**кафедра комп’ютерних технологій, ДНУ**

**2016/2017 н.р.**

**ЗВІТ З ЛАБОРАТОРНОЇ РОБОТИ №1**

**за курсом «Програмування»**

**студента групи ПК-15-1**

**Гулого Тараса Олександровича**

# 1. Постановка задачі

**Постановка задачі**.

Розробити об’єктно-орієнтовану бібліотеку для роботи зі структурами даних за однією з нижченаведених тем у відповідності з нижченаведеними вимогами. Властивості та методи для класів розробити у відповідності з відомими визначеннями відповідних структур даних. Скласти тести для перевірки працездатності бібліотеки. Скласти програму, що демонструє можливості розробленої бібліотеки.

**Індивідуальне завдання №8**

Дерева довільні та генеалогія. Використання псевдозмінних для дерева може бути виконано таким чином: по-перше елементу (i, j) в дереві (і – номер рівня, j – номер елемента на j-му рівні) може бути присвоєно деяке значення, по-друге, наприклад, при d(i,j)=d2; //де d, d2 – дерева замінити в дереві d j-й елемент на і-му рівні на піддерево d2

# 2. Опис розв’язку

Дана лабораторна має досить складну структуру отже розіб’ємо описання вихідного коду програму на декілька змістовних блоків.

## 1.Фактичне визначення довільних дерев

**Де́рево** (англ. *tree*)  — в [інформатиці](https://uk.wikipedia.org/wiki/%D0%86%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0) та [програмуванні](https://uk.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F) одна з найпоширеніших [структур даних](https://uk.wikipedia.org/wiki/%D0%A1%D1%82%D1%80%D1%83%D0%BA%D1%82%D1%83%D1%80%D0%B0_%D0%B4%D0%B0%D0%BD%D0%B8%D1%85). Формально [дерево визначається](https://uk.wikipedia.org/wiki/%D0%94%D0%B5%D1%80%D0%B5%D0%B2%D0%BE_(%D1%82%D0%B5%D0%BE%D1%80%D1%96%D1%8F_%D0%B3%D1%80%D0%B0%D1%84%D1%96%D0%B2)) як скінченна множина Т з однією або більше вершин (вузлів, nodes), яка задовольняє наступним вимогам:

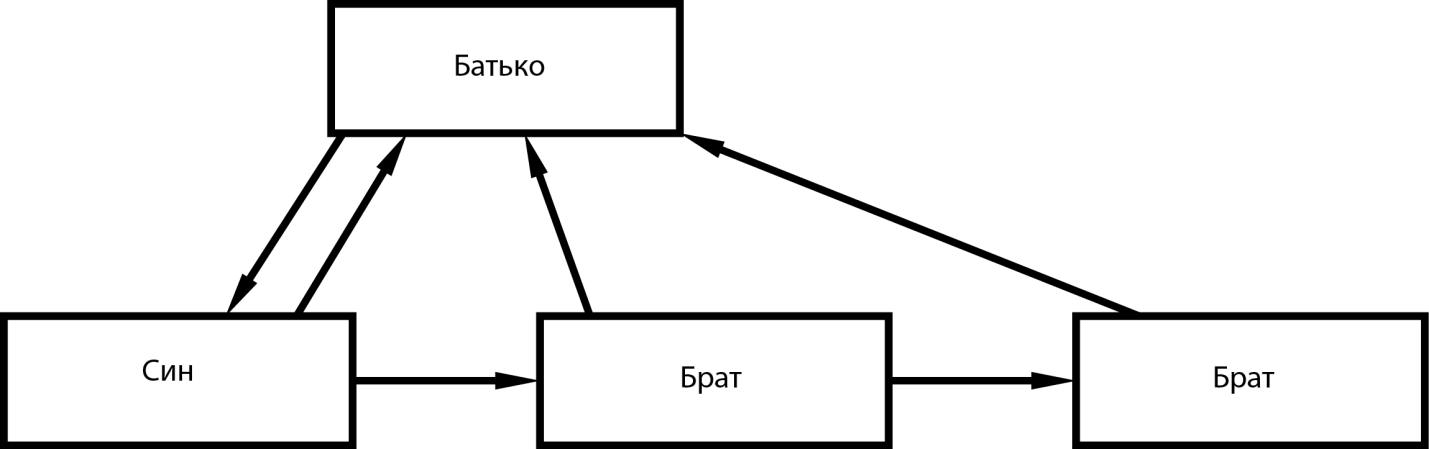
існує один відокремлений вузол — корінь (root) дерева

інші вузли (за винятком кореня) розподілені серед m ≥ 0 непересічних множин T1…Tm і кожна з цих множин, в свою чергу, є деревом. Дерева T1…Tm мають назву **піддерев** (subtrees) даного кореня.

