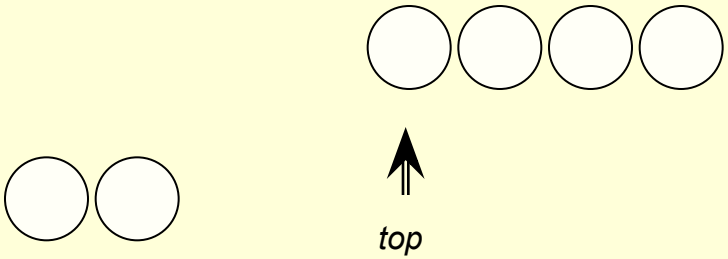
# Стеки и очереди

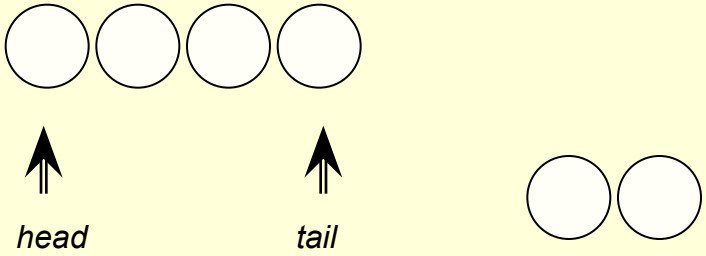
## 1. Абстрактный стек

```
public interface Stack {
    static class Underflow extends Exception {
        public Underflow() { super("Stack underflow"); }
    };

    void    push(Object element);
    Object  pop() throws Underflow;
    Object  peek() throws Underflow;
    boolean empty();
}
```



# 2. Абстрактная очередь

```
public interface Queue {
    static class Underflow extends Exception {
        public Underflow() { super(“Queue underflow"); }
    };

    void    enqueue(Object element);
    Object  dequeue() throws Underflow;
    Object  head() throws Underflow;
    Object  tail() throws Underflow;
    boolean empty();
}
```



# Различные подходы к реализации стека

```
public interface List {
    // Elements counting
    boolean isEmpty();
    int size();

    // Access to elements
    Object first();
    Object last();

    // Changes in list
    Object addFirst(Object o);
    Object addLast(Object o);
    Object removeFirst();
    Object removeLast();

    // List iteration
    void iterate(Visitor visitor);
    Iterator iterator();
}
```

```
public class LinkedStack
             extends AbstractStack {
    public LinkedStack()
    { super(new LinkedList()); }
}
```

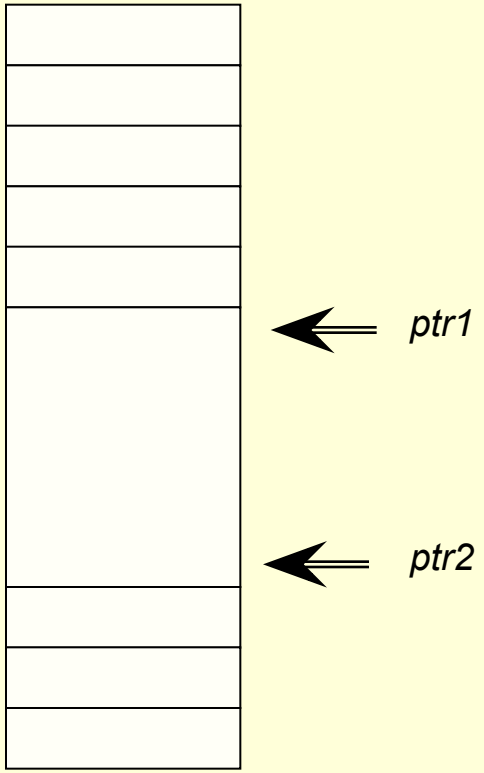
```
public abstract class MyStack
                implements List, Stack {
    boolean empty() { return isEmpty(); }
    Object peek() { return first(); }
    Object push(Object o)
           { return addFirst(o); }
    Object pop() { return removeFirst(); }
}
```

```
public abstract class AbstractStack
                implements Stack {
    private List list;

    public AbstractStack(List list)
    { this.list = list; }

    public boolean isEmpty()
    { return list.isEmpty(); }

    public Object push(Object o)
    { return list.addFirst(o); }

    public Object pop()
    { return list.removeFirst(); }

    public Object peek()
    { return list.first(); }
}
```

# Реализация стека в виде массива

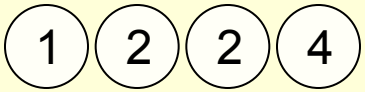
```
public class ArrayStack implements Stack {
    private Object[] stack;   // Массив стековых элементов
    private int topPtr;       // Число элементов в стеке - указатель вершины

    static class Underflow extends Exception {
        public Underflow() { super("Stack underflow"); }
    }

    public ArrayStack(int maxElems) {
        stack = new Object[maxElems];
        topPtr = 0;
    }

    public ArrayStack() { this(100); }

    public boolean empty() { return topPtr == 0; }

    public Object push(Object element) {
        if (topPtr == stack.length) throw new Overflow();
        return stack[topPtr++] = element;
    }

    public Object peek() throws Underflow {
        if (topPtr == 0) throw new Underflow();
        return stack[topPtr-1];
    }

    public Object pop() throws Underflow {
        if (topPtr == 0) throw new Underflow();
        return stack[--topPtr];
    }
}
```

