Final Exam
(Total: 100 points)

1. The following context-free grammar describes part of the syntax of a Pascal-like language. Non-terminals are given in italic capitals and terminals in bold lower case. **var** represents a variable name and **const** represents a constant. The productions for *ASSN-STMT* are not given, as they do not affect the question.

   PROGRAM → **procedure** *STMT-LIST*

   STMT-LIST → *STMT STMT-LIST*

   STMT-LIST → *STMT*

   STMT → **do var = const to const begin** *STMT-LIST* **end**

   STMT → *ASSN-STMT*

   (1) [**5**] Show the parse tree for the following input:

   ```
   procedure
     do i = 1 to 100 begin
       ASSN-STMT
       ASSN-STMT
     end
     ASSN-STMT
   ```

   (2) [**10**] Write an attribute grammar that computes the number of executed statements for a program conforming to this grammar. Use the fact that the number of times a loop executes can be computed from the two constants that specify the range of the loop variable. You may assume that the lower bound of the range is less than or equal to the upper bound and `const` has an attribute, called *val*, which is an integer value returned by the lexical analyzer.

   (3) [**5**] Annotate the parse tree for the program fragment given in part (1) with the computed attribute values. Annotate the parse tree you drew in part (1), rather than re-draw it.

2. [**8**] Eliminate left recursion of the following translation scheme:

   $$A \to A_1 Y \quad \{A.a = g(A_1.a, Y.y)\}$$
   $$A \to X \quad \{A.a = f(X.x)\}$$

3. [**10**] Multidimensional arrays can be stored in row-major order (last subscript varies fastest), as in C++, or in column-major order (first subscript varies fastest), as in Fortran. **Develop the access function for three-dimensional *column-major* arrays**.

   HINT: Let the subscript ranges of the three dimensions be named `min(1)`, `min(2)`, `min(3)`, `max(1)`, `max(2)`, and `max(3)`, where `min(1)`, `min(2)`, and `min(3)` are lower bounds for dimension one, dimension two, and dimension three, respectively, and `max(1)`, `max(2)`, and `max(3)` are upper bounds for dimension one, dimension two, and dimension three, respectively. Assume the element size is `size`.)

4. In regards to an activation record, what is:

   (1) [**3**] The purpose of the dynamic link?

   (2) [**3**] The return address?

   (3) [**3**] The return value?

5. [**6**] When a function is called, an activation record (or activation frame) is dynamically created and pushed on the call stack (or control stack). This is known as the function-call overhead. What is the main purpose of the call stack? Why a stack is used?

6. [**12**] Indicate which storage (stack, heap, or static data) is bound for each *named* or *unnamed* variable in the following C program.

```
1  #include <stdlib.h>
2  int A;
3  static float B;
4  void foo(short);
5
6  int main() {
7    short C;
8    int *D = malloc(20 * sizeof(int));
9    ...
10 }
11
12 void foo(short E) {
13   static float F;
14   char G[10];
15   ...
16 }
```

7. Considering the following C function:

```
1  int f(int a, int b) {
2      int k, n;
3      k = a;
4      n = 0;
5      while (k < b) {
6          n = n + k;
7          k = k + 1;
8      }
9      return n;
10 }
```

(1) [**6**] Draw the interference graph for the variables and parameters of this function. You are not required to draw the control flow graph, but it could be useful to sketch it out to help with the solution and to leave clues about what might have happened if the graph is not quite correct.

(2) [**4**] Give an assignment of (groups of) variables to registers using the minimum number of registers possible, based on the information in the interference graph. You do not need to go through the steps of the graph coloring algorithm explicitly, although it may be helpful as a guide to assigning registers. If there is more than one possible answer that uses the minimum number of registers, any of them will be fine. Use R1, R2, R3, ... for the register names.

8. Considering the following code fragment:

```
1       a = 1
2       b = 0
3  L0:  a = a + 1
4       b = p + 1
5       if (a > b) goto L3
6  L1:  a = 3
7       if (b > a) goto L2
8       b = b + 1
9       goto L1
10 L2:  a = b
11      b = p + q
12      if (a > b) goto L0
13 L3:  t1 = p * q
14      t2 = t1 + b
15      return t2
```

(1) [**8**] Draw the control-flow graph for the code.

(2) [**5**] Identify the natural loops and determine if they are nested or not (explain why or why not).

(3) [**8**] Indicate the solution for the Reaching-Definitions (RD) Data-flow Analysis problem for the variables `b` only. You do not have to show all the passes of your solution. Simply show the final results of this analysis by indicating for each of the uses, which definition reaches it. Indicate each use as u_x and each definition as d_x where the subscript x indicates the line number for which the use/definition corresponds to.

(4) [**4**] Indicate what optimization could be done according to the result in part (3). Discuss the profitability of your transformation.