|  |  |
| --- | --- |
| **Российский университет транспорта (МИИТ)**  **Институт транспортной техники и систем управления**  **Кафедра «Управление и защита информации»** | |
| **Отчет**  **по практическому заданию**  **по теме «Двусвязный циклический список»**  **по дисциплине «Системы управления базами данных»** | |
|  | Выполнил:  Студент группы ТКИ-441  Макаров Д.И. |
|  | Проверил:  к.т.н. доцент  Васильева М.А. |
| Москва 2023 | |

Оглавление

[Оглавление 2](#_Toc153910015)

[1 Задание 3](#_Toc153910016)

[1.1 Описание задания 3](#_Toc153910017)

[1.2 Выбранная структура данных 3](#_Toc153910018)

[2 Теоритические сведения 4](#_Toc153910019)

[2.1 Общие теоритические сведения 4](#_Toc153910020)

[2.2 Отличия двухсвязного циклического списка 4](#_Toc153910021)

[3 Описание Программной реализации 5](#_Toc153910022)

[3.1 UML-диаграмма 5](#_Toc153910023)

[3.2 Описание использованных классов и структур 5](#_Toc153910024)

[3.2.1 Структура Element 5](#_Toc153910025)

[3.2.2 Класс List 6](#_Toc153910026)

[3.3 Листинг кода 6](#_Toc153910030)

[3.3.1 List.h 6](#_Toc153910031)

[3.3.2 test.cpp 12](#_Toc153910032)

[3.3.3 Main.cpp 14](#_Toc153910033)

[4 Проверка правильности работы программы 15](#_Toc153910034)

[4.1 Выполнение программы 15](#_Toc153910035)

[4.2 Выполнение тестов 15](#_Toc153910036)

[5 Заключение 17](#_Toc153910037)

[6 Список источников 18](#_Toc153910038)

1. Задание
   1. Описание задания

Разработать структуру данных на языке программирования С++ в ООП парадигме. Реализовать библиотеку классов, обеспечить выполнение операций CRUD (create, read, update, delete) для любого типа данных (template). Переопределить операторы сдвига влево и вправо для возможности чтения и записи из любого потока (консоль, строка, файл). Ни один метод не должен зависеть от консоли или файла, используйте представление в строку. Написать итератор для работы с коллекцией (разработанной структурой данных). По возможности использовать умные указатели (smart pointer), заместо «сырых» (raw). Использование стандартных структур данных возможно в качестве вспомогательных, например, при обходе дерева можно использовать std::vector. Для избежания недоразумений, лучше обсудить использование с преподавателем индивидуально.

Все методы покрыть тестами (модульное тестирование).

* 1. Выбранная структура данных

Двусвязный циклический список.

1. Теоритические сведения
   1. Общие теоритические сведения

**Список** (англ. list) — это абстрактный тип данных, представляющий собой упорядоченный набор значений, в котором некоторое значение может встречаться более одного раза. Экземпляр списка является компьютерной реализацией математического понятия конечной последовательности. Экземпляры значений, находящихся в списке, называются **элементами** списка, если значение встречается несколько раз, каждое вхождение считается отдельным элементом. На рисунке 1 представлен пример представления списка.

|  |
| --- |
| undefined |
| 1. – Односвязный список из трёх элементов |

* 1. Отличия двухсвязного циклического списка

В данной работе выбрана реализация двусвязного циклического списка. В отличие от списка, приведённого ранее, в двусвязном циклическом списке каждый элемент «знает» кто идет за ним, а кто после него. Также эта структура данных «закольцована», т. е. следующий элемент для последнего есть первый элемент списка и, наоборот, для первого предыдущий – последний. На рисунке 2 показан пример такого списка.

|  |
| --- |
|  |
| 1. – Двусвязный циклический список |

1. Описание Программной реализации
   1. UML-диаграмма

На рисунке 3 представлена UML-диаграмма, показывающая связь классов в проекте.

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

* 1. Описание использованных классов и структур
     1. Структура Element

Эта структура является реализацией отдельного элемента списка. Она состоит из 3 полей: поле data, в котором находится информация об элементе и поля previous и next, в которых хранятся ссылки на следующий и предыдущий элементы.

* + 1. Класс List

Этот класс является реализацией двусвязного циклического списка. Он состоит из 3 полей: head и tail, в которых хранятся ссылки на первый и последний элементы этого списка, а также size, в котором хранится размер списка.

* 4. Листинг кода
     1. List.h

#pragma once

#include <iostream>

#include <stdexcept>

/\*

\* @brief Структура для реализации класса List, в которой хранится информации об элементе, а также ссылки на предыдущий и следующий элементы.

