INTRODUCTION TO C# PROGRAMMING

FIRST EDITION

Author
Avonn C. Nova, MIT

Series Editor
Jaime D.L. Ca ro, Ph.D.

First printing of the first edition, 2016
ISBN 978-621-8000-73-5

Published by TechFactors Inc.
Printed in the Philippines

Author Avonn C. Nova, MIT
Series Editor Jaime D.L. Caro, Ph.D.
Cover Design Christian Sabado

Content and Editorial Alvin Ramirez, Eireen Camille S. Linang, John Louie Nepomuceno, Kenneth T. Salazar, and Allan Nicole C. Celestino
Creatives Jiyas Suministrado-Morales, Christian Sabado, Loughem Laquindanum, and Joseph Timothy S. Bago
Systems Mark B. Abliter, Robie Marc R. Peralta, and Carlo M. Espinoza

Exclusively distributed by TechFactors Inc.
101 V. Luna Road Ext., Sikatuna Village
Diliman, Quezon City
1101 Philippines

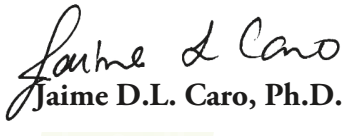
Telefax number: (632) 929 6924
E-mail address: info@techfactors.com
Website: www.techfactors.com

FOREWORD

We live in an age of marvels.

Nowadays, we can move machines with the touch of a button. We can communicate with people halfway across the world in real-time. Innovations occur everyday because of technology. For adolescents today, technology is already a part of daily living. They use mobile phones, computers, iPods, and other electronic gadgets that assist and expand their awareness of the world. However, they have so much more to learn about the full potential and limitations of the technology within their reach.

In an increasingly computer-dependent world, it is important to be aware of the changes in computing technology, and to be keen in understanding on how computer communicates to human. This courseware is not only intended to be an instructional manual discussing individual topics, but also to be a means of exploring the continually improving and expanding world of computer programming – and our society as well.



Jaime D.L. Caro, Ph.D.

Series Editor

ABOUT THE AUTHOR

Avonn C. Nova, MIT has more than 10 years of teaching experience in the field of Computer Science and Information Technology. He graduated Bachelor of Science in Computer Science (cum laude) at Cavite State University in 2001 and received his Masters in Information Technology degree at Technological University of the Philippines in 2006. He already earned units for Doctor of Philosophy major in Education Administration at Manila Central University and served the institution as Dean of the College of Computer Studies for more than 5 years where he pioneered the specialization in Biomedical Informatics. He is currently the Trustee-In-Charge for Membership of the Philippine Society of Information Technology Educators – National Capital Region.

ABOUT THE SERIES EDITOR

Jaime D. L. Caro, Ph.D. has more than 20 years of experience in education and research in the areas of Computer Science, Information Technology, and Mathematics. He received the degrees of Bachelor of Science major in Mathematics (cum laude) in 1986, Master of Science in Mathematics in 1994, and Doctor of Philosophy in Mathematics in 1996, all from the University of the Philippines, Diliman. He spent a year as a post doctorate research fellow at the University of Oxford from 1997 to 1998. He is presently Assistant Vice President for Development of the University of the Philippines, Program Director of the UP Information Technology Development Center (UP ITDC), and a professor of Computer Science in UP Diliman. He is an honorary member of the Philippine Society of Information Technology Education (PSITE), President of the Computing Society of the Philippines (CSP), and a member of the Technical Panel on Information Technology Education of the Commission on Higher Education (CHED). Dr. Caro is a recognized expert on Complexity Theory, Combinatorial Network Theory, Online Communities, and e-Learning.

TABLE OF CONTENTS

COURSE DESCRIPTION

This introductory course is designed to help learners understand basic C Sharp (C#) Programming in the simplest way possible. C# is a general purpose language which contains well- structured features for programming. In order to fully understand the course, you should already be familiar in programming using C and C++ since these languages are related and similar in nature.

COURSE OUTCOMES

The learners should be able to:

- solve simple problems using the fundamental syntax of C# programming language;
- apply elementary techniques in C# programming;
- use generally accepted programming style;
- write C# programs that use selection;
- write C# programs that use loops;
- write C# programs that use methods for program modularization;
- write C# programs that use 2-dimensional arrays;
- work with string data; and
- choose appropriate loops, and decision structures in creating C# programs.

Lesson 1

INTRODUCTION

Content Standards

Learners demonstrate understanding of:

- reasons why C# is the programming choice for mobile development;
- different features of the C# language;
- components of the .NET framework;
- different IDE development tools for C#; and
- alternative versions for other operating systems.

Performance Standard

Learners shall be able to:

- familiarize themselves with the integrated development environment of C# and the use of different functions.

Learning Outcomes

Learners shall be able to:

- compare and contrast C, C++, and C#; and
- evaluate other open source versions of .NET framework.

In January 1999, the principal designer and lead architect of Microsoft, Anders Hejlsberg, designed and built a new programming language with his team called Cool (C-like Object Oriented Language). Though Microsoft had wanted to keep the name "Cool" as the name of the language, they decided not to continue using the name due to trademark issues. It was then renamed to C# when the .NET project was announced to the public in July 2000 during the Professional Developers Conference.

OVERVIEW

C# (pronounced as *see sharp*), is an object-oriented programming language developed by Microsoft. You cannot fully appreciate the beauty of C# if you don't have any experience in C and C++ programming, both of which have features similar to Java.

C# is designed to work with the Microsoft .NET (read as DotNet) platform which consists of executable codes and a runtime environment that allows you to use various high-level languages on different platforms.

There are several reasons why C# is a preferred programming language, especially for mobile development.

- simplified approach in programming
- easily-structured codes
- memory management features
- quicker and easier to detect and isolate errors
- adoptable
- faster execution
- seamless interoperability with native codes
- highly portable

C# LANGUAGE FEATURE

As is the case with Java programming, C# has several programming features that make it pleasing to programmers.

- Arrays
- Constructors and Destructors
- Automatic Garbage Collection
- Indexers
- Main() Method
- Properties
- Passing Parameters
- XML Documentation

.NET FRAMEWORK

The C# program runs on the .NET framework that is an integral component of Windows. Source codes written in C# are compiled through an **intermediate language (IL)** that follows the **common language infrastructure (CLI)** specifications. CLI is an international standard which is the reference in creating environments for development and execution which serve as platforms for the interoperability of libraries and languages.

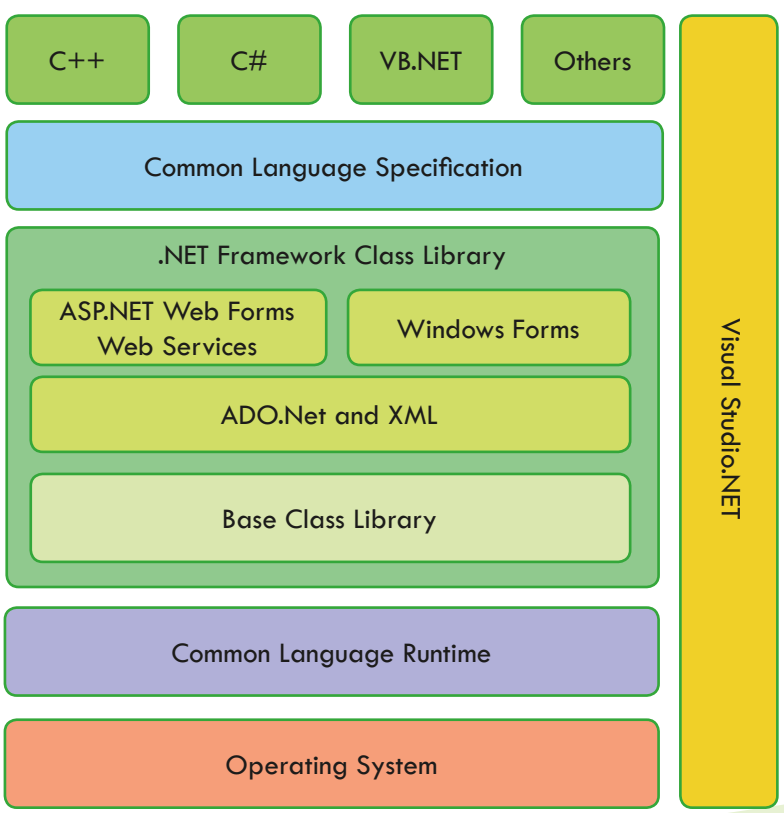
One of the essential components of the .Net framework is its language interoperability because the codes it produces can talk with other .Net versions of C# and C++, as well as Javascript, Visual.Net, and other languages.

The .NET framework helps in writing programs for the following applications:

- Windows
- Web
- Web Services

Following are some of the components of the .NET framework:

- Common Language Runtime (CLR)
- .NET Framework Class Library (FCL)
- Common Type System (CTS)
- Common Language Specification (CLS)
- Metadata and Assemblies
- Windows Forms
- ASP.NET and ASP.NET AJAX
- ADO.NET
- Windows Workflow Foundation (WF)
- Windows Presentation Foundation (WPF)
- Windows Communication Foundation (WCF)
- LINQ



C# INTEGRATED DEVELOPMENT ENVIRONMENT (IDE)

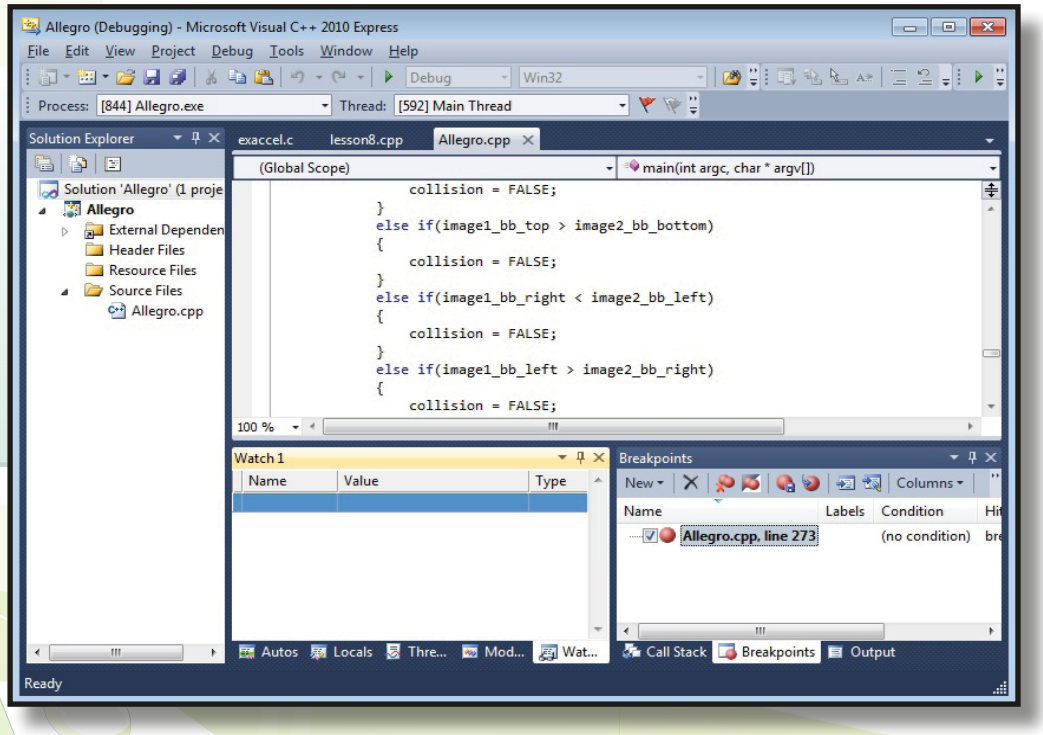
The C# integrated development environment is a host of development tools with a common user interface such as:

- Visual Studio 2010 (VS)
- Visual C# 2010 Express (VCE)
- Visual Web Developer (VWD)

VCE and VWD can be downloaded free from the Microsoft official website. C# is similar to other programming languages in that it lets you write both simple and complex application programs. You can also write codes using a text editor such as Notepad and compile using command-lines. These two editions are compressed versions of Visual Studio and are used mostly by the academic community.

The following are the most important tools and windows in C#.

- **Code editor** to write source code
- **Compiler** to convert source code into an executable program
- **Debugger** to test your program
- **Toolbox and Designer** for rapid development of user interface using a mouse
- **Solution Explorer** to view and manage project files and settings
- **Project Designer** to configure compiler options, deployment paths, resources and more
- **Class View** to navigate source codes
- **Properties Window** to configure controls of properties and events
- **Object Browser** to view the methods and classes available
- **Document Explorer** to browse and search product documentation



C# ON OTHER OPERATING SYSTEMS

If you are using an operating system other than Windows, there are some alternative software that you can use in C# projects, such as Mono. Mono is an open-source version of the .NET framework which includes a C# compiler and runs on several operating systems such as Android, BSD, iOS, Linux, OS X, Windows, Solaris, and UNIX.

C# AND JAVA

According to James Gosling and William Nelson "Bill" Joy, originators of Java programming at Sun Microsystems, C# is an "imitation" of Java. Other authors have said that Java and C# are almost identical programming languages and are boring repetitions that lack innovation. Since the release of C# 2.0, The C# and Java languages have become less similar. They have included additional generics with different implementations. Some of the clear features of C# is its ability to accommodate functional-style programming and a support framework for lambda expressions, extension methods, and anonymous classes.

Vocabulary

Common Language Infrastructure (CLI) – international standard reference in creating the execution and development environments wherein languages and libraries work together

Integrated Development Environment (IDE) – collection of development tools with a common user interface

Mono – open-source version of .NET framework which includes a C# compiler that runs on several operating systems

ASSESSMENT ACTIVITY

1. Conduct a research on the similarities and differences of C, C++, and C# programming languages.

Similarities:

a. _____

b. _____

c. _____

d. _____

e. _____

Differences:

a. _____

b. _____

c. _____

d. _____

e. _____

2. List down and describe some open-source versions of .Net framework. Write down your reference.

a. _____

_____

_____

_____

b. _____

_____

_____

_____

c. _____

_____

_____

_____

Lesson 2

FIRST C# PROGRAM

Content Standards

Learners demonstrate understanding of:

- different parts of the C# program;
- ways to compile and execute a simple program;
- identifiers and keywords;
- variables;
- ways to define variables; and
- ways to initialize variables.

Performance Standard

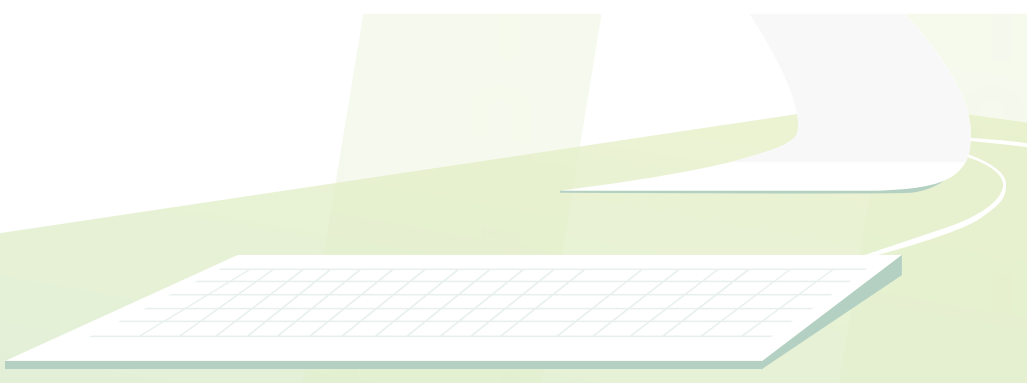
Learners shall be able to:

- familiarize with the different parts of a program, reserve keywords, contextual keywords, and categories of data types.

Learning Outcomes

Learners shall be able to:

- write the first Hello World program using C#; and
- apply the basic parts of a program.



MY FIRST HELLO WORLD

The C# program consists of several parts:

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

Now take a look at the "HELLO WORLD" code below.

```
1 using System;
2
3 namespace HelloWorld
4 {
5       class HelloWorld
6       {
7             public static void Main (string[] args)
8             {
9                   /* my first progam in C# */
10                  Console.WriteLine ("Hello World!");
11                  Console.ReadKey ();
12            }
13      }
14}
```

PARTS OF HELLO WORLD PROGRAM

Line	Parts	Description
1	using System;	**using**, as a keyword, is used to include the **System** namespace in the program A program can include multiple **using** statements.
2	**namespace** declaration	**namespace** is a collection of classes. The *HelloWorld* namespace contains the class *HelloWorld.*
5	**class** declaration	class *HelloWorld* contains the data and method definitions that you use in the program; classes contain multiple methods which define the behavior of the class, but the *HelloWorld class* example above has only one method, wich is **Main**
7	**Main** method	the entry point for all C# programs; states what the class does when executed

9	/*…*/	**comments** in the program are ignored by the compiler; *you can also use the'//' symbol for single-line comments*
10	WriteLine	Main method specifies its behavior with the statement **Console.WriteLine("Hello World!");** **WriteLine** is a method of the *Console* class defined in the *System* namespace which triggers the message "Hello, World!" to be displayed on the screen
11	Console.ReadKey();	Makes the program wait for a key to be pressed and prevents the screen from running and closing quickly when the program is launched

Remember:

- C# is case sensitive.
- All statements and expressions end with a semicolon (;)
- Program execution starts at the Main method
- Program file name could be different from the class name
- Program does not use the global method for Main; methods and variables are not supported at the global level
- Program does not use either ": :" or "->"; the ": :" is not an operator at all, and the "->" operator is used in only a small fraction of programs; the separator "." is used in compound names such as Console.Writeline
- Program does not contain forward declarations
- Program does not use #include to import program text

COMPILING AND EXECUTING MY HELLO WORLD

When your Hello World code is compiled and executed, it produces the following result:

```
Hello World!
```

If you are using Visual Studio.Net for compiling and executing C# programs, do the following steps:

- Launch Visual Studio.
- On the menu bar, choose File -> New -> Project.
- Choose Visual C# from templates, and then choose Windows.
- Choose Console Application.
- Specify a name for your project and click OK.

