**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: «Изучение и практическое применение однонаправленных списков. Реализация модульного тестирования»**

**Вариант 11**

Студент гр. 9302 \_\_\_\_\_\_\_\_\_\_\_\_ Ковтун А.С.

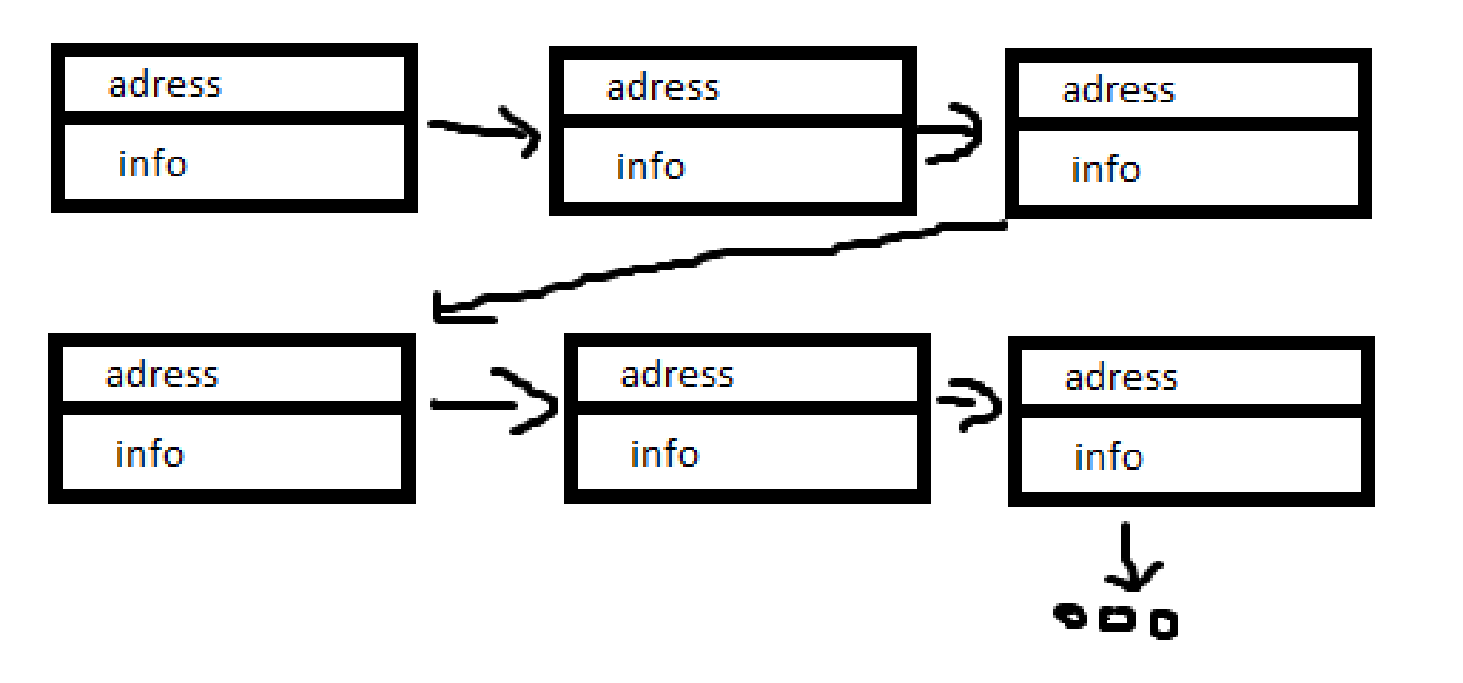
Преподаватель \_\_\_\_\_\_\_\_\_\_\_\_ Тутуева А.В.

