Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение

высшего образования

Восточно-Сибирский государственный университет технологий и управления

(ФГБОУ ВСГУТУ)

Электротехнический факультет

Кафедра «Системы Информатики»

Дисциплина

«Программирования»

Отчет по лабораторной работе №4

На тему:

Директивы препроцессора и исключения

Вариант3

|  |  |
| --- | --- |
| Выполнил: | студент гр. Б661-2 |
|  | Коковихин А.В. |
| Проверил: | преподаватель  Жамбалов Э.Б. |

Улан-Удэ

2022

1. **Словесная постановка задачи**

Внесите в готовую программу из предыдущей лабораторной работы следующие изменения:

1. Программа теперь должна состоять из нескольких файлов. Функция main() должна содержаться в главном файле, в который подключаются остальные файлы. Программу на файлы разделите по функционалу. К примеру:
   1. Файл, содержащий структуру данных (здесь должны быть только структура без экземпляров).
   2. Файл, содержащий функции по работе с файловой системой.
   3. Файл, содержащий функции по работе со списком. Если файлы содержат только функции, то лучше всего создать для них заголовочный файл (.h) с тем же именем, что и файл с описанием/кодом этих функций (.cpp). В заголовочном файле определите прототипы этих функций. В главный файл с функцией main() подгружайте именно заголовочные файлы.
2. Добавьте условную компиляцию в программу (#if, #else, #elif и #endif). Определите макрос VER (#define VER). В зависимости от того чему он равен будет компилироваться определенная версия программы: 1. Данные в структуру считываются только с клавиатуры. Нужно сделать так, чтобы при VER=1блокировались возможности считывания и сохранения данных в файл. 2. Полная версия (VER=2), позволяющая считывать данные с файла и с клавиатуры.
3. **Решение задачи**
   1. **Формальная постановка задачи**

Для выполнения данной работы потребуется переписать некоторые элементы в коде. Так как нужно будет реализовывать работу много файлового проекта.

В отдельном файле будут лежать функции и их определения, помимо них в отдельных файлах будет лежать структура и подключаемые библиотеки для удобства использования, а также некоторые другие элементы.

* 1. **Проектирование**

При разделении кода на файлы, будет иметься 5 файлов:

* в 1-м будут находиться подключаемые библиотеки;
* во 2-м будет находиться структура с данными;
* в 3-м будут находиться функции;
* в 4-м будет находиться определения под функции;
* в 5-м будет находиться главная функция.

При наличии неполной версии пользователю будут недоступны функции сохранения и загрузки из файла. Помимо этого, будет выводиться сообщение о наличии неполной версии. С полной же версией будет доступен весь функционал и о её наличии пользователю также будет сообщено.

* 1. **Реализация**

Файл с библиотеками будет содержать следующий код:

#pragma once

#include <iostream>

#include <Windows.h>

#include <string>

#include <math.h>

#include <stdio.h>

#include <exception>

#include <fstream>

#include <conio.h>

using namespace std;

#define LR 4

#include "Header4.h"

#include "Structs4.h"

#include "Functions4.h"

#include "Header5.h"

#include "Person.h"

#include "Linked\_PersonPointers.h"

#include "PersonList.h"

В нём будут лежать библиотеки, используемые в работе.

Файл со структурой будет подключать файл с библиотеками и содержать структуру «Person».

#pragma once

#include "Header.h"

#if LR == 4

struct Person

{

struct

{

string SecondName;

string FirstName;

bool Has\_MidleName = false;

string MidleName;

} Name;

string Adress;

string Phone\_Number;

};

struct Linked\_PersonPointers

{

Linked\_PersonPointers\* Next = NULL;

Linked\_PersonPointers\* Previos = NULL;

Person\* Person = NULL;

};

struct Persons\_Tree

{

Persons\_Tree\* Left = NULL;

Persons\_Tree\* Right = NULL;

Person\* Curent = NULL;

};

enum class MenuCommand

{

Exit,

Create,

Sort,

Edit,

Delete,

Show\_All,

Search,

Save\_As,

Load\_As,

Save,

Load,

};

#endif

Следующим на очереди будет файл содержащий определения функций.