- Write the code in the Code Editor.
- Click the Run button or press F5 to execute the project. A command prompt window that contains the line, “Hello World!” will appear.

IDENTIFIERS AND KEYWORDS

The identifier is a name used to identify a class, variable, function, or any other user-defined item.

- Must begin with a letter followed by a sequence of letters, digits (0-9) or underscore (_). The first character cannot be a digit.
- Must NOT contain any embedded space or symbol such as ? - + ! @ # % ^ %* ( ) [ ] { } . ; : “ ‘ / and \.

There are identifiers which have a special meaning in the context of the code, such as get and set, which are also called contextual keywords.

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

CONTEXTUAL KEYWORDS						
add	alias	ascending	descending	dynamic	from	get
global	group	into	join	let	orderby	partial (type)
partial (method)	remove	select	set			

VARIABLES

A variable is a name or alias given to a storage area. Every variable is of a specific type that determines the size and layout of the memory.

Below are basic categories of data types:

TYPE	EXAMPLE
Integral	sbyte, byte, short, ushort, int, uint, long, ulong, and char
Floating point	Float and double
Decimal	Decimal
Boolean	true or false, as assigned
Nullable	Nullable data types

You will learn more about the different data types in next chapter.

DEFINING VARIABLES

The syntax for variable definition is

```
<data_type> <variable_list>;
```

`data_type` must be a valid data type including:
`char, int, float, double` or any user-defined data types;
and `variable_list` may consist of one or more identifier names separated by commas.

Examples:

```
int a, c, n;
char von, y;
float y, wage;
double d;
```

INITIALIZING VARIABLES

You initialize or assign value to a variable with an equals sign followed by a constant expression.

The syntax for initializing variable is:

```
<variable_name> = value;
```

You can also initialize variables in their declaration.
Examples:

```
int a = 2, f = 6;
byte n = 17;
char m = 'm';
```

Take a look at the following sample program that uses different variable types:

```
1 using System;
2
3 namespace VariableDefinition
4 {
5       class Program
6       {
7             static void Main (string[] args)
8             {
9                   short a;
10                  int c ;
11                  double n;
12                  /* actual initialization */
13                  a = 18;
14                  c = 23;
15                  n = a + c;
16                  Console.WriteLine ("a = {0}, C = {1}, n = {2}", a, c, n);
17                  Console.ReadKey ();
18            }
19      }
20}
```

Dissection

- Lines 9-11 declare the variables (**a**, **c** and **n**) according to their specific types.
- Lines 13-14 initialize the variables with a specific value (**18** and **23**).
- The value of variable **n** in line 15 is the sum of the value of variables **a** and **c**.

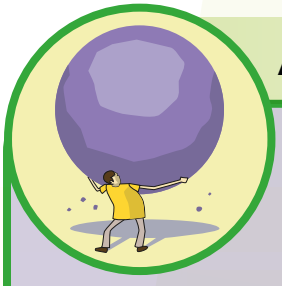
When the code is compiled and executed, it produces the following result:

```
a = 18, c = 23, n = 41
```

Vocabulary

Identifier - name used to identify class, variable, function, or any other user-defined item

Variable - name or alias given to a storage area



ASSESSMENT ACTIVITY

1. Complete the table below.

	C	C++
Ways to compile and execute a program		
Ways to define a variable		
Ways to initialize variable		

2. Recall your programming skills in C or C++ and try to write your first "Hello World" program then compare it with the codes using C#. What are your observations?

Lesson 3

DATA TYPES

Content Standards

Learners demonstrate understanding of:

- value type;
- different reference types (object, dynamic, and string);
- pointer type;
- integer literals;
- floating-point literals;
- character constants;
- string literals;
- defining constant; and
- type conversion methods.

Performance Standard

Learners shall be able to:

- recognize and apply data types in a simple program.

Learning Outcome

Learners shall be able to:

- write a simple program using variables and data types.

VALUE TYPE

Value Type comes from the class `System.ValueType` which directly contains data. When you declare a value type, the system allocates memory to store the value.

Below is a table list of available value types.

TYPE	REPRESENTS	RANGE	DEFAULT VALUE
bool	Boolean value	True or False	False
byte	8-bit unsigned integer	0 to 255	0
char	16-bit Unicode character	U +0000 to U +ffff	'\0'
decimal	128-bit decimal values with 28-29 significant digits	$(-7.9 \times 10^{28}$ to $7.9 \times 10^{28}) / 10^{0}$ to ^{28}	0.0M
double	64-bit double-precision floating point type	$(+/-)5.0 \times 10^{-324}$ to $(+/-)1.7 \times 10^{308}$	0.0D
float	32-bit single-precision floating point type	-3.4×10^{38} to $+ 3.4 \times 10^{38}$	0.0F
int	32-bit signed integer type	-2,147,483,648 to 2,147,483,647	0
long	64-bit signed integer type	-923,372,036,854,775,808 to 9,223,372,036,854,775,807	0L
sbyte	8-bit signed integer type	-128 to 127	0
short	16-bit signed integer type	-32,768 to 32,767	0
uint	32-bit unsigned integer type	0 to 4,294,967,295	0
ulong	64-bit unsigned integer type	0 to 18,446,744,073,709,551,615	0
ushort	16-bit unsigned integer type	0 to 65,535	0

You can get the exact size of a particular type or variable on a particular platform by using the `sizeof` method. This yields the storage size of the object or type in bytes.

Try to study the following sample program that uses the `sizeof` method.

```
1 using System;
2
3 namespace DataTypeApplication
4 {
5       class MainClass
6       {
7             public static void Main (string[] args)
8             {
9                   Console.WriteLine ("Size of int: {0}", sizeof(int));
10                  Console.ReadLine ();
11            }
12      }
13}
```

Dissection

- Lines 9-10 instruct the program to display the storage size of the **`int`** type.

When the code is compiled and executed, it produces the following result:

```
Size of int: 4
```

Complete the table below to get the exact size type of the following value types.

TYPE	SIZE
bool	
byte	
char	
decimal	
double	
float	
long	
sbyte	
short	
uint	
ulong	
ushort	

REFERENCE TYPE

Value Type refers to a memory location. It does not contain the actual data stores in a variable but a reference to the variable.

Object Type

Object Type is a common name for **System.Object** class. You can assign a value to any value type, i.e., reference, predetermined, and user defined. You must convert the type first before assigning values. **Boxing** is called when you convert from value type to object type. On the other hand, **unboxing** is called if you want to convert from object type to value type.

For example:

```
object obj;
obj = 50; // this is boxing conversion
```

Dynamic Type

Dynamic Type is similar to object type but the type checking for object type variable is done at compile time, whereas for the dynamic, it is done at **run time**.

The syntax for declaring a dynamic type is

```
dynamic <variable_name> = value;
```

Example:

```
dynamic a = 16;
```

String Type

String Type is an alias for the System.String class. You can assign the value using string literals in two forms: quoted and @quoted.

Example:

```
String str = “Making learning a great experience”;
@”Making learning a great experience”;
```

POINTER TYPE

Value Type stores the memory address of another type which has similar capabilities in C or C++.

The syntax for declaring a pointer type is

```
type* identifier;
```

Example:

```
char* cptr;
int* iptr;
```

INTEGER LITERALS

An Integer literal can be a decimal, octal, or hexadecimal constant. The **Constant** is a fixed value that you cannot alter during the execution. This fixed value is also called **Literal**.

Prefixes that you can use:

- 0x or 0X for hexadecimal
- 0 for octal
- no prefix for decimal

Suffixes (can be uppercase, lowercase, or in any order) that you can use:

- U for unsigned
- L for long

Examples:

```
143          /* acceptable */
82u          /* acceptable */
0xFEEL       /* acceptable */
058          /* unacceptable: only 0-7 digits are octal numbers */
041UU        /* unacceptable: you cannot repeat a suffix */
26           /* decimal */
013          /* long */
0x1C         /* hexadecimal */
11u          /* unsigned int */
11l          /* long */
11ul         /* unsigned long */
```

FLOATING-POINT LITERALS

A Floating-point literal has an:

- integer part
- decimal point
- fractional part
- exponent part

If want to represent a decimal form, include the decimal point, the exponent, or both, and if you are using the exponential form, you must include the integer part, the fraction part, or both. A signed exponent is represented by e or E.

Examples:

```
16.17
1.617E1       /* equal to 16.17 */
1617e-2       /* equal to 16.17 */
-4.5e-3       /* equal to -0.0045 */
45E-4         /* equal to 0.0045 */
```

CHARACTER CONSTANTS

Character constants are enclosed by single quotes. They can be a plain character ('a'), an escape sequence ('\t'), or a universal character ('uacn12B').

Below are some of the commonly used escape sequence codes:

Escape Sequence	Meaning
\\	\ character
\'	' character
\"	" character
\a	Alert or bell
\b	Backspace
\f	Form feed
\n	Newline
\r	Carriage return
\t	Horizontal tab
\v	Vertical tab
\ooo	Octal number of one to three digits
\xhh …	Hexadecimal number of one or more digits

Example:

```
1 using System;
2
3 namespace EscapeCharacter
4 {
5       class MainClass
6       {
7             public static void Main (string[] args)
8             {
9                   Console.WriteLine ("Making \t learning \n\na great experience!");
10                  Console.ReadLine ();
11            }
12      }
13}
```

Dissection

- Lines 9-10 instruct the program to display the phrase "**Making learning a great experience!**" using the '**\t**' and '**\n**' escape sequences.

When the code is compiled and executed, it produces the following result:

```
Making  learning
a great experience!
```

STRING LITERALS

String literals, which are similar to character literals, are enclosed in double quotes “” or with @ at the start before the opening quote mark.

Example:

```
“hi, kamusta ka?”
“hi, \ kamusta ka?”
“hi, “ “kamusta” “ka?”
@“kamusta ka?”
```

DEFINING CONSTANTS

You define constants using const keyword.

The syntax for defining a constant is

```
const <data_type> <constant_name> = value;
```

Example:

```
1 using System;
2
3 namespace EscapeCharacter
4 {
5       class MainClass
6       {
7             public static void Main (string[] args)
8             {
9                   const double pi = 3.14159; // constant declaration
10                  double r;
11                  Console.WriteLine (“Enter the value for radius: ”);
12                  r = Convert.ToDouble(Console.ReadLine ());
13                  double areaCircle = pi * (r*r);
14                  Console.WriteLine(“Radius: {0}; Area: {1}”, r, areaCircle);
15                  Console.ReadLine ();
16            }
17      }
18}
```

Dissection

- Line 9 defines the variable **`pi`** with a constant value of **`3.14159`**.
- Line 11 instructs the program to display the phrase "**`Enter the value for radius:`** "
- Line 12 asks the user to give a value for radius.
- Line 13 computes the area of a circle and stores the value to **`areaCircle`** variable using the formula pi * (r*r).
- Line 14 and 15 display the value of the radius provided by the user and the area of the circle using a double floating point type.

When the code is compiled and executed, it produces the following result:

```
Enter the value for radius
6
Radius: 6 ; Area: 113.09724
```

TYPE CONVERSION

You use type conversion or type casting if you want to convert one type of data to another type.

- Implicit – performed in a type-safe manner. For example, if you want to convert from smaller to larger integral types and from derived to base classes.
- Explicit – performed by users using pre-defined functions. This requires a cast operator.

Example:

```
1 using System;
2
3 namespace TypeConversionApplication
4 {
5       class MainClass
6       {
7             public static void Main (string[] args)
8             {
9                   double y = 1234.56;
10                  int x;
11
12                  /* cast double to int. */
13                  x = (int)y;
14                  Console.WriteLine (x);
15                  Console.ReadKey ();
16            }
17      }
18}
```

Dissection

- Line 13 uses the explicit type of conversion. The **`double`** type variable **`y`** is converted to **`int`** type. The **`int`** type is enclosed by a cast operator () followed by the variable **`y`**.
- Line 14 displays the new value of **`x`** which is converted to a whole number.

When the code is compiled and executed, it produces the following result:

```
1234
```

TYPE CONVERSION METHODS

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

Example of value type to string type conversion.

```
1 using System;
2
3 namespace TypeConversionApplication
4 {
5       class MainClass
6       {
7             public static void Main (string[] args)
8             {
9
10                  int a = 48;
11                  float b = 61.029f;
12                  double c = 1234.567;
13                  bool d = true;
14
15                  Console.WriteLine (a.ToString());
16                  Console.WriteLine (b.ToString());
17                  Console.WriteLine (c.ToString());
18                  Console.WriteLine (d.ToString());
19                  Console.ReadKey ();
20            }
21      }
22}
```

Dissection

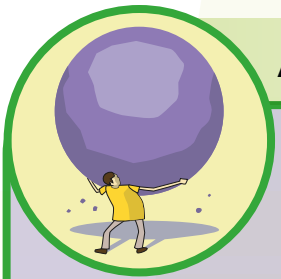
- Lines 10-13 define **a,b,c** and **d** variables with different data types and initial values.
- Lines 15-18 convert all the variables to string character type.

When the code is compiled and executed, it produces the following result:

```
48
61.029
1234.567
True
```

Vocabulary

Dynamic type - similar to object type but the type checking is done at run time
Integer literals - can be a decimal, octal, or hexadecimal constant
Object type - alias for System.Object class
Pointer type - stores the memory address of another type
Reference type - refers to a memory location
String type - alias for System.String class



ASSESSMENT ACTIVITY

1. Write a program that will accept user details as name, city, age, and birthday, then store all the values in the appropriate variable. Print all the information in the correct format.
2. Write a program that will accept and display student information such as student's name, age, and section and display them.

Lesson 4

OPERATORS

Content Standards

Learners demonstrate understanding of:

- different arithmetic operators;
- different relational operators;
- different logical operators;
- different bitwise operators;
- different assignment operators; and
- operator precedence.

Performance Standard

Learners shall be able to:

- use the different operators supported by C# properly.

Learning Outcome

Learners shall be able to:

- write a program using C# operators.

ARITHMETIC OPERATORS

An operator is a symbol that tells the compiler to perform specific mathematical or logical manipulations.

Table below shows all the arithmetic operators supported by C#.

Operator	Description	Example (C = 40, N = 50)
+	Adds two operands	C + N = 90
-	Subtracts second operand from the first	N – C = 10
*	Multiplies both operands	C * N = 2000
/	Divides numerator by the denominator	N / C = 1.25
%	Modulus operator	N % C = 10
++	Increment operator	++C = 41
--	Decrement operator	--N = 49

Example:

```
1 using System;
2
3 namespace OperatorsApplication
4 {
5       class MainClass
6       {
7             public static void Main (string[] args)
8             {
9                   int a = 5;
10                  int c = 10;
11                  int n;
12
13                  n = a + c;
14                  Console.WriteLine ("1st value of n is {0}", n);
15                  n = c - a;
16                  Console.WriteLine ("2nd value of n is {0}", n);
17                  n= a * c;
18                  Console.WriteLine ("3rd value of n is {0}", n);
19                  n = c / a;
20                  Console.WriteLine ("4th value of n is {0}", n);
21                  n = c % a;
22                  Console.WriteLine ("5th value of n is {0}", n);
23                  n = ++c;
24                  Console.WriteLine ("6th value of n is {0}", n);
25                  n = --c;
26                  Console.WriteLine ("7th value of n is {0}", n);
27                  Console.ReadLine ();
28            }
29      }
30}
```

Dissection

- Lines 13-14 compute the sum and display the result of two integer variables, **a** and **c**.
- Lines 15-16 compute the difference and display the result of two integer variables, **c** and **a**.
- Lines 17-18 compute the product and display the result of two integer variables, **a** and **c**.
- Lines 19-20 compute the quotient and display the result of two integer variables, **c** and **a**.
- Lines 21-22 compute the module and display the result of two integer variables, **c** and **a**.
- Lines 23-24 increase the value of **c** by 1 and display the result.
- Lines 25-26 decrease the value of **c** by 1 and display the result.

When the code is compiled and executed, it produces the following results:

```
1st value of n is 15
2nd value of n is 5
3rd value of n is 50
4th value of n is 2
5th value of n is 0
6th value of n is 11
7th value of n is 9
```

RELATIONAL OPERATORS

The table below shows all the relational operators supported by C#.

Operator	Description	Example (C = 40, N = 50)
==	Checks if the values of two operands are equal or not, and if yes, then the condition becomes true	(A == B) is false.
!=	Checks if the values of two operands are equal or not, and if values are not equal, then the condition becomes true	(A != B) is true.

>	Checks if the value of the left operand is greater than the value of the right operand, if yes, then the condition becomes true	(A > B) is false.
<	Checks if the value of the left operand is less than the value of the right operand, and if yes, then the condition becomes true	(A < B) is true.
>=	Checks if the value of the left operand is greater than or equal to the value of the right operand, and if yes, then the condition becomes true	(A >= B) is false.
<=	Checks if the value of the left operand is less than or equal to the value of the right operand, and if yes, then the condition becomes true	(A <= B) is true.

