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

Оглавление

[1. Условие задачи 3](#_Toc154609555)

[2. Текст программы на языке С++ 3](#_Toc154609556)

[Node.h 3](#_Toc154609560)

[Class.h 3](#_Toc154609561)

[Main.cpp 7](#_Toc154609562)

[Test.cpp 7](#_Toc154609563)

[3. Результат работы программы 10](#_Toc154609564)

[4. UML диаграмма классов 10](#_Toc154609565)

1. Условие задачи

**Стек**

Определить класс «Node», включающий следующую информацию:

* Данные любого типа;

В программе предусмотреть:

* создание БДП;
* вывод на экран;
* удаление элемента;
* вставка элемента;
* редактирование элемента;
* Ввод из файла и вывод из файла.

1. Текст программы на языке С++

Node.h

#pragma once

#include "Class.h"

template <typename T>

struct Node {

T data;

Node<T>\* next;

Node(const T& value) : data(value), next(nullptr) {}

Class.h

#pragma once

#include <iostream>

#include <fstream>

#include <sstream>

#include <stdexcept>

#include "Node.h"

template <typename T>

class Stack {

private:

Node<T>\* top;

public:

Stack() : top(nullptr) {}

~Stack() {

while (!isEmpty()) {

pop();

}

}

bool isEmpty() const {

return top == nullptr;

}

void push(const T& value) {

Node<T>\* newNode = new Node<T>(value);

newNode->next = top;

top = newNode;

}

void pop() {

if (isEmpty()) {

throw std::out\_of\_range("Stack is empty");

}

Node<T>\* temp = top;

top = top->next;

delete temp;

}

T& peek() const {

if (isEmpty()) {

throw std::out\_of\_range("Stack is empty");

}

return top->data;

}

friend std::ostream& operator<<(std::ostream& os, const Stack& stack) {

Node<T>\* current = stack.top;

while (current != nullptr) {

os << current->data << " ";

current = current->next;

}

return os;

}

friend std::istream& operator>>(std::istream& is, Stack& stack) {

T value;

is >> value;

stack.push(value);

return is;

}

friend std::ofstream& operator<<(std::ofstream& ofs, const Stack& stack) {

Node<T>\* current = stack.top;

while (current != nullptr) {

ofs << current->data << " ";

current = current->next;

}

return ofs;

}

friend std::ifstream& operator>>(std::ifstream& ifs, Stack& stack) {

T value;

while (ifs >> value) {

stack.push(value);

}

return ifs;

}

static std::string editItem(Stack<T>& stack, std::ostringstream& out, T newValue, std::ostringstream& err) {

try {

stack.peek() = newValue;

out << "Item edited successfully." << std::endl;

return out.str();

}

catch (const std::out\_of\_range& e) {

err << "Error: " << e.what() << std::endl;

return err.str();

}

}

static std::string displayItems(const Stack<T>& stack, std::ostringstream& out) {

if (stack.isEmpty()) {

out << "Stack is empty." << std::endl;

return out.str();

}

out << "Items in the stack: " << stack << std::endl;

return out.str();

}

static std::string deleteItem(Stack<T>& stack, std::ostringstream& out, std::ostringstream& err) {

if (stack.isEmpty()) {

out << "Stack is empty. Cannot delete item." << std::endl;

return out.str();

}

try {

stack.pop();

out << "Item deleted successfully." << std::endl;

return out.str();

}

catch (const std::out\_of\_range& e) {

err << "Error: " << e.what() << std::endl;

return err.str();

}

}

static std::string saveToFile(const Stack<T>& stack, std::ostringstream& out, std::ostringstream& err) {

std::ofstream outputFile("stack\_data.txt");

if (!outputFile.is\_open()) {

err << "Error opening file for writing." << std::endl;

return err.str();

}

outputFile << stack;

out << "Stack data saved to file." << std::endl;

outputFile.close();

return out.str();

}

static std::string loadFromFile(Stack<T>& stack, std::ostringstream& out, std::ostringstream& err) {

std::ifstream inputFile("stack\_data.txt");

if (!inputFile.is\_open()) {

err << "Error opening file for reading." << std::endl;

return err.str();

}

inputFile >> stack;

out << "Stack data loaded from file." << std::endl;

inputFile.close();

return out.str();

}

};

Main.cpp

#include "Class.h"

#include "Node.h"

int main() {

return 0;

}

Test.cpp

#include "pch.h"

#include "C:\Users\windo\Desktop\уник\hashtable\hashtable\Class.h"

#include "C:\Users\windo\Desktop\уник\hashtable\hashtable\Node.h"

TEST(TestCaseName, TestName) {

EXPECT\_EQ(1, 1);

EXPECT\_TRUE(true);

}

TEST(StackTest, Push) {

Stack<int> stack;

stack.push(1);

ASSERT\_FALSE(stack.isEmpty());

}

TEST(StackTest, Pop) {

Stack<int> stack;

stack.push(1);

ASSERT\_EQ(stack.peek(), 1);

}

TEST(StackTest, Peek) {

Stack<int> stack;

ASSERT\_TRUE(stack.isEmpty());

}

TEST(StackTest, Edit) {

Stack<std::string> stack;

std::string value = "3", newValue = "5";

std::ostringstream out, err;

stack.push(value);

std::string result = Stack<std::string>::editItem(stack, out, newValue, err);

ASSERT\_EQ(result, "Item edited successfully.\n");

}

TEST(StackTest, Display) {

Stack<std::string> stack;

std::string value = "3", newValue = "5";

std::ostringstream out;

stack.push(value);

stack.push(newValue);

std::string result = Stack<std::string>::displayItems(stack, out);

ASSERT\_EQ(result, "Items in the stack: 5 3 \n");

}

TEST(StackTest, Delete) {

Stack<std::string> stack;

std::string value = "3", newValue = "5";

std::ostringstream out, err;

stack.push(value);

stack.push(newValue);

std::string result = Stack<std::string>::deleteItem(stack, out, err);

ASSERT\_EQ(result, "Item deleted successfully.\n");

}

TEST(StackTest, SaveFromFile) {

Stack<std::string> stack;

std::ostringstream out, err;

std::string value = "3", newValue = "5";

stack.push(value);

stack.push(newValue);

std::string result = Stack<std::string>::saveToFile(stack, out, err);

ASSERT\_EQ(result, "Stack data saved to file.\n");

}

TEST(StackTest, LoadFromFile) {

Stack<std::string> stack;

std::ostringstream out, err;

std::string value = "3", newValue = "5";

stack.push(value);

stack.push(newValue);

std::string result = Stack<std::string>::loadFromFile(stack, out, err);

ASSERT\_EQ(result, "Stack data loaded from file.\n");

}

1. Результат работы программы

|  |
| --- |
|  |
| 1. Результат тестов |

1. UML диаграмма классов

|  |
| --- |
|  |
| 1. Uml диаграмма классов |