Санкт-Петербург

2020

* **Постановка задачи и описание реализуемого класса и методов**

Реализовать класс связного однонаправленного списка с набором методов. Используем для этой задачи класс элементов elem, информацию о списке будет хранить в классе list.



|  |  |  |
| --- | --- | --- |
| **Метод** | **Описание** | **Оценка временной сложности** |
| void push\_back(int) | Добавление в конец списка | O(n) |
| void push\_front(int) | Добавление в начало списка | O(n) |
| void pop\_back() | Удаление последнего элемента | O(n-1) |
| void pop\_front() | Удаление первого элемента | O(1) |
| void insert(int, size\_t) | Добавление элемента по индексу | O(n) |
| int at(size\_at) | Получение элемента по индексу | O(n) |
| void remove(size\_t) | Удаление элемента по индексу | O(n) |
| size\_t getSize() | Получение размера списка | O(1) |
| void print\_to\_console() | Вывод элементов в консоль | O(n) |
| void clear() | Удаление всех элементов списка | O(n) |
| void set(size\_t, int) | Замена элемента по индексу на передаваемый элемент | O(n) |
| bool isEmpty() | Проверка на пустоту списка | O(1) |
| size\_t find\_first(list) | Нахождение индекса первого элемента вхождения другого списка в список | O(n\*m),n-размер первого списка,m-размер второго списка |

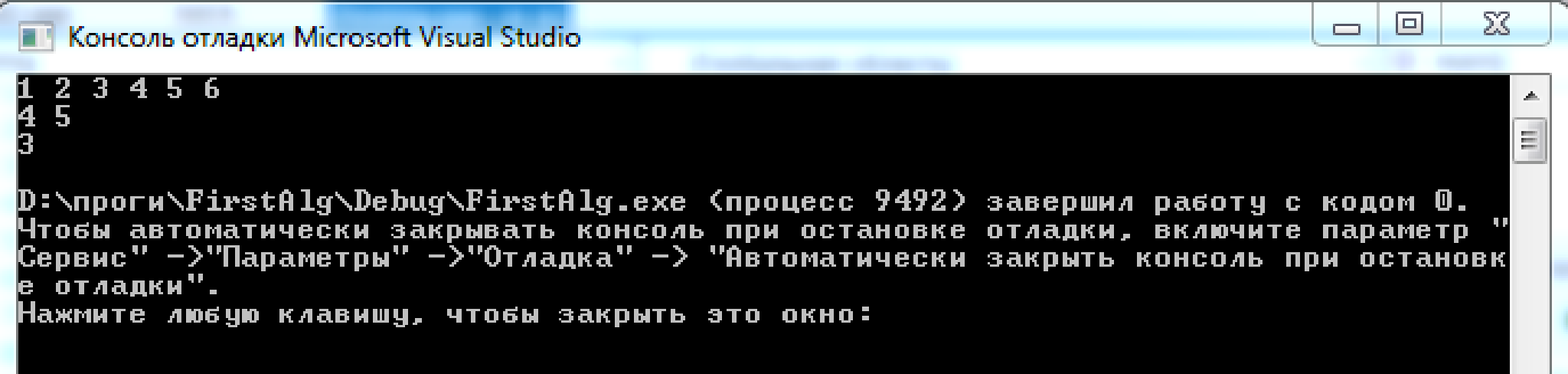
* **Описание реализованных unit-тестов**

