CSE 428, Spring 2001. Midterm 1, 15 February 2001

The exam consists of 5 questions, totalling 100 points. Write your answers directly on these pages. **Please remember to write your name and SSN at the top of each sheet.**

Question 1 (total 20 pts)

Consider the following grammar, defining a subset of the language of regular expressions:

```
E ::= A | E+E | EE | E* | (E)
A ::= a | b | c
```

where `E` and `A` are non-terminals and all the other symbols are terminal. `E` is the start symbol.

1. **(8 pts)** For each of the following strings, say whether it is derivable from the grammar or not

   (a) `(ab*c)`

   (b) `a*+c`

   (c) `*a`

   (d) `a**b`

2. **(4 pts)** Prove that the grammar is ambiguous.

3. **(8 pts)** Give an equivalent unambiguous grammar for the same language, defined in such a way that + and concatenation are left associative, `*` has precedence over concatenation, and concatenation has precedence over `+`. (Concatenation is the operation corresponding to the production `E := EE`.)

Question 2 (total 20 pts)

Consider the language of expressions seen in class

```
Exp ::=  Num | Ide | Exp Op Exp | let Ide = Num in Exp end | (Exp)
Op  ::=  + | * | - | /
Num ::=  ... // sequence of digits representing a natural number
Ide ::=  ... // sequence of letters
```

and the interpreter eval in a C++ like language for such a language of expressions:

```
int eval(tree* t, environment* r){
       node* n = t->get_root();
       string ty = n->get_type();
       switch (ty) {
         case "num":
            return n->get_value();
         case "ide":
            return r->lookup(n->get_ide());
         case "op" : {
            int k1 = eval(t->get_left(), r);
            int k2 = eval(t->get_right(), r);
            switch (n->get_op()) {
               case "+":
                  return k1 + k2;
               case "*":
                  return k1 * k2;
               case "-":
                  return k1 - k2;
               case "/":
                  return k1 / k2;
            }
         }
         case "dec": {
            string x = n->get_ide();
            int k = t->get_left()->get_root()->get_value();
            r->add(x,k);
            int result = eval(t->get_right(), r);
            r->pop();
            return result;
         }
       }
}
```

Assume that we want to modify the language by adding a function `max` that takes two generic expressions as arguments, evaluates them, and returns the greatest of the two values. For instance, we want to allow expressions like the following one

```
let x = 1
 in let y = 2
     in max(x+2,y-1)
    end
end
```

The result of the above expression should be 3.

1. **(5 pts)** Show how the grammar should be modified so to cope with the proposed extension.

2. **(15 pts)** Show how the eval function and the classes should be modified so to cope with the proposed extension.

Question 3 (total 20 pts)

Consider the following procedure declaration in a C++ like language:

```
void p(int <parameter-passing method> x){
   int temp;
   temp = y;
   y = y + 1;
   x = x + 1;
   y = x + temp;
}
```

where y is a global variable of type integer. Say what is the final value of y after the following fragment of code:

```
y = 1;
p(y);
```

under each of the following assumptions for the `<parameter-passing method>`:

1. **(6 pts)** Call by value

2. **(7 pts)** Call by reference

3. **(7 pts)** Call by value-result

Question 4 (total 20 pts)

Consider the following procedure declaration in a Pascal-like language:

```
procedure p;
   var x,y: integer;                                 (* variables x,y local to p *)
   procedure q(y: integer);                          (* procedure q local to p   *)
      begin if x = y then r(y+1) else write(y) end; (* body of q                *)
   procedure r(x: integer);                          (* procedure r local to p   *)
      begin if x = y then q(x+1) else write(y) end; (* body of r                *)
   begin                                             (* begin body of p          *)
   x := 1;                                           (* assign 1 to x            *)
   y := 2;                                           (* assign 2 to y            *)
   q(x)                                              (* call to q with actual x  *)
   end;                                              (* end body of p            *)
```

1. What is the output produced by calling `p` in:
   (a) **(6 pts)** a language with dynamic scope

   (b) **(6 pts)** a language with static scope

2. **(8 pts)** For the case of static scope, show the activation records at the moment of the execution of the instruction `write`. In particular, show the static and the control links, and the allocation of the variables and the parameters. Please use a drawing.

Question 5 (total 20 pts)

Consider the following program in a C++ like language:

```
int* y; int w;
void foo(int x){ int* z = new int; *z = x; y = z; }
void main(){
   w = 5;
   foo(w);
   cout << *y;
   delete y;
   ...
}
```

Discuss the allocation of memory during the execution of `foo()`. Namely, list the active variables during the execution of `foo()`, and, for each of them, specify:

- whether it is local, global, or dynamic,
- what is its lifetime: the entire execution of the program, the execution of `foo()`, or until the execution of a `delete` command,
- where it is allocated: stack, heap, or globals' space.