

CS 136 - Elementary Algorithm Design and Data Abstraction

Instructors: Tim Brecht and Dan Holtby

Prerequisites: CS 135 or 85% or higher in CS 115 or CS 116

Programming Language: C

Web Page:

<http://www.student.cs.uwaterloo.ca/~cs136/>

Programming Language:

Racket (for reference and understanding the difference from Functional to Imperative programming language) and C (Imperative Paradigm)

Programming Environment:

Seashell - Created specifically for this course by university

Works with both C and Racket

Integrates with our submission and testing environment

Also helps to facilitate your own testing

Website: <https://www.student.cs.uwaterloo.ca/~cs136/seashell-old/>

Version: 3.0.3

Credentials to login-

Username: WatIAM username

Password: <Set for student.cs.uwaterloo.ca server>

Textbooks:

“C Programming: A Modern Approach” (CP:AMA) by K. N. King.

Can be issued from DC for 3 hours

Course Notes are available on web page under the course notes section and printed coursepack can be bought at media.doc (MC 2018)

Styled Boxes:

Important information appears in a thick box

Comments and asides appear in a thinner box. Content that appears in these asides will not appear on the exams.

Addition "advanced" material appears in a dashed box in the course notes. The advance 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:

Assignments: 20% (roughly weekly)

Participation: 5%

Midterm: 25%

Final: 50%

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

Clicker Participation:

i>Clickers are used in this class for participation marks. Purchasable at the bookstore.

Follow Assignment 0 for Clicker registration

To receive clicker credit please attend your own lectures.

Please be wary of not using someone else's clickers it is an Academic Offense

Assignments

Assignments are weekly. Depending on the term, there may be up to 10 assignments. This term there were 8 assignments.

ADVICE: Please start assignment early and try to take help of ISA's. Also try coding on a paper first and then trace through and then code on the computer.

Unit 02: A Functional Introduction to C

Readings: CP: AMA 2.2 - 2.4, 2.6-2.8, 3.1, 4.1, 5.1, 9.1, 9.2, 10.1, 15.2

History of C

Developed by Dennis Ritchie in 1969-73 to make the Unix OS more portable. A successor to B and its successor is C++. It was specifically designed for low-level access to memory which has been discussed in later topics of this course.

It easily translates into "machine codes" which is discussed in later sections.

NOTE: C99 standard is used for this course.

From Racket to C

Racket is one language that you learned in CS 135 and now we will use Racket code and compare it with C basic syntax so it helps you to understand C syntax and we build on that.

```
;Single Line Comments in Racket

#/Multi Line Comments in Racket/#

(define my_number 42) / constant definition in racket
```

```
//Single lineComments in C

/* Multi Line Comments in C */

const int my_number = 42; //Constant in C
```

Typing: 1. Racket is a dynamic typing based language which means you do need to define the data type of the variable and a variable can have different data types but it is determined during runtime

```
(define dtype (cond [(>= x 0) 42]

                    [else "invalid"]))
```

2. Where as C uses static typing, where the type of identifier must be known before the program is run.

```
const int stype = 42;
```

Initializing: C has a different way of initializing as even when we started with the constant declaration, those were also the initializing of the the variable.

Expressions: In CS 135 we did prefix notation where the operation was followed by two operands to compute the final expression, now in CS 135 we use the infix notation which we regularly use in our daily life.

C-Operators: The following are the few operators in C and we will learn more as we proceed further
+, -, *, /.

NOTE: The / operator is the division operator but it behaves as racket quotient function and truncates down to closest integer, in layman terms rounds to zero

NOTE: The % operator is the remainder operator which works as the modulo operator as you all did in Math 135

Function Definitions: The function definitions in racket and C are quite different but still very similar. Let us take an example and see for ourselves:

```
;Racket Function

;mysqr: Int -> Int

(define (my-sqr n)

  (* n n))
```

```
//C Function

int my_sqr (int n) {

  return n * n;

}
```

Hello, World!

The very first program that you write is the basic Hello World display in any language. To display output in C we use `printf` function.

```
//hello.c

//My first C program

#include <stdio.h>

int main (void) {

  printf("Hello, World!");

}
```

`Printf` command in particular uses placeholders for printing out the type of data you want to display. For the decimals it is `%d`.

