16/8/21 Notes,

Arrays
- homogeneous collection of elements
  - all elements have the same type.
- Arrays can have one dimension
  - often called a vector
- Arrays can have 2 dimensions

Arrays are ordered
- collection of $\vec{a} = \langle a_1, \ldots a_n \rangle$.
- In C arrays start at zero
  - a[0] is the base address of array
- a[i] come before a[i+1]
- a[i] come after a[i-1] in memory

Declaring an array,
type name [const] = {initializing list};
- if you have an initialized list, you don't need a const.

In memory
- assembly language shows us what is really happening
- a is the base address of the array in assembly language

Matrices
- 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 next row, then the next
- FORTRAN is the only language that stores arrays in column major order.

In memory
- memory is one dimensional
- rows are laid out sequentially

Matrix ~~Manipulation~~ multiplication
- a, b, and c are all matrices
  - each elem is floats

$$c_{i,j} = \sum_{k=0}^{mid} a_{i,k} b_{k,j}$$

How are we changing c[][]?
- arrays are the exception rule that parameters in C are always passed by value

Why
- arrays can be large, so you don't want to copy them onto the stack
- Arrays and pointers are intimately related
- the name of array is just a pointer to element 0 of the array.
  - this is the base addr,

& (address of) operator
- & give the memory location of a variable

sizeof operator
- tells us # of bytes used by a variable
- work on array, structure, union, when compile can know how much memory is used

Is this what you expected?
- sizeof when applied to an array, gives the size of the array
- sizeof when applied to a pointer, gives size of the pointer

Pointer and array,
both are related in certain way

For a single dimension array,
a[i] == *(a + i)
- a is the address of a[0]
- a[i] is the array slot that is at a + i * sizeof(int)
- Pointer arithmetic do the multiplications by sizeof.

Array and pointer are equivalent in C

A matrix is an array of pointer
- if the matrix is allocated at compile time, it is laid out contiguously.
- if you allocate it dynamically it is an array of pointer to vectors.
- Compiler must make it behave the same.

Searching
- task of searching is common
- often entail, traversing a collection of elements, such as an array
- collection of elements need not be in order.

Find an item
- return the first index of the key if it exists
- return -1 on failure.
- the must look through entire array

Ordering the array,
if we put array in order, then we can do efficient sorting

What is a string?
- an array of characters that ends in '\0'
- written as:
  - char s[] =
  - char *s =
  - char s[] = {'C', ...}
- most people use *s

```
 0110
 0101
------
 1011
```

Assignment and Copying
char *s = "test";
char *t;
t = s;

- means s and t point to (reference) the same memory location
- it does not make a copy of s,
- sizeof does not give length of string.

strcmp()
- As long as both chars are not null, subtract them
  - < 0 means s < t
  - > 0 means s > t
  - = 0 means s == t

strlen()
strcpy()

10/11/21 Note

Computational Complexity (Simplified version)

Who's counting?

- Size of problem:
  - how long is the string
  - number?
  - how many items are in the ~~set~~ set?
- size of input

Some common functions

$\log(x)$

x

$x\log(x)$

x^2

x^3

2^x

$x!$

Examples	Order	Example
	$O(\log(n))$	Binary
	$O(n)$	Find min
	$O(n\log(n))$	merge sort
	$O(n^2)$	Bubble
	$O(n^3)$	matrix multiply
	$O(2^n)$	enumerate subsets
	$O(n!)$	

Formality
- When we say "Bubble Sort is $O(n^2)$"
- Once x_0 is large enough, there a some constant c where $f(x)$ is always bounded by $c \cdot g(x)$:
  - $f(n) \leq c \cdot g(n)$

Look for loop when figuring out complexity

Estimating Complexity
- look at the loops ~~the~~ and recursion
- add complexity of loops at the same level
- we multiply the complexity of loops nested inside of loop
  - one loop usually results in $O(n)$
  - a nested loop is $O(n^2)$

c?
- c is the overhead that an algorithm requires

So what about recursion?
factorial → $O(n)$

q

10/13/21 Sorting

Sorting
- keeping things in a defined order

First idea
- Enumerate all of the possible orderings of n objs. } bogosort
  - There are n! such orderings.
- Pick the one that is in order.

