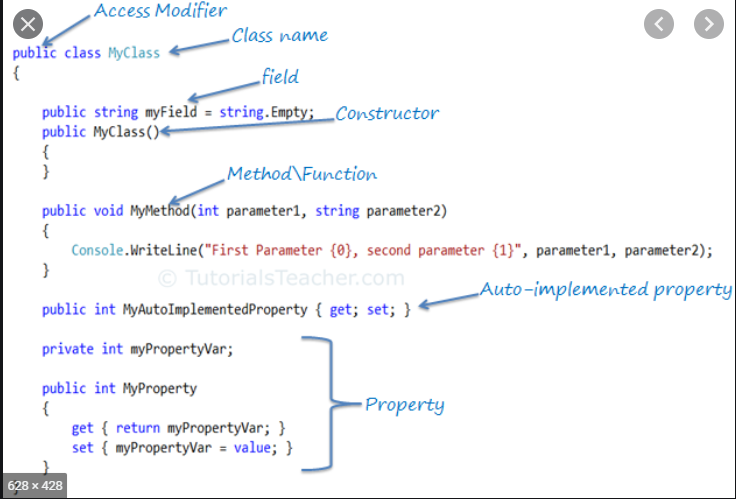
Chapters 1, 3, 4, 5, 6, 7, 8, 9



Chapter 1

**Record:** Employee

**Fields:** ◘ Employee identification number (a whole number)

◘Name (a string of characters)

◘ Address (a string of characters)

◘ Hourly pay rate (a number with a decimal point)

◘ Year-to-date earnings (a number with a decimal point)

◘ Amount of taxes withheld (a number with a decimal point)

**File:** A file is a group of related records.

**Database:** A database is a collection of data that’s organized for easy access and manipulation

**C#**: is an object-oriented programming language.

**Generic Programming:** It’s common to write a program that processes a collection of things.

**Functional Programming**: With functional programming, you specify what you want to accomplish in a task, but not how to accomplish it.

In most programming today, each task in a program must finish executing before the next task can begin **(Synchronous programming)**

C# also allows asynchronous programming in which multiple tasks can be performed at the same time.**(Asynchronous programming).**

**Common Language Runtime:** The Common Language Runtime (CLR) executes .NET programs and provides functionality to make them easier to develop and debug

**The CLR is a virtual machine:** (VM)—software that manages the execution of programs and hides from them the underlying operating system and hardware.

**Chapter 3:**

**A using directive** tells the compiler where to look for a predefined class that’s used in an app.

The **class** keyword introduces a class declaration and is immediately followed by the class name.

**C# is case sensitive**—that is, uppercase and lowercase letters are distinct, so a1 and A1 are different (but both valid) identifiers

For each app, one of the methods in a class must be called **Main**; otherwise, the app will not execute.

**IntelliSense** lists a class’s members, which include method names.

The **backslash (\)** is called an **escape character** and is used as the first character in an escape sequence.

C# 6 introduces a mechanism called **string interpolation** that enables you to insert values in string literals to create formatted strings.

