CSE 428, Spring 2000. Midterm 1, 17 February 2000

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 small language of expressions:

```
Exp ::= Num | Exp + Exp | ~ Exp | (Exp)
Num ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
```

where `Exp` and `Num` are non-terminals and all the other symbols are terminal.

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

   (a) `(~ (~ 0))`

   (b) `2 ~ 1`

   (c) `21`

   (d) `2 + ~ 1`

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 + is left associative and ~ has precedence over `+`.

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 C++like for such language

```
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();
            environment* r1 = r->add(x,k);
            return eval(t->get_right(), r1);
         }
       }
}
```

Assume that we want to modify the language by adding the possibility of declaring names associated to the result of generic expressions. For instance, we want to allow expressions like the following one (see next page)

```
let x = 1
 in let y = 2
     in let x = x + y
         in x
        end
    end
end
```

The expression in the declaration should be evaluated in the current environment at the moment in which the declaration is evaluated. Thus, for instance, the result of the above expressions should be 3.

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

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

Question 3 (total 15 pts)

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

```
void p(int <parameter passing-method> x){
   int temp;
   temp = x;
   x = y + 1;
   y = temp + 2;
}
```

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

```
   y = 0;
   p(y);
```

under each of the following assumptions:

1. **(5 pts)** Call by value

2. **(5 pts)** Call by reference

3. **(5 pts)** Call by value-result

Question 4 (total 15 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) else write(y) end;     (* body of q                *)
   procedure r(x: integer);                           (* procedure r local to p   *)
      begin if x = y then write(y) else q(x+1) 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) **(5 pts)** a language with dynamic scope

   (b) **(5 pts)** a language with static scope

2. **(5 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 30 pts)

Consider the following declarations in C++

```
class list{
   int info;
   list* next;
public:
   list(){ info = 0; next = NULL; }
   list(int x, list* l){ info = x; next = l; }
   ~list(){ delete next; }
   int length(){ //returns the length of the list
       int n = 0;
       list* l = this;
       while(l->next!=NULL){ l = l->next; n++; }
       return n;
   }
};

list* makelist(){ //reads a sequence of integers and returns them in a list
   int x;
   list* l = new list();
   cin >> x;
   while(x!=0){ l = new list(x,l); cin >> x; }
   return l;
}
```

For each of the following declarations of p, say whether a call to p leaves memory leaks and/or dangling references. (Note: “a call to p leaves ml/dr” means “after a call to p returns, we have ml/dr”.)

1. **(6 pts)**

```
void p(){
   list* l;
   int x = 0;
   while (x!=4) {
      l = makelist();
      cout << l->length();
      x = x + 1;
   }
   delete l;
}
```

2. **(6 pts)**

```
void p(){
   list* l = makelist();
   cout << l->length();
   m = l;   // m is a global variable of type list* whose value was NULL
}
```

3. **(6 pts)**

```
void p(){
   m = makelist(); // m is a global variable of type list* whose value was NULL
   list l = list(2,m);
   cout << l.length();
}
```

4. **(6 pts)**

```
void p(){
   list* l1 = makelist();
   list* l2 = l1;
   cout << l2->length();
   delete l1;
}
```

5. **(6 pts)**

```
void p(){
   list* l1 = makelist();
   list* l2 = makelist();
   l2 = l1;
   cout << l2->length();
   delete l2;
}
```