

Name:

Hemos ID:

CSE-321 Programming Languages 2010
Midterm

	Prob 1	Prob 2	Prob 3	Prob 4	Total
Score					
Max	15	30	35	20	100

1 SML Programming [15 pts]

Question 1. [5 pts] Give a tail recursive implementation of `preorder` for preorder traversals of binary trees. Fill in the blank:

```
datatype 'a tree = Leaf of 'a | Node of 'a tree * 'a * 'a tree

(* preorder : 'a tree -> 'a list *)
fun preorder t =
let
  _____
  _____
  _____
  _____
in
  _____
end
```

Question 2. [7 pts] `DICT` is a signature for dictionaries:

```
signature DICT =
sig
  type key
  type 'a dict
  val empty : unit -> 'a dict
  val lookup : 'a dict -> key -> 'a option
  val delete : 'a dict -> key -> 'a dict
  val insert : 'a dict -> key * 'a -> 'a dict
end
```

- `key` denotes the type of keys in dictionaries.
- `'a dict` denotes the type of dictionaries for `'a` type values.
- `empty ()` returns an empty dictionary.
- `lookup d k` searches the key `k` in the dictionary `d`. If the key is found, it returns the associated item. Otherwise, it returns `NONE`.
- `delete d k` deletes the key `k` and its associated item in the dictionary `d` and returns the resultant dictionary `d'`. If the key does not exist in the dictionary `d`, it returns the given dictionary `d` without any modification.
- `insert d (k, v)` inserts the new key `k` and its associated item `v` in the dictionary `d`. If the key `k` already exists in the dictionary `d`, it just updates its associated item with the given item `v`.

Implement the functor DictFn which takes a KEY structure and generates a corresponding DICT structure that uses a 'functional representation' of dictionaries. Fill in the blank:

```
signature KEY =
sig
  type t

  (* eq k k' : true   k is equal to k'   *)
  (*           false otherwise           *)
  val eq : t * t -> bool
end

functor DictFn (Key : KEY) :> DICT where type key = Key.t
=
struct
  type key = Key.t
  type 'a dict = key -> 'a option

  fun empty () =
    _____

  fun lookup d k =
    _____

  fun delete d k =
    _____

  fun insert d (k, v) =
    _____
end
```

Question 3. [3 pts] Give an implementation of `IntDict` whose key type is `int`. Fill in the blank. You may use the functor `DictFn` that you write in Question 2.

```
structure IntKey :> KEY where type t = int
=
struct
```

```
end
```

```
structure IntDict = _____
```


3 λ -Calculus [35 pts]

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

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

\mapsto

\mapsto

\mapsto

\mapsto

Question 2. [3 pts] Complete the definition of $FV(e)$ that finds the set of free variables in e .

$$FV(x) = \underline{\hspace{10em}}$$

$$FV(\lambda x. e) = \underline{\hspace{10em}}$$

$$FV(e_1 \ e_2) = \underline{\hspace{10em}}$$

Question 3. [2 pts] Fill in the blank with the set of free variables of the given expression.

$$FV(\lambda x. x) = \underline{\hspace{10em}}$$

$$FV(x \ y) = \underline{\hspace{10em}}$$

$$FV(\lambda x. x \ y) = \underline{\hspace{10em}}$$

$$FV(\lambda x. \lambda y. x \ y) = \underline{\hspace{10em}}$$

$$FV((\lambda x. x \ y) (\lambda y. x \ y)) = \underline{\hspace{10em}}$$

Question 4. [5 pts] This question assumes types `var` and `exp` that we have seen in Assignment 4:

```

type var = string
datatype exp =
  Var of var
| Lam of var * exp
| App of exp * exp

```

Suppose that we have two functions `notFv` and `varSwap`:

- `notFv : var -> exp -> bool`
`notFv x e` returns true if x is a free variable of e and false otherwise.
- `varSwap : var * var -> exp -> exp`
`varSwap (x, y) e` returns $[x \leftrightarrow y]e$.