#pragma once

#include "Header.h"

#if LR == 4

MenuCommand Show\_Menu();

string Get\_FIO(const Person p);

void Print(Person p);

Persons\_Tree\* Build\_Tree(Linked\_PersonPointers\* item);

Linked\_PersonPointers\* Tree\_To\_List(Persons\_Tree\* tree, Linked\_PersonPointers\* tail = NULL);

void Print(Person\* p);

void Print(Linked\_PersonPointers\* head);

Linked\_PersonPointers\* Add\_In\_List(Person\*& p, Linked\_PersonPointers\* tal = NULL);

Linked\_PersonPointers\* Get\_Linc\_On\_Index(int index, Linked\_PersonPointers\*& head);

void Remove\_From\_List(int index, Linked\_PersonPointers\*& head, Linked\_PersonPointers\*& tail);

int EnterIntValue();

string EnterStringValue();

Person\* Enter\_Person\_Data();

void Add\_Person(Linked\_PersonPointers\*& head, Linked\_PersonPointers\*& tail, Person\* p);

void Create\_Person(Linked\_PersonPointers\*& head, Linked\_PersonPointers\*& tail);

void Delete\_Person(Linked\_PersonPointers\*& head, Linked\_PersonPointers\*& tail);

void Edit\_Person(Linked\_PersonPointers\*& head, Linked\_PersonPointers\*& tail);

void Sort(Linked\_PersonPointers\*& head, Linked\_PersonPointers\*& tail);

void Search(Linked\_PersonPointers\* curent);

void Save(Linked\_PersonPointers\* head, const char filename[] = "LR\_3\_Individual.txt");

void Load(Linked\_PersonPointers\*& head, Linked\_PersonPointers\*& tail, const char filename[] = "LR\_3\_Individual.txt");

void Save\_As(Linked\_PersonPointers\* head);

void Load\_As(Linked\_PersonPointers\*& head, Linked\_PersonPointers\*& tail);

#endif

Самым объёмным файлом будет являться файл с основными функциями, он будет подключать файл c определениями. Этот файл будет содержать следующий код:

#include "Functions4.h"

#if LR == 4

MenuCommand Show\_Menu()

{

cout << "0) Exit" << endl;

cout << "1) Create" << endl;

cout << "2) Sort" << endl;

cout << "3) Edit" << endl;

cout << "4) Delete" << endl;

cout << "5) Show\_All" << endl;

cout << "6) Search" << endl;

#if VER >= 2

cout << "7) Save\_As" << endl;

cout << "8) Load\_As" << endl;

cout << "9) Save" << endl;

cout << "10) Load" << endl;

#endif

cout << "Chouse command: ";

int i;

cin >> i;

return (MenuCommand)i;

}

string Get\_FIO(const Person p)

{

return p.Name.SecondName + ' ' + p.Name.FirstName + ' ' + (p.Name.Has\_MidleName ? p.Name.MidleName : "");

}

void Print(Person p)

{

cout << "Person: " << Get\_FIO(p) << endl;

cout << "- Adress:\t" << p.Adress << endl;

cout << "- Phone:\t" << p.Phone\_Number << endl;

}

Persons\_Tree\* Build\_Tree(Linked\_PersonPointers\* item)

{

Persons\_Tree\* root = new Persons\_Tree();

Persons\_Tree\* branch = root;

while (item)

{

if (branch->Curent == NULL)

{

branch->Curent = item->Person;

}

else

{

string fio1 = branch->Curent->Name.SecondName + branch->Curent->Name.FirstName + branch->Curent->Name.MidleName;

string fio2 = item->Person->Name.SecondName + item->Person->Name.FirstName + item->Person->Name.MidleName;

if (fio1 > fio2)

{

if (branch->Left == NULL)

branch->Left = new Persons\_Tree();

branch = branch->Left;

continue;

}

else

{

if (branch->Right == NULL)

branch->Right = new Persons\_Tree();

branch = branch->Right;

continue;

}

}

item = item->Next;

branch = root;

}

return root;

}

Linked\_PersonPointers\* Tree\_To\_List(Persons\_Tree\* tree, Linked\_PersonPointers\* tail)

{

if (tree == NULL)

{

return NULL;

}

if (tree->Left)

{

tail = Tree\_To\_List(tree->Left, tail);

}

if (tail)

{

tail->Next = new Linked\_PersonPointers();

tail->Next->Previos = tail;

tail = tail->Next;

}

else

tail = new Linked\_PersonPointers();

tail->Person = tree->Curent;

if (tree->Right)

{

tail = Tree\_To\_List(tree->Right, tail);

}

return tail;

}

void Print(Person\* p)

{

cout << "Person" << endl;

cout << "- Name: \t" << Get\_FIO(\*p) << endl;

cout << "- Adress:\t" << p->Adress << endl;

cout << "- Phone: \t" << p->Phone\_Number << endl;

}

void Print(Linked\_PersonPointers\* head)

{

if (head == NULL)

{

cout << "List is empty!" << endl;

return;

}

Print(head->Person);

if (head->Next)

{

cout << endl;

Print(head->Next);

}

}

Linked\_PersonPointers\* Add\_In\_List(Person\*& p, Linked\_PersonPointers\* tal)

{

Linked\_PersonPointers\* next = new Linked\_PersonPointers;

next->Previos = tal;

next->Person = p;

if (tal != NULL)

tal->Next = next;

return next;

