Programming Language Concepts, CS2104
Lab/Assignment 0 (Lab Session, 3-6pm, 24th Aug 2007)
Deadline for Lab0 : 5pm 28Aug 2007 Tue (submit via IVLE)

CS2104 is a 4 credit points module (written final exam 50%, 2 midterm exams 25%, lab/tutorial assignments 25%). The module homepage is http://www.comp.nus.edu.sg/~cs2104 and IVLE. Teaching means lectures and combined tutorials/lab sessions (labs). Lectures are based of the book:

- Peter Van Roy, Seif Haridi: Concepts, Model, and Techniques of Computer Programming, The MIT Press, 2004

The main purposes of tutorials are: for self-assessment, revise material from lectures, answer questions, allow deep understanding, prepare labs assignments. Tutorials comprise simple assignments, and are good exercises for the exam. You may discuss tutorials/chapters on the IVLE discussion groups. There will be five lab assignments (please submit in time).

Overview

- This lab/assignment should be done individually.
- At the end of lab/assignment, you should have Mozart running on your computer.
- Try the examples that have been introduced in the first lecture together with some similar functions.
- Use the time available to ask questions!
- Ask your friends.
- You can also ask on IVLE's discussion group of Chapter 1.

Useful Software

- http://www.mozart-oz.org/
    - programming language: Oz
    - system: Mozart (1.3.0, released on April 15, 2004)
    - interactive system
- Requires Emacs on your computer (http://www.gnu.org/software/emacs/)
- Available from module webpage
- First tutorial will help with installation

Mozart Installation (Windows/Unix)

Details for the Windows Installation

Install Emacs and Mozart on your PC (very easily).

Details for the Unix Installation

1. Get an account on `sunfire`.
2. Add to your PATH the following new path: `/home/course/cs2104/mozart/bin`.
You can do this in two ways either (a) or (b):
(a) modify your "`.profile`" or "`.bashrc`" file such that the file will contain the following two commands:

```
        PATH=$PATH:/home/course/cs2104/mozart/bin
        export PATH
```

(b) just type the following command in the command line:

```
        export PATH=$PATH:/home/course/cs2104/mozart/bin
```

3. To run Mozart from `sunfire`, you may need `X-Window` to be installed on your Windows machine. To install `X-Window`, please use the guide from the following web page: `https://www.comp.nus.edu.sg/cf/x/index.html`

