**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)

|  |
| --- |
| I have nothing of concern with these videos, I'm sorry. These are all well recorded, especially for live in-class sessions which are ideal for proper pragmatic usage. |

**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)?

|  |
| --- |
| Think about what you would've done in 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.

|  |
| --- |
| if (x < y) {  Console.WriteLine("Yippeee!"); } |

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

|  |
| --- |
| Visual coherency and organization |

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

|  |
| --- |
| A set of statements kept within a set of brackets |

* 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”);

|  |
| --- |
| The first Yippeee is contained in an if statement, so it would check the operator; which is false. Thus, only the two Yippee! after the first print because the three aren't contained in brackets - which don't make them block statements. Then it prints "end of program". |

* 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 with '=='  not same with '!='  less than variable than the other with '<' or '>'  equal or less with '<=', '>=' |

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

|  |
| --- |
| Anything like adding it at the end of a while loop wouldn't be kosher for C#, most commonly a syntax error. So no. |

**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?

|  |
| --- |
| Experiment with each component and play around with concepts. Just get familiar with the nuances |

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

|  |
| --- |
| Instead of using  if( userChoice == 1 ) { } else if ( userChoice == 2 ) { } else if ( userChoice == 3 ) { } else {  Console.WriteLine("Bad input"); } |

* 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;  }  The switch statement takes the variable in its input and compares it to each case situation, then acts on the first case situation. When it comes onto a break it would exit the switch case. |

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

|  |
| --- |
| switch(userChoice) {  case 0:  case 1:  xxxx }  It would just take the cases next to each other one after another. |

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

|  |
| --- |
| integers, whole numbers; sometimes Strings - maybe even chars. |

* 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:** )?

|  |
| --- |
| case x can only be a whole number and a constant variable,when establishing a constant its established that its a variable that won't change.  case 1-10 doesn't work, you need to do  case 1: case 2: case 3: etc.  meticulously one by one. |

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

|  |
| --- |
| using the equivalent to Java's catch statement is default  switch(userChoice){ case 0: xxx break; case 1: xxx break; default: xxx break; } |

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

|  |
| --- |
| Yes, its required. Its required because if you don't have it, you'd fall through to the next case unchecked. |

**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 does the division normally, but will always round down - or drop everything after the decimal point as explained in the video |

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

|  |
| --- |
| If it were any other variable besides an integer it would be 0.1  As an integer, it would be 0, because the integer does not have enough bit space to keep decimals |

* What triggers integer division?

|  |
| --- |
| Both variables need to be integers to divide |

* 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?

|  |
| --- |
| As the number 100 the program recognizes the digit as an integer  To get around this issue, add decimal points to have it become 100.0 or declaring a literal double variable  Then divide the two variables to have it not become integer division |

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

|  |
| --- |
| An int is 32 bit, and can be converted into a 64 bit because it wouldn't overload its information. |

* In C#, when does real division happen?

|  |
| --- |
| Any combination of whole number variables that don't involve exclusively integers |

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

|  |
| --- |
| Before the calculations when establishing the result |

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

|  |
| --- |
| 20/100.0 = .2  result = .2 |

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

|  |
| --- |
| type casting or type 'coercion' |

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

|  |
| --- |
| Just messing up the entire code, you're missing decimals, you can get the wrong answer a bunch of times. |

**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?

|  |
| --- |
| Its a double, 64 bits |

* 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;

|  |
| --- |
| A loss of information in the process of conversion thanks to its small bit size |

* 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: 21/10 = 2, remainder of 1 |

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

|  |
| --- |
| 4: 17/4 = 4, 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:

|  |
| --- |
| You take the total number of items and take it to the mod of the capacity that the carton can hold  15%12 = 1 with 3 leftover 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?

|  |
| --- |
| First you take the total bit amount in the integer, for example 72  Then you take the mod of the total with the container, so 72 % 32 -> 8  Then you take 72/32 to find out how many integers you need to move over, which is 2  So on the third integer you head into the 8th slot |