PURPOSE

- Oz is a very high level language that incorporates all of the familiar programming paradigms of other common languages such as object-oriented design and functional programming along with some paradigms new to us like constraint programming, concurrent programming, and distributed programming.
- The inclusion of these paradigms makes complicated projects trivial to build.
- Oz’s constraint programming and distributed programming paradigms allows programs to run even after failure.
- Concurrent programming allows for algorithms that interact with each other effecting the results in real time.

SYNTAX

Basic form of a block of Mozart code:
local
  <local variables go here>
  <local procedures go here>
in
  <main code using variables and procedures goes here>
end

Basic form of a procedure:
proc {<proc name> <any # of arguments>}
  <main code using arguments goes here>
end
proc {Add a1 a2 sum}
  sum = a1 + a2
end

- Variables in Oz are declared without a type until they are assigned a value.
local A B C D in
A = 5 ← Here A only becomes an int when it is assigned 5
B = &t ← Here B became the character t
end

printing
{Show A} ← prints ‘5’ to terminal
{Show ‘my string’} ← prints the words ‘my string’ to terminal
{Show [‘my age is ’ CurrentAge]} ← brackets must be used when printing a string with a variable
conditional statements exist in Mozart as well, and have the following syntax

if <condition> then
  <code to execute>
else
  <code to execute>
end

threads allow Mozart programs to have multiple procedures running at once. When a thread is executed in a program, the program does not wait for a thread to finish executing before running the next piece of code.

Syntax: Thread { <block of code> } end

thread modifier (the delay) – You can halt the progress of a thread by using the delay procedure

{Delay N} N is the number of milliseconds.

RUNNING OZ

• Install Mozart on to Linux account by first downloading Mozart 3 at
  http://downloads.sourceforge.net/project/mozart-oz/v2.0.0-alpha.0/mozart2-2.0.0-alpha.0%2Bbuild.4105.5c06ced-x86_64-linux.tar.gz?r=https%3A%2F%2Fsourceforge.net%2Fprojects%2Fmozart-oz%2Ffiles%2Fv2.0.0-alpha.0%2F&ts=1478047283&use_mirror=pilotfiber then extracting the files to the directory of your choice

• From here on: C = Ctrl, M = Alt(for windows) M = option(for mac)

• Then type in the command line:
  <path to the Mozart directory>/bin/oz <your mozart filename here>

• Once code is written, you must copy it to the oz buffer to run it. First save it with C-x C-s then copy it using: C-x h (to select all), C-w (to cut), C-y (to paste) Then navigate to Oz → Buffers → Oz and paste here again (C-y).

• Mozart allows you to run segments of your code instead of the entire file. Move your cursor to a paragraph of code and type M-C-x to compile and run. Other execution methods are available in the OZ menu.

• The compiler/program output is the area below your code. To switch between the compiler output and program output navigate to OZ, select show/hide, select compiler or emulator.

• If you want to make any more changes to your code you can do so here, but remember to copy the changes back to your original buffer and save it there.
ASSIGNMENT

Make a simulation using Oz threads. You want to find the fastest way home from work. There are three different methods of traveling. You can walk, take the train, or drive. If you take the train, you get off at the fifth stop. It takes 10 minutes to travel between train stops. If you take your car you would have to drive 11 miles. It would take you 3 minutes to drive a mile. If you walk, you would have to walk 45 blocks. It takes you one minute to walk a block. Since you hate math so much you decide to build a concurrent simulation in Oz to find out which method is fastest.

- Use three threads to represent the three different traveling methods.
- You can simulate the real world minutes as seconds in your code.
- Use the delay procedure to control the speed of your simulated vehicles.
- Each thread should print something out when a train stop/mile/block is reached.
- A different message should be printed when a vehicle reaches home.
- After a vehicle reaches home, it is not necessary to kill the thread, but it should not print anything anymore.
- This assignment will create never-ending programs! To kill a running program type f10, navigate to Oz then Halt Oz.

You output should have output similar to this (this is not necessarily correct):
block 1
block 2
block 3
mile 1
block 4
block 5
mile 2
block 6
train stop 1
block 7

You reached home by ???

??? = train/car/foot

You reached home by ???

You reached home by ???

You reached home by ???