**Задание 1**.(факториал двумя способами).

#include <iostream>

using namespace std;

long double fact(int N)

{

if(N < 0) // если пользователь ввел отрицательное число

return 0; // возвращаем ноль

if (N == 0) // если пользователь ввел ноль,

return 1; // возвращаем факториал от нуля - не удивляетесь, но это 1 =)

else // Во всех остальных случаях

return N \* fact(N - 1); // делаем рекурсию.

}

int main()

{

int N;

int fact2=1;

setlocale(0,""); // Включаем кириллицу

cout << "Введите число для вычисления факториала: ";

cin >> N;

cout << "Факториал для числа " << N << " = " << fact(N) << endl << endl;

for (int i = 1; i <= N; i++) {

fact2 = fact2 \* i;

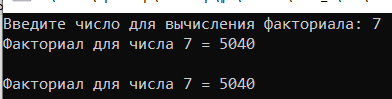
}

cout << "Факториал для числа " << N << " = " << fact2 << endl << endl;

return 0;

}

Результат работы:



**Задание 2**(способы избежать ошибок при бесконечном цикле или бесконечной рекурсии).

Обычно бесконечный цикл или бесконечная рекурсия вызваны намеренно, а выход из них предусмотрен какой-нибудь частью кода на подобии:

While(true){

Cin<<ext;

If(ext ==1){return 0;}

//какой-то код

}

Если не требуется заканчивать работу программы, то может стоять exit(1).

Также бесконечный цикл/рекурсия могут быть остановлены по достижению теми данными, что используются в цикли каких-то конкретных значений, в этом случае не нужно вводить ничего с клавиатуры, цикл будет работать, как цикл с параметром, то же касается и рекурсии.

**Задание 3**- функции из лекции и сделать свою функцию удаления

**Тело программы:**

#include <iostream>

#include "lb19\_3.h"

using namespace std;

int main() {

setlocale(0,"");

int b[] = {1, 2, 3, 4, 5, 6, 7, 8,9,10,11,12,13,14,15};

Node \*root = first(b[2]);

for (int i=0;i< 15;i++) {

search\_insert(root, b[i]);

}

print\_tree(root, 0);

DeleteNode(root,8);

cout << endl;

print\_tree(root, 0);

cout << endl;

return 0;

}

**Заголовочный файл:**

#ifndef LB19\_3\_H\_INCLUDED

#define LB19\_3\_H\_INCLUDED

struct Node {

int d;

Node \*left;

Node \*right;

};

Node \*first(int d);

Node \*search\_insert(Node \*root, int d);

void print\_tree(Node \*p, int level);

Node \*DeleteNode(Node\* node, int val);

#endif // LB19\_3\_H\_INCLUDED

**Функция удаления элемента:**

#include <iostream>

#include "lb19\_3.h"

using namespace std;

Node \*DeleteNode(Node\* node, int d){

if(node == NULL)

return node;

if(d == node->d){

Node\* tmp;

if(node->right == NULL)

tmp = node->left;

else {

Node\* ptr = node->right;

if(ptr->left == NULL){

ptr->left = node->left;

tmp = ptr;

} else {

Node\* pmin = ptr->left;

while(pmin->left != NULL){

ptr = pmin;

pmin = ptr->left;

}

ptr->left = pmin->right;

pmin->left = node->left;

pmin->right = node->right;

tmp = pmin;

}

}

delete node;

return tmp;

} else if(d < node->d)

node->left = DeleteNode(node->left, d);

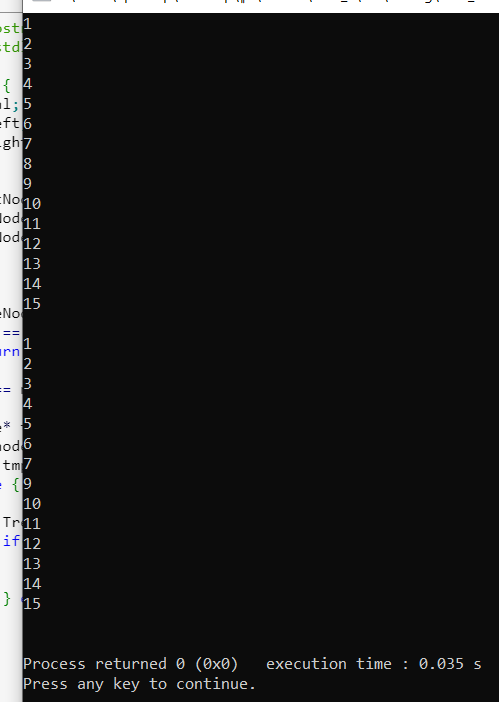
else

node->right = DeleteNode(node->right, d);

return node;

}

**Результат работы:**



Итак, ввелись числа от 1 до 15, потом вывелись потом было удалено число 8 (удалялся элемент 8 если он есть, а он был), функция сработала, также вывод был осуществлен в столбик, это потому, что я убрал пробел из функции **print\_tree,** которую скопировал из лекции.