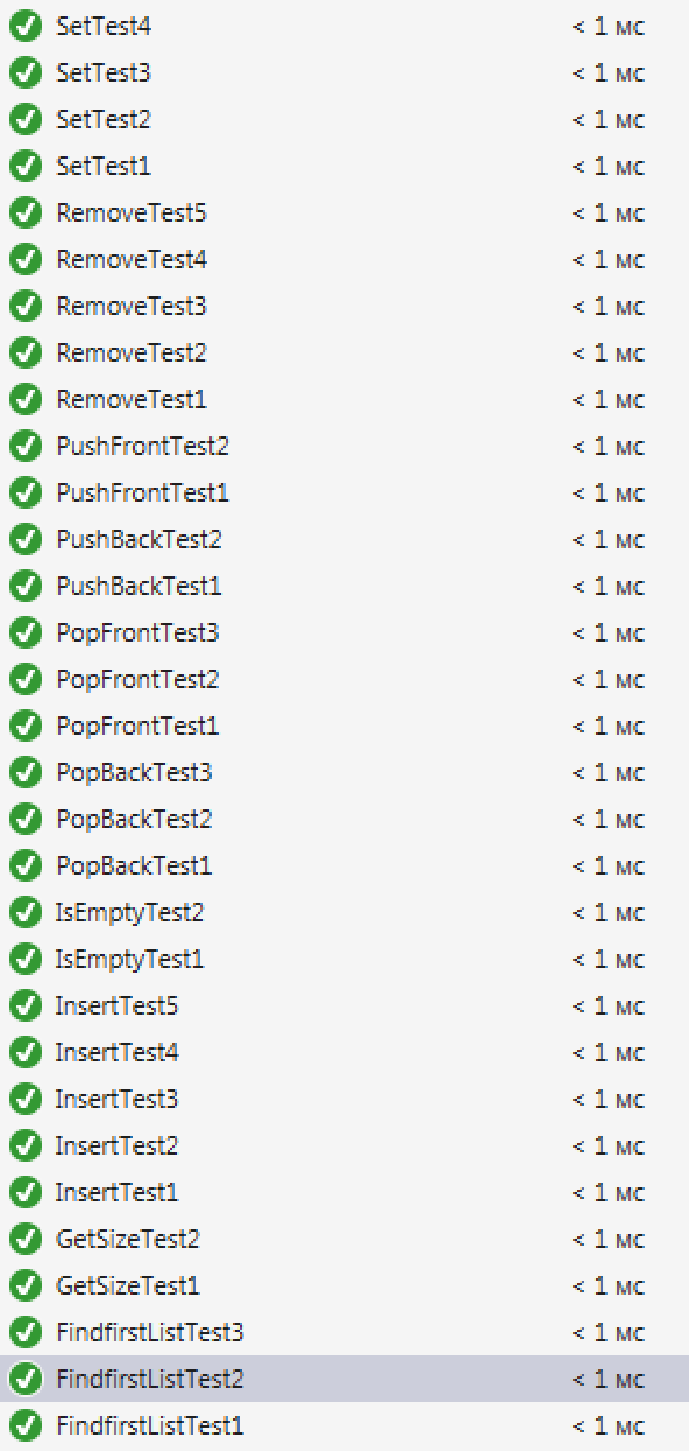
|  |  |
| --- | --- |
| Имя теста | Описание |
| CreateTest | Проверка работы конструктора |
| PushBackTest1 | Проверка добавления элемента в конец пустого списка |
| PushBackTest2 | Проверка добавления элемента в конец непустого списка |
| PushFrontTest1 | Проверка добавления элемента в начало пустого списка |
| PushFrontTest2 | Проверка добавления элемента в начало непустого списка |
| PopBackTest1 | Проверка удаления элемента из конца непустого списка |
| PopBackTest2 | Проверка удаления элемента из конца списка с одним элементом |
| PopBackTest3 | Проверка удаления элемента из конца пустого списка |
| PopFrontTest1 | Проверка удаления элемента из начала непустого списка |
| PopFrontTest2 | Проверка удаления элемента из начала списка с одним элементом |
| PopFrontTest3 | Проверка удаления элемента из начала пустого списка |
| InsertTest1 | Проверка вставки элемента в пустой список на первую позицию |
| InsertTest2 | Проверка вставки элемента в пустой список на вторую позицию |
| InsertTest3 | Проверка вставки элемента в непустой список на посл. позицию |
| InsertTest4 | Проверка вставки элемента в непустой список на первую позицию |
| InsertTest5 | Проверка вставки элемента в середину непустого списка |
| AtTest1 | Проверка получения элемента по индексу |
| AtTest2 | Проверка получения элемента с первым индексом из пустого списка |
| RemoveTest1 | Проверка удаления элемента из пустого списка |
| RemoveTest2 | Проверка удаления элемента из непустого списка с посл. позиции |
| RemoveTest3 | Проверка удаления элемента из непустого списка с первой позиции |
| RemoveTest4 | Проверка удаления элемента из середины непустого списка |
| RemoveTest5 | Проверка удаления элемента с несуществующим индексом |
| GetSizeTest1 | Проверка получения размера пустого списка |
| GetSizeTest2 | Проверка получения размера непустого списка |
| SetTest1 | Проверка замены значения элемента непустого списка |
| SetTest2 | Проверка замены значения элемента с несуществующим индексом |
| SetTest3 | Проверка замены значения элемента непустого списка на такое же |
| SetTest4 | Проверка двойной замены элемента непустого списка |
| IsEmpty1 | Проверка определения пустоты списка |
| IsEmpty2 | Проверка определения пустоты списка |
| PushFrontListTest1 | Проверка вхождения одного списка в другой |
| PushFrontListTest2 | Проверка вхождения одного списка в другой |

