Data Structures and Algorithms

—

Lab 2
Elementary Data Structures. Shunting-yard algorithm

Agenda

- Lecture recap
- Implementing `Stack` using `ArrayList` **(live coding)**
- Java's `ArrayDeque` **(code review)**
- Implementing `Queue` using `LinkedList` **(live coding)**
- Algorithm discussion: parse and evaluate arithmetic expression
- CodeForces!

Elementary Data Structures

- Which **Elementary Data Structures** do you know?

Elementary Data Structures

- List
- Stack
- Queue
- Deque (Stack + Queue)

How do we store data?

How do we store data?

- Array-like structure
- Linked structure

Question: What are the pros and cons of each?

How do we store data?

	Array-like structure	Linked structure
Random access (by index)		
Add/grow		

How do we store data?

	Array-like structure	Linked structure
Random access (by index)	O(1)	O(n)
Add/grow	O(n)	O(1)

ADT vs DS?

ADT vs DS?

- **Abstract Data Type**
  List, Stack, Queue, PriorityQueue

- **Data Structure**
  ArrayList, LinkedList, ...

Live coding: `ArrayList (dynamic array)` => `Stack`

- Stack live coding using `ArrayList`

```
interface Stack<T> {
    void push(T value);
    T pop();
    T peek();
    int size();
    boolean isEmpty();
}
```

ArrayDeque.java

The full source code for `java.util.ArrayDeque`:

http://fuseyism.com/classpath/doc/java/util/ArrayDeque-source.html

```
Object[] elements;
int head;
int tail;
```

ArrayDeque.java

```java
/**
 * Inserts the specified element at the front of this deque.
 *
 * @param e the element to add
 * @throws NullPointerException if the specified element is null
 */
public void addFirst( @NotNull() E e) {
    if (e == null)
        throw new NullPointerException();
    elements[head = (head - 1) & (elements.length - 1)] = e;
    if (head == tail)
        doubleCapacity();
}
```

ArrayDeque.java

```java
/**
 * Inserts the specified element at the end of this deque.
 *
 * <p>This method is equivalent to {@link #add}.
 *
 * @param e the element to add
 * @throws NullPointerException if the specified element is null
 */
public void addLast( @NotNull() E e) {
    if (e == null)
        throw new NullPointerException();
    elements[tail] = e;
    if ( (tail = (tail + 1) & (elements.length - 1)) == head)
        doubleCapacity();
}
```

ArrayDeque.java

```
public E pollFirst() {
    int h = head;
    /unchecked/
    E result = (E) elements[h];
    // Element is null if deque empty
    if (result == null)
        return null;
    elements[h] = null;     // Must null out slot
    head = (h + 1) & (elements.length - 1);
    return result;
}
```

ArrayDeque.java

```
public E pollLast() {
    int t = (tail - 1) & (elements.length - 1);
    /unchecked/
    E result = (E) elements[t];
    if (result == null)
        return null;
    elements[t] = null;
    tail = t;
    return result;
}
```

Live coding: `LinkedList` => `Queue`

- Queue live coding using `LinkedList`
- Which type of linking will we choose? (SLL or DLL)

```
interface Queue<T> {
    void offer(T value);
    T pool();
    T peek();
    int size();
    boolean isEmpty();
}
```

Shunting-yard algorithm

Shunting-yard algorithm is an algorithm that parses an expression in infix notation (e.g. arithmetic expression) into an internal representation. Here we will consider converting to Reverse Polish Notation (RPN), also known as postfix notation.

```
1 + (2 − 3) × 4
```

Exercise: Shunting-yard algorithm

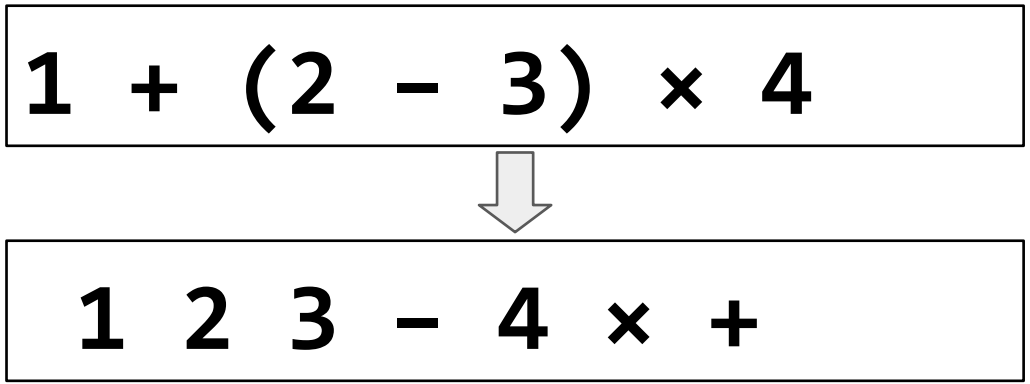
Shunting-yard algorithm is an algorithm that parses an expression in infix notation (e.g. arithmetic expression) into an internal representation. Here we will consider converting to Reverse Polish Notation (RPN), also known as postfix notation.

