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 2015)

Instructors: Ahmed Hajyasien, Dan Holtby

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!

Course Materials

Textbooks:

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

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

All work must be submitted to the Marmoset submission system:
`http://marmoset.student.cs.uwaterloo.ca/`

- For correctness marks, marmoset takes the best result from all of your submissions (there is never any harm in resubmitting)
- Marmoset provides simple feedback to ensure your program is "runnable" and **should not be used to test your code**

A0 is due this Friday: it does not count toward your grade, **but must be completed on time**.

While other courses may allow or encourage working together in groups, CS 136 assignments must be done **individually**.

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

An assignment may have "warm-up" questions that can be openly discussed.

Assignments: second chances

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

- Second chances are granted automatically by a fixed decision rule 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)

Getting Help

In addition to *office hours*, *tutorials* and *lab hours* we use an online discussion forum (**piazza**).

Instructors, ISAs and your fellow students monitor piazza and answer questions.

Post *clarification questions* to help understand assignments, but **do not** discuss assignment strategies, and **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.

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

Languages

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

While significant time is required to learn 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.

Software

We use our own customized “`Seashell`” environment, which has a browser interface.

`Seashell` works with both C and Racket, provides verbose error messages, and helps to facilitate your testing.

Our testing environment (Marmoset) uses the same environment as `Seashell` to run your program. (If it does not work with `Seashell`, it will not work with our tests).

See the website for how to use `Seashell`.

Design Recipe

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

There are two main goals with the design recipe:

- 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 and 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, examples, or tests.

Throughout this course, we will extend contracts to include *effects* and *time* (efficiency).

You are expected to test your own code. Advanced testing strategies are discussed in Section 07.

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