CS242 Midterm
Fall 2022

- 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	30	
2	15	
3	15	
4	20	
TOTAL	80	

1. **Lambda Calculus** (30 points)
Recall the following seven definitions of booleans, integers, and lists from HW2:

$$\texttt{T} = \lambda x.\lambda y.\ x;$$
$$\texttt{F} = \lambda x.\lambda y.\ y;$$
$$\texttt{not} = \lambda b.\ b\ \texttt{F}\ \texttt{T};$$
$$\texttt{0} = \lambda f.\lambda x.\ x;$$
$$\texttt{inc} = \lambda n.\lambda f.\lambda x.\ f\ (n\ f\ x);$$
$$\texttt{cons} = \lambda h.\lambda t.\lambda f.\lambda x.\ f\ h\ (t\ f\ x);$$
$$\texttt{nil} = \lambda f.\lambda x.\ x;$$

We add three additional functions:

`add`: (`add x y`) evaluates to $x + y$.
`dec`: (`dec n`) evaluates to $n - 1$ if $n \geq 1$, and to 0 otherwise.
`is_zero`: (`is_zero n`) evaluates to true (`T`) if $n = 0$, and false (`F`) otherwise.

Write the following functions in the untyped lambda calculus. You may use the 10 functions above as part of your solutions. You may define helper functions if you wish (but you must also include the implementations of any helper functions).

(a) `equal`: (`equal x y`), where `x` and `y` are natural numbers, returns true (`T`) if $x = y$ and false (`F`) otherwise.

(b) `map`: (`map f l`) applies the function `f` to every element of list `l` and returns the resulting list. For example, `map inc (cons 0 (cons 1 (cons 2 nil)))` returns `cons 1 (cons 2 (cons 3 nil))`.

2. **Object Calculus** (15 points)
Implement a stack data structure in the object calculus. Your solution should be an object with the following three methods:

- `arg`: the argument to be pushed, expressed as a constant method. The `arg` method is initially $\varsigma(x)\, x$.
- `push`: pushes the value of `arg` onto the stack and resets `arg` to $\varsigma(x)\, x$.
- `pop`: returns an object with the stack popped and the former top of the stack in `arg`.

For example, if `s` is the empty stack, the sequence of operations

$$\mathtt{s}.(\mathtt{arg} \Leftarrow \varsigma(\mathtt{x})\, 3).\mathtt{push}$$

returns an object where the stack contains 3 and `arg` is $\varsigma(x)\, x$. The sequence of operations

$$\mathtt{s}.(\mathtt{arg} \Leftarrow \varsigma(\mathtt{x})\, 3).\mathtt{push}.\mathtt{pop}$$

returns an object where the stack is empty and `arg` is $\varsigma(x)\, 3$. The sequence of operations

$$\mathtt{s}.(\mathtt{arg} \Leftarrow \varsigma(\mathtt{x})\, 3).\mathtt{push}.\mathtt{pop}.\mathtt{arg}$$

returns 3. Finally, popping an empty stack is the identity: `s.pop` returns `s`.

Initially a stack object has the form:

$$[\mathtt{arg} = \varsigma(\mathtt{x})\, \mathtt{x}, \quad \mathtt{push} = ?, \quad \mathtt{pop} = ?]$$

Give the implementation of the `push` and `pop` methods. *Hint: You may find it helpful to use a backup method.*

3. **Types** (15 points)
Recall the grammar for the simply typed lambda calculus:

$$\begin{array}{rcl} e & ::= & x \mid \lambda x{:}\,\tau.\, e \mid e\ e \\ \tau & ::= & \alpha \mid \tau \to \tau \end{array}$$

where x stands for a family of program variables $x, y, z, \ldots$ and α stands for a family of type variables $\alpha, \beta, \gamma, \ldots$.

For each of the types τ below, give an expression in the simply typed lambda calculus that has type τ. For example, for the type $\alpha \to \alpha$, one possible solution is $\lambda x{:}\,\alpha.\, x$.

(a) $\alpha \to \beta \to \alpha$

(b) $(\alpha \to \alpha) \to \alpha \to \alpha$

(c) $(\alpha \to \beta) \to (\beta \to \gamma) \to (\alpha \to \gamma)$

4. **Combinator Calculi** (20 points)

Recall the following encoding of Boolean values in SKI calculus:

$$B\ x\ y \to^* x \quad \text{if } B \text{ is true,}$$
$$B\ x\ y \to^* y \quad \text{if } B \text{ is false.}$$

In HW1, we asked for *prefix* forms of Boolean functions AND and OR. In this question, we consider *infix* forms of binary Boolean operators in SKI calculus, without changing the rules for evaluation. For example, the AND operator can be implemented as the string "$I\ (K\ F)$":

$$T\ \underbrace{I\ (K\ F)}_{\text{“AND”}}\ T \to^* T \qquad\qquad F\ I\ (K\ F)\ T \to^* F,$$
$$T\ I\ (K\ F)\ F \to^* F \qquad\qquad F\ I\ (K\ F)\ F \to^* F.$$

Note: The infix AND defined in this way is *not* a combinator! In particular, there are **no implied parentheses** around $I\ (K\ F)$.

For each of the following functions, give an infix implementation. The entry for AND is already filled in. Recall that a XOR b is true if and only if a and b are not the same Boolean value. In addition to S, K, and I, you may use the T, F, and NOT combinators.

Function	Definition (in C)	Infix string
AND	`a && b`	$I\ (K\ F)$
OR	`a || b`	
XOR	`a ^ b`	
NOR	`!(a || b)`	