Spring, 1993 Name: ______________________

Com S 541 — Programming Languages 1

# Final Exam

This test has 5 questions and pages numbered 1 through 9.
This test is open book and open notes.
Before you begin, please take a moment to look over the entire test so that you can budget your time.
Short answer questions will be graded on the basis of completeness (did you list all the relevant parts of the answer?) soundness (did you make any errors of fact or contradict yourself?) and clarity. You can help ensure clarity by giving examples where possible.
For essays, critical justifications, and the like there may be no one right answer; so the criteria for judgement will be whether your essay is:

- clear (you use examples where appropriate, you write clear sentences, diagrams, and you avoid excess verbosity),
- sound (you start from facts or reasonable assumptions, your logic is convincing), and
- complete (you consider relevant aspects of the problem and alternatives).

The highest grade will be earned by an answer that is, in addition, concise and elegant; that is, answers that say the most with the least amount of excess. Extra consideration will be given to answers that are especially creative. For programming problems, there is usually more than one right answer, hence it is important to present with your program evidence that it is correct. Answers to programming questions will be scored on the following points:

- elegance and clarity (conciseness, using relevant techniques to modularize the solution, making good use of language features, the creativity of your solution, and how clear and convincing your argument is that the code is correct)
- correctness (whether your program solves the given problem correctly in the given programming language)

You should try to eliminate all syntax errors from your programs, as they will affect both the correctness and clarity of your solution; you may receive no points for programs with major syntax errors. Run-time space and time efficiency of programs is not important, unless the problem says so. Also, checking of external inputs, prompting, labeling output, and other conventions for dealing with humans are not important (for this class), unless the problem says so.
The breakdown of points between these categories is roughly as follows.

- 50% for elegance and clarity (using relevant techniques to modularize the solution, making good use of language features, the creativity of your solution),
- 50% for correctness (whether your program solves the given problem correctly in the given subset of the given language)

You are encouraged to use auxiliary subroutines whenever they are helpful.

1. (10 points) Consider the following SML program.

```
val fti =
  (fn (from,to) =>
    (fn next =>
      (fn seed =>
       let val rec loop =
             fn (i,acc) => if (i > to)
                           then acc
                           else loop((i+1), (next(i,acc)))
       in
           loop(from,seed)
       end
       )));

val answer = (fti(1,4)) (fn (x,y::Ls) => ((x * y)::(y::Ls))) (5::[]);
```

(a) What is the type of `fti`?

(b) What is value of `answer`? Show how you calculated it for partial credit.

2. (20 points) Consider the following C++ code.

```
typedef int bool;
const int true = 1;
const int false = 0;

template <class Elem>
class Stack {
public:
  Stack();
  virtual void push(Elem e);
  virtual void pop();
  virtual Elem top();
  virtual bool isEmpty();  // returns true iff *this has no elements
private:
  StackRep* x;  // you don't need to know what a StackRep is
};
```

You are to implement the derived class `CachedStack`, which has the following declaration.

```
template <class Elem>
class CachedStack : public Stack<Elem> {
public:
  CachedStack() : Stack<Elem>(), cache_is_valid(false) {}
  void push(Elem e);
  void pop();
  Elem top();
  bool isEmpty();
private:
  bool cache_is_valid;
  Elem cache;
};
```

That is you are to give the code for the member functions `push`, `pop`, `top`, and `isEmpty` of the class `CachedStack` so that they behave as one would expect for stacks. However, these must work with the cache, in the following sense. When something is pushed on a `CachedStack`, it goes in the cache, and for `top` and `pop` the cache is used if it has anything in it (is valid).

3. (20 points) Consider the following λProlog code.

```
module typed_lambda_calc_check.

kind    term            type.
kind    type_expr       type.
kind    type_context    type.

% abstract syntax of type expressions
type    b       type_expr.      % used instead of o,
                                % because o is already used in lambda prolog
type    arrow   type_expr -> type_expr -> type_expr.

% abstract syntax of simply typed lambda_calculus terms
type    var     string -> term.
type    lambda  string -> type_expr -> term -> term.
type    app     term -> term -> term.

% type contexts
type    tc_empty type_context.
type    tc_add   type_context -> string -> type_expr -> type_context.

% type judgements
type    judgement       type_context -> term -> type_expr -> o.
```

You are to finish this module by defining the relation `judgement`. The relation `judgement` is supposed to model typing judgements in the simply typed lambda calculus. That is `(judgement H M T)` holds if and only if, using the type inference rules of the simply-typed lambda calculus, in the type context `H`, one can prove that `M` has type `T`. For example,

```
?- (judgement tc_empty (lambda "x" b (var "x")) T).
Solving ...
T == arrow b b
yes
?- (judgement (tc_add tc_empty "y" b) (app (lambda "x" b (var "x")) (var "y")) T).
Solving ...
T == b
yes
```

You can introduce auxiliary predicates to help you write the code if you wish.

4. (20 points) Consider the simply-typed lambda calculus as a programming language. It has the following (abstract) syntax, where $\langle id \rangle$ is an identifier.

$$\langle e \rangle ::= \langle id \rangle \mid \lambda \langle id \rangle \texttt{:} \langle t \rangle \ \texttt{.} \ \langle e \rangle \mid \langle e \rangle \ \langle e \rangle \mid \texttt{(} \ \langle e \rangle \ \texttt{)}$$
$$\langle t \rangle ::= \texttt{o} \mid \langle t \rangle \rightarrow \langle t \rangle \mid \texttt{(} \ \langle t \rangle \ \texttt{)}$$

A term of the form $\lambda x : o \,.\, M$ is a function that takes an argument, $y$ of type $o$ and returns $[y/x]M$. This is approximately like the SML expression `(fn x => M)`.

(a) Add syntax to this grammar for exception handling. That is, add productions for raising and handling exceptions to the above. (Use the same style of grammar as above; that is, don't use EBNF.) You may also assume that parentheses are used to disambiguate potential ambiguities of precedence and associativity.

(b) Give an example (or examples) that illustrate all of your added syntax, and say what the result of each example is.

(c) Give type checking (inference) rules for your syntax. (That is, state what conditions have to hold for expressions that have your new syntax to have a given type. Do this formally for maximum points.)

(d) State, in precise English, how the evaluation of any existing expressions is different with exceptions, and give the semantics of the your new expressions.

Part of your grade will depend on how reasonable your semantics are, and part will depend on how well they are explained. Please don't quote irrelevant stuff from a book. (You may assume that the simply-typed $\lambda$-calculus uses call-by-value.)

5. (12 points) Lawrence Flon's article "On Research in Structured Programming" (*ACM SIGPLAN Notices*, Vol 10, Num 10, Oct 1975) makes the point that one should not judge a language by what bad programs one can write in it, but rather by what kinds of good programs one can write in it.

   (a) (i) Describe applications for which SR is particularly well suited. (ii) What features of SR aid in writing good programs for such applications?

   (b) (i) Describe applications for which $\lambda$Prolog is particularly well suited. (ii) What features of $\lambda$Prolog aid in writing good programs for such applications?

   (c) (i) Describe applications for which C++ is particularly well suited. (ii) What features of C++ aid in writing good programs for such applications?