|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **CS 1400 Lab #19**  **Passing Parameters by Value, by Reference and by Out**  **Version 1.0**  **Objectives:**  The objective of this assignment is to help you understand how to write correct methods when parameters are passed by value, by ref (reference) and by out (out reference). This assignment also introduces you to the concept of a code walkthroughs.  **Study Material**  Before writing the program for this assignment, please study the following pages:   |  | | --- | | **Passing Parameters by Value, by Reference and by Out**  Consider the following method:  **static** **public** **int** **DoubleValue**(**int** **num**)  {  **return** (**num** **\*** 2);  }  Look closely at the first line (header) of the method:  **static** **public** **int** **DoubleValue**(**int** **num**)  This method header declares that the method ***DoubleValue*** is a static method, returns of value of type ***int*** and takes a single integer as its only parameter, which we will refer to as ***num*** inside of the method. The body of the method is written between the curly braces following the method header. To call or invoke this method, we could write the following code:  **int** **iVal** **=** 5;  **int** **rVal** **=** **DoubleValue**(**iVal**);  This statement would pass the argument as an ***int*** which a copy of the value of ***iVal*** which is literal value ***5***, by value, as the argument to the method, and the return ***int*** value from the method as ***10***. The return value will then be stored in the variable ***rVal*** when the method returns. Note that methods return in place of them selves a single typed value, in this case the ***int*** ***10***. Values are passed to methods on what is called the **run-time Stack Frame** and they return the a single typed value in the ***eax register***. If a method returns void this cannot be done.  The following code illustrates the way that we write and call methods in a program. In this example, the parameter ***num*** is passed **by value**. Consider the first time that the method is called. At that moment, the variable ***idx***contains the value of ***0***. The compiler makes a copy of this value and places it on the **run-time Stack Frame**. Inside the method, the value on the Stack and is the local variable ***num***. When we double ***num***, it only affects the variable ***num*** that is defined local to the function on its private Stack memory and not the original variable ***iVal***. The original variable ***iVal***is not changed, just its copy ***n*** on the methods Stack.  The following example illustrates a complete program that includes this method.  **using** **System**;  **static** **class** Program  {  /// <summary>  /// Purpose: Entry point to your C# program  /// </summary>  **static** **void** **Main**()  {  **int** **iVal** **=** 5;  **int** **rVal** **=** **DoubleValue**(**iVal**);  **const** **int** **SIZE** **=** 5;    **for** (**int** **idx** **=** 0; **idx** **<** **SIZE**; **idx++**)  {  Console**.WriteLine**( **DoubleValue**( **idx** ) );  }  Console**.WriteLine**("Doubled result = {0:D}", **rVal**);  Console**.WriteLine**("Press Enter to continue ...");  Console**.ReadLine**();  }**//End Main()**  /// <summary>  /// Purpose: DoubleVaule method takes the value num and doubles its  /// value  /// </summary>  /// <param name="num">num passed by value as an int</param>  /// <returns>num \* 2 as an int</returns>  **static** **public** **int** **DoubleValue**(**int** **num**)  {  **return** (**num** **\*** 2);  }  }**//End class Program**  Data can also be passed to a method **by ref (reference)**. When an argument is passed **by ref (reference)**, the compiler puts a handle to the address for the argument passed to the method, on it’s run-time Stack, not a copy of its value. Now, when the method uses the argument, it uses the actual variable whose handle (address) to the parameter that was passed on the Stack, not a copy of the value of the variable. We denote a ***pass-by- ref (reference)*** using the keyword **ref** before the type name in the method's signature. For example method header looks like:  **static** **public** **int** **DoubleValue**(**ref** **int** **num**)  We must also use the ***ref*** keyword when calling a method that requires a ***ref*** parameter. For example  **int** **rVal** **=** **DoubleValue**(**ref** **iVal**);  In the call to ***DoubleValue*** where the argument is passed by ***ref***, because we have given the method a handle to address of the variable ***iVal***, declared somewhere outside of the method, the method is free to change the value of that variable. Usually, this is not a desirable thing to do. Changing the value of a variable declared outside of the method is called a **side effect**. In general, we do not like to have ***side*** ***effects*** in our code. To be more precise, a method should only change the values of