# Programming Language Concepts, cs2104 Lecture 01 (2003-08-15)

**Seif Haridi**
Department of Computer Science, NUS
haridi@comp.nus.edu.sg



## Overview

- Organization
- Course overview
- Getting started

# Organization



# Programming Language Concepts cs2401

- cs2104 is a 4 credit points module
  - written final exam 40%
  - Midterm exam 20%
  - Lab/tutorial assignments 40%
- Module homepage
  - http://www.comp.nus.edu.sg/~cs2104
  - IVLE
- Teaching
  - lectures
  - Combined tutorials/lab sessions (labs)

## Team

- Course responsible [Lectures]
  Seif Haridi
  haridi@comp.nus.edu.sg
  cs2104@comp.nus.edu.sg
- Teaching assistants [Tutorials/Labs]



## Lectures

- Held by me
- One special lectures
  - guest lecture by Joe Armstrong on Concurrency Oriented Programming

## Lecture Structure

- Reminder of last lecture

- Overview
- Content
- Summary

- Reading suggestions



## Material

- Lecture based on book
  - Concepts, Model, and Techniques of Computer Programming
  - Peter Van Roy, Seif Haridi
- Book coming out this fall by MIT-Press
- Copies could be made available at the CO-OP
  - around 580 pages
  - price
  - Electronic version on module webpage

## Reading Suggestions

- Will be available on webpage (Lectures)
- Initially
  - Browse as you like
  - Abstract and Preface [casual]
  - Introduction 1.1 – 1.15 and 1.19 [careful] [try examples]
  - Appendix A.1 for Oz Development Environment [as you need]



## Tutorials/Labs

- Purpose
  - for self-assessment
  - rehearse material from lectures
  - answer questions
  - deepen understanding
  - prepare labs assignments
  - save your time!
- Simple assignments
  - done by teaching assistants/students
- Good exercises for the exam…

## Tutorial: Example

- First/Second tutorial session exercises list processing
- Is needed in first assignment
- Attend the tutorials, save some work!
- You can discuss tutorial/chapters on the IVLE discussion groups



## Assignments

- There will be five lab assignments
- One discussion group per assignment
  - discuss lab assignments
  - answer questions to assignments
  - solutions to be submitted through IVLE
  - there is a deadline for each assignment
- Code of conduct
  - no cheating
  - pairs are allowed, state your partner in the answer
  - for details, see webpage

## Assignments

- Please submit in time!
- Please submit in time!
- Please submit in time!
- Please submit in time!
- Please submit in time!
- Please submit in time!
- Please submit in time!
- Please submit in time!



## Lab Assignments: Overview

- First assignment: train shunting
  - small problem solving puzzle
  - lists, recursion, problem modeling
  - see course webpage
- Planned (maybe we will have better ideas)
  - black and white picture compression

## Used Software

- http://www.mozart-oz.org/
  - programming language: Oz
  - system: Mozart
  - interactive system
- Requires Emacs on your computer
- Available from module webpage
- Install your self. Ask your friends
- First tutorial will help with installation

**Bring your computer!**
**Already download Mozart!**



## Lab and Tutorial Groups

- Assignment done via IVLE
  - is everybody subscribed to IVLE?

# No Mailing Lists (use IVLE)

- Only on exceptional cases
  cs2104@comp.nus.edu.sg

- Questions
  - There is a discussion group for each book chapter/lectures
  - There is a discussion group for each lab assignment
- Tutorials and lab assignments
  - use lab sessions only
  - use tutorials only
  - no email
- Submit your assignments using the corresponding Workbin (IVLE)



# Course Board

- Please establish as soon as possible
- My first course taught @NUS
  - you know customs better than me
  - comments and criticism is very welcome
  - do not wait with comments and criticism until end
- Elect/Choose 4 students to form a course committee
- Inform me by email
  - cs2104@comp.nus.edu.sg

## Feedback in General

- Approach course-board
- Approach the teaching assistant
- Approach me directly, but please arrange for appointment



## Questions and Using Brakes!

- Please do ask questions during the lectures
  - repeat an explanation
  - give better explanation
  - for an example?
- Please say when things go too fast!
- Please say when things go too slow!

