|  |  |
| --- | --- |
| **Российский университет транспорта (МИИТ)**  **Институт транспортной техники и систем управления**  **Кафедра «Управление и защита информации»** | |
| **Отчёт**  **по лабораторной работе №2**  **по дисциплине «Системы управления базы данных»** | |
|  | Выполнил:  Студент группы ТКИ-441  Николаев А.М.  Проверила:  Доцент кафедры УиЗИ, к.т.н.  Васильева М. А. |
| Москва 2022 | |

s**Оглавление**

[Цель работы 3](#_Toc121749258)

[Задание 3](#_Toc121749259)

[1. Код программы 3](#_Toc121749260)

[1.1. Файл Element.h 3](#_Toc121749261)

[1.2. Файл MyList.h 3](#_Toc121749262)

[1.3. Файл Element.cpp 5](#_Toc121749263)

[1.4. Файл MyList.cpp 5](#_Toc121749264)

[1.5. Файл ElementTests.cpp 9](#_Toc121749265)

[1.6. Файл MyListTests.cpp 10](#_Toc121749266)

[2. UML-диаграмма 14](#_Toc121749267)

[3. Результаты работы 14](#_Toc121749268)

[Вывод 15](#_Toc121749269)

Цель работы

Получить навыки по программированию на языках высокого уровня с использованием ООП и получить навыки по работе со структурами данных.

Задание

1. Создать класс, описывающий Линейный Двусвязный Список.
2. Создать методы, необходимые для добавления, поиска, изменения и удаления элементов данных, хранимых при помощи данной структуры.
3. Предоставить тестовый набор данных для демонстрации работы ПО.
4. Код программы
   1. Файл Element.h

#pragma once

#include <iostream>

#include <sstream>

using namespace std;

class Element

{

public:

// Значение элемента

int data;

// Указатели на следующий и предыдущий элементы

Element\* next;

Element\* prev;

/\*\*

\* \brief Конструктор класса

\*/

Element(const int data\_) noexcept;

/\*\*

\* \brief Деструктор класса

\*/

~Element() noexcept;

};

* 1. Файл MyList.h

#pragma once

#include <iostream>

#include <sstream>

#include "Element.h"

using namespace std;

class List

{

public:

// Указатели на начало и конец списка

Element\* head;

Element\* tail;

// Количество элементов списка

size\_t count;

/\*\*

\* \brief Конструктор класса

\*/

List() noexcept;

/\*\*

\* \brief Деструктор класса

\*/

~List() noexcept;

/\*\*

\* \brief Проверяет есть такой элемент в списке

\* \param value - значение элемента

\* \return true - если есть такой элемент, иначе - false

\*/

bool Contains(const int value) const noexcept;

/\*\*

\* \brief Функция для удаления элемента списка по индексу

\* \param pos - индекс элемента

\*/

void Delete(const size\_t pos);

/\*\*

\* \brief Функция для удаления элемент списка по значению

\* \param value - значение элемента

\*/

void Remove(const int value);

/\*\*

\* \brief Функция встраивания элемента

\* \param value - значение элемента

\*/

void Insert(const int value, const size\_t pos);

/\*\*

\* \brief Функция добавления элемента в конец списка

\* \param value - значение элемента

\*/

void AddTail(const int value) noexcept;

/\*\*

\* \brief Функция добавления элемента в начало списка

\* \param value - значение элемента

\*/

void AddHead(const int value) noexcept;

/\*\*

\* \brief Функция сохранения в строку списка

\* \return строка содержащая список

\*/

string ToString() const noexcept;

/\*\*

\* \brief Копирующий конструктор.

\*/

List(const List& other) noexcept;

/\*\*

\* \brief Перемещающий конструктор.

\*/

List(List&& other) noexcept;

/\*\*

\* \brief Оператор перемещения

\*/

List& operator = (List&& other) noexcept;

/\*\*

\* \brief Оператор копирования

\*/

List& operator = (const List& other) noexcept;

};

* 1. Файл Element.cpp

#include "Element.h"

Element::Element(const int data\_) noexcept : data(data\_), next(nullptr), prev(nullptr)

{}

Element::~Element() noexcept

{

this->data = 0;

this->next = nullptr;

this->prev = nullptr;

}

* 1. Файл MyList.cpp

#include "MyList.h"

List::List() noexcept : head(nullptr), tail(nullptr), count(0)

{}

List::~List()

{

while (this->count != 0)

{

Delete(0);

}

}

void List::AddHead(const int value) noexcept

{

Element\* temp = new Element(value);

temp->prev = nullptr;

temp->next = this->head;

if (this->head != nullptr)

{

this->head->prev = temp;

}

if (this->count == 0)

{

this->tail = temp;

this->head = temp;

}

else

{

this->head = temp;

}

this->count++;

}

void List::AddTail(const int value) noexcept

{

Element\* temp = new Element(value);

temp->next = nullptr;

temp->prev = this->tail;

if (this->tail != nullptr)

{

this->tail->next = temp;

}

if (this->count == 0)

