1. (10pt) **Name:** _____

**INSTRUCTIONS:** Carefully read each question, and write the answer in the space provided. If answers to free response questions are written obscurely, zero credit will be awarded. The correct answer to a free response question with a short answer (i.e., one word or phrase) will never contain any significant words used in the question itself (i.e., "crossword rules"). You are permitted to use **two** 8.5x11 inch sheets of paper (double-sided) containing **hand-written** notes; all other aids (other than your brain) are forbidden. Questions may be brought to the instructor.

This exam is designed to take **80 minutes** (but you may take as much as **150 minutes**, if you want to). There are **800 points** available on the exam, and most questions are worth a number of points that is divisible by ten. These point values are a rough guide to how long I think you should spend on each question.

For **TRUE** or **FALSE** and multiple choice questions, circle your answer.

On free response questions only, you will receive **20%** credit for any question which you leave blank (i.e., do not attempt to answer). Do not waste your time or mine by making up an answer if you do not know. (Note though that most questions offer partial credit, so if you know part of the answer, it is almost always better to write something rather than nothing.)

To get credit for this question, you must:

- Print your name (e.g., "Martin Kellogg") in the space provided on this page.
- Print your UCID (e.g., "mjk76") in the space at the top of **each** page of the exam.

Point distribution (blanks for graders only):

| | |
|---|---|
| **Page 1:** | 10 / 10 |
| **Pages 2 and 3:** | 320 / 320 |
| **Pages 4 and 5:** | 260 / 260 |
| **Pages 6 and 7:** | 210 / 210 |
| **Extra Credit:** | 40 / 0 |
| **Total:** | 840 / 800 |

**I. Multiple Choice and Very Short Answer (120pts).** In the following section, either circle your answer (possible answers appear in **bold**) or write a very short (one word or one phrase) answer in the space provided. No partial credit is possible in this section.

2. (20pt) A context-free grammar is the theoretical grounding for which of these? (Select all that apply.)
   **A** typechecking
   **B** lexing
   **C** parsing
   **D** dataflow analysis

3. (20pt) The dataflow analyses that we use for enabling optimizations should be: (Select all that apply.)
   **A** sound
   **B** complete
   **C** conservative
   **D** monotonic

4. (20pt) List scheduling does a **topological sort** of a precedence graph of operations to construct a schedule, one cycle at a time.

5. (20pt) "If B is a subclass of A, then an object of class B can be used wherever an object of class A is expected" is the statement of which of these:
   **A** Liskov substitution principle
   **B** Rice's theorem
   **C** Church-Turing thesis
   **D** Curry-Howard correspondence

6. (20pt) Which of the following are advantages of using exceptions instead of error return codes in a program? (Select all that apply.) Exceptions...
   **A** ...avoid forgotten error checks.
   **B** ...avoid choosing arbitrary error values.
   **C** ...can carry detailed error information.
   **D** ...enable separation of normal logic and error logic.

7. (10pt) **TRUE** or **FALSE**: regional optimizations are not used in production compilers, because each extended basic block contains exponentially-many paths in the number of constituent basic blocks.

8. (10pt) **TRUE** or **FALSE**: abstract interpretation, dataflow analysis, and pluggable types are equivalently-expressive formalisms. That is, there is nothing that you can express in one of them that you cannot express in the other two.

| | | | |
|---|---|---|---|
| **A.** Stop and Copy | **B.** Boolean Satisfiability | **C.** Single-Static Assignment | **D.** Basic Block |
| **E.** Stack Machine | **F.** Peephole Optimization | **G.** Global Optimization | **H.** Duck Typing |
| **I.** Mark and Sweep | **J.** Interprocedural Optimization | **K.** Activation Record | **L.** Header File |
| **M.** Spill | **N.** Curry-Howard Correspondence | **O.** Graph Coloring | **P.** Symbol Table |

**II. Matching (200pts).** This section contains a collection of terms discussed in class in an "Answer Bank" (choices **A.** through **P.**). Each question in this section describes a situation associated with an answer in the Answer Bank. Write the letter of the term in the Answer Bank that best describes each situation. Each answer in the Answer Bank will be used at most once.

