|  |  |
| --- | --- |
| **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:  Студенты группы ТКИ-442  Жижин А.Д.. |
|  | Проверил:  Доцент кафедры УиЗИ к.т.н., с.н.с  Васильева М. А. |
| Москва 2024 | |

Оглавление

[1. Условие задачи 3](#_Toc185432782)

[2. Текст программы на языке С++ 3](#_Toc185432783)

[1.1. ../Stack/Stack.cpp 3](#_Toc185432784)

[1.2. ../Stack/Stack.h 4](#_Toc185432785)

[1.3. ../Demo/Demo.cpp 6](#_Toc185432787)

[3. Результат работы программы 8](#_Toc185432790)

[4. Диаграмма классов 10](#_Toc185432791)

1. Условие задачи

Стек

Определить класс «Node», включающий следующую информацию:

* Данные любого типа;
* Указатели на поддеревья и родителя;
* Указатель на корень дерева;

В программе предусмотреть:

* создание БДП;
* вывод на экран;
* удаление элемента по ключу;
* вставка элемента по ключу;
* поиск данных элемента по ключу;
* поиск минимального и максимального элемента.

1. Текст программы на языке С++
   1. ../Stack/Stack.cpp

// dllmain.cpp : Определяет точку входа для приложения DLL.

#include "pch.h"

#include "Stack.h"

#include <stdexcept>

#include <sstream>

Stack::Stack() {

// Конструктор

}

Stack::~Stack() {

// Деструктор

}

void Stack::push(int value) {

data\_.push\_back(value);

}

int Stack::pop() {

if (data\_.empty()) {

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

}

int val = data\_.back();

data\_.pop\_back();

return val;

}

int Stack::top() const {

if (data\_.empty()) {

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

}

return data\_.back();

}

bool Stack::empty() const {

return data\_.empty();

}

size\_t Stack::size() const {

return data\_.size();

}

bool Stack::contains(int value) const {

for (int elem : data\_) {

if (elem == value) return true;

}

return false;

}

std::string Stack::toString() const {

std::ostringstream oss;

oss << "[";

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

oss << data\_[i];

if (i + 1 < data\_.size()) oss << ", ";

}

oss << "]";

return oss.str();

}

* 1. ../src/Stack.h

#pragma once

#ifndef STACK\_H

#define STACK\_H

#include <vector>

#include <string>

#include <stdexcept>

/\*\*

\* @brief Класс, реализующий структуру данных "Стек" для целых чисел.

\*

\* Стек реализован на основе динамического массива `std::vector<int>`.

\* Поддерживает стандартные операции добавления, удаления элементов,

\* проверки состояния и поиска элементов.

\*/

class Stack {

public:

/\*\*

\* @brief Конструктор по умолчанию.

\*

\* Инициализирует пустой стек.

\*/

Stack();

/\*\*

\* @brief Деструктор.

\*

\* Освобождает ресурсы, если это необходимо.

\* В данном случае ничего специфического не требуется.

\*/

~Stack();

/\*\*

\* @brief Добавляет элемент на вершину стека.

\*

\* @param value Значение типа `int`, которое будет добавлено в стек.

\*/

void push(int value);

/\*\*

\* @brief Удаляет элемент с вершины стека и возвращает его.

\*

\* @return Значение типа `int`, которое было удалено с вершины стека.

\* @throws std::out\_of\_range Если стек пустой.

\*/

int pop();

/\*\*

\* @brief Возвращает значение элемента на вершине стека без его удаления.

\*

\* @return Значение типа `int`, находящееся на вершине стека.

\* @throws std::out\_of\_range Если стек пустой.

\*/

int top() const;

/\*\*

\* @brief Проверяет, пуст ли стек.

\*

\* @return `true`, если стек пустой, иначе `false`.

\*/

bool empty() const;

/\*\*

\* @brief Возвращает количество элементов в стеке.

\*

\* @return Размер стека типа `size\_t`.

\*/

size\_t size() const;

/\*\*

\* @brief Проверяет, содержится ли заданное значение в стеке.

\*

\* @param value Значение типа `int`, которое необходимо найти в стеке.

\* @return `true`, если значение найдено, иначе `false`.

\*/

bool contains(int value) const;

/\*\*

\* @brief Возвращает строковое представление стека.

\*

\* Формат строки: [элемент1, элемент2, ..., элементN]

\*

\* @return Строка, представляющая содержимое стека.

\*/

std::string toString() const;

private:

std::vector<int> data\_; ///< Внутреннее хранилище элементов стека.

