CS242 Midterm
Fall 2023

- Please read all instructions (including these) carefully.
- There are 4 questions on the exam, some with multiple parts. You have 80 minutes to work on the exam.
- The exam is open note. You may use laptops, phones and e-readers to read electronic notes, but not for computation or access to the internet for any reason.
- Please write your answers in the space provided on the exam, and clearly mark your solutions. Do not write on the back of exam pages or other pages.
- Solutions will be graded on correctness and clarity. Each problem has a relatively simple and straightforward solution. You may get as few as 0 points for a question if your solution is far more complicated than necessary. Partial solutions will be graded for partial credit.

NAME: _____

In accordance with both the letter and spirit of the Honor Code, I have neither given nor received assistance on this examination.

SIGNATURE: _____

Problem	Max points	Points
1	28	
2	12	
3	20	
4	20	
TOTAL	80	

1. **Lambda Calculus** (28 points)

Recall that we can represent a list (`List`) of natural numbers (`Nat`) using the algebraic datatypes

```
type List = Nil | Cons Nat List
type Nat = Succ Nat | 0
```

For all parts of this problem, you may use the following definitions in your solutions:

$$\begin{aligned}
\texttt{cons} &= \lambda h.\,\lambda t.\,\lambda x.\,\lambda f.\, f\ \ h\ \ (t\ \ x\ \ f)\\
\texttt{nil} &= \lambda x.\,\lambda f.\, x\\
\texttt{_0} &= \lambda f.\,\lambda x.\, x\\
\texttt{succ} &= \lambda n.\,\lambda f.\,\lambda x.\, f\ \ (n\ \ f\ \ x)\\
\texttt{tt} &= \lambda t.\,\lambda f.\, t\\
\texttt{ff} &= \lambda t.\,\lambda f.\, f\\
\texttt{pair} &= \lambda x.\,\lambda y.\,\lambda z.\, z\ \ x\ \ y\\
\texttt{fst} &= \lambda x.\,\lambda y.\, x \quad \text{(note: should be applied postfix, i.e., (}\texttt{pair}\ \ x\ \ y\text{) }\texttt{fst}\text{)}\\
\texttt{snd} &= \lambda x.\,\lambda y.\, y \quad \text{(note: should be applied postfix, i.e., (}\texttt{pair}\ \ x\ \ y\text{) }\texttt{snd}\text{)}
\end{aligned}$$

For each subproblem below, you may also use the functions asked about in earlier subproblems, even if you did not answer the earlier subproblem. For example, you may use `isempty` in your solutions to parts (b), (c) and (d) even if you do not answer part (a).

Implement the following functions in lambda calculus:

(a) `isempty`: takes a list l and returns `tt` if l is empty otherwise returns `ff`, e.g., `isempty([]) = tt` and `isempty([1, 2]) = ff`. (5 points)

$\texttt{isempty} = \lambda l.$ [ ]

(b) `len`: takes a list l and returns the number of elements in the list, e.g., `len([1, 2, 1]) = 3`. (7 points)

$\texttt{len} = \lambda l.$ [ ]

The function definitions from the previous page are repeated here for your convenience:

$$\begin{aligned}
\texttt{cons} &= \lambda h.\, \lambda t.\, \lambda x.\, \lambda f.\, f\ \ h\ \ (t\ \ x\ \ f) \\
\texttt{nil} &= \lambda x.\, \lambda f.\, x \\
\texttt{_0} &= \lambda f.\, \lambda x.\, x \\
\texttt{succ} &= \lambda n.\, \lambda f.\, \lambda x.\, f\ \ (n\ \ f\ \ x) \\
\texttt{tt} &= \lambda t.\, \lambda f.\, t \\
\texttt{ff} &= \lambda t.\, \lambda f.\, f \\
\texttt{pair} &= \lambda x.\, \lambda y.\, \lambda z.\, z\ \ x\ \ y \\
\texttt{fst} &= \lambda x.\, \lambda y.\, x \quad \text{(note: should be applied postfix, i.e., (}\texttt{pair}\ \ x\ \ y\text{) }\texttt{fst}\text{)} \\
\texttt{snd} &= \lambda x.\, \lambda y.\, y \quad \text{(note: should be applied postfix, i.e., (}\texttt{pair}\ \ x\ \ y\text{) }\texttt{snd}\text{)}
\end{aligned}$$