}

Linked\_PersonPointers\* Get\_Linc\_On\_Index(int index, Linked\_PersonPointers\*& head)

{

Linked\_PersonPointers\* curent = head;

int i = 0;

while (curent != NULL)

{

if (i == index)

break;

i++;

curent = curent->Next;

}

return curent;

}

void Remove\_From\_List(int index, Linked\_PersonPointers\*& head, Linked\_PersonPointers\*& tail)

{

if (head == NULL)

{

cout << "List is empty!" << endl;

return;

}

Linked\_PersonPointers\* deleting\_person = Get\_Linc\_On\_Index(index, head);

if (deleting\_person)

{

cout << "Do you want to remove: " << endl;

Print(deleting\_person->Person);

cout << "Y/N: ";

char c;

cin >> c;

if (c != 'Y' && c != 'y')

return;

if (deleting\_person->Next && deleting\_person->Previos)

{

deleting\_person->Previos->Next = deleting\_person->Next;

deleting\_person->Next->Previos = deleting\_person->Previos;

}

else if (deleting\_person->Next)

{

deleting\_person->Next->Previos = NULL;

head = deleting\_person->Next;

}

else if (deleting\_person->Previos)

{

deleting\_person->Previos->Next = NULL;

tail = deleting\_person->Previos;

}

else

{

head = NULL;

tail = NULL;

}

delete deleting\_person;

}

else

{

throw exception("Index out of range!");

}

}

int EnterIntValue()

{

int result = 0;

string bufer;

cin >> bufer;

int size = bufer.length();

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

{

char c = bufer[i];

if (isdigit(c) == false)

throw exception("Invalid type!");

int n = pow(10, (size - i - 1)) \* (c - '0');

result += n;

}

return result;

}

string EnterStringValue()

{

string result = "";

string bufer;

cin >> bufer;

int size = bufer.length();

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

{

char c = bufer[i];

if (isdigit(c))

throw exception("Invalid type!");

result += c;

}

return result;

}

string EnterByMask(const char mask[], int len)

{

string result;

char c;

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

{

c = mask[i];

if (c == '%')

{

c = mask[++i];

switch (c)

{

case 'i':

{

c = \_getch();

if (isdigit(c) == false)

throw exception("Invalid type!");

cout << c;

result += c;

break;

}

case 's':

{

while (true)

{

c = \_getch();

if (c == ' ')

break;

if (isdigit(c))

throw exception("Invalid type!");

cout << c;

result += c;

}

break;

}

case 'n':

{

do

{

c = \_getch();

if (c == ' ')

break;

if (isdigit(c) == false)

throw exception("Invalid type!");

cout << c;

result += c;

} while (c != ' ');

break;

}

}

}

else

{

cout << c;

result += c;

}

}

cout << endl;

return result;

}

Person\* Enter\_Person\_Data()

{

Person\* creating\_person = new Person();

cout << "Enter second name: ";

creating\_person->Name.SecondName = EnterByMask("%s", 3);

cout << "Enter first name: ";

creating\_person->Name.FirstName = EnterByMask("%s", 3);

cout << "Has middle name? (Y/N): ";

char c;

cin >> c;

if (c == 'Y' || c == 'y')

{

cout << "Enter midle name: ";

creating\_person->Name.MidleName = EnterByMask("%s", 3);

creating\_person->Name.Has\_MidleName = true;

}

cout << "Enter adress:" << endl;

creating\_person->Adress = EnterByMask("Sity: %s Street: %s House: %n", 30);

cout << "Enter phone number: ";

creating\_person->Phone\_Number = EnterByMask("+%i(%i%i%i) %i%i%i-%i%i-%i%i", 29);

return creating\_person;

}

void Add\_Person(Linked\_PersonPointers\*& head, Linked\_PersonPointers\*& tail, Person\* p)

{

if (head == NULL)

{

head = Add\_In\_List(p);

tail = head;

}

else

tail = Add\_In\_List(p, tail);

}

void Create\_Person(Linked\_PersonPointers\*& head, Linked\_PersonPointers\*& tail)

{

Person\* creating\_person = Enter\_Person\_Data();

Add\_Person(head, tail, creating\_person);

}

void Delete\_Person(Linked\_PersonPointers\*& head, Linked\_PersonPointers\*& tail)

{

cout << "Enter deleting person number: ";

int index;

cin >> index;

Remove\_From\_List(index, head, tail);

cout << "Deleted!" << endl;

}

void Edit\_Person(Linked\_PersonPointers\*& head, Linked\_PersonPointers\*& tail)

{

int i;

cout << "Editing person number: ";

cin >> i;

Linked\_PersonPointers\* editing\_person = Get\_Linc\_On\_Index(i, head);

if (editing\_person)

editing\_person->Person = Enter\_Person\_Data();

else

throw exception("Index out of range!");

