CSE 216 – Programming Abstractions (Spring 2019)

Programming Assignment # 4

In this assignment, you will create SML and Python programs which use recursion. The assignment has several questions together comprising 50 points. The programming languages to be used to solve the problem are specified next to the question.

Question I. Quicksort (Python and SML) **(14 pts)**

Quicksort is a divide and conquer algorithm. Quicksort first divides a large array into two smaller sub-arrays: the low elements and the high elements. Quicksort can then recursively sort the sub-arrays. The steps are:

- Pick an element, called a pivot, from the array.
- Partitioning: reorder the array so that all elements with values less than the pivot come before the pivot, while all elements with values greater than the pivot come after it (equal values can go either way). After this partitioning, the pivot is in its final position. This is called the partition operation.
- Recursively apply the above steps to the sub-array of elements with smaller values and separately to the sub-array of elements with greater values.
- The base case of the recursion is arrays of size zero or one, which are in order by definition, so they never need to be sorted.

The pivot selection and partitioning steps can be done in several different ways. You may would like to choose first element or middle element or last element of a list as pivot.

Pseudocode of quicksort algorithm:

```
/* low  --> Starting index,  high  --> Ending index */
quickSort(arr[], low, high)
{
    if (low < high)
    {
        /* pi is partitioning index, arr[pi] is now
           at right place */
        pi = partition(arr, low, high);

        quickSort(arr, low, pi - 1);  // Before pi
        quickSort(arr, pi + 1, high); // After pi
    }
}
```

Your tasks are the following:

1) Create a Python program recQsort.py which provides a recursive implementation of Quicksort algorithm using a function quickSort(num, 0, len(num)-1) where num is a list of numbers. Test your function using the following:

```
>>>num = [26, 4, 90, 14, 52, 11, 95, 83, 35, 31, 98, 30, 81, 32,
67]
>>>quickSort(num, 0, len(num)-1)
[4, 11, 14, 26, 30, 31, 32, 35, 52, 67, 81, 83, 90, 95, 98]
```

2) Create a SML program recQsort.sml which provides a recursive implementation of Quicksort algorithm using a function qsort(num, acc) where num is a list of numbers and acc is an accumulator to hold the result. Test your function using the following:

```
- val nums = [26, 4, 90, 14, 52, 11, 95, 83, 35, 31, 98, 30, 81,
32, 67];
val nums = [26,4,90,14,52,11,95,83,35,31,98,30,...] : int list
- quickSort (nums,nil);
val it = [4,11,14,26,30,31,32,35,52,67,81,83,...] : int list
```

Your implementation of quickSort may also not use the accumulator and may have variable number of parameters.

Rubric:

recQsort.py: 7 points
recQsort.sml: 7 points

Question 2. Mergesort (Python and SML) **(14 pts)**

Merge Sort is a Divide and Conquer algorithm. It divides input array in two halves, calls itself for the two halves and then merges the two sorted halves. The merge() function is used for merging two halves. The merge(arr, l, m, r) is key process that assumes that arr[l..m] and arr[m+1..r] are sorted and merges the two sorted sub-arrays into one.

Pseudocode of mergesort algorithm:

```
MergeSort(arr[], l,  r)
If r > l
     1. Find the middle point to divide the array into two halves:
             middle m = (l+r)/2
     2. Call mergeSort for first half:
             Call mergeSort(arr, l, m)
     3. Call mergeSort for second half:
             Call mergeSort(arr, m+1, r)
     4. Merge the two halves sorted in step 2 and 3:
             Call merge(arr, l, m, r)
```

Your tasks are the following:

1) Create a Python program recMsort.py which provides a recursive implementation of Mergesort algorithm using a function mergeSort(num) where num is a list of numbers. Test your function using the following:

```
>>>num = [26, 4, 90, 14, 52, 11, 95, 83, 35, 31, 98, 30, 81, 32,
67]
>>>mergeSort(num)
[4, 11, 14, 26, 30, 31, 32, 35, 52, 67, 81, 83, 90, 95, 98]
```

2) Create a SML program recMsort.sml which provides a recursive implementation of Mergesort algorithm using a function msort(num, acc) where num is a list of numbers and acc is an accumulator to hold the result. Test your function using the following:

```
- val nums = [26, 4, 90, 14, 52, 11, 95, 83, 35, 31, 98, 30, 81,
32, 67];
val nums = [26,4,90,14,52,11,95,83,35,31,98,30,...] : int list
- mergeSort (nums,nil);
val it = [4,11,14,26,30,31,32,35,52,67,81,83,...] : int list
```

Your implementation may also not use the accumulator and may have variable number of parameters.