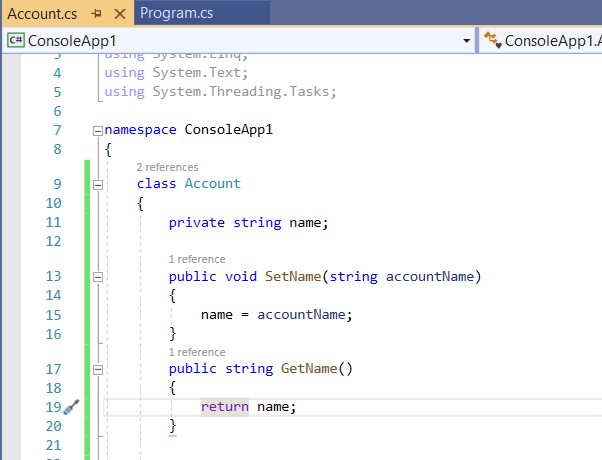
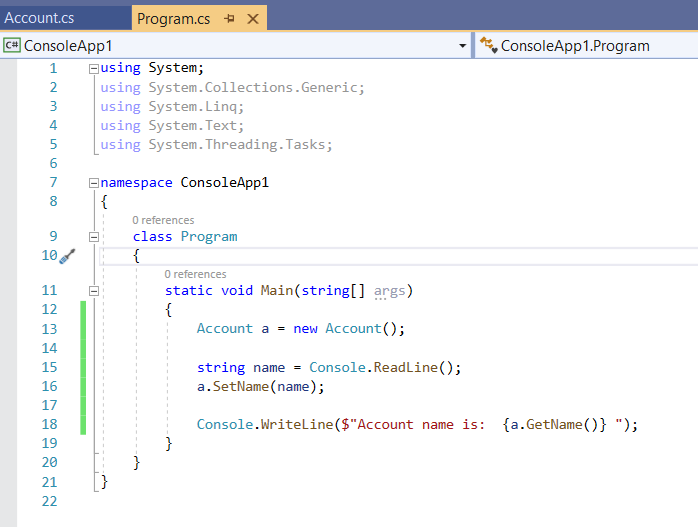
An interpolated string must begin with a $ (dollar sign). Then, you can insert interpolation expressions enclosed in braces, {}, anywhere between the quotes ("").

****

The **arithmetic operators** are summarized in following figure. (+,-,\*,/)

**Chapter 4:**

Each **class** created by a developer becomes a new **type**, and it can be used by **any developer** to create objects, so C# is an extensible programming language.

****

**class**, **property** and **method** names begin with an initial uppercase letter (**Pascal case**)

**variable** names begin with an initial lowercase letter (**camel** case).

**C# calling instance variable**: private string name; // instance variable

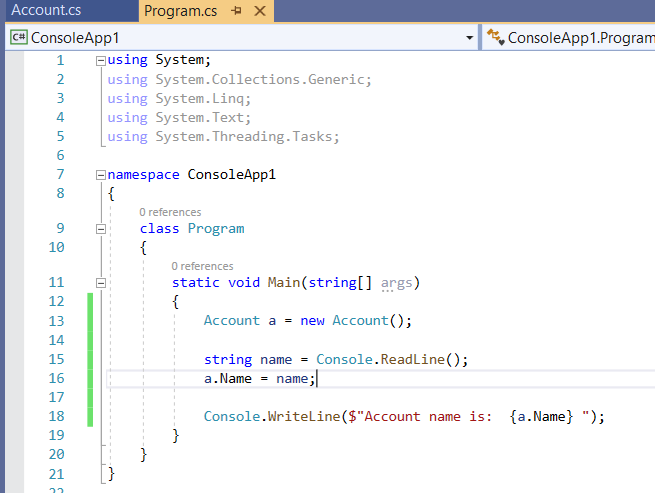
Access Modifier

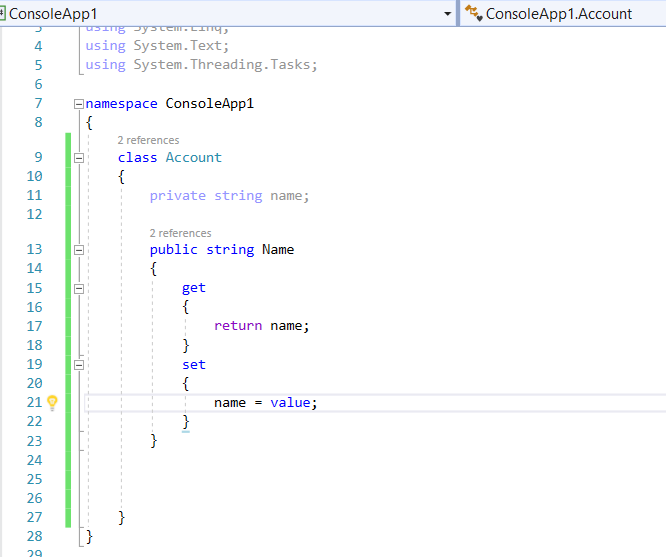
The keyword **private** is an **access modifier**.

