Homework 1: Introduction to Programming Concepts

See webcourses and the syllabus for due dates.
In this homework you will learn some of the basics of Oz and the Mozart system [UseModels], and, more importantly, you will get an overview of programming concepts [Concepts].

General Directions

Answers to English questions should be in your own words; don't just quote text from the textbook.
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 Webcourses in the attachments to problem 1.
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 pass to `Browse` or `Show` a call to `Test` or `Assert`, since neither of these procedures returns a value. If you're not sure how to use our testing code, ask us for help.

What to turn in

For problems that require code, you must turn in both: (1) the code file and (2) the output of our tests. 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 either directly into the answer box in the webcourses assignment, or as plain text files with suffix `.txt`. When you upload files, 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. (Email the staff with your code file if you need help getting it to compile.)

Other directions

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).
Don't hesitate to contact the staff if you are stuck at some point.
For background, you should read Chapter 1 of the textbook [VH04] (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.

Reading Problems

The problems in this section are intended to get you to read the book, ideally in advance of class meetings.

Read section 1.1 and 1.2 of the textbook [VH04] and answer the following questions.

1. [UseModels]
   (a) (2 points) What does `{Browse symbol}` do in Oz?
   (b) (3 points) What kind of character must a symbol, i.e., a constant, something that is not a variable identifier, start with in Oz?
   (c) (5 points) Can variables in Oz be assigned a value more than once? (Answer “yes” or “no” and give a brief explanation.

Read sections 1.3-1.15 of the textbook and answer the following questions.

2. (5 points) [Concepts] [UseModels]

   Using Oz’s pattern matching feature, write a function in Oz that returns the second element of a list that has at least two elements. Call this function `SecondItem`. The following are some tests, found in the file SecondItemTest.oz; you should run these tests on your code and hand in the output along with your code (as described above).

```
% $Id: SecondItemTest.oz,v 1.1 2010/01/11 21:57:17 leavens Exp leavens $
\insert 'SecondItem.oz'
\insert 'TestingNoStop.oz'

{StartTesting 'SecondItem...'}
% If you fix your code, then you may have to restart Oz to make these pass...
{Test {SecondItem a|b|nil} '==' b}
{Test {SecondItem "you should know that a string is also a list"} '==' &o}
{Test {SecondItem [5 6 7 8 9]} '==' 6}
{Test {SecondItem [9 10 11 12 13 14]} '==' 10}
{Test {SecondItem [how now brown cow]} '==' now}
{Test {SecondItem [the functional way uses pattern matching]} '==' functional}
{StartTesting 'done'}
```

   Hint: Note that `SecondItem` need not be written recursively; all it needs is a pattern match (or perhaps a nested pattern match). Note: Be sure to put your code in a file named `SecondItem.oz`, otherwise our testing code won’t find your solution. Our tests never invoke `SecondItem` on `nil` or on a list that has less than two items, so you don’t have to do anything special for those cases.

   You are prohibited from using the built-in Oz function `Nth` in your solution. Also, do not use `.1` and `.2` in your solution.

3. (5 points) [Concepts] What happens when the following code executes in Oz? Briefly explain why that happens.

```
local SetIt It in
   It = good
   SetIt = proc {$ X}
              It = X
           end
   {SetIt bad}
   {Browse 'It is '#It}
end
```

4. (5 points) [Concepts] What happens when the following code executes in Oz? Briefly explain why that happens.

```
local X in
   X = X+1
   {Browse 'X is '#X}
end
```

Read sections 1.3-1.15 of the textbook and answer the following questions.

5. (5 points) [Concepts] According to chapter 1, what problems does nondeterminism cause in concurrent programming?

Regular Problems

We expect you'll do the problems in this section after reading the entire chapter. However, you can probably do some of them after reading only part of the chapter.

The textbook problems are from the *Concepts, Techniques and Models of Computer Programming* book [VH04, section 1.18].

6. (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.

   See the course examples page for many examples of Oz functions.

7. (10 points) [UseModels]

   Do problem 5 in chapter 1, lazy evaluation.

8. (10 points) [UseModels]

   Do Problem 7 in chapter 1, explicit state.

9. (15 points) [UseModels]

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

10. [Concepts]

    Consider the code in Figure 1 on the following page, which is also included in the files available for download with this homework from Webcourses (see the attachments to problem 1).

    (a) (5 points) Which thread wins the phone when you run the code in Figure 1 on the next page using Oz?

    (b) (5 points) Is the implementation of Oz permitted to introduce delays in a thread where the comment *`% {Delay 10}`* appears in Figure 1 on the following page?

    (c) (5 points) 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) (5 points) Suppose the threads in Figure 1 on the next page are created by HTTP requests. Would you recommend the way Figure 1 on the following page is coded as a reliable way to achieve the effect of rewarding the fifth visitor to the associated website? Answer "yes" or "no" and give a brief reason.

```
% $Id: GooglePhone.oz,v 1.1 2010/01/12 01:38:39 leavens Exp leavens $
declare
VisitCount = {NewCell 0}
Winner = {NewCell 0}
local WINNING_NUM = 5 in
   proc {CheckForWinner Name}
      % {Delay 10}
      VisitCount := @VisitCount+1
      if @VisitCount == WINNING_NUM
      then
         Winner := Name
         {Browse 'Congratulations'#@Winner#'You have won a Google Nexus phone'}
      else
         {Visit Name}
      end
   end
   proc {Visit Name}
      {Browse 'Welcome'#Name#'You are visitor '#@VisitCount}
   end
end
thread {CheckForWinner fred} end
thread {CheckForWinner sarah} end
thread {CheckForWinner sam} end
thread {CheckForWinner harry} end
thread {CheckForWinner amy} end
thread {CheckForWinner kishore} end
thread {CheckForWinner sally} end
{Delay 3000}
{Browse 'VisitCount='#@VisitCount}
{Browse 'Winner='#@Winner}
```

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

Extra Credit Problems

Extra credit problems are entirely optional. See the course grading policy for details.

11. (20 points; extra credit) [Concepts]

    Do problem 3 in chapter 1, program correctness.

12. (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.

Points

This homework's total points: 105. Total extra credit points: 30.

References

[VH04] Peter Van Roy and Seif Haridi. *Concepts, Techniques, and Models of Computer Programming*. The MIT Press, Cambridge, Mass., 2004.