COP 4020 — Programming Languages 1

Test on Declarative Programming Techniques

Special Directions for this Test

This test has 8 questions and pages numbered 1 through 9.
This test is open book and notes.
If you need more space, use the back of a page. Note when you do that on the front.
Before you begin, please take a moment to look over the entire test so that you can budget your time.
Clarity is important; if your programs are sloppy and hard to read, you may lose some points. Correct syntax also makes a difference for programming questions.
When you write Oz code on this test, you may use anything in the declarative model (as in chapters 2–3 of our textbook). So you must not use cells or the library functions IsDet and IsFree. But please use all linguistic abstractions and syntactic sugars that are helpful.
You are encouraged to define functions or procedures not specifically asked for if they are useful to your programming; however, if they are not in the Oz base environment, then you must write them into your test. You can use the built-in functions in the Oz base environment such as Append, Nth, Reverse, Length, Filter, Map, and FoldR.

For Grading

<table>
<thead>
<tr>
<th>Question:</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Points:</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>15</td>
<td>15</td>
<td>20</td>
<td></td>
<td>100</td>
</tr>
<tr>
<td>Score:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
1. (10 points) [UseModels] Write an iterative function in Oz

FindIndex: <fun> {<List T> <T>} : <Int> >

that takes a list, Ls, of elements of some type T, and an element of type T, Sought, and returns the least integer, I, such that the Ith element of Ls is equal to (using ==) Sought. If no element of Ls is equal to Sought, then FindIndex returns ~1.

Your solution must have iterative behavior, and must be written using tail recursion. Don’t use any higher-order functions, and don’t use the Oz for loop syntax in your solution! (You are supposed to know what these directions mean.)

FindIndex counts indexes starting at 1, as you can see in the following examples.

\insert 'FindIndex.oz'
{StartTesting 'FindIndexTest'}
{Test {FindIndex nil 7} '==' ~1}
{Test {FindIndex [2] 7} '==' ~1}
{Test {FindIndex [7] 7} '==' 1}
{Test {FindIndex [7 2] 6} '==' ~1}
{Test {FindIndex [9 8 1 2 7 6 6 6 6 1 9 2 10] 6} '==' 6}
{Test {FindIndex [9 8 1 2 7 6 6 6 6 1 9 2 10] 9} '==' 1}
{Test {FindIndex [9 8 1 2 7 6 6 6 6 1 9 2 10] 10} '==' 13}
{Test {FindIndex [a c b x y z m q p f] x} '==' 4}
{Test {FindIndex [b b c d e m] b} '==' 1}
{DoneTesting}
2. (10 points) [UseModels] Without using FoldR or Map, write a function in Oz

\[
\text{GiveTitles: } \texttt{\{fun \{List \Atom\} \Atom\}: \text{List \List Atom\}} \text{\}}
\]

that takes a list of atoms, \(Ls\), and an atom, \(Title\), and returns a list of two-element lists, each of which contains the given \(Title\) followed by an atom from \(Ls\). The order in the result corresponds to the order of \(Ls\).

The following are examples.

\[
\text{\{StartTesting 'GiveTitleTest'}\}
\]

\[
\text{\{Test \{GiveTitles nil sir\} '==' nil}\}
\]

\[
\text{\{Test \{GiveTitles [elton david henry] sir\} '==' \[[\text{sir elton}] \,[\text{sir david}] \,[\text{sir henry}]\]\}
\]

\[
\text{\{Test \{GiveTitles [aretha gaga di godiva] lady\} '==' \[[\text{lady aretha}] \,[\text{lady gaga}] \,[\text{lady di}] \,[\text{lady godiva}]\]\}
\]

\[
\text{\{Test \{GiveTitles [philips detroit evil wu] dr\} '==' \[[\text{dr philips}] \,[\text{dr detroit}] \,[\text{dr evil}] \,[\text{dr wu}]\]\}
\]

\[
\text{\{Test \{GiveTitles [a b c d e f g h i j k a] ms\} '==' \[[\text{ms a}] \,[\text{ms b}] \,[\text{ms c}] \,[\text{ms d}] \,[\text{ms e}] \,[\text{ms f}] \,[\text{ms g}] \,[\text{ms h}] \,[\text{ms i}] \,[\text{ms j}] \,[\text{ms k}] \,[\text{ms a}]\]\}
\]

