|  |
| --- |
| Федеральное государственное бюджетное образовательное учреждение высшего профессионального образования |
| Московский Государственный Технический Университет имени Н.Э. Баумана |
|  |

Факультет              ФУНДАМЕНТАЛЬНЫЕ НАУКИ

Кафедра             МАТЕМАТИКА И МАТЕМАТИЧЕСКАЯ ФИЗИКА

**Отчет по лабораторной работе № 18**

**по курсу «Информатика»**

Студента           Борисов Иван Дмитриевич ­­­­­­­­­­­

(фамилия, имя, отчество)

Группа                                          ФН11-22Б

Преподаватель \_   доцент, к.т.н. Ничушкина Т. Н.

Должность, ФИО, подпись

2019

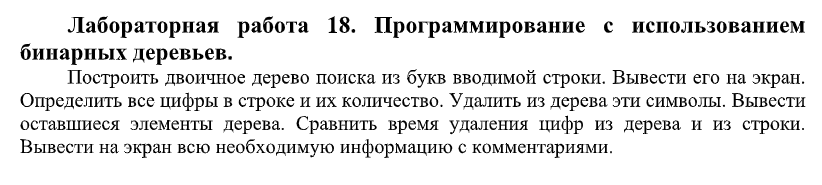
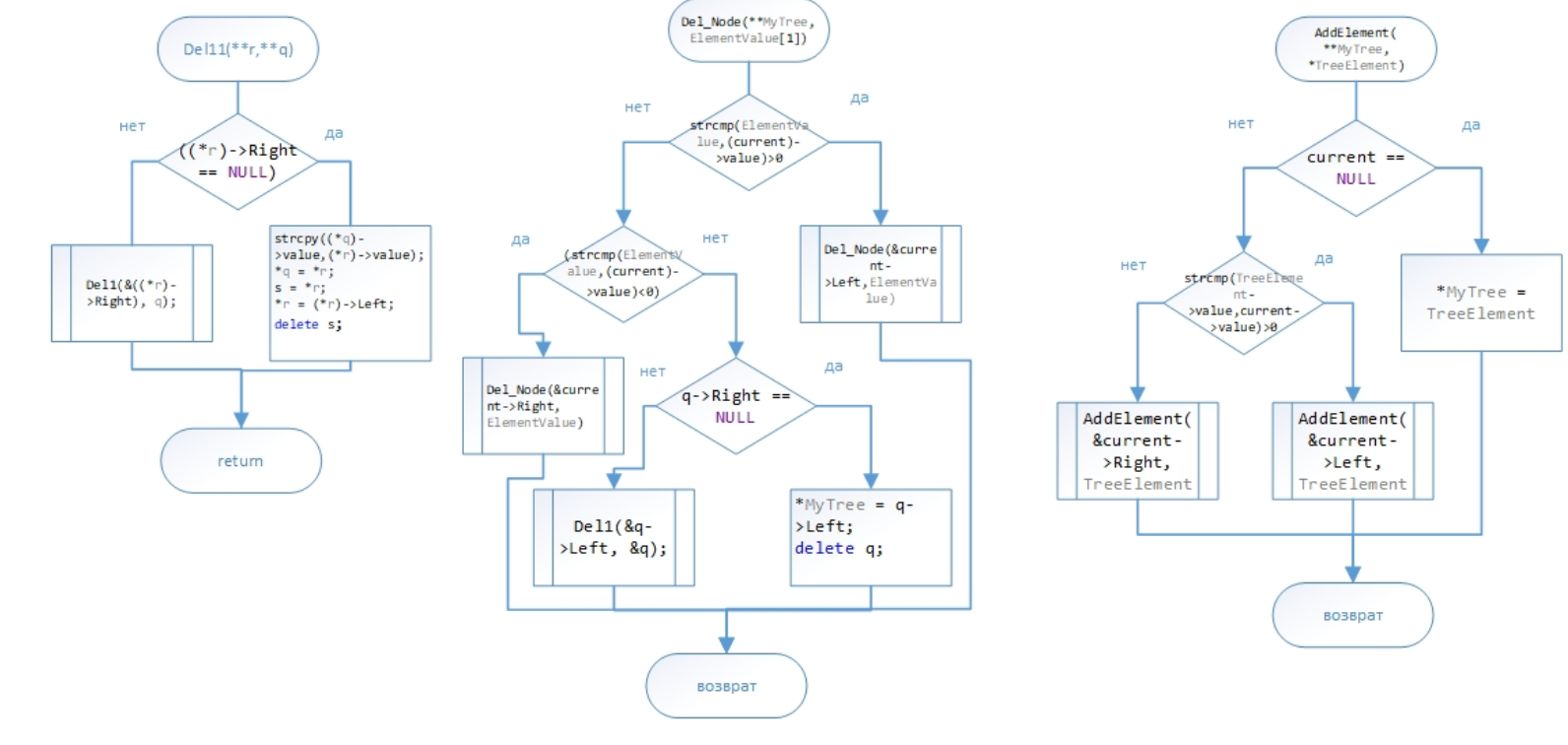
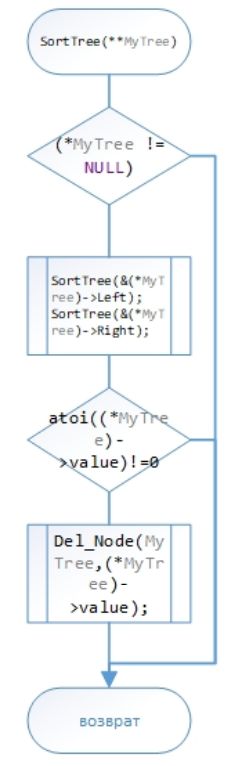
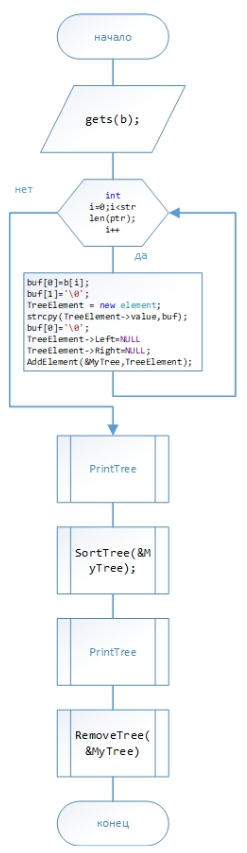