\*/

template <typename T>

struct Element {

T data;

Element\* previous;

Element\* next;

Element() : data(), previous(nullptr), next(nullptr) {};

};

/\*

\* @brief Класс List. Представляется из себя двусвязный циклический список.

\*/

template <typename T>

class List {

private:

Element<T>\* head;

Element<T>\* tail;

size\_t size;

public:

/\*

\* @breif Конструктор по умолчанию.

\*/

List();

/\*

\* @brief Метод, позволяющий получить размер списка.

\* @return Размер списка.

\*/

size\_t getSize() const;

/\*

\* @brief Метод, позволяющий добавить элемент в конец списка.

\* @param newData - информация о новом элементе

\*/

void pushBack(T newData);

/\*

\* @brief Метод, позволяющий добавить элемент в начало списка.

\* @param newData - информация о новом элементе.

\*/

void pushFront(T newData);

/\*

\* @brief Метод, позволяющий получить информацию о первом элементе списка.

\* @return Информация о первом элементе.

\*/

T getFront() const;

/\*

\* @brief Метод, позволяющий получить информацию о последнем элементе списка.

\* @return Информация о последнем элементе.

\*/

T getBack() const;

/\*

\* @brief Метод, удаляющий первый элемент списка.

\*/

void popFront();

/\*

\* @brief Метод, удаляющий последний элемент списка.

\*/

void popBack();

/\*

\* @brief Метод, удаляющий элемент списка по ключу.

\* @param position - ключ (позиция элемента в списке).

\*/

void deleteELement(size\_t position);

/\*

\* @brief Метод, позволяющий вставить элемент в список.

\* @param data - информация об элементе (его размер).

\* @param position - позиция, на которую нужно элемент поставить.

\*/

void insertElement(T data, size\_t position);

/\*

\* @brief Метод, позволяющий получить данные об элементе списка.

\* @param position - позиция элемента в списке.

\* @return Информация об элементе - его размер.

\*/

T getElement(size\_t position) const;

/\*

\* @brief Метод, удаляющий все элементы списка.

\*/

void deleteAll();

/\*

\* @brief Конструктор копирования.

\*/

List(const List& other);

/\*

\* @brief Оператор копирования.

\*/

List& operator = (const List& other);

/\*

\* @brief Конструктор перемещения.

\*/

List(List&& other) noexcept;

/\*

\* @brief Оператор перемещения

\*/

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

/\*

\* @brief Деструктор

\*/

~List();

};

template <typename T>

List<T>::List() : head(nullptr), tail(nullptr), size(0) {};

template <typename T>

List<T>::List(const List& other) : List()

{

Element<T>\* temp = other.head;

for(size\_t i = 0; i < other.getSize(); i++)

{

this->pushBack(temp->data);

temp = temp->next;

}

this->size = other.size;

}

template <typename T>

List<T>& List<T>::operator = (const List& other) {

if (this == &other)

return \*this;

List<T> copy(other);

std::swap(this->tail, copy.tail);

std::swap(this->head, copy.head);

std::swap(this->size, copy.size);

return \*this;

}

template <typename T>

size\_t List<T>::getSize() const{

return this->size;

}

template <typename T>

void List<T>::pushBack(T newData) {

Element<T>\* temp = new Element<T>();

temp->data = newData;

if (this->getSize() == 0) {

temp->previous = temp;

temp->next = temp;

this->head = temp;

this->tail = temp;

}

else if (this->getSize() == 1) {

temp->previous = this->head;

temp->next = this->head;

this->tail = temp;

this->head->next = temp;

this->head->previous = temp;

} else {

temp->previous = this->tail;

temp->next = this->head;

tail->next = temp;

this->tail = temp;

this->head->previous = this->tail;

}

++this->size;

}

template <typename T>

void List<T>::pushFront(T newData) {

Element<T>\* temp = new Element<T>();

temp->data = newData;

if (this->getSize() == 0) {

temp->next = temp;

temp->previous = temp;

this->head = temp;

this->tail = temp;

}

else if (this->getSize() == 1) {

temp->previous = this->tail;

temp->next = this->tail;

this->head = temp;

this->tail->next = temp;

this->tail->previous = temp;

} else {

temp->next = this->head;

temp->previous = this->tail;

head->previous = temp;

this->head = temp;

this->tail->next = this->head;

}

++this->size;

}

template <typename T>

T List<T>::getFront() const {

return this->head->data;

}

template <typename T>

T List<T>::getBack() const {

return this->tail->data;

}

template <typename T>

void List<T>::popBack() {

if (this->getSize() == 0)

throw std::logic\_error("Нельзя удалять элементы в пустом листе");

if (this->getSize() == 1) {

delete this->tail;

this->head = nullptr;

this->tail = nullptr;