У деревах на відміну від графів не існує циклів, кожный вузол дерева не може вказувати сам на себе.

Для реалізації довільних дерев у памяті слід розглядати модель де кожний вузол дерева має три основні зв’язки: батьківський, синовій та братерський.

Розглянемо таке дерево



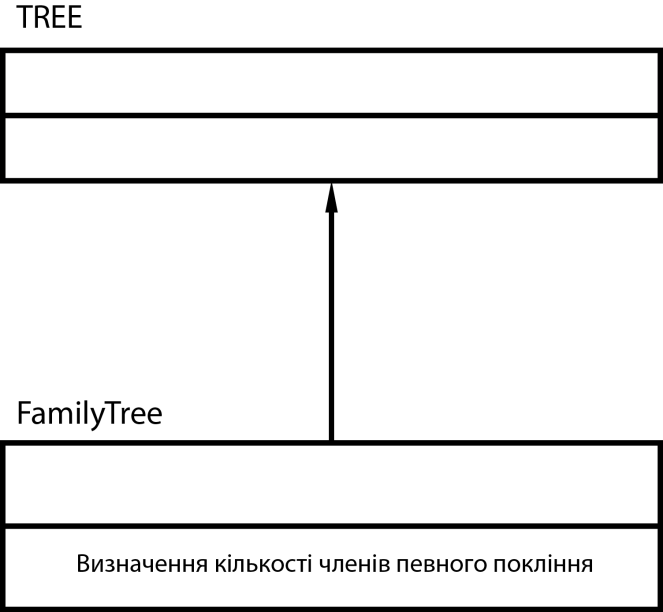
В даному дереві існує один корінь дерева та три піддерева. У кореня дерева є зв’язок лише до одного елемента – сина. Син може вказувати інші елементами, які також є піддеревами кореня і у всіх піддерев є однонаправлений зв'язок з коренем. Власне така реалізація структура дерева дає нам змогу додавати безліч піддерев до кореня без використання масивів та є зручною в реалізації.

В оперативній пам’яті зображатимемо данні зв’язки як вказівники. А дерева матимуть інформаційне поле яке буде зберігати необхідну інформацію.

Для зручності також матимемо два додаткових поля які будуть міститі рівень на якому знаходиться елемент відносно кореня та вказівник на самий корінь.

## 2.Написання класу дерев

Для створення класу дерева було запропоновано створення такої структури. Поля класу відповідають визначенню дерева яке було описано раніше. Щодо методів, їх було визначено згідно загальних вимог до лабораторної роботи за вказівок в індивідуальному завданні. Дерева мають також приховані рекурентні функції які викликаються лише функціями-членами класу.

Наслідування буде організовано згідно з застосуванням класу до певної прикладної задачі. Як визначено в індивідуальному завданні, це створення родового дерева, певного класу, який може будувати родові дерева. В цьому класі буде додатковий метод, метод підрахунку членів певного покоління.

Дана структура матиме таке ж саме меню в головній програмі але користувачу буде запропоновано заповнити таке дерево по іншому.

Класи будуть розроблені в окремих модулях. Для демонстарції наслідування буде підключено основний клас у вигляді відкомпільованого файлу.

## 3.Написання функцій до класа дерево

Відповідно до поставленої задачі було реалізовано 12 основних функцій доступних до користувача і чотири додаткові функції. Додаткові функції використовуються як рекурентні в них виконується рекурентні обходи та перетворення дерева.

В відповідності до індивідуального завдання додано перевантаження 5 операторів один з яких перевантажено фіктивно (див. зауваження).

Основною частиною доступу до елементів дерева є функція перевантаження операції круглі дужки.

В похідному класі додана ще одна функція яка реалізує визначення кількості елементів на певному рівні.

Реалізація функцій винесена в окремий cpp файл. Наслідування демонструється видаленням cpp файлу.

# 3. Текст вихідної програми

Main.cpp

#include <iostream>

#include <stdlib.h>

#include <conio.h>

#include "Tree.h"

#include "FamilyTree.h"

using namespace std;

int main()

{

Tree d1,d2;

FamilyTree fd1, fd2;

system("CHCP 1251>NUL");

cout<<"Добро пожалывать в лабораторную работу №2"<<endl;

cout<<"Данная програма реализует структуру ПРОИЗВОЛЬНОЕ ДЕРЕВО"<<endl;

while(true)

{

cout<<"ГЛАВНОЕ МЕНЮ"<<endl;

cout<<"1-Операции над деревом"<<endl;

cout<<"2-Задача «Генеология»"<<endl;

cout<<"0-Выход из программы"<<endl;

cin.clear();

\_flushall();

switch(getch())

{

case'1':{

while(true)

{

system("cls");

cin.clear();

