Week05

✨ No-Laptop Monday! ✨

TODO: record lecture

- **review:** data structures and interface
- stacks
- queues

WARMUP (from More Practice Exam Questions)

```
// Returns the indices (NOT values) of all even numbers
// NOTE: Assume all numbers in the array are non-negative.
// { 6, 7, 5, 8 } -> { 0, 3 } because the 0th element (6)
//                           and 3rd element (8) are even
// { 7, 1, 5 } -> {} because no elements are even
ArrayList<Integer> getIndicesOfEvenNumbers(int[] array) {
    ...
}
```

ANNOUNCEMENTS

welcome back i hope you had a nice mountain day :D
jk this slide was from last year 💀

WARMUP

what is this robot for? →

TODAY

stacks and queues

review: data structures

what is a data structure?
why is a data structure?

data structures

- In computer science, a **data structure** is a data organization, management, and storage format that is usually chosen for efficient access to data.[1][2][3] More precisely, a data structure is a collection of data values, the relationships among them, and the functions or operations that can be applied to the data,[4] i.e., it is an algebraic structure about data. --Wikipedia

data structures

- In computer science, a **data structure** is the organization and implementation of values and information. In simple words, it is the way of organizing information in a computer so that it can be more easily understood and worked with. --Simple Wikipedia

data structures

- A **data structure** is...a system for organizing and using information... It...make[s] information easier to understand and work with.. --Simple Wikipedia, further simplified by ChatGPT

data structures

- **data** means numbers (and letters)
- a **data structure** organizes your data
  - for a particular task, a *good* data structure is...
    - easy to work with (programmer time)
    - runs fast (runtime / user's time)

were array lists a good choice for implementing a Flip Book?

(how) could you have done it with just arrays?

well, we didn't know how many...

- pages in the flipbook?
- strokes in each page?
- points in each stroke?

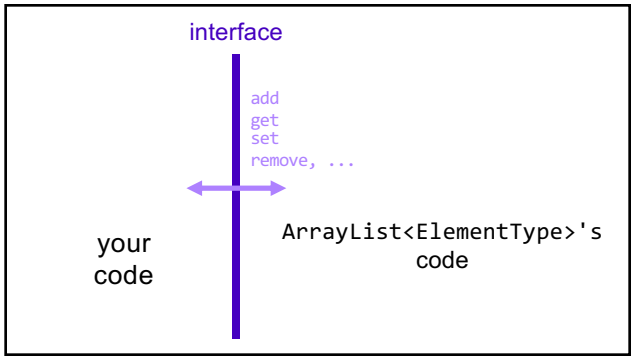
"Alternative" 1: just use arrays, but resize them yourself when needed (essentially, implement the functionality of an array list without actually having a class)

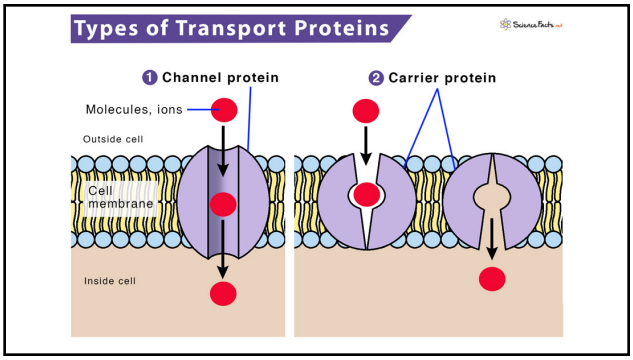
Alternative 2: use a BIG multi-dimensional array

```
// ~30,000,000 elements
Point[][][] animation = new Point[64][512][1024];
// NOTE: we also need all these counter variables
// can't just call .size() like before!
int numFrames = 0;
int[] numStrokes = new int[64];
int[][] numPoints = new int[64][512];
```

these alternatives are probably kind of trash;
array lists considered helpful 😳 👍