Rubric:

recMsort.py: 7 points
recMsort.sml: 7 points

Question 3. Recursive programs (Python) **(11 pts)**

Create a Python file recursion.py and write recursive Python functions along with test cases for the following tasks:

```
# 3.1 Computes the sum of all the even elements in the list u.       (2 pts)
# sum_evens([1, 2, 3, 4] returns 6
# sum_evens([1, 2, 3, 4, 5, 6, 7, 8, 9, 10] returns 30
#
def sum_evens(u):
    return None  # Replace this with your implementation


# 3.2 Finds the even elements in the list u and returns their indices as a
list.                                                                  (2 pts)
# find_even_indices([1, 2, 3, 4] returns [1, 3]
# find_even_indices([1, 2, 3, 4, 5, 6, 7, 8, 9, 10] returns [1, 3, 5, 7, 9]
# Note how i parameter is used in this design.
#
def find_even_indices(u):
    return find_even_indices_aux(u, 0)

def find_even_indices_aux(u, i):
    return None  # Replace this with your implementation


# 3.3 Returns the zip of two strings s1 and s2 of the same length.   (2 pts)
# zips('ftp', 'abc') returns 'fatbpc'
#
```

```
def zips(s1, s2):
    return None  # Replace this with your implementation

# 3.4 Finds and returns only the vowels in the string s in the same order
they appear in s.                                                    (2 pts)
# find_vowels('ftp') returns ''
# find_vowels('Apple') returns 'Ae'
# find_vowels('Banana Republic') returns 'aaaeui'
#
def find_vowels(s):
    return None  # Replace this with your implementation


# 3.5 Finds the vowels in a string s and returns their indices as a string. (3
pts)
# find_vowel_indices('Apple') returns '04'
# find_vowel_indices('Banana Republic') returns 13581013
# Note how i parameter is used in this design.
#
def find_vowel_indices(s):
    return find_vowel_indices_aux(s, 0)

def find_vowel_indices_aux(s, i):
    return None  # Replace this with your implementation
```

Question 4. Recursive programs (SML) **(11 pts)**

Create a SML file recursion.sml and write recursive SML functions along with test cases for the following tasks:

4.1 Write a function reverse : 'a list -> 'a list to reverse a list. (2 pts)

```
- reverse ["a", "b", "c"];
val it = ["c","b","a"]
```

4.2 Write a function compress to remove consecutive duplicates from a list. (2 pts)

```
-compress ["a","a","a","b","c","c","a","a"];
val it = ["a","b","c","a"] : string list
```

4.3 Write a function cluster that uses accumulator to cluster consecutive duplicate into nested lists. (3 pts)

```
-cluster(["a","a","a","b","c","c","a","a"], []);
val it = [["b"],["a","a","a"],["c","c"],["a","a"]] : string list list
```

4.4 Define a function sumlists: int list * int list -> int list which takes in input two lists of integers and gives as result the list of the sums of the elements in corresponding position in the input lists. The shortest list has to be seen as extended with 0's. Examples: (2 pts)

```
sumlists([],[]) = []
sumlists([1,2],[3,4]) = [4,6]
```

```
sumlists([1],[3,4,2]) = [4,4,2]
sumlists([1,6],[3]) = [4,6]
```

4.5 Define a function flatten: 'a list list -> 'a list which takes in input a list of lists and gives back the list consisting of all the elements, in the same order in which they appear in the argument. Examples: (2 pts)

```
flatten [] = []
flatten [[]] = []
flatten [[1,2],[2,3,4],[5],[],[6,7]] = [1,2,2,3,4,5,6,7]
flatten [["a"],["b","a"]] = ["a","b","a"]
```

Assignment submission: Submit only following files (no zip files) which include a number of test cases for each program:

- recQsort.py
- recQsort.sml
- recMsort.py
- recMsort.sml
- recursion.py
- recursion.sml

Assignment Evaluation: A slot will be announced for evaluating the assignments. During this time the instructor/TAs will download and run your program and it will be checked for correctness using a few use-cases. If required, instructor/TA will contact you for clarifications.

Note:

If your code does not compile, it will not be graded.

Late submissions will not be accepted under any circumstances.

To be safe, always, ALWAYS, prepare to submit ahead of time, not exactly AT last moment!

Submission deadline: Friday 17 May, 11:59 PM