9. (20pt) **J. Interprocedural Optimization** Maria's compiler does a whole-program analysis to determine that it will be profitable to inline a function.

10. (20pt) **N. Curry-Howard Correspondence** Gottfried writes a program in F*, a language whose type system allows him to encode the program's full functional specification into its type.

11. (20pt) **H. Duck Typing** Bernhard's programming language has made a design choice that favors expressiveness and flexibility over static safety.

12. (20pt) **M. Spill** Leonhard's program has too many live variables at the same time to fit into the registers on the machine.

13. (20pt) **O. Graph Coloring** Blaise's compiler reduces the register allocation problem to a standard NP-complete problem instead of solving it directly.

14. (20pt) **L. Header File** Srinivasa needs to typecheck object files that were compiled separately.

15. (20pt) **C. Single-Static Assignment** When proving the correctness of an optimization, Sofia relies on the fact that each variable in the IR is referentially transparent.

16. (20pt) **F. Peephole Optimization** René's compiler replaces an adjacent store and load of the same value with a register move.

17. (20pt) **E. Stack Machine** In order to make it easier to show that his compiler is correct, Carl decides to use a simple model of computation.

18. (20pt) **I. Mark and Sweep** Emmy is writing a garbage collector for a language that allows the programmer to store raw pointers.

**III. Short answer (470pts).** Answer the questions in this section in at most three sentences, unless the question gives other instructions (question-specific instructions have higher priority).

19. (100pt) Rewrite the following program using local optimizations, showing your work, until there are no more local optimizations (from among those we discussed in-class) that you can safely apply. At each step, apply *only one optimization*: do not mix optimizations. The first optimization has been applied for you. After applying each optimization, rewrite *only the lines that were changed by that optimization* and clearly label which optimization you have applied, as in the example.

```
1  a := 2 * x
2  b := 2 + 2
3  c := a * b
4  d := 6 + b
5  e := c
6  f := e + d
7  g := c + e
8  print g
```

Constant Folding ⟶

```
1
2  b := 4
3
4
5                    ⟶
6
7
8  ...
```

Best answer is the following:

```
1  DCE'd
2  DCE'd
3  DCE'd
4  DCE'd
5  DCE'd
6  DCE'd
7  g := x << 4
8  print g
```

Grading is based on whether you correctly labeled each step and whether you got to the end. Order doesn't matter. I was pretty generous here. A common mistake was to get nearly to the end, but then leave line 7 as "g := x * 16", which is almost certainly worse than a shift.

20. Suppose that we want to add automatic memory management to Cool. We decide to use a **reference counting** garbage collector.

(a) (20pt) Which two constructs' operational semantics rules must we modify to support reference counting? **New** and **Assignment**

(b) (50pt) Write the **standard** Cool operational semantics rule for **ONE** of these constructs. Circle the construct that you use for this part in your answer to part a. **Either of the following gets full credit:**

$$\frac{so, E, S \vdash e : v, S_1 \quad E(id) = l_{id} \quad S_2 = S_1[v/l_{id}]}{so, E, S \vdash id \leftarrow e : v, S_2}$$

$$\frac{\begin{array}{c} T_0 = \text{if } T = \text{SELF\_TYPE and so} = X(\ldots) \text{ then } X \text{ else } T \\ class(T_0) = (a_1 : T_1 \leftarrow e_1, \ldots, a_n : T_n \leftarrow e_n) \\ \forall i \in [1 \ldots n], l_i = newloc(S) \\ v = T_0(a_1 = l_1, \ldots, a_n = l_n) \\ S_1 = S[D_{T1}/l_1, \ldots, D_{Tn}/l_n] \\ E' = [a_1 : l_1, \ldots, a_n : l_n] \\ so, E', S_1 \vdash \{a_1 \leftarrow e_1 ; \ \ldots \ ; a_n \leftarrow e_n ;\} : v_n, S_2 \end{array}}{so, E, S \vdash new\ T : v, S_2}$$

