CIS 1200 Midterm 2   November 14, 2022

Benjamin C. Pierce and Swapneel Sheth, instructors

Name:

PennKey (penn login id):

I certify that I have complied with the University of Pennsylvania's Code of Academic Integrity in completing this examination.

Signature: ________________________________   Date: ______________________

- There are 60 total points. The exam length is 60 minutes.
- There are 10 pages in the exam and an Appendix for your reference.
- Please begin by writing your PennKey at the bottom of all the odd-numbered pages in the rest of the exam.
- Do not spend too much time on any one question. Be sure to recheck all of your answers.
- We will ignore anything you write on the Appendix.
- For coding problems: aim for accurate syntax, but we will not grade your code style for indentation, spacing, etc.
- If you need extra space for an answer, you may use the scratch page at the end of the exam; make sure to clearly indicate that you have done this in the normal answer space for the problem.
- Good luck!

1. **Deques and the OCaml ASM** (9 points)

   Recall the definitions of `deque` and `dqnode` from homework 4:

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

type 'a deque = {
  mutable head: 'a dqnode option;
  mutable tail: 'a dqnode option;
}
```

   In Appendix A you will find a collection of ASM drawings showing various possible configurations of the heap. In Appendix B you will find the invariant for deques, with each clause annotated with a letter between (a) and (f).

   For each of the following code sequences, please (1) choose which picture from Appendix A corresponds to the ASM's stack and heap after executing this code, and (2) check either the "satisfies invariant" box (if the heap at the end satisfies the deque invariant) or else one or more of the boxes labeled (a) through (f) to indicate which clauses of the invariant are *not* satisfied.

   (a)
```
let d =
  let node = { v = 1; prev = None; next = None } in
  { head = Some node; tail = Some node }
```

   Matches drawing

   ☐ A ☐ B ☐ C ☐ D ☐ E ☐ F

   ☐ Satisfies invariant *or*

   ☐ Clause (a) fails ("tail reachable from head via next")
   ☐ Clause (b) fails ("nothing next after tail")
   ☐ Clause (c) fails ("head reachable from tail via prev")
   ☐ Clause (d) fails ("nothing previous before head")
   ☐ Clause (e) fails ("next then prev")
   ☐ Clause (f) fails ("prev then next")

(b)
```
let d =
  { head = Some { v = 1; prev = None; next = None };
    tail = Some { v = 1; prev = None; next = None }; }
```

Matches drawing

☐ A ☐ B ☐ C ☐ D ☐ E ☐ F

☐ Satisfies invariant	*or*	☐ Clause (a) fails	(“tail reachable from head via next”)
		☐ Clause (b) fails	(“nothing next after tail”)
		☐ Clause (c) fails	(“head reachable from tail via prev”)
		☐ Clause (d) fails	(“nothing previous before head”)
		☐ Clause (e) fails	(“next then prev”)
		☐ Clause (f) fails	(“prev then next”)

(c)
```
let d =
  let node1 = { v = 1; prev = None; next = None } in
  let node2 = { v = 1; prev = Some node1; next = None } in
  { head = Some node2; tail = Some node1 }
```

Matches drawing

☐ A ☐ B ☐ C ☐ D ☐ E ☐ F

☐ Satisfies invariant	*or*	☐ Clause (a) fails	(“tail reachable from head via next”)
		☐ Clause (b) fails	(“nothing next after tail”)
		☐ Clause (c) fails	(“head reachable from tail via prev”)
		☐ Clause (d) fails	(“nothing previous before head”)
		☐ Clause (e) fails	(“next then prev”)
		☐ Clause (f) fails	(“prev then next”)

2. **Programming with Deques** (12 points)

   Now suppose we want to define a function `swapWithNext` that, given a `deque d` and a pointer `n` to a `dqnode` somewhere inside it, swaps `n` with the node immediately following it in the queue.

   Complete the code below for `swapWithNext`. Make sure that it correctly handles the case when `n` is the first or last node in `d`.

