1. Explain why the following grammar is not LL(1):

```
expr ::= constant
      | ( expr )
      | identifier
      | - expr
      | (type) expr

type ::= identifier
      | [ integer ] type
      | ^ type
```

2. Draw a parse tree for the expression

```
- ( ( ^ int ) - x )
```

3. Consider the following function

```
function foo ( j : int )
    var i : int;
    var b : [ 2 : 17 ] int;
    begin
        if (j > 2) and (j < 17) then  i = a[j]+1
    end
```

Describe the activation record that would be created for an invocation of this procedure. Show where data for each variable is stored, and explain during what part of the procedure call sequence that part of the AR would be constructed.
4. Describe in pseudo-code the code that would be generated for the body of the procedure in Figure 3. Assume you have the following instructions:

- push integer or push register or push int(register) (pushes on to stack)
- pop register or pop int(register) (pops from stack to target)
- add (adds top two things on stack, similarly subtract, multiply, and divide)
- deref (treats top of stack as addresses, pushes value at address on stack)
- comp (compares top two items on stack and pops them)
- jump condition, label (jumps if condition is satisfied)

5. Show the control flow graph that would be created for the following loop, and describe what changes would be made if we applied the reduction in strength optimization to this code. Assume \( a \) is an array with 20 reals, indexed 0 to 19, which begins at location -200 from the frame pointer. Variables \( i \) and \( j \) are global.

\[
\text{for } i = 0 \text{ to } 19 \\
\quad j = j + a[i]
\]