Second idea..
- Find the smallest item; look at all not seen
- Put it in the first slot
- Swap.
- once smallest item is found,
- repeat same process for smaller and smaller subarrays.

selection

Third idea
- if array only has one element → sorted
- if second element exist, it must go either before or after first element
  - when a third element exist
    - must go before first item, b/t 2 elements or after second
- proceed for 4, 5, ... elements

Fourth idea
- look at first 2 elements, if first is greater, swap.
- move onto second element, swap.

Is that the best we can do?

Merge sort -
- Suppose I examine my array in disjoint pairs a[0], a[1], a[2], a[3]
  - will have n/2 such pairs, but we have to look at both
  - will get element of every pair in order (ie. all smaller)
- pairs of runs, each of length 2, merge them into run of len 4.
- How many times can we do double I before we exceed n., log n
- sort finishes in

Comparing Sorting algorithms,
- $O(n^2)$ algos
  - Bubblesort
  - insertion
  - selection
  - Quick sort
- $O(n^{4/3})$ algos
  - Superslowsort
  - shell sort ←
- $O(n \log n)$ algos
  - merge sort
  - heap
  - quicksort

Heaps
- min/max heap
- leftist heap

A (Max) Heap
- a single node is a heap.
- heap if parent is a heap and the trees rooted at both children are heaps
- a parent's value (key) is greater than that of either child.
- in order going to children

10/15/21 Notes Pointers and Dynamic memory

Pointer.
- a variable that holds a memory address
  - points to the location in memory
- must all pointers are given an address
  - these are set to the NULL pointer.
  - NULL = 0
  - NULL is a macro for either:
    - ((void *) 0)
    - 0
    - 0L
    - definition depends completely on the compiler

Memory address
- memory is stored in registers that can be accessed by a specific

Pointers and addresses
- pointers are used to point at the address the type assigned
- & can point the address & vice versa, &
- multiple pointers can point to same address,

& operator

Dereferencing a Pointer
- a pointer pointing can be accessed through dereferencing
- pointer can be dereferenced copy the of data
- useful for manipulating the value of several vars through call-by-reference

Pointer have addresses too

Benefits of pointer
- Can be used when passing value is difficult
- Can return more than one value from a function
- Building a dynamic data structure
- useful for passing large data structure

Pass by value vs. pass by reference
- passing by value → duplicate value onto stack

passing by reference:
- allow returning multiple values
- allows passing large amounts of data quickly
- you're not copying the data just telling

Pointer arithmetic
- ~~Since pointers~~
- ++: increments to next address
- --: decrements to previous address
- +: can only add a numeric value to a pointer
- -: if a pointer is subtracted from another pointer, the distance b/t both addresses is calculated
- Pointers can also be compared using relational operators

Pointer arithmetic
- offset a pointer by adding/subtracting an integer
- can get the number of elements b/t 2 pointers by getting diff.
- cannot use divide, or multiply 2 pointers

Does it make sense?
- adding int makes sense
- subtracting int makes sense
- adding 2 pointers makes no sense
- mult/divid w/ pointer never makes sense

Pointers and arrays
- using pointer arithmetic is faster, but harder to understand
- arr[i] is equivalent to *(arr+i), where 0 <= i < 10
- Arrays can always be written using pointers
  - declaring an array in function allocates it on stack
  - a global array is in the data area.
  - dynamically declaring an array (to get a pointer) allocates it on heap

Strings as arrays
- In C, strings are handled as arrays
  - w/ some special syntax for certain
- A string is a pointer to an array of chars
- String can be indexed, passed by reference, etc.

Pointers to pointers
- a pointer can point to other pointers
- can be used to pass array of arrays

Multidimensional Array
- if data is of same type, and each sub-area of same length, a multidimensional array is appropriate
- can be any dimensionality

Function pointers
- points to executable code in memory instead of data like
- dereferencing a function pointer yields the referenced function
- Notice how there are parentheses around the function pointer
  - without them, the function would be parsed as a function declaration.

