**MULTIPLE CHOICE QUESTIONS:**

The **clear** method deletes all of the values in a ListBox.

The **SaveFileDialog** displays a standard Windows *Save As* dialog box.

You can use the StreamWriter class’s **WriteLine** to write a line of text to a file.

To **increment** a variable means to increase its value.

The **Add** method of the itmes property of a ListBox adds an item to a ListBox.

The **method header**, which appears at the beginning of a method definition, lists several important things about the method, including the method’s name.

One benefit of using methods is known as **code reuse** because you are writing code to perform a task once and then reusing it each time you need to perform the task.

The word directly following the return type (in a method header) is called the **method name**.

Which of the following must be true when making a method call? **The number of arguments in the call ust match the number of parameters in the header of the definition. And, the argument types must be compatible with the parameter types. (A & B)**

The **method body** is a collection of statements that are performed when the method is executed.

The default parameter passing mechanism in C# is **pass by value**.

It is standard convention among C# programmers to use **Pascal or Title Case** for method names because it differentiates method names from variable and field names.

The **do … while** statement executes at least once and continues executing until the condition is false.

In a for loop, the control variable is incremented or decremented (based on the update action of the loop) **after the body of the loop executes**.

If the user clicks the *Cancel* button, the SaveFileDialog control's ShowDialog method returns the value **DialogResult.Cancel**.

Assume that intCounter is an int variable initialized to 5. Which of the following loop continuation conditions can be used in a dowhile statement to print the values 5 through 15, inclusive? **intCounter <= 15 and intCounter < 16 (B & D)**.

An **infinite** occurs when the condition in a while loop never becomes false.

The third expression that appears in the loop header of the for loop is the **update expression**, which executes at the end of each iteration, and it is typically the statement that increments the loop’s counter variable.

ListBox controls have an **Items.Count** property that reports the number of items stored in the ListBox.

A **counter** is a variable that helps control the number of times that a set of statements repeat.

The **Save As dialog box** allows the user to browse the system and select a location and name for a file that is about to be saved.

Which one of the following statements correctly calls a method named ShowName which has no parameters and no return value? **ShowName();**

Variables declared in a method are called **local variables**.

Methods can return **0 or 1** values.

Which of the following statements correctly returns the variable intValue from a method? **Return intValue;**.

If a while loop's condition is initially false, how many times will the body execute? **0**

The dowhile statement differs from a while statement in that **the body of a do while statement will execute at least once**.

In code, you can display a *Save As* dialog box by calling the SaveFileDialog control's **ShowDialog**

The **do while loop** is a posttest loop, which means it performs an iteration before testing its Boolean expression.

The declarations and statements that compose the method definition are called the **method body**.

When you want a method to be able to change the value of a variable that is passed to it as an argument, the variable must be **passed by reference**.

Referencing a variable with block scope outside the block in which it was declared is a **syntax error**.

When you want to read data from a text file, you create a file object using the **StreamReader**.

To add an item to a ListBox control with code, you call the control’s **Items.Add** method.

The first expression that appears in the loop header of the for loop is the **initialization expressiono**, which is normally used to initialize a counter variable to its starting value.

The **while** statement executes until it's loop condition is false.

The .NET Framework provides a class named **Random** that you can use in C# to generate random numbers.

What occurs after a method call is made? **Control is given to the called method. After the method returns, the statement after the call is executed.**

With **pass by value**, changes made to the parameter variables' values do not affect the value of the variables in the calling method.

Which of these loop conditions describes the loop "repeat until i is greater than 5"? **while (I <= 5)**

The for loop header \_\_\_\_\_\_\_\_\_ can be used to vary the control variable over the odd numbers between 1 and 10. **For (i = 1; i <= 10; i += 2)**

The second expression that appears in the loop header of the for loop is the \_\_\_\_\_\_\_\_\_\_\_\_\_, which is a Boolean expression that controls the execution of the loop. **test expression**

Failing to return a value from a non-void method **is a syntax error**.

The feature of the IDE that displays a window containing the method’s argument names and types is called **parameter info**.

