CS 136: Elementary Algorithm Design and Data Abstraction

Official calendar entry: This course builds on the techniques and patterns learned in CS 135 while making the transition to use of an imperative language. It introduces the design and analysis of algorithms, the management of information, and the programming mechanisms and methodologies required in implementations. Topics discussed include iterative and recursive sorting algorithms; lists, stacks, queues, trees, and their application; abstract data types and their implementations.

Welcome to CS 136 (Spring 2017)

Instructor: Mark Petrick

Web page:

`http://www.student.cs.uwaterloo.ca/~cs136/`

Other course personnel: ISAs (Instructional Support Assistants), IAs (Instructional Apprentices), ISC (Instructional Support Coordinator): see website for details

Lectures: Tuesdays and Thursdays

Tutorials: Mondays

Be sure to explore the course website: *Lots* of useful info!

About me (your instructor)

Programming languages

Most of this course is presented in the **C** programming language.

While time is spent learning some of the C syntax, this is not a “learn C” course.

We present C language features and syntax only as needed.

We continue to use Racket (a dialect of Scheme) to illustrate concepts and highlight the similarities (and differences) between the two languages.

What you learn in this course can be transferred to most languages.

Programming environment (Seashell)

We use our own customized "`Seashell`" development environment.

- browser-based for platform independence
- works with both C and `Racket`
- integrates with our submission & testing environment
- helps to facilitate your own testing

See the website and attend tutorials for how to use `Seashell`.

Course materials

Textbooks:

- "C Programming: A Modern Approach" (CP:AMA) by K. N. King. **(strongly recommended)**
- "How to Design Programs" (HtDP) by Felleisen, Flatt, Findler, Krishnamurthi

  (very optional)

  Available for free online: `http://www.htdp.org`

Course notes:

Available on the web page and as a printed coursepack from media.doc (MC 2018).

Several different styles of “boxes” are used in the notes:

Important information appears in a thick box.

Comments and “asides” appear in a thinner box. Content that only appears in these “asides” will **not appear on exams**.

Additional **“advanced”** material appears in a “dashed” box.

The advanced material enhances your learning and may be discussed in class and appear on assignments, but you are **not responsible for this material on exams** unless your instructor explicitly states otherwise.

Marking scheme

- 20% assignments (roughly weekly)
- 5% participation
- 25% midterm
- 50% final

To pass this course, you must pass both the assignment component and the weighted exam component.

Class participation

We use `i>Clickers` to encourage active learning and provide real-time feedback.

- `i>Clickers` are available for purchase at the bookstore
- Any physical `i>Clicker` can be used, but we do **not** support web-based clickers (*e.g.,* `i>Clicker Go`)
- Register your clicker ID in Assignment 0
- To receive credit you must attend your registered lecture section (you may attend any tutorial section)
- Using someone else's `i>Clicker` is an academic offense

Participation grading

- 2 marks for a correct answer, 1 mark for a wrong answer
- Your best 75% responses (from the entire term) are used to calculate your 5% participation grade
- For each tutorial you attend, we'll increase your 5% participation grade 0.1% (up to 1.2% overall, you cannot exceed 5%)

To achieve a perfect participation mark

- answer 75% of all clicker questions correctly, **or**
- answer $\approx$ 40% of all clicker questions correctly, and attend every tutorial

Assignments

Assignments are *weekly* (approximately 10 per term).

Each assignment is weighted equally (except A0).

Make sure you **read the assignment instructions carefully**. Each assignment may have different instructions and requirements.

A0 does not count toward your grade, **but must be completed** before you can receive any other assignment marks.

Assignment *questions* are colour-coded as either “black” or “gold” to indicate if any collaboration is permitted.

For **BLACK** questions, **moderate collaboration** is permitted:

- You can discuss assignment *strategies* openly (including online)
- You can search the Internet for strategies or code examples

- You can discuss your code with *individuals*, but **not** online or electronically

  (piazza, facebook, github, email, IM, *etc.*)

- You can show your code to others to help them (or to get help), but copying code is not allowed

  (electronic transfer, copying code from the screen, printouts, *etc.*)

If you submit any work that is not your own, you must still cite the origin of the work in your source code.

For **GOLD** questions, **no collaboration** is permitted:

- **Never share or discuss your code**
- Do not discuss assignment *strategies* with fellow students
- Do not search the Internet for strategies or code examples

You may discuss your code with course staff.

Academic integrity is strictly enforced for gold questions.

Assignments: second chances

Assignment deadlines are strict, but some assignment questions may be granted a "second chance".

- Second chances are granted automatically by an automated "oracle" that considers the quantity and quality of the submissions
- Don't ask in advance if a question will be granted a second chance; we won't know
- Second chances are (typically) due 48 hours after the original
- Your grade is: $\max(\text{original}, \frac{\text{original}+\text{second}}{2})$
  (there is no risk in submitting a second chance)

Marmoset

Assignments are submitted to the Marmoset submission system:

`http://marmoset.student.cs.uwaterloo.ca/`

There are two types of Marmoset *tests*:

- **Public** (*basic / simple*) tests results are available immediately and ensure your program is "runnable".
- **Private** (*comprehensive / correctness*) tests are available after the deadline and fully assess your code.

Public tests do **not** thoroughly test your code.

