Simple recursions

Program 1:#

In this exercise we practice relatively simple recursions. All the tasks you are required to perform can, and therefore usually should, be performed using a loop, but you, in this exercise, will accomplish them using recursion, so your plan will not contain any loop command whatsoever. (Note that you should first set the task for yourself as an inclusion and only then translate it into a recursion. Careful and careful planning will make it very easy to write the exercise.)

Also note that the various tasks are 'sewn' into one plan arbitrarily, so do not look for logic in the fact that the threshing floor was built near the winery.

Write a plan that defines an array of ten integers, and initializes the array cells to a zero value.

The main program will schedule a sub-function that will run as a loop (which like any other loop will be implemented through recursion) in which the user can request to perform the following actions:

1. Entering values for the array (the user will enter ten data, and it can be assumed here, as in the rest of the program, that the input is correct).

2. Schematic of all members of the array and presentation of the result.

3. Sort the array using the bubble sorting algorithm. (Hint: each loop in the algorithm will be converted to a recursive function, meaning you will have two recursive functions).

4. Binary search of a desired value in an array. The user will enter the value, the output will be 1 if the value is in the array, and 0 if not. Hold a Boolean variable that tells whether the array is already sorted by the program, and allow the task only if the array is sorted, otherwise send a message: Error \ n "" to stderr (but you will stop the execution of the program).

5. Checking whether the array members constitute an invoice series. The output will be 1 if yes, and 0 if not.

6. Displays the contents of the array (from beginning to end).

7. Displays the contents of the array from end to beginning (i.e. the last cell will be displayed first, and the first cell will be displayed last). (One space should be displayed after each cell and at the end of the line \ n)

8. Examine whether each of the arrays (starting from the third limb) is the sum of the two limbs that precede it. The output will be 1 if yes, and 0 if not.

9. Checking whether the array is a plinth, meaning it is symmetric. The function must work properly for both an even-sized array and a Freddie-sized array. (The following arrays are plinths: {1,2,3,2,1}, {1,2,3,3,2,1}).

10. Examine how many subsets constitute an ascending or descending series and present the result. For example, for the array {9,8,6,4,1,3,6,4,5,9} the value 3 is displayed, since the sub-array {9, 8, 6, 4, 1} is descending, the sub-array {3, 6} it rises and the subset {4, 5, 9} it rises. Note that if two members are equal, the subseries continues (i.e. we are looking for an ascending or descending series and not a series ascending or descending really). Also, the first limb that does not meet the directionality of the current subseries will be the first limb in the next subseries. In light of the definition, it is not possible for a sub-series whose number of members is less than 2, except in the case where the last member in the series begins a new sub-series.

11. Displays the maximum member within the subset, with the user entering the position of the first member in the subset and the position of the last member in the subset. If the user enters incorrect values, the maximum organ must be displayed in the entire array.

Invalid values are those that exceed the boundaries of the array, or that the position of the beginning of the subseries is higher than the position of the end of the subseries.

12. Displays the values in the array that hold that initial value. The values must be displayed in the order in which they appear in the array.

13. Finish.

Each of the options listed will be selected by entering a number (ie by entering a value between 1 and 13). After selecting the desired option, the user will enter the required input (if any), and will be presented with the required output (in a single line). After the output is displayed, the program will wait for the user's next selection. Define in your program an enum type that will suit the various options (and so you will practice this topic again). The menu function (the one that reads the desired selection from the user, and returns the corresponding value to the main function of the program, the one that calls each and every function to perform the various tasks) will return a value from the above enum type. The function will repeat the reading process Appropriate value.

In this program the main program is quite degenerate, (since it is not appropriate for the main to call itself recursive. Therefore the main will schedule a function that will run the main loop of the program (by recursive reading to itself).

Remarks:

A. The exercise file name will be ex8a.cc

B. It can be assumed that the input is correct in every sense (for example: you will not be asked for an option that does not exist, the user will first enter input into the arrays and only then will perform the tests on them, and so on).

C. The output of each section will appear in a separate line.

D. If necessary, you may also configure additional arrays.

E. The question sections are not arranged easily, so you may want to write the various functions not necessarily in the order in which they appear in the question.

F. Divide the program into functions as you see fit, and do not shy away from multiple functions. There is not a certain amount of functions that is not correct to pass it, as long as there is a logical logic in the division.