Example:

```
1 using System;
2
3 namespace OperatorsApplication
4 {
5       class MainClass
6       {
7             public static void Main (string[] args)
8             {
9                   int a = 5;
10                  int c = 10;
11
12                  if (a == c)
13                  {
14                        Console.WriteLine (“1st value --> a is equal to c”);
15                  }
16                  else
17                  {
18                        Console.WriteLine (“1st value --> a is not equal to c”);
19                  }
20                  if (a < c )
21                  {
22                        Console.WriteLine (“2nd value --> a is less than c”);
23                  }
24                  else
25                  {
26                        Console.WriteLine (“2nd value --> a is not less than c”);
27                  }
28                  Console.ReadLine ();
29            }
30      }
31}
```

Dissection

- Lines 12-18 use a simple **`if`**...**`else`** condition to test if the value of variable **`a`** is equal to **`c`**. If the statement satisfies, then the program will display "**`1st value --> a is equal to c`**" otherwise, "**`1st value --> a is not equal to c`**".
- Lines 20-27 use a simple **`if`**...**`else`** condition to test if the value of variable **`a`** is less than **`c`**. If the statement satisfies, then the program will display "**`2nd value --> a is less than c`**" otherwise, "**`2nd value --> a is not less than c`**".

When the code is compiled and executed, it produces the following results:

```
1st value --> a is not equal to c
2nd value --> a is less than c
```

LOGICAL OPERATORS

The table below shows all the logical operators supported by C#.

Operator	Description	Example (A = TRUE, N = FALSE)
&&	Logical AND operator. If both the operands are non-zero, then the condition becomes true.	(A && B) is false.
||	Logical OR Operator. If any of the two operands is non-zero, then the condition becomes true.	(A || B) is true.
!	Logical NOT Operator. If a condition is true, then the Logical NOT operator will make it false.	!(A && B) is true.

Example:

```
1 using System;
2
3 namespace OperatorsApplication
4 {
5       class MainClass
6       {
7             public static void Main (string[] args)
8             {
9                   bool A = true;
10                  bool N = true;
11
12                  if (A && N)
13                  {
14                        Console.WriteLine ("1st value --> Condition is TRUE");
15                  }
16                  if (A || N )
17                  {
18                        Console.WriteLine ("2nd value --> Condition is TRUE");
19                  }
20                  Console.ReadLine ();
21            }
22      }
23}
```

Dissection

- Lines 9-10 declare the variables **A** and **N** as Boolean with an initial value of **true**.
- Lines 12-15 use a simple **if** condition to test if the statement will produce a **true** value using an AND (&&) operator and display "**1st value --> Condition is TRUE**".
- Lines 16-19 use a simple **if** condition to test if the statement will produce a **true** value using an OR (||) operator and display "**2nd value --> Condition is TRUE**".

When the code is compiled and executed, it produces the following results:

```
1st value --> Condition is TRUE
2nd value --> Condition is TRUE
```

BITWISE OPERATORS

Bitwise operators work on bits and perform bit by bit operation (0's and 1's).

Table below shows all the bitwise operators supported by C#.

Operator	Description	Example (C = 60, N = 13) C = 0011 1100 N = 0000 1101
&	The Binary AND Operator copies a bit to the result if it exits in both operands.	(C & N) = 12 which is 0000 1100
|	The Binary OR Operator copies a bit if it exits in either operand.	(C | N) = 61 which is 0011 1101
^	The Binary XOR Operator copies the bit if it is set in one operand, but not both.	(C ^ N) = 49 which is 0011 0001
~	The Binary Ones Complement Operator is unary and has the effect of 'flipping' bits.	(~C) = 61 which is 1100 0011 (in 2's complement due to a signed binary number)
<<	Binary Left Shift Operator. The left operand's value is moved left by the number of bits specified by the right operand.	C << 2 = 240 which is 1111 0000
>>	Binary Right Shift Operator. The left operand's value is moved right by the number of bits specified by the right operand.	C >> 2 = 15 which is 0000 1111

Example:

```
1 using System;
2
3 namespace OperatorsApplication
4 {
5       class MainClass
6       {
7             public static void Main (string[] args)
8             {
9                   int C = 60;
10                  int N = 13;
11                  int A = 0;
12
13                  A = C & N;
14                  Console.WriteLine ("1st value of A is {0}", A);
15                  A = C | N;
16                  Console.WriteLine ("2nd value of A is {0}", A);
17                  A= C ^ N;
18                  Console.WriteLine ("3rd value of A is {0}", A);
19                  A = ~C;
20                  Console.WriteLine ("4th value of A is {0}", A);
21                  A = C << 2;
22                  Console.WriteLine ("5th value of A is {0}", A);
23                  A = C >> 2;
24                  Console.WriteLine ("6th value of A is {0}", A);
25                  Console.ReadLine();
26            }
27      }
28}
```

Dissection

- Line 13 compares the value of variables **C** and **N** using an AND operator.
- Line 14 displays "**1st value of A is 12**" which is equivalent to **0000 1100** in binary.
- Line 15 compares the value of variables **C** and **N** using an OR operator.
- Line 16 displays "**2nd value of A is 61**" which is equivalent to **0011 1101** in binary.
- Line 17 compares the value of variables **C** and **N** using an XOR operator.
- Line 18 displays "**3rd value of A is 49**" which is equivalent to **0011 0001** in binary.
- Line 19 finds the value of **A** using a Binary Ones Complement operator to the value of variable **C**.
- Line 20 displays "**4th value of A is -61**" which is equivalent to **1100 0011** in 2's complement due to a signed binary number.
- Line 21 finds the value of **A** using a Binary Left Shift operator to the value of variable **C**.
- Line 22 displays "**5th value of A is 240**" which is equivalent to **1111 0000** in binary.
- Line 23 finds the value of **A** using a Binary Right Shift operator to the value of variable **C**.
- Line 24 displays "**6th value of A is 15**" which is equivalent to **0000 1111** in binary.

When the code is compiled and executed, it produces the following results:

```
1st value of A is 12
2nd value of A is 61
3rd value of A is 49
4th value of A is -61
5th value of A is 240
6th value of A is 15
```

ASSIGNMENT OPERATORS

The table below shows all assignment operators supported by C#.

Operator	Description	Example
=	Assigns values from right side operands to left side operands	V = O + N assigns value of O + N into V
+=	Adds the right operand to the left operand and assigns the result to the left operand	V += O is equivalent to V = V + O
-=	Subtracts the right operand from the left operand and assigns the result to the left operand	V -= O is equivalent to V = V – O
*=	Multiplies the right operand with the left operand and assigns the result to the left operand	V *= O is equivalent to V = V * O
/=	Divides the left operand with the right operand and assigns the result to the left operand	V /= O is equivalent to V = V / O
%=	Takes the modulus using two operands and assigns the result to the left operand	V %= O is equivalent to V = V % O
<<=	Left shift AND assignment operator	V <<= 2 is same as V = V << 2
>>=	Right shift AND assignment operator	V >>= 2 is same as V = V >> 2
&=	Bitwise AND assignment operator	V &= 2 is same as V = V & 2
^=	Bitwise exclusive OR and assignment operator	V ^= 2 is same as V = V ^ 2
|=	Bitwise inclusive OR and assignment operator	V |= 2 is same as V = V | 2

Example:

```
1 using System;
2
3 namespace OperatorsApplication
4 {
5       class MainClass
6       {
7             public static void Main (string[] args)
8             {
9                   int V = 35;
10                  int O;
11
12                  O = V;
13                  Console.WriteLine ("= Value of O is {0}", O);
14                  O += V;
15                  Console.WriteLine ("+= Value of O is {0}", O);
16                  O -= V;
17                  Console.WriteLine ("-= Value of O is {0}", O);
18                  O *= V;
19                  Console.WriteLine ("*= Value of O is {0}", O);
20                  O /= V;
21                  Console.WriteLine ("/= Value of O is {0}", O);
22
23                  O = 150;
24                  O %= V;
25                  Console.WriteLine ("%= Value of O is {0}", O);
26                  O <<= 2;
27                  Console.WriteLine ("<<= Value of O is {0}", O);
28                  O >>= 2;
29                  Console.WriteLine (">>= Value of O is {0}", O);
30                  O &= 2;
31                  Console.WriteLine ("&= Value of O is {0}", O);
32                  O ^= 2;
33                  Console.WriteLine ("^= Value of O is {0}", O);
34                  O |= 2;
35                  Console.WriteLine ("|= Value of O is {0}", O);
36                  Console.ReadLine ();
37            }
38      }
```

Dissection

- Line 12 assigns the value of variable **v** to the value of variable **o**.
- Line 13 displays "**= value of o is 35**".
- Line 14 adds the value of variable **v** to the value of variable **o** and assigns the result to the value of variable **o**.
- Line 15 displays "**+= value of o is 70**".
- Line 16 subtracts the value of variable **v** from the value of variable **o** and assigns the result to the value of variable **o**.
- Line 17 displays "**-= value of o is 35**".

- Line 18 multiplies the value of variable **v** by the value of variable **o** and assigns the result to the value of variable **o**.
- Line 19 displays "***= value of o is 1225**".
- Line 20 divides the value of variable **o** by the value of variable **v** and assigns the result to the value of variable **o**.
- Line 21 displays "**/= value of o is 35**".
- Line 24 takes modules using the values of the variables **o** and **v** and assigns the result to the value of variable **o**.
- Line 25 displays "**%= value of o is 10**".
- Line 26 uses left shift AND the assignment operator and assigns the result to the value of variable **o**.
- Line 27 displays "**<<= value of o is 40**".
- Line 28 uses right shift AND the assignment operator and assigns the result to the value of variable **o**.
- Line 29 displays "**<<= value of o is 10**".
- Line 30 uses bitwise AND assignment operator and assigns the result to the value of variable **o**.
- Line 31 displays "**&= value of o is 2**".
- Line 32 uses bitwise exclusive OR and the assignment operator and assigns the result to the value of variable **o**.
- Line 33 displays "**^= value of o is 0**".
- Line 34 uses bitwise inclusive OR the assignment operator and assigns the result to the value of variable **o**.
- Line 35 displays "**|= value of o is 2**".

When the code is compiled and executed, it produces the following results:

```
= value of o is 35
+- value of o is 70
-= value of o is 35
*= value of o is 1225
/= value of o is 35
%= value of o is 10
<<= value of o is 40
>>= value of o is 10
&= value of o is 2
^= value of o is 0
¦= value of o is 2
```

OTHER OPERATORS

The table below shows other operators supported by C#.

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

Example:

```
1 using System;
2
3 namespace OperatorsApplication
4 {
5       class MainClass
6       {
7             public static void Main (string[] args)
8             {
9                   /* Example of sizeof Operator */
10                  Console.WriteLine ("The size of int is {0}", sizeof(int));
11                  Console.WriteLine ("The size of int is {0}", sizeof(long));
12                  Console.WriteLine ("The size of int is {0}", sizeof(double));
13
14                  /* Example of ternary Operator ?:*/
15                  int c, n;
16                  c = 25;
17                  n = (c == 13) ? 35 : 45;
18                  Console.WriteLine ("The value of n is {0}", n);
19                  n = (c == 25) ? 35 : 45;
20                  Console.WriteLine ("The value is {0}", n);
21                  Console.ReadLine();
22            }
23      }
24}
```

Dissection

- Line 10 returns and displays the size of the integer data type.
- Line 11 returns and displays the size of the long data type.
- Line 12 returns and displays the size of the double data type.
- Line 17 states that if the value of variable **c** (which is **25**) is equal to **13**, then the value of **n** is **35**, otherwise **n** is **45**.
- Line 18 displays "**The value of n is 45**".
- Line 19 states that if the value of variable **c** (which is **25**) is equal to **25**, then the value of n is **35**, otherwise **n** is **45**.
- Line 20 displays "**The value of n is 35**".

When the code is compiled and executed, it produces the following results:

```
The size of int is 4
The size of int is 8
The size of int is 8
The value of n is 45
The value is 35
```

OPERATOR PRECEDENCE

Operator precedence determines the grouping of terms in an expression.

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

Example

```
1 using System;
2
3 namespace OperatorsApplication
4 {
5       class MainClass
6       {
7             public static void Main (string[] args)
8             {
9                   int a = 5;
10                  int v = 10;
11                  int o = 40;
12                  int n = 20;
13                  int c;
14                  c = (a * v) +  o / n;
15                  Console.WriteLine ("The value of c = (a * v) + o / n is {0}", c);
16                  c = a * (v +  o) / n;
17                  Console.WriteLine ("The value of c = a * (v + o) / n is {0}", c);
18                  c = (a * v) +  (o / n);
19                  Console.WriteLine ("The value of c = (a * v) + (o / n) is {0}", c);
20                  Console.ReadLine();
21            }
22      }
23}
```

Dissection

- Line 14 gets the product of variables **a** and **v**, the quotient of variables **o** and **n**, then finds the sum.
- Line 15 displays "**The value of c = (a*v)+o/n is 52**".
- Line 16 gets the sum of variables **v** and **o**, then multiplies by variable **a**, then divides by variable **n**.
- Line 17 displays "**The value of c = a*(v+o)/n is 12**".
- Line 18 gets the product of variables **a** and **v**, the quotient of variable **o** and **n**, then finds the sum.
- Line 19 displays "**The value of c = (a*v)+(o/n) is 52**".

When the code is compiled and executed, it produces the following results:

```
The value of c = (a * v) + o / n is 52
The value of c = a * (v + o) / n is 12
The value of c = (a * v) + (o / n) is 52
```

Vocabulary

arithmetic operator - symbol that tells the compiler to perform specific mathematical or logical manipulations

ASSESSMENT ACTIVITY

1. Write a program to calculate $(a+c)^2$.
2. Write a program to check whether input number is even or odd.
3. Write a program to calculate the area of a triangle. Take the required input from the user and then calculate the area of triangle.
4. Write a program that can:
   - add two numbers.
   - subtract two numbers.
   - multiple two numbers.
   - divide two numbers.
5. Write a program that will:
   - increment an integer by 2.
   - decrement an integer by 1.
6. Write a program that will accept two numbers and tell whether they are:
   - greater than.
   - less than.
   - equal to.

Lesson 5

CONTROL STRUCTURES

Content Standard

Learners demonstrate understanding of:

- if statement;
- if … else statement;
- if … else if … else statement;
- nested if statement;
- switch statement;
- loop statement;
- while loop;
- for loop;
- do … while loop; and
- break statement.

Performance Standard

Learners shall be able to:

- use different control structures according to specific needs

Learning Outcome

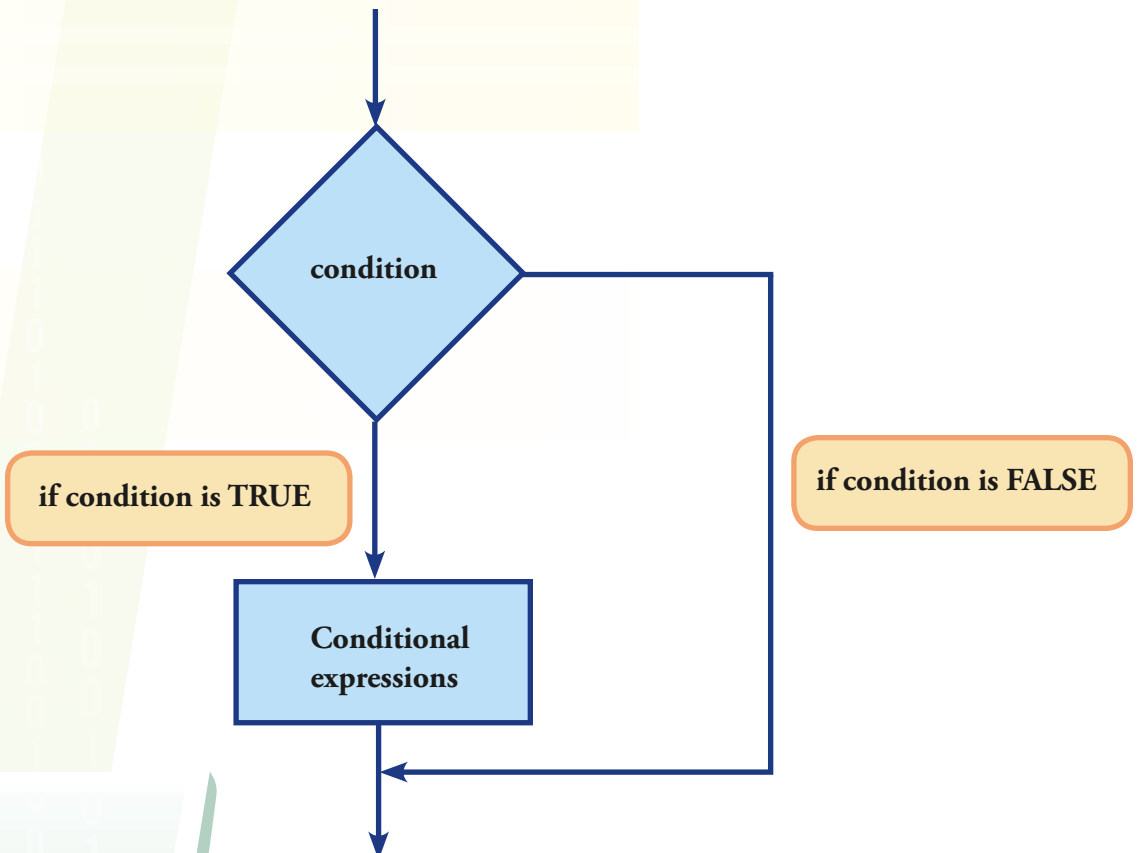
Learners shall be able to:

- write a program using control structures

CONTROL STRUCTURES

Control Structures require that you to specify one or more conditions to be evaluated or tested.

Below is the general form of a typical decision making structure.