}

void Sort(Linked\_PersonPointers\*& head, Linked\_PersonPointers\*& tail)

{

if (head == NULL)

{

cout << "List is empty!";

return;

}

Persons\_Tree\* root = Build\_Tree(head);

tail = Tree\_To\_List(root);

Linked\_PersonPointers\* curent = tail;

while (true)

{

if (curent->Previos == NULL)

{

head = curent;

break;

}

curent = curent->Previos;

}

}

void Search(Linked\_PersonPointers\* curent)

{

cout << "Enter FIO patern: ";

char find[255];

cin.ignore();

cin.getline(find, 254);

bool is\_finded = false;

while (curent)

{

string fio = Get\_FIO(\*curent->Person);

if (fio.find(find) != -1)

{

Print(curent->Person);

is\_finded = true;

}

curent = curent->Next;

}

if (is\_finded == false)

cout << "Nothing!" << endl;

}

void Save(Linked\_PersonPointers\* head, const char filename[])

{

ofstream out;

out.open(filename);

if (out.is\_open())

{

Linked\_PersonPointers\* curent = head;

while (curent)

{

Person p = \*curent->Person;

out << p.Adress << ' ' << p.Name.FirstName << ' ' << p.Name.SecondName << ' ' << p.Name.Has\_MidleName << ' ' << p.Name.MidleName << ' ' << p.Phone\_Number;

curent = curent->Next;

if (curent)

out << endl;

}

out.close();

}

else

throw exception("File is not exist!");

}

void Load(Linked\_PersonPointers\*& head, Linked\_PersonPointers\*& tail, const char filename[])

{

ifstream input;

input.open(filename);

if (input.is\_open())

{

while (input.eof() == false)

{

Person\* p = new Person;

input >> p->Adress;

input >> p->Name.FirstName;

input >> p->Name.SecondName;

input >> p->Name.Has\_MidleName;

if (p->Name.Has\_MidleName)

input >> p->Name.MidleName;

input >> p->Phone\_Number;

Add\_Person(head, tail, p);

}

input.close();

}

else

throw exception("File is not exist!");

}

void Save\_As(Linked\_PersonPointers\* head)

{

if (head == NULL)

{

cout << "List is Empty!" << endl;

return;

}

cout << "Enter SaveFile Name: ";

string filename;

cin >> filename;

filename += ".txt";

Save(head, filename.c\_str());

}

void Load\_As(Linked\_PersonPointers\*& head, Linked\_PersonPointers\*& tail)

{

cout << "Enter LoadFile Name: ";

string filename;

cin >> filename;

filename += ".txt";

Load(head, tail, filename.c\_str());

};

#endif

Ну и файл с главной функцией:

#include "Header4.h"

#if LR == 4

void All()

{

cout << "Version: " << VER << endl << endl;

Linked\_PersonPointers\* head = NULL;

Linked\_PersonPointers\* tail = NULL;

while (true)

{

try

{

switch (Show\_Menu())

{

case MenuCommand::Create:

Create\_Person(head, tail);

break;

case MenuCommand::Delete:

Delete\_Person(head, tail);

break;

case MenuCommand::Edit:

Edit\_Person(head, tail);

break;

case MenuCommand::Search:

Search(head);

break;

case MenuCommand::Show\_All:

Print(head);

break;

case MenuCommand::Sort:

Sort(head, tail);

Print(head);

break;

#if 2 <= VER

case MenuCommand::Save\_As:

Save\_As(head);

break;

case MenuCommand::Load\_As:

Load\_As(head, tail);

break;

case MenuCommand::Save:

Save(head);

break;

case MenuCommand::Load:

Load(head, tail);

break;

#endif

case MenuCommand::Exit:

return;

default:

cout << "Wrong command!";

}

}

catch (exception e)

{

cout << e.what() << endl;

