Introduction

- CS 136 builds on CS 135
- Programming in a more "real-world" environment
- Introduction of the C language and continuing with Racket (Scheme)
- Focus on design, analysis and implementation of fundamental algorithms and data structures
- Provide tools and concepts necessary to solve computational problems in a robust, efficient and verifiable manner
- Ability to decompose and tackle a problem using **abstraction**.

CS 136 Information

Web page: Your main information source:
http://www.student.cs.uwaterloo.ca/~cs136/

Discussion: Your main discussion forum – Piazza
https://piazza.com/
Post questions! DO NOT POST CODE. Instructors and tutors will monitor the forum and answer.
Can post anonymously to class, but **your identity is always visible to instructors/tutors.**

Tutorials: Mondays; check website.

Office hours: See website. To get help, you **must**:

1. Present legible, commented code
2. Present the test cases you have tried

Midterm: Monday, March 4, 7pm – 8:50pm

Materials

Textbooks:

- “C Programming: A Modern Approach” (CP:AMA) by K. N. King. **(Required)**
- “How to Design Programs” (HtDP) by Felleisen, Flatt, Findler, Krishnamurthi `http://www.htdp.org`

Clickers:

- Clickers available in the bookstore (required for participation marks)

Presentation handouts:

Will be made available on web page

Evaluation

Grading Scheme:

20% assignments
5% participation (clickers)
25% midterm
50% final exam

- Your assignment average **and** *weighted* exam average must both be at least 50% **or you will not pass!**
- **Your final grade must be 60% or greater to take CS 246 (required for CS majors)**

Clickers

Class Participation Mark

- Based on use of "clickers" (purchase at Bookstore, register as part of Assignment 0)
- Purpose: to encourage active learning and provide real-time feedback
- Several multiple-choice questions during each lecture
- Marks for answering (more for correct answer)
- Best 75% over whole term used for 5% of final grade

Assignments

- **Assignments are to be completed individually.**
- **Do not share code.**
- **A0 is due this Friday**
  A0 does not count toward your grade, but must be completed before other assignments will be marked
- All other assignments are due on Wednesdays at 11:59am
- Assignments may be submitted (or re-submitted) late up until April 8 with a $(\times\frac{1}{2})$ deduction
- Assignment solutions will not be posted.
- **A1 is due next Wednesday (Jan 16)**

Software

Software:

- `gedit` – Open source code editor
- `runC` – Add-on to `gedit` that runs your programs (both C and Racket)
- Try to avoid DrRacket (but it may be useful for debugging)

- The Marmoset system automatically marks your code. You submit your source files via the web, Marmoset runs and tests them.
- Marmoset uses `runC` to run your program. **If it doesn't work with runC, it won't work with Marmoset.**
- Options for your work environment:
    - Lab computers (recommended)
    - Use VirtualBox to run linux inside mac or windows
    - Install `gedit`/`runC` on your own linux machine

The General Themes

- Write **programs** that do stuff in the real world
- **Interact** with users, devices, other programs
  - **Input** from the outside world
  - **Output** to the outside world
- Use **Abstraction** to provide re-usable code to others without exposing our implementation details
- **Store** and **Manipulate** data and “state”

Languages

- Two languages: Racket/Scheme and C
- Racket and C offer radically different approaches
  - Racket (as we have seen it so far) is *functional*
    - about defining behaviour of functions
    - ability to construct and manipulate functions
    - identifiers are constant (do not change once defined)
  - C is *imperative*
    - about issuing sequences of commands
    - repetition emphasized over recursion
    - ability to manipulate memory directly
  - Racket is actually “multi-paradigm” as we will see.
- **This is not a “learn C” or “learn Racket” course**
- We will present language features and syntax as needed
- **What you will learn can be transferred to any language**

A Note About Terminology

- Because we are starting with a solid foundation in a *functional* language (Racket), we will be using terminology (nomenclature) that is very common in functional languages
- For example, we will soon be using the terms **mutation** and **side effects**
- Some of this terminology is rarely used with *imperative* languages (our C text doesn't use the word "mutation")
- We will also be presenting examples in C that mimic behaviour in Racket (and vice versa) to help better conceptualize the material
- This will enrich your understanding of both approaches and help make you a more well-rounded programmer

The full Racket language

- Racket is an “industrial strength” Scheme/Lisp variant
- You now can access the full language:
  `http://docs.racket-lang.org/reference/`
  `http://docs.racket-lang.org/guide/`
  in DrRacket: choose “Determine language from source”

There are some important differences from the teaching languages:

- `check-expect` and the Stepper are gone
- Functions can have no arguments: `(define (x) 10)`
- `true` and `false` are represented by “actual” values `#t` and `#f`
- Racket does not enforce that the second argument to `cons` be a list.
  - The value of the expression `(cons 1 2)` **is not a list**. It is displayed as `'(1 . 2)`. **Do not do this.**
  - `(cons 1 (cons 2 empty))` is a list

Summary

- In CS135, our focus was on designing *functions*
    - “Data” were defined as constants; exact same output every time
- In CS136, our focus is on designing *programs* that interact with other programs/data/users.

Modularization a.k.a. “abstraction,” a.k.a. “hiding detail.”
Some code does something complicated, but **using** it to do that thing is simple.

I/O For interaction with other data/programs/users

Efficiency Are our programs fast enough?
Could we do better?

Mutation For memory – storing data or “state”