CS 421 Spring 2010 Midterm 2

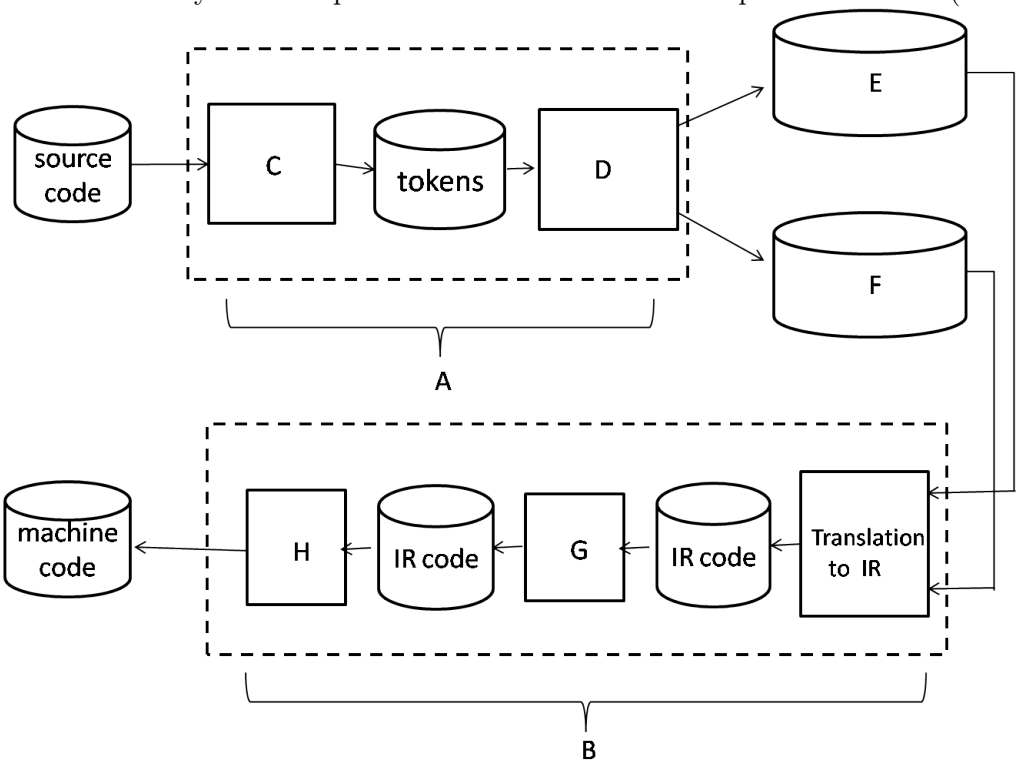
Wednesday, April 7, 2010

Name	
NetID	

- You have **75 minutes** to complete this exam
- This is a **closed book** exam.
- Do not share anything with other students. Do not talk to other students. Do not look at another student's exam. Do not expose your exam to easy viewing by other students. Violation of any of these rules will count as cheating.
- If you believe there is an error, or an ambiguous question, seek clarification from one of the TAs. You must use a whisper, or write your question out.
- Including this cover sheet, there are 11 pages to the exam. Please verify that you have all 11 pages.
- Please write your name and NetID in the spaces above, and at the top of every page.

Question	**Value**	**Score**
1	8	
2	12	
3	14 + 5 XC	
4	15	
5	22 + 5 XC	
6	15	
7	14	
Total	**100 + 10**	

1. (8 pts) Fill in the blanks below, giving the names of the various parts of a compiler. (Recall that the cylinders represent data and the boxes represent actions (i.e. functions).)



A front-end

B back-end

C lexer

D parser

E AST

F symbol table

G optimization

H code generation

2. (22 pts)

   (a) Give the type of the following function: fun f -> fun g -> fun x -> g (f x) x

   $(\alpha \to \beta) \to (\beta \to \alpha \to \gamma) \to \alpha \to \gamma$

   (b) Write an OCaml function *update* such that update f a b is a function that returns b when given a as input but otherwise behaves the same as f.

   let update f a b = fun x -> if x = a then b else f x

   (c) Write an OCaml function *double* that duplicates each element of a list, using fold_right instead of explicit recursion. For example, double [1; 2; 3] = [1; 1; 2; 2; 3; 3]. Remember that fold_right has type (α -> β -> β) -> α list -> β -> β.

   let double lis = fold_right (fun x y -> x :: x :: y) lis []

   (d) Write an OCaml function *sum_pairs* that takes a list of pairs and returns a list containing the sum of the elements of each pair, using map instead of explicit recursion. For example, sum_pairs [(1, 2); (3, 4); (5, 6)] = [3; 7; 11].

   let sum_pairs = map (fun (x, y) -> x + y)

(e) (5 pts extra credit) Write an OCaml function *maxf* that takes a function f and a list lst and returns a pair (max, index), where *max* is the largest value produced by applying f to an element of lst, and *index* is the index in lst of the element x such that f x = max, where the first element of the list has index 0. If there are multiple such elements, you may return the index of any one of them. For example, maxf (fun x -> x + 2) [1; 2; 3] = (5, 2). You may assume that lst is never empty. You may also assume that f takes elements of lst and returns only positive integers. Your function should use fold_right instead of explicit recursion.

let maxf f lst = fold_right (fun x (m, i) -> if f x > m then (f x, 0) else (m, i+1)) lst (0,0)

3. (15 pts) In homework 9, you defined multisets to be functions of type α -> int; in particular, you used the definition `type 'a multiset = 'a -> int`. In that homework, you defined functions add, member, union, disjointUnion, intersection, remove, filter, and fromList. Define the following additional functions on multisets:

   (a) fromSet: 'a set -> 'a multiset, such that fromSet s returns a multiset containing 1 copy of each element in s. Recall that the set type is defined by `type 'a set = 'a -> bool`.

   let fromSet s = fun x -> if s x then 1 else 0

   (b) count: 'a multiset -> 'a list -> int, such that count m lst returns the total number of occurrences of elements from lst in m. You may assume that lst contains no duplicate elements.

   let count m lst = fold_right (+) (map m lst) 0

   (c) subtract: 'a multiset -> 'a multiset -> 'a multiset, such that subtract a b has n copies of the value x if a has p copies and b has q copies and n = p - q. If b has more copies of x than a, then subtract a b should have 0 copies of x.

   let subtract a b = fun x -> max (a x - b x) 0