}

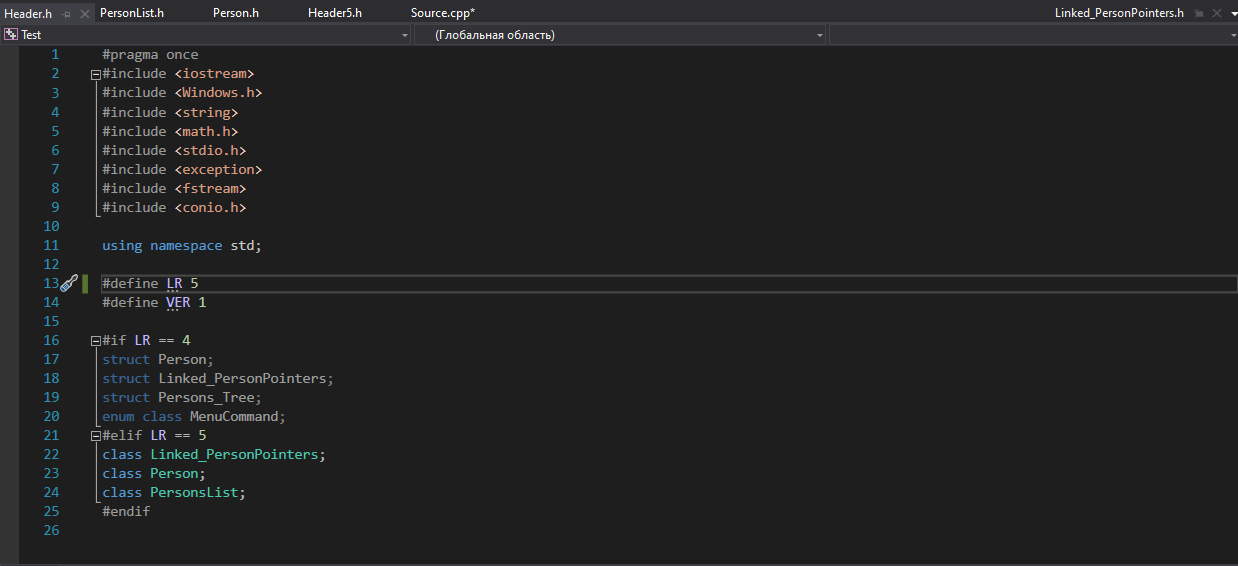
}

}

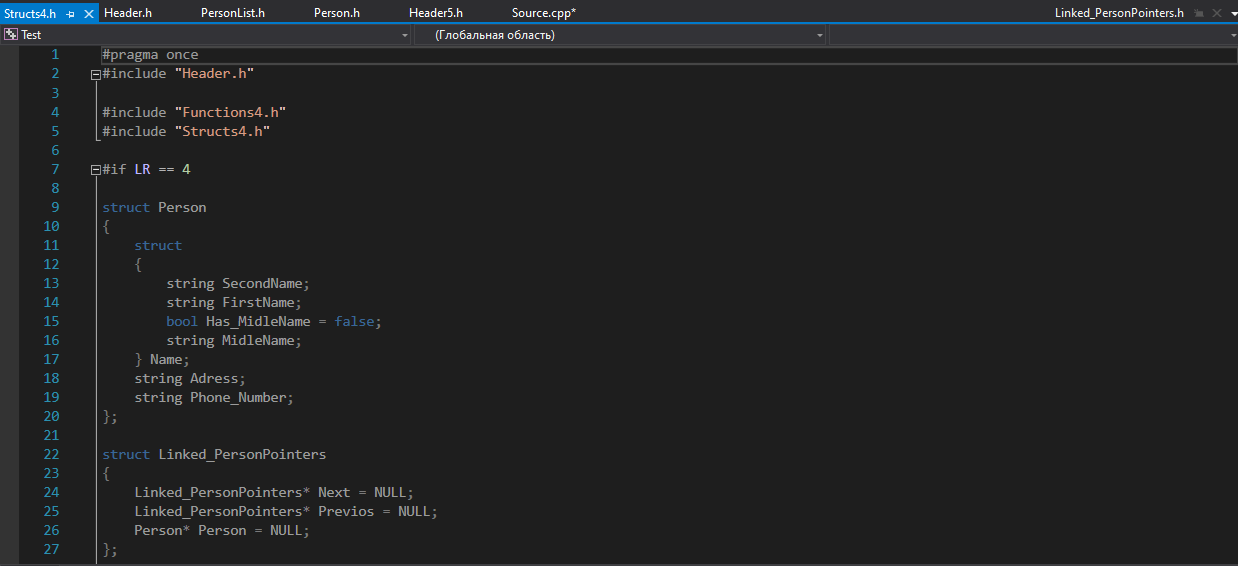
#endif

* 1. **Тестирование**

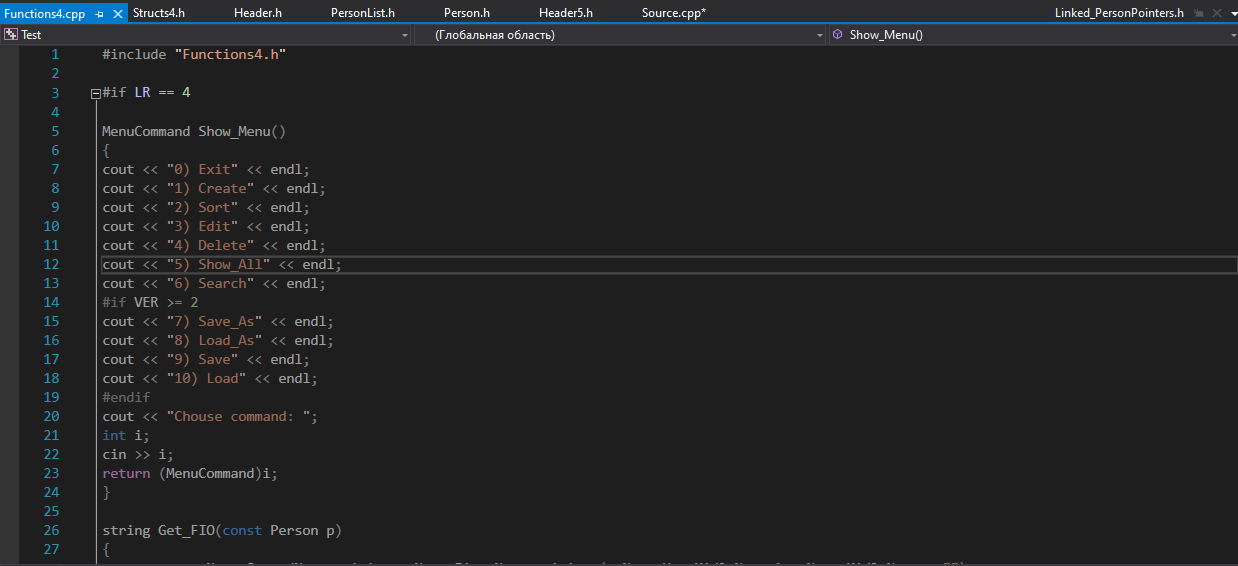
Файл с библиотеками



