CIS 120 Midterm II November 6, 2015

SOLUTIONS

1. **Programming Concepts: Closures and Encapsulation** (12 points)

Multiple choice. For each question, mark *all* correct answers. There may be zero or more than one.

a. What value is computed for `ans` by the following OCaml program?

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

let x = {contents = 4}
let f : int -> int =
  let x = {contents = 2} in
  fun (y:int) -> y + x.contents

;; x.contents <- 1
let ans = f x.contents
```

☐ 2 ☒ 3 ☐ 4 ☐ 5

b. After running the following OCaml program, the closure named `f` will contain value bindings for which identifiers?

```
let a : int = 50
let f : int -> int =
  let b = 60 in
  fun (c:int) -> a + b + c
```

☒ `a` ☒ `b` ☐ `c` ☐ `f`

c. Consider the (simplified) `Pixel` class given by the Java code below:

```
class Pixel {
   public int[] rgb = {0, 0, 0};

   public Pixel(int r, int g, int b) {
      rgb[0] = r; rgb[1] = g; rgb[2] = b;
   }

   public int[] getComponents() {
      return rgb;
   }
}
```

To properly encapsulate the state of `Pixel` objects, the programmer should:

☒ Change the `rgb` field from **`public`** to **`private`**.

☐ Additionally declare the `rgb` field as **`final`**.

☐ Additionally declare the `rbg` field as **`static`**.

☒ Return a copy of the `rbg` array in `getComponents`.

2. Java Programming (20 points)

Consider the Java code shown in Appendix C, which consists of several interfaces and classes inspired by the GUI Paint project. (You may *carefully* tear off the pages of the appendix to make them easier to refer to.)

a. (10 points) For each code snippet shown below, check the box if the Java compiler would report any *static errors* for that code (assuming it appears as part of `main`). That is, mark which of the following cannot be compiled given just the definitions provided in the Appendix:

☐
```
Area a = new Space(10,10);
```

☒
```
Event e = new Event();
```

☒
```
Area a = new Space(10,10);
Widget w = new HPair(a, a);
```

☐
```
Label lbl = new Label(null);
int i = lbl.getWidth(null);
```

☒
```
Widget lbl = new Label("CIS 120");
lbl.setLabel("is fun!");
```

b. (6 points) Consider the following code snippet (which has no static errors):

```
Widget w1 = new Label("CIS 120");
Widget w2 = new HPair(w1, w1);
Area a = w2;
w2 = w1;
w1 = new Space(10, 10);
/* HERE */
```

The static type of `w1` at the point marked "here" is: Widget

The dynamic class of `w1` at the point marked "here" is: Space

The static type of `w2` at the point marked "here" is: Widget

The dynamic class of `w2` at the point marked "here" is: Label

The static type of `a` at the point marked "here" is: Area

The dynamic class of `a` at the point marked "here" is: HPair

c. (1 point) Consider the following code snippet:

```
Gctx g1 = new Gctx(10, 20);
Gctx g2 = (new Gctx(0, 0)).translate(10, 20);
System.out.println(g1 == g2);
```

What will be the effect of running this program?

☐ The program does not compile.

☐ The program throws `NullPointerException`

☐ The program prints **`true`**

☒ The program prints **`false`**

True or False (3 points) — each of the following statements is about the code in Appendix C as either.

d. T [✓ F] It would be legal to declare a new class, based on the given code, like this:

