https://introcs.cs.princeton.edu/java/43stack/

**Stack**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Compilation: javac Stack.java

\* Execution: java Stack < input.txt

\* Data files: https://introcs.cs.princeton.edu/java/43stack/tobe.txt

\*

\* A generic stack, implemented using a linked list. Each stack

\* element is of type Item.

\*

\* % more tobe.txt

\* to be or not to - be - - that - - - is

\*

\* % java Stack < tobe.txt

\* to be not that or be (2 left on stack)

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

import java.util.Iterator;

import java.util.NoSuchElementException;

/\*\*

\* The {@code Stack} class represents a last-in-first-out (LIFO) stack of generic items.

\* It supports the usual <em>push</em> and <em>pop</em> operations, along with methods

\* for peeking at the top item, testing if the stack is empty, getting the number of

\* items in the stack, and iterating over the items in LIFO order.

\* <p>

\* This implementation uses a singly-linked list with a nested class for

\* linked-list nodes.

\* The <em>push</em>, <em>pop</em>, <em>peek</em>, <em>size</em>, and <em>is-empty</em>

\* operations all take constant time in the worst case.

\* <p>

\* For additional documentation,

\* see <a href="https://introcs.cs.princeton.edu/43stack">Section 4.3</a> of

\* <i>Computer Science: An Interdisciplinary Approach</i>

\* by Robert Sedgewick and Kevin Wayne.

\*

\* @author Robert Sedgewick

\* @author Kevin Wayne

\*

\* @param <Item> the generic type of an item in this stack

\*/

public class Stack<Item> implements Iterable<Item> {

private int n; // size of the stack

private Node first; // top of stack

// helper linked list class

private class Node {

private Item item;

private Node next;

}

/\*\*

\* Initializes an empty stack.

\*/

public Stack() {

first = null;

n = 0;

}

/\*\*

\* Returns true if this stack is empty.

\*

\* @return true if this stack is empty; false otherwise

\*/

public boolean isEmpty() {

return first == null;

}

/\*\*

\* Returns the number of items in this stack.

\*

\* @return the number of items in this stack

\*/

public int size() {

return n;

}

/\*\*

\* Adds the item to this stack.

\*

\* @param item the item to add

\*/

public void push(Item item) {

Node oldfirst = first;

first = new Node();

first.item = item;

first.next = oldfirst;

n++;

}

/\*\*

\* Removes and returns the item most recently added to this stack.

\*

\* @return the item most recently added

\* @throws NoSuchElementException if this stack is empty

\*/

public Item pop() {

if (isEmpty()) throw new NoSuchElementException("Stack underflow");

Item item = first.item; // save item to return

first = first.next; // delete first node

n--;

return item; // return the saved item

}

/\*\*

\* Returns (but does not remove) the item most recently added to this stack.

\*

\* @return the item most recently added to this stack

\* @throws NoSuchElementException if this stack is empty

\*/

public Item peek() {

if (isEmpty()) throw new NoSuchElementException("Stack underflow");

return first.item;

}

/\*\*

\* Returns a string representation of this stack.

\*

\* @return the sequence of items in this stack in LIFO order, separated by spaces

\*/

public String toString() {

StringBuilder s = new StringBuilder();

for (Item item : this) {

s.append(item);

s.append(' ');

}

return s.toString();

}

/\*\*

\* Returns an iterator to this stack that iterates through the items in LIFO order.

\*

\* @return an iterator to this stack that iterates through the items in LIFO order

\*/

public Iterator<Item> iterator() { return new ListIterator(); }

// an iterator, doesn't implement remove() since it's optional

private class ListIterator implements Iterator<Item> {

private Node current = first;

public boolean hasNext() { return current != null; }

public void remove() { throw new UnsupportedOperationException(); }

public Item next() {

if (!hasNext()) throw new NoSuchElementException();

Item item = current.item;

current = current.next;

return item;

}

}

/\*\*

\* Unit tests the {@code Stack} data type.

