CS143 Final
Spring 2013

• Please read all instructions (including these) carefully.

• There are 6 questions on the exam, each with multiple parts. You have 180 minutes to work on the exam.

• The exam is open note and open laptop, but no Internet connectivity or computation may be used—you can use your laptop to read electronic notes, but nothing more.

• Please write your answers in the space provided on the exam, and clearly mark your solutions. Please do not use any additional scratch paper.

• Solutions will be graded on correctness and clarity. Each problem has a relatively simple and straightforward solution. You may get as few as 0 points for a question if your solution is far more complicated than necessary. Partial solutions will be graded for partial credit.

NAME: ____________________________________________

In accordance with both the letter and spirit of the Honor Code, I have neither given nor received assistance on this examination.

SIGNATURE: _______________________________________

<table>
<thead>
<tr>
<th>Problem</th>
<th>Max points</th>
<th>Points</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>TOTAL</td>
<td>120</td>
<td></td>
</tr>
</tbody>
</table>
1. **Code Generation and Runtime Systems** (20 points)

Consider the following Cool code:

```cool
class C inherits IO {
    foo(i : Int, j : Int) {{
        if not i = 0 then
            foo(i - 1, j);
            foo(i - 1, j - 1);
            self.out_int(i);
            self.out_int(j);
            self.out_string("\n");
        fi;
    }}
    ...
}
```

Now consider the following stack state, which contains two activation records for `foo()`. The layout of a `foo` activation record, reading from the bottom of the stack towards the top, is: argument `j` (a raw integer), argument `i` (a raw integer), caller's frame pointer, caller's self object, and the return address.

```
0x0003
0x0003
0xffc0
0x3000
0x1240
0x0003
0x0002
0xffac
0x3000
0x1240
0x1240
              <- $sp
```

(a) Assume that the program has either printed nothing so far or the last character the program printed was a newline. What will be the next line of output the program prints? If there are multiple possibilities, show all of them.
(b) Write MIPS assembly code to load the value of $i$ into register $t0$.

(c) Assume the values for $i - 1$ and $j - 1$ are in $t1$ and $t2$, respectively. Write MIPS assembly code for expression $\text{foo}(i - 1, j - 1)$ (just show the caller-side code).
2. **Garbage Collection** (20 points)

   (a) Write a few lines of legal C code that can fail using conservative garbage collection. Explain your reasoning.

   (b) Name a common data structure that cannot be collected under reference counting. What can the programmer do to guarantee such structures are collected?
(c) Complete the code below for a mark-and-sweep collector. You may freely call the methods provided in the class `HeapObject`. Do not add new methods, do not allocate heap memory directly in your code, and modify only the indicated methods.

```cpp
class HeapObject {
public:
    ...
    // returns array of heap objects this object points to
    HeapObject[] getPointers();
    bool isMarked(void);
    setMark(bool isMarked);
    release(); // frees memory of the object
};

void markAndSweep(HeapObject allObjects[], HeapObject roots[])
{
    markAll(roots);
    sweep(allObjects);
}

// FILL IN THE METHODS BELOW
void markAll(HeapObject objs[])
{
    ...
}

void markObject(HeapObject h)
{
    ...
}

void sweep(HeapObject allObjects[])
{
    for (int i = 0; i < allObjects.size(); i++)
    {
        ...
    }
}
```

3. **Register Allocation** (20 points)

(a) On a machine with 4 registers, is it always possible to do register allocation without spilling for a program that has no more than 3 live variables at any program point? If yes, justify your answer; if no, give a counterexample.

(b) On a machine with 3 registers, is it possible to do register allocation without spilling for some program with a register interference graph where every node in the graph has 4 edges? If yes, give an example register interference graph; if no, justify your answer.
(c) Perform register allocation for the program below. Show the live variables at each program point, interference graph, the minimum number of registers required to color the graph without spilling, and your coloring.

d = b + a;
e = d - c;
b = d * a;
if (e < b)
    {   
      f = b + a;
    }
else
    {   
      f = e + a;
    }
g = 42;
g = e * f + g;
h = g * g;
b = f * f;
h = f + b + h;
return h;
4. **Types and Operational Semantics** (20 points)

We introduce pairs into Cool. There is a new type T<> representing a pair of values of type T. The static types of both components of the pair conform to T.

There are five new expressions that work with pairs:

- \( T\langle e_1, e_2 \rangle \) // allocate a new pair with the values of e1 and e2
- x.1 // evaluates to the first component of the pair
- x.2 // evaluates to the second component of the pair
- x.1 <- e // assigns the value of e to the first component of the pair
- x.2 <- e // assigns the value of e to the second component of the pair

Here is a short example Cool class that uses pairs:

```cool
class A { a1 : Int; };
class B inherits A {
    a2 : Int;
    p1 : A<> <- A<new A, new B>;

    test():Object {
        p1.1 <- new B()
    }
}
```

(a) Define subtyping for pair types as follows: \( T_1() \leq T_2() \) if \( T_1 \leq T_2 \). Is this rule sound? If yes, argue why, and if not, explain the reason with a code fragment illustrating the problem.
(b) Give a type checking rule for expression \( T<e_1, e_2> \). Your rule should be sound and also type as many expressions as possible.

(c) Give an operational semantics rule for \( T<e_1, e_2> \).
5. **Local Optimization** (20 points)

Consider the following intermediate code. All the variables are integers, and the only variable alive at the exit of the block is e.

\[
\begin{align*}
\text{b} & := 3 \\
\text{a} & := x + y \\
\text{d} & := b + 1 \\
\text{c} & := a + d \\
\text{a} & := c \times 1 \\
\text{f} & := x + y \\
\text{e} & := a - f
\end{align*}
\]

(a) Consider the following local optimizations: copy propagation, algebraic simplification, common subexpression elimination, constant folding, and dead code elimination. Show an ordering of only these optimizations (including any repeated optimizations) that leads to the best possible code in the fewest number of steps. Show the final code and the ordering of optimizations. For partial credit show the intermediate states of the code.
(b) The final form in part (a) is not optimal. How can you further improve the code—what is the shortest code sequence for this program?
6. **Language Design** (20 points)

Many languages provide the ability to cast an expression of one static type to another static type, with a run-time check to ensure the cast is safe. We’d like to add this feature to Cool using the syntax

```
expr AS T
```

If the object \(o\) to which \(expr\) evaluates has a dynamic type that conforms to the specified type \(T\), the value of the \(as\) expression is \(o\). If not, the default value for type \(T\) is returned. The static type of the \(as\) expression is \(T\).

(a) **Parsing** - Show the production(s) that must be added to the grammar. Are they likely to introduce ambiguity? If so, how would you address it? If not, why not?

(b) **Type Checking** - Although the point of a dynamic cast is to make use of dynamic type information, an \(as\) expression that will always return the default value is probably a bug. Give a sound type checking rule that also rejects \(as\) expressions that can be statically shown to always fail the conformance check at runtime.
(c) *Operational Semantics* - Give the operational semantics rule(s) for an `as` expression.