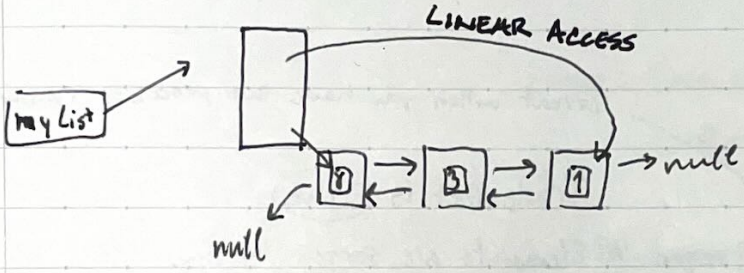
Date

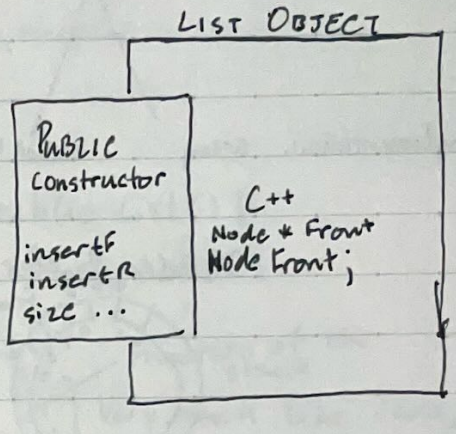
STATEMENT OOP LL

- two classes (three if data class)
- List object has exactly 1
- Node object has zero or more nodes

Making List: C++: List myList; Java: List myList = new List();



VISUAL



|:| nodes/data

Basic List Traversal

C++ pointer & Java ref are almost the same.
* Don't use look ahead

C++

```
void List::print(){
    Node *temp = front;
    while (temp) {
        cout << temp -> data << endl;
        temp = temp -> next;
    }
```

uses a pointer

Java

```
... print(){
Node temp = Front;
while (temp != null) {
    system.out.println(temp.data);
    temp = temp.next;
}
```

Using a reference

* Draw a picture/diagram if you don't understand/want a visual of it *

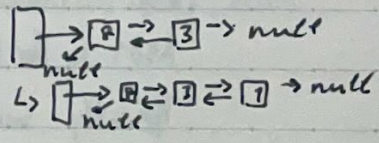
PARALLEL ARRAYS

INDEX	0	1	2				
data	3	8	1				
next	-1	0	-1				
prev	1	-1	0				

front 1
back 0

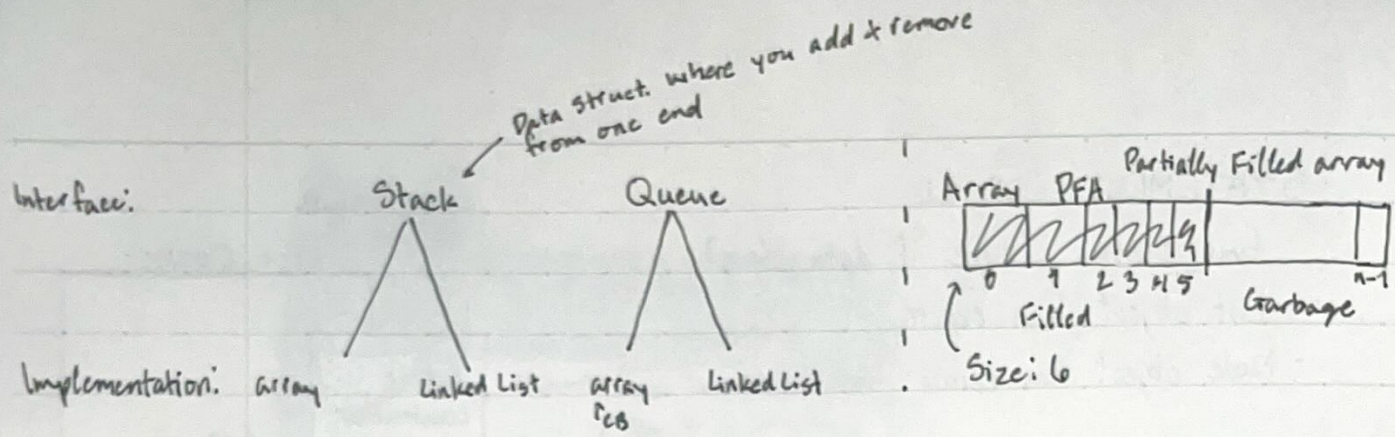
front
back

→ LOOKS LIKE:

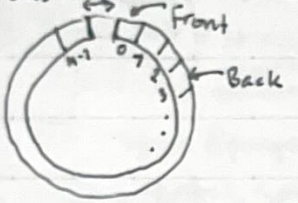


The list now has 3 nodes (elements)

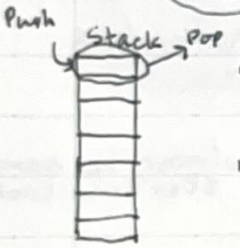
Data

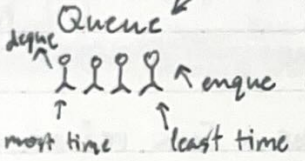


Circular Buffer (Imagine as a Circle, not a line)



Great when you have two processes running





Trade Off is (Time)

All Elements are sorted by time

As1 is using stack ABCD → DCBA

Break / GoTo Example

```
for (int i=0; i<100; i++) {
    if (a=11) {
        break; ← or a=101;
    }
}
```