\_flushall();

cout<<"МЕНЮ ОПЕРАЦИЙ НАД ДЕРЕВОМ"<<endl;

cout<<"1-Создание дерева"<<endl;

cout<<"2-Визуализация дерева"<<endl;

cout<<"3-Ввод данных в дерево"<<endl;

cout<<"4-Перегрузка квадратных скобочек"<<endl;

cout<<"5-Удаления ветки/дерева/елемента"<<endl;

cout<<"6-Объеденение двух деревьев"<<endl;

cout<<"7-Вывод дерева в файл"<<endl;

cout<<"8-Ввод дерево из файла"<<endl;

cout<<"0-Выход из этого меню"<<endl;

switch(getch())

{

case '1':{system("cls");

cout<<"Выберете дерево"<<endl;

switch(getch())

{

case '1':{system("cls"); cout<<"Выберано дерево 1"<<endl; cin>>\*(d1.set(&d1)); break;}

case '2':{system("cls"); cout<<"Выберано дерево 2"<<endl; cin>>\*(d2.set(&d2)); break;}

}

getch();

continue;}

case '2':{system("cls");

cout<<"Ввыберете дерево"<<endl;

switch(getch())

{

case '1':{

system("cls");

cout<<"Выберано дерево 1"<<endl;

if(&d1(0,0)==NULL){cout<<"Вы не создали дерево!"<<endl; break;}

d1.visual(&d1);

break;}

case '2':{

system("cls");

cout<<"Выберано дерево 2"<<endl;

if(&d2(0,0)==NULL){cout<<"Вы не создали дерево!"<<endl; break;}

d2.visual(&d2);

break;}

}

getch();

continue;}

case '3':{system("cls");

cout<<"Выберете дерево"<<endl;

switch(getch())

{

case '1':

{

system("cls");

cout<<"Выберано дерево 1"<<endl;

if(&d1(0,0)==NULL){cout<<"Вы не создали дерево!"<<endl;getch();break;}

int i,j;

cout<<"Введите уровень и номер елемента на нем"<<endl;

cin>>i>>j;

if(&d1(i,j)==NULL){cout<<"Елемент не найден"<<endl;getch();break;}

cin>>\*(d1.set(&d1(i,j)));

break;

}

case '2':

{

system("cls");

cout<<"Выберано дерево 2"<<endl;

if(&d2(0,0)==NULL){cout<<"Вы не создали дерево!"<<endl;getch();break;}

int i,j;

cout<<"Введите уровень и номер елемента на нем"<<endl;

cin>>i>>j;

if(&d2(i,j)==NULL){cout<<"Елемент не найден"<<endl;getch();break;}

cin>>\*(d2.set(&d2(i,j)));

break;

}

}

getch();

continue;}

case '4':{

system("cls");

cout<<"Выберете дерево"<<endl;

switch(getch())

{

case '1':

{

system("cls");

cout<<"Выберано дерево 2"<<endl;

if(&fd1(0,0)==NULL){cout<<"Вы не создали дерево!"<<endl;getch();break;}

int i;

cout<<"Введите уровень "<<endl;

cin>>i;

fd1[i];

break;

}

case '2':

{

system("cls");

cout<<"Выберано дерево 2"<<endl;

if(&fd2(0,0)==NULL){cout<<"Вы не создали дерево!"<<endl;getch();break;}

int i;

cout<<"Введите уровень "<<endl;

cin>>i;

fd2[i];

break;

}

}

getch();

continue;}

case '5':{system("cls");

cout<<"Выберете дерево"<<endl;

switch(getch())

{

case '1':

{

system("cls");

cout<<"Выберано дерево 1"<<endl;

if(&d1(0,0)==NULL){cout<<"Вы не создали дерево!"<<endl;getch();break;}

int i,j;

cout<<"Введите уровень и номер елемента на нем"<<endl;

cin>>i>>j;

if(&d1(i,j)==NULL){cout<<"Елемент не найден"<<endl;getch();break;}

d1.remove(&d1(i,j));

}

case '2':

{

system("cls");

cout<<"Выберано дерево 2"<<endl;

if(&d2(0,0)==NULL){cout<<"Вы не создали дерево!"<<endl;getch();break;}

int i,j;

cout<<"Введите уровень и номер елемента на нем"<<endl;

cin>>i>>j;

if(&d2(i,j)==NULL){cout<<"Елемент не найден"<<endl;getch();break;}

d2.remove(&d1(i,j));

}

}

getch();

continue;}

case '6':

{

cout<<"Выберете дерево"<<endl;

switch(getch())

{

case '1':

