CSE 505-Spring 2019

Assignment 2 – Python and ML

(may be done by a team of two students)

Due: **Fri, March 15, 2019** (11:59 pm, online code submission)

Problem 1: Stacks. Consider the Python program discussed in Lecture 8, slide 36:

```
def inorder_gen(tr, thk):
      def thk2(x):
            thk(x)
      if tr != None:
            inorder_gen(tr.left, thk2)
            thk(tr.value)
            inorder_gen(tr.right,thk2)

tree = node(20, node(10,None,None),node(40,node(30,None,None),None))

def thunk(x):
      print(x)

inorder_gen(tree, thunk)
```

Draw the scope and stack diagrams at the point in execution when `thunk` at the top-level is called:

(i) For the scope diagram, you need to only show the nesting of frames, the name of each frame and its dynamic link.

(ii) For the stack diagram, you need to show the order of frames on the stack, the name of each frame and its static and dynamic links.

Note: For the above program, the names of frames are of the form: `inorder_gen`^{1}, `inorder_gen`^{2}, …, `thk2`^{1}, `thk2`^{2}, …, and `thunk`^{1}.

Save your diagram as a file called `scopestack.png`.

Problem 2: Generators. Define a Python generator, called `flatten`, that takes as input a list of integers with arbitrary levels of nesting and *yields* one by one the integers in the list in left-to-right order. E.g.,

```
flatten([[[1],2],[[[[[3]]]]],[4,[5],[[6]]]])
```

should *yield* one by one the values 1, 2, 3, 4, 5, and 6. Test your program by executing:

```
inlist = [[[1],2],[[[[[3]]]]],[4,[5],[[6]]]

outlist = [x for x in flatten(inlist)]

print(outlist)
```

The printed list should be `[1, 2, 3, 4, 5, 6]`. Create a file `flatten.py` containing your definition of `flatten` and the above statements for testing `flatten`.

Problem 3: Tail Recursion. Consider the following ML function for depth-first traversal of a binary search tree and formation of a list of values in ascending order.

```
datatype 'a tree = leaf | node of 'a * 'a tree 'a tree;

fun dfirst(leaf) = []
  | dfirst(node(v,t1,t2)) = dfirst(t1) @ [v] @ dfirst(t2);
```

Develop a tail-recursive version of `dfirst`, called `dfirst2`, as follows. Write a helper (inner) function `df: 'a tree list * 'a list → 'a list`, which uses an accumulator-passing style in order to construct the answer.

Starter code for your solution and a tester function are provided in file `dfirst2.sml` posted on Piazza. Your task is to complete the definition of function `dfirst2`.

Problem 4: Higher-order Functions. The ML type definition below is for a general tree, called ntree, where each internal node has a *list of zero of more subtrees* and each leaf node holds a single value:

```
datatype 'a ntree = leaf of 'a | node of 'a ntree list;
```

4a. Using the `map(f,l)` higher-order function, define a function `subst(tr,v1,v2)` which returns a new ntree in which all occurrences of `v1` in the input ntree `tr` are replaced by `v2` in the output tree. For example,

```
subst(node([leaf("x"), node([leaf("y"), leaf("x"), leaf("z")])]), "x", "w")
    = node([leaf("w"), node([leaf("y"), leaf("w"), leaf("z")])])
```

4b. Using the `reduce(f,b,l)` higher-order function, define a function `toString(tr)` which returns the concatenation of all strings at the leaf nodes of `tr`, adding a space after each value. For example,

```
toString(node([leaf("x"),node([leaf("y"),leaf("x"),leaf("z")])])) = "x y x z "
```

Starter code for both parts and testing code are provided in file `ntree.sml` posted on Piazza. Your task is to complete the definitions of `subst` and `toString`.

WHAT TO SUBMIT:

Prepare a top-level directory named *A2_UBITId1_UBITId2* if the assignment is done by two students; otherwise, name it as *A2_UBITId* if the assignment is done solo. (Order the *UBITId*'s in alphabetic order, in the former case.)

In this directory, place `scopestack.png`, `flatten.py`, `dfirst2.sml`, and `ntree.sml`.

Compress the directory and submit the resulting compressed file using the `submit_cse505` command. Only one submission per team is required.

End of Assignment #2