--this->size;

return;

}

else if (this->getSize() == 2) {

delete this->tail;

this->head->next = this->tail;

this->tail = this->head;

--this->size;

return;

}

auto temp = this->tail;

this->tail = this->tail->previous;

this->tail->next = this->head;

this->head->previous = this->tail;

delete temp;

--this->size;

}

template <typename T>

void List<T>::popFront() {

if (this->getSize() == 0)

throw std::logic\_error("Нельзя удалять элементы в пустом листе");

if (this->getSize() == 1) {

delete this->head;

this->head = nullptr;

this->tail = nullptr;

--this->size;

return;

}

else if (this->getSize() == 2) {

delete this->head;

this->tail->previous = nullptr;

this->head = this->tail;

--this->size;

return;

}

auto temp = this->head;

this->head = this->head->next;

this->head->previous = this->tail;

this->tail->next = this->head;

delete temp;

--this->size;

}

template <typename T>

void List<T>::deleteELement(size\_t position) {

if (position >= this->getSize())

throw std::out\_of\_range("Позиция больше размера списка");

if (position == 0) {

this->popFront();

return;

}

else if (position == this->getSize() - 1) {

this->popBack();

return;

}

auto temp = this->head;

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

{

temp = temp->next;

}

temp->previous->next = temp->next;

temp->next->previous = temp->previous;

delete temp;

--this->size;

}

template <typename T>

void List<T>::insertElement(T data, size\_t position) {

if (position > this->getSize())

throw std::out\_of\_range("Позиция больше размера списка");

if (position == 0) {

this->pushFront(data);

return;

}

else if (position == this->getSize()) {

this->pushBack(data);

return;

}

auto insert = this->head;

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

{

insert = insert->next;

}

Element<T>\* temp = new Element<T>();

temp->data = data;

if (insert->previous != nullptr && this->getSize() != 1)

insert->previous->next = temp;

temp->next = insert;

temp->previous = insert->previous;

insert->previous = temp;

++this->size;

}

template <typename T>

T List<T>::getElement(size\_t position) const{

if (position >= this->getSize())

throw std::out\_of\_range("Позиция больше размера списка");

auto get = this->head;

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

{

get = get->next;

}

return get->data;

}

template <typename T>

void List<T>::deleteAll() {

while (this->getSize() != 0)

this->popBack();

}

template <typename T>

List<T>::~List() {

this->deleteAll();

}

template <typename T>

List<T>::List(List&& other) noexcept : List() {

\*this = other;

}

template <typename T>

List<T>& List<T>::operator = (List&& other) noexcept {

std::swap(this->tail, other.tail);

std::swap(this->head, other.head);

std::swap(this->size, other.size);

return \*this;

}

template <typename T>

std::ostream& operator << (std::ostream& os, const List<T>& list) {

for (size\_t i = 0; i < list.getSize(); i++)

{

os << list.getElement(i) << " ";

}

return os;

}

* + 1. test.cpp

#include "pch.h"

#include "../List/List.h"

#include <string>

using namespace std;

class ListPushTest : public testing::Test {

protected:

List<int> l1;

List<string> l2;

};

TEST\_F(ListPushTest, PushBackStringWorks) {

l2.pushBack("rty");

l2.pushBack("qwe");

EXPECT\_EQ(l2.getSize(), 2);

EXPECT\_EQ(l2.getBack(), "qwe");

}

TEST\_F(ListPushTest, PushFrontStringWorks) {

l2.pushFront("rty");

l2.pushFront("qwe");

EXPECT\_EQ(l2.getSize(), 2);

EXPECT\_EQ(l2.getFront(), "qwe");

}

TEST\_F(ListPushTest, PushBackIntWorks) {

l1.pushBack(5);

l1.pushBack(1);

EXPECT\_EQ(l1.getSize(), 2);

EXPECT\_EQ(l1.getBack(), 1);

}

TEST\_F(ListPushTest, PushFrontIntWorks) {

l1.pushFront(3);

l1.pushFront(10);

EXPECT\_EQ(l1.getSize(), 2);

EXPECT\_EQ(l1.getFront(), 10);

}

class ListTest : public testing::Test {

protected:

List<int> l1;

List<string> l2;

void SetUp() override {

l1.pushBack(1);

l1.pushBack(5);

l2.pushFront("qwe");

}

};

TEST\_F(ListTest, GetElementIntWorks) {

l1.pushBack(10);

l1.pushBack(15);

EXPECT\_EQ(l1.getElement(2), 10);

