CSI 3120, Lab 9

Exercise 1: variable as a 6-tuple

Consider the 6-tuple model of a variable in imperative languages. How does this model work for Scheme and Prolog?

Exercise 2 (5.9, p. 243)

Consider the following program:

```
program main;
        var x, y, z : integer;
        procedure sub1;
                var a, y, z : integer;
                begin { sub1 }
                ...
                end; { sub1 }
        procedure sub2;
                var a, x, w : integer;
                procedure sub3;
                        var a, b, z : integer;
                        begin { sub3 }
                        ...
                        end; { sub3 }
                begin { sub2 }
                ...
```

```
                end; { sub2 }

begin { main }

...

end. { main }
```

List all the variables; along with the program units where they are declared, that are visible in the bodies of sub1, sub2 and sub3, assuming static scoping is used.

Exercise 3 (5.12, p. 245)

Consider the following program:

```
program main;

        var x, y, z : integer;

        procedure sub1;

                var a, y, z : integer;

                begin { sub1 }

                ...

                end; { sub1 }

        procedure sub2;

                var a, b, z : integer;

                begin { sub2 }

                ...

                end; { sub2 }

        procedure sub3;

                var a, x, w : integer;
```

```
                begin { sub3 }

                ...

                end; { sub3 }

begin { main }

...

end. { main }
```

Given the following calling sequences and assuming that dynamic scoping is used, what variables are visible during execution of the last subprogram activated? Include with each visible variable the name of the unit where it is declared.

1. main calls sub1; sub1 calls sub2; sub2 calls sub3.
2. main calls sub1; sub1 calls sub3.
3. main calls sub2; sub2 calls sub3; sub3 calls sub1.
4. main calls sub3; sub3 calls sub1.
5. main calls sub1; sub1 calls sub3; sub3 calls sub2.
6. main calls sub3; sub3 calls sub2; sub2 calls sub1.

Exercise 4 (7.9, pp. 345-346)

Assume the following rules of associativity and precedence for expressions:

```
Precedence: Highest             *, /, not

                                +, -, &, mod

                                - (unary)

                                =, /=, <, <=, >=, >

                                and

            Lowest              or, xor
```

```
Associativity:  Left to right
```

Show the order of evaluation of the following expressions by parenthesizing all subexpressions and placing a superscript on the right parenthesis to indicate order. For example, for the expression

```
a + b * c + d
```

the order of the evaluation would be represented as

```
((a + (b * c)1 )2 + d)3
```

```
a. a * b - 1 + cb. a * (b - 1) / c mod d
c. (a - b) / c & (d * e / a - 3)
d. -a or c = d and e
e. a > b xor c or d <= 17
f. -a + b
```

Exercise 5 (8.4, p. 388)

Consider the following C program segment. Rewrite it using no goto or break statements.

```
j = -3;
for ( i = 0; i < 3; i++) {
     switch (j+2) {
          case 3:
          case 2: j--; break;
          case 0: j += 2; break;
          default: j = 0;
     }
     if ( j > 0 ) break;
     j = 3 - i;}
```

Exercise 6 (10.2 and 10.3, p. 477-478)

Show the stack with all activation record instances, including static and dynamic chains, when execution reaches position 1 in the following skeletal program. Assume Bigsub is at level 1. Apply the

detailed format from the class notes; work out *all* stack states between the beginning and the point requested in the exercise.

Code from Exercise 10.2, adjusted a little

```
procedure Bigsub is
  MySum : Float;
  procedure C(Plums : Float) is
    begin -- of C
      ... < ---------------- 1
    end; -- of C
  procedure A is
    X : Integer;
    procedure B(Sum : Float) is
      Y, Z: Float;
      begin -- of B
        ...
        C(Z);
        ...
      end; -- of B
    begin -- of A
      ...
      B(X);
      ...
    end; -- of A
  L : Float;
  begin -- of Bigsub
    ...
    A;
    ...
  end; -- of Bigsub
```

Code from Exercise 10.3, adjusted a little (Bigsub calls A, A calls B, B calls A, A calls C, C calls D)

```
procedure Bigsub is
  procedure C is
    procedure D is
      begin -- of D
        ... < ---------------- 1
      end; -- of D
    begin -- of C
      ...
      D;
      ...
    end; -- of C
  procedure A(Flag : Boolean) is
    procedure B is
      begin -- of B
```

```
      ...
      A(false);
    end; -- of B
  begin -- of A
    if flag then B; else C; end if;
  end; -- of A
begin -- of Bigsub
  ...
  A(true);
  ...
end; -- of Bigsub
```