Instance variable name is private to indicate that name is accessible only to class Account’s methods

* and other members, like properties, as you’ll see in subsequent examples

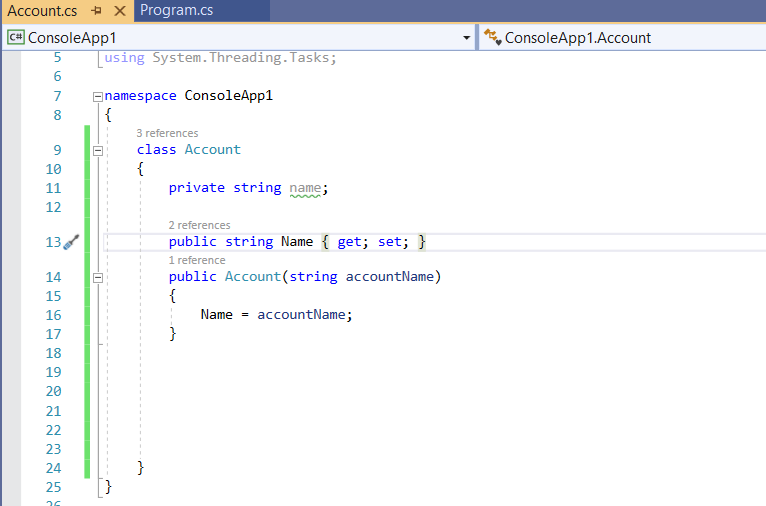
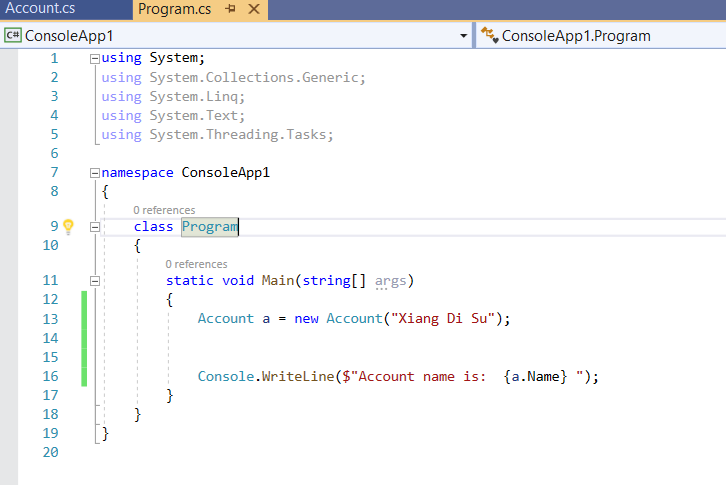
**USING PROPERTY FOR GET AND SET**