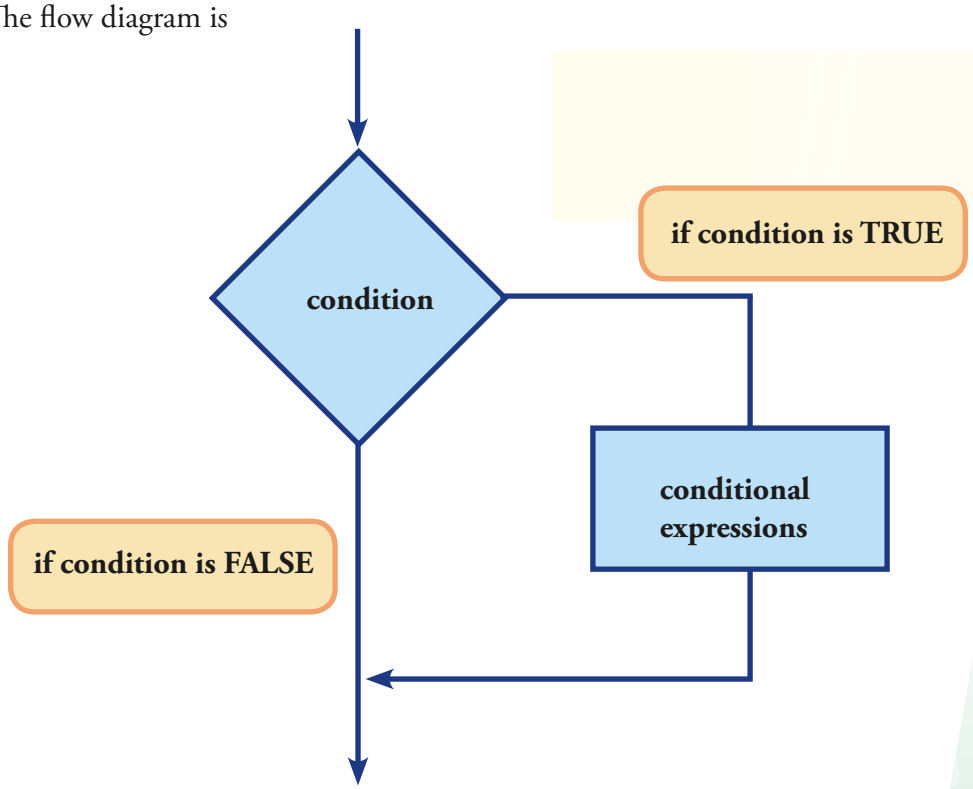
if Statement

The **if** statement consists of a Boolean expression followed by one or more statements.

The syntax of an **if** statement is

```
if(Boolean_expression)
{
      /*statement(s) will be executed if the expression is TRUE*/
}
```

The flow diagram is



Example:

```
1 using System;
2
3 namespace OperatorsApplication
4 {
5       class MainClass
6       {
7             public static void Main (string[] args)
8             {
9                   int a = 15;
10                  if (a < 25)
11                  {
12                        Console.WriteLine ("a is less than 25;");
13                  }
14                  Console.WriteLine ("The value of a is : {0}", a);
15                  Console.ReadLine();
16            }
17      }
18}
```

Dissection

- Line 10 tests the condition.
- Line 12 displays "**`a is less than 25;`**" if the condition of line 10 is TRUE.
- Line 14 displays "**`The value of a is : 15`**".

When the code is compiled and executed, it produces the following results:

```
a is less than 25
The value of a is : 15
```

If … else Statement

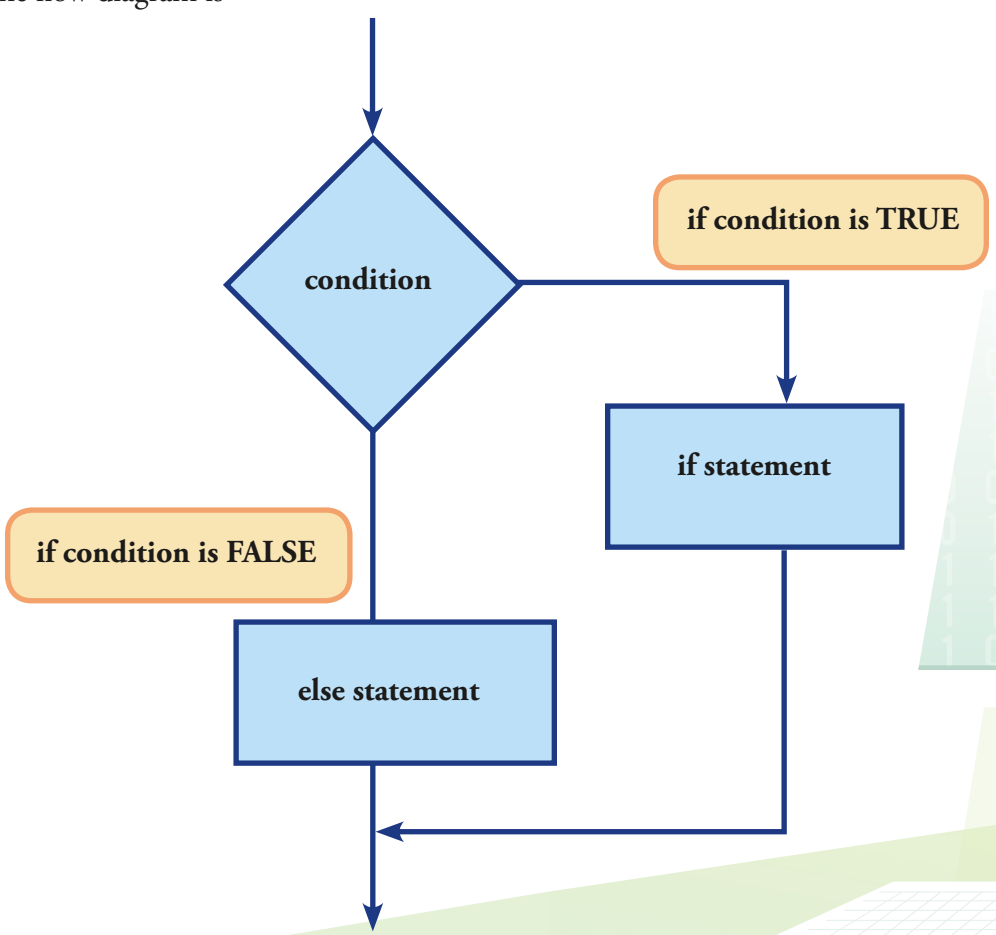
If … else is used if you want to execute a statement when the boolean expression is FALSE.

If the Boolean expression evaluates to **TRUE**, then the **if block** is executed, otherwise, the **else block** is executed.

The syntax of an **if … else** statement is

```
if(Boolean_expression)
{
      /*statement(s) will be executed if the expression is TRUE*/
}
else
{
      /*statement(s) will be executed if the expression is FALSE*/
```

The flow diagram is



Example:

```
1 using System;
2
3 namespace DecisionMaking
4 {
5       class MainClass
6       {
7             public static void Main (string[] args)
8             {
9                   int a = 80;
10                  if (a < 25)
11                  {
12                        Console.WriteLine ("a is less than 25;");
13                  }
14                  else
15                  {
16                        Console.WriteLine ("a is not less than 25;");
17                  }
18                  Console.WriteLine ("The value of a is : {0}", a);
19                  Console.ReadLine();
20            }
21      }
22}
```

Dissection

- Line 10 tests the condition.
- Line 12 displays "**`a is less than 25;`**" if the condition in line 10 is TRUE.
- Line 16 displays "**`a is not less than 25;`**" if the condition in line 10 is FALSE.
- Line 18 displays "**`The value of a is : 80`**" once the **`if ... else`** statement exits.

When the code is compiled and executed, it produces the following results:

```
a is not less than 25;
The value of a is : 80
```

If … else if … else Statement

If … else if … else is used if you want to test several conditions using a single if statement.

- An **if** can have zero or one **else** and it must come after any **else if**.
- An **if** can have zero and above number of **else if***s* and they must come before the **else**.
- Once an **else if** satisfies, none of the remaining **else if***s* or **else***s* will be tested.

The syntax of an **if … else if … else** statement is

```
if(boolean_expression 1)
{
      /*executes when the boolean expression 1 is TRUE*/
      }
      else if(boolean_expression 2)
      {

      /*executes when the boolean expression 2 is TRUE*/
      }
      else
      {
      /*executes when none of the boolean expressions is TRUE*/
}
```

Example:

```
1 using System;
2
3 namespace DecisionMaking
4 {
5       class MainClass
6       {
7             public static void Main (string[] args)
8             {
9                   int c = 100;
10
11                  if (c == 40)
12                  {
13                        Console.WriteLine ("The value of c is 40");
14                  }
15                  else if (c == 60)
16                  {
17                        Console.WriteLine ("The value of c is 60);
18                  }
19                  else if (c == 80)
20                  {
21                        Console.WriteLine ("The value of c is 80");
22                  }
23
24                  else
25                  {
26                        Console.WriteLine ("Value does not match");
27                  }
28                  Console.WriteLine ("The value of c is : {0}", c);
29                  Console.ReadLine();
30            }
31      }
32}
```

Dissection

- Line 11 tests the condition, if the value of **c** is equal to **40**.
- Line 13 displays "**The value of c is 40**" if the condition in line 11 is TRUE.
- Line 15 states the condition, if the value of **c** is equal to **60**.
- Line 17 displays "**The value of c is 60**" if the condition in line 15 is TRUE.
- Line 19 states the condition, if the value of **c** is equal to **80**.
- Line 21 displays "**The value of c is 80**" if the condition in line 19 is TRUE.
- Line 26 displays "**Value does not match**" if all the conditions in lines 11, 15, 19 are all FALSE.

- Line 28 displays "**`The value of c is 100`**" once the **`if ... else if ... else`** statement exits.

When the code is compiled and executed, it produces the following results:

```
Value does not match
The value of c is : 100
```

Nested if Statement

You can use one **if** or **else ... if** statement inside another **if** or **else if** statement(s).

The syntax of a **nested if** statement is

```
if(boolean_expression 1)
{
      /*executes when the boolean expression 1 is TRUE*/

      if(Boolean_expression 2)
      {
      /*executes when the boolean expression 2 is TRUE*/

      }
}
```

Example:

```
1 using System;
2
3 namespace DecisionMaking
4 {
5       class MainClass
6       {
7             public static void Main (string[] args)
8             {
9                   int c = 100;
10                  int n = 150;
11
12                  if (c == 100)
13                  {
14                        if (n == 150)
15                        {
16                              Console.WriteLine ("The value of c is 100 and n is 150");
17                        }
18                  }
19                  Console.WriteLine ("The exact value of c is: {0}", c);
20                  Console.WriteLine ("The exact value of n is: {0}", n);
21                  Console.ReadLine();
22            }
23      }
24}
```

Dissection

- Line 12 tests the condition, if the value of **c** is equal to 100.
- Line 14 tests the condition, if the value of **n** is equal to 150.
- Line 16 displays "**The value of c is 100 and n is 150**" if the condition in line 14 is TRUE.
- Line 19 displays "**The exact value of c is 100**" if the condition either in line 12 or 14 is FALSE.
- Line 20 displays "**The exact value of n is 150**" if the condition either in line 12 or 14 is FALSE.

When the code is compiled and executed, it produces the following results:

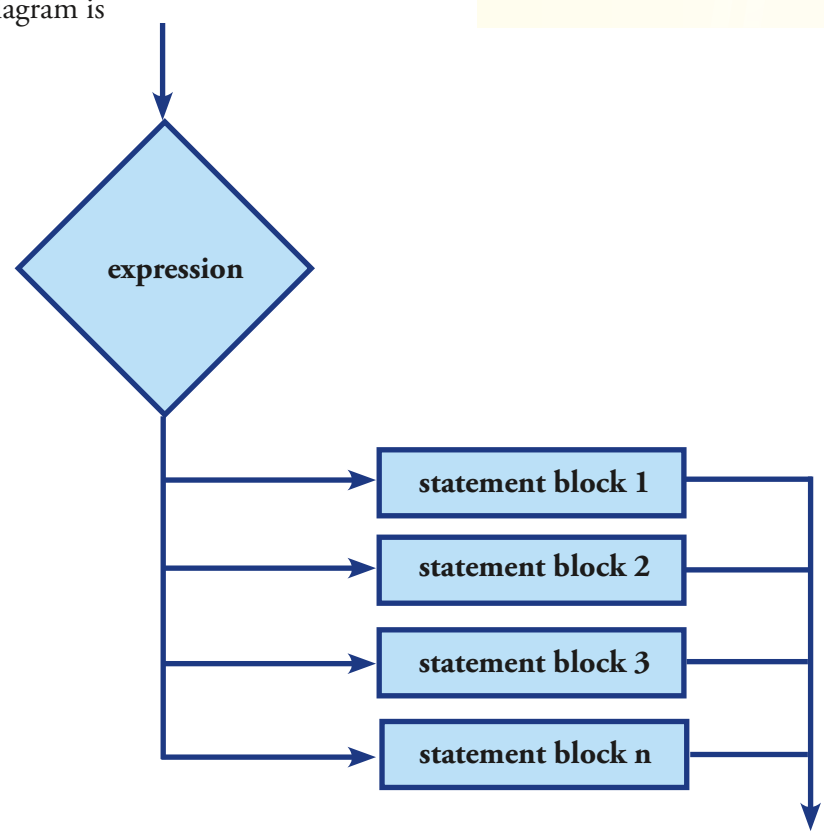
```
The value of c is 100 and n is 150
The exact value of c is: 100
The exact value of n is: 150
```

Switch Statement

A **switch** statement enables the testing of a variable to verify that it agrees with the different value or case lists.

- **Expression** used must have an integer or enumerated types.
- You can have more than one case statement within a switch followed by the value to be compared and a colon.
- **Constant-expression** must be constant or literal and have the same data type as the variable for each case.
- When the switched-on variable satisfies the case, the succeeding statements will be executed until they reach the **break** statement
- When a break statement is reached, the flow of control of the terminated switch statement will proceed in the next line.
- Not every case needs to contain a break.
- A default case for a switch statement is an option, but it must be placed at the end of the switch statement/s if one is used.

The flow diagram is



The syntax of a **switch** statement is

```
switch(expression)
{
      case constant-expression:
            statement(s);
            break;            /*optional*/
      case constant-expression:
            statement(s);
            break;
      default:                /*optional*/
            statement(s);
}
```

Example:

```
1 using System;
2
3 namespace DecisionMaking
4 {
5       class MainClass
6       {
7             public static void Main (string[] args)
8             {
9                   char grade = 'B' ;
10
11                  switch (grade)
12                  }
13                  case 'A':
14                        Console.WriteLine ("Outstanding!");
15                        break;
16                  case 'B':
17                  case 'C':
18                        Console.WriteLine ("Very Good!");
19                        break;
20                  case 'D':
21                        Console.WriteLine ("Good!");
22                        break;
23                  default:
24                        Console.WriteLine ("Invalid Grade");
25                        break;
26                  }
27                  Console.WriteLine ("Your grade is: {0}", grade);
28                  Console.ReadLine();
29            }
30      }
31}
```

Dissection

- Line 11 reads the value of **`grade`** variable.
- Line 13 reads the expression.
- Line 14 displays "**`Outstanding!`**" if the expression in line 13 satisfies.
- Line 16 and 17 read the expression.
- Line 18 displays "**`Very Good!`**" if the expression in line 16 or 17 satisfies.
- Line 20 reads the expression.
- Line 21 displays "**`Good!`**" if the expression in line 20 satisfies.
- Line 24 displays the "**`Invalid Grade`**" optional statement.
- Line 27 displays "**`Your grade is B`**" once the switch statement exists.

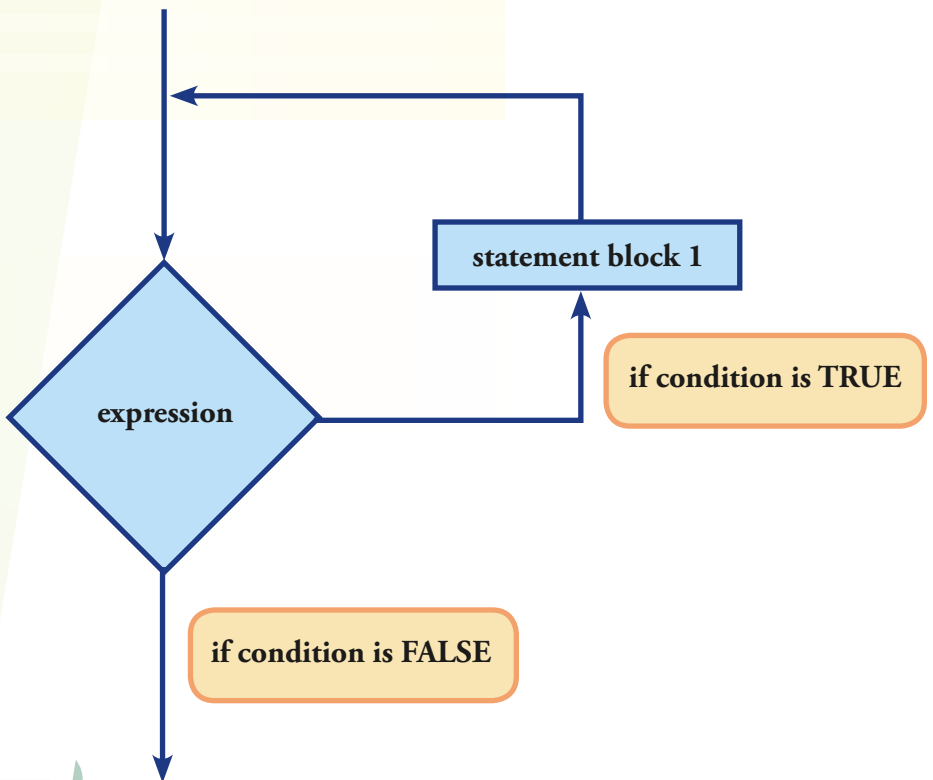
When the code is compiled and executed, it produces the following results:

```
Very Good!
Your grade is B
```

LOOP STATEMENT

A **loop statement** allows you to execute a statement or group of statements several times.

