discovered the concept of primitive types in C# and the common C# primitive types as well as their corresponding aliases. You may now proceed to the tutor marked assignment. Good luck!

6.0 TUTOR MARKED ASSIGNMENT

List at least 5 primitive types in C# and their corresponding aliases.

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# EXPRESSIONS AND OPERATORS

CONTENTS

1.0 INTRODUCTION

In this unit you learn about the common expressions and operators in C#. Enjoy your studies.

OBJECTIVES

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

- Identify the role of operators in C#
- Discover how to modify precedence of operators
- Identify the operator categories and their corresponding operators

3.0 MAIN CONTENT

3.1 Common Expressions and Operators in C#

Expressions are an integral part of programming, and C# treats expressions much the same as many languages. Operators such as +, *, and / are used to manipulate or affect the data in variables, and operators have precedence. And as is common, operator precedence can be modified using parenthesis. Expressions like the following are common in programming, and are also common with C#.

```
Int32 x = 10*(y+z);
```

C# is a *very* type safe language, however, and expressions are strongly typed. For example, the preceding line of code is an expression of type Int32. The compiler will automatically convert between numeric primitive types, so long as no data-loss occurs in the conversion. However, some conversions that are common in languages like C or C++ are illegal in C#.

An example of such an illegal type conversion in an expression is the conversion from an Int32 or some other numeric type to a Boolean type. The following code would be valid in C, but is an error in C#, because a Boolean expression is expected, and an Int32 expression is provided.

```
Int32 x = 10;
while(x--) DoSomething();
```

SELF ASSESSMENT EXERCISE

State how you would modify the precedence of an operator in C#?

_____
_____
_____

3.2 Operators used in C# Expressions

The operators in the following table can be used in C# expressions.

Operator category	Operators
Arithmetic	+ - * / %
Logical (boolean and bitwise)	& | ^ ! ~ && ||
String concatenation	+
Increment, decrement	++ --
Shift	<< >>
Relational	== != < > <= >=
Assignment	= += -= *= /= %= &= |= ^= <<= >>=
Member access	.

Indexing	[]
Cast	()
Conditional (Ternary)	?:
Delegate concatenation and removal	+ -
Object creation	
Type information	
Overflow exception control	
Indirection and Address	* -> [] &

Figure 1 C# Operators

4.0 CONCLUSION

In this unit, we were made to understand that ***expressions*** are an integral part of programming, and C# treats expressions much the same as many languages. We equally discovered that operators such as +, *, and / are used to manipulate or affect the data in variables, and operators have precedence. Operator precedence can be modified using parenthesis.

5.0 SUMMARY

To wrap up, recall that we learnt about the role of operators in C# and how to modify precedence of operators. We equally identified the operator categories and their corresponding operators. Let us now attempt the questions below.

6.0 TUTOR MARKED ASSIGNMENT

- State the common operators used to manipulate data in variables
- List the categories of operators and their corresponding operators

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".