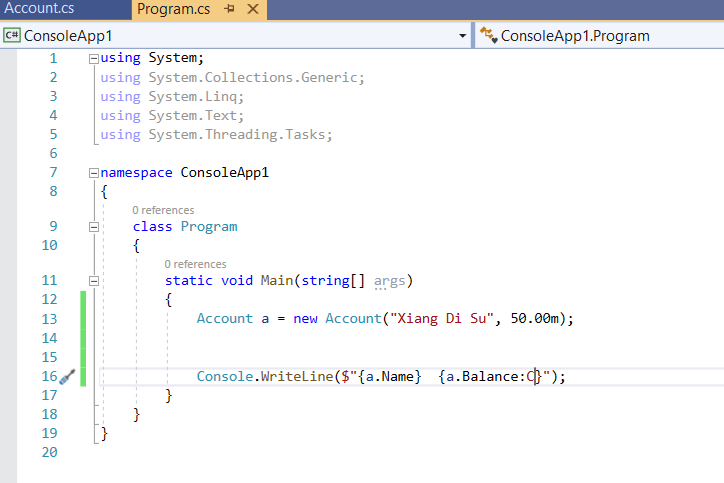
Output is whatever you type as name

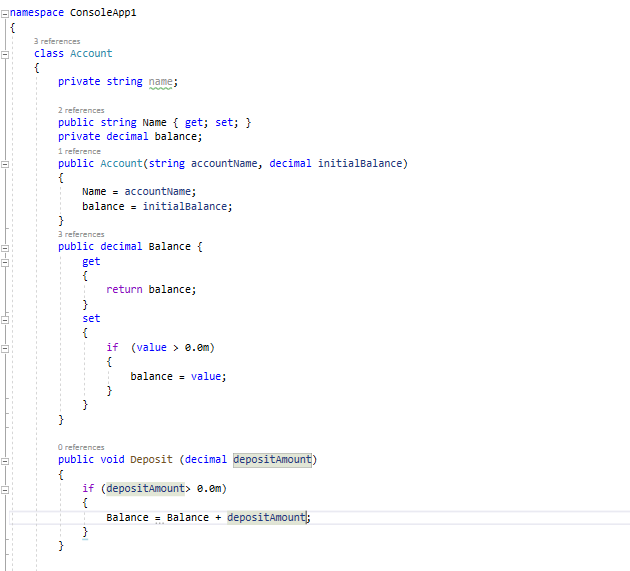
**ADDING CONSTRUCTOR**



OUTPUT: Xiang di su

**ACCOUNT CLASS WITH A DECIMAL BALANCE INSTANCE VARIABLE**

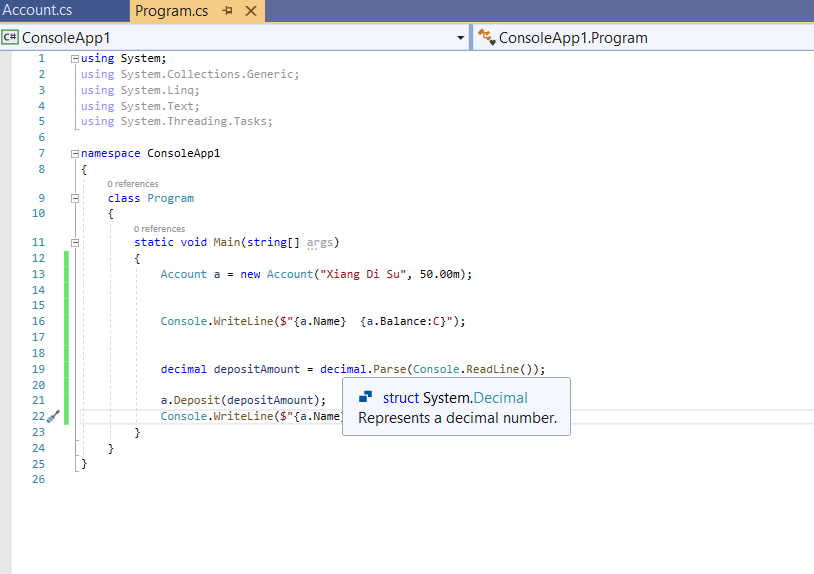




OUTPUT IS : “Xiang Di Su $50.00”