\*

\* @param args the command-line arguments

\*/

public static void main(String[] args) {

Stack<String> stack = new Stack<String>();

while (!StdIn.isEmpty()) {

String item = StdIn.readString();

if (!item.equals("-")) stack.push(item);

else if (!stack.isEmpty()) StdOut.print(stack.pop() + " ");

}

StdOut.println("(" + stack.size() + " left on stack)");

}

}

**Array Stack Of String**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Compilation: javac ArrayStackOfStrings.java

\* Execution: java ArrayStackOfStrings

\* Data files: https://introcs.cs.princeton.edu/java/43stack/tobe.txt

\*

\* Stack of strings implementation with a fixed-size array.

\*

\* % more tobe.txt

\* to be or not to - be - - that - - - is

\*

\* % java ArrayStackOfStrings 5 < tobe.txt

\* to be not that or be

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

import java.util.Iterator;

import java.util.NoSuchElementException;

public class ArrayStackOfStrings implements Iterable<String> {

private String[] items; // holds the items

private int n; // number of items in stack

public ArrayStackOfStrings(int capacity) {

items = new String[capacity];

}

public boolean isEmpty() {

return n == 0;

}

public boolean isFull() {

return n == items.length;

}

public void push(String item) {

items[n++] = item;

}

public String pop() {

return items[--n];

}

public Iterator<String> iterator() {

return new ReverseArrayIterator();

}

// an iterator, doesn't implement remove() since it's optional

private class ReverseArrayIterator implements Iterator<String> {

private int i = n-1;

public boolean hasNext() { return i >= 0; }

public void remove() { throw new UnsupportedOperationException(); }

public String next() {

if (!hasNext()) throw new NoSuchElementException();

return items[i--];

}

}

public static void main(String[] args) {

int capacity = Integer.parseInt(args[0]);

ArrayStackOfStrings stack = new ArrayStackOfStrings(capacity);

while (!StdIn.isEmpty()) {

String item = StdIn.readString();

if (!item.equals("-")) {

stack.push(item);

}

else {

StdOut.print(stack.pop() + " ");

}

}

StdOut.println();

}

}

**Resizing Array Stack Of Strings**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Compilation: javac ResizingArrayStackOfStrings.java

\* Execution: java ResizingArrayStackOfStrings < input.txt

\* Dependencies: StdIn.java StdOut.java

\* Data files: https://introcs.cs.princeton.edu/java/43stack/tobe.txt

\*

\* Stack implementation with a resizing array.

\*

\* % more tobe.txt

\* to be or not to - be - - that - - - is

\*

\* % java ResizingArrayStack < tobe.txt

\* to be not that or be (2 left on stack)

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

import java.util.Iterator;

import java.util.NoSuchElementException;

public class ResizingArrayStackOfStrings implements Iterable<String> {

private String[] items; // array of strings

private int n = 0; // number of elements on stack

// create an empty stack

public ResizingArrayStackOfStrings() {

items = new String[2];

}

public boolean isEmpty() {

return n == 0;

}

public int size() {

return n;

}

// resize the underlying array holding the elements

private void resize(int capacity) {

assert capacity >= n;

String[] temp = new String[capacity];

for (int i = 0; i < n; i++)

temp[i] = items[i];

items = temp;

}

// push a new item onto the stack

public void push(String item) {

if (n == items.length) resize(2\*items.length); // double array length if necessary

items[n++] = item; // add item

}

// delete and return the item most recently added

public String pop() {

if (isEmpty()) throw new NoSuchElementException("Stack underflow");

String item = items[n-1];

items[n-1] = null; // to avoid loitering

n--;

// shrink size of array if necessary

if (n > 0 && n == items.length/4) resize(items.length/2);

return item;

}

public Iterator<String> iterator() {

return new ReverseArrayIterator();

}

// an iterator, doesn't implement remove() since it's optional

private class ReverseArrayIterator implements Iterator<String> {

private int i = n-1;

public boolean hasNext() { return i >= 0; }

public void remove() { throw new UnsupportedOperationException(); }

public String next() {

if (!hasNext()) throw new NoSuchElementException();

return items[i--];

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Test routine.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

public static void main(String[] args) {

ResizingArrayStackOfStrings stack = new ResizingArrayStackOfStrings();

while (!StdIn.isEmpty()) {

String item = StdIn.readString();

if (!item.equals("-")) stack.push(item);

else if (!stack.isEmpty()) StdOut.print(stack.pop() + " ");

}

StdOut.println("(" + stack.size() + " left on stack)");

}

}