# Course Overview



## Aim

- Knowledge and skills in
  - Programming languages Concepts
  - Corresponding programming techniques
- Acquaintance with
  - central concepts in computer science related to programming (languages)
  - Focus on concepts and not on a particular language

## Programming Concepts, Models and Techniques

- Declarative programming
- Concurrent programming
- Programming with explicit state
- Object-oriented programming



## Programming Models

- Combine
  - data types together with operations on them
  - language to write programs
- Each model supports different techniques
- Sometimes also “Programming Paradigm”
  - buzzword-kind of style

## Programming Languages

- Different programming languages support different programming models

- We want to study many models!
- Do we have to study many languages?
  - Learn syntax…
  - Learn semantics…
  - Learn the software…



## Pragmatic Way Out

- Just one programming language
  - which supports several programming models
  - sometimes “multi-paradigm” programming language

- Our choice here is Oz
  - supports computation models of our interest

- The focus is on
  - the programming models!
  - techniques and concepts!
  - **not** the particular language!

## Presenting Computation Models

- Based on kernel language
  - simple language
  - small number of **significant** language constructs
  - goal: simple, minimal
- Richer language on top of kernel language
  - expressed in terms of kernel language
  - goal: support common programming tasks



## Our Incremental Approach

- Start with one kernel language
  - declarative programming model
- Add constructs
  - yields the other programming models
  - very few constructs!
  - very little to understand!

## Models You Know (CS1102)

- Java supports
  - programming with state
  - object-oriented programming
- Clearly, these two models matter!



## Why the Other Models?

- Models new to you
  - declarative programming model
  - concurrent programming model
- Do they matter?
  - yes, of course
  - relevant to **complex** systems
  - will become clear during course
  - guest lecture will discuss practical relevance

## Our First Model

- Declarative programming model
- Approach
  - informal introduction to important concepts and techniques
  - introducing the underlying kernel language
  - formal semantics based on abstract machine
  - in depth study of programming techniques



## Declarative Programming Model

- Ideal of declarative programming
  - say **what** you want to compute
  - let computer find **how** to compute it
- More pragmatically
  - let the computer provide more support
  - free the programmer from some burden

## Properties of Declarative Model

- Computations are evaluating functions on data structures
- Widely used
  - functional languages: LISP, Scheme, ML, Haskell, …
  - logic languages: Prolog, Mercury, …
  - representation languages: XML, XSL, …
- Stateless programming
  - no update of data structures
  - yields simplicity



## Getting Started

## Programming Language

- Implements a programming model
- Describes programs
  - composed of **statements**
  - compute with **values**
- Let's have a first look
  - statements
  - values



## Interactive System

```
declare
X = 1234 * 5678
{Browse X}
```

- Mark program fragment in Emacs buffer
- Feed marked region
  - program fragment is compiled
  - compiled program fragment is executed
- Interactive system: use like a calculator
- Start practice after the lecture

## Interactive System: What Happens?

```
declare
X = 1234 * 5678
{Browse X}
```

- Declares variable `X`
- Assigns the variable the value `7006652`
  - obtained by computing `1234*5678`
- Applies procedure `Browse` to value stored under X
  - pops up a window that shows `7006652`



## Variables

- Short-cuts for values
- Can be assigned at most once
- Are dynamically typed as opposed to Java
- Two aspects
  - variable identifier: what you type
    a string starting with capital letter
    `Var, A, X123, OnlyIfFirstIsCapital`
  - store variable: part of memory system
    initially, an empty box

## Variable Declaration

```
declare
X = …
```

- `declare` statement
  - creates new **store variable**
  - links identifier `X` in environment to store variable
- Environment
  - maps identifiers to variables (values)



## Assignment

```
declare
X = 42
```

- Assignment takes store variable and replaces with value
- Environment will map `X` to value `42` now

## Single Assignment

- Variable can only be assigned at most once
  - also: **single assignment variable**
- Incompatible assignment: raise error

```
X = 43
```