{

system("cls");

cout<<"Выберано дерево 1"<<endl;

if(&d1(0,0)==NULL){cout<<"Вы не создали дерево-источник!!"<<endl;getch();break;}

int i,j;

cout<<"Введите уровень и номер елемента на нем"<<endl;

cin>>i>>j;

if(&d1(i,j)==NULL){cout<<"Елемент не найден"<<endl;getch();break;}

d2=d1(i,j);

d2.visual(&d2);

}

case '2':

{

system("cls");

cout<<"Выберано дерево 2"<<endl;

if(&d1(0,0)==NULL){cout<<"Вы не создали дерево-источник!"<<endl;getch();break;}

int i,j;

cout<<"Введите уровень и номер елемента на нем"<<endl;

cin>>i>>j;

if(&d1(i,j)==NULL){cout<<"Елемент не найден"<<endl;getch();break;}

d1=d2(i,j);

d2.visual(&d2);

}

}

getch();

continue;}

case '7':

{

system("cls");

cout<<"Выберете дерево"<<endl;

switch(getch())

{

case '1':

{

system("cls");

char name[256];

cout<<"Выберано дерево 1"<<endl;

cout<<"Введите имя файла"<<endl;

cin>>name;

d1.filein(name);

break;

}

case '2':

{

system("cls");

char name[256];

cout<<"Выберано дерево 2"<<endl;

cout<<"Введите имя файла"<<endl;

cin>>name;

d2.filein(name);

break;

}

}

getch();

continue;

}

case '8':{

system("cls");

cout<<"Выберете дерево"<<endl;

switch(getch())

{

case '1':

{

system("cls");

char name[256];

cout<<"Выберано дерево 1"<<endl;

cout<<"Введите имя файла"<<endl;

cin>>name;

d1.fileout(name);

break;

}

case '2':

{

system("cls");

char name[256];

cout<<"Выберано дерево 2"<<endl;

cout<<"Введите имя файла"<<endl;

cin>>name;

d2.fileout(name);

break;

}

}

getch();

continue;

}

case '0':{break;}

default:{ continue;}

}

break;

}

break;

}

case'2':{

while(true)

{

system("cls");

cin.clear();

\_flushall();

cout<<"МЕНЮ ЗАДАЧИ «ГЕНЕОЛОГИЯ»"<<endl;

cout<<"1-Создание генеологического дерева"<<endl;

cout<<"2-Визуализация генеологического дерева"<<endl;

cout<<"3-Ввод родственников в дерево"<<endl;

cout<<"4-Определение количества родственников на опредленном уровне"<<endl;

cout<<"5-Вывод генеологического дерева"<<endl;

cout<<"6-Ввод из файла"<<endl;

cout<<"7-Вывод в файл"<<endl;

cout<<"0-Выход из этого меню"<<endl;

switch(getch())

{

case '1':{system("cls");

cout<<"Выберете дерево"<<endl;

switch(getch())

{

case '1':{system("cls"); cout<<"Выберано дерево 1"<<endl;cout<<"Введите главного члена"<<endl; cin>>\*(fd1.set(&fd1)); break;}

case '2':{system("cls"); cout<<"Выберано дерево 2"<<endl;cout<<"Введите главного члена"<<endl; cin>>\*(fd2.set(&fd2)); break;}

}

getch();

continue;}

case '2':{system("cls");

cout<<"Ввыберете дерево"<<endl;

switch(getch())

{

case '1':{

system("cls");

cout<<"Выберано дерево 1"<<endl;

if(&fd1(0,0)==NULL){cout<<"Вы не создали дерево!"<<endl; break;}

fd1.visual(&fd1);

break;}

case '2':{

system("cls");

cout<<"Выберано дерево 2"<<endl;

if(&fd2(0,0)==NULL){cout<<"Вы не создали дерево!"<<endl; break;}

fd2.visual(&fd2);

break;}

}

getch();

continue;}

case '3':{system("cls");

cout<<"Выберете дерево"<<endl;

switch(getch())

{

case '1':

{

system("cls");

cout<<"Выберано дерево 1"<<endl;

if(&fd1(0,0)==NULL){cout<<"Вы не создали дерево!"<<endl;getch();break;}

int i,j;

cout<<"Введите уровень и номер елемента на нем"<<endl;

cin>>i>>j;

if(&fd1(i,j)==NULL){cout<<"Елемент не найден"<<endl;getch();break;}

cout<<"Введите члена родового дерева"<<endl;

cin>>\*(fd1.set(&fd1(i,j)));

break;

}

case '2':

{

system("cls");

cout<<"Выберано дерево 2"<<endl;

if(&fd2(0,0)==NULL){cout<<"Вы не создали дерево!"<<endl;getch();break;}

int i,j;

cout<<"Введите уровень и номер елемента на нем"<<endl;

cin>>i>>j;

if(&fd2(i,j)==NULL){cout<<"Елемент не найден"<<endl;getch();break;}

cout<<"Введите члена родового дерева"<<endl;

cin>>\*(fd2.set(&fd2(i,j)));

break;

