Министерство образования Республики Беларусь

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Кафедра микро- и наноэлектроники

**Лабораторная работа №5**

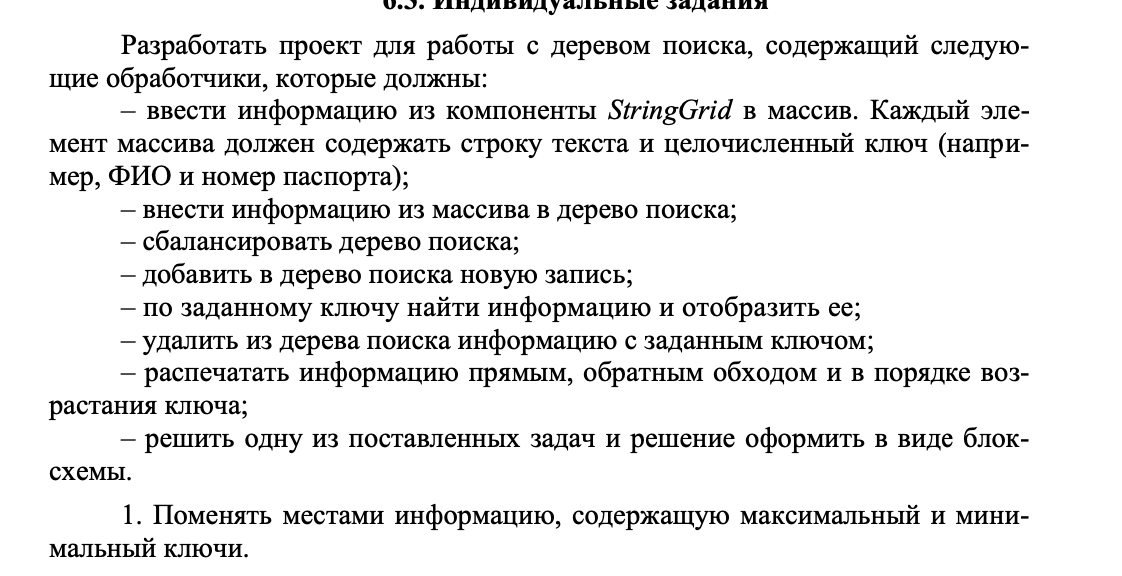
**«Нелинейные списки(Деревья) »**

**Вариант 1**

|  |  |
| --- | --- |
| Выполнил: | студент группы 142701  Ахремчик Никита Андреевич |
| Проверил: |  |

Минск 2022

**Задачи:** изучить алгоритмы обработки данных с использованием нелинейных структур в виде дерева.

**Индивидуальное задание**

Текст программы:

#**include** <iostream>

#**include** <stdio.h>

#**include** <stdlib.h>

#**include** <string.h>

**struct** tNode{

**int** value;

tNode \*left, \*right;

} \*tree;

tNode\* **createTree**(**int** key){

tNode \*tree = **new** tNode;

tree->value = key;

tree->right = tree->left = nullptr;

**return** tree;

}

// Добавление листа в дерево

**void** **addNode**(tNode \*\*tree, **int** key){

**if**((\*tree)==nullptr){

(\*tree) = **new** tNode;

(\*tree)->value = key;

(\*tree)->left=(\*tree)->right=NULL;

**return**;

}

**if**(key > (\*tree)->value) addNode(&(\*tree)->right, key);

**else** addNode(&(\*tree)->left, key);

}

tNode\* **minKey**(tNode \*tree) {

**while** (tree->left != NULL) tree = tree->left;

**return** tree;

}

tNode\* **maxKey**(tNode \*tree){

**while** (tree->right != NULL) tree = tree->right;

**return** tree;

}

// Поиск элемента в дереве по значению ключа

// Возвращает указатель на найденный элемент или NULL

tNode \***findElem**(tNode \*tree, **int** key){

tNode \*t = tree;

**while**(t){

**if** (key == t->value) **break**;

**if** (key < t->value) t = t->left;

**else** t = t->right;

}

**return** t;

}

// Вывод дерева методом обхода по возрастанию

**void** **printGrowth**(tNode \*tree, **int** level){

**if** (tree == nullptr) **return**;

printGrowth(tree->left, ++level); // левое поддерево

**for**(**int** i=0; i<level; ++i) std::cout << "|";

std::cout << tree->value << std::endl;

level--;

printGrowth(tree->right, ++level); // правое поддерево

}

// Вывод дерева методом обратного обхода

**void** **printBack**(tNode \*tree, **int** level){

**if**(tree == nullptr) **return**;

printBack(tree->left, ++level); // левое поддерево

printBack(tree->right, ++level); // правое поддерево

**for**(**int** i=0; i<level; ++i) std::cout << "|";

std::cout << tree->value << std::endl;

level--;

