**МИНОБРНАУКИ РОССИИ**
**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**
**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**
**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**
**КАФЕДРА САПР**

# Отчет по лабораторной работе № 1
по дисциплине «Алгоритмы и структуры данных»
Тема: «Списки»
Вариант 9

Студент гр. 9302 Чугуй В. В.

Преподаватель Тутуева А. В.

Санкт-Петербург
2019

# Оглавление

## Цель работы

Получить практические навыки в разработке алгоритма и написании программы на языке C++ для знакомства с синтаксисом, а также правилами написания кода на языке C++.

## Задание

Реализовать класс связного списка с набором методов.

**Тип списка**:

односвязный список.

**Реализуемые функции**:

1. void push_back(int); // добавление в конец списка
2. void push_front(int); // добавление в начало списка
3. void pop_back(); // удаление последнего элемента
4. void pop_front(); // удаление первого элемента
5. void insert(int, size_t) // добавление элемента по индексу (вставка перед элементом, который был ранее доступен по этому индексу)
6. int at(size_t); // получение элемента по индексу. Можно сделать типа size_t
7. void remove(size_t); // удаление элемента по индексу
8. size_t get_size(); // получение размера списка
9. void print_to_console(); // вывод элементов списка в консоль через разделитель, не использовать at
10. void clear(); // удаление всех элементов списка
11. void set(size_t, int); // замена элемента по индексу на передаваемый элемент
12. bool isEmpty(); // проверка на пустоту списка
13. bool contains(List); // проверка на содержание другого списка в списке

## Описание реализуемого класса и методов

В данной работе реализован класс List который представляет из себя список записей и набор методов для работы с этим списком. В private части данного класса находится класс Node (запись) который представляет из себя какие-то данные со ссылкой на следующий элемент. Так же там присутствуют переменные, необходимые для работы списка, такие как голова списка, конец списка и его размер.

В public части списка находятся методы для работы со списком. Для начала это конструктор и деструктор (List и ~List). Также метод get_head для получения головы списка. Далее реализованы методы из списка задания. Назначение методов push_back и push_front интуитивно понятно, можно лишь сказать, что в данных методах есть одно условие для их корректной работы. Это условие пустоты списка. Для методов pop_back и pop_front условие похожее, только если список пуст, то методы ничего не делают, различие в работе находится только в условии количества записей в списке (если запись одна, то не происходит прохода по циклу). Для метода insert сделана проверка на выход за пределы списка, а так условия похожи, если список пуст, то не происходит проход циклом по списку и запись вставляется в начало. Метод at схож по коду с методом insert, метод at вместо создания новой записи просто присваивает другое значение информации в уже существующую запись. Метод remove очень неудобно выполнять для односвязного списка, так как для удаления одной записи необходимо циклом дойти до записи, идущей перед необходимой, запомнить запись идущей следующей за необходимой, удалить выбранную запись и перепривязать две записи. В связи с неудобным алгоритмом в данном методе лучше его разбивать на частные случаи и общий случай. Метод get_size просто возвращает значение переменной размера списка, описанной в private части класса List. Метод print_to_console выполняет циклом вывод списка на экран. Метод clear выполняет очистку списка путём поочередного удаления всех записей. Метод set выполняет банальную замену записей, если бы в записи было только одно поле с

информацией, то можно было бы заменить просто значение этой информации, но я реализовал полную замену, те создание новой записи, удаление старой и вставка новой записи на место старой. Метод isEmpty я решил сделать проверкой на не пустоту (значение nullptr) переменных голова и конец списка. Далее идёт метод contains. Метод должен возвращать одно из логических значений, либо список состоит в другом списке, либо нет. Метод реализуется двумя вложенными циклами, суть которых проста, первым циклом перебираем начальную точку отсчёта сравнения списков, вторым перебираем второй список. Если получится так, что от какой-то начальной записи одного списка информация в нём совпадёт с информацией в другом, то метод возвращает true, если этого не произойдёт, то метод вернёт значение false.

## Оценка временной сложности методов

Методы List, ~ List, push_back, push_front, pop_front, get_size, isEmpty имеют временную сложность O(1), так как в них нет никаких циклов и в общем случае они выполняются в одно-два действия.

Методы pop_back, insert, at, remove, print_to_console, clear, set имеют временную сложность O(N), так как они содержат в себе один цикл, в большинстве случаев (кроме частных) будет выполнен проход по циклу, поэтому их временная сложность равно O(N).

