Homework 1: Introduction to Programming Concepts

Due: Monday, September 3, 2008 at 11:00 PM.

In this homework you will learn some of the basics of Oz and the Mozart system [UseModels], and will get an overview of programming concepts [Concepts].

For all Oz programing exercises, you must run your code using the Mozart/Oz system. See the course's "Running Oz" page for instructions about installation and troubleshooting of the Mozart/Oz system on your own computer.

For programming problems for which we provide tests, you can find them all in a zip file, which you can download from the course resources web page or from Webcourses.

If the tests don't pass, please try to say why they don't pass, as this enhances communication and makes commenting on the code easier and more specific to your problem.

Our tests use the functions in the course library's `TestingNoStop.oz`. The `Test` procedure in this file can be passed an actual value, a connective (which is used only in printing), and an expected value, as in the following statement.

```
{Test {CombA 4 3} '==' 24 div (6*1)}
```

The `Assert` procedure in this file can be passed a boolean, as in the following statement

```
{Assert {Comb J I} == {CombB J I}}
```

Calls to `Assert` produce no output unless they are passed the argument **`false`**. Note that you would not use `Browse` or `Show` around a call to `Test` or `Assert`. If you're not sure how to use our testing code, ask us for help.

Turn in (on Webcourses) your code and output of your testing for all exercises that require code. Please upload code as text files with the name given in the problem or testing file and with the suffix `.oz`. Please use the name of the main function as the name of the file. Please upload test output and English answers as plain text files with suffix `.txt` or as PDF files with suffix `.pdf`. (In any case, don't put spaces or tabs in your file names!)

Your code should compile with Oz, if it doesn't you probably should keep working on it. If you don't have time, at least tell us that you didn't get it to compile.

You should use helping functions whenever you find that useful. Unless we specifically say how you are to solve a problem, feel free to use any functions from the Oz library (base environment), especially functions like `Map` and `FoldR`.

Don't hesitate to contact the staff if you are stuck at some point.

For background, you should read Chapter 1 of the textbook [RH04] (except section 1.7). But you may also want to refer to the reference and tutorial material on the Mozart/Oz web site. See also the course resources page.

Problems

The textbook problems are from the *Concepts, Techniques and Models of Computer Programming* book [RH04, section 1.18].

1. (5 points) [Concepts] [UseModels]

   (a) What is pattern matching used for in Oz? (b) Give a brief example.

2. (20 points) [UseModels]

   Do problem 2 in chapter 1, calculating combinations. Note that this should be done without using cells or assignment (that is, in the declarative model).

Your solution's Oz code should be in a file `Comb.oz`, and that file should contain two functions. Part (a)'s solution should be called `CombA`, and part (b)'s solution called `CombB`.

Hint: use the function `Comb` from section 1.3, and use recursion. Don't write the same code twice, instead make function calls.

You must test your code using Mozart/Oz. After doing your own tests (with `Show` or `Browse`) you must run our tests. To do this, put your file `Comb.oz` and our test file `CombTest.oz` in the same directory. Then run our tests by feeding the buffer `CombTest.oz` to Oz. You will have to look at the `*Oz Emulator*` buffer to see the output. Save that buffer when done in CombTest.txt. Then turn in both your `Comb.oz` and the `CombTest.txt` files to Webcourses. (See also the general directions at the beginning of this homework.)

If you have trouble running our tests, see the troubleshooting section of the course's running Oz page. If that doesn't help, contact the course staff.

3. (20 points; extra credit) [Concepts]

   Do problem 3 in chapter 1, program correctness.

4. (10 points) [UseModels]

   Do problem 5 in chapter 1, lazy evaluation.

5. (10 points) [UseModels]

   Do Problem 7 in chapter 1, explicit state.

6. (5 points) [Concepts] [MapToLanguages]

   What is the most important difference between a variable in Oz and a variable in C++ or Java?

7. (5 points) [Concepts]

   In Oz, what happens when a variable is used before it is bound to a value? What is the technical term for this behavior?

8. (10 points; extra credit) [UseModels]

   Do problem 8 in chapter 1, explicit state and functions.

   Your solution's Oz code should be in a file `Accumulate.oz`. Run our tests in `AccumulateTest.oz` Then turn in both your `Accumulate.oz` code and the testing output files to Webcourses.

9. (15 points) [UseModels]

   Do problem 10 in chapter 1, explicit state and concurrency.

10. (20 points) [Concepts]

    Consider the code in Figure 1 on the following page.

    (a) What do you see in the Oz Browser when you run the code in Figure 1 on the next page using Oz?

    (b) Is the implementation of Oz permitted to introduce delays in a thread where the comment `% {Delay 10}` appears in the then part of the if-statement in the procedure `Reserve`?

    (c) What happens when you uncomment that comment; that is, what is shown in the browser when there is a delay of 10 milliseconds in the code at the point where the comment appears?

    (d) Suppose this program was used to reserve seats for a small airplane or a bus. Would the code in `Reserve` correctly ensure that at most `NUM_SEATS` are reserved at every instant of the program's execution, even when there are multiple threads calling `Reserve`?

11. (7 points) [Concepts]

    (a)According to chapter 1, why is programming with both cells and concurrency difficult?

    (b) Have you seen enough evidence to agree or disagree with the book's position on this question?

```
% $Id: Reserve.oz,v 1.2 2008/08/27 03:44:52 leavens Exp $
declare
Count = {NewCell 0}
Passengers = {NewCell nil}
local NUM_SEATS=4 in

   proc {Reserve Name ?Success}
      if @Count =< NUM_SEATS-1
      then % {Delay 10}
         Count := @Count+1
         {ReserveFor Name}
         Success=true
      else
         Success=false
      end
   end

   proc {ReserveFor Name}
      Passengers := Name | @Passengers
   end
end
thread {Reserve alice _} end
thread {Reserve bob _} end
thread {Reserve carl _} end
thread {Reserve denise _} end
thread {Reserve elvira _} end
thread {Reserve fred _} end
{Delay 3000}
{Browse 'Count='#@Count}
{Browse 'Passengers='#@Passengers}
```

Figure 1: Oz code in the file `Reserve.oz`, which is included in the hw1-test.zip file.

References

[RH04] Peter Van Roy and Seif Haridi. *Concepts, Techniques, and Models of Computer Programming*. The MIT Press, Cambridge, Mass., 2004.