<table>
<thead>
<tr>
<th></th>
<th>Prob 1</th>
<th>Prob 2</th>
<th>Prob 3</th>
<th>Prob 4</th>
<th>Prob 5</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Score</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Max</td>
<td>15</td>
<td>25</td>
<td>55</td>
<td>25</td>
<td>20</td>
<td>140</td>
</tr>
</tbody>
</table>
1 SML Programming [15 pts]

Question 1. [5 pts] Give a tail-recursive implementation of \(\text{fib}\) for computing Fibonacci numbers.

(Type) \(\text{fib} : \text{int} \to \text{int}\)

(Description)

\(\text{fib} \ n\) returns \(\text{fib} \ (n - 1) + \text{fib} \ (n - 2)\) when \(n \geq 2\).

\(\text{fib} \ n\) returns 1 if \(n = 0\) or \(n = 1\).

(Invariant) \(n \geq 0\).

fun fib n =
let
    fun fib' ______________________________________
        ______________________________________
in
    fib' ____________
end

Question 2. [10 pts] Consider the signature MATRIX similar to the one that we have seen in Assignment 3.

signature MATRIX =
sig
    type t (* type of square matrices *)
    val identity : int -> t (* creates an identity matrix *)
    val dim : t -> int (* dimension *)
    val ++ : t * t -> t (* addition *)
    val ** : t * t -> t (* multiplication *)
    val == : t * t -> bool (* equality *)
end

• \(t\) denotes the type of square matrices.

• identity \(n\) returns an identity matrix of dimension \(n\).

• dim \(A\) returns the dimension of matrix \(A\).

• ++ (\(A_1, A_2\)) adds two matrices \(A_1\) and \(A_2\).

• ** (\(A_1, A_2\)) multiplies two matrices \(A_1\) and \(A_2\).

• == (\(A_1, A_2\)) returns true if two matrices \(A_1\) and \(A_2\) are equal and false otherwise.

The closure of a square matrix \(A\) is defined as \(I + A + A^2 + A^3 + \cdots\) where \(I(= A^0)\) is the identity matrix. Alternatively the closure of \(A\) can be defined as \(I + A + A^2 + \cdots + A^i\) where \(i\) is the first positive integer such that \(I + A + A^2 + \cdots + A^i\) is equal to \(I + A + A^2 + \cdots + A^i + A^{i+1}\).

Implement the functor ClosureFn where the member closure computes the closure of a given matrix. closure \(A\) should terminate if \(A\) has a closure.
functor ClosureFn (Mat : MATRIX) :
  sig
    val closure : Mat.t -> Mat.t
  end
= struct
    fun closure m =
      let
        val one = Mat.identity (Mat.dim m)
      in
        findClosure one
      end
  end
2 Inductive definitions [25 pts]

Question 1. [5 pts] Consider a system consisting of the following inference rules where \( n \ \text{nat} \) is a judgment meaning that \( n \) is a natural number:

\[
\frac{}{0 \ \text{nat} \ \text{Zero}} \quad \frac{n \ \text{nat}}{\text{S} \ n \ \text{nat} \ \text{Succ}}
\]

Give an inference rule that is derivable:

\[
\frac{}{n \ \text{nat}}
\]

Given an inference rule that is admissible, but not derivable:

\[
\frac{\text{S} \ n \ \text{nat}}{n \ \text{nat} \ \text{Succ}}
\]

Question 2. [20 pts] Consider the following system from the Course Notes where \( s \ \text{lparen} \) means that \( s \) is a string of matched parentheses.

\[
\frac{}{\epsilon \ \text{lparen} \ \text{Leps}} \quad \frac{s_1 \ \text{lparen}}{s_2 \ \text{lparen} \ \text{s lparen} \ \text{Lseq}}
\]

Prove the following theorem. The proof does not proceed by rule induction on the judgment \( (\cdots(s \ \text{lparen})_k \.) \).