(c) `dupl`: takes a list l and returns a list with every element of l duplicated, e.g., `dupl([1, 2, 3]) = [1, 1, 2, 2, 3, 3]`. (7 points)

$\texttt{dupl} = \lambda l.$ [ ]

(d) `tail`: takes a list l and returns l excluding the first element of the list (unless $l = \texttt{nil}$, in which case it returns `nil`), e.g., `tail([1, 2, 3, 4]) = [2, 3, 4]` and `tail([]) = []`. (9 points)

$\texttt{tail} = \lambda l.$ [ ]

2. **Evaluation Order** (12 points)

Define the following functions:

$$\begin{aligned}
\texttt{cons} &= \lambda h.\, \lambda t.\, \lambda x.\, \lambda f.\, f\ \ h\ \ (t\ \ x\ \ f) \\
\texttt{nil} &= \lambda x.\, \lambda f.\, x \\
\texttt{_0} &= \lambda f.\, \lambda x.\, x \\
\texttt{succ} &= \lambda n.\, \lambda f.\, \lambda x.\, f\ \ (n\ \ f\ \ x) \\
\texttt{head} &= \lambda l.\, l\ \ \texttt{nil}\ \ (\lambda h.\, \lambda t.\, h) \\
\texttt{add} &= \lambda n.\, \lambda m.\, n\ \ \texttt{succ}\ \ m \\
\texttt{mul} &= \lambda n.\, \lambda m.\, n\ \ (\texttt{add}\ \ m)\ \ \texttt{_0} \\
\texttt{sum} &= \lambda l.\, l\ \ \texttt{_0}\ \ \texttt{add} \\
\texttt{prod} &= \lambda l.\, l\ \ (\texttt{succ}\ \ \texttt{_0})\ \ \texttt{mul} \\
\texttt{zeros} &= ((\lambda s.\, \texttt{cons}\ \ \texttt{_0}\ \ (s\ \ s))\ \ (\lambda s.\, \texttt{cons}\ \ \texttt{_0}\ \ (s\ \ s)))
\end{aligned}$$

Recall from lecture that given a reducible expression $(\lambda x.\, e)\ \ e'$, "call-by-name" first performs beta reduction (i.e., $(\lambda x.\, e)\ \ e' \to e[x := e']$) and then evaluates e', while "call-by-value" first evaluates e' and then performs beta reduction.

For each of the following expressions and evaluation orders, state either that the program does not terminate (write "nonterminating") or state what the program evaluates to:

(a) call-by-value, `head zeros` (2 points)

`head zeros` $\to$ [ ]

(b) call-by-name, `head zeros` (2 points)

`head zeros` $\to$ [ ]

(c) call-by-value, `sum zeros` (2 points)

`sum zeros` $\to$ [ ]

(d) call-by-name, `sum zeros` (2 points)

`sum zeros` $\to$ [ ]

(e) call-by-value, `prod zeros` (2 points)

`prod zeros` $\to$ [ ]

(f) call-by-name, `prod zeros` (2 points)

`prod zeros` $\to$ [ ]

3. **Algebraic Data Types** (20 points)
A *2-3 tree* with integer values has the following algebraic data type definition:

```
TTTree =
      Leaf |
      Two(TTTree,TTTree,Int) |
      Three(TTTree,TTTree,TTTree,Int,Int)
```

The full definition of 2-3 trees also has some balance conditions on the height of sibling subtrees, which we ignore for this problem. Give the implementation of 2-3 trees in the lambda calculus using the encoding of algebraic data types introduced in Lecture 4. Write the lambda term for each constructor below:

(a) (6 points)

`Leaf` =

(b) (7 points)

`Two` =

(c) (7 points)

`Three` =

4. **Type Polymorphism** (20 points)

We can write the S, K, and I combinators in lambda calculus and assign them the following simple types:

$$\begin{aligned} \texttt{I} &: a \to a \\ \texttt{K} &: b \to (c \to b) \\ \texttt{S} &: (d \to (e \to f)) \to (d \to e) \to d \to f \end{aligned}$$

For each of the following expressions give the type (up to alpha-renaming of type variables) that would be inferred for the simply typed lambda calculus (the algorithm given in Lecture 5). If no type can be inferred, write *untypable* — no justification is required.

(a) (5 points)

`S I` :

(b) (5 points)

`S K` :

(c) (5 points)

`S K I` :

(d) (5 points)

`S I I` :