```
let swapWithNext (d : 'a deque) (n1 : 'a dqnode) : unit =
  begin match n1.next with
  | None -> ()
  | Some n2 ->
     begin match n1.prev with

     | None -> d.head <- Some n2

     | Some n0 -> __________________________________________________
     end;

     begin match n2.next with

     | None -> _____________________________________________________

     | Some n3 -> n3.prev <- Some n1
     end;

     n2.prev <- n1.prev;

     _______________________________________________________________

     _______________________________________________________________

     _______________________________________________________________
end
```

3. **OCaml Objects and GUI Concepts** (5 points)

   (a) A *closure* is simply the text of a function that has been copied into the heap.
   True ☐ False ☐

   (b) The only way to change the encapsulated state of any of the widgets defined in the `widget` module is by calling the `handle` method of that widget.
   True ☐ False ☐

   (c) According to the design principles of our GUI library, calling the `repaint` or `size` method of a widget should not change its state—only `handle` should do that.
   True ☐ False ☐

   (d) When writing a `repaint` method in the style of our GUI library, every call to a low-level drawing primitive from the `Graphics` module should use the provided `Gctx.gctx` to transform from the widget's local coordinates to screen coordinates.
   True ☐ False ☐

   (e) In our GUI library, a `notifier` is a first-class function stored in the hidden state of an `event_listener` widget. When an event occurs on the widget, the `event_listener` invokes all of the stored `notifier`s.
   True ☐ False ☐

4. **GUI Programming** (12 points)

   Consider the GUI library from HW05, part of which is also shown in Appendix C and D.

   Several widgets (`label`, `border`, etc.) draw on the screen in whatever pen color is found in the `Gctx.gctx` they are passed. This means that an outer widget can use `Gctx.with_color` to change how they look.

   Suppose we want to extend the widget library in the `Widget` module with a version of `with_color` that works on *widgets* instead of on graphics contexts—that is, it has type

   ```
   Widget.widget -> Gctx.color -> Widget.widget
   ```

   instead of:

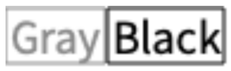
   ```
   Gctx.gctx -> Gctx.color -> Gctx.gctx
   ```

   Internally, it will call `Gctx.with_color` as needed; additionally, it will take care of passing `repaint`, `handle`, and `size` calls down to its inner widget.

   For example, writing this

   ```
   let wgray : widget =  with_color (border (label "Gray")) Gctx.gray
   let wblack : widget = border (label "Black")
   let top : widget = hpair wgray wblack
   ;; Eventloop.run top
   ```

   should display this:



   Complete the implementation of `Widget.with_color` on the next page.

```
let with_color (w: widget) (c: Gctx.color) : widget = {
```

```
}
```

5. **Java Objects and Equality** (6 points)

Consider the following Java interface and class definitions:

```
interface Incrementable {
  int incr ();
}

class Counter implements Incrementable {
  private int x = 0;
  public int incr () { x = x+1; return x; }
}

class Box implements Incrementable {
  public Incrementable i;
  public Box (Incrementable init) { i = init; }
  public int incr () { return i.incr(); }
  public Incrementable contents () { return i; }
}
```

Fill in the blanks in the following JUnit test cases so that all the tests pass.

```
@Test
public void test1 () {
  Counter c1 = new Counter();
  Counter c2 = new Counter();
  Box b1 = new Box(c1);

  assertEquals(b1.incr(), _______________);

  assertEquals(c2.incr(), _______________);

  assertEquals(c1.incr(), _______________);

}

@Test
public void testB() {
  Counter c = new Counter();
  Box x1 = new Box(c);
  Box x2 = new Box(x1);

  assertEquals(x1 == x2,           ________________);  // true or false

  assertEquals(x1.contents() == x2.contents(),

                                   ________________);  // true or false

  assertEquals(x1 == x2.contents(), ________________);  // true or false
}
```

6. **Java Array Programming** (16 points)

Write a function `find` that takes two **`int`**`[]` arrays, `original` and `pattern`, as parameters and returns **`true`** if the `original` array contains the elements of `pattern` *in a single contiguous block* and otherwise returns **`false`**.

E.g., the following calls to `find` should return **`true`**...

```
original          pattern
{1, 2, 3, 3, 4}   {1, 2, 3}
{1, 2, 3, 3, 4}   {2, 3}
{1, 2, 3, 3, 4}   {3, 4}
{1, 2, 3, 3, 4}   {3, 3, 4}
{1, 2, 3, 3, 4}   {}
```

... whereas these calls should return **`false`**:

```
original          pattern
{1, 2, 3, 3, 4}   {7}
{1, 2, 3, 3, 4}   {2, 2}
{1, 2, 3, 3, 4}   {2, 4}
{1, 2, 3, 3, 4}   {1, 2, 3, 3, 4, 5}
```

You may assume that neither input array is **`null`**. Do not use any helper functions.

```
public static boolean find (int[] original, int[] pattern) {










}
```

Scratch Space

Use this page for work that you do not want us to grade. If you run out of space elsewhere in the exam and you ***do*** *want to put something here that we should grade, make sure to put a clear note in the normal answer space for the problem in question.*