|  |  |
| --- | --- |
| **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»** | |
|  | Выполнили:  Студенты группы ТКИ-442  Зинченко Б.А.  Круглов В.А.  Проверил:  Доцент кафедры УиЗи, к.т.н., с.н.с.  Васильева М.А. |
| Москва 2023 | |

Оглавление

[Задание 3](#_Toc153532003)

[1. Текст программы на языке C++ 3](#_Toc153532004)

[1.1. Код файла node.hpp 3](#_Toc153532005)

[1.2. Код файла bst.hpp 4](#_Toc153532006)

[1.3. Код файлa main.cpp 8](#_Toc153532007)

[1.4. Код файлa bstTest.cpp – тесты для bst.hpp 9](#_Toc153532008)

[2. Результат работы программы 11](#_Toc153532009)

[3. UML диаграмма классов 12](#_Toc153532010)

[Заключение 13](#_Toc153532011)

# Задание

Разработать структуру данных на языке программирования С++ в ООП парадигме. В нашем случае структура данных – это бинарное дерево поиска (далее - БДП), основные операции которого будут поиск элемента, добавление элемента, удаление узла и обход дерева. Структура способна обрабатывать любые типы данных (template).

# 1. Текст программы на языке C++

## Код файла node.hpp

#ifndef NODE\_HPP

#define NODE\_HPP

template**<**typename T**>**

struct Node **{**

T key**;**

Node**<**T**>\*** left**;**

Node**<**T**>\*** right**;**

/\*\*

\* **@brief** Конструктор узла

\*/

Node**();**

/\*\*

\* **@brief** Конструктор узла

\* **@param** key Ключ узла

\*/

explicit Node**(**T key**)** **:** key**(**key**),** left**(nullptr),** right**(nullptr)** **{}**

/\*\*

\* **@brief** Конструктор копирования

\* **@param** other Копируемый узел

\*/

Node**(**const Node**&** other**)** **=** **delete;**

/\*\*

\* **@brief** Оператор присваивания копированием

\* **@param** other Копируемый узел

\*/

Node**&** **operator=(**const Node**&** other**)** **=** **delete;**

/\*\*

\* **@brief** Конструктор перемещения

\* **@param** other Копируемый узел

\*/

Node**(**Node**&&** other**)** noexcept **=** **default;**

/\*\*

\* **@brief** Оператор присваивания перемещением

\* **@param** other Копируемый узел

\*/

Node**&** **operator=(**Node**&&** other**)** noexcept **=** **default;**

/\*\*

\* **@brief** Оператор вывода узла

\* **@param** stream Входной поток

\* **@param** node Выводимый узел

\* **@return** Выходной поток

\*/

friend std**::**ostream**&** **operator<<(**std**::**ostream**&** stream**,** const Node**&** node**){**

stream **<<** node**.**key**;**

**return** stream**;**

**}**

**};**

template **<**typename T**>**

Node**<**T**>::**Node**()** **:** key**{** 0 **},** left**{** **nullptr** **},** right**{** **nullptr** **}** **{}**

#endif

## Код файла bst.hpp

#ifndef BST\_HPP

#define BST\_HPP

#include "node.hpp"

#include <sstream>

template**<**typename T**>**

class BST **{**

private**:**

Node**<**T**>\*** root**;**

/\*\*

\* **@brief** Вставка ключа в узел дерева

\* **@param** node Узел для добавления

\* **@param** key Ключ для добавления

\* **@return** Узел дерева

\*/

Node**<**T**>\*** insert**(**Node**<**T**>\*** node**,** T key**);**

/\*\*

\* **@brief** Удаление ключа из узла дерева

\* **@param** root Корневой узел

\* **@param** key Ключ для удаления

\* **@return** Узел дерева.

