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

Оглавление

[Оглавление 2](#_Toc153731450)

[1 Задание 3](#_Toc153731451)

[2 Описание структуры – Односвязного списка 4](#_Toc153731452)

[2.1 UML-диаграмма 4](#_Toc153731453)

[2.3 Листинг кода 4](#_Toc153731457)

[2.3.1 List.h 4](#_Toc153731458)

[2.3.1 Node.h 8](#_Toc153731459)

[2.3.2 Main.cpp 8](#_Toc153731460)

[2.3.1 test.cpp 9](#_Toc153731461)

[3.1Результат работы программы 11](#_Toc153731462)

[3.2Тестирование программы 11](#_Toc153731463)

[4. Заключение 12](#_Toc153731464)

1. Задание

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).

Выбранная структура данных: Односвязный список

1. Описание структуры – Односвязного списка
   1. UML-диаграмма

На рисунке 2 представлена UML-диаграмма, показывающая связь классов в проекте.

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

* 4. Листинг кода
     1. List.h

#include "Node.h"

#include <iostream>

template<typename T>

class List final

{

private:

size\_t size;

Node<T>\* head;

public:

List();

List(const List<T>& other);

~List();

List(List<T>&& other) noexcept;

List<T>& operator=(List<T>&& other) noexcept;

List<T>& operator=(const List<T>& other);

/\*

\* @brief удаление первого элемента в списке

\*/

void pop\_front();

/\*

\* @brief получение первого элемента в списке

\* @return информация об элементе

\*/

T top() const;

/\*

\* @brief добавление элемента в конец списка

\* @param data информация об элементе

\*/

void push\_back(T data);

/\*

\* @brief очистить список

\*/

void clear();

/\*

\* @brief получить количество элементов в списке

\* @return количество элементов в списке

\*/

size\_t GetSize() const { return size; }

/\*

\* @brief перегруженный оператор []

\*/

const T operator[](const size\_t index) const;

/\*

\* @brief перегруженный оператор []

\*/

T operator[](const size\_t index);

/\*

\* @brief добавление элемента в начало списка

\* @param data информация об элементе

\*/

void push\_front(T data);

/\*

\* @brief добавление элемента в список по указанному индексу

\* @param data информация об элементе

\* @param index индекс элемента

\*/

void insert(T data, size\_t index);

/\*

\* @brief удаление элемента в списке по указанному индексу

\* @param index индекс элемента

\*/

void removeAt(size\_t index);

/\*

\* @brief удаление последнего элемента в списке

\*/

void pop\_back();

};

template<typename T>

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

template<typename T>

List<T>::~List()

{

clear();

}

template<typename T>

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

{

if (this == &other)

return \*this;

List<T> copy(other);

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

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

return \*this;

}

template<typename T>

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

{

auto temp = other.head;

while (temp != nullptr)

{

this->push\_back(temp->data);

temp = temp->pNext;

}

this->size = other.size;

}

template <typename T>

List<T>::List(List&& other) noexcept : head(other->head), size(other->size) {

other.count = 0;

other.head = nullptr;

}

template <typename T>

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

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

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

return \*this;

}

template<typename T>

void List<T>::pop\_front()

{

auto\* temp = head;

head = head->pNext;

delete temp;

size--;

}

template<typename T>

void List<T>::push\_back(T data)

{

if (head == nullptr)

{

head = new Node<T>(data);

}

else

{

auto\* current = this->head;

while (current->pNext != nullptr)

{

current = current->pNext;

}

current->pNext = new Node<T>(data);

}

size++;

}

template<typename T>

void List<T>::clear()

{

while (size > 0)

{

pop\_front();

}

}

template<typename T>

T List<T>::operator[](const size\_t index)

{

size\_t counter = 0;

auto current = this->head;

while (current != nullptr)

{

if (counter == index)

{

return current->data;

}

current = current->pNext;

counter++;

}

return current->data;

}

template<typename T>

const T List<T>::operator[](const size\_t index) const

{

size\_t counter = 0;

auto current = this->head;

while (current != nullptr)

{

if (counter == index)

{

return current->data;

}

current = current->pNext;

counter++;

}

return current->data;

}

template<typename T>

void List<T>::push\_front(T data)

{

auto temp = new Node<T>(data, head);

this->head = temp;

size++;

}

template<typename T>

void List<T>::insert(T data, size\_t index)

{

if (index == 0)

{

push\_front(data);

}

else

{

auto\* previous = this->head;

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

{

previous = previous->pNext;

}

auto\* newNode = new Node<T>(data, previous->pNext);

previous->pNext = newNode;

size++;

}

}

template<typename T>

void List<T>::removeAt(size\_t index)

{

if (index == 0)

{

pop\_front();

}

else

{

auto\* previous = this->head;

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

{

previous = previous->pNext;

}

auto\* toDelete = previous->pNext;

previous->pNext = toDelete->pNext;

delete toDelete;

size--;

}

}

template<typename T>

void List<T>::pop\_back()

{

removeAt(size - 1);

}

template <typename T>

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

return this->head->data;

* + 1. Node.h

template<typename T>

struct Node

{

Node\* pNext;