};

#endif // STACK\_H

* 1. ../Demo/Demo.cpp

#include <iostream>

#include "../Stack/Stack.h"

int main() {

Stack s;

s.push(10);

s.push(20);

s.push(30);

std::cout << "Stack content: " << s.toString() << std::endl;

std::cout << "Top element: " << s.top() << std::endl;

std::cout << "Pop element: " << s.pop() << std::endl;

std::cout << "Stack after pop: " << s.toString() << std::endl;

std::cout << "Contains 20? " << (s.contains(20) ? "Yes" : "No") << std::endl;

std::cout << "Contains 100? " << (s.contains(100) ? "Yes" : "No") << std::endl;

return 0;

}

* 1. ../src/CmakeLists.txt

cmake\_minimum\_required(VERSION 3.0)  
project("BinarySearchTree")  
  
set(CMAKE\_CXX\_STANDARD 20)  
  
add\_executable(7th\_semester  
 Node.hpp  
 BST.hpp  
 main.cpp  
)

* 1. ../Stack\_Test/Stack\_Test.cpp

#include "pch.h"

#include "CppUnitTest.h"

#include "../Stack/Stack.h"

using namespace Microsoft::VisualStudio::CppUnitTestFramework;

namespace UnitTest

{

TEST\_CLASS(StackTest)

{

public:

TEST\_METHOD(Stack\_Empty\_OnCreation\_IsEmpty)

{

Stack s;

Assert::IsTrue(s.empty(), L"Новый стек должен быть пустым.");

Assert::AreEqual((size\_t)0, s.size(), L"Размер пустого стека должен быть 0.");

}

TEST\_METHOD(Stack\_Push\_WhenOneElementPushed\_SizeIsOne)

{

Stack s;

s.push(10);

Assert::AreEqual((size\_t)1, s.size(), L"После добавления одного элемента размер должен быть 1.");

Assert::IsFalse(s.empty(), L"Стек не должен быть пустым после добавления элемента.");

}

TEST\_METHOD(Stack\_Push\_WhenMultipleElementsPushed\_TopIsLastElement)

{

Stack s;

s.push(10);

s.push(20);

s.push(30);

Assert::AreEqual(30, s.top(), L"top() должен возвращать последний добавленный элемент (30).");

}

TEST\_METHOD(Stack\_Pop\_WhenCalledOnNonEmptyStack\_ReturnsLastPushedElement)

{

Stack s;

s.push(10);

s.push(20);

int popped = s.pop();

Assert::AreEqual(20, popped, L"pop() должен вернуть последний добавленный элемент (20).");

Assert::AreEqual((size\_t)1, s.size(), L"После pop размер должен уменьшиться на 1.");

}

TEST\_METHOD(Stack\_Pop\_WhenCalledOnEmptyStack\_ThrowsOutOfRange)

{

Stack s;

bool exceptionThrown = false;

try {

s.pop();

}

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

exceptionThrown = true;

}

Assert::IsTrue(exceptionThrown, L"pop() должен бросать исключение при вызове на пустом стеке.");

}

TEST\_METHOD(Stack\_Top\_WhenCalledOnEmptyStack\_ThrowsOutOfRange)

{

Stack s;

bool exceptionThrown = false;

try {

int value = s.top(); // используем, чтобы подавить warning о неиспользуемой переменной

(void)value;

}

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

exceptionThrown = true;

}

Assert::IsTrue(exceptionThrown, L"top() должен бросать исключение при вызове на пустом стеке.");

}

TEST\_METHOD(Stack\_Contains\_WhenElementIsPresent\_ReturnsTrue)

{

Stack s;

s.push(10);

s.push(20);

Assert::IsTrue(s.contains(10), L"contains(10) должен вернуть true, если 10 присутствует в стеке.");

Assert::IsTrue(s.contains(20), L"contains(20) должен вернуть true, если 20 присутствует в стеке.");

}

TEST\_METHOD(Stack\_Contains\_WhenElementIsNotPresent\_ReturnsFalse)

{

Stack s;

s.push(10);

Assert::IsFalse(s.contains(30), L"contains(30) должен вернуть false, если 30 отсутствует в стеке.");

}

TEST\_METHOD(Stack\_ToString\_WhenMultipleElements\_ReturnsCorrectFormat)

{

Stack s;

s.push(5);

s.push(10);

Assert::AreEqual(std::string("[5, 10]"), s.toString(), L"toString() должен вернуть \"[5, 10]\".");

}

};

}

1. Результат работы программы

|  |
| --- |
|  |
| 1. Результат работы Demo.cpp |

|  |
| --- |
|  |
| 1. UnitTests |

1. Диаграмма классов

|  |
| --- |
|  |
| 1. Диаграмма классов |