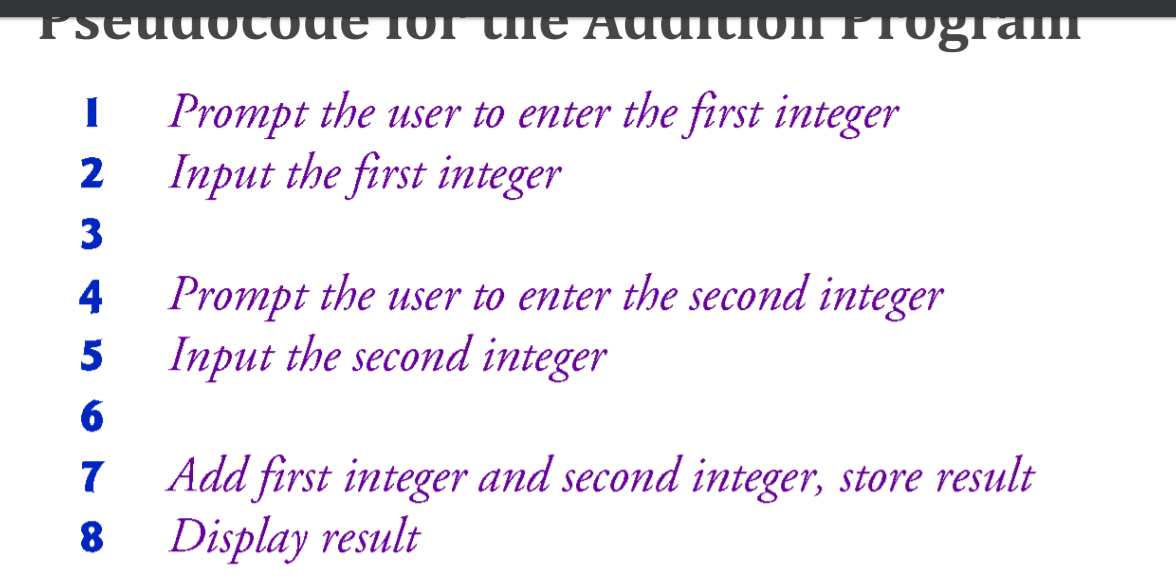
a.Balance:C means in dollar sign

**Creating deposit method to add the amount in balance**

****

**Chapter 5 :**

The **pseudocode** statements are simply English statements that convey what task is to be performed in C#.



**If (single selection statement)**

**If else (double selection statement)**

**Nested if else (if else (if else if else if else)) or (if, else if , else if , else if , else)**

**Block:** Statements contained within a pair of braces ({ and }) form a **block**

Ex:

if (studentGrade >= 60)

{

Console.WriteLine("Passed");

}

else

{

Console.WriteLine("Failed");

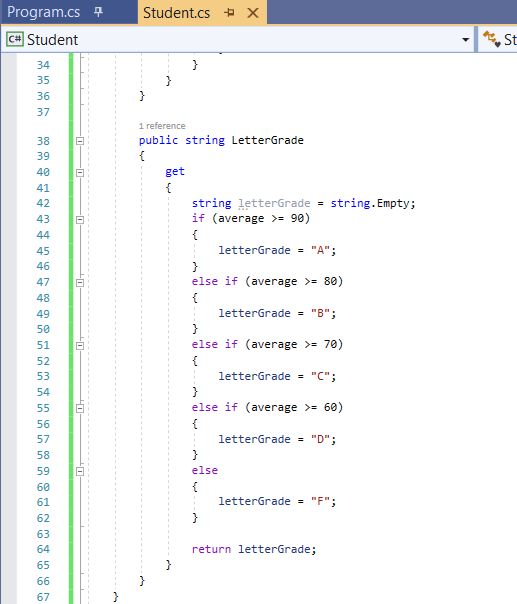
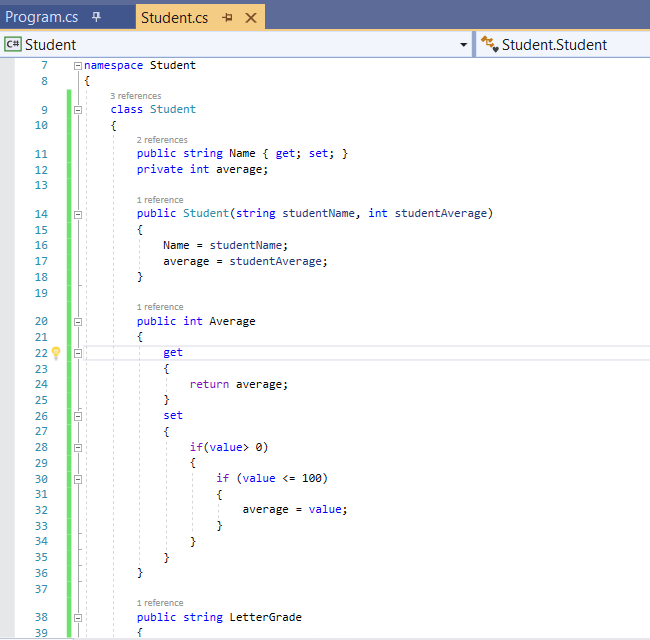
Console.WriteLine("You must take this course again.");

