Name: __________________________________________

Student ID: ______________________________________

Email ____________________________________________

Status  ___ registered for the course
        ___ not registered: trying to improve a previous grade
        ___ not registered: just taking the exam for practice

Program  ___ undergrad
               ___ undergrad (MSE submatriculant)
               ___ CIS MSE
               ___ CIS MCIT
               ___ CIS PhD
               ___ other
<table>
<thead>
<tr>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>5</td>
</tr>
<tr>
<td>6</td>
</tr>
<tr>
<td>7</td>
</tr>
<tr>
<td>8</td>
</tr>
<tr>
<td>9</td>
</tr>
<tr>
<td>10</td>
</tr>
<tr>
<td>11</td>
</tr>
<tr>
<td>12</td>
</tr>
<tr>
<td>Total</td>
</tr>
</tbody>
</table>

Total: 1
Instructions

• This is a closed-book exam: you may not make use of any books or notes.

• You have 80 minutes to answer all of the questions. The entire exam is worth 80 points.

• Questions vary significantly in difficulty, and the point value of a given question is not always exactly proportional to its difficulty. Do not spend too much time on any one question.

• Partial credit will be given. All correct answers are short. The back side of each page may be used as a scratch pad.

• Good luck!
Operational semantics

The first three questions concern the following simple programming language:

\[
t ::= \begin{array}{ll}
  \text{true} & \text{constant true} \\
  \text{false} & \text{constant false} \\
  \text{if } t \text{ then } t \text{ else } t & \text{conditional} \\
  \text{pair } t \text{ } t & \text{pairing}
\end{array}
\]

\[
v ::= \begin{array}{ll}
  \text{true} & \text{true value} \\
  \text{false} & \text{false value} \\
  \text{pair } v \text{ } v & \text{pair value}
\end{array}
\]

and its large-step operational semantics.

\[
\begin{align*}
\text{true} & \Downarrow \text{true} & \text{(B-TRUE)} \\
\text{false} & \Downarrow \text{false} & \text{(B-FALSE)} \\
\text{if } t_1 \text{ then } t_2 \text{ else } t_3 & \Downarrow v & \text{(B-IFTRUE)} \\
\text{if } t_1 \text{ then } t_2 \text{ else } t_3 & \Downarrow v & \text{(B-IFFALSE)} \\
\text{pair } t_1 \text{ } t_2 & \Downarrow \text{pair } v_1 \text{ } v_2 & \text{(B-PAIR)} \\
\text{fst } t & \Downarrow v_1 & \text{(B-FST)} \\
\text{snd } t & \Downarrow v_2 & \text{(B-SND)}
\end{align*}
\]
1. (5 points) Show the derivation of the *large-step* evaluation of the following term.

\[ \text{fst (if true then pair true false else pair false true)} \]
2. (10 points) We might also want to define a small-step semantics for this language, such that

\[ t \downarrow v \text{ if and only if } t \rightarrow^* v \]

Recall that a small-step semantics is composed of both computation and congruence rules. The congruence rules for this language are as follows:

\[
\begin{align*}
  t_1 &\rightarrow t_1' \\
  \text{if } t_1 \text{ then } t_2 \text{ else } t_3 &\rightarrow \text{if } t_1' \text{ then } t_2 \text{ else } t_3 \quad \text{(E-If)}
\end{align*}
\]

\[
\begin{align*}
  t_1 &\rightarrow t_1' \\
  \text{pair } t_1 \ t_2 &\rightarrow \text{pair } t_1' \ t_2 \quad \text{(E-Pair1)}
\end{align*}
\]

\[
\begin{align*}
  t_2 &\rightarrow t_2' \\
  \text{pair } v \ t_2 &\rightarrow \text{pair } v \ t_2' \quad \text{(E-Pair2)}
\end{align*}
\]

\[
\begin{align*}
  t_1 &\rightarrow t_1' \\
  \text{fst } t_1 &\rightarrow \text{fst } t_1' \quad \text{(E-Fst)}
\end{align*}
\]

\[
\begin{align*}
  t_1 &\rightarrow t_1' \\
  \text{snd } t_1 &\rightarrow \text{snd } t_1' \quad \text{(E-Snd)}
\end{align*}
\]

