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 programmin
g 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 facilitae 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 bx. Content that appears in these asides will not appear on the exmas.

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 einstructor 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 els's clickers it an Academic Offense

Assignments

Assignments are weekly. Depending on the term, there may be upto 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

Histroy 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 syatax 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 ruunntime

```
(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 intializing of the the variable.

Expressions: In CS 135 we did prefix notation where the operation was followed by two operands to computer the final expression, now in CS 135 we use the infix notaation 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 ad 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 ad 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 tyoe 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

NOTE: Recurrsions 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 cncept 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 seperate file.

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

NOTE: The module dependancy 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. Maintainabiltity
3. Abstraction

`Modularization is also knoen in computer science as the seperation 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 acan 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 off functions that the module provides.

Interface Documentation
Overall description of the module
List of functionsit 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-sqaures


;;;;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 imp[ossible 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 cloent, we gain flexibility for change in implementation in the future.

Data Structures & Abstract Data Types (ADTs)
Three implementations with thtree different data structures.

1. A teo 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, tou do not know how the data structre is implemented and you can only access the data thriugh interface functions (operations) provided by the ADT

Colletion ADT
It is an ADT designed to story any arbitrary number of item. 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 kknown as a map, association array or symbol table is acollection 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 y 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 anADT of items 'stacked' one of 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 ad text editors.

Typical Stack Operations are

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

STACK ADT Vs RACKET 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 aded 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.

SEQUENC ADT
It is a useful ADT swhen you want to be able to retrieve, insert and delete at any posiiton 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 prgramming 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 functgional 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: Memory & Control Flow

Models of Computing
In CS 135, we remodelled the computational behaviour of Racket with substitutions

In this course, we model the behaviour of C with memory and control flow

Memory Review
A bit of storage has only two possible states: 0 or 1, that is binary.

A byte is 8 bits of storage. Each byte in memory is one of the possible 256states possible.

Accessing Memory
To access a byte in memory you have to go its position which is know as address of the memory in storage.

Sizeof
The space required by every variable depends on its type.
We can use a built-in function called `sizeof ()` to find the size of a type.

The placeholder for `sizeof` is `%zd`. The type of sizeof is size_t

Integer Limits
In C, integers are limited to a certain range which can be used. The limits are defined in the header file called limits and is included as follows: `#include <limits.h>`

The maximum value can be used by `INT_MAX (2147483648)` and the minimum value can be used by `INT_MIN (-2147483648)`. The values are signed values that means it stores the sign for negative values in 1 byte and the value so these are the values. We do not need to memorize these values.

Overflow
If we try to represent values outside the range of the integer limits, we get overflow which is defined as undefined behaviour in the world of computers.

Char Type
Char is the the type we usde to work with one character. IT only takes 1 byte in the memory.

American Standard Code for Information Interchange (ASCII)
It is the system of interchange we use for converting values from integer to char. For eg. `A` is `65` in integer. we can work with character A in two following ways

```
char val = 'A';    //This is the value of A in characters

int val = 65;      //This is the value of A in integers

//Many problems in assignment are solved using one of these approaches so always keep them in m

