TDT4165 Programming Languages Assignment 1

Martin Hegnum Johannessen

August 27, 2024

Task 1: Hello World!

Figure 1 displays the following code: Show 'Hello World' after choosing "Oz/Feed Buffer" to feed the code. The output from Show is displayed in the "Oz Emulator", displayed in Figure 2. buffer called "Oz Emulator". Figure 3 displays the executed code "Browse 'Hello World!'".

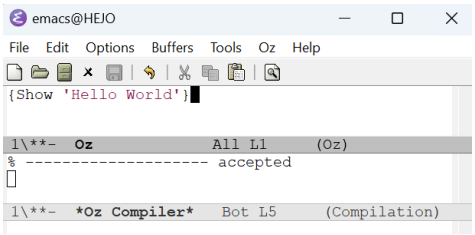

Figure 1: Oz Compiler

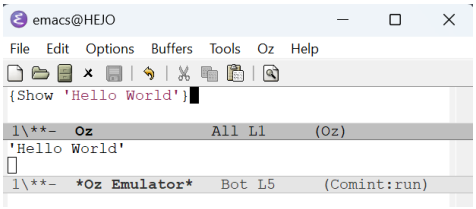

Figure 2: Oz Emulator

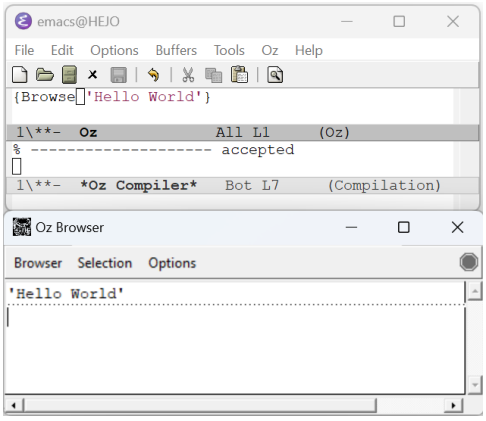

Figure 3: Oz Browser

Task 2: Using other text editors

Figure 4 displays the "insert" command being used to retrieve the oz file made in the previous task (Figure 3).

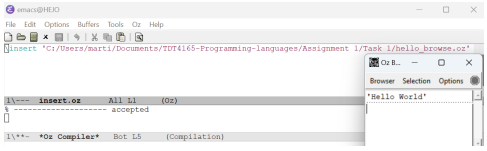

Figure 4: insert.oz

Task 3: Variables

A The code that was provided in the task description has been rewritten to assign value to two variables and calculate X from these instead of calculating X directly. Figure 5 illustrates this.

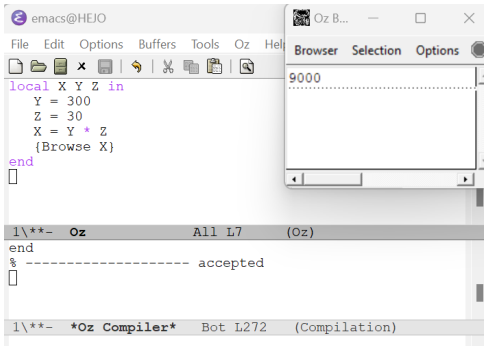

Figure 5: Local variables

B showInfo prints Y after it's assigned. It does so because of dataflow synchronization in Oz, which pauses the thread until Y is bound. Variables in Oz are logic variables, therefore they can be created without an initial value and bound at a later point.

This behavior is useful for concurrent programming, allowing natural synchronization based on when data becomes available. The statement Y = X binds Y to the value of X, making Y another reference to "This is a string". Figure 6 displays the executed code example.

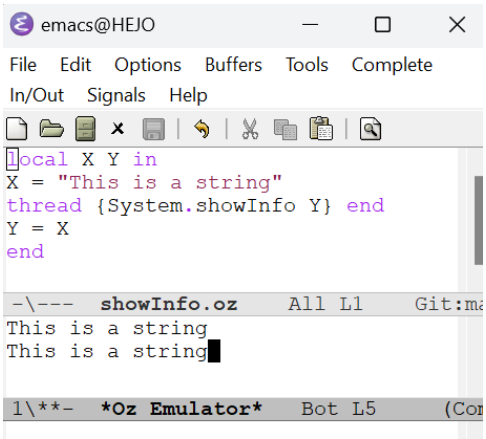

Figure 6: showInfo

Task 4: Functions and procedures

A Figure 7 illustrates a function Max Number1 Number2 that returns the maximum of Number1 and Number2.

