CIS 120 Midterm II April 4, 2014

SOLUTIONS

1. Ocaml types and heap values

Consider the OCaml definitions and commands shown below.

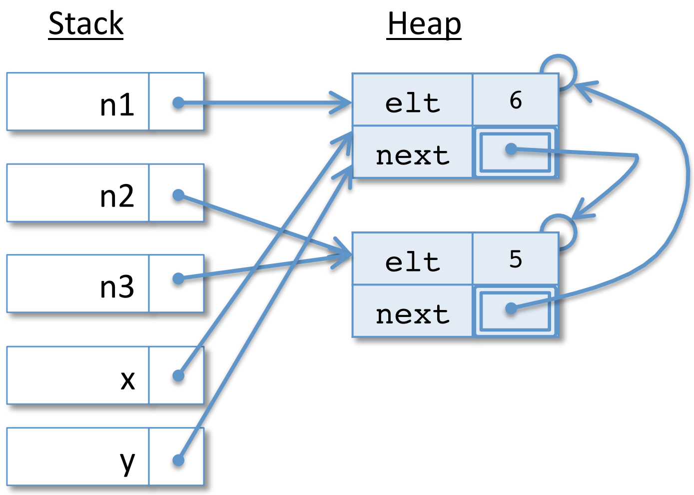
```
type 'a node = { mutable elt : 'a; mutable next : 'a node option; }

let n1 = { elt = 4; next = None }
let n2 = { elt = 5; next = Some n1 }
let n3 = n2
let x = begin match n3.next with
       | None -> failwith "None"
       | Some x -> x
       end
;; x.next <- Some n2
let y = begin match n2.next with
       | None -> failwith "None"
       | Some y -> y
       end
;; y.elt <- 6
```

a. (14 points) Write the *type* of each of the following expressions in the blank provided, or *ill-typed* if the expression does not type check.

`n1`	`int node`
`n1.elt`	`int`
`Some n1`	`int node option`
`n1.next`	`int node option`
`n1.next.next`	*ill-typed*
`x`	`int node`
`x.next <- Some n2`	`unit`

Grading Scheme: 2 points per blank. half credit for `node` *or* `'a node` *instead of* `int node`

b. (10 points) Next, complete the drawing of the stack and heap at the **end** of this computation. The variables on the stack and nodes in the heap have been given for you, you need to fill in the boxes with appropriate values (don't forget to add `Some` bubbles when necessary.) An example drawing of the OCaml ASM appears in the Appendix.



Grading Scheme: 1 point per box, plus 1 point for the two Some bubbles (both must be present). -1 for extra marks.

2. Program Design - Queues

Recall the definition of queues in OCaml as presented in class. For reference, the queue operations appear in the Appendix.

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

Use the design process to implement a function, called `join`, that takes two queues and modifies them so that all of the `qnodes` of the second queue are moved to the end the first queue while retaining their order.

For example, suppose queue `q1` contains the values `1,2` and queue `q2` contains the values `3,4`. Then after an execution of `join q1 q2`, the queue `q1` should contain `1,2,3,4`, and `q2` should be empty.

The join function may assume that `q1` and `q2` are valid queues, generated by the standard queue operations. If `q1` and `q2` are aliases to the *same* queue, then `join` should have no effect. As the purpose of this function is to mutate its arguments, it should always return `()`.

a. (0 points) The first step is to *understand the problem*. There is nothing to write here—your answers to the other parts will show how well you have accomplished this step.

b. (3 points) The next step is to *define the interface*. Write the type of the operation as you might see it in a .mli file. Use a generic type.

val join : ___________ 'a queue -> 'a queue -> unit _______________

Grading Scheme: -1 for incorrect argument, -1 for extra argument, -1 for return anything other than `unit`.

c. (8 points) The next step is to *write test cases*. For example, one possible test case is derived from the program description. (This test case uses the operations `from_list` and `to_list`, defined in the appendix.)

```
let test () =
  let q1 = from_list [1;2] in
  let q2 = from_list [3;4] in
  join q1 q2;
  to_list q1 = [1;2;3;4] && to_list q2 = []
;; run_test "given from the problem description" test
```

Below, describe **four** more *interesting* test cases for this method. (You don't need to actually define the testing code, just give a short description of the test cases that you would write).