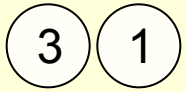
# Реализация пары стеков в массиве




```
public class StackPair {
    private Object[] stack;
    int ptr1, ptr2;

    public StackPair(int max) {
        stack = new Object[max];
        ptr1 = 0; ptr2 = stack.length-1;
    }

    public StackPair() { this(100); }

    Object push1(Object element) throws Stack.Overflow {
        if (ptr1 > ptr2) throw new Stack.Overflow();
        return stack[ptr1++] = element;
    }

    Object push2(Object element) throws Stack.Overflow {
        if (ptr1 > ptr2) throw new Stack.Overflow();
        return stack[ptr2--] = element;
    }

    ...

    boolean empty1() { return ptr1 == 0; }
    boolean empty2() { return ptr2 == stack.length-1; }
}
```

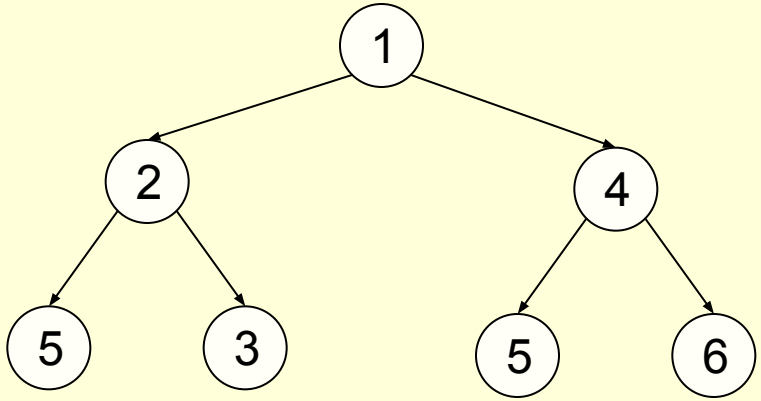
# Очередь с приоритетами




```
public interface Prioritized {
    int getPrio();
    void setPrio(int prio);
}

public class PrioQueue implements Queue {
    Object enqueue(Object element) {...}
    Object dequeue() throws Underflow {...}
    Object head() throws Underflow {...}
    Object tail() throws Underflow
                  { throw new RuntimeException("tail: no implementation"); }
    boolean empty() {...}
    Prioritized setPrio(Prioritized obj, int prio)
}
```

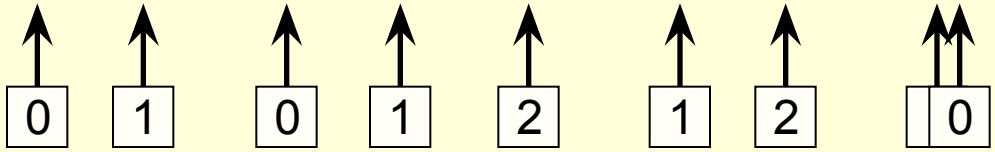
Пирамида – один из возможных способов реализации очереди с приоритетами:



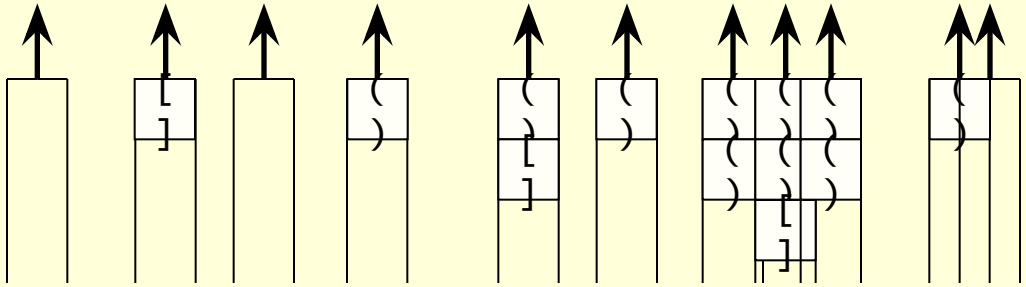
# Применение стеков для анализа выражений

1. Проверка правильности расстановки скобок.

```
1 + (a + b) * (2 – (c – d) * (a + b))
```




```
1 + a[i+1] * (2 – c[i–1] * (a[i] + 1))
```