\*/

Node**<**T**>\*** deleteNode**(**Node**<**T**>\*** root**,** T key**);**

/\*\*

\* **@brief** Вывод узлов дерева в центрированном порядке

\* **@param** root Корень дерева

\* **@param** os Выходной поток

\*/

void inorder**(**Node**<**T**>\*** root**,** std**::**ostream**&** os**)** const**;**

/\*\*

\* **@brief** Поиск ключа в дереве

\* **@param** root Корень дерева

\* **@param** key Ключ для поиска

\* **@return** Узел с ключом, если он существует

\*/

Node**<**T**>\*** search**(**Node**<**T**>\*** root**,** T key**)** const**;**

public**:**

/\*\*

\* **@brief** Конструктор дерева

\*/

BST**();**

/\*\*

\* **@brief** Конструктор копирования

\* **@param** other Копируемое дерево

\*/

BST**(**const BST**<**T**>&** other**)** **=** **delete;**

/\*\*

\* **@brief** Конструктор перемещения

\* **@param** other Копируемое дерево

\*/

BST**(**BST**<**T**>&&** other**)** noexcept **=** **default;**

/\*\*

\* **@brief** Оператор присваивания копированием

\* **@param** other Копируемое дерево

\*/

BST**&** **operator=(**const BST**<**T**>&** other**)** **=** **delete;**

/\*\*

\* **@brief** Оператор присваивания перемещением

\* **@param** other Копируеое дерево

\*/

BST**&** **operator=(**BST**&&** other**)** noexcept **=** **default;**

/\*\*

\* **@brief** Деструктор дерева

\*/

**~**BST**();**

/\*\*

\* **@brief** Вставка ключа в дерево

\* **@param** key Ключ для вставки

\*/

void insert**(**T key**);**

/\*\*

\* **@brief** Удаление ключа из дерева

\* **@param** key Ключ для удаления

\*/

void deleteKey**(**T key**);**

/\*\*

\* **@brief** Вывод дерева

\*/

void print**()** const**;**

/\*\*

\* **@brief** Поиск ключа в дереве

\* **@param** key Ключ для поиска

\* **@return** Узел с ключом, если он существует

\*/

Node**<**T**>\*** search**(**T key**)** const**;**

/\*\*

\* **@brief** Наименьший узел в дереве

\* **@param** node Узел для поиска

\* **@return** Узел с наименьшим ключом

\*/

Node**<**T**>\*** minValueNode**(**Node**<**T**>\*** node**);**

/\*\*

\* **@brief** Получить корень дерева

\* **@return** Корень

\*/

Node**<**T**>\*** getRoot**();**

/\*\*

\* **@brief** Дерево в строку

\* **@return** Строка

\*/

std**::**string to\_string**()** const **{**

std**::**ostringstream oss**;**

inorder**(**root**,** oss**);**

**return** oss**.**str**();**

**}**

**};**

template**<**typename T**>**

BST**<**T**>::**BST**()** **:** root**(nullptr)** **{}**

template**<**typename T**>**

Node**<**T**>\*** BST**<**T**>::**getRoot**()** **{**

**return** root**;**

**}**

template**<**typename T**>**

Node**<**T**>\*** BST**<**T**>::**insert**(**Node**<**T**>\*** node**,** T key**)** **{**

**if** **(**node **==** **nullptr)**

**return** **new** Node**<**T**>(**key**);**

**if** **(**key **<** node**->**key**)**

node**->**left **=** insert**(**node**->**left**,** key**);**

**else** **if** **(**key **>** node**->**key**)**

node**->**right **=** insert**(**node**->**right**,** key**);**

**return** node**;**

**}**

template**<**typename T**>**

Node**<**T**>\*** BST**<**T**>::**deleteNode**(**Node**<**T**>\*** root**,** T key**)** **{**

**if** **(**root **==** **nullptr)**

**return** root**;**

**if** **(**key **<** root**->**key**)**

root**->**left **=** deleteNode**(**root**->**left**,** key**);**

**else** **if** **(**key **>** root**->**key**)**

root**->**right **=** deleteNode**(**root**->**right**,** key**);**

**else** **{**

**if** **(**root**->**left **==** **nullptr)** **{**

Node**<**T**>\*** temp **=** root**->**right**;**

**delete** root**;**

**return** temp**;**

**}**

**else** **if** **(**root**->**right **==** **nullptr)** **{**

Node**<**T**>\*** temp **=** root**->**left**;**

**delete** root**;**

**return** temp**;**

**}**

Node**<**T**>\*** temp **=** minValueNode**(**root**->**right**);**

root**->**key **=** temp**->**key**;**

root**->**right **=** deleteNode**(**root**->**right**,** temp**->**key**);**

**}**

**return** root**;**

**}**

template**<**typename T**>**

Node**<**T**>\*** BST**<**T**>::**minValueNode**(**Node**<**T**>\*** root**)** **{**

