CS 536 Final Exam

Monday, December 18, 1995

12:25 PM – 2:25 PM CST

107 Psychology Bldg

Instructions

Answer *any five* questions. (If you answer six or seven, only the first five will count.) Each question is worth 20 points. Please try to make your answers neat and coherent. Remember, if we can't read it, it's wrong. Partial credit will be given, so try to put something down for each question (a blank answer always gets 0 points!).

Question #1

Assume we have the following function defined in CSX:

```
int proc f(int i);
      return i+2;
end;
```

Explain how the following assignment statement would be evaluated at run-time:

```
a = 1 + f(2);
```

Be sure to include the steps needed to call and execute function `f`. You may show the actual MIPS instructions that would be executed or explain, in detail, the machine-level actions that would occur.

Question #2

Assume we add a simple "for loop" to CSX:

```
for id = init to limit do
      stmts
end ;
```

`id` is any integer variable. `init` and `limit` are integer valued expressions that are evaluated once, at the beginning of the loop. Iteration terminates as soon as `id` is greater than `limit`. Zero iterations are possible if `init` is greater than `limit`. At the end of each iteration `id` is incremented by 1.

Show an outline of the MIPS code you'd generate for this kind of loop. Design an AST structure appropriate for this for loop. Then extend your code generator to handle an AST representing a for loop.

Question #3

Is the following context-free grammar LL(1)? If it is, explain why. If it isn't, transform it into an equivalent LL(1) grammar.

$$
\begin{aligned}
S &\rightarrow E\ \text{Eof} \\
E &\rightarrow P\ E\ a \\
E &\rightarrow b \\
P &\rightarrow c \\
P &\rightarrow \lambda
\end{aligned}
$$

Question #4

Assume we add the following production to CSX:

```
stmt → begin decls stmts end ;
```

This is a "begin-end block" similar to the "{ ... }" block of C and C++. Declarations within the block are local to it. Statements within the block may access the identifiers declared within the block or identifiers defined in containing scopes, using the usual "nearest definition" rule.

Explain how you'd modify your CSX type checker to handle begin-end blocks.

Where in memory would you allocate the variables and constants that are declared in a begin-end block?

Question #5

CSX has the production

```
name → id [ exp ]
```

which defines a subscripted variable.

Explain what your CSX code generator does to translate a `name`, so that its value is computed into a register.

How would you change your code generator so that it also does *subscript checking*. That is, whenever a subscripted variable is evaluated, its subscript is checked (at run-time) to verify that only elements within the array are accessed.

Question #6

Let G be any context-free grammar that contains no λ productions. Let X and Y be any two terminal symbols (tokens). Adjacent(X,Y) is a predicate (boolean-valued function) that determines whether X and Y can ever be adjacent in any program defined by G.

Explain how to implement Adjacent(X,Y). (C code is not required; just explain the steps needed to test for possible adjacency.)

Question #7

In CSX, a *leaf proc* is a procedure or function that calls no other procedures or functions (ever). Given an AST for a CSX program, explain how you'd determine which procedures and functions are leaf procs.

Assume that we know that `p` is a leaf proc. How can execution of calls to `p` be made more efficient? (In particular, what steps in the caller and callee can be simplified or removed?)