}

}

getch();

continue;}

case '4':{

system("cls");

cout<<"Выберете дерево"<<endl;

switch(getch())

{

case '1':

{

system("cls");

cout<<"Выберано дерево 2"<<endl;

if(&fd1(0,0)==NULL){cout<<"Вы не создали дерево!"<<endl;getch();break;}

int i;

cout<<"Введите уровень "<<endl;

cin>>i;

fd1.myhasband(i);

break;

}

case '2':

{

system("cls");

cout<<"Выберано дерево 2"<<endl;

if(&fd2(0,0)==NULL){cout<<"Вы не создали дерево!"<<endl;getch();break;}

int i;

cout<<"Введите уровень "<<endl;

cin>>i;

fd2.myhasband(i);

break;

}

}

getch();

continue;

}

case '5':

{

system("cls");

cout<<"Выберете дерево"<<endl;

switch(getch())

{

case '1':

{

system("cls");

cout<<"Выберано дерево 1"<<endl;

if(&fd1(0,0)==NULL){cout<<"Вы не создали дерево!"<<endl;getch();break;}

int i,j;

cout<<"Введите уровень и номер елемента на нем"<<endl;

cin>>i>>j;

if(&fd1(i,j)==NULL){cout<<"Елемент не найден"<<endl;getch();break;}

fd1.remove(&fd1(i,j));

}

case '2':

{

system("cls");

cout<<"Выберано дерево 2"<<endl;

if(&fd2(0,0)==NULL){cout<<"Вы не создали дерево!"<<endl;getch();break;}

int i,j;

cout<<"Введите уровень и номер елемента на нем"<<endl;

cin>>i>>j;

if(&fd2(i,j)==NULL){cout<<"Елемент не найден"<<endl;getch();break;}

fd2.remove(&fd2(i,j));

}

}

getch();

continue;

}

case '6':

{

system("cls");

cout<<"Выберете дерево"<<endl;

switch(getch())

{

case '1':

{

system("cls");

char name[256];

cout<<"Выберано дерево 1"<<endl;

cout<<"Введите имя файла"<<endl;

cin>>name;

fd1.filein(name);

break;

}

case '2':

{

system("cls");

char name[256];

cout<<"Выберано дерево 2"<<endl;

cout<<"Введите имя файла"<<endl;

cin>>name;

fd2.filein(name);

break;

}

}

getch();

continue;

}

case '7':{

system("cls");

cout<<"Выберете дерево"<<endl;

switch(getch())

{

case '1':

{

system("cls");

char name[256];

cout<<"Выберано дерево 1"<<endl;

cout<<"Введите имя файла"<<endl;

cin>>name;

fd1.fileout(name);

break;

}

case '2':

{

system("cls");

char name[256];

cout<<"Выберано дерево 2"<<endl;

cout<<"Введите имя файла"<<endl;

cin>>name;

fd2.fileout(name);

break;

}

}

getch();

continue;

}

case '0':{break;}

default:{ continue;}

}

break;

}

break;

}

case'0':{return 0;}

}

system("cls");

}

return 0;

}

Tree.cpp

#include "Tree.h"

Tree::Tree()

{

Tree::son=NULL;

Tree::brother=NULL;

Tree::fath=NULL;

Tree::lvl=0;

Tree::root=NULL;

}

istream& operator>>(istream& cin, Tree& d)

{

cin>>d.info;

return cin;

}

Tree &Tree::operator()(int lvl, int elem)

{

Tree\* Pointer=root;

Tree\* temp=NULL;

Tree\* temp2=NULL;

bool found=0;

int l=lvl;

int e=elem;

if(lvl==0) return \*Pointer;

if(lvl==1)

{

Pointer=Pointer->son;

if(Pointer==NULL) return \*Pointer;

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

{

Pointer=Pointer->brother;

if(Pointer==NULL) return \*Pointer;

}

return \*Pointer;

}

temp2=Pointer->son;

l--;

gettorightlvl(temp,temp2,l,e,found);

Pointer=temp;

return \*Pointer;

}

void Tree::gettorightlvl (Tree\* &temp, Tree\* temp2,int curlvl, int &curelem,bool &found)

{

if(found) return;

if(curlvl==0)

{

if(found) return;

if(curelem==0){found=1; temp=temp2;}

if(found) return;

curelem--;

if(temp2->brother!=NULL){gettorightlvl(temp,temp2->brother,curlvl,curelem,found);}

return;

}

if(temp2->son!=NULL)gettorightlvl(temp,temp2->son,curlvl-1,curelem,found);

if(found) return;

if(temp2->brother!=NULL)gettorightlvl(temp,temp2->brother,curlvl,curelem,found);