- Fill in the blank. Use as much space as you need.
- As is conventional in the Course Notes, place conclusion in the left and justification in the right.

Theorem 2.1. For any string \( s \), if \( (\cdots(s \ \text{lparen})_k \.) \), then \( (\cdots(())_k \.) \).

Proof. By mathematical induction on \( k \).

Case \( k = 0 \):
Case $k = n$ where $n > 0$: 

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---
3  λ-Calculus [55 pts]

**Question 1. [5 pts]** Show the reduction sequence under the call-by-name strategy. Underline the redex at each step.

\[
((\lambda x_1. x_1) (\lambda x_2. x_2)) ((\lambda x_3. x_3) (\lambda z. z))
\]

\[
\mapsto
\]

\[
\mapsto
\]

\[
\mapsto
\]

**Question 2. [5 pts]** Complete the inductive definition of substitution. You may use \([x \mapsto y]e\) for the expression obtained by replacing all occurrences of \(x\) in \(e\) by \(y\) and all occurrences of \(y\) in \(e\) by \(x\).

\[
[e/x]x = \underline{\text{___________}}
\]

\[
[e/x]y = \underline{\text{___________}} \quad \text{if } x \neq y
\]

\[
[e/x](e_1 e_2) = \underline{\text{___________}}
\]

\[
[e'/x]\lambda x. e = \underline{\text{___________}}
\]

\[
[e'/x]\lambda y. e = \underline{\text{___________}} \quad \text{if } x \neq y, y \notin \text{FV}(e')
\]

\[
[e'/x]\lambda y. e = \lambda z. \underline{\text{___________}} \quad \text{if } x \neq y, y \notin \text{FV}(e')
\]

where \(z \neq y, z \notin \text{FV}(e), z \neq x, z \notin \text{FV}(e')\)
**Question 3. [5 pts]** A Church numeral encodes a natural number \( n \) as a \( \lambda \)-abstraction \( \hat{n} \) which takes a function \( f \) and returns \( f^n = f \circ f \circ \cdots \circ f \) (\( n \) times):

\[
\hat{n} = \lambda f. f^n = \lambda f. \lambda x. f f f \cdots f x
\]

Define an exponentiation function \( \exp \) such that \( \exp \hat{m} \hat{n} \) evaluates to \( \hat{m^n} \).

\[
\exp = \quad \text{_______________________________}
\]

**Question 4. [10 pts]** Define a function \( \text{halve} \) which halves a given natural number (encoded as a Church numeral):

- \( \text{halve} \hat{2} \times k \) returns \( \hat{k} \).
- \( \text{halve} \hat{2} \times k + 1 \) returns \( \hat{k} \).

You may use the following pre-defined constructs: \( \text{zero} \), \( \text{succ} \), and \( \text{pair}/\text{fst}/\text{snd} \).

- \( \text{zero} \) encodes the natural number zero.
  \[
  \text{zero} = \hat{0} = \lambda f. \lambda x. x
  \]

- \( \text{succ} \) finds the successor of a given natural number.
  \[
  \text{succ} = \lambda \hat{n}. \lambda f. \lambda x. \hat{n} f (f x)
  \]

- \( \text{pair} \) creates a pair of two expressions, and \( \text{fst} \) and \( \text{snd} \) are projection operators.
  \[
  \text{pair} = \lambda x. \lambda y. \lambda b. b x y
  \]
  \[
  \text{fst} = \lambda p. p (\lambda t. \lambda f. t)
  \]
  \[
  \text{snd} = \lambda p. p (\lambda t. \lambda f. f)
  \]

\[
\text{halve} = \quad \text{_______________________________}
\]
Question 5. [10 pts] This question assumes types \texttt{var} and \texttt{expr} that we have seen in Assignment 4:

\begin{verbatim}
    type var = string
    datatype exp =
        Var of var
      | Lam of var * exp
      | App of exp * exp
\end{verbatim}

Suppose that we have two functions \texttt{subst} and \texttt{isValue}:

- \texttt{subst : expr -> var -> expr -> expr}
- \texttt{isValue : expr -> bool}

Below is a function \texttt{step} of type \texttt{expr -> expr} such that \texttt{step e} returns \texttt{e'} if \texttt{e} reduces to \texttt{e'} and raises \texttt{Stuck} otherwise.

\begin{verbatim}
    fun step (App (Lam (x, e), e2)) =
        if isValue e2 then subst e2 x e
        else App (Lam (x, e), step e2)
    | step (App (e1, e2)) =
        if isValue e2 then App (step e1, e2)
        else App (e1, step e2)
    | step _ = raise Stuck
\end{verbatim}

We write \(e \mapsto e'\) if \(e\) reduces to \(e'\). Give exactly three reduction rules corresponding to the above definition of \texttt{step}.

\[
\begin{array}{ccc}
\rightarrow & \rightarrow & \rightarrow \\
\end{array}
\]
Question 6. [5 pts] Convert the following expression to a de Bruijn expression.

\[ \lambda x. \lambda y. (\lambda z. (\lambda u. x \ y \ z \ u) \ (x \ y \ z)) \ (\lambda w. w) \]

\[ \equiv_{dB} \]

Question 7. [5 pts] Following is the definition of de Bruijn expressions:

\[
\begin{align*}
M & ::= \ n \mid \lambda. \ M \mid M \ M \\
n & ::= 0 \mid 1 \mid 2 \mid \cdots
\end{align*}
\]

Complete the definition of \( \tau^n_i(N) \), as given in the Course Notes, for shifting by \( n \) (i.e., incrementing by \( n \)) all de Bruijn indexes in \( N \) corresponding to free variables, where a de Bruijn index \( m \) in \( N \) such that \( m < i \) does not count as a free variable.

\[
\begin{align*}
\tau^n_i(N_1 \ N_2) & = \\
\tau^n_i(\lambda. \ N) & = \\
\tau^n_i(m) & = \quad \text{if } m \geq i \\
\tau^n_i(m) & = \quad \text{if } m < i
\end{align*}
\]

Question 8. [10 pts] Define a mapping \( FV(M) \) that finds the set of de Bruijn indexes corresponding to free variables in \( M \). Here are a few examples:

- \( FV(\lambda. 0 \ 1 \ 2) = \{1, 2\} \)
- \( FV(\lambda. \lambda. 0 \ 1 \ 2) = \{2\} \)
- \( FV(\lambda. 0 \ 1 \ (\lambda. 0 \ 2)) = \{1, 2\} \)
- \( FV(\lambda. \lambda. \lambda. 0 \ 1 \ 2) = \{\} \)

Perhaps you will need an auxiliary function and use it in the definition of \( FV(M) \). If you introduce an auxiliary function, briefly state its meaning.

\[
\begin{align*}
FV(M) & = \\
& = \\
& = \\
& = \\
& = \\
& = \\
& =
\end{align*}
\]
4 Simply-typed $\lambda$-calculus [25 pts]

**Question 1. [10 pts]** We use the following reduction and typing judgments in formulating the semantics of the simply-typed $\lambda$-calculus:

\[
e \mapsto e' \iff e \text{ reduces to } e' \\
\Gamma \vdash e : A \iff \text{expression } e \text{ has type } A \text{ under typing context } \Gamma
\]

State the weakening property of typing judgments:

(Weakening). 

State two theorems, progress and type preservation, constituting type safety:

(Progress).

(Type preservation).

