Question 1 – C++ Programming

Q1.1: Class Design

Q1.1.a: The constructor of the class takes in the size, sets the count to 0, and using the template class, the user defines the type of array to be created.

Q1.1.b: A constructor and deconstructor are defined

Q1.1.c: The function ‘numOfElements’ is defined, and takes no parameters

Q1.1.d: The operator is defined, and takes in an integer

Q1.1.e: The function ‘addElements’ is called, and taken an object of the type of the array

Q1.1.f: The function ‘printAllElements’ is called with no parameters

Q1.2 – Implementation

Q1.2.a: The constructor has one parameter, which is the user choosing the size of the array. the count is initialized to 0, and the array is created of the type that the user chooses.

Q1.2.b: The operator ‘[]’ takes an integer, and then returns the index from that number. The operator is used on the object, but then returns an element from the array.

Q1.2.c: Every time an element is added to the array, the count is incremented by 1, the function ‘numOfElements’ returns the count.

Q1.2.d: The function ‘addElements’ is called, and the first thing it checks if they counter has reached the size of the array. if this is the case, then ‘resizeArray’ is called. This function increases the ‘size’ variable by 10, and then created a new array of that new size. After that, every element from the old array is added to the new array. At the end of that, the new array is assigned the name of the old array as if nothing’s changed. Once that is done, ‘shiftElements’ is called, which moves every element to the next index value in the array, leaving the first index in the array empty. The new element to be added is put in that position.

Q1.2.e: To print all the values, ‘printAllElements’ is called. Within the function, the type of the class is checked. If the class is of type ‘double’ then the output will be in scientific notation, otherwise, it’ll be printed out normally.

Q1.3 – Testing

Q1.3.a: The integer array is initialized with an initial size of 5, and then using ‘srand’ and ‘rand’, five random numbers between 0 and 100 are added.

Q1.3.b: Five elements are added one by one, each of them going to the beginning of the array.

Q1.3.c: The double array is initialized with an initial size of 5, and then using ‘srand’ and ‘rand’, five random numbers between 0 and 1 are added.

Q1.3.b: Five elements are added one by one, each of them going to the beginning of the array.

Q1.3.e: Both arrays are printed using ‘printAllElements’ function.

Question 2 – Haskell Programming

Q2.1: Reverse Polish Notation Evaluator

Q2.1.a: The function ‘step’ takes an array of integers, and a symbol (string), and returns an array of integers. I used pattern matching to check what the symbol coming in is, if the symbol is any of the arithmetic operations, then it does those respective operations on the first two elements of the array of integers. If the symbol coming in is a number, it simply gets added to the array of integers.

Q2.1.b: The function ‘rpn’ takes an array of strings and returns an integer. In this case, the array of strings is actually the integers that need to be evaluated. The ‘foldl’ function is used with three parameters; the first being the actual function that will be used, which is ‘step’. Then we pass in an empty array, this will act as the stack, and then the list of strings. The ‘foldl’ function applies step to every single value of the array of integers and puts the returns into the initially empty

stack. Once all of the operations are done, there is only one value in the stack left, which is the answer, and that is extracted using "!!".

Q2.1.c: To implement the same function without using 'foldl' I used a helped function called "rpnRecHelp" which takes an array of integers acting as the stack (initially empty), an array of string which are the numbers that need to be evaluated, and returns an array of integers (stack). I use pattern matching in this case, the main case is where 'rpnRecHelp' is recursively called, but instead of passing the empty stack, we pass the result of step being called on the head of list of strings (an array of integers), and then the second argument is the tail of the list of strings. This is done over and over again until the base case is reached, which is when the list of strings is empty, in which case all the operations is done, and the stack is returned. Then 'rpnRec' takes that return and extracts the first value of that stack.

Q2.2: A Polish Notation Evaluator

As opposed to a reverse polish notation, the polish notation goes from right to left. To solve this, I decided to reverse the list of strings that need to be evaluated. Once they are reversed, they get passed through the 'pnStep' function, which is a slightly modified version of the 'step' function to help solve the issue with minus operations (where order matters). Once the stack is returned, the first value from the stack is extracted and is the answer.

Q2.3: Errors

Error 404: Not Found

Q2.4: Caller-Defined Extensions

Due to difficulties with the question and the time constraints, I was unable to attempt both 2.3 and 2.4, my apologies.

Question 3 – Prolog Programming

Q3.1 Recursive Predicates

Q3.1.a: Ten words are defined under 'translation', where the first value of the tuple is an English word, and the second is its equivalent French translation.

Q3.1.b: The function ‘translator’ is called, it takes two function, the base case being where either list is empty, return an empty list. The second case is where the English list is split into head and tail, and the French list is ready to accept a value into its head. ‘translation’ is used to find the French equivalent of the head of the English list, and then ‘translator’ is recursively called and passed the tail of the English list, and then returned the new tail, which acts as the tail of the French list.

Q.3.2: Shortest Path

Q3.2.a: Multiple ‘edge’ facts are created where two parameters are the connecting vertices, and the final parameter is the weight of the corresponding edge.

Q3.2.b: To handle this, I used a ‘check_connected’ function, which takes in two vertices and a length, and compares it to ‘edge’. If not found, the two vertices are swapped in position, and them compared to ‘edge’ again. This ensures that it will find the edge, if it exists between two vertices.

Q3.2.c: The function ‘path_taken’ is called, and passed the beginning vertex, the destination vertex, and the path and length are to be found. Within this function, I called another function named ‘travel_path’ which gets passed all the same parameters as ‘path_taken’ and an extra parameter to keep track of all the visited vertices, which already has the beginning vertex added to it. The first thing it does is check if V1 and V2 are connected (V1 being the starting vertex, V2, being the destination vertex), if they are, the length is returned, and V2 is added to the visited list. Then, it checks connected, but for anything V1, rather than just V2. If there is something, it is added to the visited, and the function is called again until the first connection check is true. All the lengths are added, and the paths returned.

Q3.2.d: The paths returned from ‘path_taken’ are sorted, and the first value is returned as the shortest length.

Q3.3: Constrains Solving

Due to lack of time, could not attempt, sorry.