**if** **(**root **==** **nullptr)**

**return** **nullptr;**

**else** **if** **(**root**->**left **==** **nullptr)**

**return** root**;**

**else**

**return** minValueNode**(**root**->**left**);**

**}**

template**<**typename T**>**

void BST**<**T**>::**inorder**(**Node**<**T**>\*** root**,** std**::**ostream**&** os**)** const**{**

**if** **(**root **!=** **nullptr)** **{**

inorder**(**root**->**left**,** os**);**

os **<<** root**->**key **<<** " "**;**

inorder**(**root**->**right**,** os**);**

**}**

**}**

template**<**typename T**>**

Node**<**T**>\*** BST**<**T**>::**search**(**Node**<**T**>\*** root**,** T key**)** const **{**

**if** **(**root **==** **nullptr** **||** root**->**key **==** key**)**

**return** root**;**

**if** **(**root**->**key **<** key**)**

**return** search**(**root**->**right**,** key**);**

**return** search**(**root**->**left**,** key**);**

**}**

template**<**typename T**>**

BST**<**T**>::~**BST**()** **{**

**while(**root **!=** **nullptr)**

deleteKey**(**root**->**key**);**

**}**

template**<**typename T**>**

void BST**<**T**>::**insert**(**T key**)** **{**

root **=** insert**(**root**,** key**);**

**}**

template**<**typename T**>**

void BST**<**T**>::**deleteKey**(**T key**)** **{**

root **=** deleteNode**(**root**,** key**);**

**}**

template**<**typename T**>**

void BST**<**T**>::**print**()** const **{**

std**::**ostringstream oss**;**

inorder**(**root**,** oss**);**

std**::**cout **<<** oss**.**str**()** **<<** "\n"**;**

**}**

template**<**typename T**>**

std**::**ostream**&** **operator<<(**std**::**ostream**&** os**,** const BST**<**T**>&** bst**)** **{**

bst**.**print**();**

**return** os**;**

**}**

template**<**typename T**>**

Node**<**T**>\*** BST**<**T**>::**search**(**T key**)** const **{**

**return** search**(**root**,** key**);**

**}**

template class BST**<**int**>;**

#endif

## Код файлa main.cpp

#include <iostream>

#include <Windows.h>

#include "bst.hpp"

int main**()** **{**

SetConsoleOutputCP**(**CP\_UTF8**);**

BST**<**int**>** tree**;**

tree**.**insert**(**50**);**

tree**.**insert**(**30**);**

tree**.**insert**(**20**);**

tree**.**insert**(**40**);**

tree**.**insert**(**70**);**

tree**.**insert**(**60**);**

tree**.**insert**(**80**);**

std**::**cout **<<** "Вывод дерева:\n"**;**

tree**.**print**();**

std**::**cout **<<** "Удаление узла с ключом 20\n"**;**

tree**.**deleteKey**(**20**);**

std**::**cout **<<** "Вывод измененного дерева:\n"**;**

tree**.**print**();**

std**::**cout **<<** "Поиск узла с ключом 60\n"**;**

Node**<**int**>\*** searchResult **=** tree**.**search**(**60**);**

**if** **(**searchResult **!=** **nullptr)**

std**::**cout **<<** "Найден узел с ключом " **<<** searchResult**->**key **<<** "\n"**;**

**else**

std**::**cout **<<** "Узел с ключом 60 не найден в дереве\n"**;**

std**::**cout **<<** tree**;**

Node**<**int**>\*** minNode **=** tree**.**minValueNode**(**tree**.**getRoot**());**

**if** **(**minNode **!=** **nullptr)**

std**::**cout **<<** "Минимальный элемент: " **<<** minNode**->**key **<<** std**::**endl**;**

**else**

std**::**cout **<<** "Дерево пустое" **<<** std**::**endl**;**

tree**.**deleteKey**(**60**);**

std**::**cout **<<** tree**;**

tree**.**insert**(**45**);**

std**::**cout **<<** tree**;**

std**::**cout **<<** tree**.**to\_string**();**

**return** 0**;**

**}**

## Код файлa bstTest.cpp – тесты для bst.hpp

#include "pch.h"

#include "CppUnitTest.h"