interface





interface

- a data structure's **interface** (**API**) or **abstract data type** is a set of functions a data structure must have
  - a **list** has...
    - **get**
    - **set**
    - **add**
    - **remove**
    - etc.
- a **data structure** is a specific **implementation** (code that does the thing) of that interface
  - an **array list** implements the list interface using an array
  - a **linked list** implements a list using nodes that refer to nodes

you can get (very) formal about this

The abstract list type L with elements of some type E (a monomorphic list) is defined by the following functions:

nil: $() \to L$
cons: $E \times L \to L$
first: $L \to E$
rest: $L \to L$

with the axioms

first (cons (e, l)) = e
rest (cons (e, l)) = l

for any element e and any list l. It is implicit that

cons $(e, l) \neq l$
cons $(e, l) \neq e$
cons (e_1, l_1) = cons (e_2, l_2) if $e_1 = e_2$ and $l_1 = l_2$

Note that first (nil ()) and rest (nil ()) are not defined.
These axioms are equivalent to those of the abstract stack data type.

i typically won't.

just know there is a difference between
interface ("a list") and
implementation (ArrayList<Element>)

stack

analogy

a stack is like a helicopter pilot
with a socially unacceptable hobby

`stack.push(🐨)`

`stack.push(🐻)`

stack.push(🐷)

stack.pop()

stack.pop()

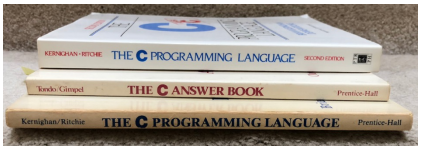
stack.push(🐮)

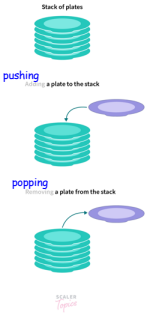
stack.pop()

stack.pop()


I VOTED FOR TEAM ALPHA WOLF
SQUADRON.

stack


KERNIGHAN · RITCHIE THE C PROGRAMMING LANGUAGE SECOND EDITION
Tondo/ Gimpel THE C ANSWER BOOK Prentice-Hall
Kernighan/Ritchie THE C PROGRAMMING LANGUAGE Prentice-Hall


Stack of plates
pushing
a plate to the stack
popping
a plate from the stack

Our Standard Stack Analogy Is Wrong

In search of a better stack metaphor

October 7, 2021

The most common way we introduce the stack data structure is by saying it's like a spring-loaded plate dispenser, the kind found in a cafeteria:

Indeed when I looked up stacks in an old computer architecture textbook[1] of mine I found:

> stacks are frequently described by the way plates are stored and used in a cafeteria. New plates are added to the top of the stack, or pushed, and plates already on the stack move down to make room for them.

Often we augment the comparison with a diagram showing the members of the stack being pushed down or popped up[2]:

Using a Stack

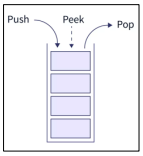
stack interface

```
// Push (add) a new element to the top of the stack.
void push(ElementType element);

// Pop (remove) the top element of the stack.
// and returns it.
ElementType pop();

// Peek (look) at the top element of the stack
// (without removing it) and return it.
ElementType peek();

// Returns the number of elements currently in the stack.
int size();
```



queue

analogy

a queue is like a line of very polite people waiting patiently

queue.remove()

queue.remove()

queue.add( )

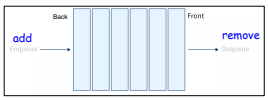
queue.add( )

queue.remove()

queue

queue interface

- `// Add (enqueue) a new element to the back of the queue.`
  `void add(ElementType element);`
- `// Remove (dequeue) the front`
  `// element and return it.`
  `ElementType remove();`



- `// Peek (look) at the front element (without removing it)`
  `// and return it.`
  `ElementType peek();`
- `// Returns the number of elements currently in the queue.`
  `int size();`

why stacks and queues in the same lecture?