- `q1` and `q2` are aliases
- both empty queues
- first empty, second nonempty
- first nonempty, second empty

Grading Scheme: 2 points per blank. Also accepted answers which tested singleton queues. These test cases are not interesting and received no credit:

- *duplicate of given test case with different values but same structure (the operation of join doesn't care about the values in the queue, so testing with two other two element queues is not likely to find new bugs.)*
- *duplicate of given test case but with different type of arguments (i.e. strings or booleans instead of ints)*
- *queues that don't alias but share qnodes (queues created by the library will never share nodes)*
- *test cases that are impossible to implement (i.e. queues with different types of elements.)*

d. (15 points) The final step is to *implement the function*. Your implementation should not create any new `qnode`s in the heap; it should only move them from one queue to another. Therefore, you **cannot use any of the queue operations defined in the appendix, such as `enq` or `from_list`**.

Hint: you do not need to use recursion to solve this problem.

```
let join (q1:'a queue) (q2 : 'a queue) : unit =
  if (q1 == q2 || q2.head == None) then ()
  else begin match q1.tail with
  | Some qn -> (qn.next <- q2.head;
             q1.tail <- q2.tail)
  | None -> (q1.head <- q2.head;
           q1.tail <- q2.tail)
  end;
  q2.head <- None;
  q2.tail <- None
```

Grading Scheme: Grading roughly follows correctness for the various cases, with points assigned as follows:

- *(2) queues aliased – no change to either queue*
- *(3) q2 empty – no change to either queue*
- *when q1 empty, q2 nonempty – make sure that q1's head and tail are updated (3), and that q2 is made empty (i.e. head and tail set to none) (2).*
- *when q1 nonempty, q2 nonempty – make sure that q1's tail node updated (2) and tail updated (2) and that q2 is made empty (1).*
- *shouldn't use a loop*
- *shouldn't use enq or* `from_list` *(at most half credit even if it works)*
- *shouldn't return either q1 or q2 (-1)*
- *no partial credit for nonsense like* `q1 <- q2` *or* `q2 <- Empty`

3. Java subtyping and interfaces (14 points)

Consider the following interface and class definitions. (Most of the class definitions have been omitted below, but you can assume that the classes contain the appropriate methods required by the interfaces.)

```
interface CanScoop {
  public void scoop();
}
interface CanStab {
  public void stab();
}
class Utensil { ... }
class Fork extends Utensil implements CanStab { ... }
class Spoon extends Utensil implements CanScoop { ... }
class Spork extends Fork implements CanStab, CanScoop { ... }
```

Which of these sequences of statements are well typed? Circle T if the definition type checks, and F if there is some problem with the code.

a. [T] F `Object x = new Fork();`

b. [T] F `CanStab x = new Spork();`

c. T [F] `CanScoop x = new CanScoop();`

d. T [F] `Spoon x = new Spork();`

e. [T] F `Utensil x = new Spoon(); x = new Fork();`

f. [T] F `Spork x = new Spork(); x.stab();`

g. T [F] `CanScoop x = new Spork(); x.stab();`

4. Arrays and Objects together (20 points)

Consider the following class declaration:

```
public class Counter {
   public int[] r;
   public Counter (int x) {
     r = new int[x];
   }
   public int inc (int i) {
       r[i] = r[i] + 1;
       return r[i];
   }
}
```

For each code snippet below, write the value of `ans` at the end of the computation or one of two exceptions if computation does not reach that point (write *NPE* for a `NullPointerExcepion`, and *AIOOBE* for an `ArrayIndexOutOfBoundsException`).

a.
```
Counter a;
Counter b = a;
int ans = b.inc(0);
```

NPE

b.
```
Counter a = new Counter (1);
Counter b = a;
a.inc(0);
int ans = b.inc(0);
```

2

c.
```
Counter a = new Counter (1);
int ans = a.inc(1);
```

AIOOBE

d.
```
Counter a = new Counter(2);
int[] s = a.r;
s[1]=2;
int ans = a.inc(1);
```