**Question 2. [5 pts]** Consider the extension of the simply-typed $\lambda$-calculus with sum types:

\[
\text{type} \quad A ::= \cdots | A + A \\
\text{expression} \quad e ::= \cdots | \text{inl}_A e | \text{inr}_A e | \text{case } e \text{ of } \text{inl } x . e | \text{inr } x . e
\]

Write the typing rule for \text{case } e \text{ of } \text{inl } x . e | \text{inr } x . e: 

\[
\begin{array}{c}
\hline
\hline
\end{array}
\]

\[+E\]
**Question 3. [5 pts]** Specify the lazy reduction strategy for the constructs for sum types. You should extend the definition of values and give reduction rules that maintain type safety.

\[
\text{value} \quad v \ ::= \ · \ · \ · \ | \quad \text{-----------------------------------}
\]

\[
\text{-----------------------------------}
\]

\[
\text{-----------------------------------}
\]

\[
\text{-----------------------------------}
\]

**Question 4. [5 pts]** Give an expression in the extended simply typed λ-calculus that denotes a recursive function \( f \) of type \( A \to B \) whose formal argument is \( x \) and whose body is \( e \).

\[
\text{-----------------------------------}
\]
5 Substitution [20 pts]

In this problem, we use the following fragment of the simply typed $\lambda$-calculus. We do not consider base types.

<table>
<thead>
<tr>
<th>type</th>
<th>$A ::= \ P \mid A \to A$</th>
</tr>
</thead>
<tbody>
<tr>
<td>base type</td>
<td>$P$</td>
</tr>
<tr>
<td>expression</td>
<td>$e ::= x \mid \lambda x : A \cdot e \mid e \ e$</td>
</tr>
<tr>
<td>value</td>
<td>$v ::= \lambda x : A \cdot e$</td>
</tr>
<tr>
<td>typing context</td>
<td>$\Gamma ::= \cdot \mid \Gamma, x : A$</td>
</tr>
</tbody>
</table>

\[
\begin{array}{c}
\frac{x : A \in \Gamma}{\Gamma \vdash x : A} \quad \text{Var} \\
\frac{\Gamma, x : A \vdash e : B}{\Gamma \vdash \lambda x : A \cdot e : A \to B} \quad \rightarrow I \\
\frac{\Gamma \vdash e : A \to B \quad \Gamma \vdash e' : A}{\Gamma \vdash e + e' : B} \quad \rightarrow E \\
\frac{e_1 \mapsto e_1' \quad e_2 \mapsto e_2'}{\lambda (\lambda x : A \cdot e) \mapsto (\lambda x : A \cdot e)'} \quad \text{App} \\
\end{array}
\]

Fill in the blank to complete the proof of the substitution lemma. We assume that a typing context is an unordered set and that variables in a typing context are all distinct.

**Lemma 5.1 (Substitution).** If $\Gamma \vdash e : A$ and $\Gamma, x : A \vdash e' : C$, then $\Gamma \vdash [e/x]e' : C$.

**Proof.** By rule induction on the judgment $\Gamma, x : A \vdash e' : C$. We consider only two cases shown below. In the first case, we assume (without loss of generality) that $y$ is a fresh variable such that $y \notin FV(e)$ and $y \neq x$. If $y \in FV(e)$ or $y = x$, we can always choose a different variable by applying an $\alpha$-conversion to $\lambda y : C_1 \cdot e''$.

Case \[
\frac{\Gamma, x : A, y : C_1 \vdash e'' : C_2}{\Gamma, x : A \vdash \lambda y : C_1, e'' : C_1 \to C_2} \quad \rightarrow I \quad \text{where } e' = \lambda y : C_1, e'' \text{ and } C = C_1 \to C_2:
\]

\[
\begin{array}{c}
\text{by induction hypothesis} \\
\text{by the rule } \rightarrow I \\
\text{from } y \notin FV(e) \text{ and } x \neq y
\end{array}
\]

Case \[
\frac{\Gamma, x : A \vdash e_1 : B \to C \quad \Gamma, x : A \vdash e_2 : B}{\Gamma, x : A \vdash e_1 e_2 : C} \quad \rightarrow E \quad \text{where } e' = e_1 e_2:
\]

\[
\begin{array}{c}
\text{by IH on } \quad \text{by IH on}
\end{array}
\]

\[
\begin{array}{c}
\text{from}
\end{array}
\]

\[
\Gamma \vdash [e/x] (e_1 e_2) : C
\]