stack interface

- `// Push (add)  a new element to the top  of the stack.`
  `void push(ElementType element);`
- `// Remove (pop)  the top  element of the stack.`
  `// and returns it.`
  `ElementType pop();`
- `// Peek (look) at the top  element of the stack`
  `// (without removing it) and return it.`
  `ElementType peek();`
- `// Returns the number of elements currently in the stack.`
  `int size();`

queue interface

- `// Add (enqueue) a new element to the back of the queue.`
  `void add(ElementType element);`
- `// Remove (dequeue) the front element of the queue`
  `// and return it.`
  `ElementType remove();`
- `// Peek (look) at the front element of the queue`
  `// (without removing it) and return it.`
  `ElementType peek();`
- `// Returns the number of elements currently in the queue.`
  `int size();`

what does this print?

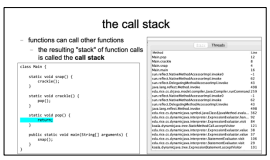
```
Stack<Integer> stack = new Stack<>();
stack.push(2);
stack.push(3);
stack.push(4);
stack.push(5);
PRINT(stack.pop());
PRINT(stack.peek());
PRINT(stack.size());
```

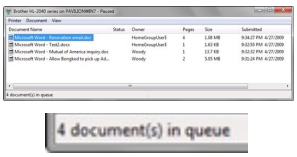
```
Stack<Integer> stack = new Stack<>();
stack.push(2);
stack.push(3);
stack.push(4);
stack.push(5);
PRINT(stack.pop());  // 5
PRINT(stack.peek()); // 4
PRINT(stack.size()); // 3
```

```
ArrayDeque<Integer> queue = new ArrayDeque<>();
queue.add(7);
queue.add(8);
queue.add(9);
PRINT(queue.remove());
PRINT(queue.peek());
PRINT(queue.size());
```

```
ArrayDeque<Integer> queue = new ArrayDeque<>();
queue.add(7);
queue.add(8);
queue.add(9);
PRINT(queue.remove()); // 7
PRINT(queue.peek());   // 8
PRINT(queue.size());   // 2
```

what are some example uses of stacks and queues in computer science?





ANNOUNCEMENTS

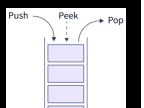
today is Laptop Wednesday!

replace interpret(...tokens) with version from **the Writeup**

WARMUP

a couple (slightly silly) interview questions

- implement a stack using a (basic) queue
  - just implement push and pop (and peek?)
  - (don't worry about efficiency)
  - pseudocode is fine
  - what is the runtime (big O)?
- implement a queue using 2 stacks



TODAY stacks and queue tutorial

Stack and Queue tutorial

ANNOUNCEMENTS

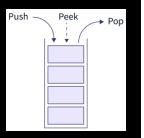
today is Friday! (fun Colloquium today at 2:35 in TCL 123)

apply to **TA** next semester! also next next (next?) semester!

WARMUP

implement a queue using two stacks

- just implement add and remove (and peek?)
- (don't worry about efficiency)
- pseudocode is fine
- what is the runtime (big O)?



TODAY odds and ends

where did the Starter Code come from?

(forgive me, I know this should have been on Wednesday)

the big decision in the Starter Code is that we're going to chop the program into a list of "tokens"

and each token could be a boolean, double, String, or list

in Python, a variable's type is **dynamic**

```
token = True
print(type(token)) # <class 'bool'>

token = 5.0
print(type(token)) # <class 'float'>

token = "exch"
print(type(token)) # <class 'str'>

token = ["exch", 0.0, "add"]
print(type(token)) # <class 'list'>
```

in Java, a variable's type is **static**

0	false	0.0	null	null

```
class Token {
    int type;
    private boolean _valueIfTypeBoolean;
    private double _valueIfTypeDouble;
    private String _valueIfTypeString;
    private ArrayList<Token> _valueIfTypeList;
    static final int TYPE_BOOLEAN = 0;
    static final int TYPE_DOUBLE = 1;
    static final int TYPE_STRING = 2;
    static final int TYPE_LIST = 3;
}
```