}

Tree\* Tree::set(Tree\* nodefath)

{

if (nodefath->lvl==0)

{

nodefath->lvl=1;

nodefath->root=this;

return this;

}

if(nodefath->son!=NULL)

{

Tree \*b= new Tree;

Tree \*temp=nodefath->son;

while(temp->brother!=NULL)

temp=temp->brother;

temp->brother=b;

b->fath=nodefath;

b->lvl=(nodefath->lvl)+1;

b->root=nodefath->root;

return b;

}

if(nodefath->son==NULL)

{

Tree \*s= new Tree;

nodefath->son=s;

s->lvl=(nodefath->lvl)+1;

s->root=nodefath->root;

s->fath=nodefath;

return s;

}

}

ostream& operator<<(ostream& cout, const Tree d)

{

cout<<d.info;

return cout;

}

void Tree::visual(Tree\* node)

{

for(int i=0;i<node->lvl-1;i++)

cout<<" ";

cout<<\*node<<endl;

if (node->son!=NULL)

visual(node->son);

if (node->brother!=NULL)

visual(node->brother);

return;

}

void Tree::remove(Tree\* node)

{

if((node->lvl==0)||(node==NULL)){cout<<"невозможно удалить то что еще не создано"<<endl;}

if(node->brother!=NULL)

{

Tree\* temp=node->fath->son;

while(temp->brother!=node)

temp=temp->brother;

temp->brother=node->brother;

destroy(node);

return;

}

if(node==root)

{

strcpy(info,"\n");

Tree::fath=NULL;

Tree::lvl=0;

Tree::root=NULL;

if(node->son!=NULL) destroy(node->son);

Tree::son=NULL;

return;

}

if(node->fath!=NULL)

{

if(node->fath->son==node)

{

if (node->brother!=NULL) node->fath->son=node->brother;

destroy (node);

return;

}

}

destroy (node);

}

void Tree::destroy(Tree \*node)

{

if(node->son!=NULL)

destroy(node->son);

if(node->brother!=NULL)

destroy(node->brother);

delete node;

}

void Tree::operator=(Tree &Tree2)

{

strcpy(info,Tree2.info);

if(Tree2.son!=NULL)

{

destroy(this->son);

Tree \*s= new Tree(\*(Tree2.son));

this->son=s;

s->lvl=(this->lvl)+1;

s->root=this->root;

s->fath=this;

if((((Tree2.son)->son)!=NULL)||(((Tree2.son)->brother)!=NULL))

copyleftover(s,Tree2.son);

}

}

void Tree::copyleftover(Tree\* copynode,Tree\* wherenode)

{

if(wherenode->son!=NULL)

{

Tree \*s= new Tree(\*(wherenode->son));

copynode->son=s;

s->lvl=(copynode->lvl)+1;

s->root=copynode->root;

s->fath=copynode;

copyleftover(s,wherenode->son);

}

if(wherenode->brother!=NULL)

{

Tree \*b= new Tree(\*(wherenode->brother));

copynode->brother=b;

b->fath=copynode->fath;

b->lvl=copynode->lvl;

b->root=copynode->root;

copyleftover(b,wherenode->brother);

}

}

Tree::Tree(const Tree &A)

{

strcpy(info,A.info);

Tree::son=NULL;

Tree::brother=NULL;

Tree::fath=NULL;

Tree::lvl=0;

Tree::root=NULL;

}

void Tree::filein(char name[256])

{

ifstream fin(name);

if (!fin.is\_open())

cout << "Файл не может быть открыт!"<<endl;

else

{

fin >>\*(set(this));

while(fin)

{

int i=0;

int j=0;

fin >> i;

fin >> j;

if(&(operator()(i,j))==NULL){cout<<"Нарушена последовательность"<<endl;continue;}

fin>>\*(set(&(operator()(i,j))));

}

}

fin.close();

}

void Tree::fileout(char name[256])

{

ofstream fout(name);

if (!fout.is\_open())

cout << "Файл не может быть открыт!"<<endl;

else

{

visualfile(fout,this);

}

fout.close();

}

void Tree::visualfile(ofstream& fout,Tree\* node)

{

for(int i=0;i<node->lvl-1;i++)

fout<<" ";

fout<<\*node<<endl;

if (node->son!=NULL)

visualfile(fout,node->son);

if (node->brother!=NULL)

visualfile(fout,node->brother);

return;

}

void Tree::operator [](int lvl)

{

cout << "Вы запустили функцию перегрузки [] c параметром "<<lvl<<endl;

}

Tree::~Tree()

{

}

Tree.h

#include <iostream>

#include <stdlib.h>

#include <conio.h>

#include <fstream>

using namespace std;

