UNIVERSITY OF PUERTO RICO AT BAYAMON

**COTI 4039 – COMPARATIVE PROGRAMMING LANGUAGES**

**ASSIGNMENT #4 – 80 points**

**NOTE**: It is not allowed to use features of the programming language that have not been discussed in class, assigned to read or explicitly authorized.

1. (15 points) Write a F# program (employee\_payroll.fs) that creates a list of at least five different kinds of employees and displays their data, including their weekly salary with the appropriate format. Each employee has an id and a full name. There are two kinds of employees: hourly employees and sales employees. Each hourly employee has a pay rate and a number of hours worked. Each sales employee has a commission rate and a sales amount. Thus, you should use the following data structure:

type Employee =

| HourlyEmployee of string \* string \* float \* float

| SalesEmployee of string \* string \* float \* float

The weekly salary of an hourly employee is calculated as the product of the pay rate and the hours worked. If the hours worked is greater than 40, an overtime rate of 1.5 is applied. On the other hand, the weekly salary of a sales employee is calculated as the product of the sales amount and the commission rate.

3. (25 points) Modify linked\_list.fs to add the following:

* A function that returns the number of elements in the given list. Its signature should be:

length\_list: IntList -> int

* A function that returns the minimum element in the given list. Its signature should be:

minimum\_list: IntList -> int

* A function that reverses the elements in the given list. Its signature should be:

reverse\_list: IntList -> IntList

* A function that removes an element from the given list. Its signature should be:

remove\_list: IntList -> int -> IntList

* A function that concatenates the given lists. Its signature should be:

append\_list: IntList -> IntList -> IntList

4. (20 points) Modify binary\_tree.fs to add the following:

* A function that returns the binary search tree that corresponds to the given list. Its signature should be:

construct\_tree: int list -> IntTree

* A function that returns the number of elements in the given tree. Its signature should be:

size\_tree: IntTree -> int

* A function that returns the number of levels in the given tree. Its signature should be:

depth\_tree: IntTree -> int

* A function that returns the minimum element in the given tree. It should use a tail-recursive helper function and its signature should be:

minimum\_tree: IntTree -> int

5. (20 points) Write a F# source file (queue.fs) that implements the Queue data type using a linked list as its data structure. A queue is a list in which elements are added in one end and removed from the opposite end. The interface for this data type is located on queue.fsi and you can test your implementation using queue\_menu.fs. When implementing the queue, use the following data structure:

type Queue = QueueList of 'a list \* 'a list

**Hint:** Although a queue can be implemented using a single list, this is inefficient since a complete traversal of the list would be needed to add a new item to the queue. A more efficient alternative is to implement the queue using a tuple containing two lists: a list is used to remove items and another list is used to add items.

In this implementation, items are always added at the start of the second list. This means that the elements in the second list are maintained *in reversed order*. On the other hand, the items are always removed from the start of the first list. If the first list is empty, the elements from the second list are removed and placed in the first one but *reversed*.

This is the internal state of an empty queue:

QueueList ([], [])

This is the internal state of the queue after adding the following items: ‘a’, ‘b’ and ‘c’:

QueueList ([], ['c';'b';'a'])

This is the internal state of the queue after removing an item:

QueueList (['b';'c'], [])

This is the internal state of the queue after removing another item:

QueueList (['c'], [])

This is the internal state of the queue after adding the following items: ‘d’, ‘e’ and ‘f’:

QueueList (['c'], ['f';'e';'d'])

**Note:**

Do not forget to document your programs with the name of the source file, your name and student number, the date of creation and the purpose. Document each function with a sentence that summarizes what it does, what it receives and what it returns.