CSE13S notes
Week of 1/23/23

Assignment 2

- When linking libraries, put them after the program that needs it
    - Ex: clang ramanujan.c -lm (-lm is the math library)

Numerical Methods

- Computers can only do basic operations such as addition, subtraction, multiplication, and division
    - Multiplication is shift and addition
    - Division is shift and subtraction
    - Addition is little more than XOR
- What about trig?
    - Some processors are capable of doing them internally, but they are still using basic operations
- What about integrals? Transforms? …?
    - These are all numerical methods
- Absolute value
    - We have to make our own

Library???

- In general, it is best to use the library because it is widely used
    - It is also very efficient and fast
- But there is no magic, so we need to learn how functions within the library work

Taylor series

- We will be using taylor series to approximate functions like e^x, sin(x)
- More terms make the approximation better

What happens if a series converges too slowly?

- log(x+1) taylor series converges very very slowly
- However:
    - log(x) = -log(1/x), for x>0
        - Still slow
- Recall that:
    - X = log(e^x) =e^log(x)

Some equations have no series [sqrt(x)]

- Example: sqrt(x)
    - Binary search can be used
        - There are better methods for differntiable functions
- Inverting a function
    - sqrt(n) using newton's method
        - See slides
    - log(x) using newton's method
        - Newton's formula provides linear approximation
        - Solve for x_{n+1}
            - See slides
- Scaling
    - sqrt(4x) = 2sqrt(x)
    - log(y) = log(xe^f)
- Inverse trig
    - Inverse sin instead of using series for arcsin

Dangers of Floating point arithmetic

- See slides
- Comparing floating-point numbers
    - Direct comparison of floating point num should be avoided
        - This is due to small round-off errors
- Can check is the absolute error is a small error

Floating point numbers

- Not real numbers
- Take care when using them
    - Subtracting 2 numbers of different magnitudes can lost precision
- Taylor series can approximate an infinitely differentiable function to arbitrary precision

What should we do?

- Use a good library when you can
    - But understand what is going on!!!

Floating point arithmetic

- In normal math, real numbers are exact
- In computer arithmetic, non-integer numbers are approximations
- Floating point arithmetic results in rounding errors
    - This is because rounding occurs to fit values into finite representation
    - Floating point =/= Q

Round-off errors

- IEEE 754 defines 5 standard rounding modes:
    - Round to nearest
    - Round to nearest
    - Round up (ceiling)
    - Round down (floor)
    - Round to zero (truncate)

Dangers of floating point arithmetic

- Relative error more important than absolute error
    - If you're comparing small things, small errors matter

Arrays in C

- Collection of elements of the same type
- Arrays can have one dimension
    - Called vectors
- Can have 2 dimensions
    - Called matrix
    - C treats matrices as an array of vectors
- More dimensions
    - Called tensors by the machine learning community
    - C treats these as arrays of arrays of … some type
- Arrays in C are ordered
    - In C, arrays start at 0
    - A[i] comes before a[i+1] in memory
        - You can go past memory to a[n], but do not do that!
    - A[i] comes after a[i-1] in memory
        - Does a[-2] make sense?
            - No, its before the end of the array! Except if you mess with pointers
- Declaring an array
    - int a[] = {1,2,3,4,5,6,7,8,9,10};
    - int b[10];
    - Float c[] = {3.1416, 2.7183, 0.57722, 1.6180};
    - If you have a count, then you don't need an initialization list'
- In memory:
    - ASM language shows us what really happens
    - __a is the base address of the array in assembly language
- Matrices
    - In mathematics we write a matrix as m = (imagine an array of vectors)
    - In C, we write: type m[3][3]

        - The elements are m[0][0] to m[2][2]
    - C stores the array in row major order
        - That means one row in memory, then the next row, then the next row
    - In memory (ASM) matrices stores in 2d

Matrix Multiplication

- Check slides

Important! :To a point, you can speed up your algorithm by a constant factor

- Arrays are an exception to the rule that parameters in C are always passed by value
    - This is because arrays can be very large and you don't want to copy them to stack

sizeof() operator

- The sizeof() operator tells us the number of bytes used by a variable
- Works on arrays, structures, unions, when the compiler can know how much memory is used
- When applied to an array, sizeof gives the size of the array
    - However, when appleid to a pointer, it also gives the size of the pointer
        - Are they not the same?