1 + (2 − 3) × 4

1 2 3 − 4 × +

Exercise: Shunting-yard algorithm

Shunting-yard algorithm is an algorithm that parses an expression in infix notation (e.g. arithmetic expression) into an internal representation. Here we will consider converting to Reverse Polish Notation (RPN), also known as postfix notation.



Discuss the algorithm.
What are the corner cases?
What data structures would you use? Why?

Input buffer	Stack	Output Queue
1 + (2 − 3) × 4		
+ (2 − 3) × 4		1
(2 − 3) × 4	+	1
2 − 3) × 4	+ (	1
− 3) × 4	+ (	1 2
3) × 4	+ (−	1 2
) × 4	+ (−	1 2 3
× 4	+	1 2 3 −
4	+ ×	1 2 3 −
	+ ×	1 2 3 − 4
		1 2 3 − 4 × +

RPN evaluation

Given an expression in Reverse Polish Notation it is fairly straightforward to evaluate it.

```
1 2 3 - 4 × +
```

RPN evaluation

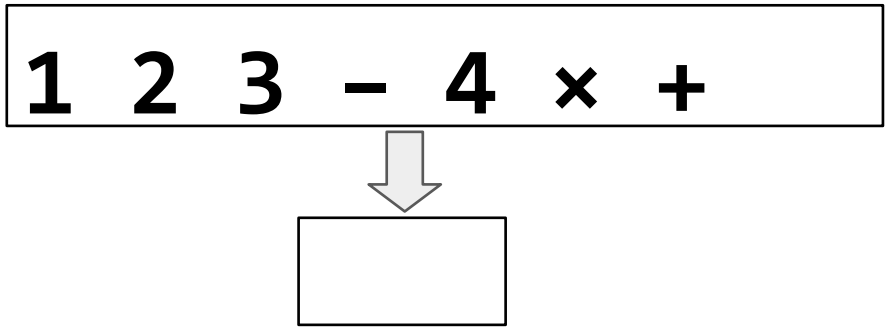
Given an expression in Reverse Polish Notation it is fairly straightforward to evaluate it.

```
1 2 3 - 4 × +
```

-3

RPN evaluation

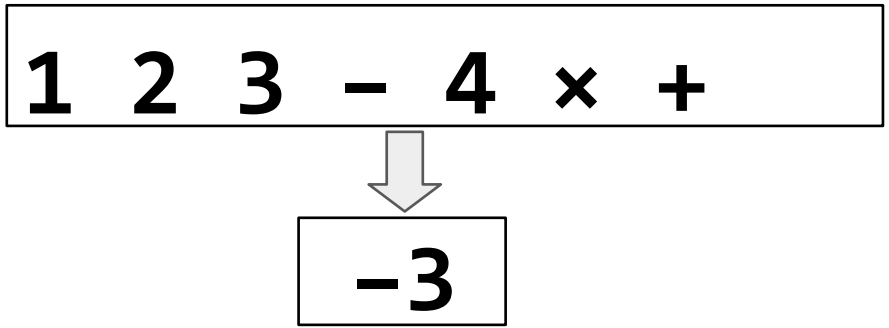
Given an expression in Reverse Polish Notation it is fairly straightforward to evaluate it.



-3

RPN evaluation

Given an expression in Reverse Polish Notation it is fairly straightforward to evaluate it.



- Discuss the algorithm.
- What are the corner cases?
- What data structures would you use? Why?

Input Queue	Evaluation Stack
1 2 3 − 4 × +	
2 3 − 4 × +	1
3 − 4 × +	1 2
− 4 × +	1 2 3
4 × +	1 −1
× +	1 −1 4
+	1 −4
	−3

See you next week!