- Marmoset uses the best result from all of your submissions (there is never any harm in resubmitting).
- For questions that are *hand-marked*, the most recent submission (before the deadline) with the highest score is marked.
- You can *submit* your assignments via `Seashell` and view *public* test results.
- Every submission is stored (backed up) for your convenience.

You must log into Marmoset to view your **private** test results (after the deadline).

Design recipe

In CS 135 you were encouraged to use the ***design recipe***, which included: contracts, purpose statements, examples, tests, templates, and data definitions.

The design recipe has two main goals:

- to help you **design** new functions from scratch, and
- to aid **communication** by providing **documentation**.

In this course, you should already be comfortable designing functions, so we focus on **communication** (through documentation).

Documentation

In this course, every function you write must have:

- a **purpose** statement, and
- a **contract** (including a **requires** section if necessary).

Unless otherwise stated, you are **not** required to provide: templates, data definitions or examples.

Later, we extend contracts to include *effects* and *time* (speed / efficiency).

Hand-marking

Questions that are hand-marked for *"style"* may be evaluated for:

- documentation and comments
- code readability
- whitespace and indentation
- identifiers (variable & function names)
- appropriate use of helper functions
- testing methodology

The purpose of hand-marking is not to "punish" or "torture" you.

It is **formative feedback** to improve your learning.

Unfortunately, we do not have the resources (staff) to hand-mark all assignment questions.

Well formatted and documented code is still expected, even if it is not hand-marked.

We will not provide assistance (office hours or piazza) if your code is poorly formatted or undocumented.

View your formative feedback on `MarkUs`.

Testing strategies

You are expected to test your own code.

Simply relying on the public marmoset tests is not a viable strategy to succeed in this course.

There are two key testing strategies used in this course:

- assertions
- I/O

Assertion-based testing

In assertion-based testing, the main program is a sequence of function calls, each ***"asserted"*** to be correct.

A successful run passes all tests and does "nothing".

This is used in CS 135:

```
(define (my-sqr n)
  (* n n))

(check-expect (my-sqr 5) 25)
(check-expect (my-sqr -3) 9)
```

In CS 136 we do not use `check-expect` (alternatives are discussed later).

I/O

This course introduces ***Input and Output (I/O)***.

Informally, you might say that a function has "inputs" and "outputs" (if you "input" `5` into `my-sqr` it will "output" `25`).

We avoid this casual use of "input" and "output".

The function `my-sqr` is **passed** (or *consumes*) the *argument* `5` and **returns** (or *produces*) the value `25`.

In this course, **input** is read from a keyboard (or a file), and **output** is displayed to the screen (or a file).

I/O testing

In I/O-based testing, the main program reads *input* from a ***test file*** (`.in` file) and then generates output. The output is then checked against the expected results (`.expect` file).

For example, consider a program that reads in integers from input and then outputs the square of each integer.

An I/O test for such a program would be:

`mytest.in`

```
5
-3
```

`mytest.expect`

```
25
9
```

Our `Seashell` environment supports multiple test files. Each test is run independently.

Getting help

- office hours (see website)
- lab hours (Thursdays 10 – 6 in MC 3004)
- tutorials (Wednesdays)
- textbook
- piazza

Course announcements are made on piazza and are considered **mandatory reading**.

Piazza etiquette

- **read** the *official assignment post* before asking a question
- **search** to see if your question has already been asked
- **use** meaningful titles
- **ask** *clarification questions* for assignments
  (do not ask *leading questions* for **GOLD** questions)
- **do not** discuss strategies for **GOLD** questions
- **do not** post any of your assignment code *publicly*
- you can post your **commented** code *privately*, and an ISA or Instructor *may* provide some assistance.

Appendix

The notes also include an ***appendix***, which contains additional content, examples and language syntax details that may not be covered in the lectures. Some of this content will be covered in tutorials.

At this point in the course, you should review:

- Appendix A.1: A Review of CS 135
- Appendix A.2: Full Racket

You are still responsible for content in the Appendix, even if it is not presented in the lectures.

Main topics & themes

- imperative programming style
- elementary data structures & abstract data types
- modularization
- memory management & state
- introduction to algorithm design & efficiency
- designing “medium” sized, “real world” programs with I/O

In this Section we introduce some of the main topics.

Three of the most common programming paradigms are functional, imperative and object-oriented. The first three CS courses at Waterloo use different paradigms to ensure you are "well rounded" for your upper year courses. Each course incorporates a wide variety of CS topics and is **much more** than the paradigm taught.

CS 135	$\Rightarrow$	CS 136	$\Rightarrow$	CS 246
functional		*imperative*		*object-oriented*

At the end of each Section there are ***learning goals*** for the Section (in this Section, we present the learning goals for the entire course).

These learning goals clearly state what our expectations are.

Not all learning goals can be achieved just by listening to the lecture. Some goals require reading the text or using `Seashell` to complete the assignments.

Course learning goals

At the end of this course, you should be able to:

- produce well-designed, properly-formatted, documented and tested programs of a moderate size (200 lines) that can use basic I/O in both Racket and C
- use imperative paradigms (e.g., mutation, iteration) effectively
- explain and demonstrate the use of the C memory model, including the explicit allocation and deallocation of memory
- explain and demonstrate the principles of modularization and abstraction

- implement, use and compare elementary data structures (structures, arrays, lists and trees) and abstract data type collections (stacks, queues, sequences, sets, dictionaries)
- analyze the efficiency of an algorithm implementation