CIS 120 Midterm II November 7, 2014

SOLUTIONS

1. General Understanding of OCaml and Java: True/False (20 points)

Circle T or F.

a. [T] F In both Java and OCaml, if `v1` and `v2` are references, then the expression `v1 == v2` evaluates to true exactly when they alias.

b. T [F] The following OCaml function will terminate by exhausting stack space if called as `loop 10`.

```
let rec loop (x:int) : int =
  if x = 0 then 1 else loop x
```

c. [T] F There is a type that can be filled in for the blank in the following OCaml program to make it well typed.

```
type 'a ref = { mutable contents : 'a }

let x : _____________________ = { contents = [] }
;; x.contents <- ((fun () -> ()) :: x.contents)
```

d. T [F] A well-typed OCaml program can fail with an error because it tried to use a `null` pointer.

e. [T] F In Java, a class that is a subtype of an interface must directly implement or inherit every method mentioned in the interface.

f. T [F] Two Java classes can both have a method named `m` only when one is a subclass of the other.

g. T [F] In Java, if `s` and `t` are variables of type `String` such that `s == t`, then `s.equals(t)` is guaranteed to return **`true`**.

h. [T] F In Java, *simple inheritance* refers to the idea that a subclass extends its parent only by adding new fields or methods.

i. [T] F In Java, a *static method* is associated with the class containing it, not an instance of the class.

j. [T] F In Java, a field declared `final` is immutable (i.e. it cannot be assigned to) once it has been initialized.

2. Java Static Types and Dynamic Classes (18 points)

The following well-typed program refers to the Java interfaces and class declarations in the Appendix on the last page of this exam. We recommend that you *carefully* tear off that page so that you can refer to it while completing this question.

```
public class Exam {
  public static Pullable swap(Pullable p) {
    return new GroceryCart();
  }
  public static void main(String[] args) {
    Cart c = new Cart();             // (1)
    Pullable p = new Wagon();        // (2)
    Slidable s = new GroceryCart(); // (3)
    s = c;                           // (4)
    p = swap(p);                     // (5)
    p.pull();                        // (6)
    Object o = new Cart();           // (7)
                                     // (8)
  }
}
```

a. What is the static type of `s` on line `(4)`?

Answer: `Slidable`

b. What is the dynamic class of `s` after executing line `(4)`?

Answer: `Cart`

c. What is the static type of `p` on line `(5)`?

Answer: `Pullable`

d. What is printed by the invocation of `p.pull()` on line `(6)`?

Answer: `Squeak`

e. What types could replace `Object` as the type declared for variable `o` on line `(7)` without leading to a static type error?

Answer: `Cart`, `Slidable`, `Pullable`, `Pushable`

f. Which of the following lines of code would lead to a static (a.k.a. a type checking or compile-time) error if inserted at line `(8)`? Circle *all* erroneous statements.

i. `c.push();`
ii. `Pullable p2 = new Pullable();`
iii. `swap(s).pull();`
iv. `s = swap(s);`
v. `Pushable q = s;`
vi. `(new Wagon()).pull();`

Answer: `(ii)` and `(iv)` are ill-typed.

3. Java Array Programming (18 points)

Recall that in Java a two dimensional array might be *ragged*, which means that it is not "rectangular" in shape. More precisely, a ragged 2D array `a` has an index `i` such that `a[0].length` is not equal to `a[i].length`.

Write a function `pad`, that takes a potentially ragged 2D array of integers and returns a "padded" copy `p`, which is the smallest rectangular array such that if `a[i][j]` is defined (i.e. doesn't lead to an `ArrayIndexOutOfBoundsException`) then `p[i][j] = a[i][j]` and otherwise `p[i][j] = 0`.

Pictorially, if `a` is as shown below, then `pad(a)` will be the same as `a` but with `0`'s filling out the rectangle.:

```
     a                pad(a)
0 1 2 3 0          0 1 2 3 0
4 5                4 5 0 0 0
6 7 8              6 7 8 0 0
9                  9 0 0 0 0
```

You may assume that the input array `a` is not null and that it contains no null sub-arrays.

```
// assume that a is non-null and that it contains no null elements
public static int[][] pad(int[][] a) {
  int[][] result = new int[a.length][];
  int max = 0;
  for(int i=0; i<a.length; i++) {
    if (a[i].length > max) {
      max = a[i].length;
    }
  }
  for(int i=0; i<a.length; i++) {
    result[i] = new int[max];
    for(int j=0; j<a[i].length; j++) {
      result[i][j] = a[i][j];
    }
  }
  return result;
}
```