3

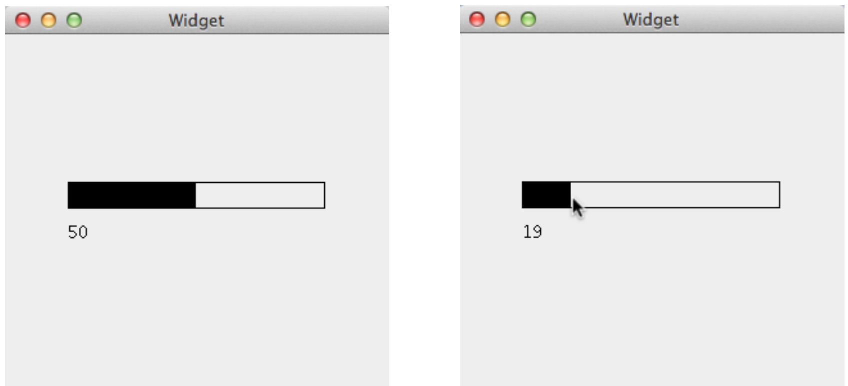
e.
```
Counter a = new Counter(2);
int[] s = { a.r[0], a.r[1] };
s[1]=2;
int ans = a.inc(1);
```

1

Grading Scheme: 4 points each

5. **Reactive Programming**

A `Slider` is a widget that allows users to set a percentage by moving a bar. For example, the window on the left displays a slider that starts with an initial value of 50 percent. (The slider is the halfway filled rectangle). Below the slider is a label that displays the current value of the slider. When the user clicks inside the slider rectangle, it changes the value of the slider to a new percentage, as shown in the window on the right. The label also updates to the new slider value.



This question asks you complete a `Slider` widget as part of the GUI library from Homework 8.

We have given you code to get started. Read *both* parts of this question before completing either one to make sure that you understand the design.

For reference, documentation for the GUI library for Homework 8 appears in the Appendix.

(See next page.)

a. (6 points) Below is the main function that generates the window shown in the example. Complete the body of the event listener for the slider so that as the slider changes value in response to mouseclicks, the label always displays the slider's current value.

Hint: read over the next part to see the interface of the Slider widget.

Hint: you can use the static method `Integer.toString` *to convert an int to a String.*

Hint: we have given you several lines for your answer. You may or may not need to use all of them.

```
class Main {
  public static void main(String[] args) {
     final Label lab = new Label("50");
     final Slider slider = new Slider(new Dimension(200,20));

     slider.setEventListener(new EventListener() {
       public void listen(Event e) {

              int v = slider.getValue();
        ______________________________________

            String s = Integer.toString(v);
        ______________________________________

                 lab.setLabel(s);
        ______________________________________

       }
      });

    Widget spacer = new Space(new Dimension(10,10));
    Widget group = new Vpair(new Vpair(slider,spacer), lab);
    Widget toplevel = new Centered(group);
    GUI.createAndShowGUI(toplevel);
  }

}
```

Grading Scheme: 2 points for accessing the current value of the slider, 2 points for converting it to a string and 2 points for setting the label. (1pt partial credit for using `value` *instead of* `getValue`*. 1pt partial credit for calling toString on the wrong value.)*

b. (10 points) Now fill in the blanks in the `repaint` and `handle` methods to complete the implementation of the `Slider` class. Note: the `handle` method is shown on the next page.

```
public class Slider implements Widget, NotifierController {

   /* Private state of the Slider */
   private final Dimension min;
   private EventListener listener = null;
   private int value = 50;


   /* Construct a slider with an initial percentage value of 50,
      no attached event listener, and a minimum size of d */
   public Slider(Dimension d) {
      min = d;
   }

   /* Return the current percentage value of the slider */
   public int getValue() {
      return value;
   }

   /* Set an object to serve as an EventListener for the slider. The listen
      method of this object will be invoked whenever the slider changes value */
   public void setEventListener(EventListener el) {
     listener = el;
   }


   /* Draw the Slider as a rectangle taking up the entire space specified
      by the graphics context. The filled portion of the rectangle should
      match the percentage value of the slider. */
   public void repaint(Gctx gc) {
     int w = gc.getWidth();
     int h = gc.getHeight();

     int filledWidth = ___________value * w / 100;___________

     Position origin = new Position(0,0);
     gc.drawRect(origin, w, h);
     gc.fillRect(origin, filledWidth, h);
   }
```

```
/* Handle an event that occurs in this widget. You may or may not
   need to use all of the lines below. */
public void handle(Gctx gc, Event e) {
  int w = gc.getWidth();
  Position p = gc.eventPosition(e.getPosition());

        if (!e.isMouseClick() || p == null) return;
  ______________________________________________________________

                    value = p.x * 100 / w;
  ______________________________________________________________

                    if (listener != null)
  ______________________________________________________________

                      listener.listen(e);
  ______________________________________________________________


  ______________________________________________________________

}

/* Access the stored initial size */
public Dimension minSize() {
  return min;
}



}
```

