|  |  |
| --- | --- |
| **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  Куминов В. П.  Проверила:  Доцент кафедры УиЗИ, к.т.н.  Васильева М. А. |
| Москва 2022 | |

**ОГЛАВЛЕНИЕ**

[1. Цель работы 3](#_Toc120041458)

[2. Задание 3](#_Toc120041459)

[3. Код программы 3](#_Toc120041460)

[3.1. Файл DEC.h 3](#_Toc120041461)

[3.2. Файл NODE.h 5](#_Toc120041462)

[3.3. Файл DEC.cpp 5](#_Toc120041463)

[3.4. Файл DEC\_Tests.cpp 8](#_Toc120041464)

[4. UML-диаграмма 13](#_Toc120041465)

[5. Окно вывода 13](#_Toc120041466)

[6. Вывод 14](#_Toc120041467)

1. Цель работы

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

1. Задание
2. Для структуры «DECK» разработать API;
3. На языке «C++» написать конструктор и деструктор класса разрабатываемой структуры.
4. Разработать метод вывода структуры в строку;
5. Реализовать CRUD;
6. Разработать тесты.
7. Код программы
   1. Файл DEC.h

#pragma once

#include "NODE.h"

#include <iostream>

class Deck

{

Node\* head,\* bottom;

size\_t size;

public:

/\*

\brief Конструктор по умолчанию для класса Deck.

\*/

Deck() :head(nullptr), bottom(nullptr), size(0) {};

/\*

\brief Деструктор для класса Deck.

\*/

~Deck() noexcept;

/\*

\brief Конструктор копирования.

\param const Deck& - указатель на Deck

\*/

Deck(const Deck&) noexcept;

/\*

\brief Конструктор перемещения.

\param Deck&& - указатель на Deck

\*/

Deck(const Deck&&) noexcept;

/\*

\brief Оператор присваивания копирования.

\param Deck& - указатель на Deck

\*/

Deck& operator=(const Deck&) noexcept;

/\*

\brief Оператор присваивания перемещения.

\param Deckk&& - указатель на Deck

\*/

Deck& operator=(const Deck&&) noexcept;

/\*

\brief Метод вноса нового элемента в ДЕК справа.

\param value - значение, которое будет записано в элементе дека.

\*/

void PushRight(const int value) noexcept;

/\*

\brief Метод вноса нового элемента в ДЕК слева.

\param value - значение, которое будет записано в элементе дека.

\*/

void PushLeft(const int value) noexcept;

/\*

\brief Метод удаления из ДЕКа элемента справа.

\*/

int DeleteRight();

/\*

\brief Метод удаления из ДЕКа элемента слева.

\*/

int DeleteLeft();

/\*!

\brief Метод возврата количества элементов в ДЕКе.

\*/

const size\_t GetSize() noexcept;

/\*

\brief Метод изменения элемента ДЕКа справа.

\param value - значение, которое будет записано в элементе дека.

\*/

void ChangeRight(const int value);

/\*

\brief Метод изменения элемента ДЕКа слева.

\param value - значение, которое будет записано в элементе дека.

\*/

void ChangeLeft(const int value);

/\*

\brief Метод будет выводить на экран элемент дека справа.

\*/

int PrintRight() noexcept;

/\*!

\brief Метод будет выводить на экран элемент дека слева.

\*/

int PrintLeft() noexcept;

void Clean() noexcept;

friend std::ostream& operator<< (std::ostream& out, Deck& deck);

};

* 1. Файл NODE.h

#pragma once

#include "DEC.h"

class Node

{

int \_value;

Node\* next;

public:

/\*

\brief Метод является конструктором с параметрами для класса Node.

\param value - значение, которое будет записано в элемент дека.

\*/

Node(const int value) { \_value = value; next = nullptr; };

friend class Deck;

};

* 1. Файл DEC.cpp

#include "DEC.h"

#include "NODE.h"

#include <sstream>

#include <string>

#include <iostream>

using namespace std;

const size\_t Deck::GetSize() noexcept

{

return size;

};

Deck::~Deck() noexcept

{

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

this->DeleteRight();

};

size = 0;

};

Deck::Deck(const Deck& other) noexcept

{

if (this != &other)

{

Node\* temp = other.head;

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

{

this->PushRight(temp->\_value);

temp = temp->next;

}

}

};

Deck::Deck(const Deck&& other) noexcept

{

if (this != &other)

{

Node\* temp = other.head;

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

{

this->PushRight(temp->\_value);

temp = temp->next;

}

other.~Deck();

}

};

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

{

if (this != &other)

{

Node\* temp = other.head;

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

{

this->PushRight(temp->\_value);

temp = temp->next;

}

}

return \*this;

};

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

{

if (this != &other)

{

Node\* temp = other.head;