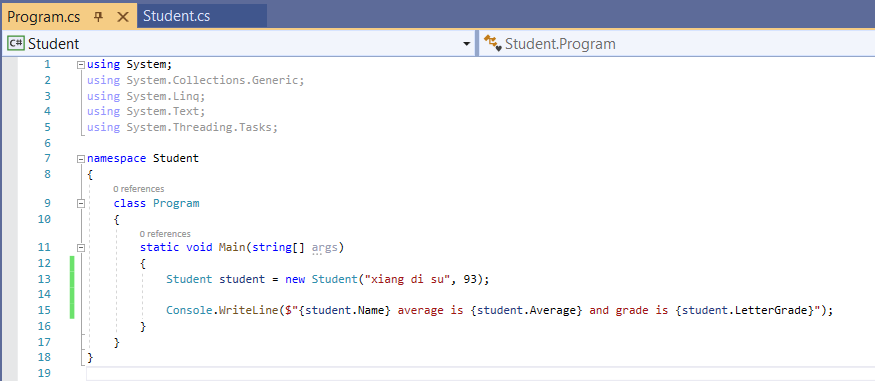
}

The **conditional operator** (?:) can be used in place of an if…else statement.

Ex : Console.WriteLine(studentGrade >= 60 ? "Passed" : "Failed");

**STUDENT CLASS THAT STORES A STUDENT NAME, AVERAGE GRADE AND GRADE**

****

****

OUTPUT IS: Xiang di su average is 93 and grade is A

**While Iteration Statement**

**A iteration** statement allows you to specify that an app should repeat an action: consider a code segment designed to find the first power of 3 larger than 100:

Example:

Int product = 3;

While (product <= 100)