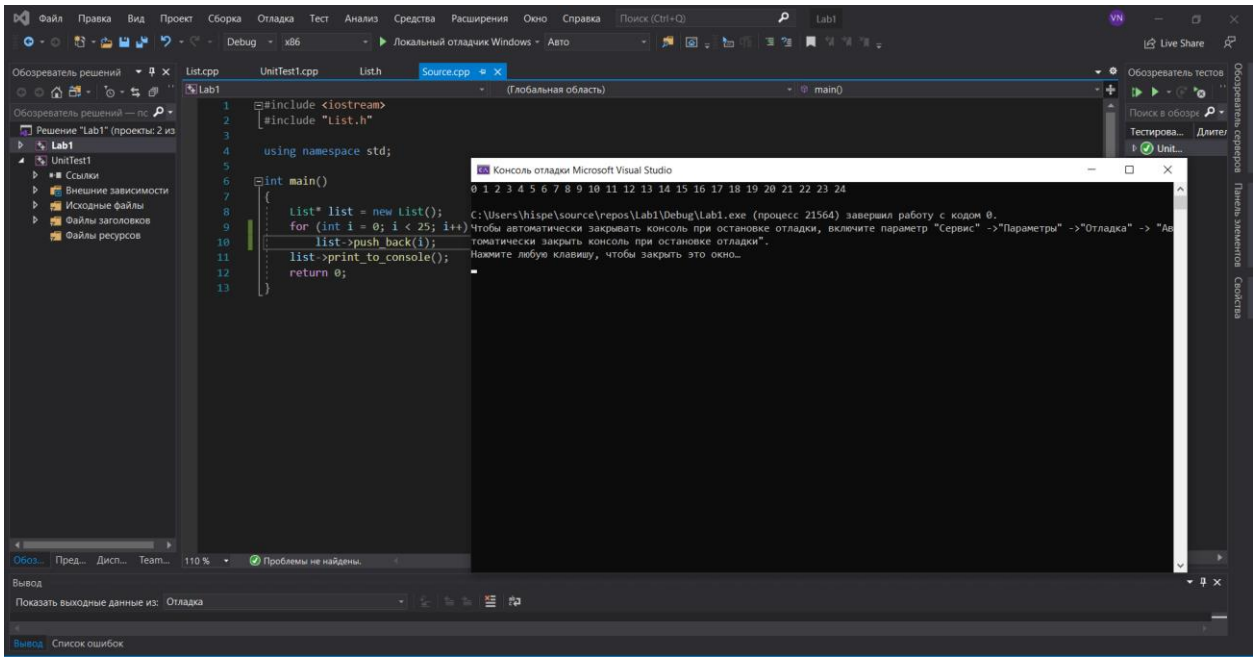
Метод contains имеет временную сложность O(N*M), где N – длина первого списка и M – длина второго, так как в нём содержатся вложенные циклы, размером равные двум размерам списков, которые могут различаться.

## Описание реализованных unit-тестов

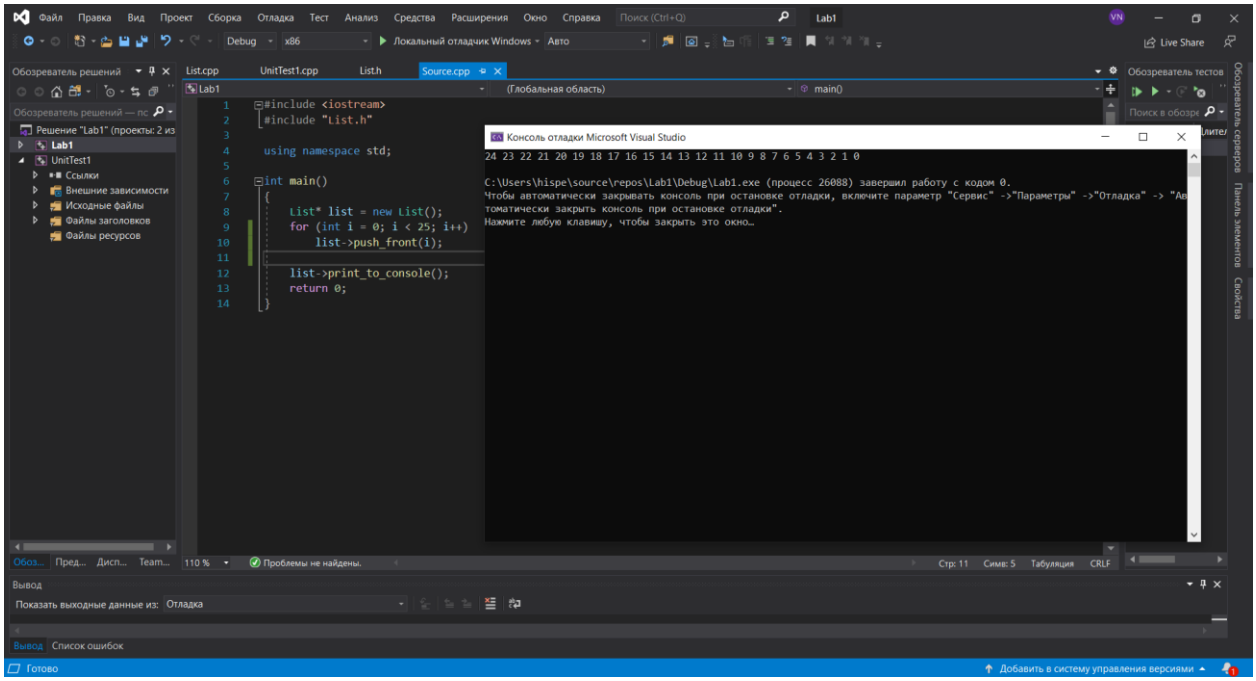
Для юнит тестов сделаны два дополнительных метода: TEST_METHOD_INITIALIZE и TEST_METHOD_CLEANUP. Данные методы сделаны для запуска какого-то кода до запуска каждого юнит теста и после каждого юнит теста. В данной работе они создают и удаляют класс List. Для каждой функции было создано по несколько юнит тестов, которые проверяют тот или иной случай использования метода. Например, для метода push_back сделано два юнит теста проверяющих работу метода при пустом списке и при уже существующем. То же самое сделано и для метода push_front. Для метода pop_back созданы два теста, первый проверяет удалился ли последний элемент, второй проверяет очищает ли этот метод при единственной записи в списке. Для pop_front тест делает проверку на удаление первой записи и очистку списка. Тест для метода insert проверяет работу вставки и выход за пределы списка, а также вставку в пустой список. Для метода remove тесты проверяют удаление конкретной записи и выход за пределы списка. Тесты для get_size проверяют изменение размера списка, при работе других методов. Тесты для clear проверяют очистку для списка с различным количеством записей. Для метода set юнит тесты проверяют измененные данные в конкретных позициях, а также выход за пределы списка. Юнит тесты для метода contains проверяют содержание одного списка в другом (пустой список выполняет условие содержания в другом).

