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 №5

На тему:

Введение в ООП

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. Вместо файла, содержащего структуру данных у вас должен быть определен файл, содержащий класс списка.
2. Класс должен включать в себя все необходимые для работы со списком элементы: функции по работе со списком и переменные-указатели на список. При этом переменные указатели должны быть недоступны извне, то есть, определены как private. По факту в главной функции будет создаваться объект класса списка, и вся работа со списком должна вестись только через его методы.
3. Определите конструкторы списка. К примеру, пустой конструктор и конструктор принимающий данные для создания первого элемента списка. Также определите деструктор для списка (функция очистки списка).
4. Сам класс должен быть определен в заголовочном файле (.h), однако реализация его методов должны быть вынесена в файл кода (.cpp).
5. **Решение задачи**
   1. **Формальная постановка задачи**

Для выполнения данной работы потребуется переписать структуры на классы и переместить все методы в него, так как работа с классом осуществляется через его методы.

Помимо этого, требуется пара конструкторов, один из которых будет содержать значения по умолчанию или же пустой, а другой, в который будут передаваться параметры.

Так как структура изначально была вынесена в другой файл, она не требует перемещения и дополнительной траты времени.

* 1. **Проектирование**

Класс будет иметь точно такие же поля что и структура, только вот он будет не во вложенной структуре, а во вложенном классе. До этого у меня была организована такая система, поэтому трудности в изменений не последует.

Для объявления конструкторов будет использоваться имя класса и разные сигнатуры или же разное количество аргументов, подаваемых в конструктор.

Деструктор будет очищать память занятую экземплярами.

* 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 5

#define VER 2

#if LR == 4

struct Person;

struct Linked\_PersonPointers;

struct Persons\_Tree;

enum class MenuCommand;

#elif LR == 5

class Linked\_PersonPointers;

class Person;

class PersonsList;

#endif

Файл с классом будет содержать следующий код:

#include "Header.h"

#include "Person.h"

#include "Linked\_PersonPointers.h"

#include "PersonList.h"

#if LR == 5

enum class MenuCommand

{

Exit\_Command,

Create\_Command,

Sort\_Command,

Edit\_Command,

Delete\_Command,

Show\_All\_Command,

Search\_Command,

Save\_As\_Command,

Load\_As\_Command,

Save\_Command,

Load\_Command,

};

void Load(PersonsList& list, 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->FullName.FirstName;

input >> p->FullName.SecondName;

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

if (p->FullName.Has\_MidleName)

input >> p->FullName.MidleName;

input >> p->Phone\_Number;

list.Add(\*p);

}

input.close();

}

else

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

}

void Save(PersonsList& list, const char filename[])

{

ofstream out;

out.open(filename);

if (out.is\_open())

{

Linked\_PersonPointers\* curent = list.head;

while (curent)

{

Person p = \*curent->PersonData;

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

curent = curent->Next;

if (curent)

out << endl;

}

out.close();

}

else

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

}

void Load\_As(PersonsList list)

{

cout << "Enter LoadFile Name: ";

string filename;

cin >> filename;

filename += ".txt";

Load(list, filename.c\_str());

}

void Save\_As(PersonsList list)

{

if (list.Lenght() == 0)

{

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

return;

}

cout << "Enter SaveFile Name: ";

string filename;

cin >> filename;

filename += ".txt";

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

}

Person Enter\_Person\_Data()

{

Person creating\_person;

cout << "Enter second name: ";

cin >> creating\_person.FullName.SecondName;

cout << "Enter first name: ";

cin >> creating\_person.FullName.FirstName;

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

char c;

cin >> c;

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

{

cout << "Enter midle name: ";

cin >> creating\_person.FullName.MidleName;

creating\_person.FullName.Has\_MidleName = true;

}

cout << "Enter adress: ";

cin >> creating\_person.Adress;

cout << "Enter phone number: ";

cin >> creating\_person.Phone\_Number;

return creating\_person;

}

MenuCommand Choose\_Command()

{

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

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

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

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

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

cout << "5) Print all Persons" << endl;

cout << "6) Search Person by FIO" << endl;

cout << "7) Save As" << endl;

cout << "8) Load As" << endl;

cout << "9) Default save" << endl;

cout << "10) Default load" << endl;

int i;

cout << "Enter command number: ";

cin >> i;

cout << endl;

if (i < 0 || 10 < i)

{

cout << "Wrong command!" << endl;

return Choose\_Command();

}

else

return (MenuCommand)i;

}

void All()

{

PersonsList list;

MenuCommand c = MenuCommand::Create\_Command;

while (c != MenuCommand::Exit\_Command)

{

try

{

c = Choose\_Command();

switch (c)

{

case MenuCommand::Create\_Command:

list.Add(Enter\_Person\_Data());

break;

case MenuCommand::Sort\_Command:

list.Sort();

break;

case MenuCommand::Edit\_Command:

{

int index;

cout << "Enter index: ";

cin >> index;

list.Edit(index, Enter\_Person\_Data());

break;

}

case MenuCommand::Delete\_Command:

{

int index;

cout << "Enter index: ";

cin >> index;

list.Delete(index);

break;

}

case MenuCommand::Show\_All\_Command:

cout << list.Get\_AllData();

break;

case MenuCommand::Search\_Command:

{

cout << "Enter template: ";

char name\_template[255];

cin.getline(name\_template, 255);

cin.ignore();

cout << list.Get\_AllData(name\_template);

break;

}

case MenuCommand::Save\_As\_Command:

Save\_As(list);

break;

case MenuCommand::Load\_As\_Command:

Load\_As(list);

break;

case MenuCommand::Save\_Command:

Save(list, "LR5.txt");

break;

case MenuCommand::Load\_Command:

Load(list, "LR5.txt");

break;

default:

break;

}

}

catch (exception e)

{

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

}

}

}

class Rect

{

private:

int \_width, \_height;

public:

Rect()

{

\_width = \_height = 0;

}

Rect(int width, int height)

{

\_width = width;

\_height = height;

}

void Set\_Width(int value)

{

if (value >= 0)

\_width = value;

}

void Set\_Height(int value)

{

if (value >= 0)

\_height = value;

}

int Get\_Height()

{

return \_height;

}

int Get\_Width()

{

return \_width;

}

int Get\_S()

{

return \_width \* \_height;

}

int Get\_P()

{

return 2 \* \_width + 2 \* \_height;

}

};

enum class Gender\_Type {

Male, Female, Undefinded

};

class PersonData

{

public:

char\* Name;

char\* Surename;

char\* Patronim;

int Age;

Gender\_Type Gender;

private:

void Set\_StrData(const char value[], char\*& field)

{

int lenght;

lenght = strlen(value);

if (field == NULL || strlen(field) != lenght + 1)

{

if (field != NULL)

delete field;

field = new char[lenght + 1];

}

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

{

field[i] = value[i];

}

field[lenght] = '\0';

}

public:

PersonData()

{

Name = Surename = Patronim = NULL;

Age = 0;

Gender = Gender\_Type::Undefinded;

}

PersonData(const char name[], const char surename[], const char patronim[], int age, Gender\_Type gender): PersonData()

{

Set\_StrData(name, Name);

Set\_StrData(surename, Surename);

Set\_StrData(patronim, Patronim);

Age = age;

Gender = gender;

}

PersonData(const PersonData\* copy): PersonData()

{

cout << "Copy: " << copy << " this: " << this << endl;

Set\_StrData(copy->Name, Name);

Set\_StrData(copy->Surename, Surename);

Set\_StrData(copy->Patronim, Patronim);

Age = copy->Age;

Gender = copy->Gender;

}

PersonData(const PersonData& copy) : PersonData()

{

cout << "Copy: " << &copy << " this: " << this << endl;

Set\_StrData(copy.Name, Name);

Set\_StrData(copy.Surename, Surename);

Set\_StrData(copy.Patronim, Patronim);

Age = copy.Age;

Gender = copy.Gender;

}

~PersonData()

{

cout << "Delete: " << this << endl;

if (Name) delete[] Name;

if (Surename) delete[] Surename;

if (Patronim) delete[] Patronim;

Name = Surename = Patronim = NULL;

}

void Print(const char format[])

{

int lenght = strlen(format);

bool key = false;

for (const char\* c = &format[0]; c < &format[lenght]; c++)

{

if (\*c == '%')

{

key = true;

continue;

}

if (key)

{

key = false;

switch (\*c)

{

case 'N':

case 'n':

cout << (Name != NULL ? Name : "NULL");

break;

case 'S':

case 's':

cout << (Surename != NULL ? Surename : "NULL");

break;

case 'P':

case 'p':

cout << (Patronim != NULL ? Patronim : "NULL");

break;

case 'A':

case 'a':

cout << Age;

break;

case 'G':

case 'g':

cout << (Gender == Gender\_Type::Male ? "Man" : Gender == Gender\_Type::Female ? "Wooman" : "Unedfinded");

break;

default:

cout << '%' << \*c;

}

}

else

cout << \*c;

}

}

};

void main()

{

system("pause");

All();

}

#endif

#if LR == 4

#endif

Файл с определениями класса и некоторые вспомогательные файлы будет иметь следующий код:

Файл «Person.h»

#pragma once

#include "Header.h"

#if LR == 5

class Person

{

public:

class Full\_Name

{

public:

string SecondName;

string FirstName;

bool Has\_MidleName = false;

string MidleName;

string Get\_FullName()

{

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

}

} FullName;

string Adress;

string Phone\_Number;

Person()

{

FullName.FirstName = "";

FullName.SecondName = "";

