Задача 1.

Для решения этой задачи, необходимо лишь понять, как именно функция от различного количества аргументов будет записана в формате обратной польской записи.

Для функции факториала практически ничего не изменится:

5! = 5 ! . Функция факториал просто применяется к последнему аргументу перед ней.

(2 + 3)! = 2 3 + !

Для функции функция max записывается в обратной польской даже короче, чем в изначальной форме:

max(2, 3, 4) = 2 3 4 max . Больше не требуются скобки и запятые — функция применяется к последним трем аргуентам.

max(2+3, 4*5, 7) = 2 3 + 4 5 * 7 max

Таким образом в классе вычислений добавим две функции факториала и максимума. Первая будет брать только один элемент из стека, вторая же целых три и вычислять соответствующие значения.

В классе парсинга, добавим приоритеты функциям, добавим пробелы между функциями, уберем все запятые из текста, а также будет раньше добавлять функцию максимума на стек, так как она записывается до выражения.

```
#include <iostream>
#include <algorithm>
#include <vector>
#include <string>
#include <stack>
#include <sstream>
#include <map>
class PolishCompute{
public:
    PolishCompute(){}
    ~PolishCompute(){}
    long int compute(std::string e) {
        std::stringstream ss(e); // создаем поток, чтобы было удобно из него
читать
        std::string current;
        while(ss >> current) { // считываем элементы из строки
            if(current == "+") add(); // вызываем соответствующую функцию для
каждого оператора
            else if(current == "-") subl();
            else if(current == "*") mult();
            else if(current == "/") div();
            else if(current == "max") maximum();
            else if(current == "!") fac();
            else st.push(current); // и просто кладем на стек, если это число
        }
        long int result = std::stoi(st.top());
        st.pop();
        return result;
    }
private:
    void add() {
        long int x = std::stoi(st.top()); st.pop();
        long int y = std::stoi(st.top()); st.pop();
        st.push(std::to_string(y + x)); // считаем значение оператора для
последних двух чисел на стеке
    }
    void subl() {
        long int x = std::stoi(st.top()); st.pop();
```

```
        long int y = std::stoi(st.top()); st.pop();
        st.push(std::to_string(y - x));
    }
    void mult() {
        long int x = std::stoi(st.top()); st.pop();
        long int y = std::stoi(st.top()); st.pop();
        st.push(std::to_string(y * x));
    }
    void div() {
        long int x = std::stoi(st.top()); st.pop();
        long int y = std::stoi(st.top()); st.pop();
        st.push(std::to_string(y / x));
    }
    void fac() {
        long int x = std::stoi(st.top()); st.pop();
        int result = 1;
        for(int i = 2; i <= x; i++) result *= i;
        st.push(std::to_string(result));
    }
    void maximum() {
        long int x = std::stoi(st.top()); st.pop();
        long int y = std::stoi(st.top()); st.pop();
        long int z = std::stoi(st.top()); st.pop();
        st.push(std::to_string(std::max(std::max(x, y), z)));
    }
    std::stack<std::string> st;
};
class ComputeEngine {
public:
    ComputeEngine(){
        priority["("] = 0;
        priority[")"] = 0;
        priority["+"] = 1; // выставляем приоритет операциям.
        priority["-"] = 1; // Само значение не важно - главное чтобы их можно
было сравнивать
        priority["*"] = 2;
        priority["/"] = 2;
        priority["!"] = 3;
        priority["max"] = 4;
    }
    ~ComputeEngine(){}
    int compute(std::string e) {
        return pc.compute(to_polish(e)); // вычисляем значение строки,
переведенной в польскую нотацию
    }
    std::string to_polish(std::string e) {
        add_spaces(e); // приведем к удобному для четния формату
        std::stringstream result; // итоговый (сдела потоком для удобства)
        std::stack<std::string> ops; // операторы
        std::stringstream ss(e); // поток для удоства
        std::string current; // текущий
        while(ss >> current) { // считываем элементы из ввода
            if(priority.count(current) == 1) { // если для данного символа
указан приоритет, то это оператор или скобка
                if (current == "(") {
                    ops.push(current);
                    continue;
                }
                if (current == ")") {
```

```
                    while (ops.top() != "(") { // пока не дойдем до открывающей
                        result << ops.top() << " "; // добавляем в строку
                        ops.pop(); // удаляем
                    }
                    ops.pop();
                    continue;
                }
                if (current == "max" ) {
                    ops.push(current);
                    continue;
                }
                // если оператор
                while (!ops.empty() and priority[current] <=
priority[ops.top()]) { // пока больше приоритет
                    result << ops.top() << " "; // добавляем к строке
                    ops.pop(); // удаляем
                }
                ops.push(current);
            } else { // число или факториал
                result << current << " ";
            }
        }
        while(!ops.empty()) { // оставшиеся операторы
            result << ops.top() << " ";
            ops.pop();
        }
        std::string polish_string;
        std::getline(result, polish_string);
        return polish_string;
    }
private:
    // для удобного перевода в польскую нотацию, значащие символы
    // но обычно строка вводится без пробелов
    // эта функция добавляет эти пробелы
    void add_spaces(std::string &e) {
        std::stringstream result;
        for(auto s : e) {
            if(priority.count(std::string(1, s)) == 1) result << ' ' << s << '
';
            else if(s == '(' or s == ')') result << ' ' << s << ' ';
            else if(s == '!') result << " ! ";
            else if(s == ',') result << ' ';
            else result << s;
        }
        std::getline(result, e);
    }
    PolishCompute pc;
    std::map<std::string, int> priority;
};
int main() {
    ComputeEngine ce;
    std::cout << ce.compute("(2 + 3) * 7") << std::endl;
    std::cout << ce.compute("max(6, 2, 3) + 2") << std::endl;
    std::cout << ce.compute("max(6, 2, 3)! + 3") << std::endl;
    std::cout << ce.compute("max( 5+7, 12!, 10*14 ) * 7") << std::endl;
    return 0;
}
```