examples

- `new Token(true)`

0	true	0.0	null	null

- `new Token(5.0)`

1	false	5.0	null	null

- `new Token("exch")`

2	false	0.0	0x02343 af34...	null

→ `"exch"`

in Java, a variable's type is **static**

```
// Create a String-type Token.
Token token = new Token();
token.type = Token.TYPE_STRING;
token._valueIfTypeString = "exch";
```

```
// Get the value of a String-type token.
assert token.type = Token.TYPE_STRING : "..."; // check type!
String string = token._valueIfTypeString;
```

this usage code makes me sad ☹️
let's write some ✨**functions**✨!

in Java, a variable's type is **static**

```
// Create a String-type Token.
Token token = new Token("exch");
```

```
// Get the value of a String-type token.

String string = token.getString();
```

all better 😀

tl;dr

use these functions 😀 👍

Token

- Simple class that can represent any type of token.
  - Convenient getters with assert statements.
  - Static method to "tokenize" a PostScript program (turn it into a list of tokens).

```
class Token {
    int type; // type of Token; can be Token.TYPE_BOOLEAN, Token.TYPE_DOUBLE, Token.TYPE_STRING,

    Token(boolean          value); // creates a new boolean type token with given value
    Token(double           value); // creates a new double  type token with given value
    Token(String           value); // creates a new string  type token with given value
    Token(ArrayList<Token> value); // creates a new list    type token with given value

    boolean          getBoolean(); // gets token's value as a  boolean          (crashes if not
    double           getDouble();  // gets token's value as a  double           (crashes if not
    String           getString();  // gets token's value as a  String           (crashes if not
    ArrayList<Token> getList();    // gets token's value as an ArrayList<Token> (crashes if not

    static ArrayList<Token> tokenize(String postScriptProgram); // turns PostScript program into
}
```

and these functions!! 😳 👍

Interpreter

- Class that wraps around the stack and the map.
  - Method to interpret a PostScript program that helpfully prints out the stack and map for you.
  - Method to interpret a single token (the only thing you need to implement).
  - Convenient methods for working with the stack and map with assert statements.

```
class Interpreter {
    void interpret(ArrayList<Token> tokens); // interprets a list of tokens
    void interpret(Token token); // interprets a single token

    Token           stackPopToken();   // pops a Token off of the stack
    boolean         stackPopBoolean(); // pops a Token off of the stack and gets its value as a
    double          stackPopDouble();  // pops a Token off of the stack and gets its value as a
    String          stackPopString();  // pops a Token off of the stack and gets its value as a
    ArrayList<Token> stackPopList();   // pops a Token off of the stack and gets its value as a
    void stackPush(Token           token); // pushes a token onto the stack
    void stackPush(boolean         value); // creates a new boolean type token with given value
    void stackPush(double          value); // creates a new double  type token with given value
    void stackPush(String          value); // creates a new string  type token with given value
    void stackPush(ArrayList<Token> value); // creates a new list    type token with given value

    void    mapPut(String key, Token value); // puts a key-value pair into the map
    Token   mapGet(String key); // for the given key, looks up the corresponding value in the ma
    boolean mapContainsKey(String key); // returns whether the map contains a key-value pair wit
}
```

```
// Pop a double off the stack
assert _stack.size() > 0;
Token token = _stack.pop();
double num = token.getDouble();
```

```
// Push a double onto the stack.
_stack.push(new Token(5.0));
```

😟

```
// Pop a double off the stack


double num = stackPopDouble();
```

```
// Push a double onto the stack.
stackPush(5.0);
```

😳

Tungsten break 😀

did you know the density of Tungsten is 19.25 g/cm^3?
Gold is 19.32 g/cm^3

did you know Tungsten is really hard?
Tungsten alloys can be as hard as Sapphire! that's really hard!

Tungsten is expensive (that cube is $60) but not like...that expensive

what could YOU do with Tungsten?