SW483, Summer 2017
CS131 Homework Assignment #12, “List Checker”, in Lisp!

A List Checker Program

In this assignment you will implement a program that records a sequence of 10 integers entered by the user, and checks to see if the sequence of integers was in increasing (also called “non-decreasing”) order. If the sequence is in order, the program prints the message “List is in order.” to the console. Otherwise the program prints “List is not in order.”

Here is an example of what the input and output from your program should look like:

```
Enter ten integers separated by spaces, then press return.
10 3 0 -4 200 80 -95 87 23 -67
List is not in order.
```

Here is another example:

```
Enter ten integers separated by spaces, then press return.
-5 -4 -3 0 1 2 2 3 10 11
List is in order.
```

Notice that, even when there are repeated numbers, as long as the sequence never decreases in value from one number to the next, the list is considered to be in order. So, sequences like “0 0 0 0 0 0 1 1 1 1” are considered to be in order. Your program needs to print the messages **exactly** as written above for the autograder. “List” must have a capital “L” and there must be a period after “order”.

There are many ways to implement this program, and there are no constraints on the number of calls to setq or format. However, we strongly suggest that you attempt to implement the program as follows: We suggest that you implement your program so that it stores the integers entered by the user by storing them in a list, just as with the ArrayRecorder and ArrayReverser assignments. You will need to place the code that accepts the integers from the user in a loop (a while or for loop) that executes 10 times. We also suggest that you place the code that checks the sequence of integers to see that it is non-decreasing in another loop that executes 9 times (more details on how to do this below). So, in total, you’ll need to create two separate loops.

Your starter code for this assignment will be a file called list_checker.lisp, which takes 10 integers from the console and has a loop, but is otherwise incorrect because it does not print them back out to the console in reverse. This file is basically identical to the starter code for the previous two assignments. You can compile, run, and test the incomplete program using the same steps as for previous assignments. There is a zip archive containing the code (list_checker.lisp).

To compile and run the program:

1. Download ListChecker.zip. It contains a Lisp source code file, list_checker.lisp, a bash shell script, assignment_12_grader.sh, and a file with test cases: test_cases.csv.

2. Use the bash shell and change directories to the folder created by unpacking the .zip file.

3. Then simply execute "clisp list_checker.lisp" and follow the instructions in the console to run the program.

Checking that the List is in Order

After reading the integers from the console into a list, your program will need to check that the sequence was in order. To check whether the sequence of integers is in order from lowest to highest, your program only needs to compare pairs of adjacent elements in the list to assure that the one appearing later in the sequence is equal to or larger than the one appearing earlier in the sequence.

For example, say that your program stores the integers in an list under the symbol "mylist". Then if the first element of the list, which you could access using "(nth 0 mylist)", is smaller than or equal to the second element of the list, which you could access using "(nth 1 mylist)", then the list may be in order (you still need to check the rest of the array). However, if the first element is larger than the second element, then the list is definitely not in order. Similarly with the adjacent pairs of elements, "(nth 1 mylist)" and "(nth 2 mylist)", the pair "(nth 2 mylist)" and "(nth 3 mylist)", and so on, up to the last pair "(nth 8 mylist)" and "(nth 9 mylist)".

We suggest that you start the array checking part of your program by assigning a symbol variable (you might call it "isInOrder") the value T. Then compose a loop, like you have in previous assignments. One crucial difference: your counter variable or loop index (let's assume you named it "i") rather than going from 0 to 9, it should range from 0 to **8**.

Inside the body of the loop, you should place an "if" expression. The test expression for the if should be an expression that evaluates to T if the element of the list at index i is greater than the element of the list at i+1. As "i" ranges from 0 to 8, this expression evaluates to T when adjacent elements of the array have the earlier element larger than the later element, indicating that the list is not in order. Inside the body of this if statement, you should assign the value NIL (roughly the same as "false" in Java) to isInOrder. Notice that when i is 8, that the expression (+ i 1) which you would use in the "nth" function on mylist, will be 9, the index of the 10th and last element. Since the array has 10 elements, you only have to make 9 comparisons of adjacent elements. If the expression comparing the adjacent array elements never evaluates to T, then all of the checks pass, and the list is in order. The symbol isInOrder, which was assigned the value T originally, keeps the value T in that case.

After the while loop, your program should have an if or cond which prints "List is in order." to the console when isInOrder has the value T, and "List is not in order." when isInOrder is NIL.

Extra Credit: Using Recursion

Another way to do this is to use recursion. You can implement a recursive function, called, say, "recursive-check-list" which takes an s-expression list of numbers as an argument. In previous recursive functions, our argument was a number, and our base case was to test that our input was 0 or 1. In this case the argument is a list, and our tests will compare adjacent elements of the list. If you have a list with at least two elements, you can obtain the first element of the list using the "first" function (also known as "car") on the list, and you can obtain the second element of the list using the "second" function (also known as "cadr") on the list.

Our cond in the recursive function will be somewhat complicated but it will eliminate the need for an extra "isInOrder" symbol and the extra "if" on the end if you do it right. Here are the cases to consider in the cond

1. If the list has at least two elements, then compare the first two elements of the list.

    a. If the first element is greater than the second element, then for certain the list is not in order. We can then use "format" or "print" to print "List is not in order." and do nothing else. We don't need any recursive calls.

    b. If the first element is not greater than the second element, then the list may or may not be in order. In that case, don't print anything and simply recursively call recursive-check-list on the remainder of the list without the first element, which you can obtain using the "rest" function (also known as "cdr").

2. If the list only has 1 element or it has no elements, this means that our recursive calls have checked all the pairs of adjacent elements in the list, and there were no cases when an earlier element in a pair was larger than a later element in the pair. Therefore the list is in order. We can then use format to print "List is in order." and do nothing else. This is our "base case" and no other recursive calls or evaluations are needed.

> Note that we can't try to make our base case the empty list, because then one of our recursive calls would try to call "second" or "cadr" on a list with only one element, which will return NIL. Then your code would attempt to compare a number to NIL, which will raise an error. Our base case is either an empty list or a list with 1 element, which you can test for using the expression "(null (cadr x))" (there are other ways to do this as well).

Then simply call recursive-check-list on mylist to check it. You won't need an extra "isInOrder" symbol or an "if" expression to print the result because your recursive function takes care of everything.

Because of the way cond works (it tests its expressions in order and ignores the later expressions if one of them matches), we suggest that you make the first s-expression in the cond be the base case (number 2 above), then the second can test for the first element of the list being greater than the second (1a above), and finally the third s-expression (with the test expression "T") can be the recursive call (1b above).

Running the Autograder Script

Run the autograder script by executing "./assignment_12_grader.sh" from the bash shell in the zip directory. Your Lisp file must be named "list_checker.lisp" with a lower-case "l" and all other letters lower-case.

The script will attempt to compile your list_checker.lisp and will test your code by executing it using the clisp interpreter on a number of inputs. When it completes, the autograder script will print your final score (out of 100) to the console. Ignore any compiler warnings.

The autograder does not run your program with a timeout, so be careful. If your program loops infinitely it will freeze the autograder script and you will need to press Ctrl-C to exit it.