CS 137 Part 1 Learning Goals

By the end of this section, students should be able to...

- Throughout all parts of this course, students should be able to solve problems both new and familiar using the techniques taught in the course. This is highlighted as one of the major goals of this course.
- Feel comfortable compiling C code, including compiling code on the student environment or on the shell command line (or in some other suitable format - see the first lab)
- Understand the basics of the `main` function.
- Use functions such as `printf, scanf` and understand what `#include <stdio.h>` means and why it is needed.
- Understand and be able to apply the use of primitive C commands such as `while, if`.
- Understand, be able to apply, and know when to apply the Euclidean Algorithm to help compute the greatest common divisor of two numbers.
- Have a working understanding of how variables are stored
- Be able to print variables using the `printf` command.
- Have a preliminary understand of how the stack works.
- Understand how C treats data types like `int, char, short int, long int, long long int` and their unsigned counterparts in memory (ie, as binary numbers).
- Explain the difference between little and big endian byte orders.
- Convert to and from binary.
- Understand the difference between signed and unsigned integer types.
- Describe how two's complement works and how it is used in computations.
- Be able to read, interpret and write code using incremental operators, including `++, --` both prefix and suffix as well as `+=, -=, *=, %=, /=` and others as needed.
- Understand how relational and logical operators work. This includes `&&, ||, ==, !=, < , <=, > , >=, !` and others as needed.
- Have a working understanding of which operators in C are left associative and which are right associative. Be able to describe the difference between each.
- Be able to read and write code using bitwise operators. These include `&, |, ^ , ~, >>, <<`
- Describe the "dangling else" issue as well as how to nest if statements, use else and else if statements and so on.
- Understand how the ternary conditional operator works and be able to both read and use code involving it.
- (Optional depending on year) Understand how `switch` statements work including identifying fall through errors.

CS 137 Part 2 Learning Goals

By the end of this section, students should be able to...

- Be able to understand code that uses and be able to apply the use of `do while` and `for` loops in C. This includes `break` and `continue` statements.
- Be able to write a program that compute whether or not a number is prime. Discuss ways to optimize such code (for example, using the result that if a number is composite, one of its prime factors is less than or equal to the square root of the number).
- Understand boolean values in C and be comfortable with the basics of the `<stdbool.h>` library.
- Explain to a novice programmer what recursion is.
- Describe the two golden rules for recursion.
- Be able to use recursion to solve basic problems (for example, write a program that performs multiplication using recursion and without the multiplication operator).
- Be able to use `<assert.h>` to flag errors
- Have a working knowledge of the Gregorian calendar.
- Understand how separate compilation works in C; in particular, be able to write header files and understand how to include them in other code.
- Describe the four components of the C compilation process (preprocessing, compiling, assembling, linking)
- Be able to use macros as needed for constant values.
- Understand how arrays are stored in memory (and in particular on the stack).
- Describe and be able to code the Sieve of Eratosthenes.
- Explain how the `sizeof` command works.
- Describe pointer decay in C.
- Explain how multidimensional arrays are stored in C.

CS 137 Part 3 Learning Goals

By the end of this section, students should be able to...

- Use floating point data types such as `float, double`.
- Identify the differences between `%e, %f, %g` flags in printf.
- Be able to use the `%m.px` commands for different values of `x` from `d,e,f,g`.
- Convert a decimal fraction to a binary fraction.
- Be able to identify the relative and absolute error in values and be able to describe a useful application of both types of error.
- Explain some of the many errors caused by floating point numbers and describe some of the solutions.
- Describe the advantages of using Horner's Method over more traditional methods of computing powers.
- Be able to write a polynomial using Horner's Method.
- Describe algorithms for root finding. In particular, explain the Bisection Method.
- Be able to give different stopping conditions for the method and explain the pros and cons to doing so.
- Using the midpoint break condition, compute the number of times a bisection method algorithm might run.
- Explain the fixed point iteration method for finding a root.
- Be able to use function pointers in your code to allow for larger abstraction.

CS 137 Part 4 Learning Goals

By the end of this section, students should be able to...