T data;

Node<T>(T data, Node\* pNext = nullptr) : pNext(pNext), data(data)

{}

* + 1. Main.cpp

#include "List.h"

int main()

{

setlocale(LC\_ALL, "ru");

List<int> lst;

lst.push\_front(5);

lst.push\_front(7);

lst.push\_front(101);

for (int i = 0; i < lst.GetSize(); i++)

{

std::cout << lst[i] << std::endl;

}

std::cout << std::endl << "pop\_back " << std::endl << std::endl;

lst.pop\_back();

for (int i = 0; i < lst.GetSize(); i++)

{

std::cout << lst[i] << std::endl;

}

std::cout << std::endl << "= copy" << std::endl << std::endl;

List<int> lst2;

lst2 = lst;

for (int i = 0; i < lst2.GetSize(); i++)

{

std::cout << lst2[i] << std::endl;

}

std::cout << std::endl << "copy constructor" << std::endl << std::endl;

List<int> lst3 = lst;

for (int i = 0; i < lst3.GetSize(); i++)

{

std::cout << lst3[i] << std::endl;

}

List<int> lst4;

List<int> lst5 = lst4;

return 0;

* + 1. test.cpp

#include "pch.h"

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

#include <string>

TEST(ListPushBack, ListPushBackWorkTest) {

List<int> lst1;

List<std::string> lst2;

lst1.push\_back(101);

lst2.push\_back("qwe");

EXPECT\_EQ(lst1.GetSize(), 1);

EXPECT\_EQ(lst1.top(), 101);

EXPECT\_EQ(lst2.GetSize(), 1);

EXPECT\_EQ(lst2.top(), "qwe");

}

/\*

\* @brief класс для тестирования

\*/

class ListTest: public testing::Test {

protected:

List<int> lst1;

List<std::string> lst2;

void SetUp() override {

lst1.push\_back(0);

lst1.push\_back(1);

lst1.push\_back(2);

lst2.push\_back("qwe");

lst2.push\_back("rty");

}

};

TEST\_F(ListTest, PopFrontWorks) {

lst1.pop\_front();

EXPECT\_EQ(lst1.GetSize(), 2);

EXPECT\_EQ(lst1.top(), 1);

}

TEST\_F(ListTest, OperatorWorksInt) {

EXPECT\_EQ(lst1.top(), lst1[0]);

lst1.push\_back(3);

EXPECT\_EQ(3, lst1[3]);

}

TEST\_F(ListTest, OperatorWorksString) {

EXPECT\_EQ(lst2.top(), lst2[0]);

lst2.push\_back("iop");

EXPECT\_EQ("iop", lst2[2]);

}

TEST\_F(ListTest, PopBackWorks) {

lst1.pop\_back();

EXPECT\_EQ(lst1.GetSize(), 2);

}

TEST\_F(ListTest, PushFrontWorksInt) {

lst1.push\_front(-1);

EXPECT\_EQ(lst1.GetSize(), 4);

EXPECT\_EQ(lst1.top(), -1);

}

TEST\_F(ListTest, PushFrontWorksString) {

lst2.push\_front("iop");

EXPECT\_EQ(lst2.GetSize(), 3);

EXPECT\_EQ(lst2.top(), "iop");

}

TEST\_F(ListTest, ClearWorks) {

lst1.clear();

EXPECT\_EQ(lst1.GetSize(), 0);

}

TEST\_F(ListTest, InsertWorksInt) {

lst1.insert(5, 1);

EXPECT\_EQ(lst1.GetSize(), 4);

EXPECT\_EQ(lst1[1], 5);

}

TEST\_F(ListTest, InsertWorksString) {

lst2.insert("---", 1);

EXPECT\_EQ(lst2.GetSize(), 3);

EXPECT\_EQ(lst2[1], "---");

}

TEST\_F(ListTest, RemoveAtWorks) {

lst1.removeAt(1);

EXPECT\_EQ(lst1.GetSize(), 2);

EXPECT\_EQ(lst1[1], 2);

}

TEST\_F(ListTest, CopyConstructorAndOperatorWork) {

List<int> lst3;

EXPECT\_NO\_FATAL\_FAILURE(lst3 = List<int>(lst1));

EXPECT\_EQ(lst3.top(), 0);

lst3.clear();

EXPECT\_EQ(lst1.GetSize(), 3);

}

TEST\_F(ListTest, MoveConstructorAndOperatorWork) {

List<int> lst3;

lst3 = std::move(lst1);

EXPECT\_EQ(lst3.top(), 0);

lst3.clear();

EXPECT\_EQ(lst1.GetSize(), 0);

**3.Работа программы**

3.1Результат работы программы

|  |
| --- |
|  |
|  |
| 1. – Результаты работы программы |

3.2Тестирование программы

Чтобы оценить, что программа работает верно, её нужно протестировать. Результаты работы тестов показаны на рисунке 6.

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

4. Заключение

В ходе задания была изучена теория, связанная с односвязными списками, были освоены методы работы с ними.

Итогом выполнения задания стала реализация структуры односвязный список на языке программирования C++.