10/18/21 Note

Dynamic memory allocation
- act of allocating memory for variable on the heap during program runtime

why is it good?
- stack space is limited
- we want vars to last beyond the life time of its current scope
- Var on stack don't do

Dynamic memory allocation
- There are 3 standard C lib functions to allocate memory:
  - malloc, calloc, realloc
- These dynamically allocate memory on the heap, and return a pointer to the allocated memory
- allocated memory must be freed using free(), not freeing can lead to a memory leak

The heap
- a large region of unmanaged, anonymous memory.
- only limitations are computers physical limitations.
- slow to read from/write to due to the need of fn pointers.
- Vars using heap can be accessed globally w/ alloc pointer
- possible memory fragmentation.

sqr is assigned
result is complex

malloc():
- void* malloc(size_t size)
- returns a pointer to [size] bytes of uninitialized memory allocated to the heap
- The memory allocated by malloc() may contain junk data
- What happens when 0 is passed as the specified # of byte to allocate is implementation defined.
- Doesn't check data for overflow of [size] if the result of an arithmetic operation, unlike calloc()

calloc():
- void* calloc(size_t nmemb, size_t size);
- nmemb denotes # of objects and [size] size of objects
- returns a pointer of zeroed memory

realloc():
- Defined as:
  - void *realloc(void *ptr, size_t size)
- reallocate [ptr] to newly allocated memory on the heap
- realloc() deallocate the old obj. for pointed to by [ptr] and return a pointer to [size] bytes of allocated but uninitialized memory
- if [size] > size of originally allocated block of memory, contents contain unspecified values from old block.

stack_create();

```
Stack * stack_create(void){
    Stack *s = (Stack *) malloc (sizeof(Stack));
    s->size = MIN_STACK;
    s->top = 0
    s->items = (item *) calloc (MIN_STACK, sizeof(item));
    return s;
}
```

free()

- defined as free(void *ptr)
- de allocates memory space pointed to by [ptr]
- memory leaks occur if allocated memory isn't freed.
- segmentation fault/core dump can occur if a program tries to access memory location that isn't allowed to access
- pointers that have been freed should be set to NULL to mitigate use-after-free vulnerabilities.

Detecting Memory Leaks and Segmentation Faults

- mem leaks and seg faults can be detected with gdb, infer, or valgrind
- should be used in tandem when debugging

valgrind
- collection of dynamic analysis tools.
- usually used for its memcheck tool, detects:
  - uninit memory
  - read/write memory after it has been freed
  - reading/writing off the end of allocated blocks of mem
  - reading/writing inappropriate areas of stack.
  - memory leaks.
- command line option:
  - --leak-check=full
  - --show-leak-kinds=all

Static vs. dynamic analysis:
- static
  - analyzes the source code for a program before it is run
    - code is compared against a set of coding rules for bugs.
    - only surface level.
- Dynamic analysis, like valgrind, operates by tracking down errors that occur during program execution

infer on a debugged program
- false errors or infer must be read README.md

10/20/21 Notes Recursion

A recursive function is defined in term of itself

Sum the first K natural number.
- $\Sigma k = k + \Sigma(k-1)$ and $\Sigma 1 = 1$, or
- we can loop through and add them up, or
- we can use Gauss's formula.

Fibonacci number.
- the recursive algo runs in $O(2^n)$
- for loop runs for $O(n)$

Are recursive algorithms efficient?
- a function call requires creating a stack frame, and that takes time and space
- all tail recursive functions can be rewritten as iterative (often by the compiler).

When do I use recursion?
- when it makes sense.
- Do not use it when it doesn't make code clearer

Binary search
- done in $O(\log(n))$ time.
- if list is empty, then not there.
- if key is smaller, look in the left half.
- if " " larger " " right

Are fast at array search?
- $O(\log(n))$.

String Table
- Compilers have to keep track of a lot of strings.