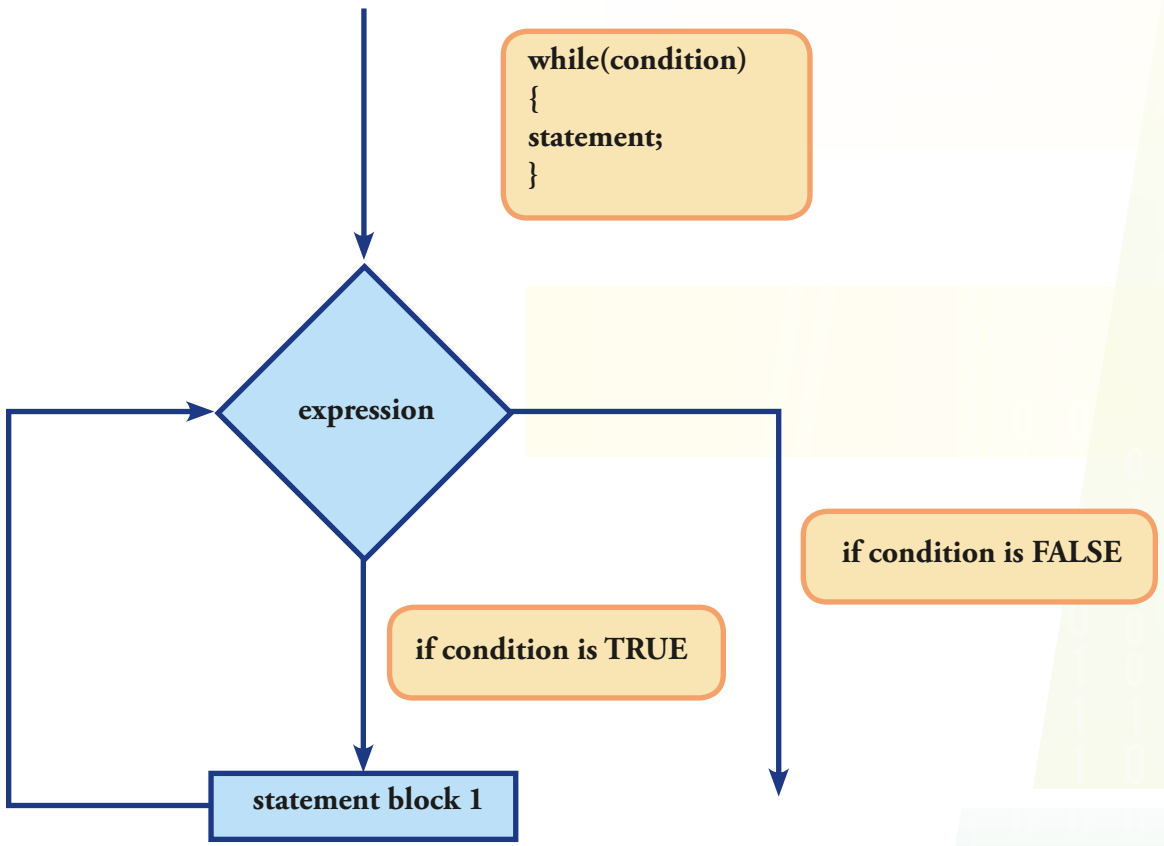
Below is the flow diagram.



While Loop

A **while loop** statement allows you to execute a target statement as long as a given condition is TRUE.

Below is the flow diagram.



The syntax of while loop statement is

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

Example:

```
1 using System;
2
3 namespace Loops
4 {
5       class MainClass
6       {
7             public static void Main (string[] args)
8             {
9                   int c = 15;
10                  while (c < 20)
11                  {
12                        Console.WriteLine ("The value of c: {0}", c);
13                        c++;
14                  }
15                  Console.ReadLine();
16            }
17      }
18}
```

Dissection

- Line 10 tests the condition.
- Line 12 displays "**The value of c: 15**". Line 13 increments the value of **c** by
  - c = 16; display "**The value of c: 16**" on the next line; increments the value of **c** by 1.
  - c = 17; display "**The value of c: 17**" on the next line; increments the value of **c** by 1.
  - c = 18; display "**The value of c: 18**" on the next line; increments the value of **c** by 1.
  - c = 19; display "**The value of c: 19**" on the next line; increments the value of **c** by 1.
- The loop ends if the condition in line 10 is already FALSE.

When the code is compiled and executed, it produces the following results:

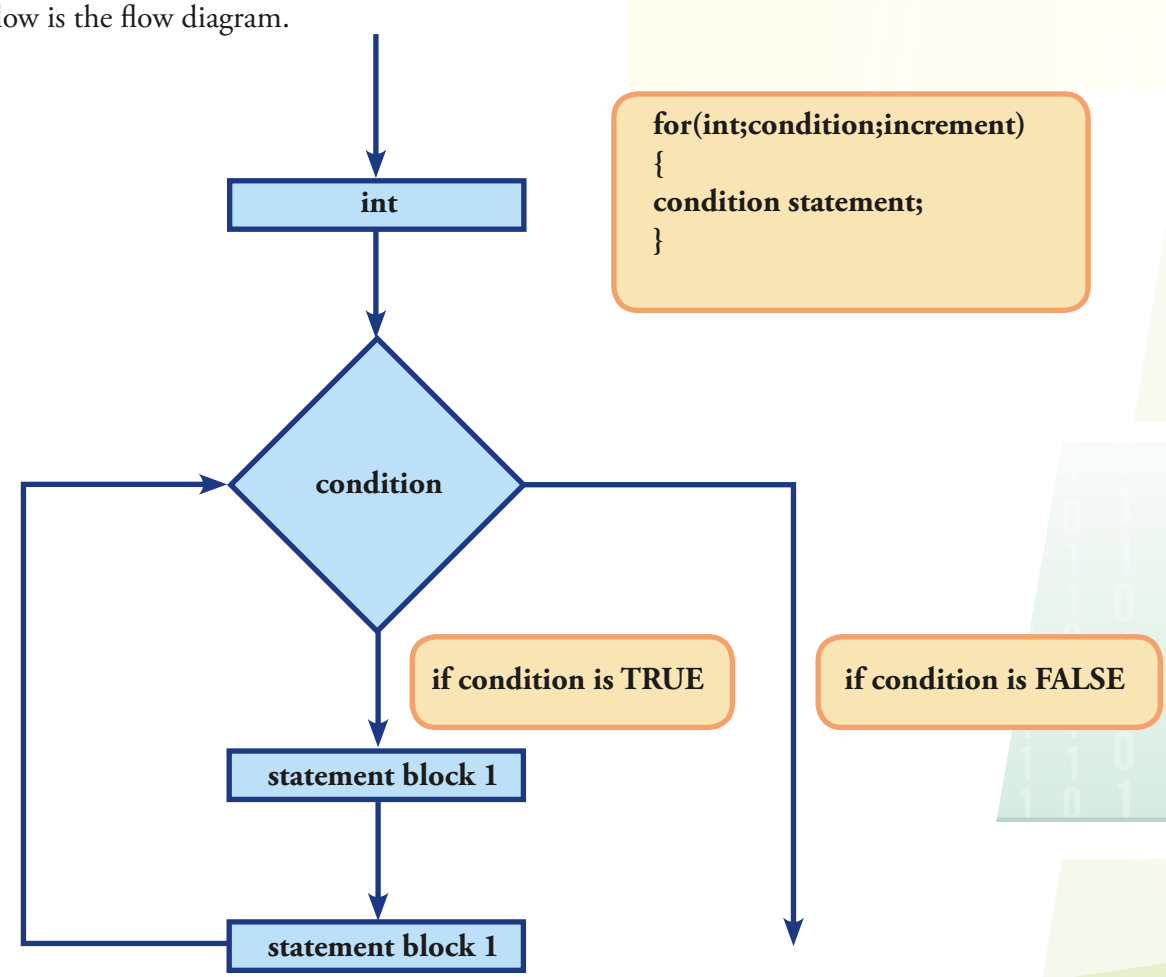
```
The value of c : 15
The value of c : 16
The value of c : 17
The value of c : 18
The value of c : 19
```

For Loop

A **for loop** is a structure meant for repetition control and must execute a defined number of times before the instruction proceeds to another statement.

- The **int** step is executed first. This allows you to declare and initialize any loop control variable. If there is a semicolon, you do not need to put a statement.
- Next the **condition** is evaluated. If it is TRUE, the body of the loop is executed, else, the body of the loop does not execute and flow of control jumps to the next statement just after the for loop.
- After the initial execution of the for loop statement, the flow of control will return to the increment statement which serves to increase the value of the counter to a defined numeric value.
- The for loop is terminated if the condition is FALSE, otherwise it will be re-evaluated.

Below is the flow diagram.



The syntax of for loop statement is

```
for (int; condition; increment)
{
      Statement(s);
}
```

Example:

```
1 using System;
2
3 namespace DecisionMaking
4 {
5       class MainClass
6       {
7             public static void Main (string[] args)
8             {
9                   for (int n = 25; n < 50; n = n+1)
10                  {
11                        Console.WriteLine ("The value of n : {0}", n);
12                  }
13                  Console.ReadLine();
14            }
15      }
16}
```

Dissection

- Line 9 tests the condition.
  - **`n = 25`** (integer type declaration)
  - **`25 < 50`** (TRUE)
  - **`n = 25 + 1`** (the new value of n is **`26`**)
- Line 11 displays "**`The value of n is 26`**".
- Go back to line 9 to test the condition again.
  - **`26< 50`** (TRUE)
  - **`N = 26+1`** (the new value of n is **`27`**)
  - "**`The value of n is 27`**" displays on the next line
- The loop ends if the condition in line 9 is already FALSE.

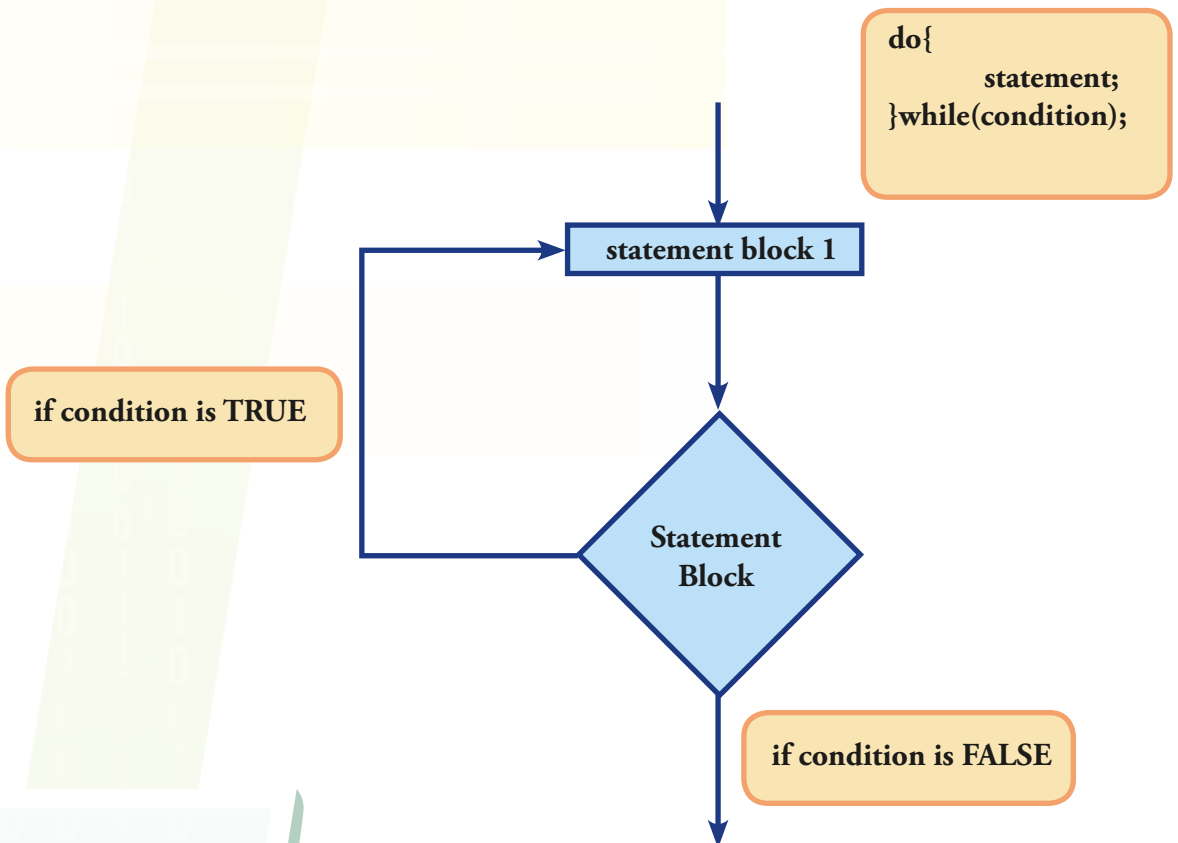
When the code is compiled and executed, it produces the following results:

```
The value of n : 26
The value of n : 27
The value of n : 28
The value of n : 29
The value of n : 30
The value of n : 31
The value of n : 32
The value of n : 33
The value of n : 34
The value of n : 35
The value of n : 36
The value of n : 37
The value of n : 38
The value of n : 39
The value of n : 40
The value of n : 41
The value of n : 42
The value of n : 43
The value of n : 44
The value of n : 45
The value of n : 46
The value of n : 47
The value of n : 48
The value of n : 49
```

Do … While Loop

A **do … while loop** is similar to the while loop, except that it is guaranteed to execute at least one time.

Below is the flow diagram.



The syntax of do... while statement is

```
do
{
      Statement(s);
} while(condition);
```

Example:

```
1 using System;
2
3 namespace DecisionMaking
4 {
5       class MainClass
6       {
7             public static void Main (string[] args)
8             {
9                   int d = 15;
10                  do {
11                        Console.WriteLine ("The value of d : {0}", d);
12                        d = d + 1;
13                  } while (d < 25);
14                        Console.ReadLine();
15            }
16      }
17}
```

Dissection

- Line 10-13 shows the **do ... while loop** statement
  - Line 11 displays "**The value of d : 15**".
  - Line 12 increments the value of **d** by **1** which makes it 16.
  - While the value of **d** is less than 25, the program will continue to display characters similar to line 11 with the new value of **d**.
  - The loop ends if the condition in 14 is already FALSE.

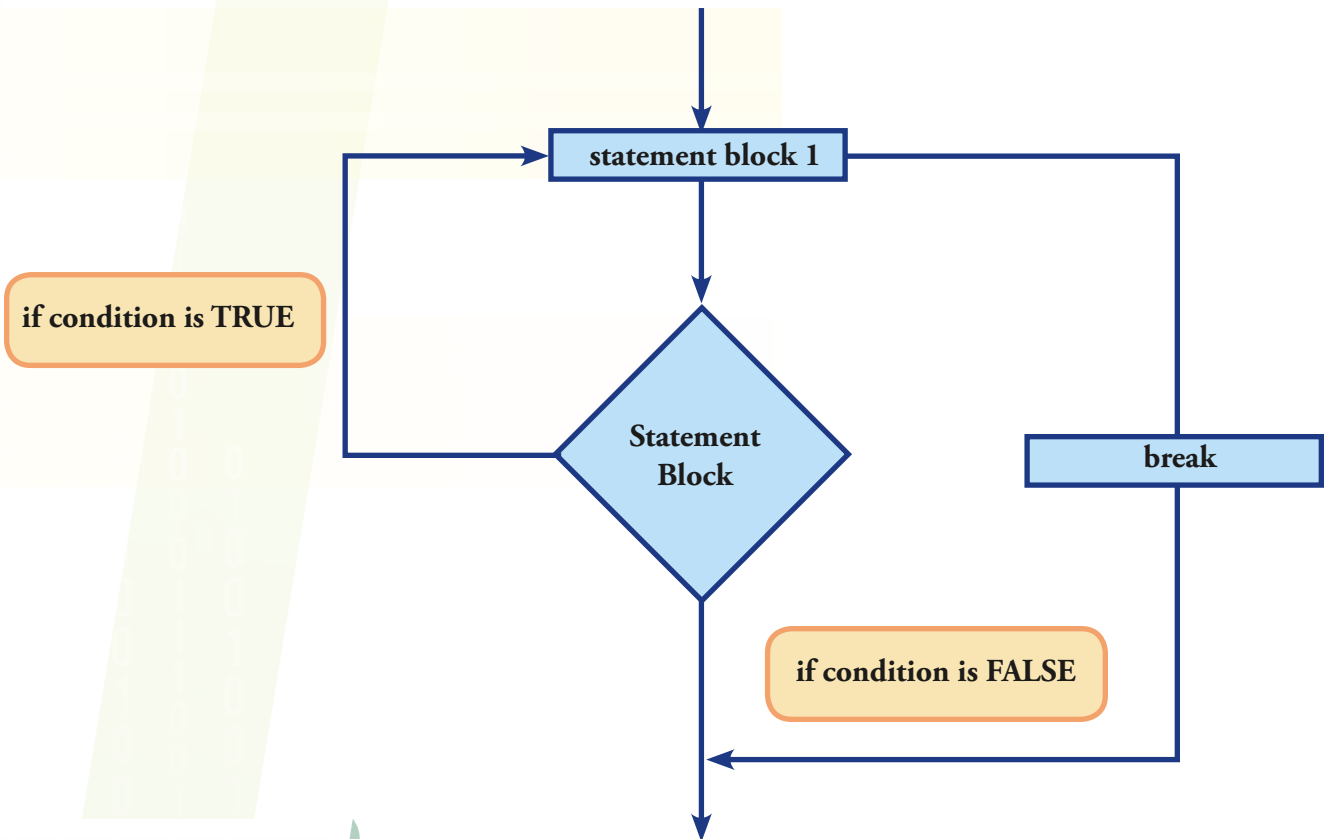
When the code is compiled and executed, it produces the following results:

```
The value of c : 15
The value of c : 16
The value of c : 17
The value of c : 18
The value of c : 19
The value of c : 20
The value of c : 21
The value of c : 22
The value of c : 23
The value of c : 24
```

Break Statement

A loop is immediately terminated and program control resumes in the next statement following the loop. You can use this statement in the switch statement.

