Midterm Exam

CS131: Programming Languages

Tuesday, May 3, 2011

Name:____

ID:____

Rules of the game:

- **Write your name and ID number above.**
- The exam is closed-book and closed-notes.
- Please write your answers directly on the exam. Do not turn in anything else.
- If you have any questions, please ask.
- The exam ends promptly at 11:50am.

A bit of advice:

- Read questions carefully. Understand a question before you start writing. *Note: Some multiple-choice questions ask for a single answer, while others ask for all appropriate answers.*
- For questions that are not multiple choice, write down thoughts and intermediate steps so you can get partial credit.
- The questions are not necessarily in order of difficulty, so skip around.
- Relax!

1. (5 points) Implement an OCaml function `count`, of type `’a -> ’a list -> int`, in an explicitly recursive manner. The function takes a value and a list and returns the count of the number of times that value appears in the list. For example, `count 3 [1;7;3;4;3;5]` returns the answer `2`.

```
let rec count e lst =
  match lst with
    [] -> 0
  | x::xs -> (if x=e then 1 else 0) + (count e xs)
```

2. (2 points each) Circle the best answer.

   (a) Parametric polymorphism in OCaml allows programmers to:

      i. define multiple functions of the same name
      ii. define one function with multiple names
      iii. define one function that can be called with arguments of different types
      iv. define one function that can be called with lists of different lengths

      iii

   (b) OCaml does not support overloading. What might an OCaml programmer do to work around this limitation when it arises?

      i. use polymorphism instead
      ii. give different functions different names
      iii. put functions of the same name in different modules
      iv. ii and iii above
      v. all of the above

      iv

   (c) Consider an invocation of a polymorphic function in OCaml:

      i. the function implementation to invoke is determined at run time and the invocation is typechecked at run time
      ii. the function implementation to invoke is determined at run time and the invocation is typechecked at compile time
      iii. the function implementation to invoke is determined at compile time and the invocation is typechecked at run time
      iv. the function implementation to invoke is determined at compile time and the invocation is typechecked at compile time

      iv

   (d) Suppose OCaml supported function overloading (for example, the way that `+` is overloaded in Standard ML), and consider an invocation of an overloaded function:

      i. the function implementation to invoke is determined at run time and the invocation is typechecked at run time
      ii. the function implementation to invoke is determined at run time and the invocation is typechecked at compile time
      iii. the function implementation to invoke is determined at compile time and the invocation is typechecked at run time
      iv. the function implementation to invoke is determined at compile time and the invocation is typechecked at compile time

      iv

3. (2 points each) **Circle all answers that are true for each question.**

   (a) Which functions always return a list of length less than or equal to that of the argument list?

      i. `List.map`
      ii. `List.filter`
      iii. `List.fold_right`
      iv. none of the above

      i and ii

   (b) Which functions always return a list of the same type as the argument list?

      i. `List.map`
      ii. `List.filter`
      iii. `List.fold_right`
      iv. none of the above

      ii

   (c) Which functions always return a list?

      i. `List.map`
      ii. `List.filter`
      iii. `List.fold_right`
      iv. none of the above

      i and ii

   (d) Which functions never return a list?

      i. `List.map`
      ii. `List.filter`
      iii. `List.fold_right`
      iv. none of the above

      iv

4. (5 points) Implement `count` from Problem 1 again, but this time using a single call to `List.fold_right` instead of explicit recursion.

```
let count e lst =
  List.fold_right (fun x curr -> (if x=e then 1 else 0) + curr) lst 0
```

5. (2 points each) Circle the best answer.

   (a) What are the advantages of static typechecking over dynamic typechecking?

      i. earlier error checking
      ii. each function body can be typechecked once, for all possible callers
      iii. static typechecking never signals a false error
      iv. the first two answers above
      v. the first three answers above
      vi. none of the above

   iv

   (b) What are the advantages of dynamic typechecking over static typechecking?

      i. earlier error checking
      ii. each function body can be typechecked once, for all possible callers
      iii. dynamic typechecking never signals a false error
      iv. the second and third answers above
      v. the first three answers above
      vi. none of the above

   iii

   (c) OCaml is considered statically typed because:

      i. the majority of its typechecking is done at compile time
      ii. it never allows type errors to occur at run time
      iii. the value of a variable never changes after initialization
      iv. a well-typed program can never raise an exception at run time

   i

   (d) C is considered weakly typed because:

      i. it does not support polymorphism
      ii. it does not prevent array bounds violations
      iii. it performs some typechecking at run time
      iv. it does not support type inference

   ii

6. (2 points each)

   Assume the following OCaml declarations have been entered in this order into the OCaml interpreter:

   ```
   let x = 3;;
   let f y = x + y;;
   let x = 4;;
   ```

   Give the value of each expression below, or say "error" if it would cause a static or dynamic error.

   (a) `f 5`
   8

   (b) `f x`
   7

   (c) `f 5`, assuming OCaml used dynamic scoping
   9

7. (2 points each)

   Assume the following OCaml declarations have been entered in this order into the OCaml interpreter:

   ```
   let g x y = x + y;;
   let f = g 3;;
   let y = 2;;
   ```

   Give the value of each expression below, or say "error" if it would cause a run-time error.

   (a) `f 5`
   8

   (b) `f x`
   error

   (c) `f 5`, assuming OCaml used dynamic scoping
   error

8. (2 points) Which is the best definition of static scoping?

   (a) Each variable usage can be bound to its associated declaration at compile time.

   (b) Each variable's lifetime can be determined at compile time.

   (c) Each variable usage can be bound to its associated value at compile time.

   (d) Each variable's value never changes after initialization

i

9. (5 points each) (Continues on the next page) Here's a simple signature for modules that implement a counter abstraction:

```
module type COUNTER = sig
  type counter
  val init : counter
  val increment : counter -> counter
  val value : counter -> int
  val reset : counter -> counter
end
```

The value `init` is a counter initialized to zero. The function `increment` increments the given counter by one, `value` returns the number of times the counter has been incremented (since last being reset), and `reset` resets the given counter.

(a) Complete the following implementation of the `COUNTER` signature, in which the `counter` type is just a synonym for `int`:

```
module Counter : COUNTER = struct
  type counter = int
  (* provide implementations of init, increment, value, reset *)

  let init = 0

  let increment c = c + 1

  let value c = c

  let reset c = 0









end
```

(b) Complete the following implementation of the `COUNTER` signature, in which the `counter` type is a datatype that explicitly represents each increment of the counter (e.g., `Inc(Inc Zero)` denotes the counter value 2):

```
module Counter : COUNTER = struct
  type counter = Zero | Inc of counter
  (* provide implementations of init, increment, value, reset *)


  let init = Zero

  let increment c = Inc c

  let rec value c =
    match c with
      Zero -> 0
      Inc c1 -> 1 + (value c1)

  let reset = Zero
```

```
end
```

10. (2 points) **Circle all answers that apply.** Hiding the definition of the type `counter` (see the previous problem) provides which of the following benefits for the `Counter` module?

    (a) It allows the definition of `counter` to be later changed without breaking clients of the module

    (b) It enables strong, static typechecking of the module

    (c) It ensures that clients can only modify a counter's value via the provided operations

    (d) It increases efficiency at run time

    i and iii