Recursion is natural for search
- we use recursion for searching by dividing the space up.
- Places we have searched
- Places we have yet to search
- If we get stuck, we can back up and try a different path

Questions to ask
- check if matrix has a weight in dfs.
- how to construct a struct w/ pointer function
  * stack_create
- would you use malloc or calloc

File parsing

fgets(first line)

↓

allocate a char array of ~~strlen~~ n size

	0	1	2	3
0	0	0	6	8
1	3	0	0	0
2	0	0	7	0
3	0	0	0	9

DFS(0):

↓

10/22/21 Notes Graphs

Network Routing
- The internet is a graph
  - nodes = computers
  - edges are connections

Formal definition
- $G = \langle V, E \rangle$
- defined by its vertices and edges

- V is set of vertices
  - $V = \{v_1, v_2 \ldots v_n\}$
- E is the set of edges
  - each edge is a tuple of vertices

Directed and undirected graphs
- edges may have a direction $n_1 \Rightarrow n_2 \Rightarrow$ directed graph
- undirected $n_1 \Leftrightarrow n_2$

Representing a graph
- Adjacency matrix
  - nxn matrix
  - binary: edge present or absent
  - weighted: $n \neq 0$
- Adjacency list
  - column array of for nodes

Adjacency matrix
- a non-zero entry in M_{ij} means there is an edge $n_i \to n_j$
- matrix is symmetric about the diagonal represents an undirected graph

A graph ADT (
- use dynamic 2d matrix
- O(1) ⇒ adding edge

Adjacency list
- each node is represented as an entry in a dynamic array
- each entry is the head of a linked list
- list elements contain:
  - destination node
  - weight of the edge
- prefer over matrix?
  - matrix is $O(n^2)$ space.

Basic graph algorithms
- 2 ways of searching a graph:
1) breadth-first-search
   - uses a queue
   - explore the set of vertices immediately reachable
     - repeat process for each vertex in set.
   - also known as "level-order" traversal
2) depth-first search
   - uses recursion or a stack
   - search as far as possible before backing up
   - we will often use DFS using a stack

Trees
- Tree are form of acyclic Graph
  - acyclic → it go follow edges, there are no loop.

Topological Sort
- linearly orders vertices in a DAG such that, for any directed edge $\in \langle u, v \rangle$ u appears before v.
- Typically used to indicate ordering of dependency

Single-Source shortest path (SSSP)

Hamiltonian Path.
- A path is an undirected or directed graph that visit each vertex exactly once.
- must start from an origin vertex and end at that origin

10/25/21 Notes Stacks and Queue

Arrays
- allow random access
  - each element can be accessed directly in constant time

Linked Lists
- allow sequential access
  - elements can be accessed in sequential order

Stacks and Queues
- operate as containers for other data types
- Both have
  - well defined semantics
  - fixed set of operations
  - distinct ordering for their elements

Stacks
- objects are added and removed using last-in-first-out order.
- capacity of a stack is the # of stack elements that will fit.
- capacity can be increased, so # of elements increases through reallocation
- often implemented w/ arrays, but other possibilities exist as long as the semantics are preserved.

Stacks using linked lists
- stacks can be implemented easy linked list

Queue.h
- an ADT
- we need:
  - ahead, tail, and a size
  - a pointer to an array
- a constructor
- a destructor
- empty and full predicates
- enqueue - inserts into queue
- dequeue - removes from the queue

Unbounded queues
- implemented w/ a list

Stack vs. Queue
- removing elements

Queue implementations:
- uses vector-based data structure
- some implementations are more-to-fun

Priority Queues
- every element has a priority associated w/ it
- element of higher priority is dequeued before elements w/ lower priority
- In the case of 2 elements of the same priority preference is given to the position of the element in the queue

What to start
- stack
- pq
- node
} start with these

top;
array of nodes;

- code → tells us how to understand bits
- i.o

Lecture Notes
Bit vectors and sets

Units of Information
- Bit — size → 1 — Value → 0/1
- Nibble → 4 bits → Hex digit
- Byte → 8 → Ascii
- Half-word → 16
- word → 32
- long word → 64

