0/32 Questions Answered

Midterm Exam 2

Student Name

Search students by name or email...

Q1 Instructions and affirmation

0 Points

Before beginning this exam, make sure you have read and understood the exam instructions that were distributed ahead of the exam and are available at https://url.cs51.io/exam-instructions, and check the boxes below.

Q1.1

0 Points

- [ ] I have read and understood the exam instructions that were distributed ahead of the exam.

Save Answer

Q1.2

0 Points

- [ ] I affirm my awareness of the standards of the Harvard College Honor Code. I will not discuss the contents of this exam with anyone except for course staff until after the scheduled standard completion time of the exam on April 24, 2024.

Save Answer

Q2 True/false questions

9 Points

In this section, mark each statement as to whether it is true or false.

Q2.1

1 Point

Tail-recursive functions in OCaml need to be explicitly declared with the `rec` keyword.

- true
- false

Save Answer

Q2.2

1 Point

OCaml's type inference allows you to omit type annotations on functions, as the compiler can deduce the type based on how functions are used.

- true
- false

Save Answer

Q2.3
1 Point

OCaml's variant types cannot have constructors without any arguments.

- true
- false

Save Answer

Q2.4
1 Point

Lazy evaluation in OCaml ensures that values are computed at the time of their declaration, not at their first use.

- true
- false

Save Answer

Q2.5
1 Point

Looping expressions in OCaml, such as `while` loops or `for` loops, do not return a value and are used solely for their side effects.

- true
- false

Save Answer

Q2.6
1 Point

Once a reference is created in OCaml, it is not possible to change the data type of the value it refers to.

- true
- false

Save Answer

Q2.7
1 Point

OCaml's type system includes a specific type for streams, distinct from lists or arrays, to handle lazy evaluation more efficiently.

- true
- false

Save Answer

Q2.8
1 Point

According to the definition of substitution provided in the textbook, $(\mathtt{x}\ +\ 5)[\mathtt{x} \mapsto 37] = 42$.

- true
- false

Save Answer

Q2.9
1 Point

According to the definition of free variables provided in the textbook, $FV(\texttt{let x = y in let y = z in f x y z}) = \{\texttt{f}, \texttt{z}\}$.

true

false

Save Answer

Q3 In a world without while loops...

16 Points

As a reminder of how `while` loops work in OCaml, here's an iterative version of reversing a list using a `while` loop.

```
let rev (xs : 'a list) : 'a list =
  let xs = ref xs in
  let accum = ref [] in
  while !xs <> [] do
    accum := (List.hd !xs) :: !accum;
    xs := List.tl !xs
  done;
  !accum ;;
```

But suppose there were no `while` loop in OCaml. Could we write a function (we'll call it `while_`; note the underscore) that provides the same functionality? Such a function would have to take as arguments a condition and a body. The condition (`!xs <> []` in the example) would be of type `bool`. The body (`accum := (List.hd !xs) :: !accum; xs := List.tl !xs`) would be of type `unit`.

```
let while_ (condition : bool) (body : unit) : unit = ...
```

The `while_` would evaluate the `body` repeatedly so long as the `condition` was `true`.

The first issue to deal with is that, because OCaml is an eager language, when the application of `while_` to its two arguments is evaluated, the condition and body are first evaluated once and their values are used by `while_`. There's no option to postpone evaluation of the body or to reevaluate the body and condition.

Instead, we'd like to *delay* evaluation of the condition and body until the `while_` function needs to evaluate them. Luckily, you already know how to delay computations: *wrap them in a function*. We'll therefore stipulate that `condition` be of type `unit -> bool` and `body` be of type

`unit -> unit`, so as to delay their evaluation.

```
let while_ (condition : unit -> bool) (body : unit -> unit) : unit = ...
```

Q3.1
4 Points

Suppose we had such a `while_` function. Rewrite the definition of `rev` from above to use this `while_` function instead of the built-in `while` loop construct.

Save Answer

Q3.2
4 Points

Now provide a definition of the `while_ : (unit -> bool) -> (unit -> unit) -> unit` function.

Your definition should not use any loop constructs itself. (In particular, no using `while` to implement `while_`!)

Save Answer

Q3.3

4 Points

Next, provide another definition of `while_`, this time defining it directly in terms of the built-in `while` loop construct.

Save Answer

Q3.4

4 Points

You know of another way of delaying computation — the `lazy` construct. Provide a definition of `while_` using the `lazy` construct to delay the computations instead of `fun () -> ...` or explain why it isn't possible to do so.