Below is the flow diagram.



The syntax of a **break** statement is

```
break;
```

Example:

```
1 using System;
2
3 namespace DecisionMaking
4 {
5       class MainClass
6       {
7             public static void Main (string[] args)
8             {
9                   int d = 15;
10                  while (d < 25)
11                  {
12                        Console.WriteLine ("The value of d : {0}", d);
13                        d++;
14                        if (d > 20)
15                        {
16                              break;
17                        }
18                  }
19                  Console.ReadLine ();
20            }
21      }
22}
```

Dissection

- Line 16 shows the break statement.
  - The statement is executed if the condition in line 14 is already FALSE.

When the code is compiled and executed, it produces the following results:

```
The value of d : 15
The value of d : 16
The value of d : 17
The value of d : 18
The value of d : 19
The value of d : 20
```

Vocabulary

break statement - loop is terminated immediately and the program resumes in the next statement following the loop
control structure - requires you to specify one or more conditions to be evaluated or tested
do … while loop - similar to the while loop, except that it is guaranteed to execute at least one time
for loop - used to write a loop that needs to execute a specific number of times
if statement - consists of a Boolean expression followed by one or more statements
if ... else statement - used to execute a statement when the Boolean expression is FALSE
if ... else ... if statement - used to test several conditions using a single if statement
loop statement - used to execute a statement or group of statements several times
switch statement - allows a variable to be tested for equality against a list of values
while loop - used to execute a target statement as long as a given condition is TRUE

ASSESSMENT ACTIVITY

1. Write a program for tax calculation. Accepted money shall be used as input from the user. Calculate the tax using the following pattern.

Money	Percentage	Total Tax
Less than 10,000	5%	
10,000 to 100,000	8%	
More than 100,000	8.5%	

2. Write a program for a library system which will display the user option and be based on an option that shows the available book.

Option:

w – Computer books

x – Mathematics books

y – Science books

z – English books

If the user inputs the incorrect value, then the program should show them the message: Try again!

Lesson 6

OBJECT-ORIENTED PROGRAMMING

Content Standards

Learners demonstrate understanding of:

- public access;
- private access;
- base and derive classes;
- static polymorphism; and
- dynamic polymorphism.

Performance Standard

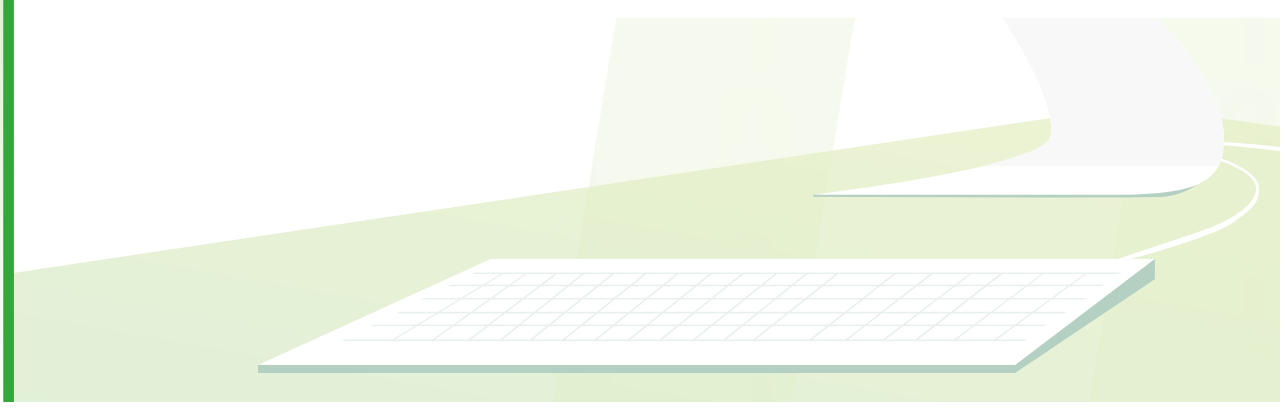
Learners shall be able to:

- apply the protection of data from accidental corruption.

Learning Outcome

Learners shall be able to:

- write a program that applies the concept of encapsulation, inheritance, and polymorphism.



ENCAPSULATION

Encapsulation is an object-oriented programming methodology that groups data and functions into a single class. It provides a different way to protect data from accidental corruption.

Public Access

This allows a class to expose its member variables and functions to other functions and objects. Any public member can be accessed from outside the class.

For example:

```
1 using System;
2
3 namespace RectangleApplication
4 {
5       class Rectangle
6       {
7               public double length;
8               public double width;
9               public double area()
10              {
11                      return length * width;
12              }
13              public void Display ()
14              {
15                      Console.WriteLine ("The length is: {0}", length);
16                      Console.WriteLine ("The width is: {0}", width);
17                      Console.WriteLine ("The area is: {0}", area());
18              }
19      }
20      class FindTheArea
21      {
22              static void Main(String[] args)
23              {
24                      Rectangle p = new Rectangle ();
25                      p.length = 5.6;
26                      p.width = 2.5;
27                      p.Display();
28                      Console.ReadLine();
29              }
30      }
31}
```

Dissection

- In the example, the member variables **`length`** (line 7) and **`width`** (line 8) are declared **`public`**, so they can be accessed from the function **`Main ()`** using an instance of the **`Rectangle`** class, named **`p`**.
- The member functions **`Display()`** and **`area()`** can also access these variables directly without using any instance of the class.
- The member function **`Display()`** is also declared **`public`**, so it can also be accessed from **`Main()`** using an instance of the **`Rectangle`** class, named **`p`**.

When the code is compiled and executed, it produces the following results:

```
The length is: 5.6
The width is: 2.5
The area is: 14
```

Private Access

This allows a class to hide its member variables and member functions from other functions and objects. Only functions of the same class can access thier private members. Even an instance of a class cannot access its private members.

For example:

```
1 using System;
2
3 namespace RectangleApplication
4 {
5       class Rectangle
6       {
7               private double length;
8               private double width;
9
10              public void GetDetails()
11              {
12                      Console.WriteLine ("Enter length value: ");
13                      length = Convert.ToDouble (Console.ReadLine());
14                      Console.WriteLine ("Enter width value: ");
15                      width = Convert.ToDouble (Console.ReadLine());
16              }
17                      public double area()
18                      {
19                              return length * width;
20                      }
21                      public void Display()
22              {
23                      Console.WriteLine ("The length is: {0}", length);
24                      Console.WriteLine ("The width is: {0}", width);
25                      Console.WriteLine ("The area is: {0}", area());
26              }
27              }
28
29      class FindTheArea
30      {
31              static void Main(String[] args)
32              {
33                      Rectangle p = new Rectangle ();
34                      p.GetDetails();
35                      p.Display();
36                      Console.ReadLine();
37              }
38      }
39}
```

Dissection

- In the example, the member variables **`length`** (line 7) and **`width`** (line 8) are declared **`private`**, so they cannot be accessed from the function **`Main()`**.
- The member functions **`GetDetails()`** and **`Display()`** can access these variables.
- Since the member functions **`GetDetails()`** and **`Display()`** are declared public, they can be accessed from **`Main()`** using an instance of the **`Rectangle`** class, named **`p`**.

When the code is compiled and executed, it produces the following results:

```
Enter length value:
45
Enter width value:
23
The length is: 45
The width is: 23
The area is: 1035
```

INHERITANCE

Inheritance is another important concept in object-oriented programming. It allows you to determine a class in terms of another class. When creating a class, you can designate that the new class should inherit the members of an existing class. The existing class is called **base** and the new class is **derived.**

Base and Derived Classes

A class can be derived from one or more classes or interface.

The syntax in creating derived classes is

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

For example:

```
1 using System;
2
3 namespace InheritanceApplication
4 {
5       class Shape
6       {
7               public void setwidth(int w)
8               {
9                       width = w;
10              }
11              public void setHeight(int h)
12              {
13                      height = h;
14              }
15              protected int width;
16              protected int height;
17      }
18      //Derived Class
19      class Rectangle: Shape
20      {
21              public int getarea()
22              {
23                      return width * height;
24              }
25      }
26      class TestRectangle
27      {
28              static void Main(string[] args)
29              {
30                      Rectangle acn = new Rectangle ();
31                      acn.setwidth (9);
32                      acn.setheight (6);
33                      Console.WriteLine ("The total area: {0}", acn.getarea());
34                      Console.ReadLine();
35              }
36      }
37}
```

Dissection

- Lines 15-16 use the **`protected`** member access modifier. Protected member is accessible within its class and derived class instances.
- Class **`Rectangle`** (line 19) is derived from **`Shape`** (line 5). You can access the protected members of the base class directly from the derived class.

When the code is compiled and executed, it produces the following results:

```
The total area: 54
```

POLYMORPHISM

Polymorphism is one of the most important concepts in object-oriented programming together with encapsulation and inheritance. Polymorphism has a couple of unique attributes:

- During run time, objects from a derived class can also be handled as objects from a base class in code structure such as method parameters, collections, and arrays.
- The code assigns the method at run-time and the Common Language Runtime (CLR) declares the type of run-time for the object and calls up the virtual method override.

Example:

- Teacher A behaves with his students
- Teacher A behaves with his superiors

Teacher A is an object but the attitude in a different in a different situation.

A method or property can perform multiple actions that vary in the context of polymorphism, but it all depends on the type of run-time that uses the instance.

Static or Compile Time Polymorphism

Static polymorphism links a function with an object during compile time.

Example:

```
1 using System;
2
3 namespace PolymorphismApplication
4 {
5       class PrintData
6       {
7             void print(int m)
8             {
9                   Console.WriteLine ("Printing int: {0}", m);
10            }
11            void print(double n)
12            {
13                  Console.WriteLine ("Printing float: {0}", n);
14            }
15            void print(string x)
16            {
17                  Console.WriteLine ("Printing int: {0}", x);
18            }
19            static void Main(String[] args)
20            {
21                  PrintData y = new PrintData ();
22                  y.print (6);
23                  y.print (123.45);
24                  y.print ("Mabuhay Pilipinas");
25                  Console.ReadKey ();
26            }
27      }
28}
```

Dissection

- The example uses method overloading where a class, **`PrintData`** (line 5), can have more than one method with the same name which is **`print`**, and different parameters.
- The compiler checks the type and number of parameters passed on to the method and decides which method to call at compile time. It will give an error if there are no methods that match the method signature of the method that is called at compile time.
- In the example,
  - lines 7-10 match with line 22.
  - lines 11-14 match with line 23.
  - lines 15-18 match with line 24.

When the code is compiled and executed, it produces the following results:

```
Printing int: 6
Printing float: 123.45
Printing string: Mabuhay Pilipinas
```

Dynamic or Runtime Polymorphism

Dynamic polymorphism allows you to create abstract classes that are used to provide partial class implementation of an interface.

Example:

```
1 using System;
2
3 namespace PolymorphismApplication
4 {
5       abstract class Shape
6       {
7             public abstract int area();
8       }
9       class Rectangle: Shape
10      {
11            private int length;
12            private int width;
13            public Rectangle (int z=0, int y=0)
14            {
15                  length =  z;
16                  width =  y;
17            }
18            public override int area ()
19            {
20                  Console.WriteLine ("Rectangle class area:");
21                  return width * length;
22            }
23      }
24      class TestRectangle
25      {
26            static void Main(String[] args)
27            {
28                  Rectangle o = new Rectangle (5, 9);
29                  double z = o.area();
30                  Console.WriteLine ("area: {0}", z);
31                  Console.ReadKey ();
32            }
33      }
34}
```

Dissection

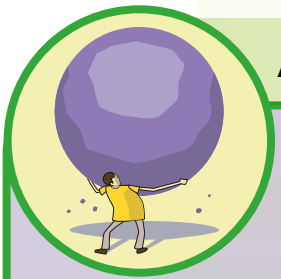
- Line 5 uses abstract classes which contains an abstract method and are implemented by the derived class.
  - You cannot create an instance of an abstract class
  - You cannot declare an abstract method outside an abstract class

When the code is compiled and executed, it produces the following results:

```
Rectangle class area:
Area: 45
```

Vocabulary

Encapsulation - a methodology that covers data and functions into a single class
Inheritance - used to determine a class in terms of another class
Polymorphism - expressed as one interface with multiple functions



ASSESSMENT ACTIVITY

1. Write a program that will explain:
   a. public access
   b. private access
   c. inheritance
   d. polymorphism

Lesson 7

CLASSES

Content Standards

Learners demonstrate understanding of:

- classes;
- member functions and encapsulation;
- constructors; and
- destructors.

Performance Standard

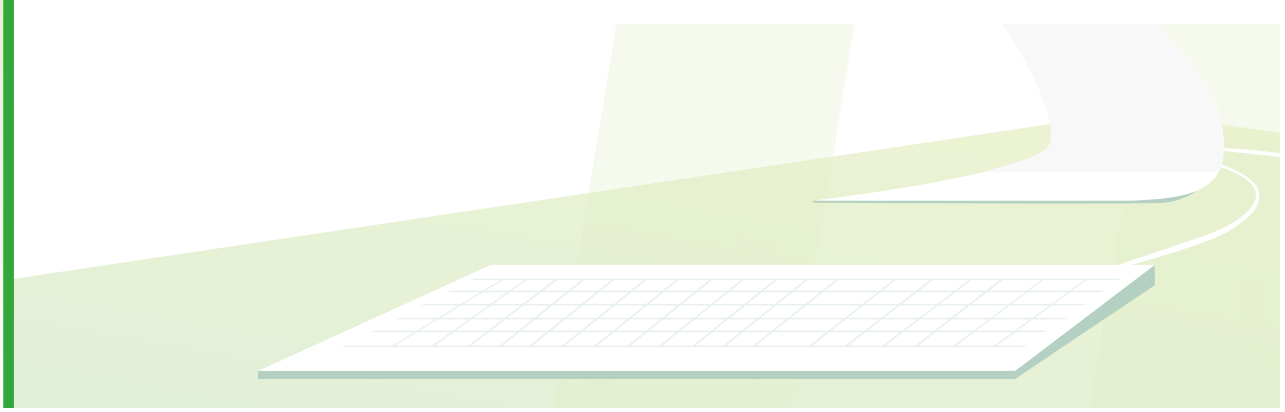
Learners shall be able to:

- apply the importance of code reusability and security.

Learning Outcome

Learners shall be able to:

- write a program that applies the concept of classes, encapsulation, constructors, and destructors.



DEFINING A CLASS

Class is a blueprint of a data type. It defines the meaning of the class. You start defining a class with the keyword class followed by the class name; and the body is enclosed by a pair of curly braces.

- **Access specifiers** specify the access rule for the members as well as the class itself.
- **Data type** specifies the type of variable, and **return type** specifies the data type of the data the method returns.
- To access the class member, use the dot (.) operator.
- The dot operator links the name of an object with the name of a member.

The syntax in defining a class is

```
<accessspecifier> class class_name
{
        //member variable

        <accessspecifier><data type> variable 1;

        <accessspecifier><data type> variable 2;

        . . .

        <accessspecifier><data type> variable n;

        //member methods

        <accessspecifier><return type> method 1(parameter_list)
        {
                //method body
        }
        <accessspecifier><return type> method 2(parameter_list)
        {
                //method body
        }
        . . .
        <acessspecifier><return type> method n(parameter_list)
        {
                //method body
        }
}
```

Example:

```
1 using System;
2
3 namespace BoxApplication
4 {
5       class Box
6       {
7               public double length;
8               public double breadth;
9               public double height;
10      }
11      class TestBox
12      {
13              static void Main(String[] args)
14              {
15                      Box Boxa = new Box ();
16                      Box Boxb = new Box ();
17                      double volume = 0.0;
18
19                      Boxa.height = 7.0;
20                      Boxa.length = 15.0;
21                      Boxa.breadth = 7.0;
22
23                      Boxb.height = 11.0;
24                      Boxb.length = 13.0;
25                      Boxb.breadth = 14.0;
26
27                      volume = Boxa.height * Boxa.length * Boxa.breadth;
28                      Console.WriteLine ("The volume of Box A is : {0}", volume);
29
30                      volume = Boxb.height * Boxb.length * Boxb.breadth;
31                      Console.WriteLine ("The volume of Box B is : {0}", volume);
32                      Console.ReadKey ();
33              }
34      }
35}
```

Dissection

- Line 7 specifies the length of a box.
- Line 8 specifies the breadth of a box
- Line 9 specifies the height of a box
- Line 15 declares **Boxa** of type Box
- Line 16 declares **Boxb** of type Box
- Line 17 stores the volume of a Box
- Lines 19-21 specify the **Boxa** specifications
- Lines 23-25 specify the **Boxb** specifications
- Lines 27-28 specify the volume of **Boxa**
- Lines 29-30 specify the volume of **Boxb**

When the code is compiled and executed, it produces the following results:

```
The value of Box A is : 735
The value of Box B is : 2002
```

MEMBER FUNCTIONS AND ENCAPSULATION

A **member function** of a class is a function that has its own definition within the class definition. It operates and has access to all object components of the class where it is also belongs.

