First Midterm Exam

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 box below.

- [ ] I affirm my awareness of the standards of the Harvard College Honor Code.

Save Answer

Q2 Finger exercises

10 Points

Q2.1

2 Points

Provide a succinct definition of a function named `add_three` that returns the sum of three and its integer argument.

Enter your answer here

Save Answer

Q2.2
5 Points

Define a curried function named `power` that returns its first argument (an integer) raised to the power of its second argument (also an integer, which you can assume is nonnegative), returning an integer result.

Enter your answer here

Save Answer

Q2.3
3 Points

Now define an uncurried version of `power`.

Enter your answer here

Save Answer

Q3 Types of subexpressions in context
10 Points

Consider this snippet of code, which defines an algebraic data type

`'a combine` and a function `f`:

```
type 'a combine =
  | Combine of ('a -> 'a) * ('a combine)
  | Base of ('a -> 'a) * ('a option) ;;

let rec f x a =
  match x with
  | Base (f, None) -> f a [a]
  | Base (f, Some x) -> f [b] x
  | Combine (g, r) -> f [c] r [d] (g a) [e] ;;
```

For each of the subexpressions in the labeled boxes, specify the type of the subexpression in the context in which it appears. If the boxed element is not a single subexpression (and therefore has no type), answer "*no type*".

Q3.1

2 Points

What is the type of the expression `f a` in the box labeled *a*?

Enter your answer here

Save Answer

Q3.2

2 Points

What is the type of the expression `f` in the box labeled *b*?

Enter your answer here

Save Answer

Q3.3

2 Points

What is the type of the expression `f` in the box labeled *c*?

Enter your answer here

Save Answer

Q3.4

2 Points

What is the type of the expression `f r` in the box labeled *d*?

Enter your answer here

Save Answer

Q3.5

2 Points

What is the type of the expression `(g a)` in the box labeled *e*?

Enter your answer here

Save Answer

Q4 Short answer questions

8 Points

Each of the code snippets below has a blank in it, marked with a long underline. Your job is to find a **single OCaml pattern or expression** that can fill the blank such that the final expression evaluates to `42`.

If no such expression exists, select "no such expression". If an expression does exist, select "this expression" and provide the expression in the answer box.

(Note that the expressions may reference and make use of earlier definitions and functions in the exam.)

For example, if we provide the snippet

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

f _______________ ;;
```

you would check the "this expression" option and provide an answer such as

```
39
```

or

```
3 * 13
```

or any of a wide variety of other expressions.

Q4.1

2 Points

```
let setup = _____________ in
match setup with
| [] -> 21
| hd :: _tl -> hd ;;
```

- no such expression
- this expression

Enter your answer here

Save Answer

Q4.2

2 Points

```
let setup pair = _____________ in
let (x, y) = setup (14, 42) in
x + y + setup (14, 0) ;;
```

- no such expression
- this expression

Enter your answer here

Save Answer

Q4.3

2 Points

```
let setup = _____________ in
setup (setup 21) ;;
```

- no such expression
- this expression

Enter your answer here

Save Answer

Q4.4

2 Points

```
type various = {first : int; second : bool} ;;

let setup = _____________ ;;

let {first; second} = setup in
first + if not second then ~- first
        else first * first ;;
```

- no such expression
- this expression

Enter your answer here

Save Answer

Q5 Higher-order functional programming

10 Points

The following problems ask you to define simple functions. Each solution should nontrivially use **one** of the `List` module higher-order functions for mapping, folding, and filtering **exactly once** to implement the function (in addition to constructs from the base OCaml language). You may assume that the `List` module has already been opened.

Q5.1

5 Points

The `tower` function takes a list of integers `[a1; a2; a3; ...; an]` and returns their nested exponentiation

$$a_1^{a_2^{a_3^{\cdot^{\cdot^{\cdot^{a_n}}}}}}$$

For example,

```
# tower [2; 2] ;;
- : int = 4
# tower [2; 3] ;;
- : int = 8
# tower [2; 3; 2] ;;
- : int = 512
```

(Notice that the last example computes $2^{(3^2)} = 512$ and not $(2^3)^2 = 64$.)

Implement `tower` (using map/fold/filter as specified above). You

may assume that all of the exponents are nonnegative. You may assume availability of a correct implementation of the `power` function as described in Question 2.2 as well. Your implementation may handle the empty list however you see fit.

Hint: $a^1 = a$

Enter your answer here

Save Answer

Q5.2

5 Points

The `List.find` function is described as follows in the `List` module documentation:

> `val find : ('a -> bool) -> 'a list -> 'a`
> `find p l` returns the first element of the list `l` that satisfies the predicate `p`.
> Raises `Not_found` if there is no value that satisfies `p` in the list `l`.

It has the following behavior:

```
# find ((=) 4) [1; 3; 5; 7] ;;
Exception: Not_found.
# find ((=) 4) [1; 3; 4; 7] ;;
- : int = 4
# find ((<) 4) [1; 3; 5; 1] ;;
- : int = 5
```

Implement `find` (using only one map/fold/filter as specified above).

Enter your answer here

Save Answer

Q6 Circuits of resistors

16 Points

This problem concerns circuits of electrical resistors, but you don't need to know anything about the topic; we'll cover everything you need to know about resistors here.

In an electrical circuit, a *resistor* is a component that impedes the flow of electricity by a certain amount, called its *resistance*, and measured in *ohms*. You can see some in the picture above. We depict a resistor whose resistance is R ohms with a graphic like the one in Figure 1(*a*).

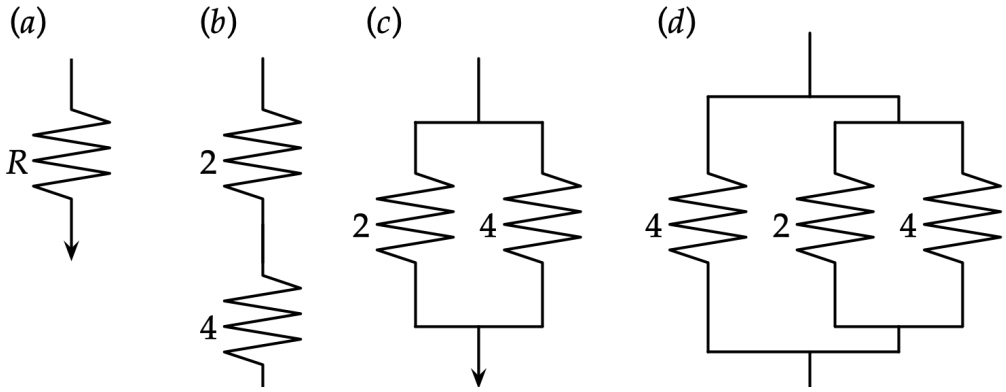

Figure 1: Circuits of resistors: (*a*) A resistor of R ohms. (*b*) A circuit of two resistors of 2 ohms and 4 ohms in series. (*c*) A circuit of two resistors of 2 ohms and 4 ohms in parallel. (*d*) a more complex circuit made of four resistors, three combined in parallel followed by one in series.

A resistor can serve all by itself as a simple circuit. But circuits can also be combined to form more complex circuits. Circuits are formed by combining simpler circuits (including individual resistors) using one of two modes of combination: series or parallel. For instance, Figure 1(*b*) depicts a circuit formed from two resistors (with resistances 2 and 4 ohms, respectively) combined in series, and Figure 1(*c*) depicts one formed from combining the same resistors in parallel. By combining circuits with other circuits in series and in parallel, we can generate more complex circuits, like the one in Figure 1(*d*). The three-way parallel circuit can be thought of as constructed by two nested parallel circuits.

The resistance R of a circuit formed by combining two circuits of resistance R_1 and R_2 in series is simply the sum of the resistances:

$$R = R_1 + R_2$$

So the resistance of the series circuit in Figure 1(b) is $2 + 4 = 6$ ohms.

The resistance R of a circuit formed by combining two circuits of resistance R_1 and R_2 in parallel is

$$R = \frac{1}{\frac{1}{R_1} + \frac{1}{R_2}}$$

So the resistance of the parallel circuit in Figure 1(c) is $\frac{1}{\frac{1}{2}+\frac{1}{4}} = \frac{4}{3}$ ohms.

Q6.1

5 Points

Define an algebraic data type `circuit` for circuits that can be composed of a single resistor (with its resistance given as a `float`) or composed of a pair of circuits connected either in series or in parallel.

Enter your answer here

Save Answer

Q6.2

2 Points

Define a value `circ_a` of type `circuit` that represents a single resistor of 3 ohms.

Enter your answer here

Save Answer

Q6.3

2 Points

Define a value `circ_c` of type `circuit` that represents the circuit in Figure 1(*c*).

Enter your answer here

Save Answer

Q6.4
2 Points

Define a value `circ_d` of type `circuit` that represents the circuit in Figure 1(*d*).

Enter your answer here

Save Answer

Q6.5
5 Points

Define a function `resistance : circuit -> float` that returns the total resistance of its `circuit` argument.

Enter your answer here

Save Answer

Q7 A functor for finite sequences
40 Points

We'll define a finite sequence as a series of elements, finite in number, all of a single type starting with some initial element and

with each succeeding element being generated from the previous one by a *next* function. For instance, here is a length-5 sequence of natural numbers (implemented in OCaml as an `int list`):