Save Answer

Q4 Streams

6 Points

The questions in this section make use of the implementation of lazy streams based on OCaml's native `Lazy` module. For your reference, here is the `nativeLazyStreams.mli` file from problem set 7.

```
(*
                        CS 51 Problem Set 7
                     Refs, Streams, and Music
                       Native Lazy Streams
 *)

(*.....................................................................
An implementation of lazy streams using OCaml's native `Lazy` module,
along with with some useful functions.
 *)

type 'a stream_internal = Cons of 'a * 'a stream
 and 'a stream = 'a stream_internal Lazy.t ;;

(* head strm -- Returns the first element of `strm`. *)
val head : 'a stream -> 'a ;;
(* tail strm -- Returns a stream containing the remaining elements of
   `strm`. *)
val tail : 'a stream -> 'a stream ;;

(* first n strm -- Returns a list containing the first `n` elements
   of the `strm`. *)
val first : int -> 'a stream -> 'a list ;;

(* smap fn strm -- Returns a stream that applies the `fn` to each
   element of `strm`. *)
val smap : ('a -> 'b) -> 'a stream -> 'b stream ;;

(* smap2 fn strm1 strm2 -- Returns a stream that applies the `fn` to
   corresponding elements of `strm1` and `strm2`. *)
val smap2 : ('a -> 'b -> 'c) -> 'a stream -> 'b stream -> 'c stream ;;

(* sfilter condition strm -- Returns a stream of all elements of
   `strm` for which `condition` holds. *)
val sfilter : ('a -> bool) -> 'a stream -> 'a stream ;;
```

and here is its implementation from `nativeLazyStreams.ml`:

```
(*
```

```
                        CS 51 Problem Set 7
                      Refs, Streams, and Music
                        Native Lazy Streams
 *)

(*.....................................................................
An implementation of lazy streams using OCaml's native `Lazy` module,
along with with some useful functions.

See the corresponding .mli file for documentation.
 *)

type 'a stream_internal = Cons of 'a * 'a stream
 and 'a stream = 'a stream_internal Lazy.t ;;

let head (s : 'a stream) : 'a =
  let Cons (hd, _tl) = Lazy.force s in hd ;;

let tail (s : 'a stream) : 'a stream =
  let Cons (_hd, tl) = Lazy.force s in tl ;;

let rec first (n : int) (s : 'a stream) : 'a list =
  if n = 0 then []
  else head s :: first (n - 1) (tail s) ;;

let rec smap (f : 'a -> 'b)
             (s : 'a stream)
           : ('b stream) =
  lazy (Cons (f (head s), smap f (tail s))) ;;

let rec smap2 (f : 'a -> 'b -> 'c)
              (s1 : 'a stream)
              (s2 : 'b stream)
            : 'c stream =
  lazy (Cons (f (head s1) (head s2),
              smap2 f (tail s1) (tail s2))) ;;

let rec sfilter (pred : 'a -> bool) (s : 'a stream) : 'a stream =
    lazy (if pred (head s) then
            Cons((head s), sfilter pred (tail s))
          else Lazy.force (sfilter pred (tail s))) ;;
```

Q4.1
3 Points

Define a polymorphic function `stream_of` such that `stream_of x` returns an infinite stream all of whose elements are the value `x`. For instance,

```
# first 5 (stream_of 3) ;;
- : int list = [3; 3; 3; 3; 3]

# first 5 (stream_of 3.14159) ;;
- : float list = [3.14159; 3.14159; 3.14159; 3.14159; 3.14159]
```

Save Answer

Q4.2
3 Points

Define a function `every_nth : int -> int stream` such that `every_nth n` returns a stream of integers, starting with 0 and continuing with every n-th integer thereafter. For instance,

```
# first 5 (every_nth 4) ;;
- : int list = [0; 4; 8; 12; 16]
```

Save Answer

Q5 Lists versus arrays

11 Points

Arrays in OCaml are much like lists, one notable difference being that each element can be updated. For instance, here's an array of integers, which we update an element of:

```
# let arr = [|0; 5; 10|] ;;
val arr : int array = [|0; 5; 10|]
# arr.(1) <- 42 ;;
- : unit = ()
# arr ;;
- : int array = [|0; 42; 10|]
```

It should be possible to implement something like array updating by using lists of `ref`s. We'll explore the possibility in this problem.

Q5.1

2 Points

Define a value `listarr` of type `int ref list` that initially contains the integer values 0, 5, and 10 similar to the original definition of `arr` above.