## Пример работы программы

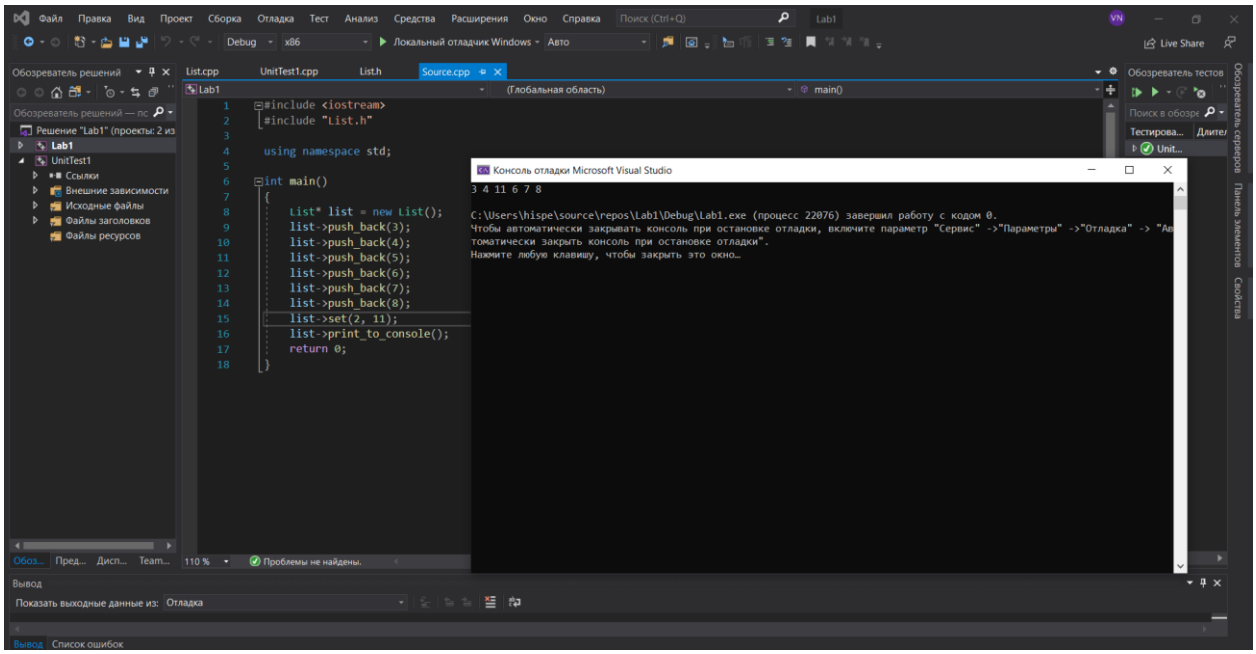
Создание списка при помощи push_back:




Создание списка при помощи push_front:



Замена данных методом set:



## Заключение

**Выводы:**

При выполнении лабораторной работы были получены практические навыки в разработке алгоритмов решения задач, а также изучение синтаксиса языка C++.

## Ссылка на программу

https://github.com/VolodyaZAVR/ALGSTR_Lab1