- Compatible assignment: do nothing

```
X = 42
```



## Redeclaring Variables

```
declare
X = 42
declare
X = 11
```

- Variables can be redeclared
- Environment will always map variable identifier to store variable introduced last
  - here X will refer to 11

## Data Structures (Values)

- Simple data structures
  - integers `42, ~1, 0`
    `~` means unary minus
  - floating point `1.01, 3.14`
  - atoms `atom, 'Atom', nil`
- Compound data structures
  - lists
  - tuples, records



## Functions

- Defining a function
  - give a statement that defines what to compute
- Applying a function
  - use the function to compute according to its definition
  - also: calling a function

## Our First Function

- Computes the negative value of an integer
  - takes one argument: the integer
  - returns a value: the negated integer
- In mathematical notation:

$$\text{minus:} \begin{cases} \text{Integer} & \to \quad \text{Integer} \\ n & \mapsto \quad \sim n \end{cases}$$



## Function Definition `Minus`

```
declare
fun {Minus X}
   ~X
end
```

- Declares the variable `Minus`
- Assigns the function to `Minus`

## Function Application `Minus`

```
declare
Y = {Minus ~42}
{Browse Y}
```

- `Y` is assigned value computed by application of `Minus` to `~42`



## Syntax

- Function definition

  `fun` {*Identifier Arguments*}
  
  *body of function*
  
  `end`

- Function application

  `X = `{*Identifier Arguments*}

## Maximum Function

- Computes maximum of two integers
  - takes two argument: integers
  - returns a value: the larger integer
- In mathematical notation:

$$\text{max:} \begin{cases} \text{Integer} \times \text{Integer} \rightarrow \text{Integer} \\ n, m \mapsto \begin{matrix} n, & n>m \\ m, & \text{otherwise} \end{matrix} \end{cases}$$



## Function Definition Max

```
declare
fun {Max X Y}
   if X>Y then X
   else Y
   end
end
```

- New construct: conditional (if-then-else)

## Function Application `Max`

```
declare
X = {Max 42 11}
Y = {Max X 102}
{Browse Y}
```



## Now the Minimum

- Possible: cut and paste
  - just repeat what we did for `Max`
- Better: compose from other functions
  - reuse what you did before
  - good, when complicated functions can be reused
- For minimum of two numbers

  min($n$,$m$) = ~max(~$n$,~$m$)

## Function Definition Min

```
declare
fun {Min X Y}
   {Minus {Max {Minus X}
               {Minus Y}}}
end
```



## Function Definition Min (With ~)

```
declare
fun {Min X Y}
   ~{Max ~X ~Y}
end
```

## Inductive Function Definition

- Factorial function  $n!$
  - inductively defined as

$$0! = 1$$
$$n! = n * ((n-1)!)$$

  - program as function `Fact`

- How to compute?
  - by recursive application of `Fact`



## Function Definition `Fact`

```
fun {Fact N}
   if N==0 then % Equality test
      1
   else
      N * {Fact N-1}
   end
end
```

## Recursion

- General structure
  - base case
  - recursive case
- For natural number *n* often
  - base case: *n* is zero
  - recursive case: *n* is different from zero
    *n* is greater than zero
- Much more: next lecture



## Functions Are Special Case

- General concept
  - procedure
  - plus variable to return computed value
- Again: next lecture…

## Summary

- Variable
  - variable declaration
  - store variables
  - assignment
- Data structures (Simple values)
  - numbers and atoms
- Functions
  - definition
  - call (application)
- Recursion



## Preview of Next

- Interesting data structures
  - lists, tuples, records
- More on variables
  - bound and unbound variables
  - partial values
  - dataflow variables and dataflow synchronization
- More on computing
  - pattern matching
- Recursion
  - recursion over lists
  - recursion over integers
- Computation model: what, why, how?

