CSE 428, Spring 2002. Midterm 1, 14 February 2002

The exam consists of 6 questions, totalling 100 points. Write your answers directly on these pages. **Please remember to write your section, name and SSN at the top of each sheet.**

Question 1 (total 20 pts)

Consider the following grammar, defining a small language of statements:

```
Stm ::= Ide := Num
      | Stm ; Stm
      | if Ide < Ide then Stm else Stm
      | repeat Stm until Ide < Ide
      | try Stm catch Stm

Ide ::= x | y | z

Num ::= -1 | 0 | 1
```

where `Stm`, `Ide` and `Num` are non-terminals and all the other symbols are terminal. `Stm` is the start symbol.

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

   (a) `if x < y then x := 1`

   (b) `try x := y ; y := z catch x := 1`

   (c) `repeat x := -x until x < y`

   (d) `x;y := 1`

2. **(12 pts)** This grammar has three different kinds of ambiguities. Prove each of them by showing three examples of strings with two different derivation trees. (Please use the back of this page if necessary.)

   Note: by "three different kinds of ambiguities" here we mean ambiguities involving three different (combinations of) productions/operators. For instance, precedence between op1 and op2 and precedence between op1 and op3 count as two different ambiguities if op2 $\neq$ op3.

Question 2 (total 20 pts)

The logical conjunction “&&” in C++ is non-strict, namely it has the following semantic rules:

```
   r |- e1  eval  false
----------------------------
 r |- (e1 && e2) eval false


 r |- e1  eval  true    r |- e2  eval  v
------------------------------------------
      r |- (e1 && e2)  eval  v
```

This means that, for instance, the expression

```
(x != 0) && (1/x > 0)
```

gives `false` when the value of `x` is `0`.

In Pascal, on the contrary, the logical conjunction “`and`” is strict, which means that the expression

```
(x != 0) and (1/x > 0)
```

gives `error` (division by 0) when the value of `x` is `0`.

1. **(10 pts)** Write the semantic rules for the “`and`” operator of Pascal.

2. **(10 pts)** Write a pseudo-code in Pascal equivalent to the expression `e1 && e2`, using only the “`if-then-else`”, the “`not`” (if necessary), the “`and`” (if necessary), and the generic expressions `e1` and `e2`.

Question 3 (total 10 pts)

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

```
void p(int <parameter-passing method> x, int <parameter-passing method> y){
   x = y + 1;
   y = z + 2;
   z = x + 3;
}
```

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

```
z = 1;
p(z,z);
```

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

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

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

Question 4 (total 10 pts)

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

```
procedure p;
   var x: integer;                                (* variable x, local to p *)
   var b: boolean;                                (* variable b, local to p *)
   procedure q(x: integer);                         (* procedure q local to p   *)
      begin if x = 0 then r(b) else r(not b) end;   (* body of q                *)
   procedure r(b: boolean);                         (* procedure r local to p   *)
      begin if b then write(x) else q(x-1) end;     (* body of r                *)
   begin                                            (* begin body of p          *)
   x := 1;                                          (* assign 1 to x            *)
   b := true;                                       (* assign true to b         *)
   q(2)                                             (* call to q with actual 2  *)
   end;                                             (* end body of p            *)
```

Assume that the language has static scope. Note: the expression `not b` is true if `b` is false, and viceversa.

1. **(6 pts)** Show the activation records at the moment of the first (and only) 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.

2. **(4 pts)** What is the value printed by the instruction `write`?

Question 5 (total 20 pts)

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

```
int* x;
void p(int& y){ int z = 1; y = y + z; *x = *x + z; }
int main(){
   x = new int;
   *x = 2;
   p(*x);
   cout << *x;
   delete x;
   ...
}
```

1. **(5 pts)** What is the value printed by the instruction `cout << *x;` ?

2. **(15 pts)** Discuss the allocation of memory during the execution of the program. Namely, list the active variables during the execution of `main()` (including the anonymous variables) up to the execution of `delete x`, 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 `p()`, or until the execution of the `delete` command,
   - where it is allocated: stack, heap, or globals' (aka static) space.

Question 6 (total 20 pts)

For each of the following procedure declarations in C++, say whether its call leaves (after returning) dangling pointers and/or memory leaks. If it leaves dangling pointers, specify whether they are to the heap or to the stack. Assume that `p` is a global variable declared as a pointer to `int` and initialized to `NULL`.

1. `void foo(int x) { x = 1; p = &x; }`.

2. `int* foo() { int* q = new int; *q = 1; return q; }`.
   Assume that the call is `p = foo();`

3. `void foo(int* r) { int* q = new int; *q = 1; r = q;}`.
   Assume that the call is `foo(p);`

4. `void foo() { int* q = new int; *q = 1; p = q; delete q; }`

5. `void foo() { int* q = new int; *q = 1; p = new int; *p = *q; delete p; }`