МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ

Федеральное государственное бюджетное образовательное учреждение

Высшего профессионального образования

**«Вятский государственный университет»**

**(ФГБОУ ВПО «ВятГУ»)**

Факультет автоматики и вычислительной техники

Кафедра электронных вычислительных машин

Работа с динамической памятью

Отчет по лабораторной работе №2 дисциплины

«Технологии программирования»

Выполнил студент группы ИВТб-21\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/Седов М.Д./

Проверил доцент \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/Долженкова М.Л./

Киров 2019

**Цель работы**: получить базовые навыки работы с динамической памятью, получить навыки работы с отладчиком

**Задание:**

1) Написать программу, реализующую бинарное дерево.

2) Бинарное дерево должно осуществлять поиск по ключу, вставку нового элемента в дерево, удаление.

3) Вершина дерева содержит структуру из целого числа и массива типа double.

4) Вставка и удаление элементов осуществляется через операторы new, delete.

**Работа программы:**

1. Занесение элемента



* Индикатор начала – 0816 = 0000 0**1**00 (Выделенный бит – флаг занятости участка).
* Указатель на предыдущий занятый элемент.
* Указатель на следующий занятый элемент.
* Указатель на имя файла подкачки.
* Номер строки в файле подкачки.
* Тип участка памяти. (01 – пользовательский тип)
* Размер выделенного участка памяти.
* Количество обращений.
* Индикатор начала пользовательского участка памяти.
* Пользовательская структура из 20 байт.

1. Число типа int.
2. Указатель на массив типа double.
3. Размер массива типа int
4. Указатель на левый потомок.
5. Указатель на правый потом.

* Индикаторы конца пользовательского участка памяти.

1. Удаление элемента



Произошло освобождение области памяти после удаления элемента из дерева, флаг занятости сменился на 0 - 0016 = 0000 0**0**00

**Исходный текст программы:**

**“Tree.h”**

#pragma once

struct TreeNode {

int value;

double \*array\_d;

int size;

TreeNode \*left;

TreeNode \*right;

};

class Tree

{

public:

Tree();

~Tree();

void insert(int key, double \*&arr, int size);

TreeNode \*search(int key);

void delete\_tree();

void print();

TreeNode \*get\_root() { return root; }

private:

void insert(int key, double \*arr, int size, TreeNode \*&leaf);

TreeNode \*search(int key, TreeNode \*leaf);

void delete\_tree(TreeNode \*&leaf);

TreeNode \*root;

void print(TreeNode \*leaf);

};

**“Tree.cpp”**

#include "Tree.h"

#include <iostream>

#include <queue>

using std::cout;

using std::queue;

Tree::Tree()

{

root = nullptr;

}

Tree::~Tree()

{

delete\_tree();

}

void Tree::insert(int key, double \*&arr, int size)

{

if (root != nullptr) insert(key, arr, size, root);

else {

root = new TreeNode;

root->value = key;

root->array\_d = arr;

root->size = size;

root->left = nullptr;

root->right = nullptr;

}

}

TreeNode \* Tree::search(int key)

{

return search(key, root);

}

void Tree::delete\_tree()

{

delete\_tree(root);

}

void Tree::print()

{

print(root);

}

void Tree::insert(int key, double \*arr, int size, TreeNode \*&leaf)

{

if (key < leaf->value) {

if (leaf->left != nullptr) insert(key, arr, size, leaf->left);

else {

leaf->left = new TreeNode;

leaf->left->value = key;

leaf->left->array\_d = arr;

leaf->left->size = size;

leaf->left->left = nullptr;

leaf->left->right = nullptr;

}

}

else if (key >= leaf->value) {

if (leaf->right != nullptr) insert(key, arr, size, leaf->right);

else {

leaf->right = new TreeNode;

leaf->right->value = key;

leaf->right->array\_d = arr;

leaf->right->size = size;

leaf->right->left = nullptr;

leaf->right->right = nullptr;

