**Name: Jason Truong**

For each of the below questions, write a short sentence or two to express (in your own words) your answer. Keep the answers short, but use complete, correct, English sentences.

If it helps to clarify the questions, feel free to mentally prefix all the questions with the phrase "According to the video…"

* After you’ve watched all the videos, please answer this question:  
  Of all the videos that you watched, if you could pick one video to be re-recorded by the instructor outside of class which would you choose? Why?  
  (Keep in mind the recording outside of class will omit any pauses from the instructor answering student questions, have less hemming and hawing, etc, and generally be more concise)

|  |
| --- |
| The videos are rather quiet... |

**VIDEO: If, If/Else**

* If you can’t remember the syntax for something in C#, you can try whatever you would have done in what other language (and it will probably (but not always) be right)?

|  |
| --- |
| Java. |

* Copy down the example of the if statement from the video (the one involving x, y, and printing “Yippee!”). For each part of the if statement put in a note about the syntax (just like the video does). This means that you should have a note about “if” being lowercase, etc.

|  |
| --- |
| int x = 10;  int y = 20;  if (x < y)  {  Console.WriteLine("Yippee!");  } |

* Why is it useful for us to move rightwards 1 tab stop for all the statements that are “within” the if statement?

|  |
| --- |
| Visual Clarity |

* What is a “block statement”? Why is it useful?

|  |
| --- |
| A series of statements wrapped around brackets would be treated as a single statement by the previous conditional statement; if true. |

* For the following program, EXACTLY what will be printed and why?

int x = 100;

int y = 20;

if( x < y )

Console.WriteLine(“Yippee!”);

Console.WriteLine(“Yippee!”);

Console.WriteLine(“Yippee!”);

Console.WriteLine(“End Of program”);

|  |
| --- |
| Yippee!  Yippee!  End of program  After the x and y are intialized, the if statement tests if the x variable is less than the y variable.  The (x<y) is false, and since the two 'Yippee!'s following the first aren't encased in brackets, it would be read as white-space in a single-statement.  Then it writes the 'End of program' as per usual. |

* How would you check if two variables are the same in C#?  
  How would you check if two variables are NOT the same in C#?  
  How would you check if one variable is less than another variable in C#?  
  How would you check if one variable is less than or equal to another variable in C#?

|  |
| --- |
| ==, same  !, not; thus, != not same  < or >, less than another value  =< or => less than or equal to another variable |

* Can you put an “else” clause onto anything other than an “if” statement?

|  |
| --- |
| C# syntax won't allow you. |

**VIDEO: Switch**

* In light of the fact that we will NOT be doing a lot of the ‘warming up’ exercises from classes like BIT 115 (i.e., no find-and-fix errors in the homework assignments, no tracing, etc, etc), what does the instructor HIGHLY recommend that you do for each new concept that you want to use BEFORE you try to use it in something like assignment 1?

|  |
| --- |
| Make a new blank project and create as a playground. Just play around with the syntax. |

* Instead of using a switch statement, how could we use if/else statements to accomplish the same goal?

|  |
| --- |
| When given a series of options in a long array of various characters and/or integers, they would do the same checks as an if/else chain around one input variable. |

* Copy the example switch statement from the video (when it only has case 1 and case 2), then (briefly, intuitively) explain what will happen when the user userChoice has the value 2.

|  |
| --- |
| switch(userChoice)  {  case 1:  Console.WriteLine("You typed 1");  break;  case 2:  Console.WriteLine("You typed 2");  break;  } |

* How can you have a switch statement do the same actions for when userChoice is either 0 or 1, WITHOUT duplicating code?

|  |
| --- |
| Have them right next to each other with no statements to break them  case 0:  case 1:  XXXXXX |

* In C#, what data type do you typically use switch statements on?

|  |
| --- |
| Integers |

* What limitations are there on the values after the word **case**? Can you use variables (like **case x:**)? Can you specify ranges (like **case 1 – 10:** )?

|  |
| --- |
| You can't have non-constant values called as a case.  Only if x is initialized as a constant integer.  No, syntax is poor in C# |

* How do you catch the situation where none of the (other) cases match the variable that you’re switching on?