Save Answer

Q5.2

4 Points

Define a function `update` that updates the value stored in an element of an `'a ref list` at a particular index with a new value. Its type should be `'a ref list -> int -> 'a -> unit`. It should raise a `Failure` exception if the list is too short, and `Invalid_argument` if the index is is negative. You may find the function `List.nth` helpful.

```
# update listarr 1 43 ;;
- : unit = ()
# listarr ;;
- : int ref list = [{contents = 0}; {contents = 43}; {contents = 10}]
```

Save Answer

Q5.3

1 Point

One of the nice properties of OCaml arrays is that they can be indexed and updated in constant time. Let's take a look at the time required for indexing into a list (including an `'a ref list`). Here's a version of the `List.nth` function:

```
let rec nth lst index =
  if index < 0 then raise (Invalid_argument "nth")
  else
    match lst with
    | [] -> raise (Failure "nth")
    | hd :: tl -> if index = 0 then hd else nth tl (pred index) ;;
```

What is the type of `nth`?

Save Answer

Q5.4
2 Points

What is the big-O worst-case time complexity of this implementation of the `nth` function in terms of the length n of the list being indexed into? In case more than one of these holds, select the tightest one.

- $O(1)$
- $O(n)$
- $O(n^2)$
- $O(2^n)$
- $O(n^3)$
- $O(\log n)$
- $O(n \log n)$

Save Answer

Q5.5
2 Points

What does that tell you about relative advantages or disadvantages of arrays versus lists of references? A sentence or two should suffice.

Save Answer

Q6 Object-oriented library holdings

20 Points

You're building an application to store the holdings of Lamont Library, which are items such as books and DVDs, and you decide to do so in OCaml (of course) using an object-oriented approach. Books and DVDs have various kinds of information in common, but here, we'll just consider a title and an internal ID, which we'll model as strings. In addition, books have an author (a string) and DVDs have a runtime (an integer).

To capture what books and DVDs have in comon, we start with a base class for library items in general. It should obey the following class type:

```
class type library_item_type =
  object
    method get_title : string
    method get_id : string
    method get_details : string
  end ;;
```

It specifies that there be methods that return the title (`get_title`) and ID (`get_id`) along with a string that provides full details about the item `get_details`.

Here's an implementation of a `library_item` class that satisfies that class type:

```
class library_item (title : string) (id : string) : library_item_type =
  object (this)
    val title : string = title
    val id : string = id
    method get_title : string = title
    method get_id : string = id
    method get_details : string = "Title: " ^ title ^ ", ID: " ^ id
  end;;
```

Now we can make some items and get their details:

```
# let alice = new library_item "Alice in Wonderland" "LC5432" ;;
val alice : library_item = <obj>
# alice#get_details ;;
- : string = "Title: Alice in Wonderland, ID: LC5432"
```

Q6.1

4 Points

Now it's your turn to provide some code. Define a class type `dvd_type` for DVDs, which have a title and ID (as all library items do), and also an integer runtime as described above. It should specify a `get_runtime` method to return the runtime.

Save Answer

Q6.2

4 Points

Define a `dvd` class that implements this class type. It should have the following behavior:

```
# let my_dvd = new dvd "Learning OCaml" "456DEF" 120 ;;
val my_dvd : dvd = <obj>
# my_dvd#get_details ;;
- : string = "DVD Title: Learning OCaml, ID: 456DEF, Runtime: 120 minutes"
```

(Notice that the `get_details` method adds to the details that the item is a DVD and that it has a runtime of 120 minutes.)

Save Answer

Q6.3

3 Points

We could also go on to add a `book_type` class type and its `book` class, but you can just assume that's already been done in a similar manner.

Now suppose we want to add a `print_details` method to these objects, which prints the details about the object to the standard output, as with the `print_endline` function. It would work as follows:

```
# my_dvd#print_details ;;
DVD Title: Learning OCaml, ID: 456DEF, Runtime: 120 minutes
- : unit = ()
```

What changes need to be made to the various class types and classes? Below, you should provide full implementations of `library_item_type`, `library_item`, `dvd_type` and `dvd`, not just the modifications. (You can just cut and paste the previous definitions and make whatever changes are necessary, but the implementations should be complete.)

The changes should be succinct and idiomatic. ***If no changes are necessary, select "No changes needed" and leave the text box empty.*** (If you select "No changes needed" and also fill in the text box, we will ignore the contents of the text box in grading the question.)