* **Код программы**

|  |
| --- |
| list.h  #pragma once  #include <iostream>  #ifndef LIST\_H  #define LIST\_H  class elem {  private:  int inf;  elem\* next;  public:  elem(int elem\_inf) { inf = elem\_inf; next = nullptr;}  ~elem() {};  elem\* getNext() { return next; }  void setNext(elem\* elem\_next) { next = elem\_next; }  int getInf() { return inf; }  void setInf(int elem\_inf) { inf = elem\_inf; }  };  class list {  private:  elem\* begin, \* end;  size\_t size;  public:  list() { begin = nullptr; end = nullptr; size = 0;}  ~list() { clear(); }  //Adding an element to the end of the list  void push\_back(int temp) {  elem\* newElem = new elem(temp);  if (isEmpty()) begin = newElem;  else end->setNext(newElem);  end = newElem;  size++;  }  //Adding an element to the begin of the list  void push\_front(int temp) {  elem\* newElem = new elem(temp);  if (isEmpty()) end = newElem;  else newElem->setNext(begin);  begin = newElem;  size++;  }  //Deleting last element of the list  void pop\_back(){  if (!isEmpty()) {  if (size == 1) {  end = nullptr;  begin = nullptr;  size = 0;  }  else {  elem\* newEnd = begin;  while (newEnd->getNext() != end) newEnd = newEnd->getNext();  newEnd->setNext(nullptr);  elem\* deleted = end;  delete deleted;  end = newEnd;  size--;  }  }  else throw "List is Empty";  }  //Deleting first element of the list  void pop\_front() {  if (!isEmpty()) {  if (size == 1) {  begin = nullptr;  end = nullptr;  size = 0;  }  else {  elem\* newBeg = begin->getNext();  elem\* deleted = begin;  delete deleted;  begin = newBeg;  size--;  }  }  else throw "List is Empty";  }  //Adding an element to any position in the list  void insert(int data, size\_t pos) {  if (pos <= size) {  if (isEmpty() || pos == size) push\_back(data);  else {  if (pos == 0) push\_front(data);  else {  elem\* newElem = new elem(data);  elem\* iter = begin;  while (pos-- > 1)  iter = iter->getNext();  newElem->setNext(iter->getNext());  iter->setNext(newElem);  size++;  }  }  }  else throw "Wrong index";  }  //Getting an element by index  int at(size\_t pos) {  if (pos < size) {  elem\* iter = begin;  while (pos-- != 0) iter = iter->getNext();  return iter->getInf();  }  else throw "Wrong index";  }  //Deleting an element by index  void remove(size\_t pos) {  if (pos < size) {  if (pos == 0) pop\_front();  else {  if (pos == size - 1) pop\_back();  else {  elem\* iter = begin;  while (pos-- > 1) iter = iter->getNext();  elem\* nextElem = iter->getNext();  iter->setNext(nextElem->getNext());  size--;  }  }  }  else throw "Wrong index";  }  //Getting size of the list  size\_t getSize() { return size; }  //Output an elements from the list to console  void print\_to\_console() {  elem\* iter = begin;  for (size\_t i = 0; i < size; i++) {  std::cout << iter->getInf() << " ";  iter = iter->getNext();  }  std::cout << std::endl;  }  //Deleting an elements of the list  void clear() {  while (size) pop\_back();  }  //Replacing an element by index with new one  void set(size\_t pos, int data) {  if (pos < size) {  elem\* iter = begin;  while (pos-- != 0) iter = iter->getNext();  iter->setInf(data);  }  else throw "Wrong index";  }  //Checking the list for completeness  bool isEmpty() {  if (size == 0) return true; // 1 - Empty  else return false; // 0 - Filled  }  //Finding the first occurrence of another list in the list  size\_t find\_first(list \*lst) {  size\_t counter = 0;  size\_t firstpos = 0;  size\_t counter1 = 0;  size\_t counter2 = 0;  size\_t counter3 = 0;  while (counter2 < size) {  if (at(counter2) == lst->at(0)) {  firstpos = counter2; // memorizing the position, and then checking whether the list is included in the list  counter3 = counter2;  for (size\_t i = 0; i < lst->getSize(); i++) {  if (lst->at(i) == at(counter3)) {  counter++;  counter3++;  //std::cout << lst->at(i) << std::endl;  if (counter == lst->getSize()) {  counter1 = counter;  }  }  else {  counter = 0;  break;  }  }  counter2 = counter3;  }  if (counter1 == lst->getSize()) break;  counter2++;  }  if (counter1 == lst->getSize()) {  std::cout << firstpos << std::endl;  return firstpos;  }  else {  throw "The entry is missing";  }  }  };  #endif |
| FirstAlg.cpp |
| #include <iostream>  #include "list.h"  int main() {  list list1, list2;  list1.push\_back(1);  list1.push\_back(2);  list1.push\_back(3);  list1.push\_back(4);  list1.push\_back(5);  list1.push\_back(6);  list2.push\_back(4);  list2.push\_back(5);  list1.print\_to\_console();  list2.print\_to\_console();  list1.find\_first(&list2);  return 0;  } |
| UnitTest2.cpp  #include "pch.h"  #include "CppUnitTest.h"  #include "D:\проги\FirstAlg\FirstAlg\list.h"  #include <iostream>  using namespace Microsoft::VisualStudio::CppUnitTestFramework;  namespace ListTests  {  TEST\_CLASS(ListTests)  {  public:  TEST\_METHOD(CreateTest)  {  list\* list1 = new list();  Assert::IsTrue(list1->getSize() == 0);  }  TEST\_METHOD(PushBackTest1) {  list\* list1 = new list();  list1->push\_back(1);  Assert::IsTrue(list1->at(0) == 1 && list1->getSize() == 1);  }  TEST\_METHOD(PushBackTest2) {  list\* list1 = new list();  list1->push\_back(1);  list1->push\_back(2);  Assert::IsTrue(list1->at(0) == 1 && list1->at(1) == 2 && list1->getSize() == 2);  }  TEST\_METHOD(PushFrontTest1) {  list\* list1 = new list();  list1->push\_front(1);  Assert::IsTrue(list1->at(0) == 1 && list1->getSize() == 1);  }  TEST\_METHOD(PushFrontTest2) {  list\* list1 = new list();  list1->push\_front(1);  list1->push\_front(2);  Assert::IsTrue(list1->at(0) == 2 && list1->at(1) == 1 && list1->getSize() == 2);  }  TEST\_METHOD(PopBackTest1) { // 2 elements  list\* list1 = new list();  list1->push\_back(55);  list1->push\_back(232);  list1->pop\_back();  Assert::IsTrue(list1->at(0) == 55 && list1->getSize() == 1);  }  TEST\_METHOD(PopBackTest2) { // 1 element  list\* list1 = new list();  list1->push\_back(77);  list1->pop\_back();  Assert::IsTrue(list1->getSize() == 0);  }  TEST\_METHOD(PopBackTest3) { // empty list  list\* list1 = new list();  try {  list1->pop\_back();  }  catch (const char\* warning) {  Assert::AreEqual(warning, "List is