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

{

this->PushRight(temp->\_value);

temp = temp->next;

}

other.~Deck();

}

return \*this;

};

void Deck::PushRight(const int value) noexcept

{

Node\* temp = new Node(value);

if (bottom == nullptr)

{

bottom = head = temp;

}

else

{

bottom->next = temp;

bottom = temp;

};

size++;

};

void Deck::PushLeft(const int value) noexcept

{

Node\* temp = new Node(value);

if (head == nullptr)

{

head = bottom = temp;

}

else

{

temp->next = head;

head = temp;

};

size++;

};

int Deck::DeleteRight()

{

if (size == 0) throw out\_of\_range("Пусто");

const int value = bottom->\_value;

if (size == 1) {

delete head;

head = nullptr;

bottom = nullptr;

}

else {

Node\* tempNode = head;

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

{

tempNode = tempNode->next;

}

tempNode->next = nullptr;

delete bottom;

bottom = tempNode;

};

size--;

return value;

};

int Deck::DeleteLeft()

{

if (size == 0) throw out\_of\_range("Пусто");

const int value = head->\_value;

if

(size == 1) {

delete head;

head = nullptr;

bottom = nullptr;

}

else {

Node\* tempHead = head->next;

delete head;

head = tempHead;

}

size--;

return value;

};

void Deck::ChangeLeft(const int value)

{

if (size == 0) throw out\_of\_range("Пусто");

this->DeleteLeft();

this->PushLeft(value);

};

void Deck::ChangeRight(const int value)

{

if (size == 0) throw out\_of\_range("Пусто");

this->DeleteRight();

this->PushRight(value);

};

int Deck::PrintLeft() noexcept

{

return head->\_value;

};

int Deck::PrintRight() noexcept

{

return bottom->\_value;

};

void Deck::Clean() noexcept

{

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

{

this->DeleteRight();

};

size = 0;

}

std::ostream& operator<< (std::ostream& out, Deck& deck)

{

const size\_t size = deck.GetSize();

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

out << deck.DeleteLeft() << " ";

};

return out;

};

* 1. Файл DEC\_Tests.cpp

#include "pch.h"

#include "CppUnitTest.h"

#include "..\DEC\DEC.cpp"

#include "..\DEC\NODE.h"

#include <random>

using namespace Microsoft::VisualStudio::CppUnitTestFramework;

namespace DeckTests

{

TEST\_CLASS(DeckTests)

{

public:

TEST\_METHOD(Default\_valid\_data\_constructor\_successed)

{

Deck a;

Assert::IsTrue(a.GetSize() == 0);

}

TEST\_METHOD(Default\_valid\_data\_destructor\_successed)

{

Deck a;

size\_t size = 9;

int LowLimit = -100;

int HightLimit = 100;

random\_device rd;

mt19937 gen(rd());

uniform\_int\_distribution<> distrib(LowLimit, HightLimit);

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

{

int k = distrib(gen);

a.PushLeft(k);

}

a.~Deck();

Assert::IsTrue(a.GetSize() == 0);

}

TEST\_METHOD(Default\_valid\_data\_copy\_constructor\_successed)

{

Deck a;

size\_t size = 9;

int LowLimit = -100;

int HightLimit = 100;

random\_device rd;

mt19937 gen(rd());

uniform\_int\_distribution<> distrib(LowLimit, HightLimit);

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

{

int k = distrib(gen);

a.PushLeft(k);

}

Deck b = a;

int c = a.PrintLeft();

int d = b.PrintLeft();

Assert::IsTrue(c = d);

}

TEST\_METHOD(Default\_valid\_data\_moving\_constructor\_successed)

{

Deck a;

size\_t size = 9;

int LowLimit = -100;

int HightLimit = 100;

random\_device rd;

mt19937 gen(rd());

uniform\_int\_distribution<> distrib(LowLimit, HightLimit);

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

{

int k = distrib(gen);

a.PushLeft(k);

}

int c = a.PrintLeft();

Deck b = std::move(a);

int d = b.PrintLeft();

Assert::IsTrue(a.GetSize() == 0);

Assert::IsTrue(c = d);

}

TEST\_METHOD(PushLeft\_valid\_data\_successed)

{

Deck a;

size\_t size = 9;

int LowLimit = -100;

int HightLimit = 100;

random\_device rd;

mt19937 gen(rd());

uniform\_int\_distribution<> distrib(LowLimit, HightLimit);

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

{

int k = distrib(gen);

a.PushLeft(k);

Assert::IsTrue(a.PrintLeft() == k);

}

}

TEST\_METHOD(PushRight\_valid\_data\_successed)

{

Deck a;

size\_t size = 9;

int LowLimit = -100;

int HightLimit = 100;

random\_device rd;

mt19937 gen(rd());

uniform\_int\_distribution<> distrib(LowLimit, HightLimit);

