

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!'".

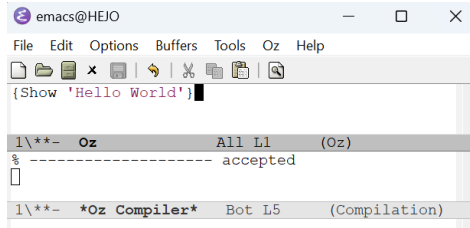


Figure 1: Oz Compiler

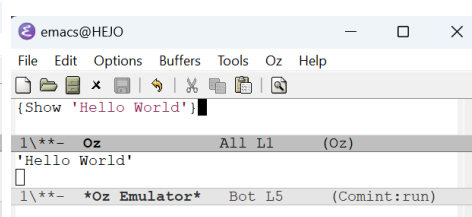


Figure 2: Oz Emulator

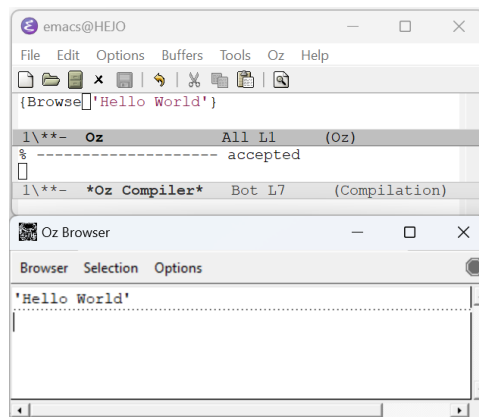


Figure 3: Oz Browser

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).

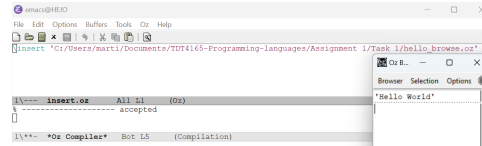


Figure 4: insert.oz

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.

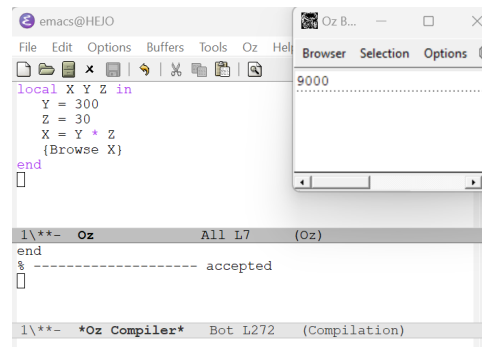


Figure 5: Local variables

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.

```

emacs@HEJO
File Edit Options Buffers Tools Complete
In/Out Signals Help
local X Y in
X = "This is a string"
thread {System.showInfo Y} end
Y = X
end

-\\--- showInfo.oz All L1 Git:ma
This is a string
This is a string

1\\*- *Oz Emulator* Bot L5 (Cor

```

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.

```

emacs@Hejo
File Edit Options Buffers Tools Oz Help
local
fun {Max X Y}
if X > Y then
X
else
Y
end
end
in
{System.showInfo {Max 10 89}}
end

-\\--- max.oz All L8 (Oz)
89
*** output flushed ***
89

1\\*- *Oz Emulator* Bot L13 (Comin

```

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.

```

emacs@DESKTOP-I83B1E2
File Edit Options Buffers Tools Oz Help

local
  proc {PrintGreater Number1 Number2}
    if Number1 > Number2 then
      {System.showInfo Number1}
    else
      {System.showInfo Number2}
    end
  end
in
  {PrintGreater 10 89}
end

-\\--- printgreater.oz All L8 (Oz)
89
89
89
[]

1\\*- *Oz Emulator* Bot L7 (Comint:run)
Beginning of buffer