4. OCaml Abstract Stack Machine (12 points)

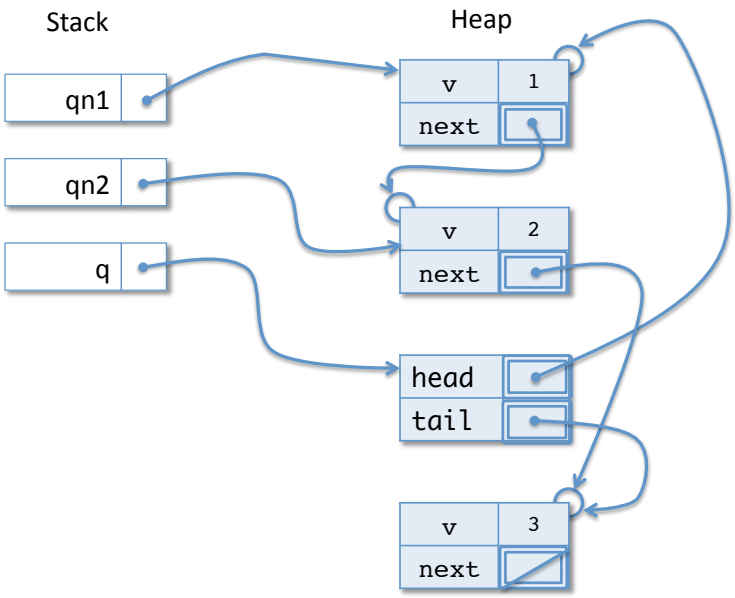
Appendix A gives you the code for the mutable queue implementation from class and homework 5. Appendix B gives you an example OCaml abstract stack machine diagram for the queue data types.

Following those conventions, complete the stack and heap diagram showing the state of the machine when it reaches the point marked *(∗ HERE ∗)*. We have provided some of the parts of the diagram for you; you need to complete the picture.

- Show only the *final* state of the machine.
- If you need to erase a line, mark it clearly with an X.

```
let qn1 : int qnode = { v = 1; next = None }
let qn2 : int qnode = { v = 2; next = Some qn1 }
(* NOTE: head and tail are SWAPPED compared to the example in the appendix. *)
let q : int queue = { head = Some qn1; tail = Some qn2; }

;; qn1.next <- Some qn2
;; enq 3 q
(* HERE *)
```



5. OCaml First-class Functions and Mutable State (10 points)

Consider the following OCaml program that mixes first-class functions and mutable state:

```
type 'a ref = { mutable contents : 'a }

let a : int ref = { contents = 0 }
let b : int ref = { contents = 5 }
let mk_f () : int -> int =
  let x = { contents = 1 } in
  fun (w : int) -> (
    x.contents <- x.contents + a.contents + w;
    x.contents
    )

let f : int -> int = mk_f () (* <-- create a closure named f *)

;; print_int (f 2)
;; a.contents <- 4
;; print_int (f 2)
```

a. Recall that first-class functions are stored in the heap using a *closure*. Circle the identifiers of stack bindings that **must** be included in the closure named `f` when it is created at the point indicated by `<--` in the code above.

`a` `b` `mk_f` `x` `w` `f`

Answer: `a` and `x` must be included

b. What is printed when the above program is run?

Answer: `3` and then `9`

6. Mutable Queues Implementation (22 points)

Implement a function, called `rev`, that reverses the nodes of a queue *in place*. In other words, this function should reverse the link structure of the queue *without* allocating any new `qnode` values in the heap. For example, if a queue `q` contains the values `1,2,3` (in that order) then, after an execution of `rev q`, the queue `q` should contain `3,2,1` (in that order). Note that your function should ensure that after calling `rev q`, `q` still refers to a valid queue (it should satisfy all of the queue invariants).

The type of `rev` is `'a queue -> unit` since it imperatively updates the queue structure. Its implementation requires a helper function called `loop` that traverses the link structure of the queue, updating each node's `next` field to point to its original predecessor in `q`. The `loop` input `p` names the node that was last updated; input `n` is the node (if any) that followed `p` *before* `p` was updated.