Logical shift

Arithmetic shift
Arithmetic shift left: zeros are shifted in on the right
" " right: sign bits are shifted in on left

Bitwise operations in C
- &

Vectors right shift (>>)
- result of V1 >> V2 right-shifted V2 bits

Bitwise left shift (<<)
- v1 will be shifted v2 bits
- if v1 unsigned

& Truth table

&	0	1
0	0	0
1	0	1

(|) OR Truth table

|	0	1
0	0	1
1	1	1

^ XOR Truth table

^	0	1
0	0	1
1	1	0

~ NOT

Getting a high-order nibble
→ most sig. 4 bit
→ bit shift right 4 times to so that high nibble takes place of the low-order nibble
→ AND w/ 0x0F

Setting a high-order nibble
→ AND byte w/ 0x0F
→ Bit shift nibble left 4 times

Sets
- well defined unordered collection that are characterized by elements they contain
.

Setting a bit

Clear a bit
- mov
- not
- &

fix heap = no more shifting

store store bytes in bit array and then and from there through each bit or check

last

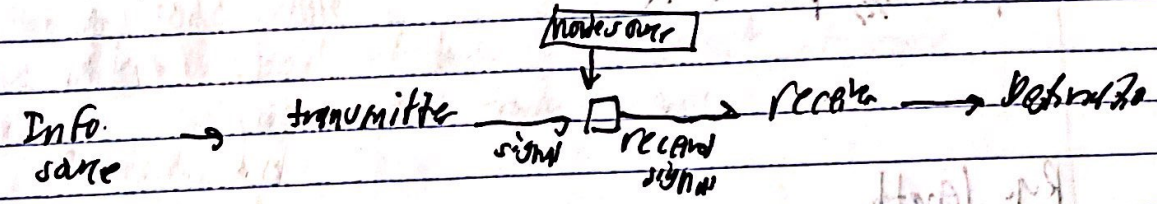
4 bit → 1111 0000

zero fresh out

10/24/21 Note
Data Compression

Communication
- 5 parts
  - information source
  - Transmitter
  - Channel
  - Receiver
  - Destination

Info source
- produces a message, or a seq of messages, to be communicated to the receiver
- A message takes on various forms



Channel
- medium through which a signal is transmitted
- Could be, but not limited to
  - radio
  - Beam of light
  - Coaxial cable
  - wire
  - Fiber optic

Entropy
- measure of uncertainty of the occurrence of an event
- Assume a set of probs $p = (p_1, p_2, \dots p_n)$
- if a measure exists, call it $H(p_1, p_2, p_3, \dots p_n)$ → has properties
  - H is cont. in p_i
  - If all $p_i = 1/2$ then H is an increasing

$H = -\sum_{i=1}^{n} p_i \log_2(p_i)$ ⇒ entropy

Consider: AABL
- $p(A) = 1/2$, $p(B \text{ or } L) = 1/2$

Run length
- Idea: split up a message into sequence of identical symbols
- These seq

Huffman Coding
- developed by Huffman
- Idea: Assign each symbol a unique bit string code
  - generate histogram of unique symbols in data,
  - more frequent a symbol, shorter code

Priority Queue,
- each symbol w/ a non-zero freq is added to a priority queue & ordered
- lower freq, higher priority

Building a Huffman tree
- we will create a tree using nodes in priority que
- while queue has more than one node
  1) pop queue node, this will be left child.
  2) " " another node, right child
  3) create a parent node for left and right child nodes
  4) Enqueue parent node
- last node is root of tree

Building the Code table
- we utilize a stack of bits to keep track of code.
- to build code table, post-order traversal
  - left-child = 0
  - right-child = 1
  - reach leaf node the code for the node's symbol is the code in bit stack

Dumping the tree
- perform a post-order traversal of the tree.
  - if a leaf is reached, output L followed by node's symbol
  - internal nodes are labeled w/ I
- # of symbols representing tree dump = (3 × leaves) - 1

I = pop
L = push

11/1/21 Notes Make

What's the make program?
- utility on most Unix systems that builds executable program and lib. from source code.
- cmake → cross platform build system gen, is a program that produce a Makefile
- has ton of command line flags/ options to select from
- requires a makefile

Makefile
- Plain text file that contain instructions to make
  - has syntax like a program lang.

