CS 280
Spring 2018

John Bowers, Professor
Mike Lam, Professor

Lists

Linear data structures

- Abstract data types (set of values with defined operations)
  - List (random index-based access)
  - Stack (add/remove from one end)
  - Queue (add from one end, remove from the other)

- Data structures (actual layout of data in memory)
  - Array (fixed-size)
  - Dynamic Array (variable-sized)
  - Linked list

List ADT

- `add(item)`: Adds item to end of List.
- `add(idx, item)`: Adds item at index idx.
- `get(idx)`: returns item at index idx
- `set(idx, item)`: Replace object at idx with item.
- `remove(idx)`: removes item at idx.

Stack ADT

- `push(item)`: Pushes item to top of stack.
- `pop()`: Returns and removes top of stack.
- `peek()`: Returns but does not remove top of stack.
- `empty()`: Tests if the stack is empty.

Queue ADT

- `offer(item)`: Adds item to back of queue.
- `poll()`: Removes and returns item at front of queue.
- `peek()`: Returns item at front of queue (doesn't remove)
- `isEmpty()`: Tests if the queue is empty.

Deque ADT

- `offerFirst(item)`: Adds item to back of queue.
- `pollFirst()`: Removes and returns item at front of queue.
- `peekFirst()`: Returns item at front of queue (but does not remove)
- `offerLast(item)`: Adds item to back of queue.
- `pollLast()`: Removes and returns item at front of queue.
- `peekLast()`: Returns item at front of queue (but does not remove)

Java Data Structures

- List
  - `ArrayList<E>`
  - `LinkedList<E>`
- Stack
  - `Stack<E>`
  - `ArrayDeque<E>`
- Queue
  - `ArrayList<E>`
  - `LinkedList<E>`
  - `ArrayDeque<E>`
- Deque
  - `LinkedList<E>`
  - `ArrayDeque<E>`

Asymptotics

	add(E)	addFirst	add(i,E)	get/set
List				
– ArrayList<E>	O(1)		O(n)	O(1)
– LinkedList<E>	O(1)	O(1)	O(n)	O(n)

	push	pop	peek
Stack			
– Stack<E>	O(1)	O(1)	O(1)
– ArrayDeque<E>*	O(1)	O(1)	O(1)

	offer	poll	peek
Queue			
– LinkedList<E>	O(1)	O(1)	O(1)
– ArrayDeque<E>*	O(1)	O(1)	O(1)

	offer(First|Last)	poll(First|Last)	peek(First|Last)
Deque			
– LinkedList<E>	O(1)	O(1)	O(1)
– ArrayDeque<E>*	O(1)	O(1)	O(1)

* ArrayDeque<E> is “likely to be faster than Stack<E> when used as a Stack and faster than LinkedList<E> when used as a queue.”

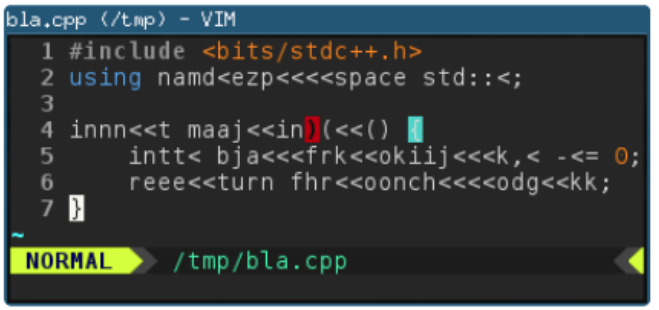
Takeaway

- If you need a list, use `ArrayList<E>`
- If you need a stack/queue/dequeue use `ArrayDeque<E>`

- `LinkedList<E>` is generally not worth it
- Under no circumstances should you use `Vector<E>`

Solved problem

Backspace



Bjarki having trouble

Shortly before the programming contest started, Bjarki decided to update his computer. He didn't notice anything strange until he started coding in his favorite editor, Bim (Bjarki IMproved). Usually when he's writing in an editor and presses the *backspace* key a single character is erased to the left. But after the update pressing that key outputs the character <. He's tested all the editors on his machine, Bmacs, Neobim, bjedit, NoteBjad++ and Subjark Text, but they all seem to have the same problem. He doesn't have time to search the web for a solution, and instead decides to temporarily circumvent the issue with a simple program.

Help Bjarki write a program that takes as input the string that was written in the text editor, and outputs the string as Bjarki intended to write it. You can assume that Bjarki never intended to write the character <, and that Bjarki never pressed the backspace key in an empty line.

Input

One line containing the string that was written in the text editor. The length of the string is at most 10^6, and it will only contain lowercase letters from the English alphabet as well as the character <.

Output

One line containing the string as Bjarki intended to write it.

Solved problem

Input

One line containing the string that was written in the text editor. The length of the string is at most 10^6, and it will only contain lowercase letters from the English alphabet as well as the character <.

Output

One line containing the string as Bjarki intended to write it.

Sample Input 1

```
a<bc<
```

Sample Output 1

```
b
```

Sample Input 2

```
foss<<rritun
```

Sample Output 2

```
forritun
```

Sample Input 3

```
a<a<a<aa<<
```

Sample Output 3

Solved problem

```
Scanner in = new Scanner(System.in);
Stack<Character> stack = new Stack();

String line = in.nextLine();
for (char c: line.toCharArray()) {
    if (c == '<') {
        stack.pop();
    } else {
        stack.push(c);
    }
}

StringBuilder sb = new StringBuilder();
for (Character c: stack) {
    sb.append(c);
}

System.out.println(sb.toString());
```

Warning: buffered output

- Output is not free
  - For some problems, you will exceed the time limit if you have a lot of output
- Solution: buffer the output!
  - Create a PrintWriter
    - `PrintWriter out = new PrintWriter(new BufferedOutputStream(System.out));`
  - Has `print/println/printf` just like `System.out`
  - This saves output in a temporary buffer
  - Finalize output using `out.flush()`