```
class FancySpace extends Space, Label implements Widget {
  // ... omitted ...
}
```

e. [✓ T] F The use of `Math.max` in the `HPair` implementation of `getHeight` is an example of a static method invocation.

f. [✓ T] F It is possible to add a second method to the `LabelController` interface *without* modifying the `Label` class and still have the label class compile without error.

3. Mutable Queues and the OCaml Abstract Stack Machine (26 points)

Recall the mutable queue implementation from class and homework and shown in Appendix A. In this problem, we will explore an implicit assumption that the queue implementation makes about the use of the data structure.

Suppose that we have the stack and heap of the OCaml ASM shown below.

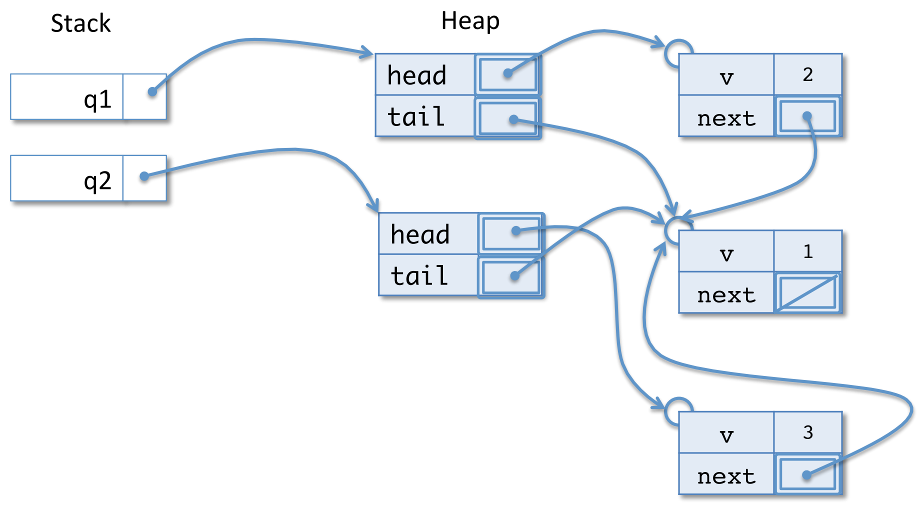


Figure 1: OCaml ASM state.

a. (8 points) Mark *all* of the following code snippets that, when run as the workspace of the ASM, construct the state shown above. *There may be more than one program that works!*

☐ (A)

```
let q1 = create ()
let q2 = create ()
;; enq 2 q1
;; enq 1 q1
;; enq 3 q2
;; enq 1 q2
```

☐ (B)

```
let q1 = create ()
let q2 = create ()
;; enq 2 q1
;; enq 1 q1
;; enq 3 q2
;; q2.head <- q1.head
```

☒ (C)

```
let q1 = create ()
let q2 = create ()
;; enq 2 q1
;; enq 1 q1
;; enq 3 q2
;; q2.tail <- q1.tail
;; begin match q2.head with
  | None -> failwith "impossible"
  | Some qn -> (qn.next <- q2.tail)
 end
```

☐ (D)

```
let q1 = create ()
let q2 = create ()
;; enq 2 q1
;; enq 1 q1
;; enq 3 q2
;; enq 1 q2
;; q1.tail <- q2.tail
```

Recall the definition of the (singly-linked) queue invariant used in class and in the homework:

> A value `q : 'a queue` satisfies the queue invariant if either
>
> 1. `q.head` and `q.tail` are both `None`,
>   or
> 2. `q.head` is `Some n1`, `q.tail` is `Some n2` and
>   - `n2` is reachable from `n1` by following `next` pointers
>   - `n2.next` is `None`

True or False (10 points)

b. [✓ T] F The `q1` value from Figure 1 satisfies the queue invariant shown above.

c. [✓ T] F The `q2` value from Figure 1 satisfies the queue invariant shown above.

d. [✓ T] F If we start the ASM in the state shown in Figure 1 and run the command

```
;; enq 5 q1
```

then `q1` still satisfies the queue invariant shown above.

e. T [✓ F] If we start the ASM in the state shown in Figure 1 and run the (same) command