What's a rule
- composed of:
  - target
  - set of dependencies
  - set of commands
- general struct of a rule:
  - target ... : dependencies
    <tab> command

What's a target?
- the name of a rule or node
- use "make <targ-name>"
- Usually name of the file that is generated by creating the rule's command, but can also be a name of an action to perform
- if targ isn't specified:
  - designate a default make

Phony target
- a target that doesn't produce a file w/ the same name.

Conventional phony targets
- clean:
  - remove any file produced by running a make command
  - only remove exec and object files

Variables in Makefile
- 4 types of variables
  1) lazy assignment
  2) := immediate
  3) ?= conditional assign
  4) += concatenation
- to use value variable:
  - $(var_name)

Lazy assignment
- lazily assigned vars, the "=" operator can called recursively expanded vars

Immediate assign:
- vars immediately assigned w/ ":=" ops. ex

conditional assign
- "?=" behave like "=" except assignment occurs iff var hasn't been assigned a value yet.

Concatenation - "+="
- ex. other like "+="

Dependency
- either a target or a file name
- ex. build:

Topological ordering
- think of targets in a Makefile as nodes

Command:
- an action to be executed

Commands, Compilers, and Compiler Flags
- Vars are used to factor makefiles to make them easier to maintain

Automatic Variables
- "$@": name of the target
- "$^": list all dependencies for targ
- "$?": list of dependencies more recent than target
- "$<": name of first dependency

The shell function
- communicate w/ the world outside of make,
  perform command expansion - takes a shell command
- newlines are converted to spaces

patsubst
- formatted as $(patsubst pattern, replacement, text)

%
w. %
$(wildcard *.c) performs wildcard expansion to get all source file
% can be used to match all text leading up to .c
".c" is then replaced by ".o" in all instances of
matches found

Pattern matching
- utilizes pattern match placeholder "%"

Recursive use of make
- useful for when you want separate Make files for
various subdirectories that are part of a larger system
-

Code = size ⟹ top

empty → top = 0

code full → top == Alphabet

push and pop use [c → top]

ed HZ 91 for Headers

ASK ABOUT

- if pass the ADS psyb are graded or not.

10/3/21 Notes

Long term info storage

- We want large amount of data
- info stored must survive process termination
  - Memory (DRAM) does not survive turning off
- files are accessed using name, memory is accessed using address
- computer has millions of files
  - data center has billion of file
  - the internet has trillion of file

File naming

- Unix → extension is only used by convention

File operations

- Create
- Delete
- Open
- Close
- Read
- Write
- Append
- Seek
- Get attribute
- Set attribute
- Rename

sex is assigned
gender is complex

10/5/21 notes

Linked Lists
- a linked list; each node contains a data field and a pointer to the next node in the list

Linked Structures
- Linked trees
- Trees
- Tries
- Graphs
- Sparse matrices
- Lists

Advantages
- no fixed memory allocation
  - grow and shrink @ run-time what pre allocated memory
- Insertion and Deletion
  - no need to shift elements after insertion

Disadvantages
- Memory usage
  - storing pointer to next node requires extra memory
  - Arrays are friendlier to processor cache.
  - less efficient than arrays
- Traversal
  - cannot randomly access elements, must traverse all elements up to the element we want to access
  - reverse is difficult in a singly linked list
    - easy in doubly linked list but uses extra memory to store an additional pointer.

Doubly Linked List
- each node has a pointer to both the previous and next node
- allows traversal in 2 directions
- less memory efficient than a normal linked list

Sentinel nodes
- designated dummy nodes used to mark the ends of a linked list
- In a doubly linked list, sentinel nodes are placed at the head and tail.