Empty");  }  Assert::IsTrue(list1->getSize() == 0);  }  TEST\_METHOD(PopFrontTest1) { // 2 elements  list\* list1 = new list();  list1->push\_back(2);  list1->push\_back(1);  list1->pop\_front();  Assert::IsTrue(list1->at(0) == 1 && list1->getSize() == 1);  }  TEST\_METHOD(PopFrontTest2) { // 1 element  list\* list1 = new list();  list1->push\_back(1);  list1->pop\_front();  Assert::IsTrue(list1->getSize() == 0);  }  TEST\_METHOD(PopFrontTest3) { // empty list  list\* list1 = new list();  try {  list1->pop\_front();  }  catch (const char\* warning) {  Assert::AreEqual(warning, "List is Empty");  }  Assert::IsTrue(list1->getSize() == 0);  }  TEST\_METHOD(InsertTest1) { // check func insert  list\* list1 = new list();  list1->insert(5, 0);  Assert::IsTrue(list1->at(0) == 5 && list1->getSize() == 1);  }  TEST\_METHOD(InsertTest2) {  list\* list1 = new list();  try {  list1->insert(5, 1);  }  catch (const char\* warning) {  Assert::AreEqual(warning, "Wrong index");  }  Assert::IsTrue(list1->getSize() == 0);  }  TEST\_METHOD(InsertTest3) {  list\* list1 = new list();  list1->push\_back(1);  list1->insert(5, 1);  Assert::IsTrue(list1->at(0) == 1 && list1->at(1) == 5 && list1->getSize() == 2);  }  TEST\_METHOD(InsertTest4) {  list\* list1 = new list();  list1->push\_back(2);  list1->insert(1, 0);  Assert::IsTrue(list1->at(0) == 1 && list1->at(1) == 2 && list1->getSize() == 2);  }  TEST\_METHOD(InsertTest5) {  list\* list1 = new list();  list1->push\_back(1);  list1->push\_back(3);  list1->insert(2, 1);  Assert::IsTrue(list1->at(0) == 1 && list1->at(1) == 2 && list1->at(2) == 3 && list1->getSize() == 3);  }  TEST\_METHOD(AtTest1) { // check func at  list\* list1 = new list();  list1->push\_back(1);  list1->push\_back(2);  list1->push\_back(3);  Assert::IsTrue(list1->at(0) == 1 && list1->at(1) == 2 && list1->at(2) == 3);  }  TEST\_METHOD(AtTest2) {  list\* list1 = new list();  list1->push\_back(1);  try {  list1->at(1);  }  catch (const char\* warning) {  Assert::AreEqual(warning, "Wrong index");  }  }  TEST\_METHOD(RemoveTest1) { // check function remove  list\* list1 = new list();  try {  list1->remove(1);  }  catch (const char\* warning) {  Assert::AreEqual(warning, "Wrong index");  }  Assert::IsTrue(list1->getSize() == 0);  }  TEST\_METHOD(RemoveTest2) {  list\* list1 = new list();  list1->push\_back(1);  list1->push\_back(2);  list1->push\_back(3);  list1->remove(2);  Assert::IsTrue(list1->at(0) == 1 && list1->at(1) == 2 && list1->getSize() == 2);  }  TEST\_METHOD(RemoveTest3) {  list\* list1 = new list();  list1->push\_back(3);  list1->push\_back(1);  list1->push\_back(2);  list1->remove(0);  Assert::IsTrue(list1->at(0) == 1 && list1->at(1) == 2 && list1->getSize() == 2);  }  TEST\_METHOD(RemoveTest4) {  list\* list1 = new list();  list1->push\_back(1);  list1->push\_back(3);  list1->push\_back(2);  list1->remove(1);  Assert::IsTrue(list1->at(0) == 1 && list1->at(1) == 2 && list1->getSize() == 2);  }  TEST\_METHOD(RemoveTest5) {  list\* list1 = new list();  list1->push\_back(1);  list1->push\_back(2);  