Member variables are private attributes of an object that are made accessible through the public member functions for encapsulation implementation purposes.

Through encapsulation, a class can hide the internal details. Variables, properties, and methods can be made private to prevent from unnecessary accessibility.

Example:

```
1 using System;
2
3 namespace BoxApplication
4 {
5       class Box
6       {
7               public double length;
8               public double breadth;
9               public double height;
10              public void setLength(double leng)
11              {
12              length = leng;
13              }
14              public void setBreadth(double brea)
15              {
16              breadth = brea;
17              }
18              public void setHeight(double heig)
19              {
20              height = heig;
21              }
22              public double getVolume()
23              {
24              return length = breadth * height;
25              }
26      }
27      class TestBox
28      {
29              static void Main(String[] args)
30              {
31                      Box Boxa = new Box ();
32                      Box Boxb = new Box ();
33                      double volume;
34
35                      Boxa.height(7.0);
36                      Boxa.length(15.0);
37                      Boxa.breadth(7.0);
38
39                      Boxb.height(11.0);
40                      Boxb.length(13.0);
41                      Boxb.breadth(14.0);
42
43                      volume = Boxa.getVolume();
44                      Console.WriteLine ("The volume of Box A is : {0}", volume);
45
46                      volume = Boxb.getVolume();
47                      Console.WriteLine ("The volume of Box B is : {0}", volume);
48                      Console.ReadKey ();
49              }
50      }
51}
```

Dissection

- Lines 10-25 use the get and set methods (**`setLength`**, **`setBreadth`**, **`setHeight`**, and **`getVolume`**) to return length, breadth, and height. You use private variables (lines 7-9) which are not directly accessible. To use these variables, you use the get and set methods.

When the code is compiled and executed, it produces the following results:

```
The volume of Box A is : 735
The volume of Box B is : 2002
```

CONSTRUCTORS

A class **constructor** is a special class function executed whenever you create new objects of that class. It has the same name as the class and it does not have any return type.

Example:

```
1 using System;
2
3 namespace LineApplication
4 {
5       class Line
6       {
7             public double length;
8             public Line()
9             {
10                  Console.WriteLine ("Your line is being created");
11            }
12            public void setLength(double leng)
13            {
14                  length = leng;
15            }
16            public double getLength ()
17            {
18                  return length;
19            }
20            static void Main(string[] args)
21            {
22                  Line line = new Line ();
23                  line.setLength(10.0);
24                  Console.WriteLine ("The length of line is: {0}", line.getLength());
25                  Console.ReadKey();
26            }
27      }
28}
```

Dissection

- Line 22 uses **`Line`** as a constructor similar to that of the class **`Line`**.

When the code is compiled and executed, it produces the following results:

```
Your lane is being created
The length of line is: 10
```

On the other hand, the default constructor does not have any parameter. But if you need to have a parameter, you can use **parameterized constructors**, which help you to assign a value to an object at the time of its creation.

Example:

```
1 using System;
2
3 namespace LineApplication
4 {
5       class Line
6       {
7             public double length;
8             public Line(double leng)
9             {
10                  Console.WriteLine ("Your line is being created, length = {0}",  leng);
11                  length = leng;
12            }
13            public void setLength(double leng)
14            {
15                  length = leng;
16            }
17            public double getLength ()
18            {
19                  return length;
20            }
21            static void Main(string[] args)
22            {
23                  Line line = new Line (25.0);
24                  Console.WriteLine ("The length of line is: {0}", line.getLength());
25                  line.setLength(15.0)
26                  Console.WriteLine ("The length of line is: {0}", line.getLength());
27                  Console.ReadKey();
28            }
29      }
30}
```

Dissection

- Line 8 is used to parameterized constructor in which a value is assigned to an object.

When the code is compiled and executed, it produces the following results:

```
Your line is being crated, length = 25
The length of line is: 25
The length of line is: 15
```

DESTRUCTORS

A class **destructor** is another special class member function that runs when an object goes out of its intended range. It has the same name as that of the class with a prefixed tilde (~) and can either return a value or take any parameter. It can be used for releasing memory resources before leaving the programs. Destructors cannot be inherited or overloaded.

Example:

```
1 using System;
2
3 namespace LineApplication
4 {
5       class Lin
6       {
7             private double length;
8             public Lin
9             {
10                  Console.WriteLine ("Your line is being created");
11            }
12
13
14            public void setLength(double leng)
15            {
16                  length = leng;
17            }
18            public double getLength ()
19            {
20                  return length;
21            }
22
23            static void Main(string[] args)
24            {
25                  Lin line = new Lin ();
26                  line.setLength(6.0);
27                  Console.WriteLine ("The length of line is: {0}", line.getLength());
28            }
29            ~Lin()
30            {
31                  Console.WriteLine ("Your line is being deleted");
32                  Console.ReadLine();
33            }
34      }
35}
```

Dissection

- Line 29 uses `~Lin()` as the destructor.
- You do not have any control on when the destructor is going to be executed because this is determined by the Garbage Collector.
- The Garbage Collector checks for objects that are no longer being used by the application.
- Destructors are called program exists.

When the code is compiled and executed, it produces the following results:

```
Your line is being created
The length of line is: 6
Your line is being deleted
```

Vocabulary

class - blueprint of data type
constructor - special class function executed whenever new objects are created
destructor - used to execute when an object of its class goes out of scope
member function - function that has its own definition within the class definition

ASSESSMENT ACTIVITY

1. Write a constructor program in which you make 3 constructors.
   a. default constructors with default message
   b. parameterized constructor which accepts a string value
   c. parameterized constructor which accepts two numerical values and will show the sum

   Initialize all constructors and show the output.

Lesson 8

METHODS

Content Standards

Learners demonstrate understanding of:

- defining methods;
- calling methods;
- recursive method call;
- passing parameters to a method;
- reference parameters; and
- output parameters.

Performance Standard

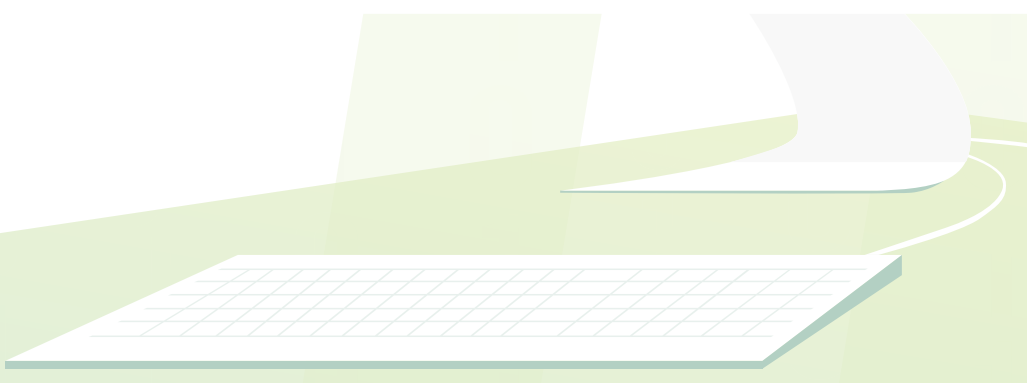
Learners shall be able to:

- apply the importance of methods and parameters.

Learning Outcome

Learners shall be able to:

- write a program that applies the concept methods and parameters.



DEFINING METHODS

A method is a group of statements that perform a given task. In defining a method, you declare the elements of its structure.

- **Access specifier** determines the visibility of a variable or a method from another class.
- **Return type** is the data type of the value the method returns. If the method is not returning any values, then the return type is void.
- **Method name** is a unique identifier and case sensitive. It cannot be similar to any other identifier declared in the class.
- **Parameter list** is enclosed in parentheses. Parameters are used to pass and receive data from method. Parameters can refer to type, order, and number of the parameter of a method.
- **Method body** contains set of instructions needed to complete a given task.

Example:

```
1 using System;
2
3 namespace ManipulatingNumbers
4 {
5       class ManipulatingNumbers
6       {
7               public int MN(int number1, int number2)
8               int result;
9
10              if (number1 > number2)
11                  result = number1;
12              else
13                  result = number2;
14              return result;
15      }
16      ...
17 }
```

CALLING METHODS

You can call a method using the name of the method.

Example:

```
1 using System;
2
3 namespace Calculator
4{
5       class Calculator
6       {
7               public int MN(int number1, int number2)
8               {
9                       int result;
10
11                      if (number1 > number2)
12                              result = number1;
13                      else
14                              result = number2;
15                      return result;
16              }
17      static void Main(string[] args)
18              {
19              int c = 60;
20              int n = 70;
21              int von;
22              Calculator x = new Calculator();
23
24              von = x.MN(c,n);
25                Console.WriteLine("The maximum value is: {0}", von );
26              Console.ReadLine();
27              }
28      }
29}
```

Dissection

- Lines 24-26 shows the calling of MN method.

When the code is compiled and executed, it produces the following results:

```
The maximum value is: 70
```

RECURSIVE METHOD CALL

Recursion is a method of calling itself.

Example:

```
1 using System;
2
3 namespace Calculator
4 {
5 class Calculator
6 {
7       public int factorial(int number)
8       {
9             int result;
10                  if (number == 1){
11                        return 1;
12                  }
13                  else
14                  {
15                        result = factorial (number - 1) * number;
16                        return result;
17                  }
18            }
19            static void Main(string[] args)
20            {
21                  Calculator x = new Calculator();
22                  Console.WriteLine("6 Factorial is: {0}", x.factorial (4));
23                  Console.WriteLine("7 Factorial is: {0}", x.factorial (5));
24                  Console.WriteLine("8 Factorial is: {0}", x.factorial (6));
25                  Console.ReadLine();
26            }
27      }
28 }
```

Dissection

- Lines 22-25 show the calling of the factorial method.

When the code is compiled and executed, it produces the following results:

```
6 Factorial is: 24
7 Factorial is: 120
8 Factorial is: 720
```

PASSING PARAMETERS TO A METHOD

When you call a method with parameters, you need to pass the parameters to the method.

Value Parameters

The value parameter is the default mechanism for passing parameters to a method. When a method is called, a unique storage location is made for every value parameter which adopts the actual value of the parameters themselves. Take note that the changes made to the parameter inside the method has no effect on the argument.

Example:

```
1 using System;
2
3 namespace Calculator
4 {
5       class Calculator
6       {
7             public void interchange(int a, int c)
8             {
9                   int temp;
10                  temp = a;
11                  a = c;
12                  c = temp;
13            }
14            static void Main(string[] args)
15            {
16                  Calculator n = new Calculator ();
17                  int a = 50;
18                  int c = 60;
19
20                  Console.WriteLine ("Before interchange, the value of a: {0}", a);
21                  Console.WriteLine ("Before interchange, the value of c: {0}", c);
22
23                  n.interchange (a,c);
24                  Console.WriteLine ("After interchange, the value of a: {0}", a);
25                  Console.WriteLine ("After interchange, the value of c: {0}", c);
26                  Console.ReadLine ();
27
28            }
29      }
30 }
```

Dissection

- Line 10 saves the value of **a** into **temp**.
- Line 11 saves the value of **c** into **a**.
- Line 12 saves the value **temp** into **c**.
- Lines 17-18 define the local variable.
- Line 23 calls the function to interchange the values.
- The output shows that there is no change in the values though they had changed inside the function.

When the code is compiled and executed, it produces the following results:

```
Before interchange, the value of a: 50
Before interchange, the value of c: 60
After interchange, the value of a: 50
After interchange, the value of c: 60
```

Reference Parameters

Unlike value parameters, reference parameters do not generate a new storage location. They use the same memory location as the method used. Values will change inside the function that will be reflected in the Main function.

Example:

```
1 using System;
2
3 namespace Calculator
4 {
5      class Calculator
6      {
7             public void interchange(ref int a, ref int c)
8             {
9                    int temp;
10                   temp = a;
11                   a = c;
12                   c = temp;
13            }
14            static void Main(string[] args)
15            {
16                   Calculator n = new Calculator ();
17                   int a = 50;
18                   int c = 60;
19
20                   Console.WriteLine ("Before interchange, the value of a: {0}", a);
21                   Console.WriteLine ("Before interchange, the value of c: {0}", c);
22
23                   n.interchange (ref a, ref c);
24                   Console.WriteLine ("After interchange, the value of a: {0}", a);
25                   Console.WriteLine ("After interchange, the value of c: {0}", c);
26                   Console.ReadLine ();
27
28                   }
29            }
30 }
```

Dissection

- Line 10 saves the value of **a** into **temp**.
- Line 11 saves the value of **c** into **a**.
- Line 12 saves the value **temp** into **c**.
- Lines 17-18 define the local variable.
- Line 23 calls the function to interchange the values using the **ref** keyword.
- The output shows that there is a change inside the **interchange** function and this change reflects in the **Main** function.

When the code is compiled and executed, it produces the following results:

```
Before interchange, the value of a: 50
Before interchange, the value of c: 60
After interchange, the value of a: 50
After interchange, the value of c: 60
```

Output Parameters

The output parameter is used if you want to return two values from a function. It is similar to the reference parameters, except that it transfers data out of the method.

Example:

```
1 using System;
2
3 namespace Calculator
4 {
5       class Calculator
6       {
7               public void getvalue(out int c)
8               {
9                       int temp = 8;
10                      c = temp;
11              }
12              static void Main(string[] args)
13              {
14                      Calculator n = new Calculator ();
15                      int d = 50;
16
17                      Console.WriteLine ("Before the method is called, the value of d: {0}", d);
18
19                      n.getvalue (out d);
20                      Console.WriteLine ("After the method is called, the value of d: {0}", d);
21                      Console.ReadLine ();
22
23              }
24      }
25 }
```

Dissection

- Line 19 calls the function to get the value of **d**.

When the code is compiled and executed, it produces the following results:

```
Before the method is called, the value of d: 50
After the method is called, the value of d: 8
```

The output parameter is used to return values of a method which are passed through parameters that have not been assigned an initial value.

Example:

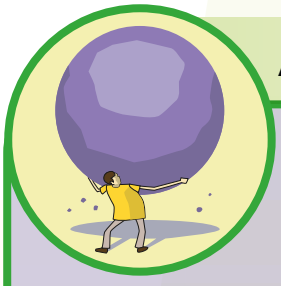
```
1 using System;
2
3 namespace Calculator
4 {
5       class Calculator
6       {
7               public void getvalue(out int c, out int d)
8               {
9                       Console.WriteLine ("Enter the first number: ");
10                      c = Convert.ToInt32 (Console.ReadLine ());
11                      Console.WriteLine ("Enter the second number");
12                      d = Convert.ToInt32 (Console.ReadLine ());
13              }
14              static void Main(string[] args)
15              {
16                      Calculator n = new Calculator ();
17                      int e, f;
18                      n.getvalue (out e, out f);
19                      Console.WriteLine ("After the method is called, the value of d: {0}", d);
20                      Console.WriteLine ("Before the method is called, the value of d: {0}", d);
21                      Console.Readline ();
22              }
23      }
24}
```

When the code is compiled and executed, it produces the following results:

```
Enter the first number: 78
Enter the second number: 23
After the method is called, the value of e: 78
After the method is called, the value of f: 23
```

Vocabulary

Method - group of statements that perform a given task
Output parameter - used to return two values from a function
Reference parameter - represents the same memory location as the actual parameters that are supplied to the method
Value parameter - default mechanism for passing parameters to a method



ASSESSMENT ACTIVITY

1. Write a program that will explain the method in C#. Create a static function add() that accepts two numbers from the user and returns the sum of the number.

2. Write a program that performs addition, subtraction, multiplication, and division. The program should accept two numbers from the user and then using the appropriate function, calculate them and show the output.

3. Write a program that shows the difference between the value type parameter and reference type parameter.

Lesson 9

STRINGS

Content Standard

Learners demonstrate understanding of:

- string objects.

Performance Standard

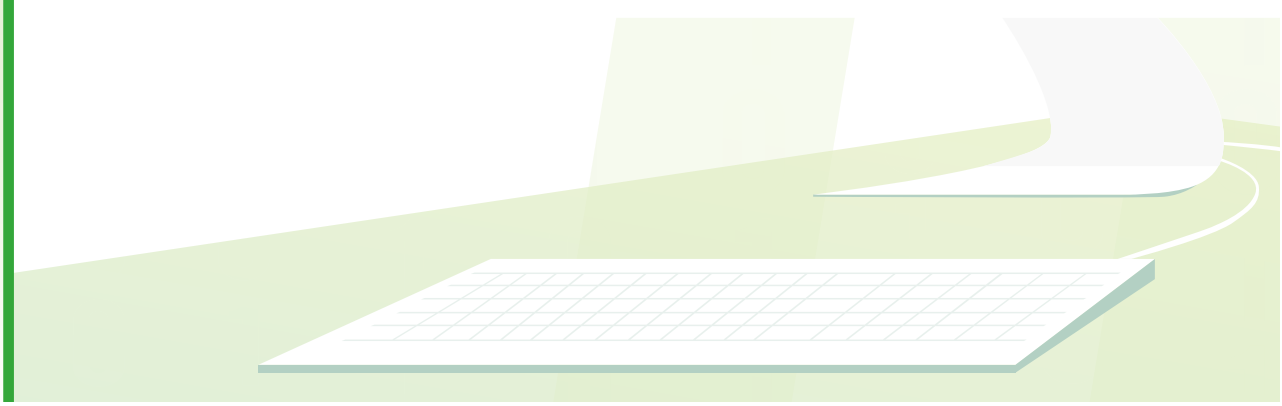
Learners shall be able to:

- use string keyword to declare a string variable.