Схема алгоритма





Текст программы

#include "stdafx.h"

#include <conio.h>

#include <locale.h>

#include <ctime>

#include <ctype.h>

#include <string.h>

#include <stdio.h>

#include <stdlib.h>

#include <iostream>

using namespace std;

struct element // тип элемента

{

char value[80]; // информационное поле 1

element \*Left; // адресное поле ≪левый≫

element \*Right; // адресное поле ≪правый≫

};

void Del1(element \*\*r, element \*\*q) // вспомогательная функция для удаления вершины

{

element \*s;

if ((\*r)->Right == NULL)

{

strcpy((\*q)->value,(\*r)->value);

\*q = \*r;

s = \*r;

\*r = (\*r)->Left;

delete s;

}

else

Del1(&((\*r)->Right), q);

}

void Del\_Node(element \*\*MyTree, char ElementValue[1]) // удаление вершины из дерева

{

element \*current, \*q;

current = \*MyTree;

if (\*MyTree == NULL) printf("\n Node with this value don't find! \n");

else

if (strcmp(ElementValue,(current)->value)>0)

Del\_Node(&current->Left,ElementValue); //если вершина является листом

else

if (strcmp(ElementValue,(current)->value)<0)

Del\_Node(&current->Right, ElementValue);

else

{

q = \*MyTree;

if (q->Right == NULL)

{

\*MyTree = q->Left;

delete q;

} //если вершина имеет одну дочернюю вершину

else

if (q->Left == NULL)

{

\*MyTree = q->Right;

delete q;

} //если у вершины две дочерних вершины

else

Del1(&q->Left, &q); /\*спускаемся

вдоль правой ветви левого поддерева удаляемой вершины и

затем заменяем существующую информацию соответствующими

значениями самой правой компоненты этого левого поддерева,

после чего от правой компоненты можно избавиться, удаляем ее

с помощью delete\*/

}

}

void AddElement(element \*\*MyTree, element \*TreeElement) //рекурсивная функция создания/добавления вершины дерева

{

element \* current;

current = \*MyTree; //вспомогательный указатель на указатель дерева

if (current == NULL)

\*MyTree = TreeElement; //если узла нет, создаем узел

else

//если значение текущего узла поддерева меньше значения узла дерева,

if (strcmp(TreeElement->value,current->value)>0)

//то зависываем значение узла в левое поддерево

AddElement(&current->Left, TreeElement);

//иначе зависываем значение узла в правое поддерево

else

AddElement(&current->Right, TreeElement);

}

void PreOrder\_ShowTree(element \*MyTree) //рекурсивная функция печати бинарного дерева

{

if (MyTree != NULL) //пока не встретится пустой узел

{

printf("%4s", MyTree->value); //печать узла дерева

PreOrder\_ShowTree (MyTree->Left); //рекурсивный вызов для левого поддерева

PreOrder\_ShowTree (MyTree->Right); //рекурсивный вызов для правого поддерева

}

}

void SymmetricOrder\_ShowTree(element \*MyTree) //рекурсивная функция печати бинарного дерева