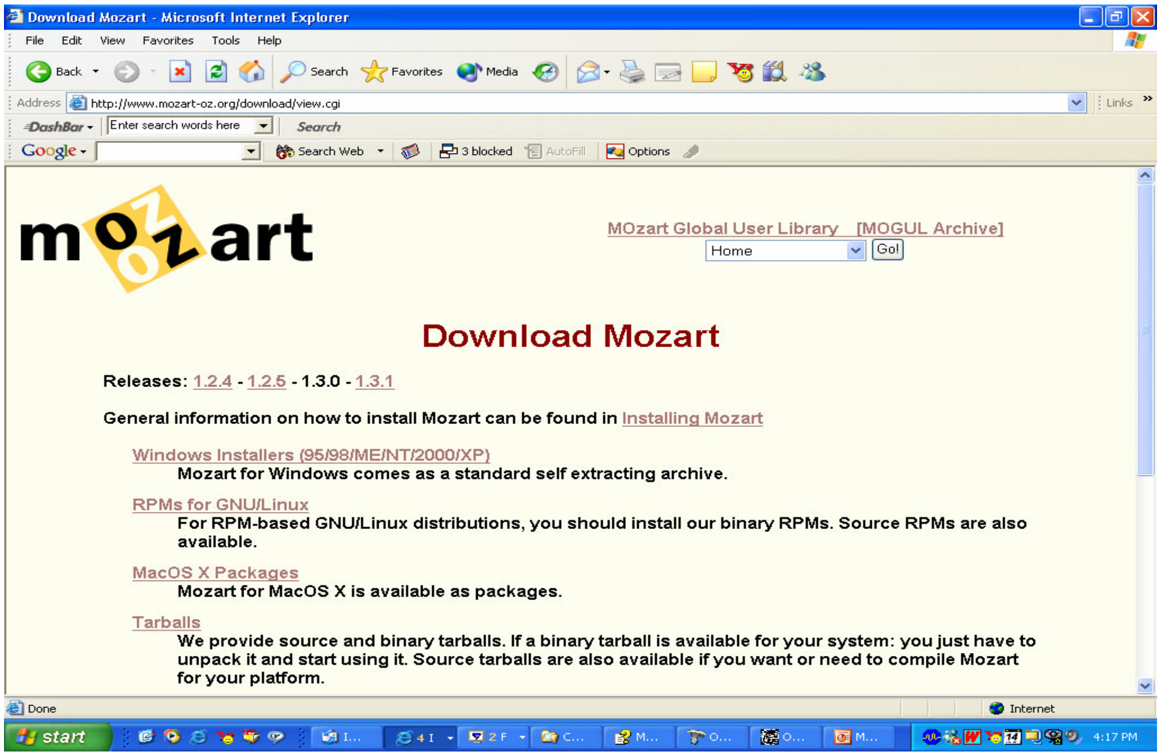
Running Mozart on sunfire

1. start `x-win32` on your windows machine.
2. connect to `sunfire` using a `ssh` client
3. type "`oz&`" in the command line.

Key Bindings

C-. C-l	Feed current line
C-. C-r	Feed selected region
C-. C-b	Feed whole buffer
C-. C-p	Feed current paragraph
C-. c	Toggle display of *Oz Compiler* buffer
C-. e	Toggle display of *Oz Emulator* buffer
C-x ‘ (i.e. Control-x backquote)	positions the transcript to make the first error message visible and moves the point, in the source buffer, to where the bug is likely to be located.
C-. n	Create a new buffer using the Oz major mode. Note that this buffer has no associated file name, so quitting Emacs will kill it without warning.
M-n	
M-p	Switch to the previous resp. next buffer in the buffer list that runs in an Oz mode. If no such buffer exists, an error is signalled.

For more details about Mozart commands, you should consult Programming Environment and Tools manual. For more details about `emacs` commands, you should consult the Emacs on-line tutorial available from the Help menu in the Emacs menu bar or an online tutorial from http://www.lib.uchicago.edu/keith/tcl-course/emacs-tutorial.html.


Download Mozart - Microsoft Internet Explorer
http://www.mozart-oz.org/download/view.cgi
MOzart Global User Library [MOGUL Archive]
Home
Go!

Download Mozart

Releases: 1.2.4 - 1.2.5 - 1.3.0 - 1.3.1

General information on how to install Mozart can be found in Installing Mozart

Windows Installers (95/98/ME/NT/2000/XP)
Mozart for Windows comes as a standard self extracting archive.

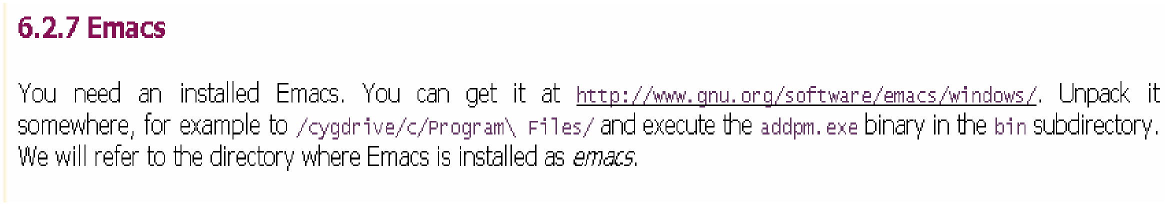
RPMs for GNU/Linux
For RPM-based GNU/Linux distributions, you should install our binary RPMs. Source RPMs are also available.

MacOS X Packages
Mozart for MacOS X is available as packages.

Tarballs
We provide source and binary tarballs. If a binary tarball is available for your system: you just have to unpack it and start using it. Source tarballs are also available if you want or need to compile Mozart for your platform.