}

// Вывод дерева методом прямого обхода

**void** **printStraight**(tNode \*tree, **int** level){

**if**(tree == nullptr) **return**;

**for**(**int** i=0; i<level; ++i) std::cout << "|";

std::cout << tree->value << std::endl;

level--;

printStraight(tree->left, ++level); // левое поддерево

printStraight(tree->right, ++level); // правое поддерево

}

tNode \***reconstruction**(tNode \*tree){

**if** (tree->left == 0){

tNode \*right = tree->right;

\*tree = \*right;

right->left = 0;

right->right = 0;

**delete** right;

}

**else**{

tNode \*c = tree->left;

tNode \*t = tree->left;

**while**(c->right){

t = c;

c = c->right;

}

t->right = c->left;

tree->value = c->value;

**delete** c;

}

**return** tree;

}

// Удаление элемента по заданному ключу

tNode \***remove**(tNode \*tree, **int** key){

**if** (tree == nullptr) **return** 0;

**if**(tree->value < key){

tree->right = remove(tree->right, key);

**return** tree;

}

**else**{

**if**(key < tree->value){

tree->left = remove(tree->left, key);

**return** tree;

}

}

**return** reconstruction(tree);

}

**void** **print**(tNode \*&root, **int** space=0){

**if** (!root)

**return**;

**enum**{

COUNT = 2

};

space += COUNT;

print(root->right, space);

**for** (**int** i = COUNT; i < space; ++i)

std::cout << " ";

std::cout << root->value << std::endl;

print(root->left, space);

}

// Построение сбалансированного дерева по упорядоченному по возрастанию массиву

**void** **makeBalance**(tNode \*\*tree, **int** left, **int** right, **int** \*arr){

**if** (left == right){

\*tree = nullptr;

**return**;

}

**else**{

**int** min = (left+right)/2;

\*tree = **new** tNode;

(\*tree)->value = arr[min];

makeBalance(&(\*tree)->left, left, min, arr);

makeBalance(&(\*tree)->right, min+1, right, arr);

}

}

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

tNode \*tree = nullptr;

**int** n; // количество элементов

**int** s; // число, передаваемое в дерево

printf("Введите количество элементов: ");

std::cin >> n;

**for**(**int** i=0; i<n; i++){

printf("Введите число: ");

std::cin >> s;

addNode(&tree, s);

}

std::cout << "Ваше дерево\n";

print(tree, 0);

puts("");

tNode \*maxt=nullptr, \*mint=nullptr;

maxt = maxKey(tree);

mint = minKey(tree);

**int** temp;

temp = maxt->value;

maxt->value = mint->value;

mint->value = temp;

std::cout << "Ваше новое дерево\n";