{

this->tail = temp;

this->head = temp;

}

else

{

this->tail = temp;

}

this->count++;

}

void List::Insert(const int value, const size\_t position)

{

if (position > this->count)

{

throw out\_of\_range("Bad position!");

}

if (position == 0)

{

this->AddHead(value);

return;

}

if (position == this->count)

{

this->AddTail(value);

return;

}

Element\* prevInsert = this->head;

for (size\_t index = 0; index < position - 1; index++)

{

prevInsert = prevInsert->next;

}

Element\* elementToInsert = new Element(value);

Element\* nextInsert = prevInsert->next;

prevInsert->next = elementToInsert;

elementToInsert->prev = prevInsert;

elementToInsert->next = nextInsert;

this->count++;

}

string List::ToString() const noexcept

{

ostringstream tmpStream;

Element\* temp = this->head;

for (size\_t index = 0; index < this->count; index++)

{

tmpStream << temp->data;

if (index != this->count - 1)

{

tmpStream << ", ";

}

temp = temp->next;

}

return tmpStream.str();

}

void List::Remove(const int value)

{

if (Contains(value) == false)

{

throw out\_of\_range("Out of range!");

}

Element\* tmp = this->head;

int index = -1;

for (size\_t i = 0; i < this->count; i++)

{

if (tmp->data == value)

{

index = i;

}

tmp = tmp->next;

}

static\_cast<size\_t>(index);

Delete(index);

}

void List::Delete(const size\_t position)

{

if (position > this->count - 1)

{

throw out\_of\_range("Выход за пределы массива");

}

Element\* tmp = this->head;

for (size\_t index = 0; index < position; index++)

{

tmp = tmp->next;

}

if (this->count == 1)

{

delete tmp;

tmp = nullptr;

this->head = nullptr;

}

else

{

Element\* prevDelete = tmp->prev;

Element\* nextDelete = tmp->next;

if (prevDelete != nullptr)

{

prevDelete->next = nextDelete;

}

if (nextDelete != nullptr)

{

nextDelete->prev = prevDelete;

}

if (tmp == this->head)

{

this->head = nextDelete;

}

if (tmp == this->tail)

{

this->tail = prevDelete;

}

delete tmp;

tmp = nullptr;

}

this->count--;

}

bool List::Contains(const int value) const noexcept

{

Element\* tmp = this->head;

for (size\_t i = 0; i < this->count; i++)

{

if (tmp->data == value)

{

return true;

}

tmp = tmp->next;

}

return false;

}

List::List(const List& other) noexcept

{

Element\* tmp = other.head;

for (size\_t index = 0; index < other.count; index++)

{

this->AddTail(tmp->data);

tmp = tmp->next;

}

delete tmp;

tmp = nullptr;

}

List::List(List&& other) noexcept

{

Element\* tmp = other.head;

for (size\_t index = 0; index < other.count; index++)

{

this->AddTail(tmp->data);

tmp = tmp->next;

}

delete tmp;

tmp = nullptr;

while (other.count != 0)

{

other.Delete(0);

}

}

List& List::operator = (List&& other) noexcept

{

if (this == &other)

{

return \*this;

}

while (this->count != 0)

{

this->Delete(0);

}

Element\* tmp = other.head;

for (size\_t index = 0; index < other.count; index++)

{

this->AddTail(tmp->data);

tmp = tmp->next;

}

delete tmp;

tmp = nullptr;

while (other.count != 0)

{

other.Delete(0);

}

return \*this;

}

List& List::operator = (const List& other) noexcept

{

if (this == &other)

{

return \*this;

}

Element\* temp = other.head;

while (temp != 0)

{

AddTail(temp->data);

temp = temp->next;

}

return \*this;

}

* 1. Файл ElementTests.cpp

#include "pch.h"

#include "CppUnitTest.h"

#include "..\MyList\Element.cpp"

using namespace Microsoft::VisualStudio::CppUnitTestFramework;

namespace ElementTests

{

TEST\_CLASS(ElementTests)

{

public:

TEST\_METHOD(Ctor\_Success)

{

// Arrange

const int my\_value = 10000;

// Act

Element my\_element(my\_value);

// Assert

Assert::AreEqual(my\_element.data, my\_value);

}

};

}

* 1. Файл MyListTests.cpp

#include "pch.h"

#include "CppUnitTest.h"

#include "..\MyList\MyList.cpp"

#include "..\MyList\Element.cpp"

using namespace Microsoft::VisualStudio::CppUnitTestFramework;

namespace MyListTests

{

TEST\_CLASS(MyListTests)

{

public:

TEST\_METHOD(AddHead\_ValidData\_Success)

{

// Arrange

const int value\_1 = 1;

const int value\_2 = 2;

// Act

List my\_list;

my\_list.AddHead(value\_1);

my\_list.AddHead(value\_2);

int expected = my\_list.head->data;

// Assert

Assert::AreEqual(value\_2, expected);

}

TEST\_METHOD(AddTail\_ValidData\_Success)

{

// Arrange

const int value\_1 = 1;

const int value\_2 = 2;