#include "..\ConsoleApplication4\bst.h"

#include <iostream>

#include<string>

#include <sstream>

**using** **namespace** Microsoft**::**VisualStudio**::**CppUnitTestFramework**;**

**namespace** bstTest

**{**

TEST\_CLASS**(**bstTest**)**

**{**

public**:**

TEST\_METHOD**(**TestConstructorIntSuccess**)**

**{**

// Arrange

BST**<**int**>\*** bst **=** **new** BST**<**int**>();**

bst**->**insert**(**5**);**

bst**->**insert**(**3**);**

bst**->**insert**(**7**);**

// Act & Assert

Assert**::**IsNotNull**(**bst**);**

Assert**::**IsNotNull**(**bst**->**search**(**5**));**

Assert**::**IsNotNull**(**bst**->**search**(**3**));**

Assert**::**IsNotNull**(**bst**->**search**(**7**));** // Tree is notempty initially

**}**

TEST\_METHOD**(**TestConstructorStringSuccess**)**

**{**

// Arrange

BST**<**std**::**string**>\*** bst **=** **new** BST**<**std**::**string**>();**

bst**->**insert**(**"5"**);**

bst**->**insert**(**"3"**);**

bst**->**insert**(**"7"**);**

// Act & Assert

Assert**::**IsNotNull**(**bst**);**

Assert**::**IsNotNull**(**bst**->**search**(**"5"**));**

Assert**::**IsNotNull**(**bst**->**search**(**"3"**));**

Assert**::**IsNotNull**(**bst**->**search**(**"7"**));** // Tree is notempty initially

