Week 01 - Lecture 1 Slides

Dr. Yeganeh Bahoo

Created: 2023-09-17 Sun 11:33

Table of Contents

Lecture 1: logistics and introduction

Learning objectives:

By the end of this lecture you should be able to:

- Familiarize yourself with your instructor and office hours booking process.
- Navigate the course website and access the Course Outline.
- Identify the learning technologies we'll use throughout the course.
- Understand the requirements for succeeding in this course.
- Describe the importance of data structures in programming.
- Get acquainted with the fundamental building blocks of the Lisp programming language.

Welcome to CPS 305 Data Structures

Agenda:

- Course logistics
- Expectations
- Let's get started!

Course logistics

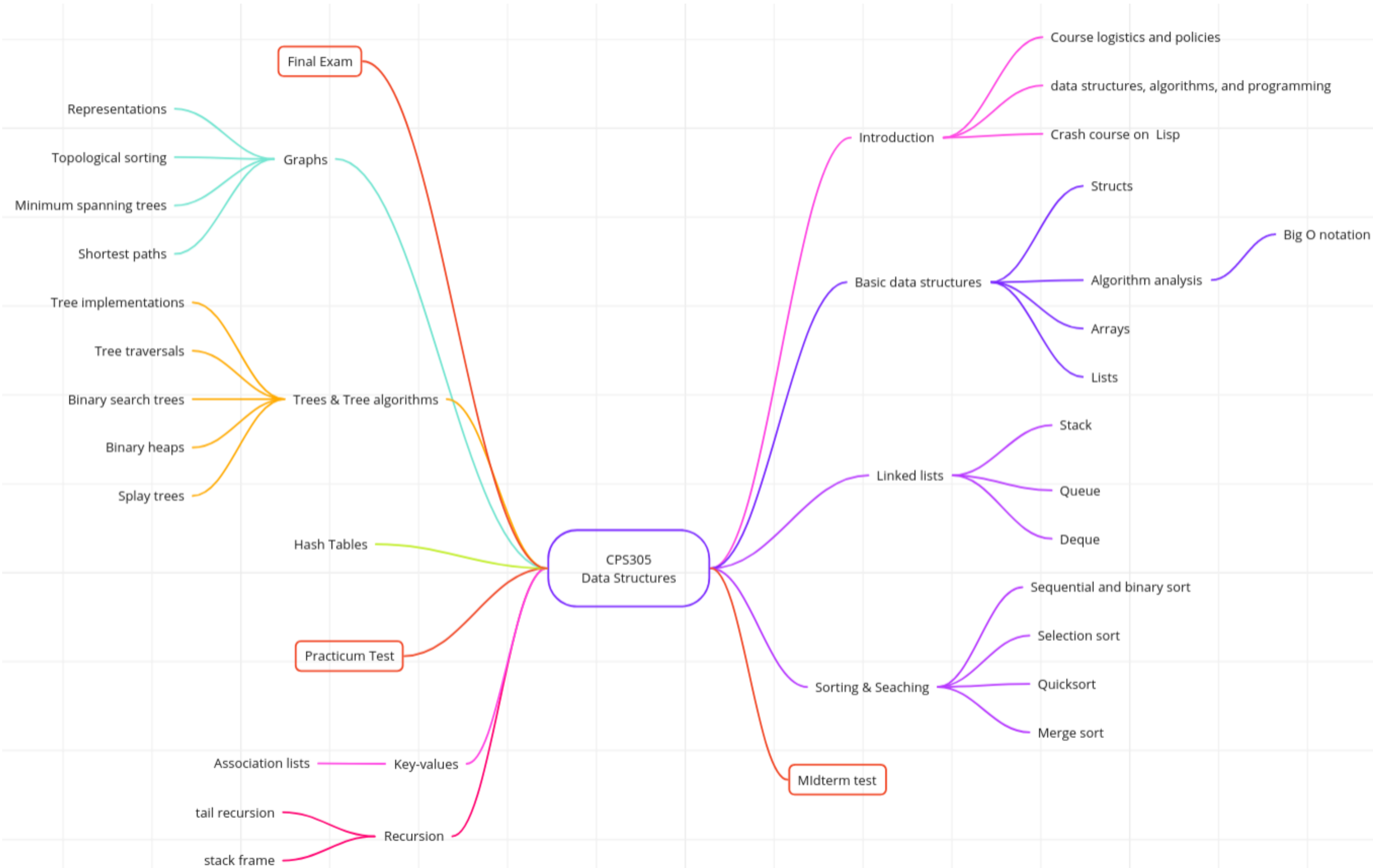
First things first, let's take a tour of our course website, available at my.torontomu.ca. There, you will find essential information such as:

- Office hours for one-on-one discussions.
- Practice lab exercises to enhance your learning.
- Course Outline document, providing details about our course policies, coverage, and examinations, including the Midterm Test, Practicum Test, and Final Exam.

My expectations of you

- Prepare for the next lecture
- Attend the Practice Labs and solve the exercises yourself
- Feel free to drop by during my office hours; I'm here to help you succeed!

Course Map



Let's get started!