Declaring a variable in the method’s body with the same name as a parameter variable in the method header is **a syntax error**.

**MATCHING QUESTIONS:**

When an argument is passed by reference, the called method can access and modify **the callers original data directly**.

When you want to write data to a text file, you create a file object using the **StreamWriter**.

Passes an initialized parameter by reference. **Ref Keyword**

Declared inside a form but outside any method of that class and have form-level scope. **Instance Variable or Form-level Variable**

Passes an uninitialized parameter by reference. **Out Keyword**

Called method can access and modify the caller’s original data directly. **Pass-By-Reference or Call-By-Reference**

The portion of the application in which an identifier (such as a variable name) can be referenced. **Scope**

The Application makes a copy of the argument’s value and passes that copy to the called method. **Pass-By-Value or Call-By-Value**

Enables all methods in the same form to access all instance variables declared in that form. **Form-level Scope**

Variables declared inside control statements, such as for statement – scope begins at the identifiers declaration and ends at the block’s closing right brace. **Block Scope**

Type of result returned from a method. **Return type**

Declared inside a block, such as a method or a for statement and have block scope. **Local variable**

Combines (or concatenates) its two operands into one string of characters. **String Concatenation operater (+)**

A control statement that executes a set of statements while the loop-continuation condition is true; the condition is tested after the loop executes. **Do...While repetition statement**

The first line of a for repetition statement and specifies all four essential elements for the counter-controlled repetition of a for repetition statement. **for Header**

Transfers the focus of the application to the control, on which the method is called. **Focus Method**

The "middle" section of the for loop header.  The loop repeats as long as this is true. **for loop condition**

Begins the for statement. **for Keyword**

The "first" section of the for loop header.  These statements are performed just once, before the loop executes the first time. **for loop initialization section**

Specifies whether a control, such as a Button, appears enabled (true) or disabled (false). **Enabled property**

A control statement that executes a set of body statements while its loop-condition is true. **While repetition statement**

Handles the details of counter-controlled repetition. The for statement uses all four elements essential to counter-controlled repetition in one line of code: the name of a control variable, the initial value, the increment or decrement value and the condition to test for the final value. **For Repetition Statement**

Allows the user to view items in a list. Items can be added to or removed from the list programmatically. **ListBox Control**

An error in which a repetition statement never terminates. **Infinite loop**

Returns the number of ListBox items. **Count property of items**

Adds an item to a ListBox control. **Add method of the Items Control Property**

Post condition loop. This loop executes the body of the loop at least once before checking the loop condition. **Do while loop**

Deletes all the values in a ListBox’s control. **Clear method of the items control property**

Returns an object containing all the values in the ListBox. **Items property in the ListBox control**

A varable oftem used to determine the number of times a block of statements in a loop will execute. **Counter**

The backslash (\) character that is used to form escape sequences. **Escape Character**

Another name for a repetition statement. **Loop**

Sends a value back to the method’s caller. **Return Statement**

A comma-separated list in which the method declares each parameter’s name and type. **Parameter List**

Follows the return type in a method header – a valid identifier, it distinguishes one method from another. **Method Name**

The method that calls another method. **Caller**

Used in place of the return type to indicate that a method will return no value. **void Keyword**

The first line of a method (including return type, the method name and the parameter list). **Method Header**

Invokes a method, specifying the method name and providing information (arguments) that the callee requires to perform its task. **Method Call**

The method header, body and braces surrounding the body. **Method Declaration**

Declared in a method’s parameter list. A “placeholder” that can be used in the body of the method. **Parameter**

The declarations and statements that appear between the set of braces that follow the method header. The method body contains C# code that performs actions, generally by manipulating or interacting with the parameters from the parameter list. **Method Body**

**CODE PRACTICE:**

Describe the function of the 3 basic control structures available in all computer programming languages.

**Sequential – statements are executed in the order they appear.**

**Selection – program branches or makes choices; if/else if/ else or switch.**

**Repetition – program executes a block of code more than once without re-writing code; for, while, and do-while loops.**

Describe 3 ways in which programmers benefit from breaking complex code into pieces or methods.