**Задание 4.** (как бы можно было дополнить, изменить 16 лабораторную работу с использованием динамической структуры данных).

Во-первых, динамическая структура занимала бы меньше памяти (в 16 лабе я создавал 256 элементов структуры просто так, то есть занимал кучу памяти, да ещё и в стеке)

Во-вторых, скорее всего было бы проще прописать удаление элемента, так как для этого хватило бы пары строчек, просто для отчистки памяти.

**Задание 5.** (словарь)

**Реализация:**

#include <iostream>

#include <cstring>

using namespace std;

#define WORD\_SIZE 256

struct Word {

char rus[WORD\_SIZE];

char eng[WORD\_SIZE];

int counter;

};

struct Node {

Word d;

Node \*left;

Node \*right;

};

inline void print\_word(Word d) {

cout << "ru: " << d.rus << " en: " << d.eng << " counter: " << d.counter;

}

void translate(Node \*root,char w[256]){

if (root) {

translate(root->left,w);

if (strcmp((root->d.eng),w)==0){

root->d.counter++;

cout<<"Perevod: "<<(root->d.rus)<<endl;

}

translate(root->right,w);

}

}

bool word\_compare(Word d1, Word d2, bool sort\_by\_counter = false) {

if (sort\_by\_counter) {

return d1.counter < d2.counter;

} else {

for (int i = 0; i < WORD\_SIZE; i++) {

if (d1.eng[i] != d2.eng[i]) {

return d1.eng[i] < d2.eng[i];

}

}

return false;

}

}

Node \*first(Word d) {

Node \*pv = new Node;

pv->d = d;

pv->left = nullptr;

pv->right = nullptr;

return pv;

}

Node \*search\_insert(Node \*root, Word d, bool sort\_by\_counter = false) {

Node \*pv = root;

Node \*prev;

bool found = false;

while (pv && !found) {

prev = pv;

if (d.eng == pv->d.eng) found = true;

else if (word\_compare(d, pv->d, sort\_by\_counter)) pv = pv->left;

else pv = pv->right;

}

if (found) return pv;

Node \*pnew = new Node;

pnew->d = d;

pnew->left = nullptr;

pnew->right = nullptr;

if (word\_compare(d, prev->d, sort\_by\_counter))

prev->left = pnew;

else

prev->right = pnew;

return pnew;

}

void print\_tree(Node \*p, int level) {

if (p) {

print\_tree(p->left, level + 1);

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

print\_word(p->d);

cout << endl;

print\_tree(p->right, level + 1);

}

}

Node \*rebuild\_tree(Node \*p, Node \*s) {

if (p != nullptr) {

Word inp2;

inp2.counter = p->d.counter;

strcpy(inp2.eng,p->d.eng);

strcpy(inp2.rus,p->d.rus);

if (s == nullptr)

s = first(inp2);

else

search\_insert(s, inp2, true);

if (p->left != nullptr) rebuild\_tree(p->left, s);

if (p->right != nullptr) rebuild\_tree(p->right, s);

}

return s;

}

void delete\_tree(Node \*p) {

if (p == nullptr) return;

if (p->right != nullptr) delete\_tree(p->right);

if (p->left != nullptr) delete\_tree(p->left);

delete p;

}

int main() {

setlocale(0,"");

int mode;

Node \*root = nullptr;

Node \*r2 = nullptr;

while (true) {

cout << "Выберете режим работы:\n"

"1. Новая пара слов\n"

"2. Перевести слово\n"

"3. Представление словаря по к-ву запросов и печать его\n"

"4. Вывод словаря по алфавиту\n"

"5. Выход из программы\n";

cin >> mode;

if (mode == 5) return 0;

switch (mode) {

case 1:

Word inp;

inp.counter = 0;

cout << "Введите английское слово: ";

cin >> inp.eng;

cout << "Введите русский перевод: ";

cin >> inp.rus;

if (root != nullptr) {

search\_insert(root, inp, false);

} else {

root = first(inp);

}

break;

case 2:

char word[WORD\_SIZE];

cout << "Введите английское слово: ";

cin >> word;

translate(root,word);

break;

case 3:

delete\_tree(r2);

r2 = nullptr;

r2 = rebuild\_tree(root, r2);

print\_tree(r2,0);

break;

case 4:

print\_tree(root,0);

break;

default:

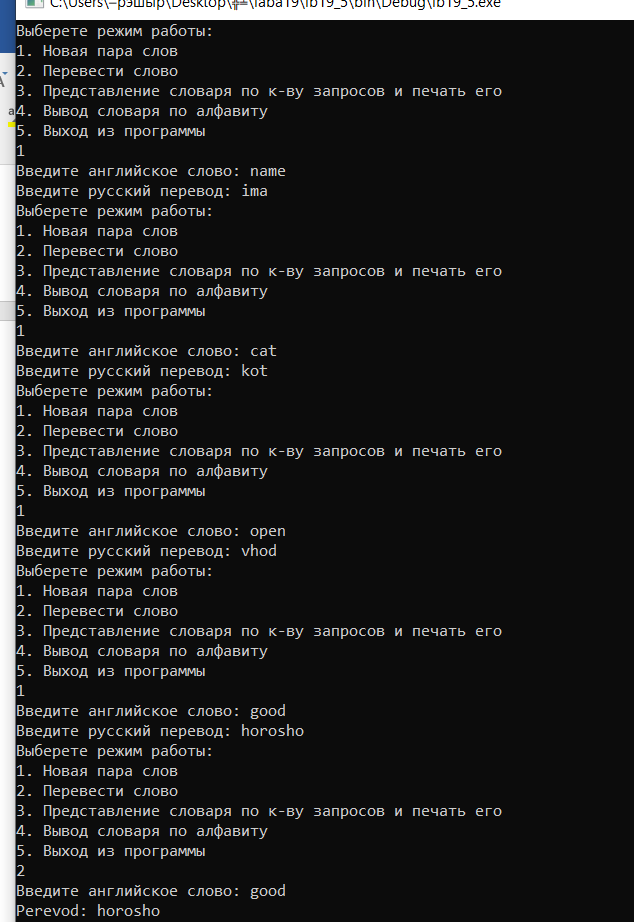
cout << "Это не число от 1 до 5!" << endl;

}

}

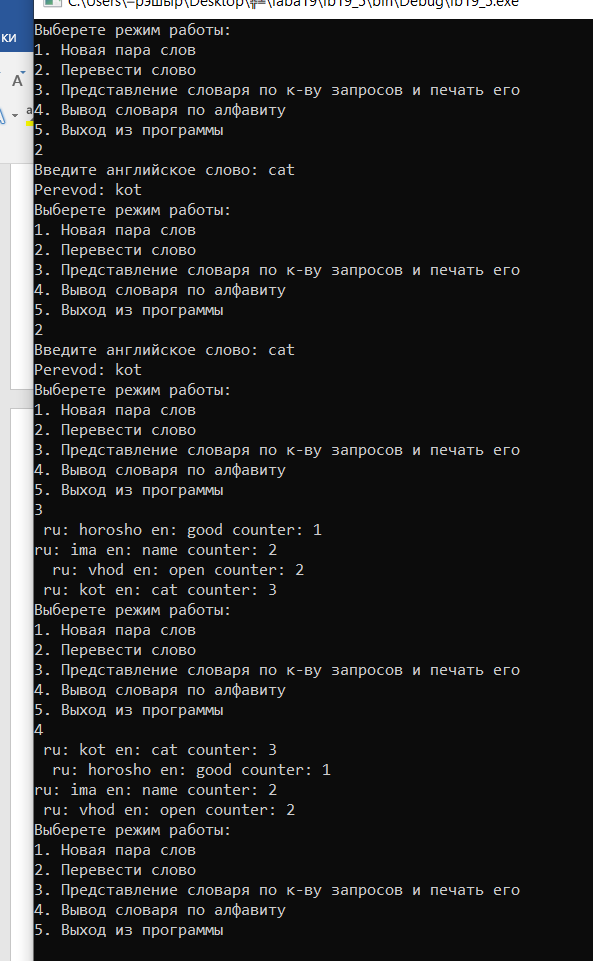
}

**Тестирование. 1 – ввод слов, потом проверка перевода одного из них.**



(после этого скриншота операция перевод вызывалась ещё несколько раз)

2(после НЕСКЛЬКИХ вызовов операции перевода для разных слов.



Как видите, я 3 раза переводил кота, 1 раз хорошо и по 2 раза 2 других слова, потом я вызвал 2 пункт меню, и он создал словарь по кол-ву запросов и вывел на экран, а потом я вызвал вывод по алфавиту и вывелся первый словарь (соответственно по алфавиту).