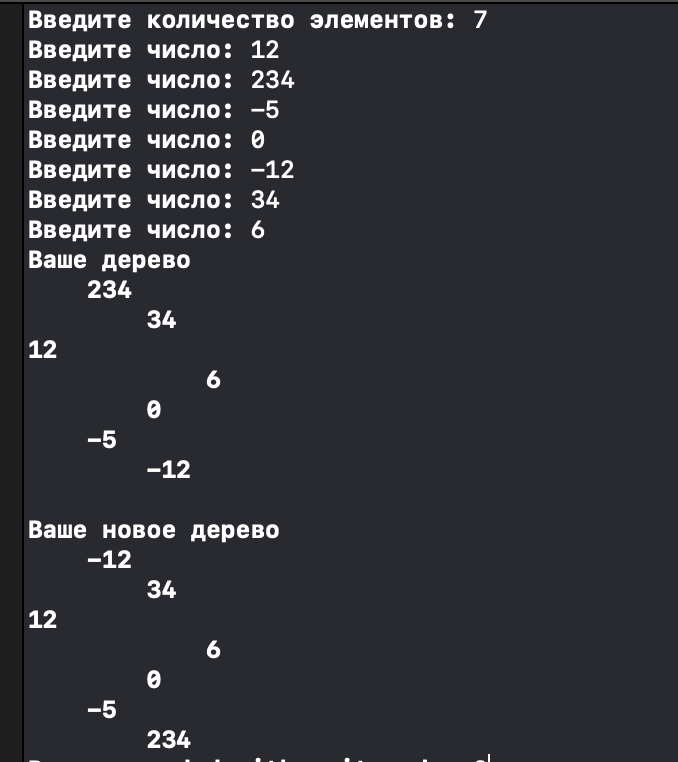
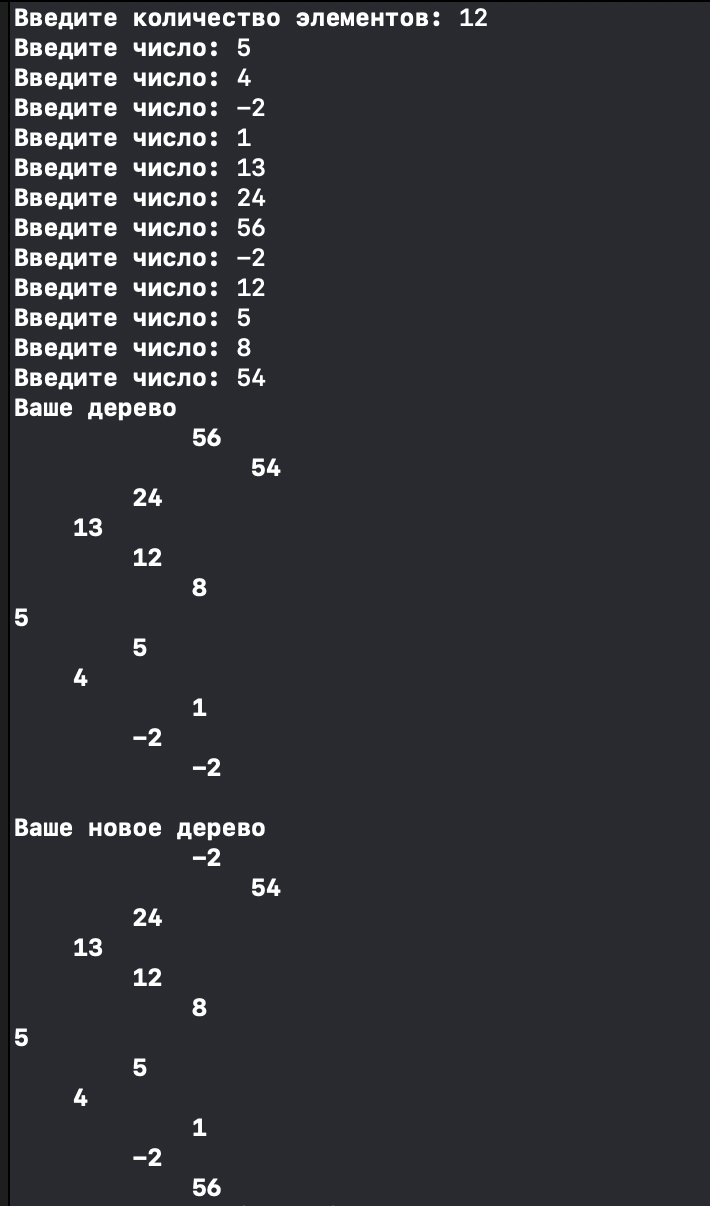
print(tree, 0);

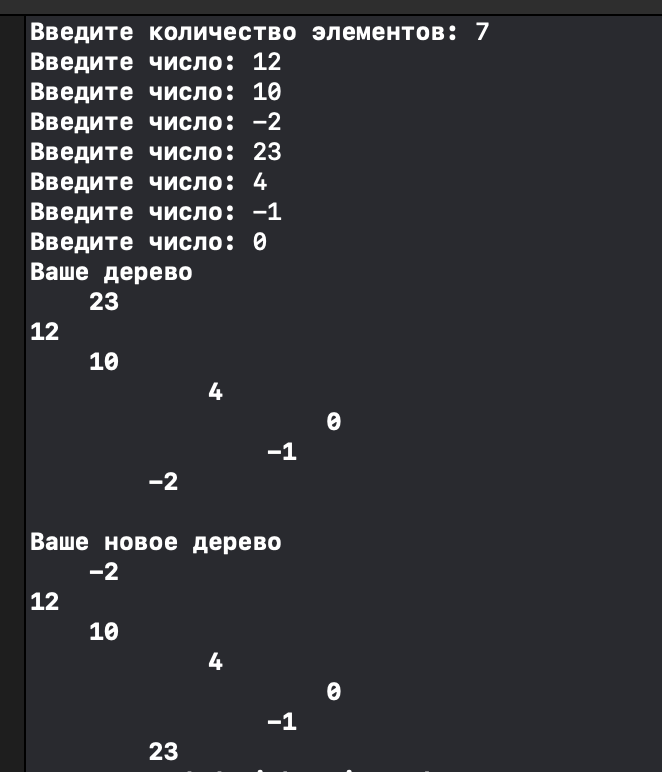
**delete**[] tree;

**return** 0;

}

**Результат вывода программы:**





**Выводы:** Проведя данную лабораторную работу, я научился работать с использованием нелинейные списки(Деревья) в С++. Также закрепил знания по прошлым темам.