Below is a function `aEqual` of type `(exp * exp) -> bool` such that `aEqual (e1, e2)` returns true if e_1 and e_2 are α -equivalent and false otherwise.

```

fun aEqual (Var x, Var y) = x = y
| aEqual (App (e1, e2), App (e1', e2')) =
  aEqual (e1, e1') andalso aEqual (e2, e2')
| aEqual (Lam (x, e), Lam (y, e')) =
  if x = y then aEqual (e, e')
  else if notFv x e' then aEqual (e, varSwap (y, x) e')
  else false
| aEqual _ = false

```

We write $e \equiv_\alpha e'$ if e can be rewritten as e' by renaming bound variables in e and vice versa. Give exactly four inference rules corresponding to the above definition of `aEqual`. Use the notation $x \notin FV(e)$ for `notFv x e` and $[x \leftrightarrow y]e$ for `varSwap (x, y) e`.

\equiv_α

\equiv_α

\equiv_α

\equiv_α

Question 5. [8 pts] A Church numeral encodes a natural number n as a λ -abstraction \hat{n} which takes a function f and returns $f^n = f \circ f \cdots \circ f$ (n times):

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

In this question, you will define three functions: `sub` for the subtraction operation, `mod` for the modulo operation, and, as an extra credit problem, `div` for the division operation.

Your answers may use the following pre-defined constructs: `zero`, `one`, `succ`, `if/then/else`, `pair/fst/snd`, `pred`, `eq`, and `fix`.

- `zero` and `one` encode the natural numbers zero and one, respectively.

$$\begin{aligned} \text{zero} &= \hat{0} = \lambda f. \lambda x. x \\ \text{one} &= \hat{1} = \lambda f. \lambda x. f x \end{aligned}$$

- `succ` finds the successor of a given natural number.

$$\text{succ} = \lambda \hat{n}. \lambda f. \lambda x. \hat{n} f (f x)$$

- `if e then e1 else e2` is a conditional construct.

$$\text{if } e \text{ then } e_1 \text{ else } e_2 = e e_1 e_2$$

- `pair` creates a pair of two expressions, and `fst` and `snd` are projection operators.

$$\begin{aligned} \text{pair} &= \lambda x. \lambda y. \lambda b. b x y \\ \text{fst} &= \lambda p. p (\lambda t. \lambda f. t) \\ \text{snd} &= \lambda p. p (\lambda t. \lambda f. f) \end{aligned}$$

- `pred` computes the predecessor of a given natural number where the predecessor of 0 is 0.

$$\text{pred} = \lambda \hat{n}. \text{fst} (\hat{n} \text{ next } (\text{pair zero zero}))$$

- `eq` tests two natural numbers for equality.

$$\text{eq} = \lambda x. \lambda y. \text{and} (\text{isZero } (x \text{ pred } y)) (\text{isZero } (y \text{ pred } x))$$

- `fix` is the fixed point combinator.

$$\text{fix} = \lambda F. (\lambda f. F \lambda x. (f f x)) (\lambda f. F \lambda x. (f f x))$$

These constructs use the following auxiliary constructs, which you do not need:

$$\begin{aligned} \text{tt} &= \lambda t. \lambda f. t \\ \text{ff} &= \lambda t. \lambda f. f \\ \text{and} &= \lambda x. \lambda y. x y \text{ ff} \\ \text{isZero} &= \lambda x. x (\lambda y. \text{ff}) \text{ tt} \\ \text{next} &= \lambda p. \text{pair} (\text{snd } p) (\text{succ } (\text{snd } p)) \end{aligned}$$

Define a subtraction function `sub` such that `sub \hat{m} \hat{n}` evaluates to $\widehat{m - n}$ if $m > n$ and $\hat{0}$ otherwise.