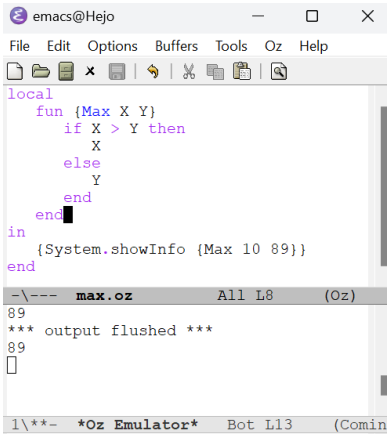

Figure 7: max.oz

B Figure 8 displays the procedure PrintGreater Number1 Number2 that prints the maximum value of the arguments.

Task 5: Variables II

Figure 9 showcase the procedure Circle R that calculates area, diameter and circumference of a circle with radius R.

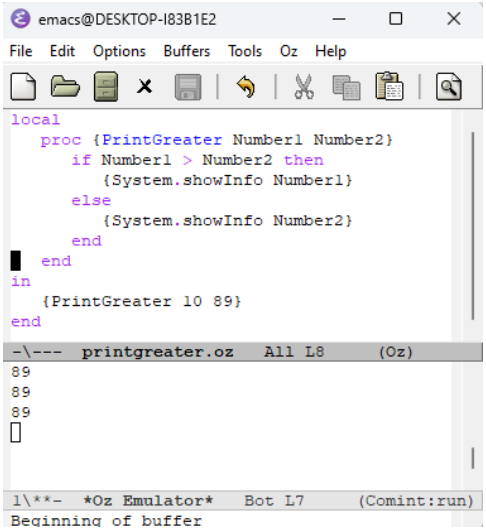

Figure 8: PrintGreater

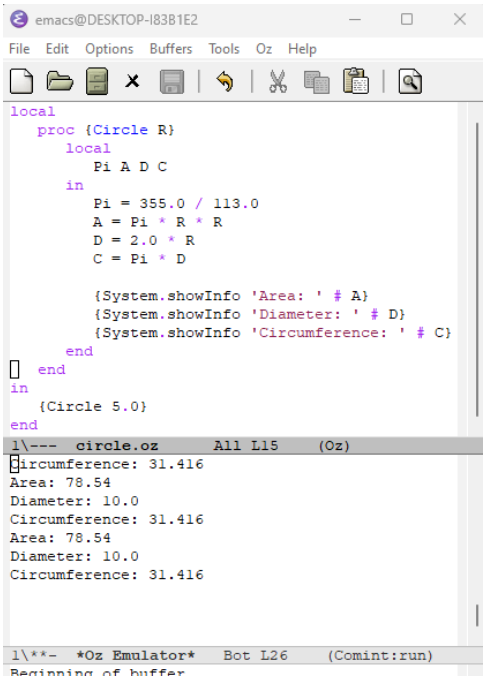

Figure 9: circle.oz

Task 6: Recursion

Figure 10 demonstrate a function Factorial N that calculates the factorial of any natural nuber using recursion.

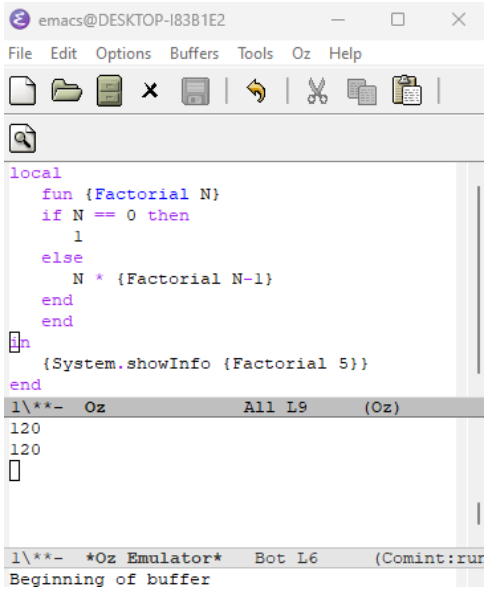

Figure 10: factorial.oz

Task 7: Lists

A Figured out here that I may use declare instead on local, making the code structured easier. Figure 11 illustrates the implementation of Length List, returning the element count of List.

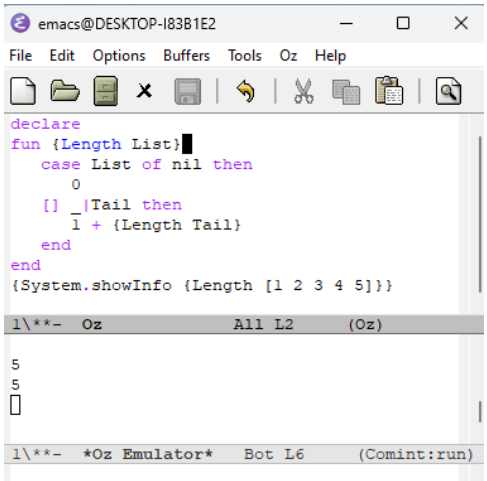

Figure 11: length_list.oz

B The Take List Count function displayed in Figure 12 recursively extracts the first Count elements from a list by reducing the count and taking the

head element until either the list is empty or the count reaches zero. The entire list is returned if Count exceeds the length of the list.