```

Figure 8: PrintGreater

```

emacs@DESKTOP-I83B1E2
File Edit Options Buffers Tools Oz Help

local
  proc {Circle R}
    local
      Pi A D C
    in
      Pi = 355.0 / 113.0
      A = Pi * R * R
      D = 2.0 * R
      C = Pi * D

      {System.showInfo 'Area: ' # A}
      {System.showInfo 'Diameter: ' # D}
      {System.showInfo 'Circumference: ' # C}
    end
  end
in
  {Circle 5.0}
end

-\\--- circle.oz All L15 (Oz)
[]Circumference: 31.416
Area: 78.54
Diameter: 10.0
Circumference: 31.416
Area: 78.54
Diameter: 10.0
Circumference: 31.416

1\\*- *Oz Emulator* Bot L26 (Comint:run)
Beginning of buffer

```

Figure 9: circle.oz

Task 6: Recursion

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

```

emacs@DESKTOP-I83B1E2
File Edit Options Buffers Tools Oz Help
[Icons]
local
  fun {Factorial N}
    if N == 0 then
      1
    else
      N * {Factorial N-1}
    end
  end
end
{System.showInfo {Factorial 5}}
end
1\*- Oz All L9 (Oz)
120
120
[
1\*- *Oz Emulator* Bot L6 (Comint:run)
Beginning of buffer

```

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.

```

emacs@DESKTOP-I83B1E2
File Edit Options Buffers Tools Oz Help
[Icons]
declare
  fun {Length List}
    case List of nil then
      0
    [] _Tail then
      1 + {Length Tail}
    end
  end
end
{System.showInfo {Length [1 2 3 4 5]}}
end
1\*- Oz All L2 (Oz)
5
5
[
1\*- *Oz Emulator* Bot L6 (Comint:run)

```

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.

```

emacs@DESKTOP-I83B1E2
File Edit Options Buffers Tools Oz Help
declare
fun {Take List Count}
  case List of
    nil then
      nil
    [] Head|Tail then
      if Count <= 0 then
        nil
      else
        Head | {Take Tail Count-1}
      end
    end
  end
end

{System.showInfo {Take [1 2 3] 5}}
1\--- take_list_count.oz All L15 (Oz)
^A^B^C
^A^B^C
^A^B^C
^A^B^C
□
1\*- Oz Emulator* Bot L8 (Comint:run)

```

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.

```

emacs@Hejo
File Edit Options Buffers Tools Oz Help
declare
fun {Drop List Count}
  case List of
    nil then
      nil
    [] Head|Tail then
      if Count <= 0 then
        List
      else
        {Drop Tail Count-1}
      end
    end
  end
end

{System.showInfo {Drop [1 2 3 4 5] 2}}
1\--- drop_list.oz All L7 (Oz)
^C^D^E
^C^D^E
^C^D^E
□

```

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.

```

emacs@HEJO
File Edit Options Buffers Tools Oz Help
declare
fun {Append List1 List2}
  case List1 of
    nil then
      List2
    [] Head|Tail then
      Head | {Append Tail List2}
    end
  end
end
{System.showInfo {Append [1 2] [3 4 5]}}

1\*- Oz All L10 (Oz)

^A^B^C^D^E
^A^B^C^D^E
[]

1\*- *Oz Emulator* Bot L6 (Comint:run)
menu-bar Oz Feed Buffer

```

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.

```

emacs@Hejo
File Edit Options Buffers Tools Oz Help
declare
fun {Member List Element}
  case List of
    nil then
      false
    [] Head|Tail then
      if Head == Element then
        true
      else
        {Member Tail Element}
      end
    end
  end
end
Bool = {Member [1 2 3] 2}
if Bool then
  {System.showInfo "true"}
end

1\*- member_list.oz Top L5 (Oz)

%***** type error *****
%***
%*** Expected type: virtualstring
%*** At argument: 1
%*** In statement: {unit true}
%***
%*** Call Stack:
%*** procedure 'System.printVS' in PC = 19698192
%***
true
true

1\*- *Oz Emulator* Bot L7 (Comint:run)

```

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**.

```

emacs@HEJO
File Edit Options Buffers Tools Oz Help
\insert 'C:/Users/marti/Documents/TDT4165-Programming-languages/Assignment 1/Tas
rk 7/list.oz'
(System.showInfo (Length [1 2 3]))
(System.showInfo (Take [1 2 3 4 5] 3))
[System.showInfo (Drop [1 2 3 4 5] 2)]
(System.showInfo (Append [1 2] [3 4 5]))
(System.showInfo (Member [1 2 3] 2))
(System.showInfo (Position [1 2 3] 3))

1\*- Oz All L5 (Oz)
true
3
^A^B^C
^C^D^E
^A^B^C^D^E
$***** type error *****
$** Expected type: virtualstring
1\*- *Oz Emulator* 9% L4 (Comint:run)