\text{\{DoneTesting\}}

3. (10 points) [UseModels] Using Oz’s built-in FoldR function, write the function

\[
\text{GiveTitles: } \texttt{\{fun \{List \Atom\} \Atom\}: \text{List \List Atom\}} \text{\}}
\]

from the previous problem.

Your solution must use Oz’s built-in FoldR function (but you can also write additional helping functions if you wish)! So you must fill in your answer by completing the code outline below.

\[
\text{\{StartTesting 'GiveTitleTest'}\}
\]

\[
\text{\{Test \{GiveTitles nil sir\} '==' nil}\}
\]

\[
\text{\{Test \{GiveTitles [elton david henry] sir\} '==' \[[\text{sir elton}] \,[\text{sir david}] \,[\text{sir henry}]\]\}
\]

\[
\text{\{Test \{GiveTitles [aretha gaga di godiva] lady\} '==' \[[\text{lady aretha}] \,[\text{lady gaga}] \,[\text{lady di}] \,[\text{lady godiva}]\]\}
\]

\[
\text{\{Test \{GiveTitles [philips detroit evil wu] dr\} '==' \[[\text{dr philips}] \,[\text{dr detroit}] \,[\text{dr evil}] \,[\text{dr wu}]\]\}
\]

\[
\text{\{Test \{GiveTitles [a b c d e f g h i j k a] ms\} '==' \[[\text{ms a}] \,[\text{ms b}] \,[\text{ms c}] \,[\text{ms d}] \,[\text{ms e}] \,[\text{ms f}] \,[\text{ms g}] \,[\text{ms h}] \,[\text{ms i}] \,[\text{ms j}] \,[\text{ms k}] \,[\text{ms a}]\]\}
\]

\text{\{DoneTesting\}}
4. (10 points) [Concepts] [UseModels] Write a curried version of the function \texttt{WeightedAverage}, shown below.

\begin{verbatim}
declare
fun {WeightedAverage W X Y}
  % Requires: 0.0 <= W and W <= 1.0
  (W*X) + ((1.0-W)*Y)
end
\end{verbatim}

The function you are to write should be called \texttt{WeightedAverageCurried}. That is, write

\begin{verbatim}
function Curried: <fun {$ <Float>}: 
  <fun {$ <Float>}: <Float>>>
\end{verbatim}

such that, \texttt{WeightedAverageCurried} takes a Float \texttt{W}, and returns a function that takes as an argument, \texttt{X}, of type Float, and which returns a function that itself takes an argument, \texttt{Y}, of type Float, and returns $(W*X) + ((1.0-W)*Y)$.

The following are examples.

\begin{verbatim}
\insert 'WeightedAverageCurried.oz'
{StartTesting 'WeightedAverageCurriedTest $Revision: 1.1 $'
{Test {{{WeightedAverageCurried 0.5} 4.0} 4.0} '==' 4.0}
{Test {{{WeightedAverageCurried 0.25} 10.0} 20.0} '==' 17.5}
{Test {{{WeightedAverageCurried 0.35} 0.9} 0.8} '==' 0.835}
{Test {{{WeightedAverageCurried 0.0} 0.9} 0.8} '==' 0.8}
{Test {{{WeightedAverageCurried 1.0} 0.9} 0.8} '==' 0.9}
{Test {{{WeightedAverageCurried 0.65} 0.9} 0.8} '==' 0.865}
{DoneTesting}
\end{verbatim}

Please write your answer below.
5. (10 points) [UseModels] Consider the following grammar (where Lists and Atoms are standard).

\[
\langle \text{Text} \rangle ::= \langle \text{List Paragraph} \rangle \\
\langle \text{Paragraph} \rangle ::= \text{para}(\langle \text{List Atom} \rangle)
\]

In Oz, write a function, \textbf{NumWords}: \texttt{\{fun \langle $\langle \text{Text} \rangle \rangle \}: \langle \text{Int} \rangle\}} which takes a Text, \texttt{Txt}, and returns the total number of atoms in it. The following are examples.

