CSE 401 Final Exam – Winter 2008 Anderson/Roberts

Name: _____

CSE Email: _____

This is a "closed everything" test. Answer all questions.

Keep this page up until told to start

Total: 90 points.

Question	Max Points	Score
1	10	
2	4	
3	5	
4	4	
5	14	
6	16	
7	16	
8	21	
Total	90	

2 Extra Credit Questions on the back of the last page. _____

1. [10] Put an X in the column of the earliest stage at which each MiniJava rule could be enforced.

	Scanning	**Parsing** (w/o extra action code)	**Type-checking**	**Intermediate Code Gen**
Curly braces grouping statements are balanced				
In expressions * has precedence over +				
Case is not significant in identifiers				
int is followed by a not-previously-declared identifier				
The ** operator is right associative				

2. [4] Given that running a program using an interpreter is often likely to be slower than running a compiled program, give two good (and well-explained) reasons why people still build and use interpreters.

3. [5] Garbage Collection:

   a) Briefly describe how *mark/sweep garbage collection* works.

   b) *Copying collection* can be thought of as an improvement over mark/sweep collection because……

   c) Although it is also true that *copying collection* can be thought of as inferior to mark/sweep collection because……

(You don't have to describe *copying collection* but feel free to if it helps make your answer clearer.)

4. [4] Why is instruction selection more difficult on a CISC machine than a RISC machine?

5. [12] In the following C++ code example, the function **p** takes two integer parameters. Parameter **a** is **passed by value**, and parameter **b** is **passed by reference** (as indicated by the **&** after **int**). As discussed in class, this determines the semantics of whether or not modifications to **b** persist after we return from function **p**. As seen in the code below, in C++ the programmer is still able to refer to **b** inside of function **p** as if it were an integer type (without dereferencing a pointer etc.).

```
int p(int a, int& b) {
   int x;
   int y;
   x = a + b;
   b = x + 5;
   y = b; ← Point B. (Stop here, after executing this stmt)
   return y;
}

void main() {
   int c = 500; ← Point A. (Start here)
   int d = 100;
   int e = 0;
   e = p(c, d);
...
```

Assuming that the code starts executing at Point A, **draw a picture of everything that would be pushed onto the stack from Point A up through Point B**. This includes local variables in **main**. Use the x86 calling convention as discussed in class. You may ignore the issue of caller- and callee-saved registers.

- Be sure to indicate what **esp** (stack pointer) and **ebp** (frame pointer) point to at this point in time.
- Be sure to indicate the actual *value* that has been pushed onto the stack, as well as the *variable name*(s) associated with each stack location. In particular, be sure we can tell how you *implement* pass by reference behind the scenes. You do not need to show the x86 assembly code that accesses variable **b**, but you should be sure that the contents of your stack support call by reference (as opposed to call by value-result, or copy-in, copy-out).

Please write your answer on the next page. -→

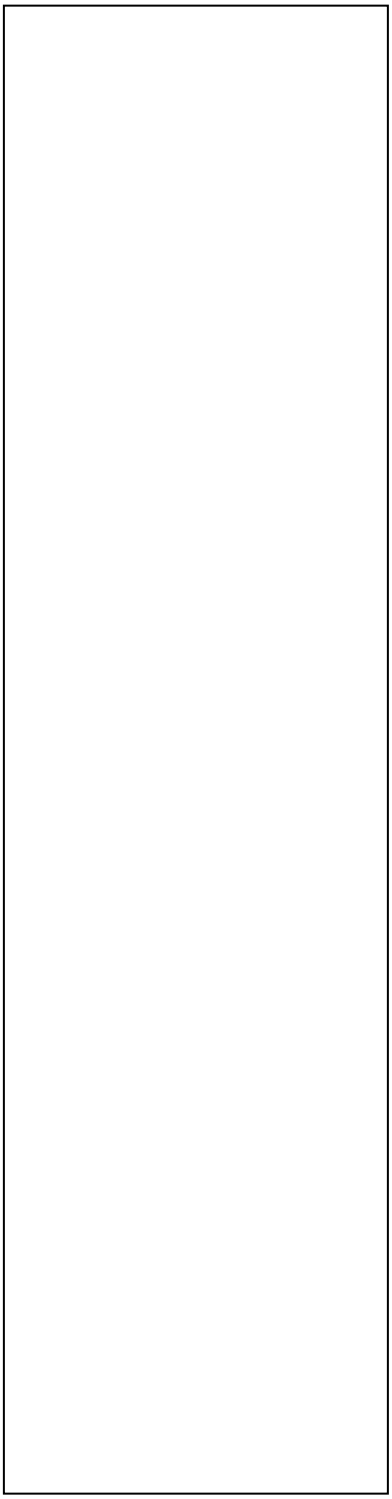
(Page for answer to previous question.)