EXPECT\_EQ(l1.getElement(3), 15);

EXPECT\_EQ(l1.getElement(0), 1);

}

TEST\_F(ListTest, GetElementStringWorks) {

l2.pushBack("---");

l2.pushBack("rty");

EXPECT\_EQ(l2.getElement(1), "---");

}

TEST\_F(ListTest, CopyConstructorIntWorks) {

List<int> l3 = List<int>(l1);

for (size\_t i = 0; i < l1.getSize(); i++)

{

EXPECT\_EQ(l3.getElement(i), l1.getElement(i));

}

}

TEST\_F(ListTest, CopyConstructorStringWorks) {

List <string> l3 = List<string>(l2);

for (size\_t i = 0; i < l2.getSize(); i++)

{

EXPECT\_TRUE(l2.getElement(i).compare(l3.getElement(i)) == 0);

}

}

TEST\_F(ListTest, PopFrontThrowsLogicError) {

List <int> l3;

EXPECT\_THROW(l3.popFront(), logic\_error);

}

TEST\_F(ListTest, PopFrontWorks ) {

l1.popFront();

EXPECT\_EQ(l1.getSize(), 1);

l2.popFront();

EXPECT\_EQ(l2.getSize(), 0);

}

TEST\_F(ListTest, PopBackThrowsLogicError) {

List <int> l3;

EXPECT\_THROW(l3.popBack(), logic\_error);

}

TEST\_F(ListTest, PopBackWorks) {

l1.popBack();

EXPECT\_EQ(l1.getSize(), 1);

l2.popBack();

EXPECT\_EQ(l2.getSize(), 0);

}

TEST\_F(ListTest, DeleteElementThrowsOutOfRange) {

EXPECT\_THROW(l1.deleteELement(4), out\_of\_range);

}

TEST\_F(ListTest, DeleteElementWorks) {

l1.pushFront(10);

l1.deleteELement(1);

std::vector<int> v3 = { 10, 5 };

for (size\_t i = 0; i < v3.size(); i++)

{

EXPECT\_EQ(l1.getElement(i), v3[i]);

}

}

TEST\_F(ListTest, InsertElementThrowsOutOfRange) {

EXPECT\_THROW(l1.insertElement(14, 4), out\_of\_range);

}

TEST\_F(ListTest, InserElementIntWorks) {

l1.insertElement(3, 1);

std::vector<int> v3 = { 1, 3, 5 };

for (size\_t i = 0; i < v3.size(); i++)

{

EXPECT\_EQ(l1.getElement(i), v3[i]);

}

}

TEST\_F(ListTest, InsertElementStringWorks) {

l2.pushBack("rty");

l2.insertElement("---", 1);

std::vector<string> v3 = { "qwe", "---", "rty" };

for (size\_t i = 0; i < v3.size(); i++)

{

EXPECT\_TRUE(l2.getElement(i).compare(v3[i]) == 0);

}

}

TEST\_F(ListTest, GetElementThrowsOutOfRange) {

EXPECT\_THROW(l1.deleteELement(4), out\_of\_range);

}

TEST\_F(ListTest, DeleteAllWorks) {

EXPECT\_NO\_FATAL\_FAILURE(l1.deleteAll());

EXPECT\_EQ(l1.getSize(), 0);

}

* + 1. Main.cpp

#include "List.h"

#include <iostream>

using namespace std;

int main() {

List<int> l;

l.pushBack(7);

l.pushBack(2);

l.pushBack(9);

l.pushFront(0);

l.pushFront(-4);

cout << l;

return 0;

}

1. Проверка правильности работы программы
   1. Выполнение программы

Для примера работы программы создадим новый лист List<int> l, добавим в него несколько элементов и выведем его на экран. Добавим в его конец 3 элемента и в начало – 2. Создание листа, добавление элементов и результат работы программы представлены на рисунке 4.

|  |
| --- |
|  |
| 1. – Пример работы программы |

* 1. Выполнение тестов

Результаты работы тестов показаны на рисунке 5.

|  |
| --- |
|  |
| 1. – Результаты работы тестов |

1. Заключение

В ходе работы была изучена структура данных – двусвязный циклический список. Также была изучена теория о тестировании работы программы с помощью Google тестов. Итогом работы стала программная реализации выбранной структуры данных.

1. Список источников

Список (информатика) – <https://ru.wikipedia.org/wiki/%D0%A1%D0%BF%D0%B8%D1%81%D0%BE%D0%BA_(%D0%B8%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0)>

Двусвязный циклический список. Пример реализации – <https://prog-cpp.ru/data-dcs/>

Построение UML-диаграмм – <https://creately.com/ru/home/>