\verb|
\insert 'NumWords.oz'
{StartTesting 'NumWords'}
{Test {NumWords nil} '==' 0}
{Test {NumWords [para([and along came java]) para([cpp was miffed with java])]} '==' 9}
{Test {NumWords [para([dum dum dum dum]) para([dum bum]) para([rum scum])]} '==' 8}
{Test {NumWords [para([it was the best 'of' times it was the worst 'of' times])]} '==' 12}
{DoneTesting}
\verb|}
6. (15 points) [UseModels] Using the same grammar as in the previous question,

\[ \text{(Text)} ::= \langle \text{List Paragraph} \rangle \\
\text{(Paragraph)} ::= \text{para}(\langle \text{List Atom} \rangle) \]

in Oz, write a function, \textit{SubstAll}: \texttt{<fun \{$<$Text$>$ \$<Atom>$ \$<Atom>$$>$: <Text>>} which takes a Text, \textit{Txt}, and two atoms \textit{Old} and \textit{New}, and returns a Text that is just like \textit{Txt}, except that each occurrence of the atom \textit{Old} is replaced by the value of \textit{New}. The following are examples.

\begin{verbatim}
\insert 'SubstAll.oz'
{StartTesting 'SubstAll'}
{Test {SubstAll nil java csharp} '==' nil}
{Test {SubstAll [para([and along came java]) para([cpp was miffed with java])] java csharp}
 '==' [para([and along came csharp]) para([cpp was miffed with csharp])]} 
{Test {SubstAll [para([dum dum dum dum]) para([dum bum]) para([rum scum])] dum ah}
 '==' [para([ah ah ah ah]) para([ah bum]) para([rum scum])]} 
{Test {SubstAll [para([it was the best 'of' times it was the worst 'of' times])] times tests}
 '==' [para([it was the best 'of' tests it was the worst 'of' tests])]} 
{DoneTesting}
\end{verbatim}
7. (15 points) [UseModels] In Oz, write the function

ListApply: <fun ($ <List <fun ($ T): S>> <List T>): <List S>>

that, for some types T and S, takes a list, Funs, of functions (each of type <fun ($ T): S>) and a list, Args, of values (each of type T), which has the same length as Funs, and returns a list that contains the results of applying the ith function in Funs to the ith argument in Args. The resulting list preserves the order of the original lists. You should assume that the lists Funs and Args have equal lengths. The following are examples.

\insert 'ListApply.oz'
{StartTesting 'ListApply'}
{Test {ListApply nil nil} '==' nil}
{Test {ListApply [fun ($ X) X+1001 end] [4020]} '==' [5021]}
{Test {ListApply [fun ($ X) X+1 end fun ($ X) X+2 end] [10 20]} '==' [11 22]}
{Test {ListApply
  local AddC = fun ($ Y) fun ($ X) X+Y end end in
    Map [1 2 3 4 5 2 27 999 6] AddC}
end
  '==' [10 20 30 40 50 60 70 1000 10]}
{Test {ListApply [fun ($ X) bread#X#bread end fun ($ X) pita#X#pita end] [turkey humus]}
  '==' [bread#turkey#bread pita#humus#pita]}
{DoneTesting}
8. (20 points) [UseModels] This problem works with the type \langle Entry \rangle, as defined by the following grammar (where a \langle String \rangle is, as usual, a \langle List Char \rangle).

