# Problem A

# Overview

- There are only 1000 queries max
- So algorithms with quadratic complexity work

# Optimization

- Deque/Linked List
- Allow 'N' query to be done in constant time
- Also allow 'E x' to be done faster than using an array

# Alternatives

- STL set and/or map
- Can achieve Clog(min(P,C) ) complexity

# Problem B

# Overview

- Bit manipulation problem
- May be less tricky than problem A

# Main problems

- Find all neighbors of a corner
- Toggle each bit in the mask
- Trick: use xor operation

# Problem C

# Overview

- Classical problem
- Variations may appear in programming contests like ACM ICPC (KL 2011 problem C, Dhaka 2006 problem D)
- Simple algorithm, main problem lies in implementation

# Methods

- Convert into postfix notation or abstract syntax tree then evaluate the value
- Scan the string for operations that should be evaluated first.
- Use Java ScriptEngine, likely not usable for the different variations of this problem.

# Postfix conversion (Shunting-yard algorithm)

- Put numbers on the output queue
- Push operators on stack
- Pop out operators that should be used first to output queue

```
1.          for (String s : tokens){
2.              int a = 0;
3.              try{
4.                  a = Integer.parseInt(s);
5.                  evalQue.add( a );
6.              } catch (NumberFormatException e){ //if not a number
7.                  if (s.equals(")") ){//pop until "("
8.                      char op = ')';
9.                      do{
10.                         op = ops.pop();
11.                         if (op != '(') evalQue.add(op);
12.                     }while (op != '(');
13.                 } else {
14.                     char op = s.charAt(0);
15.                     if (op != '('){   //pop all that should be calculated first
16.                         int curPred = pred.get(op);
17.                         while (!ops.isEmpty() && ops.peek() != '(' && pred.get( ops.peek()) >=
    curPred){
18.                             evalQue.add(ops.pop());
19.                         }
20.                     }
21.                     ops.push(op);
22.                 }
23.             }
24.         }
25.         while (!ops.isEmpty()) evalQue.add(ops.pop());
```