CSE 401 Final Exam

December 16, 2010

Name ____

You may have one sheet of handwritten notes plus the handwritten notes from the midterm. You may also use information about MiniJava, the compiler, and so forth handed out at the beginning of the exam. Otherwise, the exam is closed book, closed notes, closed electronics, closed neighbors, open mind, ...

Please wait to turn the page until everyone has their exam and you have been told to begin.

If you have questions during the exam, raise your hand and someone will come to you.

Legibility is a plus as is showing your work. We can't read your mind, but we'll try to make sense of what you write.

1	/ 10
2	/ 34
3	/ 32
4	/ 12
5	/ 12
Total	/ 100

Question 1. (10 points) For each of the following tasks, indicate the earliest stage of the compiler that can always perform that task or detect the situation. Assume the compiler is a conventional one that generates native code (x86, MIPS, etc.) for a language like C++ or Java. Use the following abbreviations to identify the stages:

scan – scanner
parse – parser
sem – semantics/type check
opt – optimization
instr – instruction selection & scheduling
reg – register allocation
run – runtime (i.e., when the compiled code is executed)
can't – can't be done during compilation or execution

__________ A comment beginning with /* terminates with */ before the end of the file

__________ A variable is declared at most once in a scope (function, class, etc.)

__________ Add code to temporarily store a value when there aren't enough registers to hold all live values at a given point in the program.

__________ Replace array subscripting (`a[i]=0; i++`) and a test (`i<n`) with pointer arithmetic (`*p++=0`) and a pointer test (`p<&a[n]`).

__________ Warn the programmer that the declaration of a variable in a subclass hides (shadows) the declaration of the same variable in a superclass.

__________ Detect that there is an extra "`else` *stmt*" clause in the program that is not associated with a previous "`if`"

__________ Detect that in an array reference `a[k]`, the subscript `k` is out of bounds (i.e., in Java where bounds must be checked)

__________ Guarantee that a particular pointer (reference) variable `p` will never be `NULL`.

__________ Guarantee that in a pointer reference `p.x` (`p->x` in C/C++) that the value referenced by `p` will have a field `x`, assuming that `p` is not `NULL`.

__________ Use x86 addressing modes to calculate `x*5` with `leal (%eax,%eax,4),%eax`.

Question 2. (34 points) Runtime data structures and code. Suppose we have the following two Java classes representing characters in a simulation and a third class with a small main program:

```
public class Character {
   int x, y;     // coordinates
   int age;

   void move(int dx, int dy) {
      x = x + dx;  y = y + dy;
   }
   int getX()   { return x; }
   int getY()   { return y; }
   int getAge() { return age; }
}

public class Elf extends Character {
   int weight;
   int age;

   void setAge(int years) { age = years; }
   int  getAge() { return age; }
}

public class Main {
   public static void main(Sring[] useless) {
      Character santa  = new Character();
      Character bernie = new Elf();

      santa.move(12, 24);
      ((Elf)bernie).setAge(42);

}
```

(a) (8 points) Runtime data structures. On the next page draw a diagram of the runtime data structures for this program after the variable declarations in method `main` have been executed.

Show the variables in `main`, the objects they refer to, and how the objects and their fields are organized in memory. Your diagram should show all of the fields in the objects and should assign numeric offsets to each field (0, 4, 8, …) for use in later parts of this question. You do not need to show the details of `main`'s stack frame. Just show pointer variables that refer to the objects.

Then add to your diagram any mechanisms needed to support dynamic method dispatch in Java (e.g., vtables). You should ignore constructors. The vtable diagram(s) should also show the numeric offsets of the various pointers in the table(s) (0, 4, 8, …).

(You may remove this page from the exam for reference if you wish.)

Question 2. (cont). (a) Draw your diagram below, including variable names, objects, vtables, field names, and field offsets in objects and vtables.

(continued on next page)

Question 2. (cont.) (b) (7 points) Stack frames. Suppose now that we start executing method `main`, and continue until we reach the body of method `move` after executing `santa.move(12,24)`. Draw a picture that shows the stack frames of methods `main` and `move` right after entering `move` and after executing the function prologue (allocating the stack frame and adjusting the base pointer), but before executing the first assignment statement in the body of `move`. Draw labeled arrows showing where the registers `ebp` (frame pointer) and `esp` (stack pointer) point after executing the prologue code in `move`. Your picture should show the variables and parameters of both methods and their numeric offsets in each stack frame. (Notice that there will be two sets of offset numbers – one for the stack frame of `main` and one for the stack frame of `move`.)

High addresses

. . .

Question 2. (cont.) (c) (9 points) Method call. Now suppose we continue execution and return to the `main` method after method `move` finishes. The next line of code is `((Elf)Bernie).setAge(42)`. Translate this line of code into x86 assembly language as it would be compiled in `main`. You should assume that the stack frame for method `main` is organized as you've drawn it in the previous part of this question, and that the object and vtable layouts are as drawn in part (a). You should use the standard calling conventions described in class, and use register `ecx` as the "this" pointer. You may assume that no code is generated to check the `(Elf)` cast at runtime. Any reasonable x86 code that obeys the regular calling conventions is ok – you don't need to reproduce the code that would be generated by your compiler. You should use the GNU/AT&T x86 assembly language for your code.

