Programming Languages
CSCI-GA.2110-001 Fall 2014

Practice Midterm Exam Questions

Don't forget to review the questions on Homework 1 as well.

1. Define the following terms: Syntax, semantics, compiler, interpreter, type, imperative language, functional language, Turing complete, universal Turing machine, low-level language, high-level language, higher order functions, dynamic typing, static typing, dynamic scoping, static scoping, abstract data type (ADT), grammar.

2. For each of the following sets, indicate whether a regular expression can be used to define the set. If so, provide the regular expression.

   (a) The set of all numeric literals representing positive integers less than 5000.

   (b) The set of all strings consisting of "(" and ")", such that the parentheses are balanced.

   (c) The set of all names that are palindromes (read the same backwards and forwards).

   (d) The set of all names that have "foo" within them.

3. For each of the following sets, indicate whether a context free grammar (CFG) can be used to define the set. If so, provide the CFG.

   (a) The set of all strings consisting of "(" and ")", such that the parentheses are balanced.

   (b) The set of all compound statements, where a compound statement is either of the form "`<identifier> := <identifier>;`" where `<identifier>` is an identifier (which you may consider a terminal), or of the form "`begin <compound statement> end`".

   (c) The set of all blocks, where a block is a string of the form "`<declarations> <compound statement>`", where `<declarations>` is a list of zero or more declarations of the form "`var <identifier>;`" and `<compound statement>` is defined above. Furthermore, any identifier appearing in the compound statement must be declared in the declarations list.

4. (a) Write a reverse function REV in Scheme, such that it not only reverses a list, but if any of the elements are themselves lists, then they are reversed as well. For example,

   ```
   (REV '((1 2) (3 4) (5))  ==>  ((5) (4 3) (2 1))
   ```

   (b) As you all know, given a value for x, the value of the polynomial

   $$2x^3 + 4x^2 + 3x + 5$$

   can be computed, using Horner's rule, as

   $$((2x + 4)x + 3)x + 5$$

   Write a Scheme function POLY that takes a list of coefficients $(c_n \ldots c_0)$ and returns a function that takes a parameter x and returns the value of $c_n x^n + c_{n-1}x^{n-1} + ... + c_0$, computed using Horner's rule.

5. (a) Suppose there were a dialect of LISP that had the same syntax and semantics as Scheme, except that it was dynamically scoped. Write a *single expression* in that language that would evaluate to a different value than the same expression in Scheme.

(b) What would need to change in your Scheme interpreter in order to support dynamic scoping?

(c) Write, in scheme, the function REDUCE which takes a binary operator (function) and a list and applies the operator to the first element of the list and the result of reducing the operator on the rest of the list. For example, writing

```
(reduce + '(2 3 4 5 6)
```

is equivalent to writing `(+ 2 (+ 3 (+ 4 (+ 5 6))))` and would return 20.

(d) Now modify REDUCE so that if the list has nested sublists, the nested sublists are reduced in the same manner and their result is used in the reduction of the parent list. For example,

```
(reduce + '(2 3 (4 5 (6 7) 8) 9)
```

would return 44.

6. In Scheme, write definitions for cons, car, and cdr that don't use any of the built-in Scheme list operations (such as the built-in cons, car, cdr, list, etc.). That is, the new cons, car, and cdr should appear to the user as if they are behave just like the built-in versions, but are in fact implemented completely differently (Hint: use a lambda).

7. (a) Give an expression in the λ-calculus such that a naive beta-reduction without variable renaming would give an incorrect result.

(b) Given some lambda abstraction f in the lambda-calculus, what can be said about the value of $(Y\,f)$, where Y is the fixed-point operator?

(c) Show that if Y is defined as

$$(\lambda h.(\lambda x.h(xx))(\lambda x.h(xx)))$$

then it satisfies your previous answer.

8. (a) Give a simple example, in an imperative language (e.g. Ada, C, C++, etc.) that would illustrate the difference between static and dynamic scoping. That is, write a short piece of code whose result would be different depending on whether static or dynamic scoping was used.

(b) In a block structured, statically scoped language, what is the rule for resolving variable references (i.e. given the use of a variable, how does one find the declaration of that variable)?

(c) In a block structured but dynamically scoped language, what would the rule for resolving variable references be?

9. In Algol, procedure parameters are represented as closures. Why is it that they can be represented simply as pointers in C?

10. Consider the Ada procedure shown below.

(a) In `add`, why did I introduce the variable `num`? That is, why not perform the addition on `sum` and the call to `another_procedure` using `x`, inside the accept statement?

(b) If `some_procedure` is a short procedure and `another_procedure` is a long procedure, `produce` will spend alot of time waiting for the rendezvous to occur each time through the loop.

Suppose I want `produce` to execute `lotsa_work` as soon as possible after the 100 numbers are sent to `add`. To do this, I don't want `produce` to wait until `add` has received all of those numbers. Modify the program so that, after sending a number, `produce` can continue executing (and sending more numbers), even though `add` was not yet ready to receive it. (hint: You will need another task to serve as an intermediary).

```
procedure foo is

  task produce;
  task add is
    entry input(x:integer);
  end add;

  task body produce is
  begin
    for i in 1..100 loop
         add.input(i);
         some_procedure(i);
    end loop;
    add.input(-1);
    lotsa_work;
   end produce;

  task body add is
    num: integer;
    sum: integer := 0;
    done: boolean := false;
  begin
    while not done loop
       accept input(x:integer) do
          num := x;
       end input;
       if (num >= 0) then
          sum := sum+num;
          another_procedure(num);
       else
          done := true;
       end  if;
    end loop;
    put(sum);
 end add;

begin
 null;
end foo;
```

11. Assume the following procedure is defined in a language that uses pass-by-value-result.

```
procedure f(x,y :integer);
 begin
  x := x + 1;
  y := y * 2;
 end;
```

Write a procedure in C, C++ or Java that simulates this procedure.

12. Show how Ada tasks can be used to implement a binary semaphore; the operations P and V should be provided. Your solution should be of the form:

```
task type Semaphore is
        entry P;
        entry V;
end Semaphore;
task body Semaphore is
        ...
end Semaphore;
```

A semaphore is an object used to enforce mutual exclusion among multiple processes. When a process A performs a P operation on a semaphore, the process can only proceed if no other process has already executed a P on the same semaphore. If another process, B, has already executed a P, then process A will suspend until B has executed a V on the semaphore. At that point, A can continue executing (thus all other processes attempting to execute a P on the same semaphore will suspend until A executes a V on the semaphore).

If you need a better explanation of semaphores, don't hesitate to ask me.

13. What would the following Algol program print (remember, Algol used pass-by-name)?

```
begin
     integer array A[10];
     integer n;

     procedure F(x,y)
       value y; integer x, y;
     begin
       y := y + 1;
       x := 5;
     end;

  n := 1;
  A[1] := 1;
  A[2] := 2;
  F(A[n], n);
  print(A[1],A[2]);
end
```

14. Algol60 used *pass by name* parameter passing. Consider the following Algol60 (more or less) program:

```
integer x,y;

procedure f(a,b);
integer a,b;
begin
        integer i;
        for i := 1 to 10 do
        begin
                b := b + 1;
                print(a);
        end;
end;

begin
        x := 0; y := 1;
        f(x+y, x);
end;
```

Write in C, C++, or Java a program whose run-time behavior is identical (in terms of its implementation) to the above program. It is not sufficient to write a C program that happens to produce the same output. Think about how pass by name is actually implemented (it's not textual substitution).