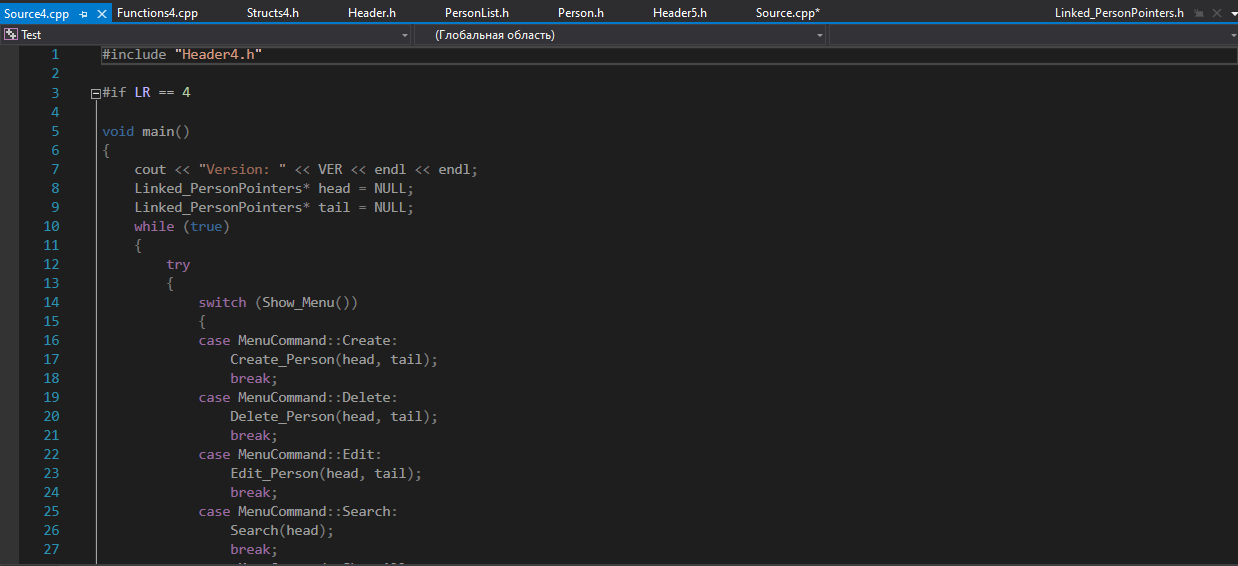
Файл со структурой



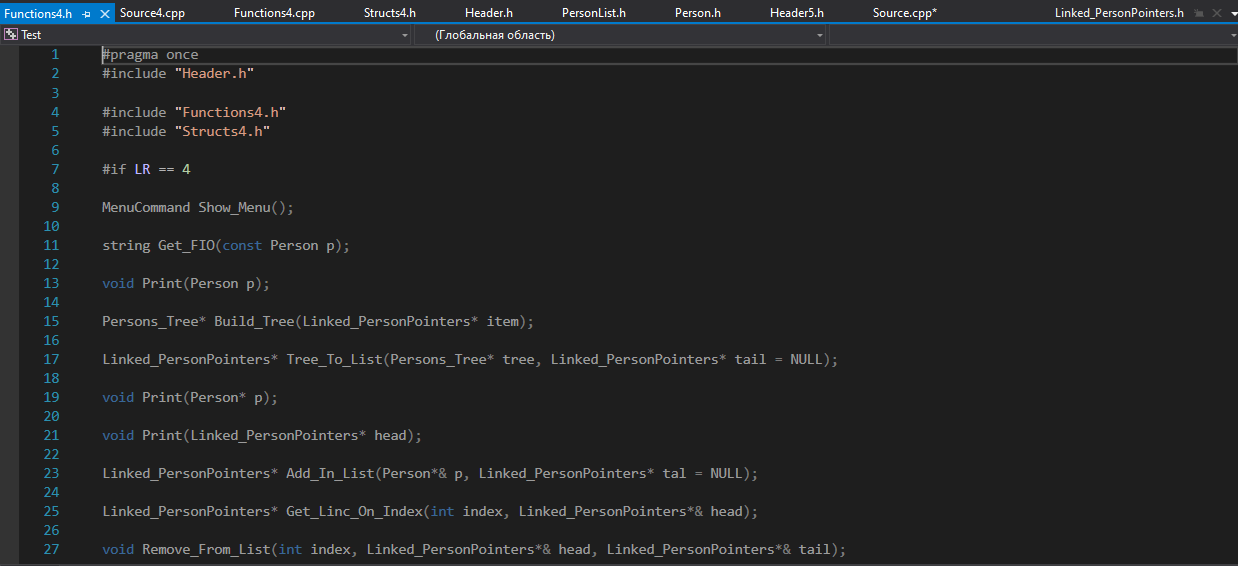
Файл с основными функциями



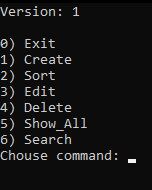
Файл с главной функцией