{

if (MyTree != NULL) //пока не встретится пустой узел

{

SymmetricOrder\_ShowTree (MyTree->Left); //рекурсивный вызов для левого поддерева

printf("%4s", MyTree->value); //печать узла дерева

SymmetricOrder\_ShowTree (MyTree->Right); //рекурсивный вызов для правого поддерева

}

}

void PostOrder\_ShowTree(element \*MyTree) //рекурсивная функция печати бинарного дерева

{

if (MyTree != NULL) //пока не встретится пустой узел

{

PostOrder\_ShowTree (MyTree->Left); //рекурсивный вызов для левого поддерева

PostOrder\_ShowTree (MyTree->Right); //рекурсивный вызов для правого поддерева

printf("%4s", MyTree->value); //печать узла дерева

}

}

void RemoveTree(element\*\*MyTree) // очистка памяти

{

element \*current;

current = \*MyTree;

if (\*MyTree != NULL){

RemoveTree(&(\*MyTree)->Left);

RemoveTree(&(\*MyTree)->Right);

delete \*MyTree;

}

}

void SortTree(element\*\*MyTree)

{

element \*current;

current = \*MyTree;

if (\*MyTree != NULL){

SortTree(&(\*MyTree)->Left);

SortTree(&(\*MyTree)->Right);

if (atoi((\*MyTree)->value)!=0)

{

puts(" ");

printf("\nчисло из дерева %4d",atoi((\*MyTree)->value));

Del\_Node(MyTree,(\*MyTree)->value);

}

}

}

int main(int argc, char\* argv[])

{

char b[80], \*ptr,buf[80];

setlocale(0,"russian");

puts("введите строку");

gets(b);

puts("исходная строка");

puts(b);

ptr = b;

while ((ptr = strstr(ptr, " "))!=NULL)

strcpy(ptr, ptr + 1);

ptr = b;

if (b[0] == ' ')

strcpy(ptr, ptr + 1);

if (b[strlen(b) - 1] == ' ')

b[strlen(b) - 1] = '\0';

puts("строка без лишних пробелов");

puts(b);

ptr=b;

element \* MyTree =NULL;

element \*TreeElement;

for (int i=0;i<strlen(ptr);i++)

{

buf[0]=b[i];

buf[1]='\0';

TreeElement = new element;

strcpy(TreeElement->value,buf);

buf[0]='\0';

TreeElement->Left=NULL;

TreeElement->Right=NULL;

AddElement(&MyTree,TreeElement);

}

puts(" ");

puts("Исходное дерево");

//Вывод дерева разными способами

puts(" ");

PreOrder\_ShowTree(MyTree);

puts(" ");

SymmetricOrder\_ShowTree(MyTree);

puts(" ");

PostOrder\_ShowTree(MyTree);

puts(" ");

const std::chrono::high\_resolution\_clock::time\_point t1=std::chrono::high\_resolution\_clock::now();

SortTree(&MyTree);

const std::chrono::high\_resolution\_clock::time\_point t2=std::chrono::high\_resolution\_clock::now();

puts(" ");

puts("Время удаления из дерева ");

std::cout << std::chrono::duration\_cast<std::chrono::microseconds>(t2 - t1).count(); puts("\nИзмененное дерево");

//Вывод дерева разными способами

puts(" ");

PreOrder\_ShowTree(MyTree);

puts(" ");

SymmetricOrder\_ShowTree(MyTree);

puts(" ");

PostOrder\_ShowTree(MyTree);

//Удаление дерева

puts(" ");

puts(" ");

puts("Удаление из строки");

puts(b);

ptr=b;

const std::chrono::high\_resolution\_clock::time\_point t3=std::chrono::high\_resolution\_clock::now();

int n=strlen(b);

for (int i=n-1;i>=0;i--)

{

if (isdigit(ptr[i]))

{

for (int j=i;j<n;j++)

{

ptr[j]=ptr[j+1];

}

n--;

}

}

puts(ptr);

puts(" ");

puts("Время удаления из строки: ");

const std::chrono::high\_resolution\_clock::time\_point t4=std::chrono::high\_resolution\_clock::now();

std::cout << std::chrono::duration\_cast<std::chrono::microseconds>(t4 - t2).count(); RemoveTree(&MyTree);

puts(" ");

system("pause");

return 0;

}

Таблица тестовых примеров

|  |  |  |
| --- | --- | --- |
| Входные данные | Ожидаемый результат | Вывод программы |
| a12b | a b  b a  b a |  |
| sadabsdj213123sabdjbad12313213 | sadsjsjbdddbbaaa  sssjjddddbbbaaaa  sjjsdddbbbdaaaas |  |

Из тестов мы видим, что время сортировки зависит от того, какое дерево было построено.

Вывод

Я закрепил теоретические знания и практические навыки, необходимые при разработке алгоритмов и программ с использованием иерархических структур данных, научился распознавать такие структуры и реализовывать алгоритмы с ними.