`sub` = _____

Define a modulo function `mod` such that `mod \hat{m} \hat{n}` evaluates to \hat{r} if r is the remainder of division of m by n . `mod` never takes $\hat{0}$ as the second argument. Hence the result of evaluating `mod \hat{m} $\hat{0}$` is unspecified. You may use the subtraction function `sub` that you define above.

`mod` = _____

Extra credit question. [10 pts] Define a division function `div` such that `div \hat{m} \hat{n}` evaluates to \hat{q} if q is the quotient of m divided by n . `div` never takes $\hat{0}$ as the second argument. Hence the result of evaluating `div \hat{m} $\hat{0}$` is unspecified. In this question, you are not allowed to use the fixed point combinator (and its definition), but you may use the subtraction function `sub` that you define above.

`div` = _____

Question 6. [3 pts] Give an expression whose reduction does not terminate.

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

$$\begin{array}{ll} \text{de Bruijn expression} & M ::= n \mid \lambda. M \mid M M \\ \text{de Bruijn index} & n ::= 0 \mid 1 \mid 2 \mid \dots \end{array}$$

Suppose that you are given the definition of $\tau_i^n(N)$ 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.

Complete the definition of $\sigma_n(M, N)$ for substituting N for every occurrence of n in M where N may include free variables.

$$\sigma_n(M_1 M_2, N) = \underline{\hspace{15em}}$$

$$\sigma_n(\lambda. M, N) = \underline{\hspace{15em}}$$

$$\sigma_n(m, N) = \underline{\hspace{10em}} \qquad \text{if } m < n$$

$$\sigma_n(n, N) = \underline{\hspace{15em}}$$

$$\sigma_n(m, N) = \underline{\hspace{10em}} \qquad \text{if } m > n$$

Question 8. [4 pts] Show the reduction of the given expression where the redex is underlined.

$$\lambda. \lambda. \underline{(\lambda. (\lambda. 3 \ 2 \ 1 \ 0) (2 \ 1 \ 0))} (\lambda. 0) \mapsto$$

$$\underline{(\lambda. (\lambda. 1) 0)} (\lambda. 2 \ 1 \ 0) \mapsto$$

4 Simply-typed λ -calculus [20 pts]

Question 1. [2 pts] Consider the following simply-typed λ -calculus:

type $A ::= \text{bool} \mid A \rightarrow A$
expression $e ::= x \mid \lambda x:A. e \mid e e \mid \text{true} \mid \text{false} \mid \text{if } e \text{ then } e \text{ else } e$

Write the typing rules for x , $\lambda x:A. e$, and $e e$:

$$\Gamma \vdash x :$$

$$\Gamma \vdash \lambda x:A. e :$$

$$\Gamma \vdash e e' :$$

Question 2. [3 pts] Consider the extension of the simply-typed λ -calculus with product types:

type $A ::= \dots \mid A \times A$
expression $e ::= \dots \mid (e, e) \mid \text{fst } e \mid \text{snd } e$

Write the reduction rules for these constructs under *lazy* reduction strategy:

$$\mapsto$$

$$\mapsto$$

$$\mapsto$$

$$\mapsto$$

Question 3. [5 pts] Consider the extension of the simply-typed λ -calculus with sum types:

type	$A ::= \dots \mid A + A$
expression	$e ::= \dots \mid \text{inl}_A e \mid \text{inr}_A e \mid \text{case } e \text{ of inl } x.e \mid \text{inr } x.e$

Write the typing rules:

$$\Gamma \vdash \text{inl}_A e :$$

$$\Gamma \vdash \text{inr}_A e :$$

$$\Gamma \vdash \text{case } e \text{ of inl } x_1.e_1 \mid \text{inr } x_2.e_2 :$$

Question 4. [5 pts] Consider the extension of the simply-typed λ -calculus with fixed point constructs

expression $e ::= \dots \mid \text{fix } x:A.e$

Write the typing rule for $\text{fix } x:A.e$ and its reduction rule.

$$\Gamma \vdash \text{fix } x:A.e :$$

$$\mapsto$$

