**Name: Patrick O’Brien**

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…"

1. 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 link to the demo review video for inheritance review is broken. |

|  |
| --- |
| **VIDEO: NUnit Overview VIDEO: Quick NUnit/Autograder How-To**  These videos were removed – there is NO NUnit / Autograding in this class anymore. Please inform your instructor if you find any references to NUnit / Autograding |

**VIDEO: Stacks: ADT, Examples**

1. What are some example of simple/primitive data types? Where is the memory needed to store these items allocated?

|  |
| --- |
| Ints, doubles, floats, bool  Local variables are stored on the stack, instance and static variables are stored on the heap. |

1. What does ADT stand for? What is the purpose of ADTs in programming?

|  |
| --- |
| Abstract data types. ADTs in programming is a language-independent view of a more complicated data type ‘pattern’.  Consists of data, and a set of actions that can be done on the type.  We’ll use the idea of a stack in many languages, and contexts.   * Clarification * Reusability * Decoupling * Encapsulation & Information Hiding |

1. Fill in the blank: “A stack is what’s called a \_\_(1)\_\_ data structure. Meaning that the \_\_\_(2)\_\_\_\_ thing that you’ve added to the stack is the first thing you’re going to get back out”

|  |
| --- |
| 1. Abstract 2. Last |

1. In a “pure stack” which item(s) are you allowed to examine/get?

|  |
| --- |
| The last element added to the stack |

1. What does the “push” method do?

|  |
| --- |
| Adds an element to the top of the stack |

1. What does the “pop” method do?

|  |
| --- |
| Removes the top element from a stack |

1. *Briefly,* intuitively summarize what a call stack is, and why it must be a stack (why does it make sense that the program will only access the top-most stack frame)?  
   (The purpose of this question is to make sure that you’ve seen a couple of examples of how to use a stack, and NOT to be able to explain this particular example in detail!)

|  |
| --- |
| A call stack in C# is a list of names of methods called at run time from the beginning of a program until the execution of the current statement.  In C#, any application beings with a “main” method, which in turn calls other methods. On every call to a method, the method is added to the top of the stack and is removed from the stack on its return to the caller.  The scope of a variable declared in a block is determined from the time its value pushed onto the stack until the execution leaves the block when the variable and the call stack are popped off the stack. Thus, the stack maintains both local variables (value types) and the call stack (stack frames), the size of which indicates the complexity of the program. -techopedia.com |

1. *Briefly,* intuitively summarize how does the ‘Undo’/’Redo’ feature use a stack(s)?  
   (The purpose of this question is to make sure that you’ve seen a couple of examples of how to use a stack, and NOT to be able to explain this particular example in detail!)

|  |
| --- |
| The undo redo feature uses a stack by storing all the prior actions in a stack. Then when you ctrl-c you are pushing a new action to the stack. When you use ctrl-v, you are popping an item off the stack and reading it. |

1. How is “Reverse Polish Notation” set up? Give a quick example:

|  |
| --- |
| 2 4 + 1 – 2 3 + =  With reverse polish notation  Take the first thing on a stack, if it’s data, push it on to a stack. Take the next thing and repeat.  If it’s an operator, figure out how many things you need. Pop the first thing off, pop the second thing off, add the two of them together, and then push it back to the stack.  Don’t need to worry about order of operations with reverse polish notation. Already specified by the order that is specified in the string. Much easier to write a parser for this type of notation. |

1. How can a stack be used to parse/calculate a mathematical expression written using “Reverse Polish Notation”?

|  |
| --- |
| 2 4 + 1 – 2 3 + + =  Take 2  Push onto stack  Take 4  Push onto stack  See +  Get top two things on the stack  Add them together 2 + 4 = 6  Push the result back into the stack  Subtract 1 and 6  Push onto stack  Add the 2 + 3  And the 5 and the 10  = sign says you’re done |

1. Why is “Reverse Polish Notation” sometimes called a “postfix” notation? What sort of notation we normally used? What is a “prefix” notation?

|  |
| --- |
| Sometimes called postfix because: Because the operator comes after post  Normally use: infix notation, where the operator is between the two numbers  Prefix notation: A prefix notation is when the operator comes before the numbers |

**VIDEO: Stacks: API, Implementation**

1. What does the acronym API stand for? In a nutshell, what is an API?

|  |
| --- |
| API: Application Programming Interface  A set of functions, methods and data types that you can use to write actual code. |

1. What we will use as the API for the **push** function?   
   (Include the return value, function/method name, any parameters, and describe any possible errors that might happen.)

|  |
| --- |
| Stack.push  If the stack is not full, add item to the top of the stack.  If the stack is full, an overflow error has occurred, and an error value is returned.  ErrorCode Push(int item); 🡪 put into the stack |

1. What we will use as the API for the **pop** function?   
   (Include the return value, function/method name, any parameters, and describe any possible errors that might happen.)

|  |
| --- |
| If the stack is not empty, then the top item is removed & returned via the out parameter.  If the stack is empty, then an underflow error has occurred, and an error value is returned.  ErrorCode Pop(out int TopVal); |