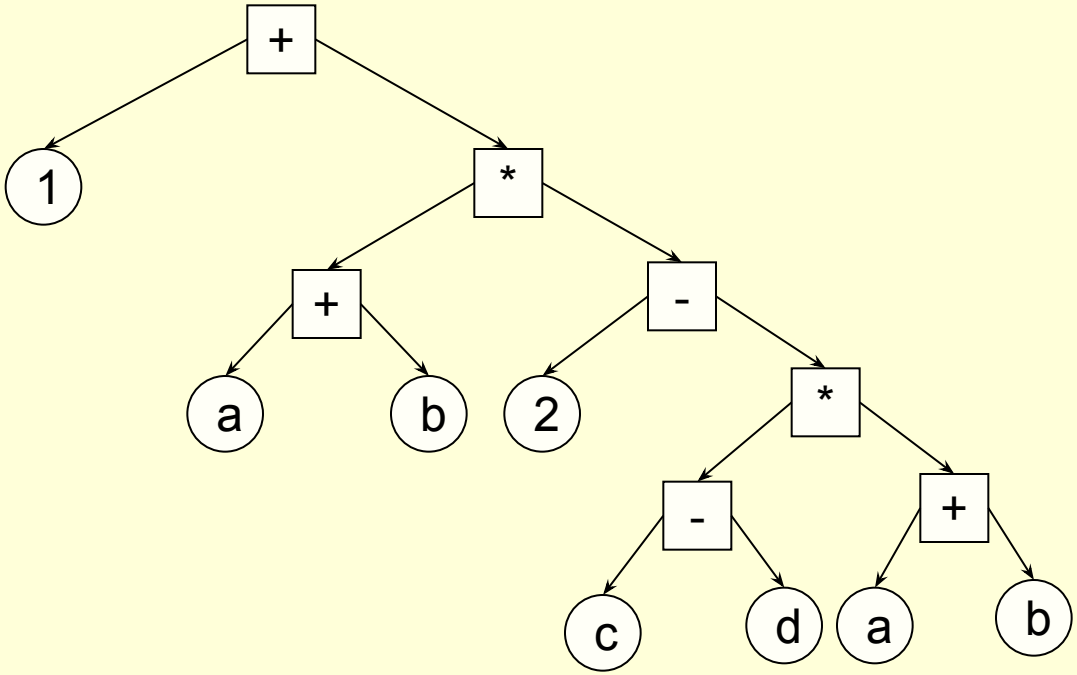


# Реализация анализа правильности расстановки скобок

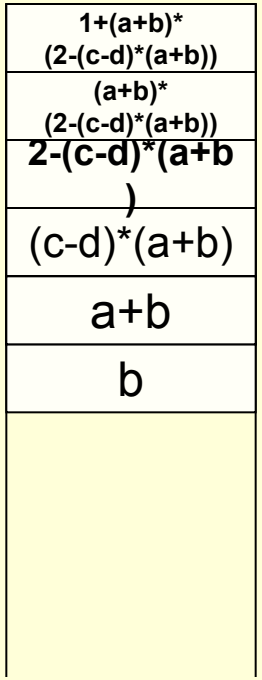
```
public static boolean parentheses(String openBrackets,
                                  String closeBrackets,
                                  String source)
{
    Stack pars = new LinkedStack();
    for (int i = 0; i < source.length(); i++) {
        char c = source.charAt(i);

        // 1. Проверка открывающей скобки
        int j = openBrackets.indexOf(c);
        if (j >= 0) {
            pars.push(new Character(closeBrackets.charAt(j)));
            continue;
        }

        // 2. Проверка закрывающей скобки
        j = closeBrackets.indexOf(c);
        if (j >= 0) {
            try {
                if (!pars.pop().equals(new Character(c)))
                    return false;
            } catch (Stack.Underflow u) {
                return false;
            }
        }
    }
    return pars.empty();
}
```

# Перевод выражения в обратную польскую запись

```
1 + (a + b) * (2 – (c – d) * (a + b))
```




```
Loadc 1
Load  a
Load  b
Add
Loadc 2
Load  c
Load  d
Sub
Load  a
Load  b
Add
Mult
Sub
Mult
Add
```



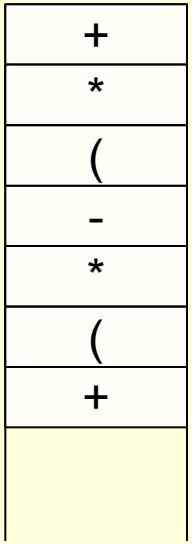

```
1 a b + 2 c d – a b + * – * +
```

# Алгоритм анализа выражения

```
1 + (a + b) * (2 – (c – d) * (a + b))
```

Для каждой из лексем:
- если это первичное, выводим его;
- если это ‘(‘, кладем ее в стек;
- если это операция, выводим не менее приоритетные и кладем ее знак в стек;
- если это ‘)’, выводим операции до знака ‘(‘ и вычеркиваем скобку из стека.

В конце работы выводим все операции из стека.




```
1 a b + 2 c d – a b + * – * +
```