// Act

List my\_list;

my\_list.AddTail(value\_1);

my\_list.AddTail(value\_2);

int expected = my\_list.tail->data;

// Assert

Assert::AreEqual(value\_2, expected);

}

TEST\_METHOD(Contains\_ValidData\_Success)

{

// Arrange

const int value\_1 = 1;

const int value\_2 = 2;

// Act

List my\_list;

my\_list.AddHead(value\_1);

// Assert

Assert::IsTrue(my\_list.Contains(value\_1));

Assert::IsFalse(my\_list.Contains(value\_2));

}

TEST\_METHOD(ToString\_Success)

{

// Arrange

const int value\_1 = 1;

const int value\_2 = 2;

// Act

List my\_list;

my\_list.AddHead(value\_1);

my\_list.AddHead(value\_2);

string expected = "2, 1";

string real = my\_list.ToString();

// Assert

Assert::AreEqual(expected, real);

}

TEST\_METHOD(Insert\_ValidData\_Success)

{

// Arrange

const int value\_1 = 1;

const int value\_2 = 2;

int valueToInsert = 10;

size\_t indexToInsert = 0;

// Act

List my\_list;

my\_list.AddHead(value\_1);

my\_list.AddHead(value\_2);

my\_list.Insert(valueToInsert, indexToInsert);

int expected = my\_list.head->data;

// Assert

Assert::AreEqual(expected, valueToInsert);

}

TEST\_METHOD(Insert\_InvalidData\_Failure)

{

// Arrange

const int value\_1 = 1;

const int value\_2 = 2;

const int valueToInsert = 10;

const size\_t indexToInsert = 99;

// Act

List my\_list;

my\_list.AddHead(value\_1);

my\_list.AddHead(value\_2);

// Assert

Assert::ExpectException<std::out\_of\_range>([&]() { my\_list.Insert(valueToInsert, indexToInsert); });

}

TEST\_METHOD(Delete\_ValidData\_Success)

{

// Arrange

const int value\_1 = 1;

const int value\_2 = 2;

const size\_t indexToDelete = 0;

// Act

List my\_list;

my\_list.AddHead(value\_1);

my\_list.AddHead(value\_2);

my\_list.Delete(indexToDelete);

// Assert

Assert::AreEqual(value\_1, my\_list.head->data);

}

TEST\_METHOD(Delete\_InvalidData\_Failure)

{

// Arrange

const int value\_1 = 1;

const int value\_2 = 2;

const size\_t indexToDelete = 30;

// Act

List my\_list;

my\_list.AddHead(value\_1);

my\_list.AddHead(value\_2);

// Assert

Assert::ExpectException<std::out\_of\_range>([&]() { my\_list.Delete(indexToDelete); });

}

TEST\_METHOD(CopyCtor\_ValidData\_Success)

{

// Arrange

const int value\_1 = 1;

const int value\_2 = 2;

List my\_list;

my\_list.AddHead(value\_1);

my\_list.AddHead(value\_2);

// Act

List new\_list = my\_list;

// Act & Assert

Assert::AreEqual(my\_list.head->data, new\_list.head->data);

}

TEST\_METHOD(MoveCtor\_ValidData\_Success)

{

// Arrange

const int value\_1 = 1;

const int value\_2 = 2;

List my\_list;

my\_list.AddHead(value\_1);

my\_list.AddHead(value\_2);

int expected = my\_list.head->data;

size\_t expectedSize = 0;

// Act

List new\_list = move(my\_list);

// Assert

Assert::AreEqual(expected, new\_list.head->data);

Assert::AreEqual(my\_list.count, expectedSize);

}

TEST\_METHOD(Remove\_ValidData\_Success)

{

// Arrange

const int value\_1 = 1;

const int value\_2 = 2;

const int valueToDelete = 2;

List my\_list;

my\_list.AddHead(value\_1);

my\_list.AddHead(value\_2);

// Act

my\_list.Remove(valueToDelete);

// Assert

Assert::AreEqual(my\_list.head->data, value\_1);

}

TEST\_METHOD(Remove\_InvalidData\_Failure)

{

// Arrange

const int value\_1 = 1;

const int value\_2 = 2;

const int valueToDelete = 10;

List my\_list;

my\_list.AddHead(value\_1);

my\_list.AddHead(value\_2);

// Act & Assert

Assert::ExpectException<std::out\_of\_range>([&]() { my\_list.Delete(valueToDelete); });

}

};

}

1. UML-диаграмма

На рисунке 1 приведена UML-диаграмма.

|  |
| --- |
|  |
| 1. UML-диаграмма |

1. Результаты работы

На рисунке 2 приведен вывод программы.

|  |
| --- |
|  |
| 1. Результаты работы |

Вывод

В данной практической работе мной были получены навыки по программированию на языках высокого уровня с применением парадигмы ООП и получены навыки по работе со структурами данных. Был реализован класс List, создан API. Все методы и функции проверены и работают корректно, тесты на класс List реализованы и успешно пройдены. Поставленная задача выполнена в полном объёме.