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

{

int k = distrib(gen);

a.PushRight(k);

Assert::IsTrue(a.PrintRight() == k);

}

}

TEST\_METHOD(DeleteLeft\_invalid\_data\_successed)

{

Deck a;

Assert::ExpectException<std::out\_of\_range>([&]() {a.DeleteLeft(); });

}

TEST\_METHOD(DeleteLeft\_valid\_data\_successed)

{

Deck a;

int LowLimit = -100;

int HightLimit = 100;

random\_device rd;

mt19937 gen(rd());

uniform\_int\_distribution<> distrib(LowLimit, HightLimit);

int k = distrib(gen);

a.PushLeft(k);

Assert::IsTrue(k == a.DeleteLeft());

}

TEST\_METHOD(DeleteRight\_invalid\_data\_successed)

{

Deck a;

Assert::ExpectException<std::out\_of\_range>([&]() {a.DeleteRight(); });

}

TEST\_METHOD(DeleteRight\_valid\_data\_successed)

{

Deck a;

int LowLimit = -100;

int HightLimit = 100;

random\_device rd;

mt19937 gen(rd());

uniform\_int\_distribution<> distrib(LowLimit, HightLimit);

int k = distrib(gen);

a.PushRight(k);

Assert::IsTrue(k == a.DeleteRight());

}

TEST\_METHOD(GetSize\_valid\_data\_successed)

{

Deck a;

int LowLimit = -100;

int HightLimit = 100;

random\_device rd;

mt19937 gen(rd());

uniform\_int\_distribution<> distrib(LowLimit, HightLimit);

int k = distrib(gen);

a.PushRight(k);

Assert::IsTrue(a.GetSize() == 1);

}

TEST\_METHOD(PrintLeft\_valid\_data\_successed)

{

Deck a;

int LowLimit = -100;

int HightLimit = 100;

random\_device rd;

mt19937 gen(rd());

uniform\_int\_distribution<> distrib(LowLimit, HightLimit);

int k = distrib(gen);

a.PushLeft(k);

Assert::IsTrue(k == a.PrintLeft());

}

TEST\_METHOD(PrintRight\_valid\_data\_successed)

{

Deck a;

int LowLimit = -100;

int HightLimit = 100;

random\_device rd;

mt19937 gen(rd());

uniform\_int\_distribution<> distrib(LowLimit, HightLimit);

int k = distrib(gen);

a.PushRight(k);

Assert::IsTrue(k == a.PrintRight());

}

TEST\_METHOD(ChangeLeft\_invalid\_data\_successed)

{

Deck a;

int k = 1;

Assert::ExpectException<std::out\_of\_range>([&]() {a.ChangeLeft(k); });

}

TEST\_METHOD(ChangeLeft\_valid\_data\_successed)

{

Deck a;

size\_t size = 9;

int LowLimit = -100;

int HightLimit = 100;

random\_device rd;

mt19937 gen(rd());

uniform\_int\_distribution<> distrib(LowLimit, HightLimit);

int k = distrib(gen);

a.PushLeft(k);

k = distrib(gen);

a.ChangeLeft(k);

Assert::IsTrue(k == a.PrintLeft());

}

TEST\_METHOD(ChangeRight\_valid\_data\_successed)

{

Deck a;

int LowLimit = -100;

int HightLimit = 100;

random\_device rd;

mt19937 gen(rd());

uniform\_int\_distribution<> distrib(LowLimit, HightLimit);

int k = distrib(gen);

a.PushRight(k);

k = distrib(gen);

a.ChangeRight(k);

Assert::IsTrue(k == a.PrintRight());

}

TEST\_METHOD(Operator\_successed)

{

Deck a;

int LowLimit = -100;

int HightLimit = 100;

random\_device rd;

mt19937 gen(rd());

uniform\_int\_distribution<> distrib(LowLimit, HightLimit);

int k = distrib(gen);

a.PushLeft(k);

std::stringstream temp;

temp << k << " ";

std::string tempStr = temp.str();

std::stringstream out;

out << a;

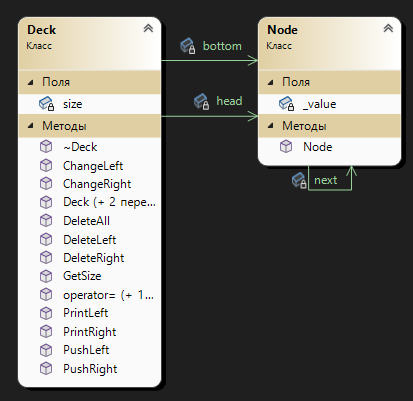
Assert::IsTrue(tempStr == out.str());

}

};

};

1. UML-диаграмма



1. Окно вывода









1. Вывод

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