COSC230 Tutorials No. 2 on Programming Languages

(1) The scope rule for names in block-structured programming languages is determined by the structure of the name table. Suppose the compiler is looking at some point in the user program. If a name at the point is found in the name table, the name is visible from that point. Show the contents of "table" of the PL/0 compiler at point P and point Q, and show the visible names at each point. This is called a static analysis because we analyse the program before we actually run the program. The stack structure for the name table reflects the block structure. Note that "X" and "Z" of proc C are not visible from point R. Procedures C and D are brothers, both children of procedure B

```
var X, Y, Z, V, W;
procedure A;
  var X, Y, U, V;
  procedure B;
    var Y, Z, V;
    procedure C;
      var X, Z;
      begin  { C }
        Z:=1;
        X:=Y+Z+W;     ---------- point P
        call A;
      end;
    procedure D;
      var X, Y;
      begin
        X:=Z+U;  ------------------ point R
        call C;
      end;
    begin  { B }
      Y:=X+U+W; ----------------- point Q
      call C;
      call D;
    end;
  begin { A }
    Z:=2;
    U:=Z+W; ---------------------- point S
    call B;
  end;
begin { main }
   X:=1; Y:=2; Z:=3; V:=4; W:=5;
   X:=V+W;
   call A;
end.
```

(2) Describe the contents of the run time stack of the object code for the above program when we have executed point P for the first time. You need not show the actual return addresses for procedure calls. Also show the contents of the stack after point S is executed for the second time. This type of analysis is called a dynamic analysis. Note that this is a hypothetical program, which causes an infinite procedure calls.

(3) Write the object code for the following PL/0 program. Trace the object code at each instruction with the snapshot of the stack.

```
var X, Y, Z;
procedure A;
  var X;
  begin { A }
    X:=Y+Z;
  end;
begin { main }
   X:=1; Y:=2;
   call A;
end.
```

(4) Convert the following grammar to a syntax chart. Design a recursive descent parser for the grammar

1. S → DSa
2. S → a
3. D → cD
4. D → d.

Trace the parser with “ccdaa”.

(5) Convert the followin grammar to a syntax chart. Design a recursive descent parser for the grammar

1. S → aA
2. S → b
3. A → cSa
4. A → null.

The selection set for 1 is {a} and that for 2 is {b}. When you determine selection sets for 3 and 4, you have to consider follow(A), which is the set of symbols that follow A in leftmost derivations. Trace the parser with “acacaaa” and “acacbaa”.