Boolean Operators

1. In C, we use **false and true** as the boolean values which are represented by the values **0 and 1 respectively**.
2. The **equality** operator is represented by `==`.

3. The **not** equality operator is `!=`.
 4. The **not** operator is denoted by `!`.
 5. The and operator is denoted by `&&`.
 6. The or operator is denoted by `||`.
- The comparison operators are `>, <, <=, >=`.

Conditionals: There is no direct C equivalent of cond racket expression but we can use if...else

```
if (condition) {  
    //code...block  
}  
  
if (condition) {  
    //code...block  
} else {  
    //code...block  
}  
  
if (condition) {  
    //code...block  
} else if (condition) {  
    //code ... block  
}
```

NOTE: Recursions behave the same way as in Racket. We also have a conditionals operator in C which is unlike to if statement.

To require a module, we use `#include <module.h>`

To require a my_module, we use `#include "my_module.h"`

Creating a module in C

We place the declarations of the functions in the interface .h file and we place the definitions in the implementation .c file.

#include is a pre-processor directive and they can modify a file before they run.

Scope is a local scope concept in the C language which is the same as Racket but in the top-level values are all program scope in C.

Assert is a type of statement that allows you to assert things in your program and check if they are false then the program won't run. To use assert in your client, one should add `#include <assert.h>`.

Bool Types are also not built in and are added with the standard library `#include <stdbool.h>`

Floating Pointer Type is the C's data type to represent real non-integer number.

Structure are compound data types in C and are similar to the ones you saw in CS 135.

Unit 03: Modularizations & ADT

Modularization is to divide your program into small modules and an example of this is the helper functions we have been making since CS 135.

A module provides collection of functions that share a common aspect or purpose.

Modules can also provide elements that are not functions like data structures and variables

Modules Vs Files

Good style to store each module is to store it in a separate file.

Terminology

Client that requires a function that requires a function that a module provides.

NOTE: The module dependency graph cannot have any cycles

There must be a root or main file that acts only as a client.

Motivation

Advantages of Modularization

1. Re-Usability
2. Maintainability
3. Abstraction

Modularization is also known in computer science as the separation of concerns (SoC)

Example: fun number module

```
;;fun.rkt

(provide fun?)

(define lofn '(-3 7 42 136 1337 4010 8675309))

;;(fun? n) determines if n is a fun integer
```

```
;;fun?: Int -> Bool

(define (fun? n)

  (not (false? (member n lofn))))
```

`provide` special form makes `fun?` function available to clients

```
//client.rkt

(require fun.rkt)

(fun? 4010)      ;#t
(fun? 4011)      ;#f
```

Scope

Local: Identifiers are only visible inside of the local region (or the function body) where it is defined.

Global: Identifiers are developed at the top level, and are visible to all code following the definition

`provide` introduces new_level of scope to the variables or functions in racket.

global identifiers can have a program scope or a module scope.

Module scope

Identifiers are only visible inside of the module they are defined in.

Program Scope

Identifiers are visible outside of the module they are defined in.

Module Interface

It is the list of functions that the module provides.

Interface Documentation

Overall description of the module

List of functions it provides

The contract and purpose for each provided

Example: Sum Module

```
;;As module for summing numbers

(provide sum0first sum-squares)

;;Design Recipe for sum-first & sum-squares
```

```
;;;IMPLEMENTATION;;;;;
```

```
(define (sum-first n)  
  (/ (* n (+ n 1)) 2)
```

```
(define (sum-squares n)  
  (/ (* n (+ n 1) (- (* 2 n) 1)) 6)
```

```
(define (private-helper p )  
  ... )
```

Testing

Good practice to create a testing client

```
(require "sum.rkt")  
(= (sum-first 1) 1)
```

All tests should produce `#t`

Designing modules

Modules usually have a behaviour with respect to the program. We call these Cohesion and Coupling

GOAL: High Cohesion and Low Coupling

High Cohesion

It means all of the program modules and interface function are related and are working towards a "common goal".

