Name:
SSN :

CSE 428

Fall 1997

Final Exam

16 December 1997

The exam consists of 10 problems on 8 pages, totaling 200 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. Recall that '`@`' is the infix append function. (20 pts)

```
datatype 'a tree = Lf of 'a | Node of 'a tree * 'a * 'a tree;
```

(a)
```
fun fl (Lf x) = x::nil
   |fl (Node(t1,x,t2)) = (fl t2) @ [x] @ (fl t1);
```

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

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

(d)
```
fun bfc n = n(0, fn k=>k+1);
```

2. Which of the following grammars are **unambiguous** ? You may assume the nonterminals N and Id unambiguously generate integers and identifiers, respectively. The symbols "+" and "*" are terminal symbols. (20 pts)

(a) $E ::= N \mid Id \mid E + Id \mid E * Id$

(b) $E ::= N \mid Id \mid E + Id \mid Id * E$

(c) $E ::= N \mid Id \mid E\,E\,+ \mid E\,E\,*$

(d) $E ::= N \mid Id \mid E + E \mid E * E$

3. Recall the definition of Church numerals and booleans: (20 pts)

$$\begin{aligned} \overline{n} &= \texttt{fn f => fn x => f(f(...(f x)...))} \quad (n \text{ applications of } \texttt{f}) \\ \overline{true} &= \texttt{fn x => fn y => x} \\ \overline{false} &= \texttt{fn x => fn y => y} \end{aligned}$$

Also recall the definition of `churchnot` and `even`:

```
fun churchnot cb = cb (fn x => fn y => y) (fn x => fn y => x)
fun even cb = cb churchnot (fn x => fn y => x);
```

(a) Write the function `churchxor` which takes a pair of Church booleans and returns their exclusive OR (also represented as a Church boolean). **Your solution may not use `tochurchbool`, `fromchurchbool`, or any other expression of type `bool`.**

(b) Write the function `mod2` which takes a Church numeral n and computes n `mod` 2, returning the result as a Church numeral. **Your solution may not use `tochurch`, `fromchurch`, `tochurchbool`, `fromchurchbool`, or any other expression of type `bool` or `int`.**

4. Indicate which of the following λ-terms are in β-normal form. (20 pts)

(a) $\lambda f.\lambda n.(b\, n\, (\lambda x.\lambda y.x)\, (f\, (p\, n)))$

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

(c) $\lambda x.\lambda y.\lambda z.(x\, z(y\, z)(\lambda w.w))$

(d) $\lambda x.(x\, (\lambda y.y))$

5. Write a first-order formula which expresses the property that integers n and m are relatively prime (n and m have no common divisors except for 1). (10 pts)

6. Which of the following partial correctness assertions are *valid*? (20 pts)

   (a) $\{X > Y\}\ X := X - Y\ \{X > 0 \wedge X > Y\}$

   (b) $\{X * Y > 0\}$ while $X \neq 0$ do $X := X - 2$ end do $\{X = 0\}$

   (c) $\{\text{true}\}$ if $X > 0$ then $X := X$ else $X := -X\ \{X \geq 0\}$

   (d) $\{X = I + 1\}\ X := X + 1\ \{X = I\}$

7. Consider the following program: (30 pts)

```
program main
  i,j,k : integer;

  procedure bianca(a : integer)
  k : integer;
  begin
    k := a + j;
    j := i + j +1;
    write(a,j,k);
  end bianca;

  procedure kate(b : integer)
  j : integer;
  begin
     j := i + 1;
     bianca(b+j);
     write(k,j);
  end kate;

begin main
  i := 1;
  j := 1;
  k := 1;
  kate(j);
  write(i,j);
end main;
```

What is output by this program under call-by-value parameter passing and

(a) static scoping

(b) dynamic scoping

8. Consider the following program: (20 pts)

```
program main
  i,j,k : integer;

  procedure grumio(a,b,c :integer)
  begin
    a := b + j;
    b := i + a;
    c := k + 1;
    write(a,b,c);
  end grumio;

begin main
  i := 1;
  j := 2;
  k := 3;
  grumio(i,j,k);
  write(i,j,k);
end main;
```

What is output by this program if **all** parameters are passed using the following (state any assumptions you need to make).

(a) call-by-value

(b) call-by-value-result

(c) call-by-reference

9. Give the inference rule for type checking the conditional expression (20 pts)

$$\texttt{if } e_1 \texttt{ then } e_2 \texttt{ else } e_3$$

Your rule should specify how to type this expression in a context Γ. Recall that the judgment for typechecking expression e is of the form $\Gamma \vdash e : \tau$.

10. Extend the following fragment of the function `typecheck: (context * term) -> tp` to handle the case for conditional expressions. Assume the conditional expression `(if` e_1 `then` e_2 `else` e_3`)` is represented as the data object `Cond(e1,e2,e3)` as specified by the given datatype declaration . (20 pts)

```
datatype tp = Int | Bool | --> of tp * tp | ** of tp * tp;
datatype term = I of int | ... | Cond of term * term * term;

type context = (string * tp) list;
infixr -->;
infixr **;
exception untypeable;

fun typecheck(G, I n) = Int
   |typecheck(G, pair(e1,e2)) = (typecheck(G,e1) ** typecheck(G,e2))
    ...
   |typecheck(G, Cond(e1,e2,e3)) =
```