Name:
SSN :

CSE 428

Fall 1999

Midterm #2

17 November 1999

The exam consists of 5 problems on 7 pages, totaling 100 points. Read each question carefully and use your time judiciously.

Write your name/number on every page.

1. Give the *most general* types for each of the following function declarations. (20 pts)

   (a) `fun flip(f,x,y) = f(y,x);`

   (b) `fun flop(f,g,z) = f(g(z),z);`

   (c)
   ```
   fun reduce(f,a,nil) = a
      |reduce(f,a, (x::xs)) = f(x,reduce(f,a,xs));
   ```

   (d)
   ```
   fun altnfold(f,g,0,x) = x + 0.0
      |altnfold(f,g,n,x) = f(g(altnfold(f,g,n-1,x)));
   ```

2. Consider the following function definitions: (20 pts)

```
fun map(f,nil) = nil
   |map(f,x::xs) = f(x) :: map(f,xs);

fun filter(p,nil) = nil
   |filter(p,x::xs) = if p(x) then x::filter(p,xs) else filter(p,xs);

fun nfold (f,0) x = x
   |nfold (f,n) x = f(nfold(f,n-1) x);

fun even n = (n mod 2 = 0);
fun inc n = n+1;
```

Give the value produced by each of the expressions below.

(5 pts)

(a) `map(inc, [2,4,6]);`

(5 pts)

(b) `nfold(inc,5) 4;`

(5 pts)

(c) `filter(even, map(inc, [1,2,3,4]));`

(5 pts)

(d) `map(nfold(inc,2), [1,2,3,4]);`

3. Consider the following representation of sets of integers as functions. A set S (of integers) (20 pts) is represented by a function f of type `int -> bool` such that

$$f(n) = true \text{ iff } n \in S$$

Each of the following definitions defines either a set or a function on sets. For each definition, give a **brief** description of what it does (in terms of sets).

(a) `fun A x = false;`

(b) `fun B (f,g) = fn x => f(x) orelse g(x);`

(c) `fun C (f,g) = fn x => f(x) andalso g(x);`

(d) `fun D (x,f) = fn y => (x=y) orelse f(y);`

(e) `fun E (f,g) = fn y => f(y) andalso (not (g(y)));`

4. Consider the following datatype declaration for a small subset of expressions using de-Bruijn notation: (20 pts)

```
datatype deBruijn = dbIntConst of int
             | ## of int   (* constructor for deBruijn indices *)
             | dbOp of (deBruijn * Oper * deBruijn)
             | dbLet of (deBruijn * deBruijn);
```

Complete the definition below such that `closed` is a function which when given a deBruijn expression returns `true` if and only if the expression contains *no free variables.* I.e., every variable occurrence must be within the scope of a declaration for that variable. (Hint: use the second parameter of `f` to keep track of the current lexical "depth".)

```
local
  fun f (dbIntConst(x),D) =


     |f (## k, D) =


     |f (dbOp(e1,_,e2),D) =


     |f (dbLet(e1,e2),D) =


in
  fun closed e = f(e, 0)
end;
```

5. Consider the problem of the shared bank account presented in Lecture 31, where for simplicity we restrict deposits and withdrawals to be of 100 dollars at the time. *Assume that there can be an arbitrary number of threads sharing the same account. Also, assume that each account holder can do an arbitrary number of deposits and withdrawals, in any order.* (20 pts)

   For each of the following solutions, say whether it is correct, namely whether it avoids interference (race hazards), deadlock, and prevents the balance from becoming negative.

   (a)
```
class Account {
    private double balance;

    public Account(double initialDeposit) {
        balance = initialDeposit;
    }

    public double getBalance() {
        return balance;
    }

    public void deposit() {
        balance += 100;
        notify();
   }

    public void withdraw() {
        if ( balance < 100 ) wait();
        balance -= 100;
   }

}

class AccountHolder extends Thread {
    private Account acc;
        ...
    public void run() {
        ...
        acc.deposit();
        ...
        acc.withdraw();
        ...
   }
}
```

(b) Like in (a), but with the methods in Account synchronized

```
class Account {
    ...
    public synchronized double getBalance() { ... }
    ...
    public synchronized void deposit() { ... }
    ...
    public synchronized void withdraw() { ... }
}
```

(c) Like in (b), but with the "if" replaced by a "while" in the method withdraw()

```
class Account {
    ...
     public synchronized void withdraw() {
         while ( balance < 100 ) wait();
         balance -= 100;
     }
}
```

(d) Like in (c), but with the "notify()" replaced by a "notifyAll()" in the method deposit()

```
class Account {
    ...
    public void deposit() {
        balance += 100;
        notifyAll();
    }
    ...
}
```

(e) Like in (d), but with the test on balance done in the AccountHolder class rather than in the Account class

```
class Account {
    ...
     public synchronized void withdraw() {
         balance -= 100;
    }
    ...
}

class AccountHolder extends Thread {
    private Account acc;
        ...
    public void run() {
        ...
        acc.deposit();
        ...
        safeWithdraw(acc);
        ...
    }
    private synchronized void safeWithdraw(Account a) {
        while ( balance < 100 ) wait();
        a.withdraw();
}
```