Homework 7

Due Thursday, 11/3/2011

Please turn in your homework solutions online at `http://www.dci.pomona.edu/tools-bin/cs131upload.php`.

1. (20 points) **Scala Programming**

   Be sure to have a computer with Scala installed (all Macs in the Pomona CS lab have Scala installed with Eclipse). The best source of information on Scala is `http://www.scala-lang.org/` Information on how to install Scala on your own computer either with or without Eclipse is available there as well.

   If you wish to run your programs through Eclipse, go to user documentation and then the "Tutorial Example" for information on how to write a program in Scala using Eclipse. Do read and follow the directions carefully. If you omit steps then Eclipse will not be able to find your `main` method. I find the eclipse implementation a bit buggy, so you may find it easier to just run it on the command line. In that case you can just edit your program using a text editor and run scalac to compile your code using scalac. See the on-line tutorial for further information on running programs and to find more info on the language.

   (a) The scala library class `List`, in `scala.collection.immutable`, provides immutable lists. You can create a list of numbers just by writing `List(1,2,3,4)`. Notice that "new" is not required (though normally it would be to create objects). You can find more information about the class and its operations by going to the general library documentation at `http://www.scala-lang.org/api/current/index.html`, selecting `scala.collection.immutable` from the packages on the left and then clicking on `List` beneath it.

   The element `Nil` is a built in object, while "::" can be used to create a new list by adding a new first element. For example, `5::List(6,7,8,9)` results in a new list containing the numbers from 5 to 9.

   Please write a function that takes a list as input and reverses it. (Do NOT use the built-in reverse method! Write it using the primitives mentioned above.) Write a main method that tests it.

   (b) Look up information about methods `map` and `foreach`. Both apply a function to all the arguments of the list, but `map` takes the results and puts them in a list.

   Use `foreach` to write a method `printall(lst:List[Int])` that prints out all of the elements in a list of integers. Write a sample program that creates a list of integers and calls `printall` to print out all of its values.

   (c) Do problem 4a from homework 4. It asks for you to code up a list comprehension operator. You may use the map and filter functions from the library class `List` from package `scala.collection.immutable`.

   (d) Write a binary search tree class over integers. [Don't bother to write a parameterized class, just have it use integers.] Write methods that will allow you to add new elements to the tree, compute the size of the tree, and to traverse the tree inorder, applying a function f to all elements of the tree and gathering the results in a list. That is the declaration of this method will look like:

```
def inorder(f:int => int):List[Int] {...}
```

If an in-order traversal of the tree would produce $[a_1, \ldots, a_n]$ then this method should provide $[f(a_1), \ldots, f(a_n)]$. Write a program that tests this on a few simple trees and some simple functions.

2. (20 points) **Parser Combinators in Scala**

   In lecture 20, we talked about Parser Combinators in Scala. Please use parser combinators to design a parser for the same grammar you worked on in problem 6 of Homework 5:

   ```
   <exp> ::= ( <tuple> ) | n
   <tuple> ::= <exp> {, <exp> }*
   ```

   where "n" stands for any integer.

   The parser should not only recognize these terms, but should return the sum of all numbers in a tuple. That is, the expression ((1,2,3),4,(1,2,4)) should be recognized as correct and return 17.

   The function that recognizes these tuples and prints out the answers should be named "exp". When executed, your program should recognize the sample tuple above and print out its value.

3. (20 points) **Activation Records for Inline Blocks**

   Please do problem 7.1 from Mitchell, page 191.

4. (10 points) **Time and Space Requirements**

   Please do problem 7.3 from Mitchell, page 193.

5. (15 points) **Ada Parameter Modes**

   The Ada programming language permits parameters to be labeled as `in`, `out`, or `in out`, as in the following procedure definitions, where `T` is some type:

   ```
   procedure test1(in x: T) is begin ... end
   procedure test2(out x: T) is begin ... end
   procedure test3(in out x: T) is begin ... end
   ```

   The modifiers, or modes, have the following meaning:

   - `in`: The value of the parameter `x` cannot be changed inside the procedure. If we call `test1(y)`, the value of `y` is the same before and after the call.
   - `out`: The parameter `x` can be written to, but it cannot be read. If we call `test2(y)`, the value of `y` after the call is the last value written to `x` in the procedure.
   - `in out`: The parameter `x` can be both read and written, and the value of `y` after a call to `test3(y)` is the last value written to `x` in the procedure.

   The language definition does not specify how each mode should be implemented, and the compiler may use any appropriate parameter passing mechanism to implement them.

(a) Which parameter passing mechanism could be used to implement `test1`, `test2`, and `test3`? The choices are pass-by-reference, pass-by-value, and pass-by-value-result (as described in problem 7.6). If more than one is possible, describe the advantages/disadvantages of each.

(b) Consider the following procedure that takes two parameters. Does the following program print the same value for all strategies you outlined for `in out` parameters above?

```
procedure incTwo(in out x:integer, in out y:integer) is
  begin
    x := x + 1;
    y := y + 1;
  end

procedure main() is
  w : integer = 3;
  begin
    incTwo(w,w);
    print w;
  end
```

(c) Discuss the advantages and disadvantages of permitting the compiler such flexibility in how it implements parameter modes.

6. (10 points) **Activation records**

Draw the stack of activation records for the following Ada program (a) after the first call to procedure b; (b) after the second call to procedure b. Show the static (access) and dynamic (control) links for each activation record. (c) Indicate how x is found in procedure c.

```
procedure env is
   integer x = 12;
   procedure a is
      integer y = 2;
      procedure b(z) is
         procedure c is
         begin
            b(x);
         end c;
      begin
         c;
      end b
   begin
      b(x+y);
   end a;
begin
   a;
end env
```