Some of the computation rules are:

\[
\begin{align*}
  \text{if } \text{true} \text{ then } t_2 \text{ else } t_3 &\rightarrow t_2 \quad \text{(E-IfTrue)}
\end{align*}
\]

\[
\begin{align*}
  \text{if } \text{false} \text{ then } t_2 \text{ else } t_3 &\rightarrow t_3 \quad \text{(E-IfFalse)}
\end{align*}
\]

However, this list is not complete. What are the remaining computation rules for the small-step semantics?
3. (5 points) With a small step semantics, there is always the possibility that a term could fail to produce a value. Are there any “stuck” terms in this language? If so, give an example. If not, explain why not.
Functional programming

The following questions are about the untyped lambda calculus. For reference, the semantics of this language appears at the end of the exam.

Recall the Church encoding of lists and booleans in the untyped lambda calculus.

\[
\text{tru} = \lambda x. \lambda y. x \\
\text{fls} = \lambda x. \lambda y. y \\
\text{not} = \lambda b. \text{fls} \text{ tru} \\
\text{and} = \lambda b1. \lambda b2. b1 \text{ b2 fls} \\
\text{or} = \lambda b1. \lambda b2. b1 \text{ tru b2} \\
\text{nil} = \lambda c. \lambda n. n \\
\text{cons} = \lambda h. \lambda t. \lambda c. \lambda n. c \ h \ (t \ c \ n) \\
\text{head} = \lambda l. \ 1 \ (\lambda h. \lambda t. \ h) \text{ fls} \\
\text{tail} = \lambda l. \\
\quad \text{fst} (1 \ (\lambda x. \lambda p. \text{pair} \ (\text{snd} \ p) \ (\text{cons} \ x \ (\text{snd} \ p)))) \\
\quad (\text{pair} \ \text{nil} \ \text{nil})) \\
\text{isnil} = \lambda l. \ 1 \ (\lambda h. \lambda t. \text{fls}) \text{ tru}
\]

4. (5 points) Which of the following terms defines the function \text{all} that takes a list of boolean terms and determines \text{all} of the terms are true? For example,
\[
\text{all} \ (\text{cons} \ \text{tru} \ (\text{cons} \ \text{fls} \ \text{nil})) \text{ should be equivalent to fls and} \\
\text{all} \ \text{nil} \text{ should be equivalent to tru. Circle the correct answer.}
\]
(a) \text{all} = \lambda l. \ 1 \ \text{and} \ \text{tru} \\
(b) \text{all} = \lambda l. \ \text{all} \ (\text{hd} \ l) \ (\text{tail} \ l) \\
(c) \text{all} = \lambda l. \ 1 \ (\lambda a. \lambda b. \ a \ \text{tru} \ b) \ \text{fls} \\
(d) \text{all} = \lambda l. \ (\lambda a. \lambda b. \ a \ \text{and} \ b) \ l \ \text{fls}

5. (5 points) Which of the following terms defines the function \text{map} that take a term \text{l}, representing a list, and a function \text{f}, applies \text{f} to each element of \text{l}, and yields a list of the results (just like the \text{List.map} function in OCaml). For example:
\[
\text{map} \ \text{not} \ (\text{cons} \ \text{tru} \ (\text{cons} \ \text{fls} \ \text{nil})) \\
\text{should be equivalent to} \\
(\text{cons} \ \text{fls} \ (\text{cons} \ \text{tru} \ \text{nil})). \text{ Circle the correct answer.}
\]
(a) \text{map} = \lambda f. \lambda l. \ 1 \ (f \ \text{cons} \ \text{nil}) \\
(b) \text{map} = \lambda f. \lambda l. \ 1 \ (\lambda h. \lambda t. \ \text{cons} \ t \ (f \ h)) \ \text{nil} \\
(c) \text{map} = \lambda f. \lambda l. \lambda c. \lambda n. \ 1 \ (\lambda h. \lambda t. \ c \ (f \ h) \ t) \ n
6. (5 points) Which of the following OCaml terms implements the map function, using recursion? Circle the correct answer.