\[
\langle Entry \rangle ::= \text{directory(\langle AList \rangle)} \mid \text{file(\langle String \rangle)}
\]

\[
\langle AList \rangle ::= \langle \text{List \langle #-Pair Atom Entry \rangle} \rangle
\]

\[
\langle #-Pair Atom Entry \rangle ::= \langle \text{Atom} \# \langle Entry \rangle \rangle
\]

In Oz, write a function

Retrieves: $\langle \text{fun} \{ $ \langle Entry \rangle \langle List Atom \rangle \}: \langle Entry \rangle \rangle$

that takes an \langle Entry \rangle, Ent, and a list of atoms, Path, and returns the \langle Entry \rangle that corresponds to the given Path in Ent. An empty Path argument names Ent itself. If Ent is a file record, then the Path can only be nil, otherwise Retrieve throws an exception. If Ent is a directory record, then each element of Path names the \langle Entry \rangle associated to it by the association list (AList). When Ent is a directory, the first element of Path names an \langle Entry \rangle in the \langle AList \rangle inside Ent. Subsequent elements in Path similarly are associated to \langle Entry \rangle values by association lists in directories within the \langle Entry \rangle value associated to the previous elements in Path. For example, in the test of \{Retrieve Users [mydir mine]\} below, the atom mydir in the Path argument [mydir mine] names the directory (MyD) that holds the file record named mine.

\[
\text{\{\text{StartTesting \ 'RetrieveTest'}\}}
\]

\[
\text{\{Test \{Retrieve file("only needs to work on a file when the path is nil") nil\} \ '==\ file("only needs to work on a file when the path is nil")\}}
\]

\[
\text{\{Test \ try _ = \{Retrieve file("otherwise throw an exception") \[some path\]} \ false}
\]

\[
\text{\quad catch _ then true end \ % returns true if expected exception is caught \ '==\ true\}
\]

\[
\text{local MyD = \text{directory(\[mine#file("my file 1") metoo#file("my file 2")\]) in}}
\]

\[
\text{\{Test \{Retrieve MyD nil\} \ '==\ MyD\}}
\]

\[
\text{\{Test \{Retrieve MyD [mine]\} \ '==\ file("my file 1")\}}
\]

\[
\text{\{Test \{Retrieve MyD [metoo]\} \ '==\ file("my file 2")\}}
\]

\[
\text{\{Test \ try _ = \{Retrieve MyD [notfound]\} \ false}
\]

\[
\text{\quad catch _ then true end \ % returns true if expected exception caught \ '==\ true\}
\]

\[
\text{local Users = \text{directory(\[mydir#MyD yourdir#directory(\[yours#file("your file 1") youtoo#file("your file 2")\])\]) in}}
\]

\[
\text{\{Test \{Retrieve Users [mydir]\} \ '==\ MyD\}}
\]

\[
\text{\{Test \{Retrieve Users [mydir mine]\} \ '==\ file("my file 1")\}} \ % \text{This test is described above}
\]

\[
\text{\{Test \{Retrieve Users [mydir metoo]\} \ '==\ file("my file 2")\}}
\]

\[
\text{\{Test \{Retrieve Users [yourdir yours]\} \ '==\ file("your file 1")\}}
\]

\[
\text{\{Test \{Retrieve Users [yourdir youtoo]\} \ '==\ file("your file 2")\}}
\]

\[
\text{local Home = \text{directory(\[users#Users lib#directory(\[cpp#file("STL") c#file("stdio")\])\]) in}}
\]

\[
\text{\{Test \{Retrieve Home [users mydir mine]\} \ '==\ file("my file 1")\}}
\]

\[
\text{\{Test \{Retrieve Home [users yourdir yours]\} \ '==\ file("your file 1")\}}
\]

\[
\text{\{Test \{Retrieve Home [lib cpp]\} \ '==\ file("STL")\}}
\]

\[
\text{\{Test \{Retrieve Home [users]\} \ '==\ Users\}}
\]

\end
\end
\text{\{\text{DoneTesting}\}}

There is room for your answer on the next page.
Please put your answer to the Retrieve problem below.