We have given you the basic structure of this code; your task is to complete it.

```
let rev (q : 'a queue) : unit =
  let rec loop (p : 'a qnode) (n : 'a qnode option) : unit =
    begin match n with
      | None -> ()
      | Some qn ->
        let rest = qn.next in
          qn.next <- Some p;
          loop qn rest
    end
  in
  begin match q.head with
    | None -> ()
    | Some p ->
      let rest = p.next in
      p.next <- None;
      loop p rest;
      q.head <- q.tail;
      q.tail <- Some p
  end
```

Appendix A: OCaml Linked Queue implementation

```
type 'a qnode = { v : 'a; mutable next : 'a qnode option; }

type 'a queue = {
 mutable head : 'a qnode option;
 mutable tail : 'a qnode option;
}

let create () : 'a queue =
  { head = None; tail = None }

let is_empty (q:'a queue) : bool =
  q.head = None

let enq (x:'a) (q:'a queue) : unit =
  let newnode_opt = Some { v = x; next = None} in
  begin match q.tail with
    | None -> q.head <- newnode_opt;
     q.tail <- newnode_opt
    | Some qn2 ->
     qn2.next <- newnode_opt;
     q.tail <- newnode_opt
  end

let deq (q:'a queue) : 'a =
  begin match q.head with
    | None -> failwith "error: empty queue"
    | Some qn ->
     begin match q.tail with
       | Some qn2 ->
        if qn == qn2 then
          (* deq from 1-element queue *)
          (q.head <- None;
           q.tail <- None;
           qn2.v)
        else
          (q.head <- qn.next;
           qn.v) (* Make sure to use parens around ; expressions. *)
       | None -> failwith "invariant violation"
     end
  end


let to_list (q : 'a queue) : 'a list =
  let rec loop (qn : 'a qnode option) (acc : 'a list) : 'a list =
   begin match qn with
     | None -> List.rev acc
     | Some qn1 -> loop qn1.next (qn1.v :: acc)
   end in
  loop q.head []
```

Appendix B: Example Abstract Stack Machine Diagram

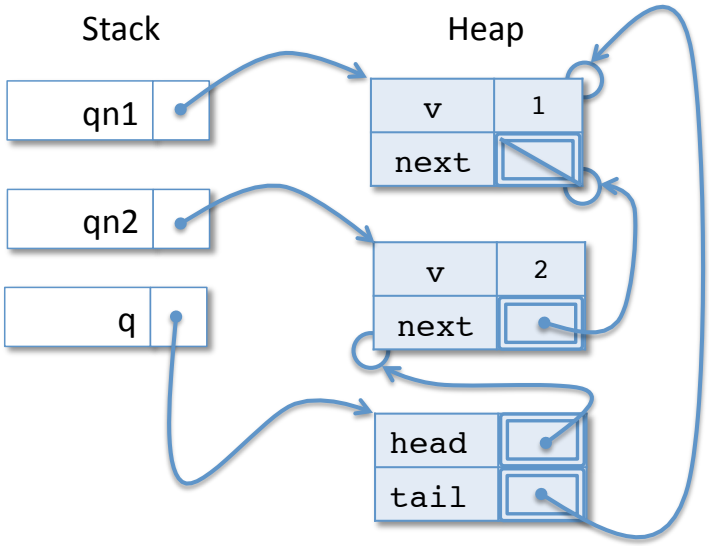
An example of the Stack and Heap components of the OCaml Abstract Stack Machine. Your diagram should use similar "graphical notation" for `Some v` and `None` values.

```
(* The types for mutable queues. *)
type 'a qnode = { v : 'a; mutable next : 'a qnode option; }

type 'a queue = {
  mutable head : 'a qnode option;
  mutable tail : 'a qnode option;
}

let qn1 : int qnode = {v = 1; next = None}
let qn2 : int qnode = {v = 2; next = Some qn1}
let q : int queue = {head = Some qn2; tail = Some qn1}
(* HERE *)
```

The OCaml program above yields the ASM Stack and Heap depicted below when the program execution reaches the point marked *(* HERE *)*.



Appendix C: Java Interfaces and Classes for Problem 2

```
interface Pushable {
  public void push();
}

interface Pullable {
  public void pull();
}

interface Slidable extends Pushable, Pullable { }

class Wagon implements Pullable {
  public void pull() { System.out.println("Wagon pulled"); }
}

class Cart implements Slidable {
  public void push() { System.out.println("Cart pushed"); }
  public void pull() { System.out.println("Cart pulled"); }
}

class GroceryCart extends Cart {
  public void pull() { System.out.println("Squeak!"); }
}
```