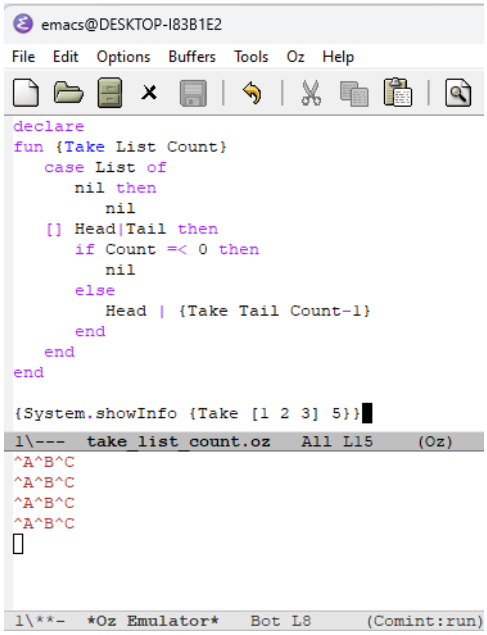

Figure 12: take_list_count.oz

C Figure 13 displays the implemented Drop List Count function that recursively removes the first Count elements from the list by skipping the head element and continuing with the tail.

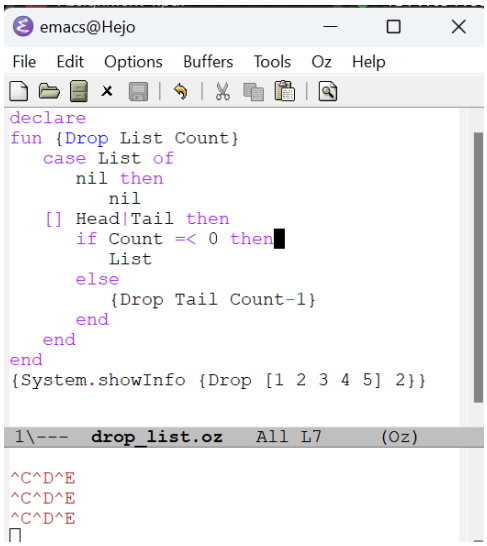

Figure 13: drop_list.oz

D Figure 14 shows the implemented Append List1 List2 function that recursively appends the elements of List1 to List2 by traversing List1 and adding its head to the result, then recursively appending the rest of the tail to List2.

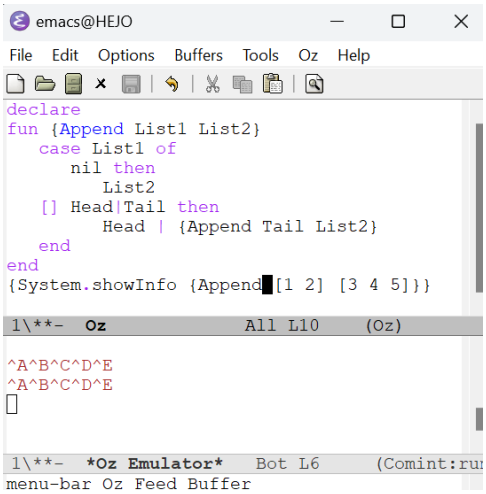

Figure 14: append_list.oz

E The Member List Element function recursively checks if the Element is in the List. I stumbled on some issues, and therefore ended up defining the function call as a Bool. Figure 15 displays the implemented function.

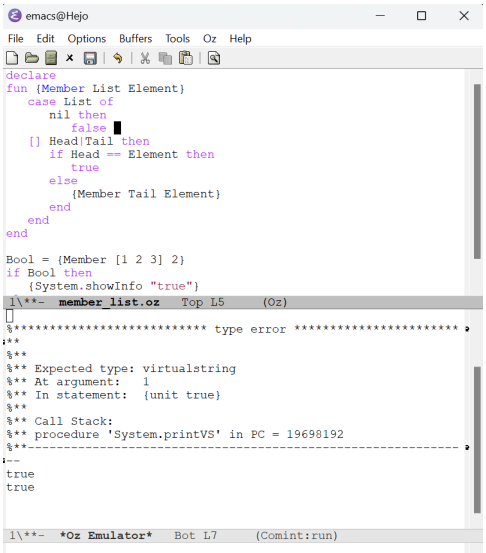

Figure 15: member_list.oz

F The Position List Element function uses a helper function PositionAux

that tracks the current position Pos as it traverses the list. It starts the search with position 1. Figure 16 displays the code implemented.

Figure 17 shows the functions in list.oz being executed. The only function with a flaw is the Member function that I haven't managed to work without the work-around described in **E**.