**Easier to debug**

**Easier to read**

**Reusable code**

**Reduce redundant code – if updating, have to change every place it exists**

Assume that you've been asked to write an application that displays all of the prime numbers between 1 and an upper limit provided by the user. The user will enter the upper limit in a textbox called limitTextBox and then will press the calculateButton. The application will display the prime numbers between 1 and the upper limit in a listbox called primeListBox. When the user enters 10, for example, the listbox will display 1, 2, 3, 5, 7 (each on a separate line in the list box). The application will also write the contents of the list box to a file.  
  
Write syntactically correct C# code for:

the definition of a method IsPrime that takes an integer parameter and returns true or false.

body of the calculateButton click event handler.

Here's an algorithm for determining if a number is prime:

IsPrime (someNumber)

prime = true

if someNumber = 1 then

prime = false

else if someNumber = 2 then

prime = true

else

divisor = 2

while divisor < someNumber and prime = true

if someNumber is divisible by divisor then

prime = false

else

divisor = divisor + 1

end if

end while

end if

return prime

end IsPrime

**Here's one reasonable solution for the event handler without the file io:  
  
calculate click  
{**

**try**

**{**

**int upperLimit;  
  
primeListBox.Items.Clear();  
upperLimit = int.Parse(limitTextBox.Text);  
  
for (int count = 1; count <= upperLimit; count++)  
{**

**if (IsPrime(count))**

**primeListBox.Items.Add(count.ToString())**

**}**

**}**

**catch (Exception ex)**

**{**

**MessageBox.Show(ex.Message);**

**}**

**}**

**OR…**

IsPrime (someNumber)

prime = true *//initializing a Boolean variable to true*

if someNumber = 1 then

prime = false

else if someNumber = 2 then

prime = true

else

divisor = 2

while divisor < someNumber and prime = true

if someNumber is divisible by divisor then *// If (someNumber % divisor == 0), fals*

prime = false

else

divisor = divisor + 1

end if

end while

end if

return prime *//returns either true or false*

end IsPrime

**PSEUDOCODE FOR CALCULATE BUTTON:**

calculateButton event handler

get upperLimit from text box

for i=1 to upperLimit

If (IsPrime(i))

Add I to listBox

End if

End for

Output the listBox items to file (another for loop to do one file at a time, could be separate method but doesn’t have to be.

End event handler

**% Mod Operater:**

Int x, y, z;

Z = x % y;

25/4 = 6 is result, 1 is remainder, z gets remainder

**private int RandomNumber(int min, int max)**

**{**

**Random random = new Random();**

**return random.Next(min, max);**

**}**

*Notice that this method takes 2 arguments First argument is inclusive Second argument is exclusive =<lower value, <upper value*

*Random method must be called by each of the buttons representing the 3 possible user choices. (Why?) When method is called, must pass values to random method*

**int returnValue = RandomNumber(1,4);**

*There are 3 buttons one each representing either rock, paper or scissors. When one is chosen by user, the computer randomly generates its choice, examines the outcome (return value), and prints out the results of that round.*

*If and else if for a choice of rock:*

**If the returnValue == 1**

**I chose rock, we tied**

**Else if the returnValue == 2**

**I chose paper, paper wraps rock, you lose**

**Else if the returnValue == 3**

**I chose scissors, rock breaks scissors, you win**

*How to add scoring:*

*Create 2 global variables (public), one representing the user’s score and the other representing the computer’s score.*

*With a tie, no one wine, no score information required.*

*With a user loss, one point gets added to the computer score.*

**Cscore=Cscore + 1;**

*With a user win, one point gets added to the user’s score.*

**Uscore=Uscore + 1;**

*More scores:*

*After each round of the game, update the score and print to screen.*

*Finalizing Game:*

*Game ends when first player scores 10.*

*Test each value after each round to see if either score equals 10*

**If (Cscore == 10)**

**Print (“I win, you lose”)**

**If (Uscore == 10)**

**Print (“Congratulations, you have won the game”)**