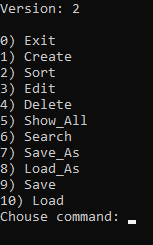


Файл с определениями функций

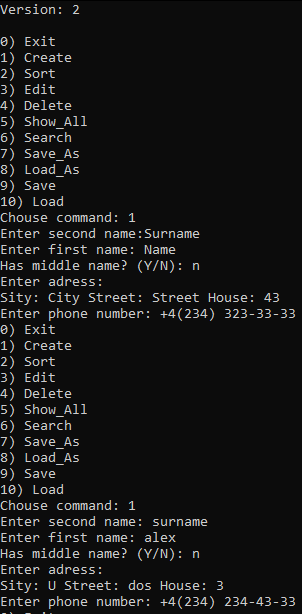


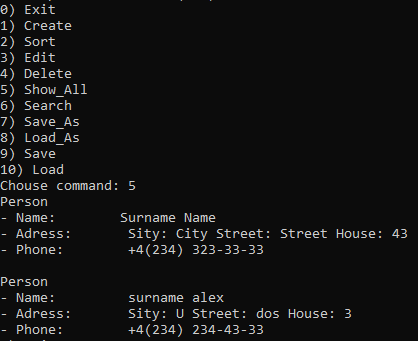
А теперь перейдём к тестированию программы

