

Final Exam  
CS 410, Fall 1998

December 17, 1998

There are 12 problems on the exam, with 100 points total available. There are 9 pages to the exam, including this one, make sure you have all of them. Don't forget to put your name and USC ID number at the top of the exam. Please read over the whole test before beginning. Good luck!!!

	value	grade
Problem 1	3 pts.	
Problem 2	10 pts.	
Problem 3	2 pts.	
Problem 4	4 pts.	
Problem 5	10 pts.	
Problem 6	6 pts.	
Problem 7	10 pts.	
Problem 8	4 pts.	
Problem 9	12 pts.	
Problem 10	6 pts.	
Problem 11	15 pts.	
Problem 12	18 pts.	
TOTAL:	100 pts.	

**Problem 1 [3 pts]**

Given an NFA with  $n$  states, what's the maximum number of states an equivalent DFA could have?  
[Hint: think about the correspondence between states in an NFA with states in a DFA created from the NFA.]

**Problem 2 [10 pts]**

Show that the following grammar is ambiguous:

$S \rightarrow x y S$   
 $S \rightarrow x x S$   
 $S \rightarrow y S$   
 $S \rightarrow x$   
 $S \rightarrow y$

### Problem 3 [2 pts]

Who pushes actual parameters onto the stack frame for a routine, the caller or callee?

### Problem 4 [4 pts]

In Cool, results of an arithmetic operation can be (circle all that apply):

- objects
- ints
- floats
- booleans (1/0)

### Problem 5 [10 pts]

This question concerns compiler routines to do type checking of an expression. (limit your answers to one sentence at most)

Part A [5 pts] What information is passed into the type-checking routine?

Part B [5 pts] What information is passed back from the type-checking routine?

### Problem 6 [6 pts]

Recall the stack-machine based code-generation scheme we discussed earlier in the semester (when we did a Fibonacci example). Here's a summary:

- all results are left in acc
- sp is the same at the start of evaluating an expression as at the finish
- to eval  $e_1$  *op*  $e_2$ :
  1. eval  $e_1$ , pushing result on stack
  2. eval  $e_2$ , pushing result on stack
  3. do  $acc = e_1$  *op*  $e_2$ , getting the operands from the stack
  4. pop  $e_1$  and  $e_2$  off of stack

What's a simple optimization we can make to this scheme? (hint: it's one we discussed in class too) Give your answer by making changes to the code generation scheme above.

### Problem 7 [10 pts]

The following is the type rule for the `let` construct in Cool. Answer the questions about it that appear below.

```
O, M |- e1 : T1
T1 <= T0
O[T0/X], M |- e2 : T2
```

-----  
O, M :- let x : T0 <- e1 in e2 end : T2

Part A [4 pts] What is the scope of `x`?

Part B [6 pts] Give an English description of any restrictions on the type of `e1`.

### Problem 8 [4 pts]

Would it be possible to apply the loop version of the reduction-in-strength optimization to the following loop? Why or why not?

```
for (int i = 0; i < n; i++) {
  cin >> foo;
  A[foo] = 1;
}
```

5

### Problem 9 [15 pts]

Give semantic rules to compute the attribute `hasPlus`, which is 1 iff the expression has a `+` somewhere in it, and is 0 if not.

$E \rightarrow E_1 + E_2$

$E \rightarrow E_1 - E_2$

$E \rightarrow E_1 * E_2$

$E \rightarrow E_1 / E_2$

$E \rightarrow \text{INTCONST}$

$E \rightarrow \text{IDENT}$

6

### Problem 10 [6 pts]

What is the output of the following program under each of the following assumptions:

- a. lexical scoping
- b. dynamic scoping

Give a brief explanation of your answers.

```
int x = 5;

void f(int y)
{
  if (x < 100)
  {
    int x = 3000;
    f(y);
  }
  x = x - y;
}

main()
{
  int x = 2;
  f(x);
}

cout << x;
```

7

### Problem 11 [15 pts total]

Consider the following Cool-like classes (function bodies elided):

```
class A {
  a : Int;
  g() : Int { . . . }
  h() : Int { . . . }
}

class B inherits A {
  b : Int;
  j() : Int { . . . }
}

class D inherits A {
  c : Int;
  g() : Int { . . . }
  k() : Int { . . . }
}
```

Consider the following dispatch expression that uses this inheritance hierarchy:

```
let x : A ← new D in
x.g()
end
```

**Part A.** At compile time, to generate code for the dispatch, we find the offset for method *g* in the method-offset table for what class?

**Part B.** At runtime where do we get the information about what dispatch table to use?

**Part C.** At runtime we use the dispatch table for which class?

8

## Problem 12 [18 pts]

**Part A [12].** Find the liveness and next-use information for the variables appearing in a statement for each statement in the following basic block. Assume that  $t1 - t4$  are temporaries (not used outside of the block), and that  $a-d$ , and  $m$  and  $n$  are programmer variables (used outside of the block).

I have left room underneath each statement for you to attach the information for that statement.

(1)  $t1 = a + 3$

(2)  $t2 = b * t1$

(3)  $m = c + m$

(4)  $t3 = t1 * 10$

(5)  $t4 = d - t2$

(6)  $n = t3 + t4$

**Part B [6].** For each of the following variables, give its liveness and next use information for when we enter the basic block above.

$b$

$m$

$n$