```

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.

```

emacs@HEJO
File Edit Options Buffers Tools Oz
Help
declare
fun {Push List Element}
  Element | List
end

% Example usage
NewList = {Push [2 3 4] 1}
{System.showInfo NewList}

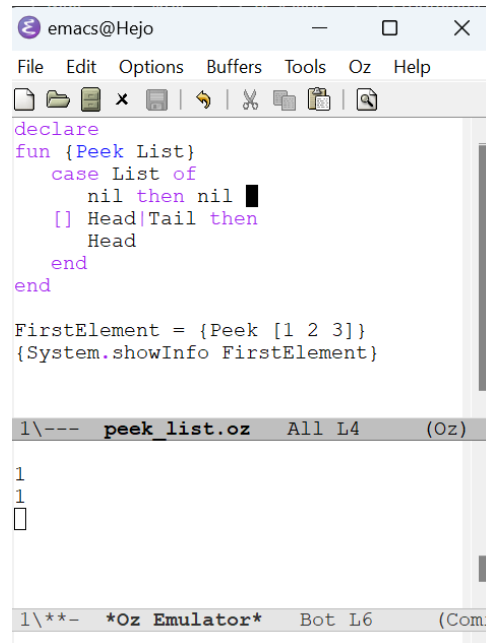
1\*- Oz All L3 (
^A^B^C^D
^A^B^C^D
^A^B^C^D
[]

1\*- *Oz Emulator* Bot L7

```

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.



```

emacs@Hejo
File Edit Options Buffers Tools Oz Help
declare
fun {Peek List}
  case List of
    nil then nil
    [] Head|Tail then
      Head
    end
  end
end

FirstElement = {Peek [1 2 3]}
{System.showInfo FirstElement}

1\--- peek_list.oz All L4 (Oz)

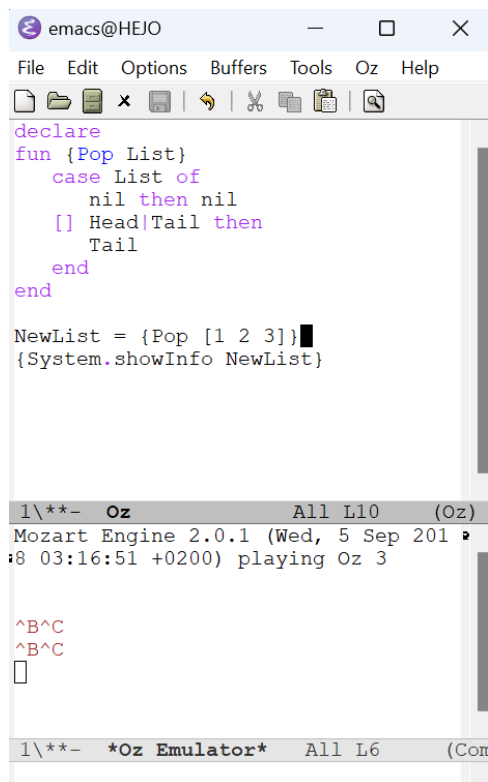
1
1
[]

1\*- *Oz Emulator* Bot L6 (Comi

```

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.



```
emacs@HEJO
File Edit Options Buffers Tools Oz Help
declare
fun {Pop List}
  case List of
    nil then nil
    [] Head|Tail then
      Tail
    end
  end
end

NewList = {Pop [1 2 3]}
{System.showInfo NewList}

1\**- Oz All L10 (Oz)
Mozart Engine 2.0.1 (Wed, 5 Sep 201
:8 03:16:51 +0200) playing Oz 3

^B^C
^B^C
[]

1\**- *Oz Emulator* All L6 (Com
```

Figure 19: pop_list.oz