- Be able to define and use examples of structures in C.
- Identify the pros and cons to using `structs` in C.
- Describe the basic and randomized Page Rank Algorithms. This can include creating an analogy to it, the formulae, and performing basic computations to compute the page rank value in these algorithms
- Explain how to find these page rank algorithms using the iterative method from class. You don't need to perform computationally difficult computations on an exam.
- Code the page rank algorithm.

CS 137 Part 5 Learning Goals

By the end of this section, students should be able to...

- Describe the difference between passing values by value and by reference in C. In particular, be able to explain what happens when you pass arrays, structs, pointers and basic first class data types to function and what happens when these values are returned.
- Explain the difference between the stack and the heap. Be able to give concrete examples of this difference.
- Understand how the address of operator `&` and the dereferencing operator `*` work.
- Be able to read and write code using pointer arithmetic. This include the addition and subtraction of pointers and integers, the difference of pointers, as well as the basic operations from part 1 such as `++, +=, <=` and so on.
- Understand the difference between pointers and arrays in terms of pointer arithmetic (for example, arrays cannot have their addresses changed whereas pointers can have their value addresses changed; also, the address of an array and the value of an array are the same but their values in `sizeof` are different).
- Understand some of the combinations of the dereference operator `*` and the prefix or suffix increment operators `++` or `--`.
- Be able to use `malloc`, `free` and be able to assert that memory has been assigned when attempting to allocate.
- Have a working understanding of the `NULL` pointer.
- Be able to code and understand code created implementing the struct hack.
- Be able to code variable size array functions (such as vectors from class). Note that you are not responsible for memorizing the entire code but may be asked to code up parts or describe the algorithms behind how the vector class should work.

CS 137 Part 6 Learning Goals

By the end of this section, students should be able to...

- Have a working understanding of ASCII. This includes what the null character is, the relative position of numbers, upper and lower case letters, the range of values, ideally one or two values (such as ‘A’ being 65 and ‘a’ being 97) and a general understanding of the groupings of values.
- Be able to perform operations with characters and numbers.
- Describe the difference between string literals and setting up character arrays.
- Describe the roll of the null character in strings. Be able to use this to help loop through strings.
- Understand the use of `const` with variables and pointers.
- Have a basic understanding of the functions inside `<string.h>`. This includes `strlen`, `strcpy`, `strncpy`, `strcat`, `strncat`, `strcmp`.
- Expanding on the previous point, you are not expected to have all of the borderline cases memorized but you should know how these functions work in typical cases. However, if you use these methods, you are expected to make them work in your situation (for example, you would be expected to know that strcpy doesn’t check whether or not the string has enough room to be correctly copied).
- Similarly to the above, students should have a working understanding of `memcopy`, `memmove`, `memcmp`, `memset`.
- Given a UTF-8 chart and a character, be able to convert the value into Unicode.

CS 137 Part 7 Learning Goals

By the end of this section, students should be able to...

- Give the definition of $g(x) = O(f(x))$.
- Use the definition to show the relationship between functions.
- Use $\lim\limits_{x\to\infty} \dfrac{g(x)}{f(x)} \implies g(x) = O(f(x))$ to show the order of functions.
- Identify the runtime of algorithms done in class.
- Identify the runtime of code.
- Describes some of the differences between best, average and worst case runtimes.
- Be able to trace and code selection and insertion sort algorithms. You may or may not be given the code on an exam to do so.

CS 137 Parts 8 and 9 Learning Goals

By the end of these sections, students should be able to...

- Be able to trace and code merge and quick sort algorithms. You may or may not be given the code on an exam to do so.
- Analyze the runtimes of merge sort, quick sort and binary searching. This includes best, average and worst case analyses.
- Describe the median of medians approach to improving worst case runtime behaviour of quick sort. (You need not be able to code this yet).
- Trace and code binary searching. You may or may not be given the code on an exam to do so.
- Describe Tail [Call] Recursion and describe the difference with it and regular recursion.
- Code the Fibonacci sequence recursively or iteratively.
- Describe the process of memoization and be able to use it to improve on an existing algorithm.