Computing, algorithms, programming, data structures: what is it, why study it?

Why data structures, algorithms, and programming languages

- 300 BC

    - Euclid (Εὐκλείδης) recognized that objects have **structure** and attributes; he defined rules for reasoning about how they change.
- Today
    - we use **data structures** to build models of objects
    - we use **algorithms** to reason about and simulate change in such models
    - we use **programming languages** to encode algorithms for execution by a computer

Data structures is about looking at the pros and cons of each structure vis-a-vis the problem you are trying to solve.

Algorithms is about analyzing the tradeoffs between space and time efficiency in the changes one or another algorithm causes in a model.

Computer program is a computer-executable specification of an algorithm.

Why study data structures in Lisp?

Our goal in this course: to learn the fundamentals of data structures such as lists, stacks, queues, trees, and graphs.

Lisp's pros and cons to help us reach the above goal:

- Pros:
    - provides enough control of concrete data (direct access on how data is accessed and manipulated by the program)
    - functional programming features can make it easy to manipulate and transform data structures
    - highly interactive development environment
    - Lisp's list data structure is unique in that it allows for easy construction of recursive data structures
    - The data structure programming skills you learn in this course will transfer to almost any language you use in the future
- Cons:
    - the unfamiliar syntax

While the syntax may be new, we'll provide guidance for a rewarding learning experience.

Learning Lisp

> “One learns by doing a thing; for though you think you know it, you have no certainty until you try.” (Sophocles)

- Programming environment:
    - Emacs+Slime: Program editor (Emacs) and Superior Lisp Interactive Mode for Emacs (SLime)
    - Common Lisp: Steel Bank Common Lisp (SBCL) compiler
    - Quicklisp: library manager
- Learning resources
    - Books
        - The Common Lisp Cookbook
        - Common Lisp: A Gentle Introduction to Symbolic Computation
    - Videos
        - Little bits of lisp
        - The Absolute beginner's guide to Emacs
    - Common Lisp complete documentation
        - Common Lisp HyperSpec

About the notation I will use in class

To denote what a form evaluates to, I will use the following equivalent notations in my lecture slides:

1. The notation bellow mimics the user's interaction with Lisp's Read-Eval-Print-Loop (REPL - pronounced "Repel"):

```
CL-USER> (/ (- 2 (- 3 (+ 6 (/ 4 5))))
            (* 3 (- 6 2)))   ; This is the form you type in at the REPL, then press Enter
29/60         ; This is the value lisp will output; it is the value the form evaluates to.
```

2. The notation below just shows the form and the value it evaluates to.

```
(/ (- 2 (- 3 (+ 6 (/ 4 5))))
   (* 3 (- 6 2)))            =>   29/60
```

What are Lisp "forms"?

A crash course on Lisp

Central to the lisp language is the idea of a **form**, which are expressions that can be evaluated by lisp, e.g.:

```
(+ 3 5)
```

The various *forms*:

- a lisp **constant**: evaluates to itself
  - **numbers**: 2, 2.0, 3/5
  - **individual characters**: `#\a`, `#\@`
  - **strings**: `"abcg"`, `"Hello World!"`
  - **boolean constants**: `T` (true) and `NIL` (false).
- a lisp **symbol**: evaluates to the value it identifies. Made up of letters, numbers, and characters like `+ - / * = < > ? ! _`
  - Examples: `foo`, `*4ice9evaluate*`, `--<<==>>--`.
- a **function application**
- a **special form**

Function application

- a **function application**: is a non-empty list whose elements are themselves forms $(\underbrace{f}_{\text{function}} \ \underbrace{a_1 \ a_2 \ a_3 \ \cdots a_n}_{\text{arguments}})$, where f must evaluate to a function

- Evaluation of a function application:

  if the *function* is **not** the name of a **special form**

  1. evaluate the *arguments* from left to right
  2. apply the *function* to the *arguments* that are the values of the respective forms

  otherwise, evaluate the form as a **special form**

Let's see some examples.

Examples

Evaluating mathematical expressions:

$$(2 + \sqrt{4} \times 6) \times (3 + 5 + 7)$$

```
CL-USER> (* (+ 2
               (* (sqrt 4) 6))
            (+ 3 5 7))
390
```

$$\frac{2-(3-(6+\frac{4}{5}))}{3\times(6-2)}$$

```
CL-USER> (/ (- 2 (- 3 (+ 6 (/ 4 5))))
            (* 3 (- 6 2)))
29/60
```

- In English, we often use pre-fix notation when describing a mathematical application of a function (or operator) to its arguments
- For example, consider how you would instruct someone to perform the operation below:

$$x + y + w + z$$

Add x, y, w, and z

```
(+ x y w z)
```

Exercise

Convert each mathematical expression into Lisp notation.

- `(a + b) * (c - d)`
- `2 * (x + y) / (3 - z)`
- `sqrt((p^2 + q^2) / r)` Note: assume there is no `^` operator in Lisp

Solution

- `(* (+ a b) (- c d))`
- `(* 2 (/ (+ x y) (- 3 z)))`
- `(sqrt (/ (+ (* p p) (* q q))) r)`