Lookup
- Walk the link list to look for a specific key.
- Linear time search complexity for singly and doubly linked lists.

Linked List stacks
- stack size is limited only by available memory
- pushing an element is inserting it at the head.
- Popping an element is removing it from the head

Linked List Queues
- add a tail
- Remove at the head.

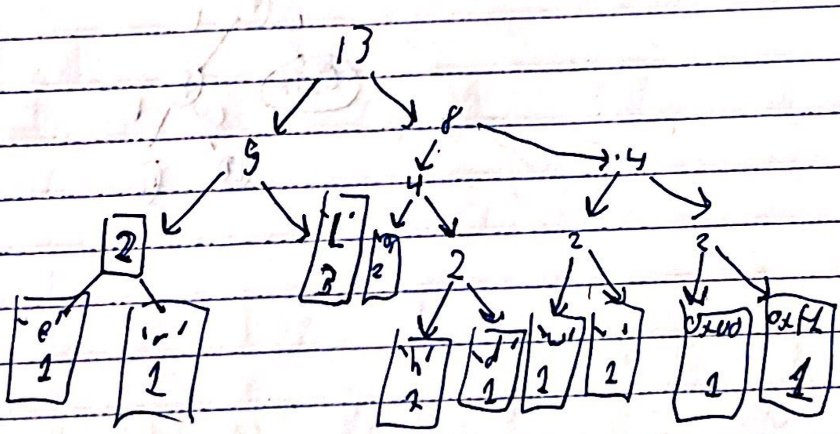
hello_world h e ll o _ w o r l d

Prepending

- prepend a node n to the list.
- if both the head and tail are NIL:
  - the only node in the list is the node to append
- else
  - node before n is the tail
  - node after tail is now n.
  - new tail is n.
  - There is no node after n.

Popping

- Disconnects and returns the head of the linked list



LeLrIILLI

11/8/21 Notes .Trees

What's a tree?
- a tree is a type of directed acyclic graph, typically connected of nodes.
- There is exactly one path b/t any 2 node.
- The definition on the right:
  - a tree can either be NULL
  - or a node pointing to 2 trees.
  - rooted.

Node
- smallest entity in a tree
- contains some value
- Binary tree.
  - has 2 children
  - some don't track the parent
- K-ary tree:
  - each node has up to k children
  - a 2-ary tree is a binary tree.

Terminology
- Root
  - start of the part of the tree.
  - If NULL, then tree is empty.
- Parent → node that points to child node.
- child → node connected to parent node = root of subtree.
- Subtree
- leaf
- Traversal
  - Successor
  - Predecessor

Level Order Traversal
- same as BFS in a graph
- Nodes are visited level by level
- Uses a queue

Terminology.
- Successor
  - next node in some order
- Predecessor
  - previous node in some order
- Both are well-defined for pre-order, in-order, and post-order
  - we will use in-order for binary search trees.

Binary Search Tree
- an example of an ordered tree
  - nodes in a tree do not necessarily need an order.
  - But it is more useful if there is an order.
- Keys less than a node's value go under its left subtree
- Keys greater than a node's value go under its right subtree.

Binary Search Trees in C
- API:
  - create empty tree
  - find min node in tree
  - find max node in tree

Finding minimum.
- Start from root
- walk all the way to the left.

Finding max
- start from root
- walk all the way to the right

Time Complexity for finding extrema
- depends on the balance.
- Balanced tree:
  - height of its left subtree differs by at most 1 from the height of its right subtree
  - for balanced: O(log n)
  - For unbalanced tree: O(n)

Finding a key
- 3 case:
  1) Current node's key greater than key
  2) Current node's key less than key
  3) Current node's key matches key.

Case (1) recursively find the key in the left subtree
Case (2) recursively find the key in the right subtree
Case (3) we have found the node of the key.

Inserting a new key

Removing a key
- the trickiest of all operations
- follow DFS to find node containing key
- 3 cases:
  1) node to remove is missing left child
  2) node to remove is missing right child
  3) node to remove has 2 children.