Emacs Installation

- http://www.gnu.org/software/emacs/windows/



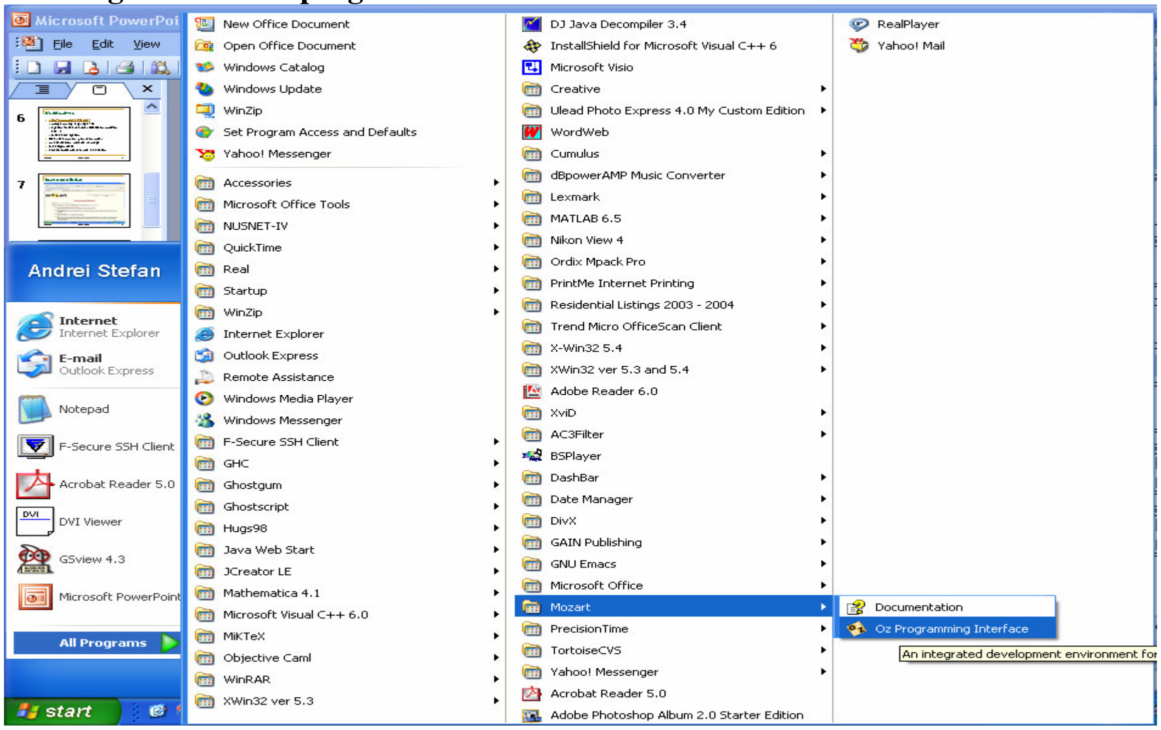
6.2.7 Emacs

You need an installed Emacs. You can get it at `http://www.gnu.org/software/emacs/windows/`. Unpack it somewhere, for example to `/cygdrive/c/Program\ Files/` and execute the `addpm.exe` binary in the `bin` subdirectory. We will refer to the directory where Emacs is installed as *emacs*.

