CIS 705: Programming Languages Exam II with suggested answers

Wednesday, November 4, 2015 2:30-3:20pm

General Notes

- You can use any amount of printed or written material.
- You may not use laptops, cell phones or any other computing devices.
- All programming answers must be in Standard ML
- If you believe there is an error or ambiguity in any question, mention that in your answer, and *state your assumptions.*
- Please write your name on this page.

Good Luck!

NAME:

1. *Recursive Functions, 40p.*

A. (20p) Write a function that concatenates each element in a binary tree (without interior values) of strings together into one large string. You may not use any SML built-in functions except concatenate (^). You may use the following definition of a binary tree:

```
datatype bi_tree =
    Leaf of string
|   Node of bi_tree  bi_tree ;
```

Answer:

```
fun conc_tree(Leaf(n)) = n
| conc_tree(Node(bt1, bt2)) =
    conc_tree(bt1) ^ conc_tree(bt2);
```

B. (10p) Draw the call tree for your function using the input:

```
val myBiTree = Node(Node(Leaf("KC"),Leaf("Won")),Node(Leaf("The"),Leaf("WS")));
```

```
         *
     /      \
    *        *
  /  \     /  \
KC  Won  The   WS
```

Answer:

```
                        conc_tree(N(N(LKC,LWon),N(LThe,LWS)))
                          /                             \
      conc_tree(N(LKC,LWon))          ^           conc_tree(N(LKC,LWon))
       /             \                             /                \
(conc_tree(LKC)  ^  conc_tree(LWon))  ^    (conc_tree(LThe)  ^   conc_tree(LWS))
       |                |                        |                    |
   ("KC"         ^    "Won")          ^       ("The"         ^        "WS")
       \               /                         \                    /
             "KCWon"                  ^                "TheWS"
                 \                                        /
                                "KCWonTheWS"
```

C. (10p) We now want to consider neighbor leaves in our binary trees. Two leaves are said to be neighbors if and only if they share the same immediate parent – for the purpose of this problem, we do not consider non-leaf nodes to have neighbors. Rewrite your function from A to swap each leaf with its neighbor before concatenation (but be sure not to swap non-leaf nodes!). For example, the previous input should result in "WonKCWSThe."

Answer:

```
fun conc_swap_tree(Leaf(n)) = n
| conc_swap_tree(Node(Leaf(n),Leaf(m))) = m ^ n
| conc_swap_tree(Node(bt1,bt2)) = conc_swap_tree(bt1) ^ conc_swap_tree(bt2);
```

2. *Datatypes, 30p.*

A. (15p) Give a datatype definition for a day of the week and month, i.e., Sunday 1, Monday 2, . . . , Friday 6, Saturday 7.

Answer:

```
datatype day =
    Sunday of int
|   Monday of int
|   Tuesday of int
|   Wednesday of int
|   Thursday of int
|   Friday of int
|   Saturday of int
```

B. (10p) Declare a type called `appt` (not a datatype!) that would correspond to an event in a calendar. Your type must include a day (for the day of the event), an integer (for the time of the event, in military-style time, ie 1400 = 2pm), and a string for the event name.

```
type day_appt = (day * int * string);
```

C. (5p) Use your type to create a calendar entry for an event of your choice (for example, today's exam).

Hint: Be sure to manually specify the type so SML doesn't assume it's a more general type!

```
val myDayAppt = (Wednesday 4, 1430, "CIS 705") : day_appt;
```

3. *Functional Paradigm, 30p.*

A: Map, Filter, and Fold (15p) Give a brief, high-level explanation of `Filter`, `Foldl`, and `Foldr`, including what they take as input and produce as output. As an example, an explanation of `Map` has been provided for you.

`Map`... applies an arbitrary function to each element of a given list, and produces the resulting list.

Answer:

`Filter` Uses an arbitrary function (predicate) to remove elements from a given list, producing a list of those elements which evaluate to "true" according to the predicate.

`Foldl` Uses an arbitrary function and initial value to combine elements from a given list – starting on the left and working right – producing a single resulting value of the type of the initial value.

`Foldr` Uses an arbitrary function and initial value to combine elements from a given list – starting on the right and working left – producing a single resulting value of the type of the initial value.

B: Lists and Tuples (15p) What is the difference between a list and a tuple?

Answer: Lists store an indeterminate number of elements of the same type, and are typically iterated over, with some operation being performed on each element individually. Tuples store a fixed number of elements of (potentially) different types and are generally treated as a monolithic unit.

Give an example of something that would be best described as a list.

Answer: The items to buy at a grocery store are well described as a list – something needs to be done to each element (buy it) and each element is of the same type (an item to be purchased).

Give an example of something that would be best described as a tuple.

Answer: A user's location is well described as a tuple – their latitude, longitude, and altitude are most useful when considered simultaneously, and there aren't meaningful operations we'd do on each element individually (i.e., we'd never want to multiply each element by some other number).