|  |
| --- |
| default:  if (xxx xxx xxx)  xxxxxxxxxxxxx |

* Is the break statement required at the end of every case (in C#), and if so, why?

|  |
| --- |
| Yes, the syntax will require you to break because of history of cases ending and causing the function to 'fall through' to the next one. |

**VIDEO: Integer Division**

* Around the 2:20 mark the video explains how to do integer division. Briefly explain (in your own words) how to do integer division:

|  |
| --- |
| It's standard division except following the same concept as rounding something down, basically dropping decimal points. |

* What is the result of dividing x by y? How/why do you get this result?

|  |
| --- |
| The actual result would be 0.1, but because it is integer division; it drops everything after the decimal. Which is actually 0.  Thus x/y = 0 |

* What triggers integer division?

|  |
| --- |
| Integer division is provoked by dividing two numbers with no decimal points following. |

* If you wanted to divide x by 100 (in order to get, say, a percentage of the form .2 (instead of the form 20, as in 20%) ), what result would you get and why?

|  |
| --- |
| Your result would be a zero because 100 isn't a decimal number like 100.0, so it would result an integer number. |

* Why is it ok for the compiler to automatically convert an int into a double?

|  |
| --- |
| Integer variables are 32 bits big, while as double variables are 64 bits. And thus, the information in a integer value can seamlessley fit into a double. |

* In C#, when does real division happen?

|  |
| --- |
| When one or more of the data types are finer decimal data type |

* How should you choose what data type each of your variables should be?

|  |
| --- |
| The data type variable you're choosing to store data information into after all the math is done is only mirroring the result.  So if the math is done with two integers, and the result is a decimal; such as 0.2, it would be 0. And the data variable will initialize as 0 regardless if its a finer decimal or not. |

* What will the expression **x / (double) r** be (around the 8:30 mark)

|  |
| --- |
| 0.2, because it isn't calling them as integer division. |

* What are the two names for the process demonstrated in the previous question?  
  (Hint: both name begin with “type”, as in “type \_\_\_\_\_\_\_\_”)

|  |
| --- |
| typecasting and typebuilder |

* What is one common symptom of accidentally doing integer division?

|  |
| --- |
| Integer division can propogate a long domino effect of decimal-related errors. |

**VIDEO: Modulus (How does it work, mechanically)?**

* In C#, when you’ve got a literal number with a decimal part (such as 10**.0** ), what data type is it?

|  |
| --- |
| A double. |

* What problem does the following line of code have? After explaining what the problem is then re-write it so that it does not have that problem.  
  float x = 10.2;

|  |
| --- |
| float is a 32-bit data type while 10.2 is read as a double 64-bit data type. There's not enough space.  You append the number with an f to imply its a float. |

* Complete this sentence: “The modulus operator (or the \_\_\_\_\_\_\_\_\_\_ operator) gets me the \_\_\_\_\_\_\_\_\_ of doing integer division”

|  |
| --- |
| 'remainder' |

* In C#, what symbol is used for the modulus operator?

|  |
| --- |
| % |

* What is the result of 21 % 10, and why?

|  |
| --- |
| 1 |

* What is the result of 17 % 4, and why?

|  |
| --- |
| After doing arithmetic with 4 going into 17; 4 times, it has a remainder of 1. |

**VIDEO: Effective usage of the modulus operator**

* One example of where the modulus / remainder operator is useful is figuring out how many items are left over after you pack equal-sized containers full of items. The video uses the specific example of packing 15 eggs into egg cartons (where each carton can contain up to 12 eggs).   
  Explain how to find out how many cartons are completely filled up, then explain how to find out how many eggs are left over after packing that 1 egg carton completely full:

|  |
| --- |
| With integer division; you can find how many cartons are filled, and with the remainder operator; you can find out how many eggs are leftover.  1 carton and 3 remaining eggs |

* How can the idea from the prior question be applied to figure out exactly which bit (which slot) to examine in the third integer? How do you know you want to look at the third integer?

|  |
| --- |
| You would look at the 3rd slot after the computer calculates the individual bit from subtracting 12 from 15 as many times as it could. |