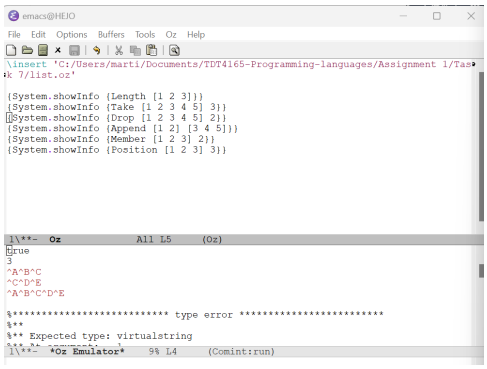

Figure 16: list.oz

Task 8: Lists II

A The function use the Oz list constructor |, which adds an element to the front of the list. As Figure 18 displays, the original list is returned with the new element in front.

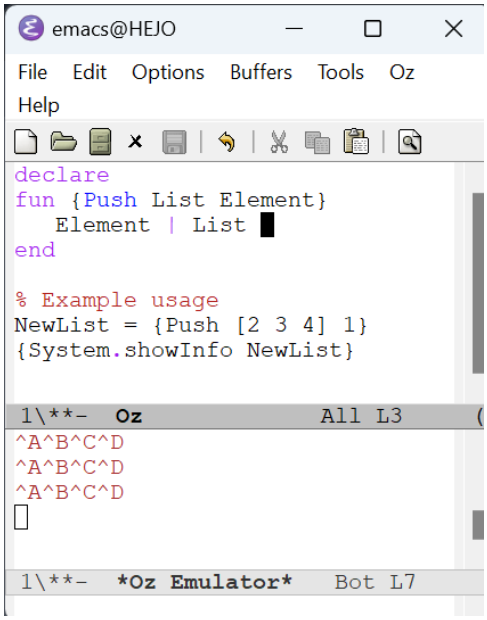

Figure 17: push_list.oz

B The Peek function uses pattern matching to check if the list is empty. Nil is returned if it is, otherwise the first element (Head) of the list is returned. Figure 19 illustrate the executed code implementation.

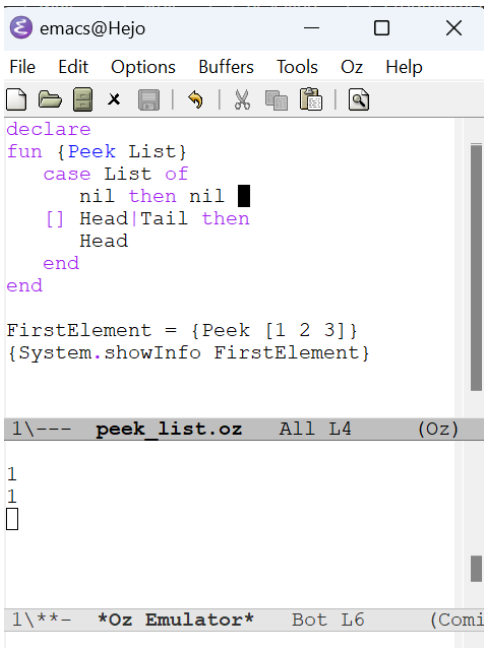

Figure 18: peek_list.oz

C The Pop function works similarly to Peek. However, instead of returning the first element, it returns the rest of the list (excluding the first element). Figure 20 showcase the executed code implementation.

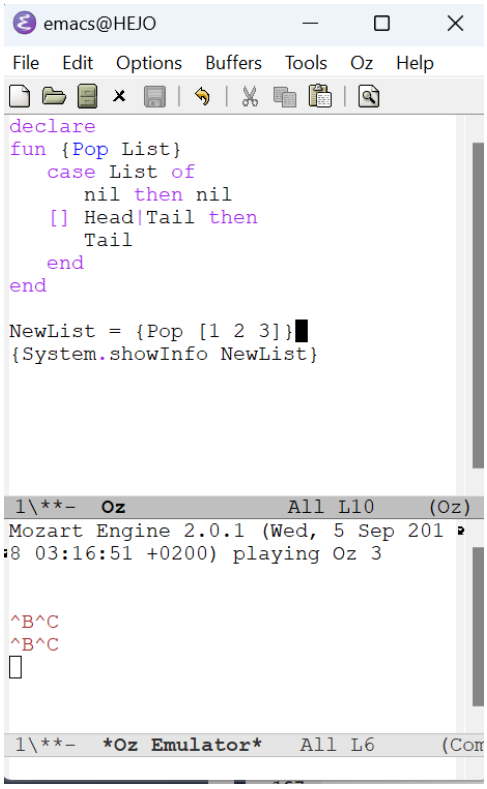

Figure 19: pop_list.oz