Министерство науки и высшего образования Российской Федерации  
Федеральное государственное автономное учреждение высшего образования  
«Пермский национальный исследовательский политехнический университет»

Электротехнический факультет  
Кафедра «Информационные технологии и автоматизированные системы»

**О Т Ч Ё Т  
по лабораторной работе**

Дисциплина: «Основы алгоритмизации и программирования»

Выполнил  
студент группы РИС-22-1б  
Бадртдинов Т.З

Проверил  
доцент кафедры ИТАС Полякова О.А.

Пермь 2023

Постановка задачи:

Требуется реализовать алгоритмы для собственного варианта бинарного дерева поиска, имеющего не менее трёх уровней .

Алгоритмы:

1. Необходимо реализовать функции для редактирования дерева: - Вставка узла.

- Удаление узла.

– Поиск элемента по ключу.

2. Реализовать алгоритмы обхода дерева:

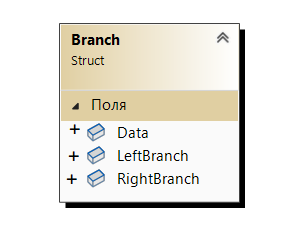
2.1 Прямой

2.2 Симметричный

2.3 Обратный

3. Реализовать алгоритм балансировки дерева.

Алгоритм программы



Код программы:

#include <iostream>

using namespace std;

int key\_count = 0;

int tabs = 0;

int kol\_vo = 0;

struct Branch

{

char Data; //Поле данных

Branch\* LeftBranch; //УКАЗАТЕЛИ на соседние веточки

Branch\* RightBranch;

};

void Add(char aData, Branch\*& aBranch)

{

if (!aBranch)

{

aBranch = new Branch;

aBranch->Data = aData;

aBranch->LeftBranch = 0;

aBranch->RightBranch = 0;

return;

}

else

if (aBranch->Data > aData)

{

Add(aData, aBranch->LeftBranch);

}

else

{

Add(aData, aBranch->RightBranch);

};

}

void print(Branch\* aBranch)

{

if (!aBranch) return;

tabs += 5;

print(aBranch->LeftBranch);

for (int i = 0; i < tabs; i++) cout << " ";

cout << aBranch->Data << endl;

print(aBranch->RightBranch);

tabs -= 5;

return;

}

void pr\_obh(Branch\*& aBranch)

{

if (NULL == aBranch) return;

pr\_obh(aBranch->LeftBranch);

cout << aBranch->Data << endl;

pr\_obh(aBranch->RightBranch);

}

int key\_counter(Branch\*& aBranch, char key)

{

if (NULL == aBranch) return 0;

if (aBranch->Data == key)

{

key\_count++;

}

key\_counter(aBranch->LeftBranch, key);

key\_counter(aBranch->RightBranch, key);

return key\_count;

}

void add\_elem(char aData, Branch\*& aBranch)

{

if (!aBranch)

{

aBranch = new Branch;

aBranch->Data = aData;

aBranch->LeftBranch = 0;

aBranch->RightBranch = 0;

return;

}

else

{

if (aData < aBranch->Data) {

add\_elem(aData, aBranch->LeftBranch);

}

else if (aData > aBranch->Data) {

add\_elem(aData, aBranch->RightBranch);

}

}

}

void is\_Empty(Branch\*& aBranch)

{

if (!aBranch)

{

cout << "Дерево пустое...";

}

else

{

cout << "Дерево не пустое...";

}

}

void FreeTree(Branch\* aBranch)

{

if (!aBranch) return;

FreeTree(aBranch->LeftBranch);

FreeTree(aBranch->RightBranch);

delete aBranch;

return;

}

Branch\* del\_elem(Branch\*& aBranch, char aData)

{

if (aBranch == NULL)

return aBranch;

if (aData == aBranch->Data)

{

Branch\* tmp;

if (aBranch->RightBranch == NULL)

tmp = aBranch->LeftBranch;

else

{

Branch\* ptr = aBranch->RightBranch;

if (ptr->LeftBranch == NULL)

{

ptr->LeftBranch = aBranch->LeftBranch;

tmp = ptr;

}

else

{

Branch\* pmin = ptr->LeftBranch;

while (pmin->LeftBranch != NULL) {

ptr = pmin;

pmin = ptr->LeftBranch;

}

ptr->LeftBranch = pmin->RightBranch;

pmin->LeftBranch = aBranch->LeftBranch;

pmin->RightBranch = aBranch->RightBranch;

tmp = pmin;

}

}

delete aBranch;

return tmp;

}

else if (aData < aBranch->Data)

aBranch->LeftBranch = del\_elem(aBranch->LeftBranch, aData);

else

aBranch->RightBranch = del\_elem(aBranch->RightBranch, aData);

return aBranch;

}

int main()

{

setlocale(LC\_ALL, "rus");

char key;

Branch\* Root = 0;

int vel;

char element;

char elem;

cout << "Введите кол-во элементов для будущего дерева: ";

cin >> vel;

cout << endl;

cout << "Проверим дерево на пустоту до его заполнения: " << endl;

is\_Empty(Root);

cout << endl;

for (int i = 0; i < vel; i++)

{

cin >> elem;

Add(elem, Root);

}

cout << "Проверим дерево на пустоту после его заполнения: " << endl;

is\_Empty(Root);

cout << endl;

cout << "Вывод бинарного дерева: " << endl;

print(Root);

cout << endl;

cout << "Прямой обход бинарного дерева: " << endl;

pr\_obh(Root);

cout << endl;

cout << "Добавим новый элемент в бинарное дерево:" << endl;

cout << "Введите новый элемент: ";

cin >> element;

add\_elem(element, Root);

cout << "Вывод бинарного дерева: " << endl;

print(Root);

cout << endl;

cout << "Удалим элемент из бинарного дерева:" << endl;

cout << "Введите элемент: ";

cin >> element;

del\_elem(Root, element);

cout << "Вывод бинарного дерева: " << endl;

print(Root);

cout << endl;

cout << "Введите ключ:" << endl;

cin >> key;

key\_counter(Root, key);

cout << "Посчитаем кол-во подходящих: " << key\_count << endl;

FreeTree(Root);

cout << "Вся динамическая память очищена..." << endl;

return 0;

}

Результат работы