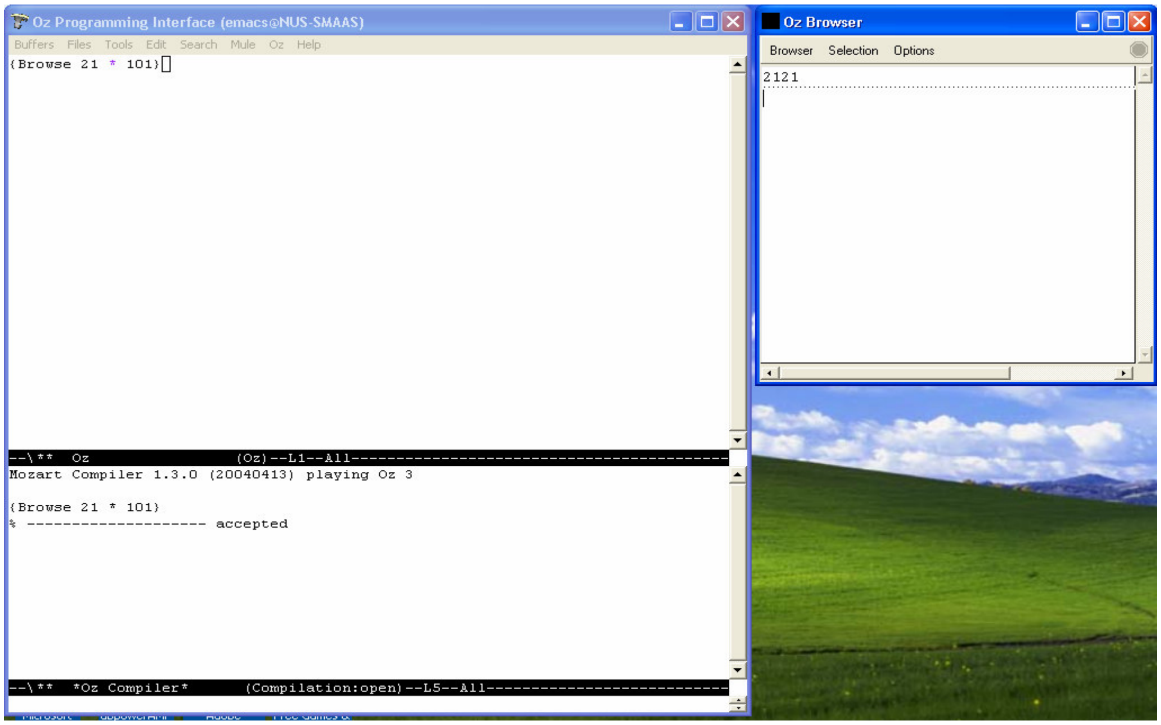
The Mozart System

- Interactive interface (the declare statement)
  - Allows introducing program fragments incrementally and execute them
  - Has a tool (Browser), which allows looking into the store using the procedure `Browse`
- `{Browse 21 * 101}` -> by selecting "Oz" panel, "Feed Line" or alternatively "C-. C-l", this will display in the Browser window the number `2121`

Running our first Oz program



The Mozart Interface


Oz Programming Interface (emacs@NUS-SMAAS)
Buffers Files Tools Edit Search Mule Oz Help
Oz Browser
Browser Selection Options

```
{Browse 21 * 101}
```

```
--\**  Oz                  (Oz)--L1--All----------------------------------------
Mozart Compiler 1.3.0 (20040413) playing Oz 3

{Browse 21 * 101}
% -------------------- accepted
```

```
--\**  *Oz Compiler*       (Compilation:open)--L5--All------------------------
```

```
2121
```

Concept of (Single-Assignment) Variable Identifier

```
declare
   X = 21
```

```
X = 22
  % raise an error
X = 21
  % do nothing
declare
X = 22
   % from now on, X will be bound to 22
```