Grading Scheme:

- *3 repaint - rounding ok but not required. No partial credit for just* `value`*.*
- *7 handle - (3 update value (rounding ok but not required, but watch integer division), 2 null check for listener and 2 for invoke listener.listen) ok to call* `repaint(gc)` *here (Swing requires it, our library does not.) ok to not make sure the event is a mouseclick. ok to not nullcheck p.*

Appendix: GUI library documentation from HW 08

```
public interface EventListener {
   public void listen(Event e);
}
public interface NotifierController {
   public void setEventListener(EventListener el);
}

public interface Widget {
   public void repaint (Gctx gc);
   public void handle(Gctx gc, Event e);
   public Dimension minSize();
}

public class Label implements Widget {
   public Label(String s) { ... }

   public String getLabel() { ... }
   public void setLabel(String s) { ... }

   public void repaint(Gctx gc) { ... }
   public void handle(Gctx gc, Event e) { ... }
   public Dimension minSize() { ... }
}

public class Position {
  public final int x;
  public final int y;
  public Position(int x, int y) { ... }
}


public class Event {

  /** Create a mouse click event at a specified position. */
  public static Event makeMouseClick(Position pos) { ... }
  /** Create a key press event for a specified character. */
  public static Event makeKeyPress(Character c) { ... }

  /** Is this a mouse click event? */
  public boolean isMouseClick() { ... }
  /** Is this a key press event? */
  public boolean isKeyPress() { ... }

  /** return the absolute location of the mouse when the event occurred (if known)
   * or null otherwise. */
  public Position getPosition()

}
```

```java
public class Gctx {
  public int getWidth() { ... }
  public int getHeight() { ... }

  /**
   * Translate a gctx
   * @param dx Amount to translate the x axis
   * @param dy Amount to translate the y axis
   * @return Copy of the gctx object - translated
   */
  public Gctx translate (int dx, int dy) { ... }

  /**
   * Change both dimensions of gctx
   * @param w The new width
   * @param h The new height
   * @return Copy of the gctx object - with new width and height
   */
  public Gctx withSize(int w, int h) { ... }

  /**
   * Draws a rectangle - just the border
   * @param p Upper-left corner of the rectangle
   * @param w Width
   * @param h Height
   */
  public void drawRect(Position p, int w, int h) { ... }

  /**
   * Fills a rectangle
   * @param p Upper-left corner of the rectangle
   * @param w Width
   * @param h Height
   */
  public void fillRect(Position p, int w, int h) { ... }

  /**
   * Gets the position of an event (in widget coordinates)
   * @param p Position (in global coordinates)
   * @return Position (in local coordinates)
   */
  public Position eventPosition(Position p) { ... }

  /**
   * Tests if the event is within the width and the height of this component
   */
  public boolean eventWithin(Event e) { ... }
```

Appendix: 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 (q:'a queue) (x:'a) : 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
            (q.head <- None; q.tail <- None; qn2.v)
         else
            (q.head <- qn.next; qn.v)
       | None -> failwith "invariant violation"
       end
     end


let from_list (lst: 'a list) : 'a queue =
 let q = create () in
 let rec loop (lst : 'a list) : 'q queue =
   begin match lst with
    | [] -> q
    | hd :: tl -> (enq q hd ; loop tl)
 in
 loop lst

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: OCaml ASM Example

This is 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 of 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 *)*.