1. What does the **peek** function do (and how does it differ from pop)? What we will use as the API for the **peek** function?   
   (Include the return value, function/method name, any parameters, and describe any possible errors that might happen.)

|  |
| --- |
| If the stack is not empty, then the top item is returned via the out parameter. The stack itself is unchanged.  If the stack is empty, then an underflow error value is returned.  ErrorCode Peek(out int TopVal); |

1. What we will use as the API for the **isEmpty** function?   
   (Include the return value, function/method name, any parameters, and describe any possible errors that might happen.)

|  |
| --- |
| If the stack is empty, then true is returned. Otherwise, returns false.  Bool Stack.IsEmpty(); |

1. What C# data type(s) will we use to implement our stack class?

|  |
| --- |
| Use an array of integers to implement the stack class.  They represent the contents of the stack  An integer to keep track of the index of the top of the stack  If there are no items in the stack, set the integer to -1 |

1. For our implementation, what does the ‘top’ instance variable mean?  
   (If it has the value -1 what does that mean? If ‘top’ has the value 0 what does that mean?)

|  |
| --- |
| The top instance refers to the integer value that was last added to the stack.  If it has a value of -1 there are no items in the array. A value of 0 means that there is an item, and it has the value of 0. |

1. Intuitively, describe how our push method will work. Make sure that you describe all of the following:  
   How does the method detect that the stack is full?   
   How does the method detect that the stack is empty?   
   If the stack is not full, what is done?

|  |
| --- |
| If iTop is not yet too large, then move it up by one, and put the item in that space, then return true. If iTop = -1; then we are in the last space in the array, so it is full. Then you return -1 which is the overflow.  If ( iTop == items.Length - 1) then full  Then otherwise, there is space available. |

**VIDEO: Queue: API, Implementation**

1. A queue is referred to as a \_\_\_\_\_\_ data structure (hint: the answer is 1 word, 4 letters long, and it’s an acronym)

|  |
| --- |
| A queue is a British word for line… yeet  FIFO First-In First Out data structure.  A queue is an abstract data type in which all the insertions are made at one end of the queue (the back, or rear), while all deletions are made at the opposite end (the front).  The first entry that was added is the first entry that will be removed. |

1. When we add a new item to the queue, which end is it placed at?  
   When we remove an new item to the queue, which end is it removed from?

|  |
| --- |
| When we add a new item: New insertions are placed at the back of the queue.  When we remove an item: The item that was first added to the queue is removed. We remove items from the front of the queue. |

1. What is another word that means “add” to a queue?  
   What is another word that means “remove” from a queue?

|  |
| --- |
| Add = Queue.Enqueue  Remove = Dequeue |

1. What is the problem with a simple, naïve implementation of a queue that uses an array (i.e., what is the problem with a non-circular implementation)?

|  |
| --- |
| Have an array of stuff, add stuff at the high end ( back of the queue).  If you add a person at slot 0 and slot 1, then add someone at slot 2, the front will be around 0.  The queue starts to crawl through memory, it leaves empty space as it queues and dequeues.  Adding and deleting items results in the queue shifting to the right. |

1. If you did use a simple, naïve implementation of a queue (as an array), why is it bad to make space within the queue by shifting down all the elements in the array?   
   What is the running time to shift all the elements down by one?

|  |
| --- |
| It works, but it is incredibly inefficient.  O(N) for running time efficiency. |

1. What is the basic idea behind a **circular queue**? What happens when the reference to the back of the queue reaches the end of the array (assuming that there’s empty space at the front of the array)?

|  |
| --- |
| When rear == MaxSize – 1, add the item to array[0] if this space is free. |

1. C#’s only ternary operator (operator that takes 3 operands) is the ?: operator. Explain (concisely) what the following line of code does (the parentheses have been color-coded so they’re easier to visually match.  
   back = **(** **(**back + 1**)** == items.Length **)** ? 0 : **(**back + 1**)**;

|  |
| --- |
| ? is a compact if else  If you move up by one and it’s beyond the length of the array, then loop back around to slot 0, otherwise just use the next space that’s available. |

1. Re-write the following code snippet to use an equivalent if…else statement:  
   back = **(** **(**back + 1**)** == items.Length **)** ? 0 : **(**back + 1**)**;

|  |
| --- |
| If ((back + 1) == items.Length))  {  Items[back] = newItem;  }  Else  {  Items[Back += 1] = newItem;  } |

1. Deqeue is very similar to how Enqueue is written. Based on what’s in the slide and what was described in the audio, write out the Dequeue function here.  
   (Note – you are not required to get this to compile & run 100% correctly – please feel free to “code this up” here in Word. The main objective is to make sure that you followed the description overall)

|  |
| --- |
| ErrorCode Dequeue(int newItem)  {  If (counter >= items.Length)  Return ErrorCode.Overflow;  Counter--;  Back = ((back+1) == items.Length) ? : (back + 1);  items.[back] = newItem;  return ErrorCode.NoError;  } |