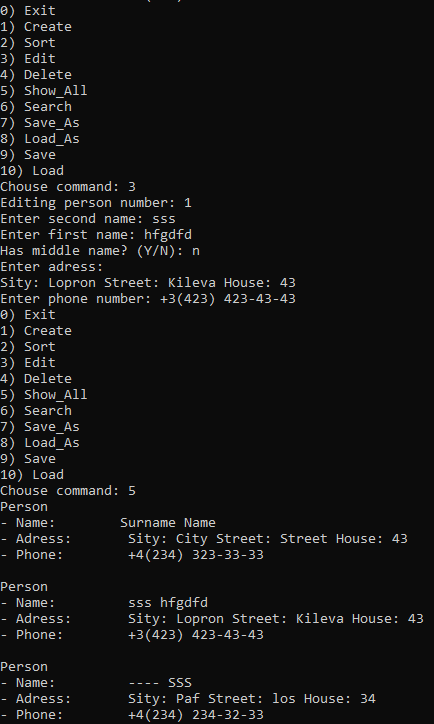


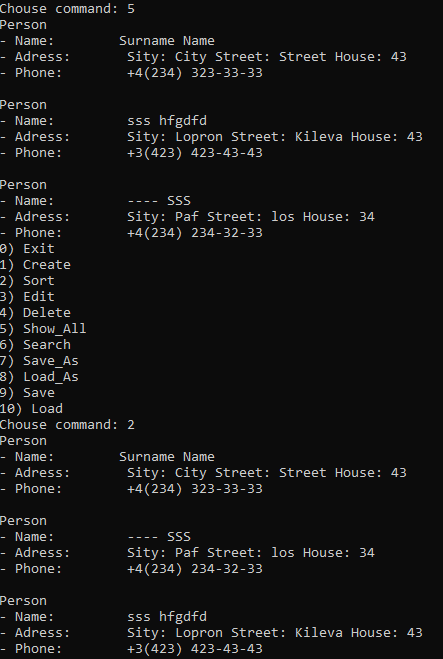


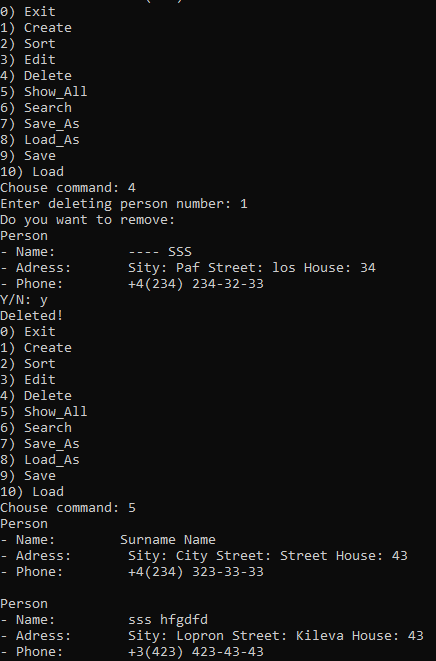
Условная компиляция работает корректно

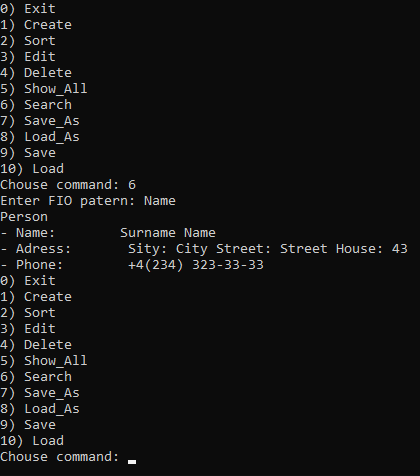


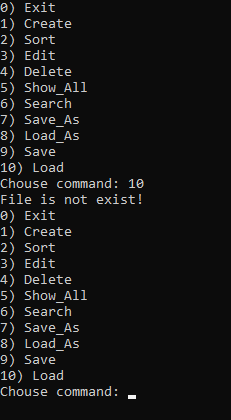


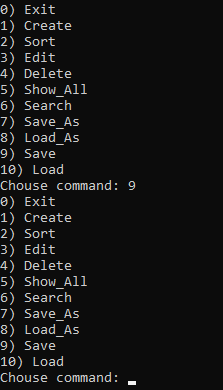


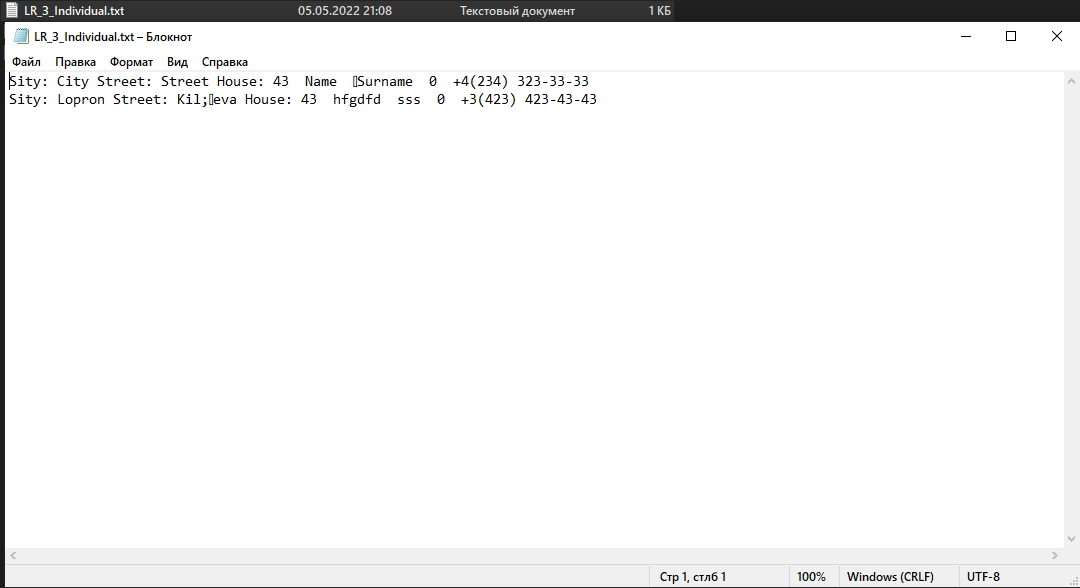












Функции, также работают безотказно

**Заключение**

В данной работе я познакомился с директивами препроцессора, условной компиляцией и исключениями, принципами их функционирования. Также закрепил это на практике переделав прошлую лабораторную работу.