Задача 1.

Так как все элементы располагаются в случайном месте оперативной памяти, то у нас нет доступа к произвольному элементу списка. В таком случае, нам ничего не остается, кроме как просто идти по списки и проверять каждый элемент на соответствие.

Для удаления элемента нам необходимо для начала найти элемент, соединить два соседних с ним элемента и после этого удалить сам элемент.

```
class DoubleNode {
public:
    DoubleNode() : right(nullptr), left(nullptr) {}
    DoubleNode(int v) : value(v), right(nullptr), left(nullptr) {}
    ~DoubleNode(){}
    int value;
    DoubleNode* right;
    DoubleNode* left;
};
class DoubleList {
public:
    DoubleList() : begin(nullptr), end(nullptr), right(true) {}
    ~DoubleList() {
        DoubleNode* current = begin;
        DoubleNode* toDel;
        while(current != nullptr) { // удаляем все элементы
            toDel = current;
            current = right ? current->right : current->left;
            delete toDel;
        }
    }
    void Insert(int value) {
        DoubleNode* e = new DoubleNode(value);
        if(begin == nullptr) end = e;
        if(right) e->right = begin;
        else e->left = begin;
        if(begin != nullptr) {
            if(right) begin->left = e;
            else begin->right = e;
        }
        begin = e;
    }
    DoubleNode* Search(int value) {
        DoubleNode* x = begin;
        while(x != nullptr and x->value != value) {
            if(right) x = x->right;
            else x = x->left;
        }
        return x;
    }
    void Delete(DoubleNode* e) {
        if(e == nullptr) return;
        if(right){
            if(e->left != nullptr) {
                e->left->right = e->right;
            } else {
                begin = e->right;
            }
            if(e->right != nullptr) {
                e->right->left = e->left;
            } else {
```

```
                end = e->left;
            }
        } else {
            if(e->left != nullptr) {
                e->left->right = e->right;
            } else {
                end = e->right;
            }
            if(e->right != nullptr) {
                e->right->left = e->left;
            } else {
                begin = e->left;
            }
        }
        delete e;
    }
    void Delete(int value) {
        Delete(Search(value));
    }
    void for_each(std::function<void(int)> f) { // конструкция для удобного
итерирования по коллекции
        DoubleNode* current = begin;
        while(current != nullptr) {
            f(current->value);
            if(right) current = current->right;
            else current = current->left;
        }
    }
    DoubleNode * begin;
    DoubleNode * end;
    bool right; // указатель на то, в какую сторону двигаться
};
int main() {
    DoubleList n;
    for(int i = 0; i < 10; i++) {
        n.Insert(i);
    }
    n.Delete(3);
    n.Delete(7);
    n.for_each([](int s){
        std::cout << s << ' ';
    });
    std::cout << std::endl;
    return 0;
}
```

Задача 2.

Связанный список очень просто использовать как стек.

- Операция Push эквивалентна операции Insert
- Операция Top эквивалентна операции взятия значения первого элемента списка
- Операция Pop эквивалентна удалению первого элемента списка.

Также, для наших целей поменяет тип хранимого значения с int на std::string в списке.
Для удобства будем хранить размер связного списка. В остальном осталось только заменить необходимые операции на соответствующие в списке и у нас будет абсолютно свой движок арифметических вычислений.

```
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 st.Insert(current); // и просто кладем на стек, если это число
        }
        long int result = std::stoi(st.begin->value);
        st.Delete(st.begin);
        return result;
    }
private:
    void add() {
        long int x = std::stoi(st.begin->value); st.Delete(st.begin);
        long int y = std::stoi(st.begin->value); st.Delete(st.begin);
        st.Insert(std::to_string(y + x)); // считаем значение оператора для
последних двух чисел на стеке
    }
    void subl() {
        long int x = std::stoi(st.begin->value); st.Delete(st.begin);
        long int y = std::stoi(st.begin->value); st.Delete(st.begin);
        st.Insert(std::to_string(y - x));
    }
    void mult() {
        long int x = std::stoi(st.begin->value); st.Delete(st.begin);
        long int y = std::stoi(st.begin->value); st.Delete(st.begin);
        st.Insert(std::to_string(y * x));
    }
    void div() {
        long int x = std::stoi(st.begin->value); st.Delete(st.begin);
        long int y = std::stoi(st.begin->value); st.Delete(st.begin);
        st.Insert(std::to_string(y / x));
    }
    DoubleList 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;
    }
    ~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)
        DoubleList 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.Insert(current);
                    continue;
                }
                if(current == ")") {
                    while(ops.begin->value != "(") { // пока не дойдем до
открывающей
                        result << ops.begin->value << " "; // добавляем в строку
                        ops.Delete(ops.begin); // удаляем
                    }
                    ops.Delete(ops.begin);
                    continue;
                }
                // если оператор
                while(!ops.empty() and priority[current] <= priority[ops.begin-
>value]) { // пока больше приоритет
                    result << ops.begin->value << " "; // добавляем к строке
                    ops.Delete(ops.begin); // удаляем
                }
                ops.Insert(current);
            } else { // число
                result << current << " ";
            }
        }
        while(!ops.empty()) { // оставшиеся операторы
            result << ops.begin->value << " ";
            ops.Delete(ops.begin);
        }
        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 result << s;
        }
        std::getline(result, e);
    }
    PolishCompute pc;
    std::map<std::string, int> priority;
};
```

```cpp
int main() {
    ComputeEngine ce;
    std::cout << ce.to_polish("2+3") << std::endl;
    std::cout << ce.to_polish("4 + 6*5") << std::endl;
    std::cout << ce.compute("(78 + 5 * 4 ) / (4 + 10)") << std::endl;
    return 0;
}
```