Low Coupling

It means there is a little interaction between the modules. It is completely impossible to remove the coupling but it should be minimal

Information Hiding

Security: It is important as we may want to prevent the client from tampering with data used in the module
Flexibility

By hiding the implementation details from the client, we gain flexibility for change in implementation in the future.

Data Structures & Abstract Data Types (ADTs)

Three implementations with three different data structures.

1. A two element list
2. A structure with one field
3. A structure with two fields

The above module is an implementation of ADT

Formally an ADT is a mathematical model for storing and accessing data operations

Data Structure Vs ADTs

With a data structure, you know how the data is structured and you can access the data directly in any manner you desire.

In an ADT, you do not know how the data structure is implemented and you can only access the data through interface functions (operations) provided by the ADT

Collection ADT

It is an ADT designed to store any arbitrary number of items. They have a well-defined operation

We have already dealt with the dictionary which is one of the ADTs we will be revisiting in this section.

Dictionary (Revisited)

The dictionary ADT, also known as a map, association array or symbol table is a collection of pairs and keys. For those who have seen or programmed in `JavaScript` will know that JSON values are a dictionary.

Typical operations of Dictionary ADT

1. Lookup
2. Insert
3. Remove

Dictionary can be implemented as an association list.

Alternatively by a Binary Search Tree (which all of you did in CS 135 and we will implement them in Unit 12 again)

More collection ADTs

1. Stack
2. Queue
3. Sequence

STACK ADT

It is an ADT of items 'stacked' one on top of the other. Items are pushed and popped off the stack. Also known as a LIFO (Last In First Out) system. For eg. Imaging a stack of the plates, you can only remove items from the top and insert on the top.

USED: They are often used in browser histories and text editors.

Typical Stack Operations are

1. Push: Adds an item
2. Pop: Removes an item
3. Top: Returns the top item
4. is_empty: Determines if the stack is empty

STACK ADT Vs QUEUE LIST

They have a very close relation in terms of operations.

The one significant difference is only the top item is accessible in a STACK ADT whereas with list data structure every element is accessible.

QUEUE ADT

A queue is where items are added to the back and removed from the front. A queue is hence a FIFO (First in First out) system. For eg. Consider standing in a queue at your favourite McDonalds and then the person at the front orders and leaves but you join in at the last. So that is FIFO.

Typical Queue ADT operations

1. Add-back: Adds an item at the back of the list
2. Remove-front: Remove the item from the front of the queue.
3. Front: Returns the front item
4. is_empty: Checks if the queue is empty or not.

SEQUENCE ADT

It is a useful ADT when you want to be able to retrieve, insert and delete at any position in a sequence of items.

Typical Sequence ADT operations

1. item-at: Returns the item at a given position
2. insert-at: Inserts the item at a given position
3. remove-at: Removes the item at a given position
4. Length: returns the number of items in a sequence

Unit 04: Introduction to Imperative C

Functional Vs. Imperative Programming

The functional programming paradigm is to only constant values that never change.

begin: It ignores all the expressions except the last one.

Side Effects

A program/expression does more than effect on a produced value. It also changes the state of the program.

An expression statement is an expression followed by a semicolon ;

State

The biggest difference between imperative and functional paradigms is the existence of side-effects.

The state refers to the value of some data at the moment in time.

Mutation

When the value of the variable is changed

Mutable Variables

The `const` keyword is required explicitly to define a constant. Without it a variable is mutable.

Unit 05: The C Model

Unit 06: Introduction to Pointers

Readings: CP: AMA 11, 17.7

Pointer is a type used for storing an address

It is defined by using a star (*) before the identifier and it is part of the declaration not the identifier itself

```
int i = 5;  
  
int *p = &i;    /// p "points at" i
```

The type of a pointer is "int pointer" also written as "int *".

Unit 07: I/O & Testing

Unit 08: Arrays and Strings

Unit 09: Efficiency

Unit 10: Dynamic Memory & ADTs in C

Unit 11: Linked Data

Unit 12: Abstract Data Types

The following section is not covered in the course but is a information to CS 246

Unit 13: Beyond