Universidad de los Andes

ISIS-4217 Paradigmas de programación
Mozart
Date: August 8, 2025

Hand-in each of the following tasks on an independent `.oz` file. Make sure of following the functions specification as well as the file name conventions. Take into account, your code will be machine graded, if it does not follow the specification, it will not be graded. Additionally, make sure to test your code before handing it in, if your code does not compile or run, it will be graded as 0.

Task 1. Write a program that, given a list, calculates the sum of the elements in odd positions of the list, and the product of the elements in even positions of the list. The result should be map, where the first position corresponds to the calculated the product, and the second position corresponds to the calculated sum.

Hand-in a file `list.oz` that implements, at least, the following:

```
1 %% OddSumEvenProduct
2 %% Input: A list of integers
3 %% Output: A tuple where
4 %%   - the first element is the product of elements at even
    positions,
5 %%   - the second element is the sum of elements at odd positions
    .
6 fun {OddSumEvenProduct L}
7 %% Your code here
8 end
```

Task 2. Write a program that given two list representations of polynomials, returns the addition of the polynomials as a single list. Lists are commonly used to represent polynomials, where each position of the list contains the coeficient of the corresponding polynomial degree. From left to right, the most significant degree until reaching the constant factor. For example, the list `[3 1 0 5 2]` represents the polynomial of degree 4 $3x^4 + x^3 + 5x + 2$.

Hand-in a file `poly.oz` that implements, at least, the following:

```
%% AddPolynomials
%% Input: Two lists of integers representing polynomials
%% Output: A list of integers representing the sum of the
   polynomials
%% Examples
%%  {AddPolynomials [1 ~2] [4 3 2 1]} = [4 3 3 ~1]
%%  {AddPolynomials [3 0 0 4 1 ~5] [0 2 ~1 1 4 10]} = [3 2 ~1 5 5
    5]

fun {AddPolynomials P1 P2}
%% Your code here
end
```

Task 3. Implement a binary tree structure and the algorithms to build a tree from its orderings (inorder + postorder and inorder + preorder). To test this program the functions `inorderPreorder2BT` and `inorderPostorder2BT` receive two lists (containing numbers, characters, or a combination of them), the corresponding orderings of a tree, and return a binary tree build from the two lists.

Tree Structure:

A binary tree is defined recursively using the following representation:

- `nil` represents an empty tree.
- `tree(Value Left Right)` represents a non-empty tree, where:
  - `Value` is any Oz value (number, character, atom, etc.),
  - `Left` is the left subtree (also a binary tree),
  - `Right` is the right subtree (also a binary tree).

Example trees:

```
1  nil
2  tree(a nil nil)
3  tree(1 tree(2 nil nil) tree(3 nil nil))
```

Hand-in a file `tree.oz` that implements, at least, the following:

```
1  %% inorderPreorder2BT
2  %% Input: Two lists representing the inorder and preorder
       traversals of a binary tree
3  %% Output: A binary tree built from the traversals
4  fun {inorderPreorder2BT In Pre}
5  %% Your code here
6  end

8  %% inorderPostorder2BT
9  %% Input: Two lists representing the inorder and postorder
       traversals of a binary tree
10 %% Output: A binary tree built from the traversals
11 fun {inorderPostorder2BT In Post}
12 %% Your code here
13 end
```

Task 4. Define a program to calculate the integral of a function f on a given interval $[a, b]$ using Simpson's rule of integration. Simposon's rule of integration is defined as

$$\frac{h}{3}\left[y_0 + 4y_1 + 2y_2 + 4y_3 + 2y_4 + \ldots 2y_{n-2} + 4y_{n-1} + y_n\right]$$

where $h = \dfrac{b-a}{n}$ and $y_k = f(a + kh)$ with $0 \leq k \leq n$. n is a given integer, increasing the size of n increases the accuracy of the integral.

Function Specification:

Implement the function `integral`, which receives:

- A unary function f (i.e., a function that takes a single numerical argument),
- Two real numbers a and b (with $a < b$), the limits of integration,
- An even positive integer n, the number of subintervals.

The result should be a single floating-point number: the approximate value of the definite integral of f over $[a, b]$ using Simpson's rule with precision n.

Hand-in a file `integral.oz` that implements, at least, the following:

```
1 %% integral
2 %% Input: A function F, real numbers A and B with A < B, and an
     even integer N > 0
3 %% Output: A real number approximating the definite integral of F
      over [A, B] using Simpson's rule
4 fun {integral F A B N}
5 %% Your code here
6 end
```

Task 5. Define a procedure to determine if two records are `equal`, `equivalent`, `subsimilar`, or `different`. Given two records, the procedure must return the corresponding atom to their category. Two records are:

equal if their names, cardinality, and items all correspond to each other

equivalent if their names and cardinality correspond, but at least one of their items have different values

subsimilar if one of the records is contained in the other

different in any other case

Hand-in a file `recordR.oz` that implements, at least, the following:

```
%% recordRelation
%% Input: Two records R1 and R2
%% Output: One of the atoms 'equal', 'equivalent', 'subsimilar',
   or 'different',
%%         describing the relationship between the two records.
fun {recordRelation R1 R2}
%% Your code here
end
```