Concept of Oz Variable Type

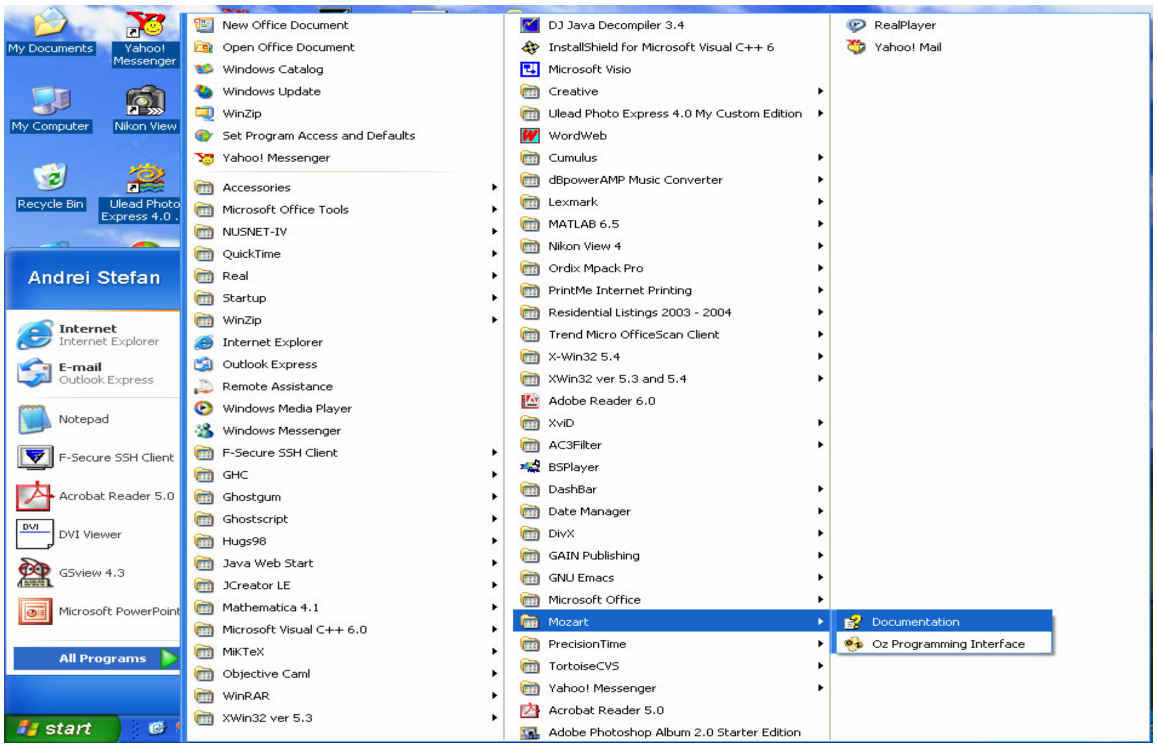
A variable type is known only after the variable is bound

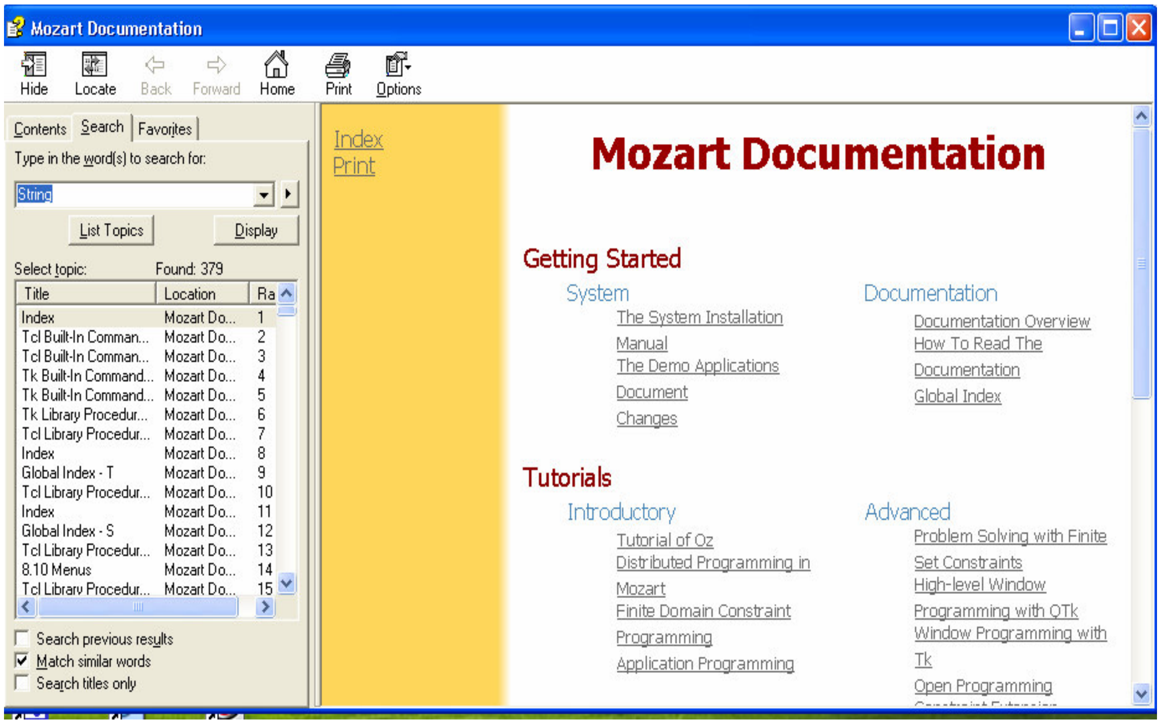
Examples:

1.
```
X < 1
X < 1.0
```

2.
```
declare X Y
X = "Oz Language"
Y = 'Oz Language'
if X == Y
  then {Browse yes}
  else {Browse no}
end
```

The Mozart Documentation





Type in the word(s) to search for:

String

Select topic: Found: 379

Title	Location	Ra
Index	Mozart Do...	1
Tcl Built-In Comman...	Mozart Do...	2
Tcl Built-In Comman...	Mozart Do...	3
Tk Built-In Command...	Mozart Do...	4
Tk Built-In Command...	Mozart Do...	5
Tk Library Procedur...	Mozart Do...	6
Tcl Library Procedur...	Mozart Do...	7
Index	Mozart Do...	8
Global Index - T	Mozart Do...	9
Tcl Library Procedur...	Mozart Do...	10
Index	Mozart Do...	11
Global Index - S	Mozart Do...	12
Tcl Library Procedur...	Mozart Do...	13
8.10 Menus	Mozart Do...	14
Tcl Library Procedur...	Mozart Do...	15

☐ Search previous results
☑ Match similar words
☐ Search titles only

Mozart Documentation

Getting Started

System
The System Installation
Manual
The Demo Applications
Document
Changes

Documentation
Documentation Overview
How To Read The
Documentation
Global Index

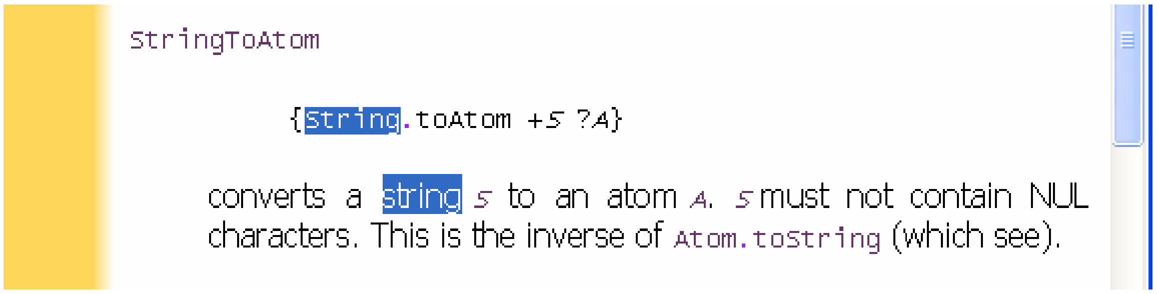
Tutorials

Introductory
Tutorial of Oz
Distributed Programming in
Mozart
Finite Domain Constraint
Programming
Application Programming

Advanced
Problem Solving with Finite
Set Constraints
High-level Window
Programming with QTk
Window Programming with
Tk
Open Programming

Concept of Oz Variable Type

```
declare X Y Z
X = "Oz Language"
Y = 'Oz Language'
{String.toAtom X Z}
if Z == Y then {Browse yes}
else {Browse no}
end
```



StringToAtom

```
{String.toAtom +S ?A}
```

converts a string *S* to an atom *A*. *S* must not contain NUL characters. This is the inverse of `Atom.toString` (which see).

Try these Functions

```
declare
fun {Minus X}
   ~X
end
{Browse {Minus 15}}
declare
fun {Max X Y}
   if X>Y then X else Y end
end
declare
```

```
X = {Max 22 18}
Y = {Max X 43}
{Browse Y}
```

Exercise 1 (Absolute Value) Write a function `Abs` that computes the absolute value of a number. This should work for both integers and real numbers.