#pragma once

class Tree

{

public:

Tree();//конструктор

Tree(const Tree &A);

void operator [](int lvl);

friend ostream& operator<<(ostream&, const Tree d);//вывести елемент дерево

friend istream& operator>>(istream& cin, Tree& d);//ввести елемет дерева

void visual(Tree\* node);//вывести все дерево;

Tree\* set(Tree\* nodefath); //задать сына елементу дерева через указатель

Tree &operator ()(int lvl, int elem);// доступ к елементу дерева

void operator=(Tree &Tree2);// объединения поддерева/дерево

void remove(Tree\* node);// удалить елемент или ветку

void filein(char name[256]);

void fileout(char name[256]);

~Tree();//деструктор

protected:

char info[80];

int lvl;

Tree\* son;

Tree\* brother;

Tree\* root;

Tree\* fath;

void visualfile(ofstream& fout,Tree\* node);

void copyleftover(Tree\* copynode,Tree\* wherenode);

void gettorightlvl (Tree\* &temp, Tree\* temp2,int curlvl,int &curelem,bool &found);

void destroy(Tree \*node);

};

FamilyTree.cpp

#include "FamilyTree.h"

void FamilyTree::myhasband(int lvl)

{

int elem=0;

for(int j=0; &(operator()(lvl,j))!=NULL;j++)

elem++;

cout<<"На "<<lvl<<" уровне у ваc "<< elem<<" родичей"<<endl;

}

FamilyTree.h

#pragma once

#include "tree.h"

class FamilyTree :public Tree

{

public:

void myhasband(int lvl);

