CS 2SC3 and SE 2S03 Fall 2008

Programming Exercise 4

Instructor: William M. Farmer

Revised: 9 November 2008

Files due: 20 November and 27 November 2008

The purpose of this programming exercise is to learn how to use linked lists in OCaml and C by implementing and testing an abstract data type of queues.

Background

An *abstract data type of queues* can be defined as having the following components:

1. A type nat of natural numbers.
2. A type element of elements that are put in a queue.
3. A type queue of queues.
4. A constructor bottom : void → queue that creates an empty queue.
5. A selector length : queue → nat that returns the length of a queue.
6. A selector front : queue → element that returns the element on the front of a queue.
7. A mutator push : element, queue → void that adds an element to the back of a queue.
8. A mutator pop : queue → void that removes an element from the front of a queue.

Part A

Write an OCaml program that satisfies the requirements listed below. Put your implementation of the abstract data type of queues in a file named `queue4.ml`, your test code in a file named `queue_test4.ml`, and your log book in a file named `log.txt`. Put all three of these files into a directory named `ex-4-a`. Using subversion, import this directory into your directory in the course subversion repository. Your files must be submitted no later than **10:30 a.m. on Thursday, November 20, 2008.**

Part B

Write a C program that satisfies the requirements listed below. Put your implementation of the abstract data type of queues in a file named `queue4.c`, your main procedure and test code in a file named `queue_test4.c`, and your log book in a file named `log.txt`. Put both of these files into a directory named `ex-4-b`. Using subversion, import this directory into your directory in the course subversion repository. Your files must be submitted no later than **10:30 a.m. on Thursday, November 27, 2008.**

Program Requirements

1. The program implements the members of the abstract data type of queues described above as linked lists of records of type `queue_node`.
2. The type `nat` is implemented as `int`.
3. The type `element` is implemented as a type of records with two immutable fields: (1) a field `data` of type string and (2) a field `id` of type `nat`.
4. The type `queue_node` includes the following two fields: (1) an immutable field `contents` of type `element` and (2) an mutable field `next` of type `queue_node` reference.
5. The type `queue` is implemented as a type of records including the following two fields: (1) a mutable field `back` of type `queue_node` that points to the back of the queue and (2) a mutable field `front` of type `queue_node` that points to the front of the queue
6. The program implements the constructor, two selectors, and two mutators described above.
7. The program should handle the following two anomalous situations:
   (a) Selecting the front of an empty queue.
   (b) Popping an empty queue.
8. Nothing in your implementation should limit the length of a queue.
9. The program tests the implementation of the abstract data type of queues by making a representative series of calls to the constructor, selectors, and mutators. The results of these calls is printed out when the program is executed.

Notes:

1. Please put your name and MAC ID at the top of each of your files.
2. Your programs must be your own work.

3. Your programs must compile and execute correctly on either mills or moore to receive full marks.

4. Files submitted late will receive no marks.