Задание 2.

В данной задаче главное — правильно организовать перебор возможных вариантов. Так как скобки могу расставляться как угодно, то перебрать все возможные случаи в обычной нотации будет очень сложно, однако мы знаем, что есть однозначное соответствие между обычной нотацией и обратной польской, а в обратной польской выражения задаются без скобок.

В таком случае, будем организовывать перебор сразу в обратной польской.

Для N чисел необходимо расставить N-1 знак. Будем идти по всем расстановкам знаков. Всего их 4^(N-1). Заметим, что если перевести такое число в 4-ную форму записи, то i — ая цифра будет указывать на номер оператора на i — том месте.

Пронумеруем операторы. 0 = +, 1 = -, 2 = *, 3 = /. В таком случае число 1203_4 будет означать расстановку - * + /. Для того, чтобы получить i — ую цифру в таком числе, необходимо поделить на 4^i и потом взять остаток от деления на 4.

Осталось правильно расставить каждую расстановку между числами. Однако не всякая конфигурация нам подойдет. Например 2 + 3 + - некорректная конфигурация, так как до + должна идти как минимум 2 агрумента. В таком случае будет следовать следующей стратегии: у нас будет функция, которая будет вставлять определенное количество операторов после указанного числа. Эта функция будет знать, сколько максимум оператором она может вставить. Таким образом она будет расставлять каждое из возможных количеств операторов после конкретного числа , а после этого вызывать себя же на следующем числе.

Таким образом функция переберет все возможные варианты расстановок. Взяв уже написанный класс для вычисления обратной польской записи осталось проверить каждый из вариантов на равенство S. Если хоть раз получилось — значит ответ Yes, если нет, то No.

```
template <class Iter>
bool check_configuration(Iter pos_begin, Iter pos_end,
                         Iter num_begin, Iter num_end,
                         int ans,
                         int remain = 0, std::string s = "") {
    bool result = false;
    if(num_begin == num_end - 1) { // последнее число
        std::string newS = s + *num_begin+ " ";
        while(pos_begin != pos_end) {
            newS += *pos_begin + " ";
            pos_begin++;
        }
        PolishCompute pc;
        result = result or (pc.compute(newS) == ans);
    } else {
        for(int i = 0; i <= remain; i++) { // пробуем вставить все варианты,
которые нам доступны
            std::string newS = s + *num_begin + " ";
            for(int j = 0; j < i; j++) newS += *(pos_begin+j) + " ";
            result = result or check_configuration(pos_begin+i, pos_end,
                                                   num_begin+1, num_end, ans, //
указываем уже следующее число
                                                   remain-i+1, newS); //
отнимаем то количество операторов, которое использовали после этого числа и
добавляем 1
        }
    }
    return result;
}
int main() {
    int N, S;
    std::cin >> N >> S;
```

```
    std::vector<std::string> nums(N);
    for(int i = 0; i < N; i++ ) std::cin >> nums[i];
    std::vector<std::string> ops {"+", "-", "*", "/"};
    bool result = false;
    for(long Q = 0; Q < pow(4, N-1); Q++) {
        std::vector<std::string> curr_conf(N-1);
        for(int i = 0; i < N-1; i++) {
            curr_conf[i] = ops[ int(Q / pow(4, i)) % 4 ]; // вычисляем
расстановку из операторов
        }
        result = result or check_configuration(curr_conf.begin(),
curr_conf.end(), nums.begin(), nums.end(), S);
    }
    std::cout << (result ? "Yes" : "No") << std::endl;
    return 0;
}
```