High Addresses

Draw arrows to indicate what ebp and esp point to:

ebp -> ??

esp -> ??



Low Addresses

6. [16] Given the following **Java** code:

```
public class Dog {
    protected int age;
    protected double weight;

    public void eat(){System.out.println("gobble");};
    public void speak(){System.out.println("bow wow");};
    public int sleep(){return 0;};
}

public class Husky extends Dog {
    protected int favoriteNumber;

    public void speak(){System.out.println("woof");};
    public double findCoug(){return 0.5;};

    public static void main(String[] args){
     Dog A = new Husky();
     Dog B = new Dog();
     Dog C = new Husky();
     B.eat();
     A.speak();
     C.age = 5;
    }
}
```

a) Draw a picture of A, B, and C, including where variables would reside and how their data members would be laid out in memory.

b) Draw any mechanisms that support dynamic method binding (as is the default in Java). (e.g. vtables) You can make the simplifying assumption that this class structure is fixed and known at compile time -- no new classes or methods will need to be added at run time. You may also ignore constructors.

c) Indicate **what part of memory** each of the items you drew in response to parts a) and b) reside in: stack, heap, or static areas. No need to redraw your picture, just be sure to clearly indicate with a label where each part is (unclear answers will be counted as wrong).

Page for answer to Dog question.

7. [16] Name the four different scopes of optimization we discussed in lecture and for each scope, give an example of an optimization appropriate and/or typically done at that scope.

   a) Name of Scope:
      Description of where it applies:

      Name of example optimization:
      Description of Optimization:

   b) Name of Scope:
      Description of where it applies:

      Name of example optimization:
      Description of Optimization:

   c) Name of Scope:
      Description of where it applies:

      Name of example optimization:
      Description of Optimization:

   d) Name of Scope:
      Description of where it applies:

      Name of example optimization:
      Description of Optimization:

8. [21] This question refers to a construct we had on the midterm exam:

```
ifequal (exp1, exp2)
  statement1
smaller
  statement2
larger
  statement3
```

The meaning of this is that *statement1* is executed if the integer expressions *exp1* and *exp2* are equal; *statement2* is executed if *exp1* < *exp2*, and *statement3* is executed if *exp1* > *exp2*. Note that `ifequal, smaller,` and `larger` are all keywords.

Here is one set of context-free grammar productions for the `ifequal` statement that allows either or both of the "`smaller`" and "`larger`" parts of the statement to be omitted, and requires that if both the "`smaller`" and "`larger`" parts of the statement appear, they should appear in that order.

stmt* ::= ifequal (*exp* , *exp*) *stmt optsmaller optlarger
***optsmaller* ::= smaller *stmt* | ε**
***optlarger* ::= larger *stmt* | ε**

The question for *this exam* is about what changes we would need to make to the MiniJava compiler in order to add this statement to the language.

a) What changes would have to be made to the **scanner** to handle this?

b) Is the grammar given above ambiguous or not? Circle one: Yes No

c) Regardless of whether the above grammar is ambiguous or not, describe **<u>two</u>** strategies for dealing with an ambiguous grammar:

The `ifequal` statement repeated again for convenience:

```
ifequal (exp1, exp2)
  statement1
smaller
  statement2
larger
  statement3
```

---

d) Draw the AST node(s) you would add to support the `ifequal` statement.

e) What changes would need to be made to **semantic analysis** to handle the `ifequal` statement?

f) The following is an example of (pseudo) **Minijava Intermediate Language code** that could be generated for an if-else statement: if (*testExpr*) *thenStmt* else *elseStmt*

```
iffalse (testExpr) goto falselabel
thenStmt
goto donelabel
falselabel:
elseStmt
donelabel:
```

On the next page, show the (pseudo) **Minijava Intermediate Language code** that could be generated for an `ifequal` statement. To keep things simple, just generate code for the case that BOTH the `smaller` and `larger` parts of the statement exist.

We have generated the first part of the code for you that will generate code for *exp1* and *exp2* and store their results in temp t1 and t2 respectively. Fill in the rest of your IL code below. Since this is pseudo IL we won't require this to match the actual MiniJava IL exactly, but try to use something as close to MiniJava IL as you can.

t1 := *exp1*
t2 := *exp2*

(Two Extra credit questions on the back of this page.-->)

Extra Credit Questions:

1) Give two advantages of dynamic linking over static linking.

2) What is the primary goal of Binary Optimization (aka Post Link Optimization) as we discussed in class? Describe one method used to achieve this goal.