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 one 8.5x11 inch sheet of paper (double-sided) containing **hand-written** notes; all other aids (other than your brain) are forbidden. Questions may be brought to the instructor.

You have **80 minutes** to complete the exam. 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 **with questions** (including this one).

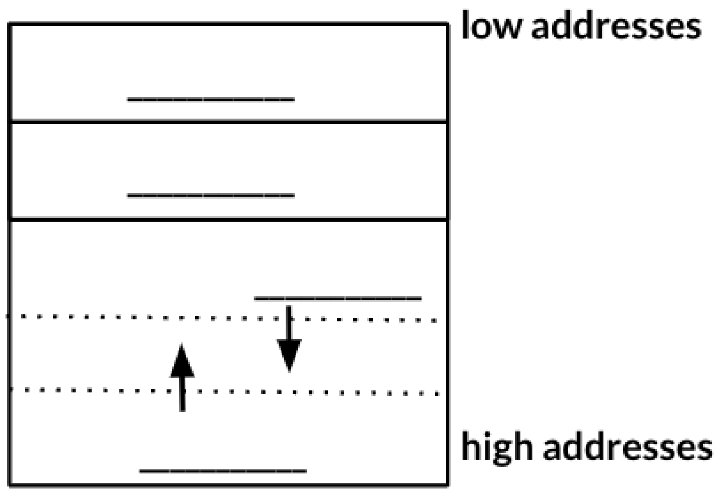
Point distribution (blanks for graders only):

Pages 1 and 2:	______________	/ 120
Pages 3 and 4:	______________	/ 340
Pages 5 and 6:	______________	/ 340
Page 7:	______________	/ 0
Total:	______________	/ 800

I. Multiple Choice and Very Short Answer (110pts). 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. (10pt) **TRUE** or **FALSE**: PA3c2 makes you generate three-address code (TAC) because TAC is a better IR than the alternatives that we discussed in class.

3. (40pt) Label the four sections in a typical program's virtual memory layout in the image below.



4. (15pt) Which of these is the theoretical grounding for *lexing*? (Select all that apply.)
   - **A** recursively-enumerable languages
   - **B** regular languages
   - **C** context-free grammars
   - **D** finite automata

5. (15pt) Which of these is the theoretical grounding for *parsing*? (Select all that apply.)
   - **A** recursively-enumerable languages
   - **B** regular languages
   - **C** context-free grammars
   - **D** finite automata

6. (10pt) **TRUE** or **FALSE**: in a stack machine, the accumulator register is an example of an optimization.

7. (20pt) The least upper bound ($\sqcup$) operator and the less-than operator ($\sqsubseteq$) in an abstract interpretation with abstract domain A must have which of these properties? (Select all that apply.)
   - **A** completeness: $\forall a, b \in A. a \sqsubseteq b$ and $b \sqsubseteq a$ are defined
   - **B** completeness: $\forall a, b \in A. a \sqcup b$ is defined
   - **C** monotonicity: $\forall a, b, c, d \in A. a \sqsubseteq b \land c \sqsubseteq d \rightarrow a \sqcup c \sqsubseteq b \sqcup d$
   - **D** monotonicity: $\forall a, b \in A. a \sqsubseteq a \sqcup b \land b \sqsubseteq a \sqcup b$

A. Functional	**B.** Basic Block	**C.** Call Graph	**D.** Dynamic Dispatch
E. Imperative	**F.** Rice's Theorem	**G.** Control-flow Graph	**H.** Static Dispatch
I. Syntax	**J.** Duck Typing	**K.** Abstract Syntax Tree	**L.** Activation Record
M. Semantics	**N.** Symbol Table	**O.** Referential Transparency	**P.** Currying

II. Matching (200pts). This section contains a collection of terms discussed in class in an "Answer Bank" (choices **A.** through **T.**). 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.

8. (20pt) ______________ Rosa's typechecker needs to track identifier bindings.

9. (20pt) ______________ Malcolm is selecting the right intermediate representation to use for his abstract interpretation.

10. (20pt) ______________ Helen's programming language manages state through destructive updates.

11. (20pt) ______________ Harvey's program will have the same behavior regardless of the run-time type of the associated object.

12. (20pt) ______________ Martin's program analysis needs to do inter-procedural reasoning.

13. (20pt) ______________ Alice is debugging her compiler's stack layout for a single procedure.

14. (20pt) ______________ Cesar uses `grep` to determine whether a program has a property.

15. (20pt) ______________ When reasoning about an OCaml program, Hank replaces a use of a variable with its declaration.

16. (20pt) ______________ Susan knows something about a non-trivial semantic property of a program.

17. (20pt) ______________ Frederick's programming language has made a design choice that favors expressiveness and flexibility over static safety.

III. Short answer (480pts). Answer the questions in this section in at most three sentences, unless the question gives other instructions (question-specific instructions have higher priority).

18. Write an insertion sort implementation in functional style using `fold`. You may assume that the input list only contains integers. A description of insertion sort appears near the end of the exam (as **Document B**) if you need a reminder.

    (a) (20pt) Fill in the following definition of `insertion_sort`. The `insert` helper is defined below.

    ```
    let insertion_sort l = List.fold_left insert _______________ _______________
    ```

    (b) (40pt) Write the `insert` function:

    ```
    let rec insert l e =
    ```

19. (40pt) Support or refute the following claim: the x86-64 calling convention does *not* permit a calling function to assume that the values in registers will be the same after the function call returns.