```
[0; 1; 2; 3; 4]
```

The initial element is `0` and the generating *next* function is the successor function.

A signature for modules that provide a finite sequence is as follows:

```
module type SEQUENCE =
  sig
    type t
    val sequence : int -> t list
  end
```

The signature requires that the module have a type `t` of the elements of the sequence and a function `sequence` that, given an integer length generates a sequence of that length as a list of elements of type `t`.

Here is a (partial) implementation of a module `Natnums` for sequences of natural numbers. (In addition to defining `t` and `sequence` as called for in the `SEQUENCE` module signature, it also defines a function `sequence_from` that is useful as an auxiliary function in defining `sequence`.)

```
module Natnums :                    a =
  struct
    type t =                        b
    let rec sequence_from from length =
      if length = 0 then []
      else from :: sequence_from (succ from) (length - 1)
    let sequence length =           c
  end ;;
```

With this module, we should be able to generate behavior like:

```
# Natnums.sequence 5 ;;
- : int list = [0; 1; 2; 3; 4]
# Natnums.sequence 10 ;;
- : int list = [0; 1; 2; 3; 4; 5; 6; 7; 8; 9]
```

Q7.1

2 Points

We've left out a few things in the implementation of the `Natnums` module, marked with the red labeled boxes.

What belongs in the box labeled *a*?

Enter your answer here

Save Answer

Q7.2

2 Points

What belongs in the box labeled *b*?

Enter your answer here

Save Answer

Q7.3

3 Points

What belongs in the box labeled *c*?

Enter your answer here

Save Answer

Q7.4

2 Points

Similarly, here is a (partial) implementation of a module `Diminishing` for a sequence where each `float` element is half of the preceding one:

```
module Diminishing : [d] =
  struct
    type t = [e]
    let rec sequence_from from length =
      if length = 0 then []
      else from :: sequence_from [f] (length - 1)
    let sequence length = [g]
  end ;;
```

For instance:

```
# Diminishing.sequence 3 ;;
- : Diminishing.t list = [1.; 0.5; 0.25]
# Diminishing.sequence 5 ;;
- : Diminishing.t list = [1.; 0.5; 0.25; 0.125; 0.0625]
```

Again, we've left out a few things in the implementation of the `Diminishing` module, marked with the red labeled boxes.

What belongs in the box labeled *d*?

Enter your answer here

Save Answer

Q7.5

2 Points

What belongs in the box labeled *e*?

Enter your answer here

Save Answer

Q7.6

2 Points

What belongs in the box labeled *f*?

Enter your answer here

Save Answer

Q7.7

2 Points

What belongs in the box labeled *g*?

Enter your answer here

Save Answer

Q7.8

6 Points

There's a lot of commonality between `Natnums` and `Diminishing`. They differ only in the *type* of their respective elements, the *initial* element, and the function for generating *next* elements. Consequently, we can package up those differences into a module obeying a signature called, let's say, `ELEMENT`. Provide a definition of this `ELEMENT` module type:

Enter your answer here

Now, we'll define a functor `Sequence` to generate `SEQUENCE` modules based on `ELEMENT` modules. Here is the skeleton of that functor definition:

```
module Sequence (Element : [          h])
                 : [          i] =
  struct
    type t = [          j]
    let rec sequence_from from length =
      if length = 0 then []
      else [          k]
    let sequence length = [          l]
  end ;;
```

Save Answer

Q7.9

3 Points

Again, we've left out some parts for you to fill in. What belongs in the box labeled *h*?

Enter your answer here

Save Answer

Q7.10

3 Points

What belongs in the box labeled *i*?

Enter your answer here

Save Answer

Q7.11

3 Points

What belongs in the box labeled *j*?

Enter your answer here

Save Answer

Q7.12
2 Points

What belongs in the box labeled *k*?

Enter your answer here

Save Answer

Q7.13
2 Points

What belongs in the box labeled *l*?

Enter your answer here

Save Answer

Q7.14
3 Points

Define a module `Natnums` by making use of the `Sequence` functor. It should have the same behavior as the definition for `Natnums` as defined in Question 7 above. Here's a start; you just need to fill in the box labeled *m*.

```
module Natnums =
  Sequence (struct
                  m
            end) ;;
```

end) ;;

Enter your answer here

Save Answer

Q7.15

3 Points

Define a module `Diminishing` by making use of the `Sequence` functor. It should have the same behavior as the definition for `Diminishing` as defined in Question 7.4 above.

Enter your answer here

Save Answer

Save All Answers

Submit & View Submission >