(d) (10 points) Method implementation. Last part(!). Give a x86 translation of method `setAge` from class `Elf`. Your code should include all of the code for method `setAge`, including the function prologue and return. Since this is a void method there is no return value required to be placed in `eax` when the function exits. Include the usual method prologue and epilogue code to save/restore the frame pointer even though there may be other ways to write the code without it.

```
Elf$setAge:
```

Question 3. (32 points) Compiler hacking: another question of many parts.

We would like to add a new kind of counting loop to our MiniJava compiler. The syntax is

for *id* **in** *exp1* **to** *exp2* **do** *stmt*

The idea is that *stmt* is to be executed repeatedly, with variable *id* having successive values *exp1*, *exp1*+1, *exp1*+2, …, *exp2* each time that *stmt* is executed. More specifically the loop is executed as follows. First, *id* is assigned the value *exp1.* Then the value of *id* is tested to see if it is less than or equal to *exp2*. If it is, the loop body, *stmt*, is executed and then the value of *id* is incremented by 1. This test-execute-increment sequence is repeated until the test becomes false because the value of *id* exceeds *exp2*.

The variable *id* must be declared previously in the function and must have type **int**. It must be a local variable in the function containing the loop; it may not be a global or class instance variable. The two expressions *exp1* and *exp2* are evaluated only once before the initial assignment to *id* and are not reevaluated during execution of the loop. Further, the loop body *stmt* may not contain any assignments to the variable *id*. (In other words, having evaluated *exp1* and *exp2* at the start of the loop, we can tell exactly how many times the loop will execute and the sequence of values that *id* will assume. Execution of the loop body is not allowed to change that sequence – although, of course, it could return from the function, throw an exception, or otherwise terminate abnormally.)

Answer the rest of this question on the next few pages. You can remove this page and use it for reference as you work on the question. You may also find the other pages handed out along with the test containing the MiniJava grammar and some of the AST class declarations to be useful.

Write

Your

Answers

On

The

Next

Few

Pages

(continued next page)

Question 3. (cont.) (a) (3 points) What new tokens would need to be added to the scanner and parser of our MiniJava compiler to add the new `for` statement to the original MiniJava grammar? Just list the tokens; you don't need to give a JFlex or CUP specification for them.

(b) (7 points) Complete the following class to define a new AST node class for this new `for` statement. You only need to define instance variables and the constructor. Assume that all appropriate package and import declarations are supplied, and don't worry about visitor code.

```
public class ForLoop extends Statement {
   // add instance variables below




   // constructor – add parameters and body

   public ForLoop ( ________________________________________________ ){
        super(line_nbr);




   }

}
```

Question 3. (cont.) (c) (6 points) Complete the CUP specification below to define a new production for the `for` statement and the associated semantic action(s) needed to parse `for` and insert an appropriate `ForLoop` node (as defined in part (b)) into the AST. We have added the necessary additional code to the parser rule for `Statement` as shown below.

```
Statement ::= ...
            | ForStatement:s  {: RESULT = s; :}
            ...
            ;

ForStatement ::=
```

(d) (6 points) Describe the checks that would be needed in the semantics/type-checking part of the compiler to verify that a `for` statement is legal. You do not need to give code for a visitor method or anything like that – just describe what rules (if any) need to be checked.

Question 3. (cont.) (e) (10 points) Describe the code that would be generated for the new loop statement. You need to show the instructions, labels, and any other assembly language code that need to be generated for the `loop` statement itself, and show where the generated code for *exp1*, *exp2*, the assignment(s) to *id*, and *stmt* would appear in the code sequence for this new loop. If you need to create any temporary variables or otherwise save values somewhere, be sure it is clear what you are doing and where the values are stored.

Question 4. (12 points) Register allocation. Considering the following C function:

```
int f(int a, int b) {
   int k, n;
   k = a;
   n = 0;
   while (k < b) {
      n = n + k;
      k = k + 1;
   }
   return n;
}
```

(a) 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.

(b) 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.

Question 5. (12 points) A little optimization. For this question we'd like to perform local constant propagation and folding, followed by dead code elimination. We have a C function with the following array declaration:

```
double A[10][10];           // 10x10 array of 8-byte doubles
```

The first column of the table below gives the three-address code generated for this assignment statement:

```
A[3][5] = a[3][5] + x;
```

(a) Fill in the second column with the statements from the first column after they have been modified by local constant propagation and folding (compile-time arithmetic).

(b) In the third colum, check the box "deleted" if the statement would be deleted by dead code elimination after performing the constant propagation/folding optimizations from part (a).

Original Code	After constant propagation & folding	"X" if deleted as dead code
t1 = *(fp+xoffset) // x		
t2 = 3		
t3 = t2 * 10		
t4 = t3 + 5		
t5 = t4 * 8		
t6 = fp + t5		
t7 = *(t6+aoffset) // A[3][5]		
t8 = t1 + t7 // A[3][5]+x		
t9 = 3		
t10 = t9 * 10		
t11 = t10 + 5		
t12 = t11 * 8		
t13 = fp + t12		
*(t13+aoffset) = t8		