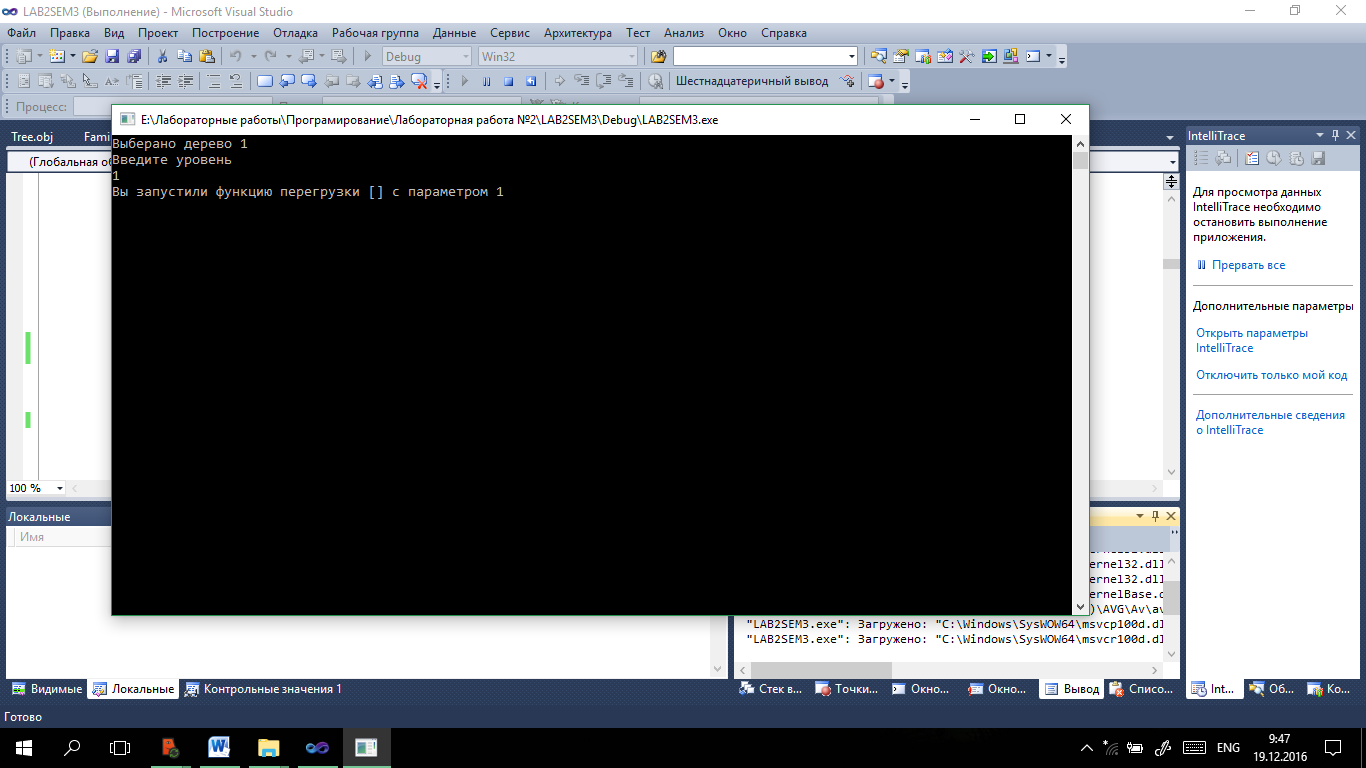
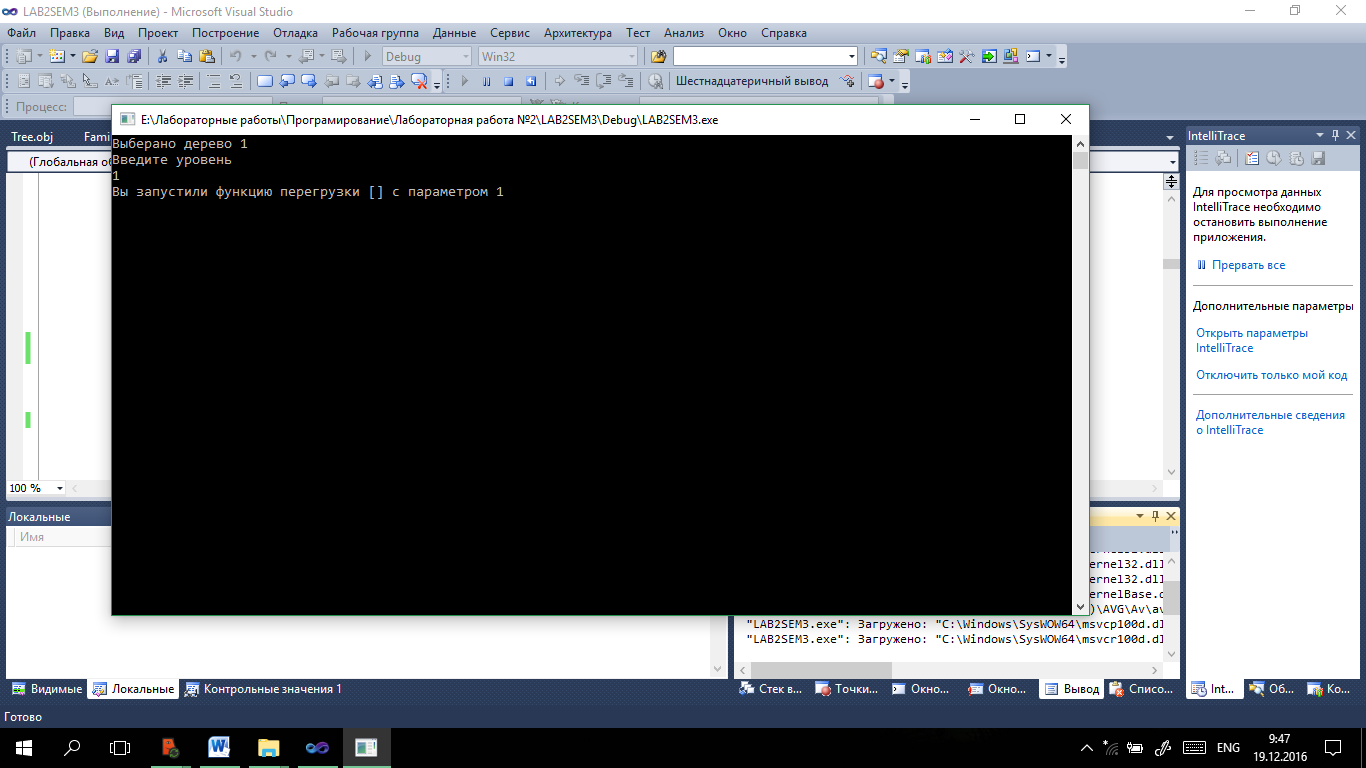
};

# 4. Опис інтерфейсу

Програма дотримується всіх принципів «дружнього інтерфейсу». Користувач працює с деревами за допомогою меню. Для розвантаження екрану кожний раз меню очищаеться та знову виводиться. Після виконання операцій користувач потрапляє у головне меню і знаходиться в ньмоу доти доки не вийде вводячи спеціальне значення.

# 5. Описання тестових прикладів

# 



# 6.Аналіз помилок

Зауваження1. Фактично у програмі відсутнє традиційне перевантаження квадратних скобочок. Для спрощення в головній програмі є лише два елементи типу дерева отже традиційне перевантаження не важливо.

Проблема1. У програмі відсутнє перевантаження «розумного доступу» адже таке перевантаження може зіпсувати вихідний код програми. Більшість функцій використовують «розумний доступ» у стандартному визначенні.

Зауваження2. У головній програмі не приведен приклад як в індивідуальному завданні на принцип перевантаження знака рівності,але він може бути продемонстрован за бажанням приймаючого.