(c) (50pt) Write a **modified** operational semantics rule for **the SAME construct you chose in part b** that adds support for reference counting. You may assume the existence of a *free* primitive that takes a value as input. **Either of the following gets full credit, as long as its the same one that was used in part b (black text is new). It is an intentional exam design decision that the new assign rule is significantly more complex than the new new rule, since old assign rule (part b) is much simpler. I gave credit for alternative formalisms that introduce a new environment (e.g., $C$) to each rule to hold the reference counts, as well.**

$$\frac{\begin{array}{l} so, E, S \vdash v.rc \leftarrow v.rc + 1 : \_, \_ \\ E(id) = l_{id} \\ v_{old} = S(l_{id}) \\ so, E, S \vdash v_{old}.rc \leftarrow v_{old}.rc - 1 : \_, \_ \\ so, E, S \vdash \text{if } v_{old}.rc = 0 \text{ then } free(v_{old}) \text{ fi} : \_, \_ \\ so, E, S \vdash e : v, S_1 \\ S_2 = S_1[v/l_{id}] \end{array}}{so, E, S \vdash id \leftarrow e : v, S_2}$$

$$\frac{\begin{array}{c} T_0 = \text{if } T = \text{SELF\_TYPE and so} = X(\ldots) \text{ then } X \text{ else } T \\ class(T_0) = (a_1 : T_1 \leftarrow e_1, \ldots, a_n : T_n \leftarrow e_n) \\ \forall i \in [1 \ldots n+1], l_i = newloc(S) \\ v = T_0(a_1 = l_1, \ldots, a_n = l_n, rc = l_{n+1}) \\ S_1 = S[D_{T1}/l_1, \ldots, D_{Tn}/l_n, 1/l_{n+1}] \\ E' = [a_1 : l_1, \ldots, a_n : l_n] \\ so, E', S_1 \vdash \{a_1 \leftarrow e_1 ; \ \ldots \ ; a_n \leftarrow e_n ;\} : v_n, S_2 \end{array}}{so, E, S \vdash new\ T : v, S_2}$$

(d) (40pt) Describe a direct impact of these modifications on code generation, from the perspective of optimization. For example, you might describe an optimization that is easier (or harder) on the resulting code, or an optimization that now more (or less) profitable. **Manipulating the reference count increases register pressure from assignments. Another sensible answer is that this makes optimizations to increment and decrement especially valuable, but for full credit I will require a specific example of how one might do this. Another good, common answer was that reference counting makes DCE and copy prop. more important, since you also save the overhead.**

21. (60pt) Show that an abstract interpretation $I = (A, \sqsubseteq, \alpha, \gamma)$ terminates. If you rely on any standard property of an abstract interpretation or one of its components that we discussed in class, you must give a correct (mathematical) definition of what it means in the context of abstract interpretation.

**We assume that the abstract domain $A$ is a lattice of finite height, or wlog that widening operators exist that allow us to treat it that way. Then, we rely on the monotonicity of $\sqsubseteq$ - that is, that $\sqsubseteq$ obeys the following property: $\forall a, b, c, d \in A. a \sqsubseteq b \wedge c \sqsubseteq d \rightarrow a \sqcup c \sqsubseteq b \sqcup d$. Because $\sqsubseteq$ is monotonic, it can only ever cause us to move *up* in the lattice. Since the lattice is of finite height, we can modify each location at most $k-1$ times, where $k$ is the height of the lattice (or, if widening is in use, the effective height of the lattice afte the widening operator has been applied). Thus, the algorithm must terminate.**
**Rubric: -20 for a correct proof but no definition of monotonicity; -10 for an informal definition. -10 for not mentioning that the lattice needs to be finite. -30 for "finite height and monotonicity" and no actual proof. No penalty for unnecessarily complicated proof structures (e.g., one of you proved this by contradiction) or unnecessary but true definitions and facts.**

22. (40pt) Support or refute the following claim: it is always profitable to unroll a loop if you can prove that doing so is safe.

