Programming Languages and Compilers Comprehensive Exam

September 2000

Please answer each question in a separate booklet.

1. **Enter your answer in the booklet labeled ??.**

   Concurrency can be viewed as a partial order of events (e.g. statements in a program). A partial order $\sqsubseteq$ on a set P is a reflexive, transitive, and antisymmetric relation on the elements of P, i.e.

   $$x \sqsubseteq x$$
   $$x \sqsubseteq y \text{ and } y \sqsubseteq z \Longrightarrow x \sqsubseteq z$$
   $$x \sqsubseteq y \text{ and } y \sqsubseteq x \Longrightarrow x = y$$

   A partial order can be illustrated as a directed graph where each node represents an element of P and there is an edge from x to y if $x \sqsubseteq y$, where x and y are distinct.

   (a) Given an Ada procedure of the form

   ```
   procedure f;
     task one;
     task two;
     task body one is
           s1;
           s2;
           s3;
     end one;
     task body two is
           s4;
           s5;
           s6;
     end two;
   begin
     s7;
     s8;
   end f;
   ```

   draw a graph representing the partially ordered set containing statements s1 through s8. To keep the graph uncluttered, if there is already a path from one node to another in the graph, there is no need to draw an edge between them due to the transitivity of $\sqsubseteq$.

   (b) Given an Ada procedure of the form

   ```
   procedure f;
     task one is entry foo; end one;
     task two is entry bar; end two
     task body one is
           s1;
           two.bar;
           s2;
           accept foo do
               s3;
   ```

```
            end foo;
            s4
      end one;
      task body two is
            s5;
            accept bar do
               s6;
               s7;
            end bar;
            s8;
      end two;
    begin
      s9;
      one.foo;
      s10;
    end f;
```

Draw a graph representing the partially ordered set containing statements s1 through s10.

(c) Suppose the Ada program in part (b) was executed on a single processor. Assuming each of statements s1 through s10 is atomic, give the conditions that must be satisfied for a given order of execution of the program to be possible.

2. **Enter your answer in the booklet labeled ??.**

(a) In Scheme, write a function APP to destructively append one list onto another. For example,

```
> (define x '(1 (2 3) 4))
...
> (define y '(5 6 7))
...
>(APP x y)
...
> x
(1 (2 3) 4 5 6 7)
```

Note that, in this example, `y` would remain unchanged.

(b) What is the efficiency advantage of using your APP procedure over the built-in `append` function?

(c) Under what conditions is it safe to use your APP function rather than `append` in a program?

3. **Enter your answer in the booklet labeled ??.**

Write in SETL a procedure (as concise as possible) that, given a set S of integers, computes a set $\{s_1 \ldots s_n\}$ where each s_i is a proper subset of S such that

$$\bigcup_i s_i \;=\; S$$

$$s_i \cap s_j = \emptyset, \text{ for all } i, j \text{ s.t. } i \neq j$$

$$\sum_{x \in s_i} x = \sum_{y \in s_j} y, \text{ for all } i,j$$

For example, if your procedure is called F, then F({1,2,3,4,5}) would return {{1,4},{2,3},{5}}. If there is no such partitioning of S, then your procedure should indicate that fact.

4. **Enter your answer in the booklet labeled ??.**

   Being of a contrary nature, you decide one day to write a compiler that reads the source program starting at the end of the program and reading backwards to the beginning of the program.

   (a) What would you call the class of the bottom-up parser that you would need to create for your compiler (analogous to LR parsers for normal people)? Justify your answer.

   (b) How would the items constructed for your parser compare to the LR(0) items in an SLR parser? Assume your parser doesn't have any more lookahead then an SLR parser.

   (c) Give the algorithm for computing the closure of a set of items, for the purpose of constructing your parser.

5. **Enter your answer in the booklet labeled ??.**

   Register allocation can be modeled by graph coloring.

   (a) Give a simple heuristic for allocating registers (i.e. coloring the graph) based on an interference graph. Be sure to consider what to do when spilling is required.

   (b) Suppose, through program analysis or using profiling (i.e. results from previous executions), the compiler knows how frequently each basic block will be executed. Refine your heuristic, above, to improve the performance of the compiled program, based on the execution frequencies of basic blocks.

6. **Enter your answer in the booklet labeled ??.**

   An important optimization in optimizing compilers is moving loop invariant computations out of the body of a loop. Assuming a program is represented as a flow graph in which the nodes are basic blocks containing three address code and for which UD-chains and DU-chains have been computed, answer the questions below.

   (a) Define formally the conditions under which a statement of the form

   ```
   x := y + z
   ```

   occurring in the body of a loop would be considered invariant.

   (b) Give the algorithm for detecting loop invariant computations.

   (c) Give the algorithm for performing code motion, once the loop invariant statements have been found.