(a) `let rec map f l = match l with [] → f [] | (hd :: tl) → f hd :: map f tl`
(b) `let rec map f l = match l with [] → [] | (hd :: tl) → f hd :: map f tl`
(c) `let rec map f l = match l with [] → f [] | (hd :: tl) → hd :: map f tl`
(d) `let rec map f l = match l with [] → [] | (hd :: tl) → hd :: map f tl`
(e) `let rec map f l = match l with [] → f [] | (hd :: tl) → f hd :: f tl`
(f) None of the above
Proofs by induction

7. (4 points) What is the structural induction principle for the untyped lambda calculus?

8. (15 points) Complete the following proof of a property of the untyped lambda calculus, by induction on the structure of lambda terms.

**Theorem:** If $t$ is closed, and $t \rightarrow t'$, then $t'$ is closed.

You may use, without proving, the following lemma about substitution.

**Lemma:** If $\lambda x. t_1$ is closed, and $t_2$ is closed, then the substitution $[x \mapsto t_2]t_1$ is also closed.

We prove the theorem by induction on the structure of the lambda term $t$.

- **Suppose** $t$ is a variable $x$. This case is trivial because

- **Suppose** $t$ is a lambda term $\lambda x. \ t_1$. This case is also trivial because

- **Suppose** $t$ is an application $t_1 \ t_2$. Consider the possible ways that $t \rightarrow t'$.
  - Suppose the last rule used was E-App1 where $t_1 \rightarrow t'_1$. 

9
– Suppose the last rule used was E-App2, where \( t_1 \) is a value and \( t_2 \rightarrow t'_2 \).

– Suppose the last rule used was E-AppAbs, where \( t_1 \) is a lambda term \( \lambda x. t_{11} \), \( t_2 \) is a value, and \( t' \) is \([x \mapsto t_2]t_{11}\).
9. (9 points) What do the following lambda calculus terms step to, using the single-step evaluation relation $t \rightarrow t'$. Write NONE if the term does not step. For reference, the semantics of the untyped lambda calculus appears in the appendix of the exam.

(a) $(\lambda x.x)(\lambda x. x x)(\lambda x. x x)$

(b) $(\lambda x. (\lambda x.x) (\lambda x. x x))$

(c) $(\lambda x. (\lambda z. \lambda x. x z) x) (\lambda x. x x)$

10. (9 points) Now consider the leftmost/outermost evaluation relation from homework 4.

Using this reduction relation, what do the following terms step to? Again, write NONE if the term does not step.

(a) $(\lambda x.x)(\lambda x. x x)(\lambda x. x x)$

(b) $(\lambda x. (\lambda x.x) (\lambda x. x x))$

(c) $(\lambda x. (\lambda z. \lambda x. x z) x) (\lambda x. x x)$
Nameless representation of terms

11. (4 points) Suppose we have defined the naming context \( \Gamma = a, b, c, d \). Then the “nameless representation” of the term \( \lambda x. d \ x \ (\lambda y. x) \) is \( \lambda. 1 \ 0 \ (\lambda.1) \).

Write down the nameless representation for each of the following terms, in the given naming context.

(a) \( \lambda x. \ \lambda y. \ x \ c \ y \)

(b) \( \lambda x. \ b \ (\lambda y. \ d \ x \ x) \ d \)

12. (4 points) Write down (in de Bruijn notation) the normal form of the following de Bruijn term:

\( (\lambda. \ 1 \ (\lambda. \ 1)) \ (\lambda. \ 0) \)
For reference: Untyped lambda calculus

Syntax

\[ t ::= \]
\[ x \quad \lambda x.t \quad t.t \]

\[ v ::= \]
\[ \lambda x.t \]

Evaluation

\[ \frac{t_1 \rightarrow t'_1}{t_1 t_2 \rightarrow t'_1 t_2} \quad \text{(E-App1)} \]

\[ \frac{t_2 \rightarrow t'_2}{v_1 t_2 \rightarrow v_1 t'_2} \quad \text{(E-App2)} \]

\[ (\lambda x.t_{12}) v_2 \rightarrow [x \mapsto v_2]t_{12} \quad \text{(E-AppAbs)} \]