Review guide

Summery of Lecture #3

Git

- Source Control system
- git clone - copys of remote repository
- git add - adds files to file in repository
- git commit - issues a commit
- git push - sends your local changes to remote repository
- git pull - brings local rep up to date with remote repository
  - ↳ merges

Summary of Lecture #4

include → includes libraries

- cc -o helo hello.c → compiles
- ./blah → runs
- in every main program, there is only one main function
  - ↳ returns an int
- {} are used to group statements
  - ↳ blocks
  - ↳ local scopes

while loop: top-test loop

↳ while (whatever is here is true) it will execute

for: initialises a variable, increments & runs the condition all in one

Char * s = string
Char c = small int
int = int
float = floating point num
double = more precise decimal num
- Scopes (§3) tells us where the variables exists
  - hide variables sometimes
if - if the boolean exp is true, execute

Summery of lecture #5
- diff kinds of numbers (ways to represent)
N set = 1, 2 - . .
Z set = - 3 - - 3 - . .
Q set = $\{\frac{a}{b} = a, b\}$
r set = irrational num
C = r + the imaginary num

bits	min	max
8	-128	127
16	-32768	32767
32	-2147483648	2147483647
64	-9223372036854775808	->

Lo signed num

Unsigned num

bits	min	max
8	0	255
16	0	65532
32	0	4294967295
64	0	18446744073709551615

- computers use 0 or 1
- digit = bit

unsigned { short → 16, long, long long } { int → at least 16 (short), at least 32 (long, long long) }

unsigned { char - 8 bits }

signed	unsigned	size
int8_t	uint8_t	8
int16_t	uint16_t	16
int32_t	uint32_t	32
int64_t	uint64_t	64
0+0=0	0x0=0	
0+1=1	1x0=0	
1+1=10	1x1=1	

if we want to add inverse of a number, do two's complement

+	-
0000	0000
0001	1111
0010	1110
0011	1101
0100	1100
0101	1011
0110	1010
0111	1001

real numbers are continuous & uncountably infinite

floating point number are proper subset of ∈ real num, rational nums, ints

→ simply an approximation not reals & rational

Type	name	implementation	bits
float	single	IEEE 754 single	32
double	double	754 double	64
long double	extended	754 quad	128

↳ This is how floating point FP are "represented"

float ∋ Q

1 = sign

11 = exponent

52 = frac(11)

what is the address of the byte holding the least significant bit

little endian = low address byte

big endian = high address byte

big endian

| 12 | 34 | 56 | 78 |

little endian

| 78 | 56 | 34 | 12 |

modulo = remander when you / two ints

@ / % + - << >> <= >= == !=

& ^ | && || ? → left

?: = += -= *= /= %= → right

Lecture # 6 Summery

∧ = and ∨ = or ¬ → not

∧ → && ∨ → || ¬ → !

A ∧ (B ∧ C) = (A ∧ B) ∧ C

A ∧ B = B ∧ A

A ∧ (B ∨ C) = (A ∧ B) ∨ (A ∧ C)

A ∧ 1 = A

A ∧ 0 = 0

A ∧ A = A

A ∨ (B ∨ C) = (A ∨ B) ∨ C

A ∨ B = B ∨ A

A ∨ (B ∧ C) = (A ∨ B) ∧ (A ∨ C)

A ∨ 0 = A

A ∨ 1 = 1

A ∨ A = A

A ∧ ¬A = 0

A ∨ ¬A = 1

¬(¬A) = A

¬(A ∨ B) = ¬A ∧ ¬B

Exclusive or
$A \oplus B = (B \vee A) \wedge \neg (A \wedge B)$
or $(A \wedge \neg B) \vee (\neg A \wedge B)$
switch = allows you to select among a fixed set of alternatives

Lecture 7 summery

- while = top-test → evaluated before entering the loop
- do-while = bottom-test loop

Lecture 8 summery

- mult = shift & add
- DIV = shift & sub
- floating point numbers are not real #s

Lecture 9 summery

- function = returns a value
  - block of code that performs a task
  - can return anything but an array
- c uses call by value
  - ↳ copy of the actual parameters is place in formal parament
- c does not use call-by reference but passing a pointer accomplishes this
  - changes have actual effects
    - ↳ address are passed as arguments

recursion: when a function calls itself

Lecture 10 Summery

- array: collection of elements
- can have mult dmensions
  type name [size] = {0}
- stored in memory
  ↳ in row masor order
- array = = pointers
  ↳ pased by reference

& = gives addess of a variable

sizeof = num of bytes used by a var

matrix = array of pointers

- str = array of char that end in \0

char s[] = blan or char *s = "blah"

or char s[] = {'L'...}

Lecture 11 Summery

pointer = var that holds a memory address.

- NULL pointer = 0
- bytes ae grouped into words
- mult pointers can point to the same thing

* = derefrences a pointer

passing by values = duplicates passed value onto the stack

passing by ref = duplicates a pointer onto the stack

↳ returns mult value

array delared in a function → stack
global array → data area
dynamically declaring an array → heap
strings are handle like arrays

Lecture 12 Summery

shift left = zeros shifted on the right
shift right = shifted in to the left
arithmetic shift left: zeros are shifted in on the right
arithmetic shift right: sign bits are shifted to the left
& = and | = or ~ = not ^ = xor << = left
>> = right

sets = unordered collections
∩ = intersection ∪ = union − = diff
$\bar{A}$ = complement

Lecture 13 Summery

- putting things into a defined order
O(n²) algorithms
- bubble sort, insertion sort, selection sort,
- quick sort (worst)
- O(n^3/2) → shell sort
- O(nlog n)
  - merge sort, heap sort, quick sort

Lecture 14

-dynamic = memory allocated at runtime
  ↳ on the heap
-use malloc, calloc or realloc
  ↳ return a pointer to allocated mem
- must be free!

heap = memory can be accessed with access to pointers

malloc may contain junk data
- doesn't check for overflow
-calloc zeroes out content
realloc = re allocate new memory or extend

regular expression summary
- search pattern
  - . any
  - + 1 or more
  - * zero or more
  - ? one or zero

Linked list summary
- linked structured
- no fixed memory allocation

sentinel nodes -> mark ends of a LL

hashing & bloom filters

takes a key & makes a # w/ it
↳ uniformly distributed
linear search -> look-ups of unsorted are O(n)
binary search with sorted -> O(log(n))
hash tables - unordered collection of key-value pairs
↳ ex python dictionary
hash collision -> two pieces of data have the same hash value using LL
↳ can be limited by chaining, double hashing, quadratic probing, linear probing
↳ open addressing
bloom filters
↳ tells you if an element is present
↳ can add & search
- memory efficient
- the more hash function, the less chance of a false positive
-> time & space complexity of O(k) & O(m)

process lectre

process → code, data, stack
add space → programs execute code
memory hierarchy = L1 (cache) → CPU chip
L2 on low chip
L3 cache = off-chip, sram

process states

→ created, ready, running, blocked, exit

file lectre

- memory accessed using add
file operations → create, del, open, close, read, write
root directory → user directories → files

Language translater lectre

- pre processor → compiler → assembler → linker
stack - local variable, return add, arguments, return values
heap → dynamic memory allocation
process of a C program
source code → processor → compiler → assembler → linker → executable

Trees & tries lectre

- consist of a node with sub trees
in a binary tree if a node → a child → it is known as a leaf
O(Log(n)) for inserting
tries → ordered data struct