```
;; enq 5 q1
```

then `q2` also still satisfies the queue invariant shown above.

f. T [✓ F] The ASM heap state shown in Figure 1 can be arrived at by using only the queue functions `create`, `enq`, and `deq` (as implemented in the Appendix).

Multiple Choice (3 points) — *choose one*

g. Which of the following should we add as an additional requirement to strengthen the queue invariant so that it ensures proper encapsulation? (And thus correctly ruling out the situation depicted in Figure 1 as invalid.)

- ☐ Every `qnode` reachable from `q.head` has no aliases.
- ☐ Every reference value reachable from `q` has at most two aliases.
- ☒ Every `qnode` reachable from `q` is reachable only by following a series of references starting from `q.head` or `q.tail`.
- ☐ Every mutable reference value reachable from `q.head` or `q.tail` is aliased only by references that are also reachable from `q.head` or `q.tail`.

Short Answer (5 points) — *answer in one concise sentence*

h. Explain one way to enforce this additional invariant.

Use a module and signature to encapsulate the abstract type (being sure not to expose the internal qnode values). *Grading Scheme: Encapsulation: 3 points Encapsulation + implementation: full points*

4. Mutable Queues Manipulation (20 points)

This problem uses the same OCaml mutable queues data structures and invariants as the previous problem.

Complete the code below so that the function reverses a queue *in place*. That is, if `q` is a queue then `reverse q` will modify the link structure of `q` to reverse the order of the elements (it should not call `enq` or `deq`). The implementation should preserve the queue invariants.

We have given you the structure of the loop helper and provided the main body of `reverse`.

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

Grading guide:

- Matching against `curr`: 2 pts
- Base case (set head): 5 pts
- Saved next: 3 pts
- Looping properly: 5 pts
- Setting pointers: 5 pts
- Errors using `option`: -1 point

5. Java Array Programming (22 points)

Implement a static method called `blockify` that takes a 2-dimensional rectangular array of integers called `src` and an integer `n` as inputs. The `blockify` method should return a new array that "expands" each entry of `src` into an `n`×`n` block of entries, each with the same source value. For example, given the 2×3 source array shown below:

```
int[][] src = { {0, 1, 2},
                {3, 4, 5} };
```

`blockify(src, 2)` would yield an 4×6 array `tgt` that could be written explicitly as:

```
{ {0, 0, 1, 1, 2, 2},
  {0, 0, 1, 1, 2, 2},
  {3, 3, 4, 4, 5, 5},
  {3, 3, 4, 4, 5, 5} }
```

- Assume the source array is rectangular—that is, each sub-array has the same length.
- Your solution should work even if one or both of the source array dimensions is `0`.
- Assume that input `n` ≥ 0 (There is no need to check that this is the case.)

One solution:

```
public static int[][] blockify(int[][] src, int n) {
   int tgtHeight = n * src.length;
   int tgtWidth = 0;
   if (src.length > 0) {
      tgtWidth = n * src[0].length;
   }
   int[][] tgt = new int[tgtHeight][tgtWidth];
   for (int i = 0; i < src.length; i++) {
      for (int j = 0; j < src[0].length; j++) {
         for(int x = 0; x < n; x++) {
            for(int y = 0; y < n; y++) {
               tgt[n*i + x][n*j + y] = src[i][j];
            }
         }
      }
   }
   return tgt;
}
```

Grading guide: (Iterate through the source)

- 1 point: creating new array
- 3 points: doing the 0 check
- 4 points: for getting the new array dimensions correct
- 5 points: for accessing every element of the output array
- 4 points: for correctly iterating through the source
- 5 points: for putting the right elements in the target

Another solution:

```
public static int[][] blockify2(int[][] src, int n) {
   int tgtHeight = n * src.length;
   int tgtWidth = 0;
   if (src.length > 0) {
      tgtWidth = n * src[0].length;
   }
   int[][] tgt = new int[tgtHeight][tgtWidth];
   for (int i=0; i<tgtHeight; i++) {
      for (int j=0; j<tgtWidth; j++) {
         tgt[i][j] = src[i/n][j/n];
      }
   }
   return tgt;
}
```

Grading guide: (Iterate through the target)

- 1 point: creating new array
- 3 points: for doing the 0 check
- 4 points: for getting the array dimensions correct
- 4 points: for accessing all of the cells of the target
- 10 points: for doing correct indexing logic

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

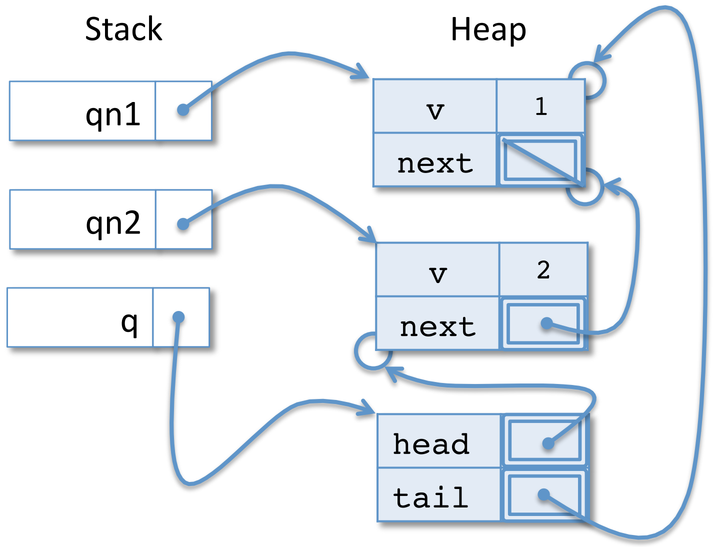
An example of the Stack and Heap components of the OCaml Abstract Stack Machine. Your diagram should use similar "graphical notation" for `Some v` and `None` values.

```
(* 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 GUI Code

```
class Gctx {
   private int x;
   private int y;

   public Gctx(int x, int y) { this.x = x; this.y = y; }

   public Gctx translate(int dx, int dy) {
      return new Gctx(this.x + dx, this.y + dy);
   }

   /* Graphics primitives (most omitted) */
   public void drawString(String s) { /* ... */ }
   public int getStringWidth (String s) { int sw = 0; /* ... */ return sw; }
   public int getStringHeight(String s) { int sh = 0; /* ... */ return sh; }
}

interface Area {
   public int getWidth (Gctx g);
   public int getHeight(Gctx g);
}

interface Event {
   public int getX(Gctx g);
   public int getY(Gctx g);
   public boolean isWithin(Gctx g, Area a);
}

interface Widget extends Area {
   public void repaint(Gctx g);
   public void handle (Gctx g, Event e);
}

interface LabelController {
   public void setLabel(String s);
}

class Space implements Widget {
   private int w;
   private int h;

   public Space(int w, int h) {
      this.w = w;
      this.h = h;
   }
   public void repaint(Gctx g) {}
   public void handle (Gctx g, Event e) {}
   public int getWidth(Gctx g) { return w; }
   public int getHeight(Gctx g) { return h; }
}
```

```
class Label implements Widget, LabelController {
   private String s;

   public Label(String s) { setLabel(s); }

   public void setLabel(String s) { this.s = s; }

   public int getWidth (Gctx g) { return g.getStringWidth(s); }
   public int getHeight(Gctx g) { return g.getStringHeight(s); }

   public void repaint(Gctx g) { g.drawString(s); }
   public void handle (Gctx g, Event e) {}
}

class HPair implements Widget {
   private Widget w1;
   private Widget w2;

   public HPair(Widget w1, Widget w2) { this.w1 = w1; this.w2 = w2; }

   public int getWidth (Gctx g) { return w1.getWidth(g) + w2.getWidth(g); }
   public int getHeight(Gctx g) { return Math.max(w1.getHeight(g), w2.getHeight(g)); }

   public void repaint(Gctx g) {
      w1.repaint(g);
      w2.repaint(g.translate(w1.getWidth(g), 0));
   }

   public void handle(Gctx g, Event e) { /* omitted */ }
}
```