**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 7**

Студент гр. 9302 \_\_\_\_\_\_\_\_\_\_\_\_ Тарабурин А.П.

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

Санкт-Петербург

2020

Оглавление

[1. Постановка задачи и описание реализуемого класса и методов 3](#_Toc55082721)

[2. Описание реализованных unit-тестов 3](#_Toc55082722)

[3. Код программы 4](#_Toc55082723)

[4. Пример работы 11](#_Toc55082724)

[5. Вывод 12](#_Toc55082725)

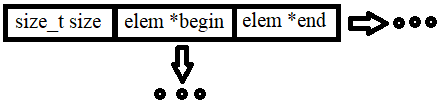
# Постановка задачи и описание реализуемого класса и методов

Реализовать класс связного однонаправленного списка с набором методов. Используем для этой задачи класс элементов elem и обеспечим хранение информации о списке в классе list.

Class elem



Class list



|  |  |  |
| --- | --- | --- |
| **Метод** | **Описание** | **Оценка временной сложности** |
| void push\_back(int) | Добавление в конец списка | O(1) |
| void push\_front(int) | Добавление в начало списка | O(1) |
| void pop\_back() | Удаление последнего элемента | O(1) |
| void pop\_front() | Удаление первого элемента | O(1) |
| void insert(int, size\_t) | Добавление элемента по индексу | O(n) |
| int at(const int) | 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 через разделитель | 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) |
| void push\_front(list) | Вставка другого списка в начало | O(n) |

# Описание реализованных unit-тестов

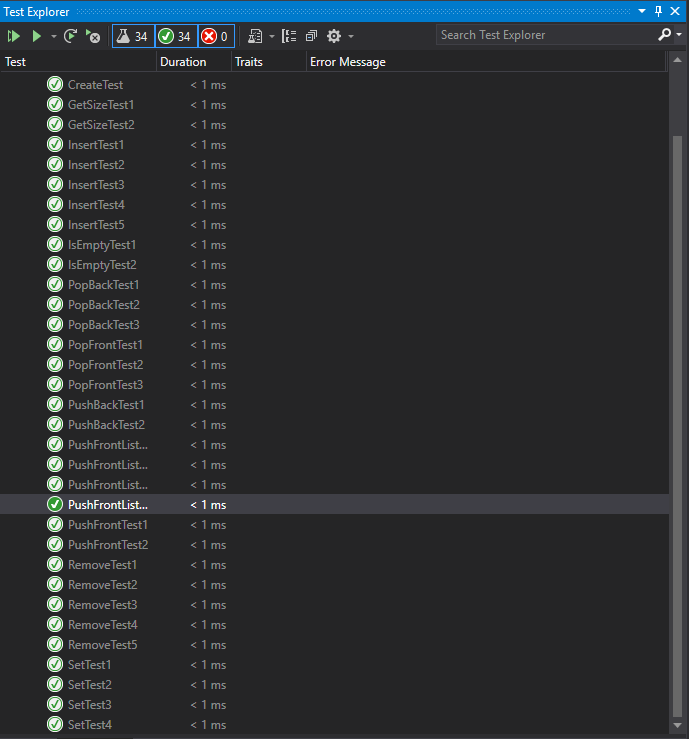
|  |  |
| --- | --- |
| Имя теста | Описание |
| CreateTest | Проверка работы конструктора |
| PushBackTest1 | Проверка добавления элемента в конец пустого списка |
| PushBackTest2 | Проверка добавления элемента в конец непустого списка |
| PushFrontTest1 | Проверка добавления элемента в начало пустого 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 |
| AtTest | Проверка получения элемента по индексу |
| 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 вставки одного непустого в начало другого непустого |
| PushFrontListTest2 | Проверка вставки одного пустого в начало другого непустого |
| PushFrontListTest3 | Проверка вставки одного непустого в начало другого пустого |
| PushFrontListTest4 | Проверка вставки одного пустого в начало другого пустого |