BAD. DON'T DO THIS

Rethink & maybe use a while loop

Use cin & cout for your programs

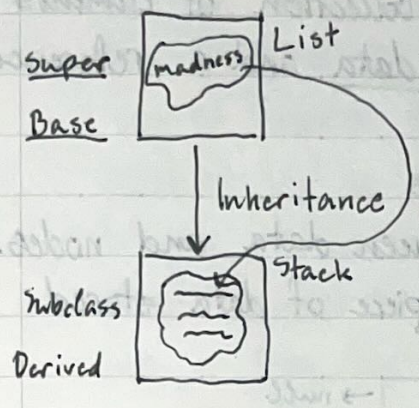
As01 is for the input to be in reverse order

Typically will be given 1 type of data, then we have to make the rest

```
class List {
    public: InsertFront(~~);
            InsertRear(~~);
```

Composition: "has a" possession I have a stapler

Inheritance: "is a" you are I am a stapler



Inheritance:

C++: class Stack: public List() {

Java: class Stack extends List() {

Inheritance:

public Push(int x) {

insertFront(x);

}

Function of the stack

Very much like Java, but we can understand it

We can treat a list & a stack polymorphically the same

Composition:

class Stack {

: ~Stack() {

: } myList = new List();

private List myList;

public Push(int x) {

myList.insertFront(x);

Function of the List

* Looking for composition, not inheritance *

a form of capsulation

* Don't forget about redirection, make it a habit: Ex java Hello < foo > test me

* # include .h files NOT .cpp files it WILL BREAK

make → touch → make clean

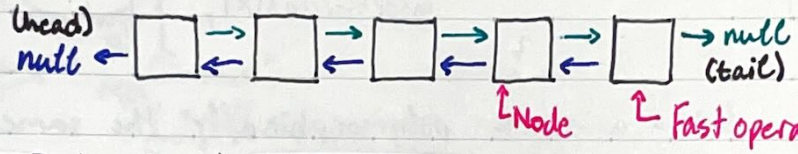
↳ Nukes all binary & leaves all text files

* ~ ← emacs backup files to the files that end in *

(Doubly) Linked List

What is a doubly linked list

- A doubly linked list is a linear collection of elements called nodes. Each node contains some data and a reference to the next and prev node.
- Provides linear access to elements
- One-to-one relationship between data and nodes. There is always one node for each piece of data stored



Linked List

- Data struct that is non-contiguous
- Contiguous in memory (unlike an array)
- Holds ref/ptr to other nodes and links them

★ List knows when it adds or changes, nodes do not. Nodes are merely to be manipulated ★

★ Good for manipulating groups of nodes ★

Linked Nodes

- Nodes can be a class/struct inside list class
- For C++ you can/might want to use friendship
- In regular oop linked list, there should be 1:1 (nodes : data) (one node per piece of data)
- Every list has 1 list object
  - ↳ You can still interact if empty

Data Structure Operations

- Time Complexity

Average			
Access	Search	Insert	Delete
O(n)	O(n)	O(1)	O(1)

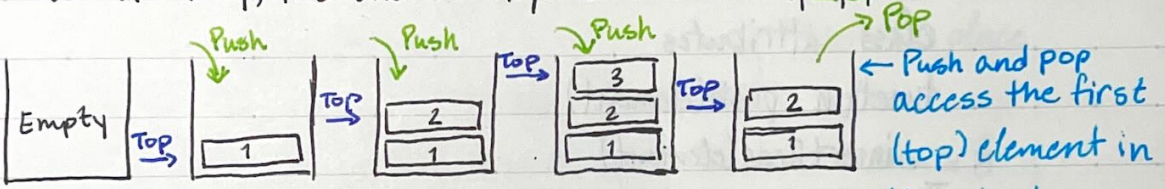
Worst			
Access	Search	Insert	Delete
O(n)	O(n)	O(1)	O(1)

- Space complexity (worst) => O(n)

STACK

What is a stack

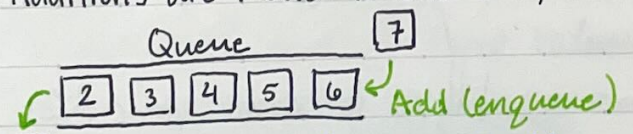
- A stack is a linear data structure who follows LIFO (Last In First Out). It has two main operations - pop & push
- pop => Removes the most recently added element (at the top).
- push => Add an element to the top
- Imagine a stack as a stack of lunch trays. A tray gets placed (pushed) on the top of the stack. If someone wants to use a tray, the one on top is removed (popped)



- Uses => Reversing Things

QUEUE

- A queue is a linear data structure that is open on both ends and the operations are performed FIFO (first In First Out) order. Additions are made to one end, and deletions the other



New elements are always added to the rear and elements are deleted from the head. Elements in the middle of the queue cannot be reordered.

- Uses => Buffer

INTERFACE & IMPLEMENTATION

Interface => How you see it — Implementation => How it works

- Stack → Linked List or Array
- Queue → Linked List or Array
- Priority Queue → Heap (Array)
- Set → Array

Composition & Inheritance

- Base a new class off one which already exists

Composition => "has a" relationship

- No functions from the original class exist in the new class.
  The old class is "walled-off"

function pushLelement)

myList.insertRear(element)

Inheritance => "is a" relationship

- Members in the base class exist in the derived class.
  The derived class can use or overwrite base
  class attributes
  function push(element)
    insertRear(element)
- The advantage is polymorphism

☆ Pengo Turn In

home directory
'cd'
pwd
  /home/username
mkdir 21-1
cd 21-1
pwd
  /home/username/21-1

Grading File
Receipt File ↗ CS21-1_COLLECTED
Receipt File ↘ CS21-1_FILE_NOT_FOUND

Where to create your files
vim, emacs] Edit your list