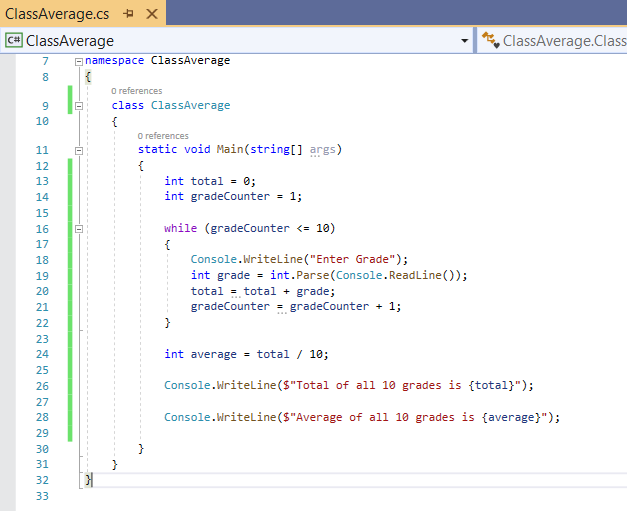
{

product = 3 \* product

}

**First Refinement: initialize variable, input the 10 results (fixed)**

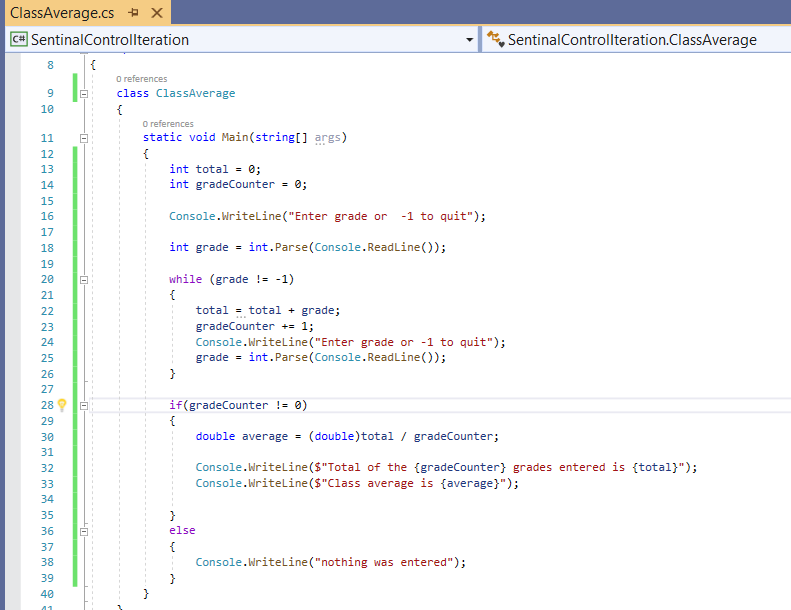
**COUNTER CONTROL ITERATION (WHILE LOOP) FOR CLASS AVERAGE**

****

OUTPUT IS : enter grade times 10 and then gives total and average

**Second Refinement:** initialize total to 0 and counter = 0

**SENTINAL CONTROL ITERATION (WHILE LOOP)**

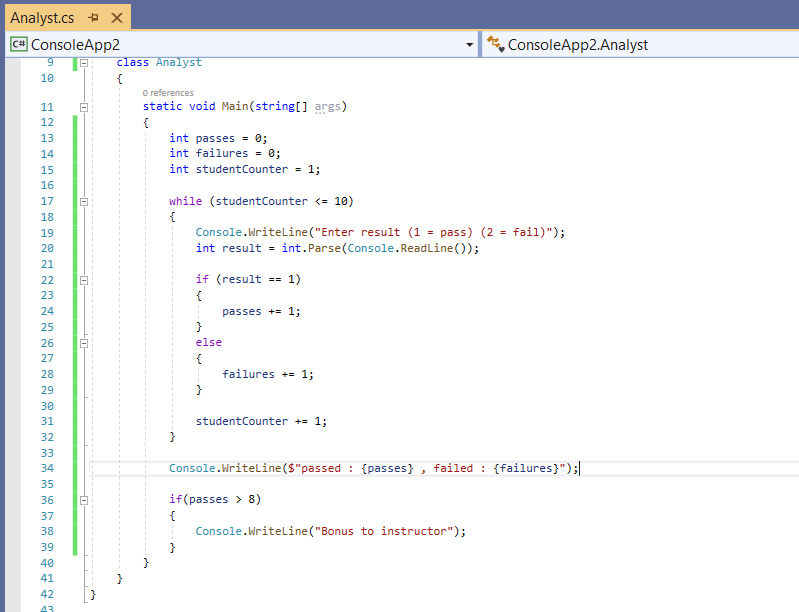


OUTPUT IS : 10 , 20 ,30 , -1 , total of the 3 grades entered is 60, class average is 20

{average:F} == xx.00

{price:C} == $xx.xx

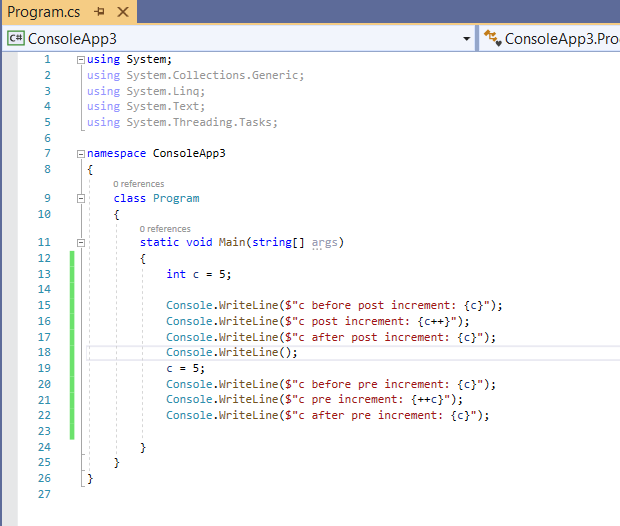
**NESTED CONTROL STATEMENT (COMBINATION OF FIRST REFINEMENT AND SECOND REFINEMENT)**



OUTPUT : 1 ,1,1,1,1,1,1,1,1,1 passed: 10 failed : 0 Bonus to instructor!

**Compound Assignment Operator:** C = C + 3; SAME AS C += 3;

**PREFIX INCREMENT AND POSTFIX OPERATOR**

****

OUTPUT IS : post increment : 5, 5, 6 pre increment: 5, 6, 6

**Chapter 6**