Learning Outcome

Learners shall be able to:

- create a program that declares a string variable.



CREATING A STRING OBJECT

You can use string as array of characters. In standard practice, the keyword string is used to declare a string variable. It is an alias for the **System.String** class.

Methods for creating a string object:

- Assigning a string literal to a String variable
- Using a String class constructor
- Using the string concatenation operator (+)
- Retrieving a property or calling a method that returns a string
- Calling a formatting method to convert a value or an object to its string representation

Example:

```
1 using System;
2
3 namespace ManipulatingNumbers
4 {
5       class Program
6       {
7               static void Main(string[] args)
8               {
9                       string FName, LName;
10                      FName = "Juan";
11                      LName = "Juan";
12
13                      string CompleteName = FName + LName;
14                      Console.WriteLine("Complete Name: {0}", CompleteName);
15
16                      //using String constructor
17                      char[] letters = {'M', 'a', 'b', 'u', 'h', 'a', 'y'};
18                      string greetings = new string(letters);
19                      Console.WriteLine("Greetings: {0}", greetings);
20
21                      //returning string
22                      string[] sarray = {"Magandang","Buhay","Sa","Lahat"};
23                      string message = String.Join(" ", sarray);
24                      Console.WriteLine("Message: {0}", message);
25
26                      //method to convert a value
27                      DateTime wt = new DateTime(2016,7,12,10,23,5);
28                      string text = String.Format("Message sent on {0:d} at {0:t}", wt);
29                      Console.WriteLine ("Message: {0}", text);
30                      Console.ReadKey();
31              }
32      }
33}
```

Dissection

- Line 9 shows an example of assigning a string literal to a string variable.
- Line 13 shows an example of string concatenation.
- Line 18 shows an example using a string constructor.
- Line 23 shows an example of returning string.
- Lines 27-28 show methods to convert a value.

When the code is compiled and executed, it produces the following results:

```
Complete Name: JuanDela Cruz
Greetings: Mabuhay
Message: Magandang Buhay Sa Lahat
Message: Message sent on 7/12/2016 at 10:23 AM
```

STRING CLASS PROPERTIES

Chars	Gets the Char object at a specified position in the current String object.
Length	Gets the number of characters in the current String object.

STRING CLASS PROPERTIES

1	**public static int Compare(string strA, string strB)** Compares two string objects
2	**public static int Compare(string strA, string strB, boolignoreCase)** Compares two string objects wherein case is disregarded if the Boolean parameter is TRUE
3	**public static string Concat(string str0, string str1)** Concatenates two string objects

4	**public static string Concat(string str0, string str1, string str2)** Concatenates three string objects.
5	**public static string Concat(string str0, string str1, string str2, string str3)** Concatenates four string objects.
6	**public bool Contains(string value)** Returns a value only if the string object appears within the string itself
7	**public static string Copy(string str)** Creates a new String object with the same value as the specified string.
8	**public void CopyTo(intsourceIndex, char[] destination, intdestinationIndex, int count)** Copies a defined number of characters from an identified part of the string object to the specified location in an array
9	**public boolEndsWith(string value)** Determines whether the end of the string object matches the specified string.
10	**public bool Equals(string value)** Determines whether the current String object and the specified String object have the same value.
11	**public static bool Equals(string a, string b)** Determines whether two specified String objects have the same value.
12	**public static string Format(string format, Object arg0)** Replaces at least one format item in a particular string ascribed to a specific object
13	**public intIndexOf(char value)** Returns the index value of the first instance of the specified character in the current string as zero

14	**public intIndexOf(string value)** Returns a null index value of the first instance of the specified string
15	**public intIndexOf(char value, intstartIndex)** Returns the zero-based index of the first occurrence of the specified Unicode character in this string, starting search at the specified character position.
16	**public intIndexOf(string value, intstartIndex)** Returns the zero-based index of the first occurrence of the specified string in this instance, starting search at the specified character position.
17	**public intIndexOfAny(char[] anyOf)** Returns the null value of the index of any character that makes an initial appearance in this instance in a specified array characters
18	**public int IndexOfAny(char[] anyOf, int startIndex)** Returns the null index value of the first appearance of any character in the array character specified, with the search begun, in this instance, at the specified character location
19	**public string Insert(intstartIndex, string value)** Returns a new string in a specific location in the current string object index
20	**public static boolIsNullOrEmpty(string value)** Identifies a null or empty string
21	**public static string Join(string separator, params string[] value)** Concatenates all string array elements using a defined separator
22	**public static string Join(string separator, string[] value, intstartIndex, int count)** Concatenates unique string array elements using a specified separator
23	**public intLastIndexOf(char value)** Returns a zero-index-value position of the last instance of the specified character found in the current string object

24	**public intLastIndexOf(string value)** Returns the zero index value position of the last instance of the specified string found in the current string object
25	**public string Remove(intstartIndex)** Removes all the extant characters from a specified starting position up to the last position identified, after which the string is returned
26	**public string Remove(intstartIndex, int count)** Removes the specified number of characters in the current string beginning at a specified position and returns the string.
27	**public string Replace(char oldChar, char newChar)** Replaces all occurrences of a specified Unicode character in the current string object with the specified Unicode character and returns the new string
28	**public string Replace(string oldValue, string newValue)** Replaces all instances of a specified string in the existing string object with a new string and returns the updated string
29	**public string[] Split(params char[] separator)** Returns a string array with the substrings in the used string object bounded by the elements of the specified character array
30	**public string[] Split(char[] separator, int count)** Returns a string array which has the substrings in the existing string object bounded by the elements of a specified character array. The maximum number of substrings to return is defined by the parameter Int
31	**public boolStartsWith(string value)** Decides if the beginning of the string instance corresponds with the specified string
32	**public char[] ToCharArray()** Returns a character array with all the characters found in the existing string object
33	**public char[] ToCharArray(intstartIndex, int length)** Returns a character array with all the characters in the existing string object from the start of the specified index until the end of its length

34	**public string ToUpper()** Converts a copy of the string in uppercase
35	**public string ToLower()** Converts a copy of the string in lowercase
36	**public string Trim()** Removes all white-space characters from the existing string object which are either leading or trailing

Example:

```
1 using System;
2
3 namespace StringApplication
4 {
5       class Program
6       {
7               static void Main(string[] args)
8               {
9                       string test1 = "Testing 1";
10                      string test2 = "Testing 2";
11
12                      if (String.Compare (test1, test2) == 0) {
13                      Console.WriteLine (test1 + " and " + test2 + " are the same. ");
14                      }
15                      else
16                      {
17                      Console.WriteLine (test1 + " and " + test2 + " are not the same.");
18                      }
19                      Console.ReadKey ();
20              }
21      }
22}
```

When the code is compiled and executed, it produces the following results:

```
Testing 1 and Testing 2 are not the same.
```

Example of String Contains String:

```
1 using System;
2
3 namespace StringApplication
4 {
5      class Program
6      {
7             static void Main(string[] args)
8             {
9                    string test1 = "Test mic";
10                   if (test1.Contains("mic"))
11                   {
12                          Console.WriteLine ("The word 'mic' was found.");
13                   }
14                   Console.ReadKey ();
15            }
16     }
17 }
```

When the code is compiled and executed, it produces the following results:

```
The word 'mic' was found.
```

Example of Getting a Substring:

```
1 using System;
2
3 namespace StringApplication
4 {
5      class Program
6      {
7             static void Main(string[] args)
8             {
9                    string test1 = "You will always be my baby";
10                   Console.WriteLine (test1);
11                   string.substring = test1.Substring (9);
12                   Console.WriteLine (substring);
13                   Console.ReadKey ();
14            }
15     }
16 }
```

When the code is compiled and executed, it produces the following results:

```
You will always be my baby
always be my baby
```

Example of Getting a Substring::

```
1 using System;
2
3 namespace StringApplication
4 {
5      class Program
6      {
7             static void Main(string[] args)
8             {
9                    string test1 = "You will always be my baby";
10                   Console.WriteLine (test1);
11                   string.substring = test1.Substring (9);
12                   Console.WriteLine (substring);
13                   Console.ReadKey ();
14            }
15     }
16}
```

When the code is compiled and executed, it produces the following results:

```
You will always be my baby
always be my baby
```

Example of Joining String:

```
1 using System;
2
3 namespace StringApplication
4 {
5      class Program
6      {
7             static void Main(string[] args)
8             {
9              string[] stringarray = new string[]
10             {
11              "We're a thousand miles from comfort, we have traveled land and sea",
12              "But as long as you are with me there's no place I'd rather be",
13              "I would wait forever, exalted in the scene",
14              "As long as I am with you, my heart continues to beat"
15             };
16             string strarray = String.Join("\n", stringarray);
17             Console.WriteLine(strarray);
18             Console.ReadKey ();
19            }
20     }
21 }
```

When the code is compiled and executed, it produces the following results:

```
We're a thousand miles from comfort, we have travled land and see
But as long as you are with me there's no place I'd rather be
I would wait forever, exalted in the scene
As long as I am with you, my heart continues to beat
```

Vocabulary

String - used to declare a string variable

ASSESSMENT ACTIVITY

1. Write a program that
   a. accepts and compares 2 strings.
   b. joins strings using your favorite song.

Lesson 10

ARRAYS

Content Standards

Learners demonstrate understanding of:

- array declaration;
- array initialization;
- assigning values;
- accessing array elements; and
- 2 dimensional arrays.

Performance Standard

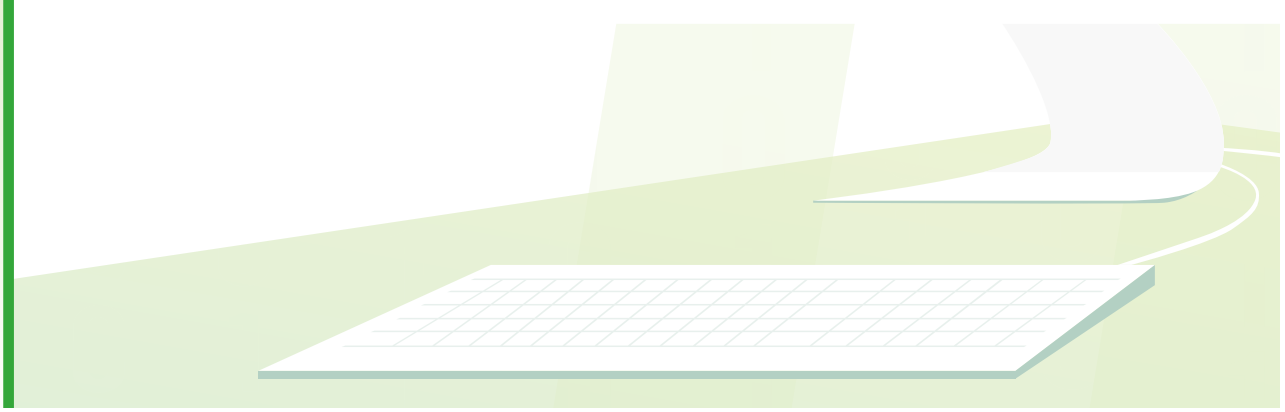
Learners shall be able to:

- describe different parts of an array as one dimensional and two dimensional arrays.

Learning Outcome

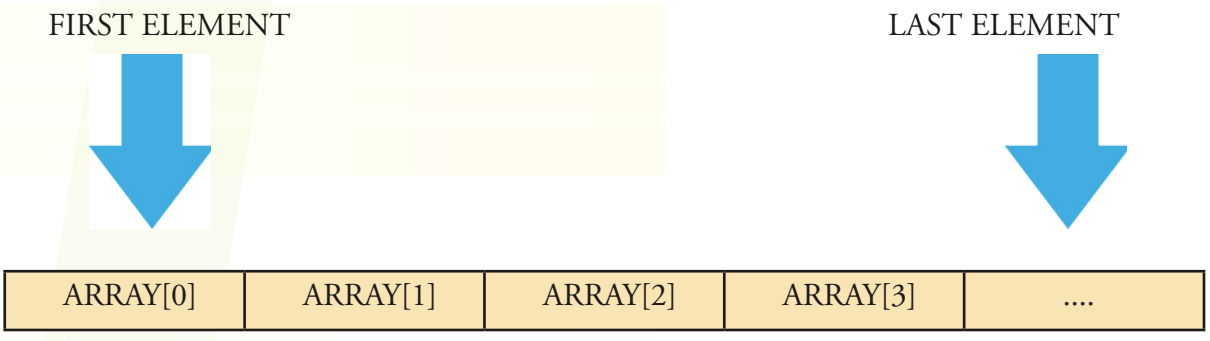
Learners shall be able to:

- write a program that uses multidimensional arrays.



ARRAY DECLARATION

An array is used to store a collection of similar data or variables in specific locations in the memory. It is also used to avoid declaring individual variables. All arrays consist of memory locations from the lowest to the highest address element.



To declare an array, use the syntax below:

```
datatype[] arrayName;
```

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

Example

```
int [] name;
```

ARRAY INITIALIZATION

You should initialize first the array variable before you can assign values. A new keyword is needed to make an instance of a reference-type array.

Example:

```
int[] number = new int[25];
```

ASSIGNING VALUES

You assign values to an individual array by using the index number:

```
int[] number = new int[25];
number[0] = 160;
```

You assign values at the time of declaration:

```
int[] number = {123, 234, 345];
```

You can create and initialize an array:

```
int[] number = new int[25] {45, 65, 76};
```

You can remove the size of the array:

```
int[] number = new int[] {45, 65, 76};
```

You can copy an array variable into another target array variable:

```
int[] numzber = new int[25] {45, 65, 76};
int [] point = number;
```

ACCESSING ARRAY ELEMENTS

You can access an element by indexing the array element.

Example:

```
double rate = balance [8];
```

Example of concept of declaration, assignment, and accessing arrays:

```
1 using System;
2
3 namespace ArrayApplication
4 {
5      class Program
6      {
7            static void Main(string[] args)
8            {
9                  int[] x = new int[10]; /* x is an array of 10 integers */
10                 int a, b;
11
12                 /*initialization of elements of array x*/
13                 for (a = 0; a < 10; a++)
14                 {
15                       x[a] = a + 10;
16                 }
17
18                 /*output of array element*/
19                 for (b = 0; b < 10; b++)
20                 {
21                    Console.WriteLine ("The element [{0}] = {1}", b, x [b]);
22                 }
23                 Console.ReadKey ();
24           }
25     }
26 }
```

When the code is compiled and executed, it produces the following results:

```
The element [0] = 10
The element [1] = 11
The element [2] = 12
The element [3] = 13
The element [4] = 14
The element [5] = 15
The element [6] = 16
The element [7] = 17
The element [8] = 18
The element [9] = 19
```

MULTIDIMENSIONAL ARRAYS

Multidimensional arrays are also called rectangular arrays.

The syntax for 2-dimensional and 3-dimentional arrays are:

```
string[,] names;
int [, ,] can;
```

Two-Dimensional Arrays

The two-dimensional array is a common array which can be compared to a table with a certain number of rows (x) and columns (y).

	Column 0	Column 1	Column 2	Column 3
Row 0	a[0][0]	a[0][1]	a[0][2]	a[0][3]
Row 1	a[1][0]	a[1][1]	a[1][2]	a[1][3]
Row 2	a[2][0]	a[2][1]	a[2][2]	a[2][3]

Every element in the array a is identified by an element name of the form a[c,n], where a is the name of the array and c and n are the subscripts that uniquely identify each element in array a.

Two-Dimensional Arrays Initialization

Multidimensional arrays can be initialized by specifying values inside the bracket for each row. The example below is with 3 rows and 4 columns.

```
int[,] number = int [3,4] = {
{a, b, c, d}, /*initializes for row indexed by 0*/
{e, f, g, h}, /*initializes for row indexed by 1*/
{i, j, k, l}, /*initializes for row indexed by 2*/
};
```

Accessing a Two-Dimensional Array

An element in a 2-dimensional array is accessed by using the subscripts.

Example:

```
intval = a[2,3];
```

Example program of handling a two-dimensional array:

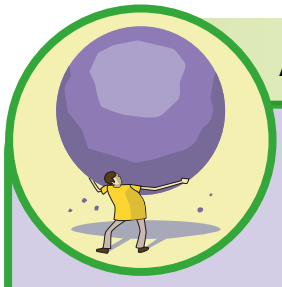
```
1 using System;
2
3 namespace ArrayApplication
4 {
5             class Program
6             {
7                    static void Main(string[] args)
8                    {
9                    int [,] c = new int[4, 2] { { 1, 2 }, { 3, 4 }, { 5, 6 }, { 7, 8
}};
10                   int a,b;
11                   for (a=0; a<4; a++)
12                   {
13                          for (b=0; b<2; b++)
14                          {
15                          Console.WriteLine ("Array [{0},{1}] = {2}", a, b, c [a, b]);
16                          }
17                   }
18                   Console.ReadKey ();
19                   }
20            }
21 }
```

When the code is compiled and executed, it produces the following results:

```
Array[0,0] = 1
Array[0,1] = 2
Array[1,0] = 3
Array[1,1] = 4
Array[2,0] = 5
Array[2,1] = 6
Array[3,0] = 7
Array[3,1] = 8
```

Vocabulary

Array - used to store a collection of data or a collection of variables of the same type stored in a memory location



ASSESSMENT ACTIVITY

1. Write a program on sorting an array. Declare a two-dimensional array and accept an 8-integer value from the user, then sort the input in ascending order and display the output.
2. Write a program to copy one array's element to another array without using an array function.