The modified `library_item_type` class type

☐ No changes needed

Save Answer

Q6.4

3 Points

The modified `library_item` class

- [ ] No changes needed

Save Answer

Q6.5

3 Points

The modified `dvd_type` class type

- [ ] No changes needed

Save Answer

Q6.6

3 Points

The modified `dvd` class

- [ ] No changes needed

Save Answer

Q7 Semantics

8 Points

In each of the questions below, we specify an environment and a store. You should construct an expression that ends in a subexpression `x` such that according to the lexical environment semantics of Figure 19.4 in the textbook, if the expression you constructed is evaluated in the context of the empty environment and store, ***the final subexpression*** `x` ***would end up being evaluated in the context of the specified environment and store***.

Here is an example:

Environment: $\{\mathtt{x} \mapsto \mathtt{42}\}$
Store: $\{\}$

ANSWER: `let x = 42 in x`

This answer reflects the fact that there is a step in the derivation for

$$\{\}, \{\} \vdash \mathtt{let\ x\ =\ 42\ in\ x} \Downarrow \cdots$$

of the form

$$\{\mathtt{x} \mapsto \mathtt{42}\}, \{\} \vdash \mathtt{x} \Downarrow \cdots$$

Your answer in the text box should provide ***just the expression***; no further explanation or discussion is required or considered.

For your convenience, we provide the rules from Figure 19.4 here:

$$E, S \vdash \overline{n} \Downarrow \overline{n}, S \qquad (R_{int})$$

$$E, S \vdash x \Downarrow E(x), S \qquad (R_{var})$$

$$E, S \vdash \texttt{fun}\ \ x\ \ \texttt{->}\ \ P \Downarrow [E \vdash \texttt{fun}\ \ x\ \ \texttt{->}\ \ P], S \qquad (R_{fun})$$

$$E, S \vdash P\ \ \texttt{+}\ \ Q \Downarrow \quad \left| \begin{array}{l} E, S \vdash P \Downarrow \overline{m}, S' \\ E, S' \vdash Q \Downarrow \overline{n}, S'' \end{array} \right. \quad \Downarrow \overline{m+n}, S'' \qquad (R_{+})$$

(and similarly for other binary operators)

$$E, S \vdash \texttt{let}\ \ x\ \ \texttt{=}\ \ D\ \ \texttt{in}\ \ B \Downarrow \quad \left| \begin{array}{l} E, S \vdash D \Downarrow v_D, S' \\ E\{x \mapsto v_D\}, S' \vdash B \Downarrow v_B, S'' \end{array} \right. \quad \Downarrow v_B, S'' \qquad (R_{let})$$

$$E_d, S \vdash P\ \ Q \Downarrow \quad \left| \begin{array}{l} E_d, S \vdash P \Downarrow [E_l \vdash \texttt{fun}\ \ x\ \ \texttt{->}\ \ B], S' \\ E_d, S' \vdash Q \Downarrow v_Q, S'' \\ E_l\{x \mapsto v_Q\}, S'' \vdash B \Downarrow v_B, S''' \end{array} \right. \quad \Downarrow v_B, S''' \qquad (R_{app})$$

$$E, S \vdash \texttt{ref}\ P \Downarrow$$
$$\Big|\ E, S \vdash P \Downarrow v_P, S' \qquad (R_{ref})$$
$$\Downarrow l, S'\{l \mapsto v_P\} \qquad \text{(where } l \text{ is a new location)}$$

$$E, S \vdash !\ P \Downarrow$$
$$\Big|\ E, S \vdash P \Downarrow l, S' \qquad (R_{deref})$$
$$\Downarrow S'(l), S'$$

$$E, S \vdash P\ \texttt{:=}\ Q \Downarrow$$
$$\Big|\ \begin{array}{l} E, S \vdash P \Downarrow l, S' \\ E, S' \vdash Q \Downarrow v_Q, S'' \end{array} \qquad (R_{assign})$$
$$\Downarrow (), S''\{l \mapsto v_Q\}$$

$$E, S \vdash P\ \texttt{;}\ Q \Downarrow$$
$$\Big|\ \begin{array}{l} E, S \vdash P \Downarrow (), S' \\ E, S' \vdash Q \Downarrow v_Q, S'' \end{array} \qquad (R_{seq})$$
$$\Downarrow v_Q, S''$$

Q7.1
2 Points

Environment: $\{\mathtt{x} \mapsto l1\}$
Store: $\{l1 \mapsto \mathtt{42}\}$

ANSWER:

Save Answer