20. (40pt) Support or refute the following claim: the Cool type system would be unsound without `SELF_TYPE`.

21. (60pt) Consider the following *typechecking* rule for Cool. What is wrong with it? Explain the problem and write the correct rule in the space provided.

$$\frac{\Gamma \vdash e_0 : \mathbf{Bool} \qquad \Gamma \vdash e_1 : \mathbf{T} \qquad \Gamma \vdash e_2 : \mathbf{T}}{\Gamma \vdash \mathbf{if}\ e_0\ \mathbf{then}\ e_1\ \mathbf{else}\ e_2\ \mathbf{fi} : \mathbf{T}}\ \textbf{[If-Then-Else]}$$

22. (60pt) Consider the following *operational semantics* rule for Cool. What is wrong with it? Explain the problem and write the correct rule in the space provided.

$$\frac{\begin{array}{c} \mathbf{T_0} = \text{if } \mathbf{T} = \mathbf{SELF\_TYPE} \text{ and } \mathbf{so} = \mathbf{X}(\ldots) \text{ then } \mathbf{X} \text{ else } \mathbf{T} \\ \text{class}(\mathbf{T_0}) = (\mathbf{a_1} : \mathbf{T_1} \text{ <- } \mathbf{e_1}, \ldots, \mathbf{a_n} : \mathbf{T_n} \text{ <- } \mathbf{e_n}) \\ \forall i \in [1 \ldots n], \mathtt{l}_i = \mathtt{newloc}(\mathbf{S}) \\ \mathbf{v} = \mathbf{T_0}(\mathbf{a_1} = \mathtt{l}_1, \ldots, \mathbf{a_n} = \mathtt{l}_n) \\ \mathbf{E'} = [\mathbf{a_1} : \mathtt{l}_1, \ldots, \mathbf{a_n} : \mathtt{l}_n] \\ \mathbf{so}, \mathbf{E'}, \mathbf{S_1} \vdash \{\mathbf{a_1} \text{ <- } \mathbf{e_1}\ ;\ \ldots\ ;\ \mathbf{a_n} \text{ <- } \mathbf{e_n};\} : \mathbf{v_n}, \mathbf{S_2} \end{array}}{\mathbf{so}, \mathbf{E}, \mathbf{S} \vdash \text{new } \mathrm{T}\ : \mathbf{v}, \mathbf{S_2}}\ \textbf{[new]}$$

Questions on this page refer to **Document A**, which you can find at the end of the exam. **Document A** describes a proposal to add Kotlin-like nullability checking to Cool. You may tear **Document A** out of the exam and refer to it while answering the questions on this page.

23. (80pt) Give a new rule for typechecking **`let`** expressions without initialization that is compatible with **Document A**'s proposal. You may ignore `SELF_TYPE` at your option when you answer this question; if you do include it, you must get it right.

24. (40pt) Support or refute the following claim: **Document A**'s proposal related to `isvoid` expressions is sound. (Hint: use your knowledge of abstract interpretation.)

25. (100pt) Give a new operational semantics rule for dynamic dispatch that is compatible with the proposal in **Document A**.

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?

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).

28. (10pt) What is the difference between the assembly code generated by Cool v1.36 and Cool v1.39?

29. (10pt) What does Susan know about the property in question 16?

30. (10pt) Propose an *answer* for a trivia question in class, of approximately the same difficulty and tone as the kind of questions that I've written. To get credit for this question, your answer must: 1) be sufficiently well-known that I think someone in class other than you knows about it, 2) not be so well-known that I think *everyone* in class will know about it, and 3) be appropriate for use in a class (this means it must be moderately serious). I will use all the answers for which I give credit over the rest of the semester.

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).

Document A: A Proposal for a "Voidable" Type in Cool

The goal of this proposal is to prevent Cool's "dispatch on void" run-time error by 1) checking for void dispatch at the semantic analysis stage, and 2) modifying the dispatch rules so that "dispatch on voidable" is safely handled. This proposal is heavily inspired by Kotlin's "nullability checking" feature (i.e., its ".?" operator), which makes it a compile-time error to directly dereference a possibly-null value by splitting types in those that can be null (e.g., "String?") and those that cannot (e.g., "String") and providing a new operator (".?") for dereferencing nullable types.

For Cool, our concern is the `void` value rather than the `null` value. However, we fundamentally have the same problem: because a value might be `void`, we need to emit code to handle dispatch on void in our compiler. We want to make that unnecessary for values that definitely cannot be void by modifying the language's rules.

Here is a collection of things that we want to be true about Cool after we implement this proposal:

- There should be a new type annotation "?" that optionally may follow each type. For example, Cool should permit us to write types like "Int?" or "Object?".
- Only variables whose types have a "?" after them should ever contain the value `void` at run time.
- The `isvoid` test, if its result is false, should convert a "?" type to its non-"?" counterpart.
- Trying to dereference a "?" type should result in a `void` value rather than a run-time error if the actual value at run-time is `void`, in all relevant expressions.
- Code generation for dereferences of non-"?" types can ignore the possibility of `void` values.
- The semantics of the language for non-"?" types should not change, except for the point directly above.

Document B

Insertion sort works like this: asically you build a sorted result list by taking an element from the unsorted one and inserting it into its proper place in the result list, which starts out empty. This is how most normal humans would sort a deck of playing cards (and even many computer scientists, who are familiar with more efficient sorting algorithms).