Arrays and pointers in C are related in fundamental and often confusing ways

- For a 1D array:
- A[i] == *(a + 1)
    - A is the address of a[0]
    - A[i] is the array slot that is at a + i * sizeof(a[0])
    - Pointers automatically do the multiplication by sizeof()
- Matrices in C are treated as an array of pointers
    - The way arrays are laid of differs depending on if they are allocated at compile time or dynamically
    - The compiler must make it behave the same

Dynamic Arrays!!!!!malloc, calloc, realloc

- int *newArray(int elements) { return (int *)calloc(elements, sizeof(int)); }

Searching

- The task of searching for an element is very common
- Generally entails traversing through arrays

Ordering the array

- If we put the array in order, we can search it more efficiently
- In ordered arrays, it is very easy to find extrema, since they are either at the beginning or end of an array

Binary search is the fastest method of finding a particular element in an ordered array

String time!!!

- In C, an array is an array of characters that end in '\0'
- It can be written as:
    - Char a[] = "Hello World!"
    - Char *s = "Goodbye Cruel World!"
    - Char a[] = {'L', 'e', 'g', 'a', 'l', 0}
    - Most people use the *s version
- String is an array of characters that everyone agrees ends in (is terminated by) a null character
- Assignment and copying
    - Char *s = "Wally wonder"
    - Char *t
    - t = s
        - The above copies the pointer, not the string. Thus changing t will also change s
    - Let *s and *t be two strings
        - s[i] - t[i] will result in 0 if character i is equivalent since characters have numerical values

For heap sort, to obtain array from 1 to n (instead of 0 to n-1) try setting separate array h equal to array h - 1 (so the pointer's 0th element is before element q)

Pointers!!!!!

- A variable tha holds a memory address
    - The variable points to the location of an object in memory
- Not all pointers contain an address
    - Pointers that don't contain an address are set to the NULL pointer
    - NULL pointer = 0
- Pointers are said to point to the address they were assigned
    - Can assign a pointer the address of a variable using the address-of operator (&)
    - Multiple pointer can point to the same address
- Derefencing a pointer
    - The object a pointer points to can be accessed through dereference (or indirection)
    - You dereference using *
    - Useful for manipulating the values of several variables through call-by-reference
- How to use * operator

    - Int foo = 5
    - Int * bar = &foo
        - Essentially, integer pointer bar is set to address of set variable foo which equals 5.
- Benefits of pointers
    - Can be used when passing actual values is difficult
    - Can “return” more than one value from a function
    - Building dynamic data structures
- Passing by value vs passing by reference
    - Pass by reference:
        - Allows “returning” multiple values
        - Allows passing large amounts of data quickly
- Pointer arithmetic
    - Since pointers in C are just an address, numeric values, you can perform arithmetic on them
    - ++ : increments to next address (increment by 4 bytes assuming 32-bit integers)
    - -- : decrements to previous address
    - + : can only add numeric value to pointer, cannot add another pointer
    - - : if a pointer is subtracted from another pointer, the distance between them is calculated
    - Pointers can also be compared using inequality(< > ==)
    - Multiplying and dividing pointes also doesn’t make sense
- Arrays and pointers are equal
    - For int arr[i]:
        - Arr[i] = *(a + 1)
- Pointers and arrays
    - Array subscripting can also be done with pointers
    - Using pointer arithmetic in general is faster, but harder to understand
    - Assuming some array int arr[10]:
        - Arr[i] is equivalent to *(arr + i), where 0 <= i < 10
    - Arrays can always be written using pointers
        - Declaring an array in a function allocates it on stack
        - A global array is in the data area
        - Dynamically declaring an array (to get a pointer) allocatets it on the heap
- Strings as arrays
    - Strings are handled as arrays, but have some special syntax
    - Strings can be indexed, passed by reference, etc (generael pointer stuff)

- Pointers to pointers
    - Pointers can point to other pointers
        - Or to pointer to another pointer to another pointer …
    - Can be used to pass arrays of arrays, such as a list of strings
    - For example, char **argv
- Function pointers
    - Points to executable code in memory instead of data value
    - *learn more/check slides on this