ind
```

The placeholder for `char` type is `%c`

Sections of Memory
In CS 135, the memory has been divided in 5 different sections.

Low
`CODE`
`READ-ONLY DATA`
`GLOBAL DATA`
`HEAP`
`STACK`
High

Sections are combined into the memory which are recognized by the hardware.

When one tries to access a memory out of the current segement, a segmentation fault occurs

The CODE Section
Source Code: The code written in ASCII characters that are human readable.

Machine Code: The code that is converted from source code to another form that is machine readable.
NOTE: You will work with this in CS 241 which is for CS Majors.

The machine code is then places in the section of memory where it can be executed.

Conversion of the source code to machine code is called **Compiling**

The READ-ONLY & GLOBAL DATA Section

The location of variable depends on how it ahs been defined. If it has been defined as a global variable it is put under global data section, whereas once defined as a constant it is moved into read-only data section.

Control Flow

We use control flow models to model how programs are executed.

Return Address
For each function call we need to remember where the function call was made and then we jump back when th function is over.
Note: This is different from the `return` statement in a function. The return statement return the final value to responsing to the data type of the function, where as every function when called has a return address. So the return address of main is `OS`

Call Stack
The history of every call from the main function is maintained as a stack. This is adifferent stack from the STACK ADT but then the implementation follows a very similar approach. A function call is pushed to the call stack and then popped when returning back to the main function.

Stack Frames
Enteries from the function are pushed to the call stack as frames called Stack Frames. Each function makes a stack frame. It does not matter if it is recursion or na new function

Stack frames store the argument values, local variables and return address.

NOTE: C makes a copy of each argument value and places the copy in the stack frame. This is called passed by value conversion. We will learn another way of passing values called pass by reference but in the next section.

Recursion in C
Each recursion in C is a new stack as mentioned before.

Stack Section (From the memory)
The call stack is stored in this section. If the stack grows too large then we can get stack overflow errors.

Model
At any point a program is in a specific state, which is a combination of The current program location, and the current contents of the memory.

Calling a function
Calling is a control flow:
Whena function is called,
a stack frame is first created and pushed with copy of arguments to the stack, then the current program position is noted and then noted as the return addres and then the initialization occurs.

Return
Whena function returns, the current program location is changed back to the return address. the stack frame is returned.

if Statement

Syntax:

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

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

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

Looping
With mitation, we can control the flow

We have different loops and concepts to take care about.

1. While Loop
2. do...While loop
3. for loop
4. nested loops

break and continue

Unit 06: Introduction to Pointers

Readings: CP: AMA 11, 17.7

Address Operator
The address operator produces the sarting of where the value of an identifier is stored in memory.

The `printf` placeholder to display an address (in hex) is `%p`

Pointers
Type for storing an address is called pointer. Defined by placingg a star `*` before the identifier name.

```
int i = 5;

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

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

Sizeof Pointer
In a k-bit system for type values, pointers are k-bit. For seashell, it is `8-bit`.

Indirection Operator
The operator is `*` star, also known as dereference operator, and is the inverse of the address operator.

The NULL pointer
`NULL` pointer is a special pointer that points to nothing.

NOTE: If you derefrence a NULL pointer your programm will likely crash and show a segmentation fault as the error.

Function Pointers
Function pointers are first class values in C, which means these are values as pointers which point to a function and we can then pass a function to a function as a parameter.

Pointer Assignment

```
int i = 5;

int j = 6;

int* p = &i;

int* q = &j;

p = q


//In the above few lines you just changed what p points to. Now it points to what q points at s

o when you print p it will now print 6.


int i = 5;

int j = 6;

int* p = &i;

int* q = &j;

*p = *q
```

```
//This is known as Aliasing


//Now in this program we changed the value of what p pointed at, now this means i = 6 now. Thin
k and understand
```

It is really important to understand what just happened in the past code. IT is what sets your abse for the rest of the course.

Mutation and Parameters
In C, we can emulate pass by reference by address and then change the values of types without making copy which takes more space than the pass by reference method

Returning more than one value
In C we can also emulate this but since the function only returns one value of a variable then we just return one value and change the value of one variable with pass by reference.

Passing Structures
Large structures if passed by value can cause stack overflow as it copies the whole structure and then make a local copy. To avoid this we use pointers.

So when we send a structure by pointer we only send a pointer which is always lesser than the actual size. In Seashell, it will always be 8 bytes. Imagine a structure with 100 bytes vs 8 bytes as a parameter.

To access, the fields in the structure we can use `(*struct_name).field` or `struct_name->field`
The arrow is **arrow operator** which combinese indirection and selection operator.

Opaque Structures in C
Opaque structures are structures which have not been defined but declared. The programmer will now change it per they want to implement it.

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