11/10/21 Notes Scripting

Scripting lets the lazy be productive

- Operates on high level objs.
  - Files
  - Processes
- Software tool philosophy lets us:
  - connect the output of one program to the input of another program. (mathematical)
  - This is program composition (like $(f \circ g \circ h)(x)$)

Who are you calling lazy?

- Smart programs reuse code, programs, components
- If I can use sort to sort a file, why should I rewrite it?

From sh to zsh

- sh was the original UNIX shell
  - called the Bourne shell
- There are many shells
  - csh, tcsh, bash, zsh
  - you can write your own!

BASH

- Bourne Again Shell
- Shells:
  - act as command interpreters.
  - Allow users to give commands to the OS either
    - Interactively: at a prompt or command line
    - Batched: as a script (a file containing a sequence of commands)

Commands
- Bash reads some input (usually a terminal or a file) line-by-line
  - each line is treated as an entire command
  - each line is split into different tokens, or words, by whitespace
  - first word on a line is the command to execute w/ the following words as arguments for command
- Examples
  - $ ls -a
    - ls is command to list file
    - -a to list file prefixed w/ . (dot)

Types of commands
- Aliases
  - Only used in interactive shell
  - Aliases are replaced w/ what they are aliased before a command is executed.
- Builtin
  - Bash-provided commands, like such as printf, echo, and cd
- Functions
  - A sequence of commands that perform some task.

$PATH

Writing a script
- a script is a sequence of commands in a file (usually made executable)
- the first line of a bash script should be the interpreter directive
  - Also referred to a hashbang or shebang.
  - Indicates which interpreter to interpret the script w.
  - #!/bin/bash - interpret script with /bin/bash.

Basic dot files
- Dotfiles are files prefixed w/ a dot (.) that sit in your home directory.
- There are 2 that directly relate w/ Bash (there are others for different shells)
  - .bashrc is a script that is executed

Parameters & Variables
- used to store data
- A variable is a kind of parameter.
  - The first part is the name by which the variable is distinguished from each.
- Variable name can only consist of letters, digits, and underscore, or

Expansion.
- Can expand parameters using the expansion operator: $.
- variable expansion substitutes the variable w/ its value.

Arithmetic
- Arithmetic operations are performed in ((...)).
- The result of these expressions can

Special parameters
- Variables are named parameters.
  - Special parameters are parameters that are not visible

Functions
- a sequence of commands
- can have multiple functions w/in one script.
- Allows for local variables
  - Prevents overwriting variable from the function caller's scope.

Arrays.
- An array is a list of strings.
- Most

Globs and patterns.
- Globs: patterns used to match strings
  - * - matches any string including the null (empty) string
  - ? - matches any single character
  - [chars] - matches any characters in chars
  - [a-z] - matches any characters b/t a and z
- Echo number of text files, JPEGs, and PDFs:
  - echo *.{ text, jpeg, pdf }

Testing and conditionals.
- logical operators that appear in/or appear nearly Bash as well
- Equality is tested with =
- There are also if, elif, and else blocks
  - Conditional blocks must end w/ fi.
- Tests are done with [[...]];
  - can also use [...] but it's older and lacks features such as pattern matching
  - use [...] if the script may be run in an environment w/o Bash.

Pipes.
- connects stdout of one process to stdin of another process.
- Pipes are denoted w/ the pipe operator: "|".
- pipes can be chained together creating a pipeline.
- programs can be written such that they can be chained together to get interesting results.
- the whole philosophy behind the software tools of the UNIX

awk

- awk is a program for simple parsing of text.
- Designed as a text processor that scans input file, splitting input lines into fields automatically
- output from awk can be processed even further w/ other tools

32, 4

t = 32

b = 32 % 4 = 0

a = 32 ;

$$\frac{n-1}{2^s} = r \qquad s = 0$$

$$\frac{104716}{2^s} = n-1$$

~~while(isnum(r)) {~~

do {

$$\frac{104716}{2^s} = r$$

} while(isnum(r));

s = [redacted]