## Data Structures

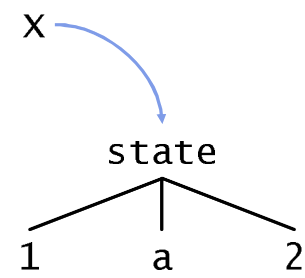
- Already seen:
  - number: integers `1, 2, ~1, 0`
    floating point (floats) `1.0, ~1.21`
  - atom: `a, 'Atom', v123`

- Now we address: compound data structures
  - tuple combining several values
  - list special case of tuple
  - record generalization of tuple



## Tuples

# Tuples

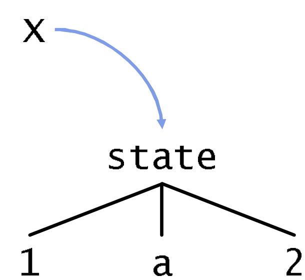



`X=state(1 a 2)`

- Combine several values (variables)
  - here: `1`, `a`, `2`
  - position is significant!
- Have a label
  - here: `state`



# Tuple Operations

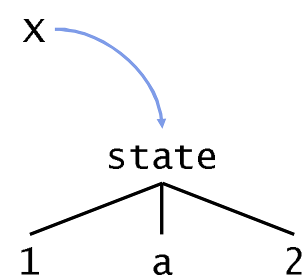



`X=state(1 a 2)`

- `{Label X}` returns *label* of tuple `X`
  - here: `state`
  - is an atom
- `{Width X}` returns the *width* (number of fields)
  - here: `3`
  - is a positive integer

# Feature Access (Dot Access)

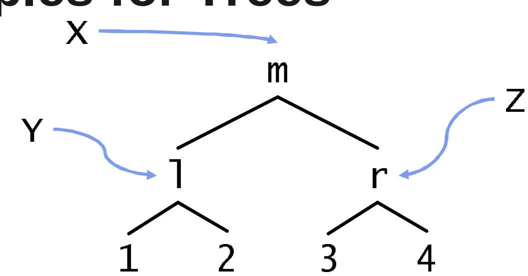
`X=state(1 a 2)`




- Fields are numbered from `1` to `{Width X}`
- `X.N` returns `N`-th *field* of tuple
  - here, `X.1` returns `1`
  - here, `X.3` returns `2`
- In `X.N`, `N` is called *feature*



# Tuples for Trees




- Trees can be constructed with tuples:

```
declare
Y=l(1 2) Z=r(3 4)
X=m(Y Z)
```

## Equality Operator (==)

- Testing equality with an atom or number
  - simple, must be the same number or atom
  - okay to use
  - we will see pattern matching as something much nicer in many cases
- Testing equality among trees
  - not so straightforward
  - don’t do it, we don’t need it (yet)



## Summary: Tuples

- Tuple
  - label
  - width
  - field
  - feature

# Records

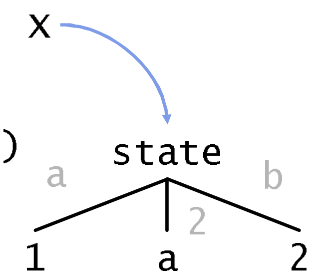


## Records

- Records are generalizations of tuples
  - features can be atoms
  - features can be arbitrary integers
    - not restricted to start with 1
    - not restricted to be consecutive
- Records also have label and width
- Needed for lab 01, will be discussed again

# Records

```
X=state(a:1 2:a b:2)
```




- Position is insignificant
- Field access is as with tuples

  `X.a` is `1`



# Tuples are Records

- Constructing

  ```
  declare
  X = state(1:a 2:b 3:c)
  ```

  is equivalent to

  ```
  X = state(a b c)
  ```

## What we have done

- Interesting data structures
  - lists, tuples, records
- More on variables
  - bound and unbound variables
  - partial values
  - dataflow variables and dataflow synchronization
- More on computing
  - pattern matching
- Recursion
  - recursion over lists
  - recursion over integers
- Computation model: what, why, how?



## What you should do?

- Chapter 1: Introduction to Programming Concepts
- **Reading suggestions:**
  - Browse as you like Abstract and Preface [casual],
  - Introduction 1.1-1.11 [careful], [try examples] Appendix A.1 for Oz Development Environment [as you need] and Appendix B.1 to B.3
  - **TO DO**: install Emacs, Mozart on your PC

## What you should do?

- Start working on Tutorial 1
- Look at Assignment 1



## See you next week!