**}**

TEST\_METHOD**(**Insert\_Int\_Success**)**

**{** // Arrange

BST**<**int**>\*** bst **=** **new** BST**<**int**>();**

bst**->**insert**(**5**);**

bst**->**insert**(**3**);**

bst**->**insert**(**7**);**

// Act & Assert

Assert**::**IsNotNull**(**bst**);**

Assert**::**IsNotNull**(**bst**->**search**(**5**));**

Assert**::**IsNotNull**(**bst**->**search**(**3**));**

Assert**::**IsNotNull**(**bst**->**search**(**7**));**

**}**

TEST\_METHOD**(**Insert\_String\_Success**)**

**{** // Arrange

BST**<**std**::**string**>\*** bst **=** **new** BST**<**std**::**string**>();**

bst**->**insert**(**"5"**);**

bst**->**insert**(**"3"**);**

bst**->**insert**(**"7"**);**

// Act & Assert

Assert**::**IsNotNull**(**bst**);**

Assert**::**IsNotNull**(**bst**->**search**(**"5"**));**

Assert**::**IsNotNull**(**bst**->**search**(**"3"**));**

Assert**::**IsNotNull**(**bst**->**search**(**"7"**));**

**}**

TEST\_METHOD**(**DeleteStringSuccess**)**

**{**

BST**<**std**::**string**>\*** bst **=** **new** BST**<**std**::**string**>();**

bst**->**insert**(**"5"**);**

bst**->**insert**(**"3"**);**

bst**->**insert**(**"7"**);**

// Act & Assert

Assert**::**IsNotNull**(**bst**);**

Assert**::**IsNotNull**(**bst**->**search**(**"5"**));**

Assert**::**IsNotNull**(**bst**->**search**(**"3"**));**

Assert**::**IsNotNull**(**bst**->**search**(**"7"**));**

bst**->**deleteKey**(**"7"**);**

Assert**::**IsNull**(**bst**->**search**(**"7"**));**

**}**

TEST\_METHOD**(**DeleteIntSuccess**)**

**{**

// Arrange

BST**<**int**>\*** bst **=** **new** BST**<**int**>();**

bst**->**insert**(**5**);**

bst**->**insert**(**3**);**

bst**->**insert**(**7**);**

//Assert

Assert**::**IsNotNull**(**bst**);**

Assert**::**IsNotNull**(**bst**->**search**(**5**));**

Assert**::**IsNotNull**(**bst**->**search**(**3**));**

Assert**::**IsNotNull**(**bst**->**search**(**7**));**

bst**->**deleteKey**(**7**);**

Assert**::**IsNull**(**bst**->**search**(**7**));**

**}**

TEST\_METHOD**(**SearchIntSuccess**)**

**{**

BST**<**int**>\*** bst **=** **new** BST**<**int**>();**

bst**->**insert**(**5**);**

bst**->**insert**(**3**);**

bst**->**insert**(**7**);**

// Act & Assert

Assert**::**IsNotNull**(**bst**);**

Assert**::**IsNotNull**(**bst**->**search**(**5**));**

Assert**::**IsNotNull**(**bst**->**search**(**3**));**

Assert**::**IsNotNull**(**bst**->**search**(**7**));**

**}**

TEST\_METHOD**(**SearchStringSuccess**)**

**{** // Arrange

BST**<**std**::**string**>\*** bst **=** **new** BST**<**std**::**string**>();**

bst**->**insert**(**"5"**);**

bst**->**insert**(**"3"**);**

bst**->**insert**(**"7"**);**

// Act & Assert

Assert**::**IsNotNull**(**bst**);**

Assert**::**IsNotNull**(**bst**->**search**(**"5"**));**

Assert**::**IsNotNull**(**bst**->**search**(**"3"**));**

Assert**::**IsNotNull**(**bst**->**search**(**"7"**));**

**}**

**};**

**};**

# 2. Результат работы программы

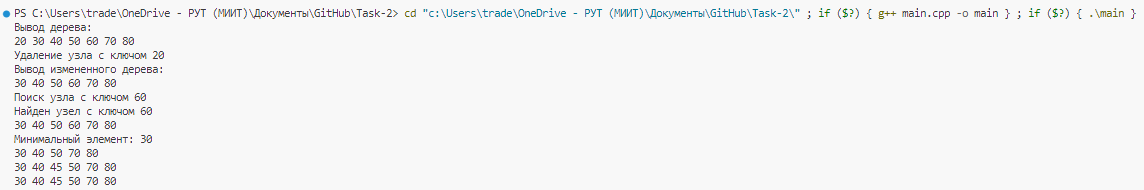


Рисунок 1 – Результат отладки программы

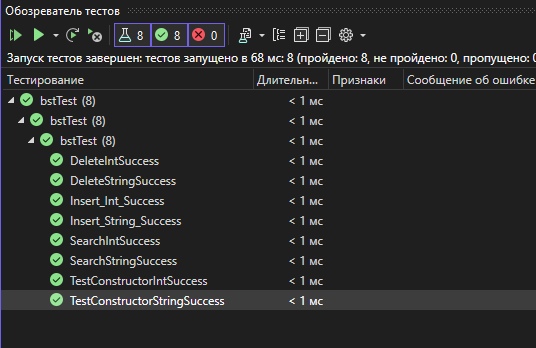


Рисунок 2 – Результат выполнения тестов для bst.hpp

# 3. UML диаграмма классов

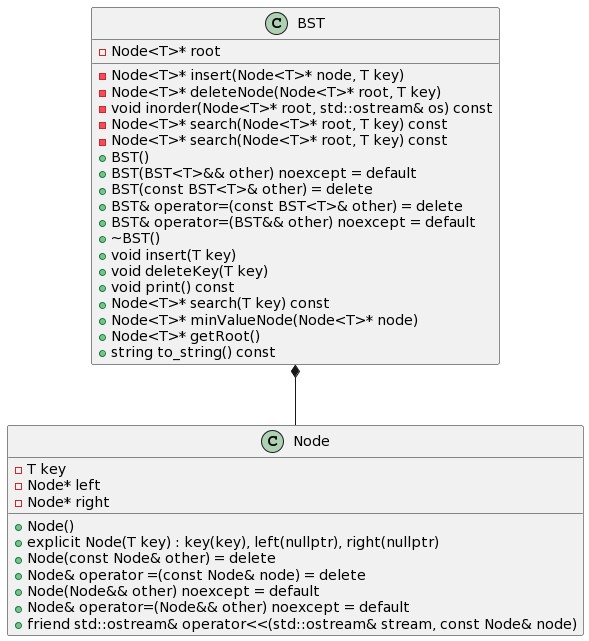


Рисунок 3 – UML диаграмма классов Node и BST

# Заключение

В результате выполнения практического задания была разработана структура данных - бинарное дерево поиска. Для обеспечения обработки различных типов данных была использована техника шаблонов, что позволило использовать данную структуру с любым типом данных.