# 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() { delete[] begin; delete[] end;}  //Adding new element to end of list  void push\_back(int temp) {  elem\* newelem = new elem(temp);  if (!isEmpty()) end->setNext(newelem);  end = newelem;  if (isEmpty()) begin = end;  size++;  }  //Adding new element to begin of list  void push\_front(int temp) {  elem\* newElem = new elem(temp);  if(!isEmpty()) newElem->setNext(begin);  begin = newElem;  if (isEmpty()) end = begin;  size++;  }  //Deleting last element from 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);  end = newEnd;  size--;  }  }  }  //Deleting first element from list  void pop\_front() {  if (!isEmpty()) {  if (size == 1) {  begin = nullptr;  end = nullptr;  size = 0;  }  else {  elem\* newBeg = begin->getNext();  begin = newBeg;  size--;  }  }  }  //Adding element to any position in 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++;  }  }  }  }  //Getting element from list by index  int at(size\_t pos) {  elem\* iter = begin;  while (pos-- != 0) iter = iter->getNext();  return iter->getInf();  }  //Deleting element from list 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--;  }  }  }  }  //Getting size of list  size\_t getSize() { return size; }  //Output elements from 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();  }  }  //Deleting elements of list  void clear() {  while (size) pop\_back();  }  //Replacing 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);  }  }  //Checking list for filling  bool isEmpty() {  if (size == 0) return true; // 1 - Empty  else return false; // 0 - Filled  }  //Adding another list to front of this one  void push\_front(list \*lst) {  for (size\_t i = 0; i < lst->getSize(); i++)  insert(lst->at(i), i);  }  };  #endif |
| Main.cpp |
| #include <iostream>  #include "list.h"  int main() {  list list1, list2;  list1.push\_back(3);  list1.push\_back(5);  list1.push\_front(1);  list1.push\_back(7);  //std::cout << list1.at(1);  list2.push\_back(2);  list2.push\_back(4);  list2.push\_back(6);  list2.push\_back(8);  list1.push\_front(&list2);  list1.print\_to\_console();  return 0;  } |
| UnitTest1.cpp |
| #include "pch.h"  #include "CppUnitTest.h"  #include "..\Project1\Project1\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(2);  list1->push\_front(1);  Assert::IsTrue(list1->at(0) == 1 && list1->at(1) == 2 && list1->getSize() == 2);  }  TEST\_METHOD(PopBackTest1) { // 2 elements  list\* list1 = new list();  list1->push\_back(1);  list1->push\_back(2);  list1->pop\_back();  Assert::IsTrue(list1->at(0) == 1 && list1->getSize() == 1);  }  TEST\_METHOD(PopBackTest2) { // 1 element  list\* list1 = new list();  list1->push\_back(1);  list1->pop\_back();  Assert::IsTrue(list1->getSize() == 0);  }  TEST\_METHOD(PopBackTest3) { // empty list  list\* list1 = new list();  list1->pop\_back();  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();  list1->pop\_front();  Assert::IsTrue(list1->getSize() == 0);  }  TEST\_METHOD(InsertTest1) { // empty list insert at first place  list\* list1 = new list();  list1->insert(1, 0);  Assert::IsTrue(list1->at(0) == 1 && list1->getSize() == 1);  }  TEST\_METHOD(InsertTest2) { // empty list insert at second place - out of range  list\* list1 = new list();  list1->insert(1, 1);  Assert::IsTrue(list1->getSize() == 0);  }  TEST\_METHOD(InsertTest3) { // same as push\_back  list\* list1 = new list();  list1->push\_back(1);  list1->insert(2, 1);  Assert::IsTrue(list1->at(0) == 1 && list1->at(1) == 2 && list1->getSize() == 2 );  }  TEST\_METHOD(InsertTest4) { // same as push\_front  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) { // insert to middle  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(AtTest) {  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(RemoveTest1) { // empty list remove  list\* list1 = new list();  list1->remove(1);  Assert::IsTrue(list1->getSize() == 0);  }    TEST\_METHOD(RemoveTest2) { // same as pop\_back  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) { // same as pop\_front  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) { // remove from middle  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) { // out of range check  list\* list1 = new list();  list1->push\_back(1);  list1->push\_back(2);  list1->push\_back(3);  list1->remove(3);  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) {  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) { //out of range  list\* list1 = new list();  list1->push\_back(1);  list1->push\_back(2);  list1->push\_back(3);  list1->set(3, 1);  Assert::IsTrue(list1->at(0) == 1 && list1->at(1) == 2 && list1->at(2) == 3 && list1->getSize() == 3);  }  TEST\_METHOD(SetTest3) { //set same  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) { //set twice  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(PushFrontListTest1) {  list\* list1 = new list();  list\* list2 = new list();  list2->push\_back(1);  list2->push\_back(2);  list2->push\_back(3);  list1->push\_back(4);  list1->push\_back(5);  list1->push\_back(6);  list1->push\_front(list2);  Assert::IsTrue(list1->at(0) == 1 && list1->at(1) == 2 && list1->at(2) == 3 && list1->at(3) == 4 && list1->at(4) == 5 && list1->at(5) == 6 && list1->getSize() == 6);  }  TEST\_METHOD(PushFrontListTest2) { //pushed list is empty  list\* list1 = new list();  list\* list2 = new list();  list1->push\_back(1);  list1->push\_back(2);  list1->push\_back(3);  list1->push\_front(list2);  Assert::IsTrue(list1->at(0) == 1 && list1->at(1) == 2 && list1->at(2) == 3 && list1->getSize() == 3);  }  TEST\_METHOD(PushFrontListTest3) { //first list is empty  list\* list1 = new list();  list\* list2 = new list();  list2->push\_back(1);  list2->push\_back(2);  list2->push\_back(3);  list1->push\_front(list2);  Assert::IsTrue(list1->at(0) == 1 && list1->at(1) == 2 && list1->at(2) == 3 && list1->getSize() == 3);  }  TEST\_METHOD(PushFrontListTest4) { //both lists are empty  list\* list1 = new list();  list\* list2 = new list();  list1->push\_front(list2);  Assert::IsTrue(list1->getSize() == 0);  }  };  } |

# Пример работы





# Вывод

При выполнении данной лабораторной работы я познакомился с модульным тестированием написанного кода.