}

}

}

TreeNode \* Tree::search(int key, TreeNode \*leaf)

{

if (leaf != nullptr) {

if (key == leaf->value) return leaf;

if (key < leaf->value)

return search(key, leaf->left);

else

return search(key, leaf->right);

}

return nullptr;

}

void Tree::delete\_tree(TreeNode \*&leaf)

{

if (leaf != nullptr) {

delete\_tree(leaf->left);

delete\_tree(leaf->right);

delete leaf;

leaf = nullptr;

}

}

void Tree::print(TreeNode \* leaf)

{

if (leaf == nullptr) return;

print(leaf->left);

cout << leaf->value << " ";

print(leaf->right);

}

**“source.cpp”**

#include "Tree.h"

#include <iostream>

#include <conio.h>

#include <string>

#include <Windows.h>

using namespace std;

const int NUMBER\_FIELD = 5;

void print\_menu(string\* menu, int n, int pos) {

system("cls");

HANDLE handle = GetStdHandle(STD\_OUTPUT\_HANDLE);

SetConsoleTextAttribute(handle, FOREGROUND\_GREEN);

for (int i = 0; i < pos; i++) cout << menu[i] << endl;

SetConsoleTextAttribute(handle, FOREGROUND\_RED);

cout << menu[pos] << endl;

SetConsoleTextAttribute(handle, FOREGROUND\_GREEN);

for (int i = pos + 1; i < n; i++) cout << menu[i] << endl;

}

int main() {

int pos = 0;

string menu[NUMBER\_FIELD] = { "1 - Add node in the tree", "2 - Search select node in the tree",

"3 - Delete tree", "4 - Print tree", "5 - Exit"};

print\_menu(menu, NUMBER\_FIELD, pos);

Tree \*t = new Tree;

char ch;

int code, temp;

double \*arr;

bool b;

do {

ch = getch();

code = static\_cast<int> (ch);

switch (code)

{

case 80:

if (pos < NUMBER\_FIELD - 1) {

++pos;

print\_menu(menu, NUMBER\_FIELD, pos);

}

break;

case 72:

if (pos > 0) {

--pos;

print\_menu(menu, NUMBER\_FIELD, pos);

}

break;

case 13:

system("cls");

switch (pos)

{

case 0:

cout << "Enter value node: ";

int n;

cin >> temp;

cout << "Enter size the array: ";

cin >> n;

arr = new double[n];

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

cin >> arr[i];

t->insert(temp, arr, n);

break;

case 1:

cout << "Enter key-value node: ";

cin >> temp;

TreeNode \*res;

res = t->search(temp);

if (res == nullptr)

cout << "Node there is no in tree with current key-value";

else {

cout << "Node there is in tree with current key-value\n";

cout << "Value: " << res->value << endl;

cout << "Array: ";

for (int i = 0; i < res->size; i++)

cout << res->array\_d[i] << " ";

}

getch();

break;

case 2:

t->delete\_tree();

break;

case 3:

if (t->get\_root() != nullptr)

t->print();

else

cout << "Tree is free";

getch();

break;

case 4:

return 0;

break;

default:

break;

}

print\_menu(menu, NUMBER\_FIELD, pos);

break;

default:

break;

}

} while (code != 27);

getch();

return 0;

}

**Вывод**

В ходе данной лабораторной работы были освоены базовые возможности для работы с динамической памятью. Так же были получены навыки работы с отладчиком Visual Studio, в котором устанавливались точки останова в необходимых местах для просмотра размещения элементов в памяти компьютера. В лабораторной работе использовались операторы языка C++, которые дают возможность работы с памятью, посредством new и delete. Был построен класс, который отвечал за реализацию дерева. В классе было одно поле root (корень дерева), и необходимые методы: вставка нового элемента в дерево (insert), удаление дерева (delete\_tree), поиск в дерево элемента с заданным ключом (search).