CSE 428, Spring 1998

Final Exam

6 May 1998

- **Please write your name and SSN at the top of the sheet!**
- The exam consists of 7 exercises, totalling 100 points. Please write your answers directly on these pages.
- **Bonus for "No answer":** If you don't know the answer to a question, you may want to leave blank the corresponding space for the solution. This signals that you give no answer for that question. For each "no answer" you will receive a bonus of 10% of the maximum score for that question. Each question counts individually for the bonus, i.e. if an exercise is composed by more than one question, answering one question does not affect the possibility of receiving the bonus for another question in the same exercise. If you begin to write something and later wish you had left it blank, write in large letters "BLANK" over that space.

Exercise 1 (15 pts)

Consider the following grammar for types in mini-ML:

$$\begin{array}{rcl} \mathit{Type} & ::= & \mathit{Tterm} \mid \mathit{Tterm} \to \mathit{Type} \\ \mathit{Tterm} & ::= & \mathit{Tfactor} \mid \mathit{Tterm} * \mathit{Tfactor} \\ \mathit{Tfactor} & ::= & \mathit{Tvar} \mid (\mathit{Type}) \end{array}$$

where *Tvar* is the category of type variables (α, β, γ, etc.) and the terminal symbols are α, β, γ, $\ldots$, $*$, $\to$, (and).

1. Is the grammar ambiguous? (yes or no)
2. What kind of associativity is enforced for $*$?
   What kind of associativity is enforced for $\to$?
   What is the precedence relation between $*$ and $\to$?
3. Construct the derivation trees for $(\alpha \to \beta) \to \gamma$ and for $\alpha \to \beta \to \gamma$.

Exercise 2 (10 pts)

The following is a recursive specification of the Euclid's algorithm for computing the Greatest Common Divisor of two positive natural numbers x and y:

> **function** $GCD(x, y) =$
>   **if** $x = y$
>   **then** x
>   **else if** $x < y$
>   **then** $GCD(x, y - x)$
>   **else** $GCD(x - y, y)$

Write an equivalent function in C or in Pascal using only the basic commands of the mini-imperative language (while, if_then_else, assignment). In particular, don't use recursion or data structures like arrays and lists.

Exercise 3 (10 pts)

Consider the following program in C:

```
int x=0;
int p(int y, int z) { x=x+1; y=y+1; z=z+1; printf(''%d'',x+y+z); }
void main(){
int x=1; p(x,x);
}
```

1. What is the number printed by the program? (Remember that C has static scope, and note that in this program there are two declarations for `x`: one in the block where `p` is declared, and one in the block where `p` is called.)

2. Assume that the head of the procedure `p` is rewritten so that both parameters are passed by reference (this would be possible for instance in C++ by writing `p(int &y, int &z)`). What will be the output of the program in this case?

Exercise 4 (18 pts)

For each of the following ML expressions, write its most general type:

- `fn f => fn g => fn x => (f x, g x)`

- `fn f => fn g => fn x => (f x) (g x)`

- `fn f => fn g => fn (x,y) => (f x) g y`

Exercise 5 (15 pts)

Consider the usual definition of the data type `'a btree`:

```
datatype 'a btree = emptybt | consbt of 'a * 'a btree * 'a btree
```

Write in ML a function `insert : int btree -> int -> int btree` which, given `t : int btree` and `x : int`, where `t` is a search tree, it returns the search tree obtained by inserting a new node labeled by `x` in `t`, if `x` does not occur in `t`, and otherwise it returns `t`. It is not required to balance the resulting tree (but remember that the result must be a search tree). Note that the easiest way to insert a new node in a search tree is to insert it as a leaf.

Exercise 6 (12 pts)

Define in prolog a predicate `eliminates(L1,L2)` which, given a list `L1`, it returns the list `L2` of all the elements of `L1` without repetitions. For instance, if `L1 = [a,b,a,c,a,b]`, then the result is `L2 = [a,b,c]`.

Exercise 7 (20 pts)

Consider the following definition of a parametric class in C++:

```
template<class T> class node {
    protected:  T info;
    public:  node(T x) { info = x; }
};
```

1. Extend `node<T>` so to define the following two subclasses (parametric on `T`):
   - `list<T>`, whose objects are the lists of nodes of type `T`. Define at least the additional field necessary to hold the pointer to the next node, and the constructor methods.
   - `btree<T>`, whose objects are the binary trees of nodes of type `T`. Define at least the additional fields necessary to hold the pointers to the left and right subtrees, and the constructor methods.

2. Define in `btree<T>` the method `invisit` which, when applied to an object of class `btree<T>`, returns an object of class `list<T>` which contains all the nodes resulting from the infix visit of the tree.