CS 536
Final Exam

Wednesday, May 16, 2007

7:25 — 9:25 PM

1325 CSST

Instructions

Answer question 1 and any *three* other questions. (If you answer more, only four questions will be graded.) Question 1 is worth 1 point and all other questions are worth 33 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!).

1. The **Java** Virtual Machine (JVM) is designed to support platform-independent execution of what programming language?

2. Recall that in our CSX code generator design when assigning arrays, we always check their sizes prior to doing the actual assignment. This is done in case the source or target (or both) are array parameters. However in the common case in which both source and target are ordinary array-valued variables this size check is unnecessary since type checking guarantees that both arrays have the same size.

   Explain how we should change the translation of array assignments so that array sizes are checked at run-time only when necessary.

3. Assume we have an abstract syntax tree T for a CSX program that contains only a single non-recursive method, `main`. Explain how you would write a recursive function accessCnt(T) that traverses T and estimates how often each field (global variable) in the class is accessed. A field is accessed each time it is read from or written to. You should assume that in conditionals the then and else parts each have a 50% probability of being selected. Moreover, all loops, on average, iterate 10 times.

   How would your design for AccessCnt change if we allow the CSX class to contain non-recursive methods in addition to `main`?

4. Recall that CSX allows no overloading. That is, in each scope each identifier must be uniquely defined. However identifiers used as labels are very different from identifiers used as variables, constants, parameters and methods. Hence allowing an identifier to be used as both a label and "something else" within a scope might be reasonable.

   Explain the changes that would have to be made to CSX symbol tables and type checkers to allow the same identifier to be used as a label and as one other thing (a variable, constant, parameter or method) within the same scope. For example, the following block would be legal in extended CSX:

```
{ int i=21;
   i: while (i > 0){
         i = i - 1;
         if (i == 17)
            break i;
      }
}
```

5. Assume that we add a new kind of conditional statement to CSX, the `signtest`. Its structure is

```
signtest ( exp ) {
        neg:  stmts
        zero: stmts
        pos:  stmts
}
```

   The integer expression `exp` is evaluated. If it is negative, the `stmts` following `neg` are executed. If it is zero, the `stmts` following `zero` are executed. If it is positive, the `stmts` following `pos` are executed.

   Show the AST you would use for this construct. Explain how you would generate code for it. Illustrate your answer using the following example.

```
signtest (A+B-3) {
        neg:  print("Expression is negative.\n");
        zero: print("Expression is zero.\n");
        pos:  print("Expression is positive.\n");
}
```

6. Assume we have a Java class

```
class K {
   int a;
   int sum(){
      int b=1;
      return a+b;
}  }
```

   and the call

```
z = (new K()).sum();
```

   Explain the run-time steps needed to call and execute `sum()` (parameter passing, frame manipulation, return address manipulation, etc.)