list1->push\_back(3);  try {  list1->remove(3);  }  catch (const char\* warning) {  Assert::AreEqual(warning, "Wrong index");  }  Assert::IsTrue(list1->at(0) == 1 && list1->at(1) == 2 && list1->at(2) == 3 && list1->getSize() == 3);  }  TEST\_METHOD(GetSizeTest1) {  list\* list1 = new list();  Assert::IsTrue(list1->getSize() == 0);  }  TEST\_METHOD(GetSizeTest2) {  list\* list1 = new list();  list1->push\_back(1);  list1->push\_back(2);  list1->push\_back(3);  Assert::IsTrue(list1->getSize() == 3);  }  TEST\_METHOD(SetTest1) { // check function set  list\* list1 = new list();  list1->push\_back(0);  list1->push\_back(2);  list1->push\_back(3);  list1->set(0, 1);  Assert::IsTrue(list1->at(0) == 1 && list1->at(1) == 2 && list1->at(2) == 3 && list1->getSize() == 3);  }  TEST\_METHOD(SetTest2) {  list\* list1 = new list();  list1->push\_back(1);  list1->push\_back(2);  list1->push\_back(3);  try {  list1->set(3, 4);  }  catch (const char\* warning) {  Assert::AreEqual(warning, "Wrong index");  }  Assert::IsTrue(list1->at(0) == 1 && list1->at(1) == 2 && list1->at(2) == 3 && list1->getSize() == 3);  }  TEST\_METHOD(SetTest3) {  list\* list1 = new list();  list1->push\_back(1);  list1->push\_back(2);  list1->push\_back(3);  list1->set(0, 1);  Assert::IsTrue(list1->at(0) == 1 && list1->at(1) == 2 && list1->at(2) == 3 && list1->getSize() == 3);  }  TEST\_METHOD(SetTest4) {  list\* list1 = new list();  list1->push\_back(1);  list1->push\_back(2);  list1->push\_back(3);  list1->set(0, 2);  list1->set(0, 1);  Assert::IsTrue(list1->at(0) == 1 && list1->at(1) == 2 && list1->at(2) == 3 && list1->getSize() == 3);  }  TEST\_METHOD(IsEmptyTest1) {  list\* list1 = new list();  Assert::IsTrue(list1->isEmpty());  }  TEST\_METHOD(IsEmptyTest2) {  list\* list1 = new list();  list1->push\_front(1);  Assert::IsTrue(!list1->isEmpty());  }  TEST\_METHOD(FindfirstListTest1) { // check func find\_first  list\* list1 = new list();  list\* list2 = new list();  list1->push\_back(0);  list1->push\_back(1);  list1->push\_back(2);  list1->push\_back(33);  list1->push\_back(4);  list1->push\_back(5);  list2->push\_back(33);  list2->push\_back(4);  list2->push\_back(5);  Assert::IsTrue(list1->find\_first(list2) == 3);  }  TEST\_METHOD(FindfirstListTest2) { //the first list consists of a repeating second list  list\* list1 = new list();  list\* list2 = new list();  list1->push\_back(0);  list1->push\_back(1);  list1->push\_back(0);  list1->push\_back(1);  list1->push\_back(0);  list1->push\_back(1);  list2->push\_back(1);  list2->push\_back(0);  Assert::IsTrue(list1->find\_first(list2) == 1);  }  TEST\_METHOD(FindfirstListTest3) {  list\* list1 = new list();  list\* list2 = new list();  list1->push\_back(0);  list1->push\_back(1);  list1->push\_back(2);  list1->push\_back(3);  list1->push\_back(4);  list1->push\_back(5);  list2->push\_back(5);  Assert::IsTrue(list1->find\_first(list2) == 5);  }  };  } |

* **Пример работы**





* **Вывод**

Выполняя лабораторную работу, я вспомнил, как работать со списками, а также научился реализовывать unit-тесты.