variables that are declared inside of the method and NOT outside of the method. However, there are a number of necessary exceptions to this rule. | | Data can also be passed back from a method via the ***out*** parameter. When an argument is passed by ***out*** the method must pass a value ***out*** through this parameter, the compiler puts a handle to the address to the ***out*** argument passed to the method on it’s run-time Stack, not a copy of its value. Now, when the method uses the argument, it uses the actual variable whose handle to it’s ***out*** parameter passed on the Stack, not a copy of the value of the variable. We denote a pass by ***out*** by using the keyword ***out*** before the type name in the method's signature. For example:  **static** **public** **void** **DoubleValue**(**int** **num**, **out** **int** **rnum**)  We must also use the ***out*** keyword when calling a method that requires the ***out*** argument. For example  **int** **iVal** **=** 5;  **int** **rVal** **=** 0;  **int** **rVal** **=** **DoubleValue**(**iVal**, **out** **rVal**);  In the call to ***DoubleValue*** where the argument is passed by ***out***, because we have given the method a handle to the address of the variable x, declared somewhere outside of the method, the method is free to change the value of that variable. | |   **Programming Exercise**   |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | | Passing Parameters by Value, by Reference and by OutIntroduction In this exercise, you have been provided with a program that contains a method where the parameters are passed by value. You will modify this program so that the parameters are passed by ***ref*** and finally modify your program to return values by ***out***. The Problem In this program we have a method that is supposed to interchange the values contained in two different variables. As written, the program does not work. Your task will be to fix the program so that it works correctly. The Code Download the code located below:    **// Project Prolog**  **// Name: Dennis A. Fairclough**  **// CS 1400 Section 002/003**  **// Project: CS1400\_Lab\_19**  **// Date: 5-Oct-2014**  **//**  **// I declare that the following code was written by me or provided**  **// by the instructor for this project. I understand that copying source**  **// code from any other source constitutes cheating, and that I will receive**  **// a zero on this project if I am found in violation of this policy.**  **// -------------------------------------------------------------------------**  **using** **System**;  **static** **class** Program  {  /// <summary>  /// Purpose: Entry point to your C# program  /// </summary>  **static** **void** **Main**()  {  **int** **iVal1** **=** 5;  **int** **iVal2** **=** 7;  **//Call the Swap method with two arguments**  **Swap**(**iVal1**, **iVal2**);  Console**.WriteLine**("Swapped values first {0:D} second {1:D}",**iVal1**  ,**iVal2**);  Console**.WriteLine**("Press Enter to continue ...");  Console**.ReadLine**();  }**//End Main()**    /// <summary>  /// Purpose: To swap the two parameters passed to this method  /// </summary>  /// <param name="num1">num1 int, first number</param>  /// <param name="num2">num2 int, second number</param>  **static** **public** **void** **Swap**(**int** **num1**, **int** **num2**)  {  **int** **tempInt** **=** **num1**;  **num1** **=** **num2**;  **num2** **=** **tempInt**;  }  }**//End class Program**  Copy the program above and save it in your Lab #19 folder. Create a new C# Project and copy this file into the Project. Compile and execute the program. You will notice that the program does not work. The values of the two integers are not exchanged, why? Change this code so that the values are passed by ***ref***. Now compile and execute the program. Change this code so that the values are passed by value and returned by ***out***. Compile and execute the program.  Add comments at the end of your program that explain why passing the parameters by ***ref*** make the program work correctly. Add comments at the end of your program that explains why returning values via ***out*** and why it works. File(s) to Submit: Place the your complete project folder into a zip file and name the zip file Lab\_19\_your-initials\_V1.0.zip. For example, I would name my file Lab\_19\_DAF\_V1.0.zip. Submit this assignment as Lab #19 on Canvas.   |  |  |  | | --- | --- | --- | |  | **Grading Checklist** |  | | # | Program | C(correct)  X(incorrect) | | 1 | Meets & works to specifications | 6 points | | 2 | Error Free, elegant & efficient | 4 points | | 3 | Pseudo-Code | -3 points | | 4 | Style Guidelines | -2 points | | 6 | Source Files(s) & Formatting | -2 points | | 7 | Project Prolog | -1 points | | 8 | Function Prologs | -1 points | | 9 | Zip Filename | -1 points | | 10 | Lab & Project Names | -1 points | | 11 | Zip File is invalid or will not unzip | Lab = 0 pts | |  | Total Points | 10 | 0-9 | | |