**Should refute. Loop unrolling's profitability depends a lot on indirect effects: it increases program size (which can overflow the instruction cache), it changes caching behavior in other ways, unrolled loop may use more registers; if that causes a value to spill, it's almost certainly not worth it, etc. A full credit answer needs to include at least one example of an indirect effect to support the argument.**

23. (40pt) Support or refute the following claim: a compiler can change the x86-64 calling convention for methods that it generates. **Support or refute answers are possible. A support answer must argue that this is possible for *internal* methods: those that the compiler does not expect to be visible beyond the code being compiled, because the calling convention is actually arbitrary. A refute answer must argue that the compiler *cannot* change the calling convention for methods that will be called by other programs, and that doing so would break separate compilation. 30 points for either of these arguments; 40 points for explicitly noting both sides; 20 points for "it can" without a good explanation of why or what the consequences will be.**

24. (30pt) The goal of a register allocator is *not* just to map the $v$ virtual registers used by a block to the $k$ physical registers that are actually present on the target machine, if the block uses more virtual registers than are actually available (i.e., $v > k$). Why not? **The intention of the question was to elicit this answer: spilling a register takes one or more registers, so the allocator needs to reserve space to actually do that.**
**However, the question was badly-worded, so I also gave full credit for "decide what registers to spill" and similar correct claims about the register allocator. I only took off points here for 1) saying false things, or 2) not actually saying anything of substance.**

25. (40pt) Support or refute the following claim: it is sound for one pluggable type system to rely on type qualifiers from a *different* pluggable type system as a premise in one of its typing judgments. **Must support. This is true as long as 1) both typecheckers are known to run, and 2) the reliance only goes in one direction (that is, we avoid circular reasoning between the typecheckers; equivalently, the typechecker must form a DAG). In other words, it's sound if typechecker A relies on some fact in a qualifier of typechecker B so long as B does not do the same with A. I showed an example of this in class with the Nullness Checker and the Initialization Checker. 30 points for an otherwise correct answer that does not mention circular reasoning, DAGs, or anything else restricting how the typecheckers rely on each other.**

**IV. Extra Credit.** Questions in this section do not count towards the denominator of the exam score.

26. (10pt) In section II (Matching), there is a theme to the names used in the situation descriptions. What is the theme? **Famous mathematicians**

27. (10pt) Give the last name of any three of the people who inspired the names used in the situation descriptions in section II (Matching). **Maria Gaetana Agnesi, Gottfried Liebnitz, Bernhard Riemann, Leonhard Euler, Blaise Pascal, Srinivasa Ramanujan, Sofia Kovalevskaya, Rene Descartes, Carl Friedrich Gauss, and Emmy Noether.**

28. (10pt) Suppose that we wanted to modify the PA3 and PA4 assignments so that it's no longer necessary to pass `-no-pie -static` to `gcc` when assembling your `.s` files. What would you, the student, need to do differently in PA3 and PA4? **You'd need to generate position-independent code.**

29. (10pt) May I include your final PA4 submission as an anonymous "boss battle" in future iterations of the PA4 leaderboard? (Answer "Yes" or "No". Either of those strings receives full credit for this question; any other string, including the empty string, recieves zero. If you are working with a partner, I will only include your submission if you both answer "Yes".) **"Yes" or "No" gets full credit; any other string gets nothing.**

This page intentionally left blank (you may use it as scratch paper, and the proctor will have more scratch paper at the front if you need more).