Try Recursive Function

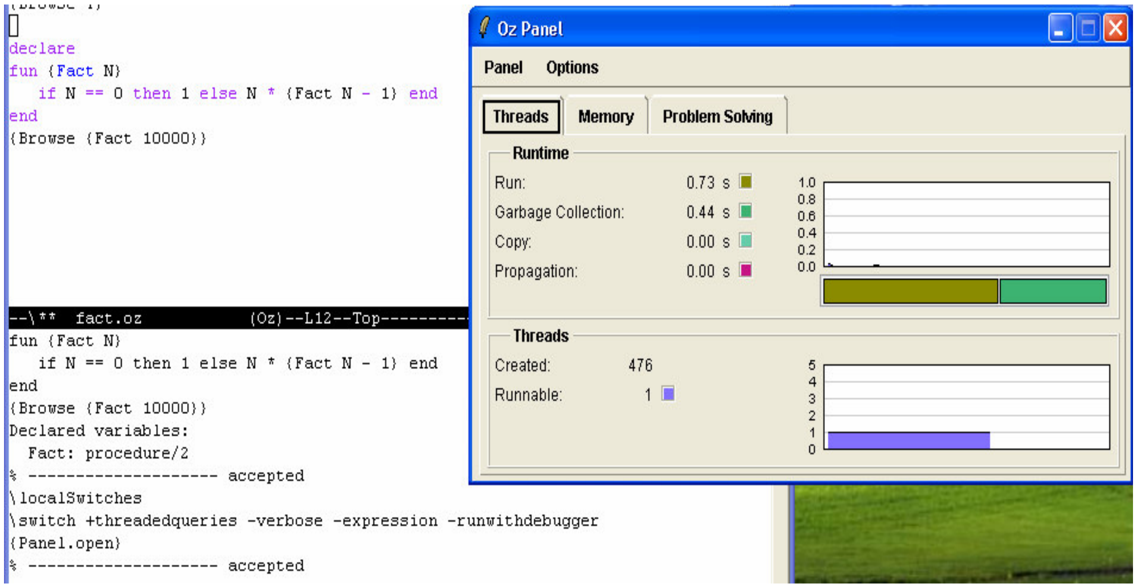
Recursive function definition

```
fun {Fact N}
   if N == 0 then 1
   else N * {Fact N-1}
   end
end
{Browse {Fact 5}}
```

Try some calls:

- `{Fact 5}`
- `{Fact 100}`
- `{Fact 10000}` Use the Oz Panel to get an idea how much memory is needed.

Oz Panel



```
declare
fun {Fact N}
   if N == 0 then 1 else N * {Fact N - 1} end
end
{Browse {Fact 10000}}
```

```
--\**  fact.oz            (Oz)--L12--Top---------
fun {Fact N}
   if N == 0 then 1 else N * {Fact N - 1} end
end
{Browse {Fact 10000}}
Declared variables:
  Fact: procedure/2
% -------------------- accepted
\localSwitches
\switch +threadedqueries -verbose -expression -runwithdebugger
{Panel.open}
% -------------------- accepted
```

```
Runtime
Run:                 0.73 s
Garbage Collection:  0.44 s
Copy:                0.00 s
Propagation:         0.00 s

Threads
Created:      476
Runnable:       1
```

Try Fibonacci Example

The execution time of a program as a function of input size, up to a constant factor, is called the program's **time complexity**.

```
declare
fun {Fibo N}
   case N of
      1 then 1
   [] 2 then 1
   [] M then {Fibo (M-1)} + {Fibo (M-2)}
   end
end
{Browse {Fibo 100}}
```

The time complexity of `{Fibo N}` is proportional to 2^N.

Try Efficient Fibonacci Example

```
declare
fun {FiboTwo N A1 A2}
   case N of
      1 then A1
   [] 2 then A2
   [] M then {FiboTwo (M-1) A2 (A1+A2)}
   end
end
{Browse {FiboTwo 100 1 1}}
```

The time complexity of `{Fibo N}` is proportional to `N`.

Exercise 2 (Power) Compute n^m where n is an integer and m is a natural number.
Hint: Use the following inductive definition of nm:

- $n^0 = 1$
- $n^m = n * n^{m-1}$

Write a function `Pow` as follows:

```
declare
fun {Pow N M}
  if  ... then
   ...
  else
   ...
  end
end
```

Exercise 3 (Maximum Recursively) Compute the maximum of two natural numbers, knowing that the only allowed test with a conditional is the test whether a number is zero (that is, `if N==0 then … else … end`).
Hint: Facts about the maximum ($n \geq 0$ and $m \geq 0$):

- max(n, m)=m, if n=0.
- max(n, m)=n, if m=0.
- max(n, m)=1 + max(n-1, m-1), otherwise.