FullName.MidleName = "";

FullName.Has\_MidleName = false;

Adress = "";

Phone\_Number = "";

}

Person(const Person& p) :Person()

{

FullName.FirstName = p.FullName.FirstName;

FullName.SecondName = p.FullName.SecondName;

FullName.MidleName = p.FullName.MidleName;

FullName.Has\_MidleName = p.FullName.Has\_MidleName;

Adress = p.Adress;

Phone\_Number = p.Phone\_Number;

}

};

#endif

Файл «PersonList.h»

#pragma once

#include "Header.h"

#if LR == 5

class PersonsList

{

private:

int \_lenght;

Linked\_PersonPointers\* head;

Linked\_PersonPointers\* tail;

public:

friend void Save(PersonsList& list, const char filename[]);

PersonsList()

{

head = tail = NULL;

\_lenght = 0;

}

PersonsList(Person person) :PersonsList()

{

head = tail = new Linked\_PersonPointers(person);

\_lenght = 1;

}

PersonsList(const PersonsList& copy) :PersonsList()

{

for (int i = 0; i < copy.\_lenght; i++)

{

Add(\*copy[i]->PersonData);

}

}

~PersonsList()

{

Linked\_PersonPointers\* temp;

while (head)

{

temp = head;

head = head->Next;

delete temp;

}

}

void Add(Person person)

{

Linked\_PersonPointers\* adding = new Linked\_PersonPointers(person);

if (head && tail)

tail->Next = adding;

else

head = adding;

tail = adding;

\_lenght++;

}

void Edit(int index, Person person)

{

if (Index\_IsOutOfRange(index))

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

delete this->operator[](index)->PersonData;

this->operator[](index)->PersonData = new Person(person);

}

void Delete(int index)

{

if (Index\_IsOutOfRange(index))

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

Linked\_PersonPointers\* before = this->operator[](index - 1);

Linked\_PersonPointers\* curent = this->operator[](index);

Linked\_PersonPointers\* after = this->operator[](index + 1);

before->Next = after;

after->Previos = before;

delete curent;

\_lenght--;

}

void Sort()

{

for (int loop = 0; loop < \_lenght; loop++)

for (int i = 0; i < \_lenght - loop - 1; i++)

{

Linked\_PersonPointers\* a = this->operator[](i);

Linked\_PersonPointers\* b = this->operator[](i + 1);

if (a->PersonData->FullName.Get\_FullName() > b->PersonData->FullName.Get\_FullName())

{

auto temp = a->PersonData;

a->PersonData = b->PersonData;

b->PersonData = a->PersonData;

}

}

}

string Get\_AllData(string name\_template = "")

{

string result = "";

for (int i = 0; i < \_lenght; i++)

{

Person p = \*this->operator[](i)->PersonData;

string fullName = p.FullName.Get\_FullName();

if (fullName.find(name\_template) != -1)

{

result += "Full Name: " + fullName + '\n';

result += "Adress: " + p.Adress + '\n';

result += "Phone: " + p.Phone\_Number + '\n' + '\n';

}

}

return result;

}

int Lenght()

{

return \_lenght;

}

bool Index\_IsOutOfRange(int index) const

{

return index < 0 || index >= \_lenght;

}

Linked\_PersonPointers\* operator [](int index) const

{

if (Index\_IsOutOfRange(index))

return NULL;

Linked\_PersonPointers\* temp = head;

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

{

if (i == index)

return temp;

temp = temp->Next;

}

}

};

#endif

Файл «LinkedPersonPointers.h»

#pragma once

#include "Header.h"

#include "Person.h"

#if LR == 5

class Linked\_PersonPointers

{

public:

Linked\_PersonPointers\* Next;

Linked\_PersonPointers\* Previos;

Person\* PersonData;

Linked\_PersonPointers()

{

Next = Previos = NULL;

PersonData = NULL;

}

Linked\_PersonPointers(const Linked\_PersonPointers& copy) :Linked\_PersonPointers()

{

PersonData = new Person(\*copy.PersonData);

}

Linked\_PersonPointers(Person person) :Linked\_PersonPointers()

{

PersonData = new Person(person);

}

~Linked\_PersonPointers()

{

delete PersonData;

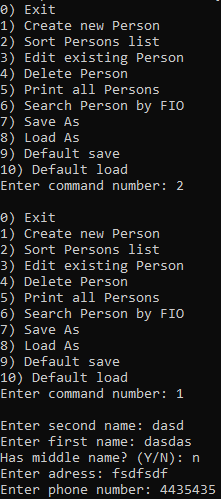
}

};

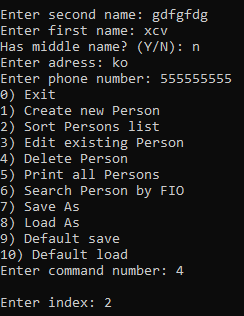
#endif

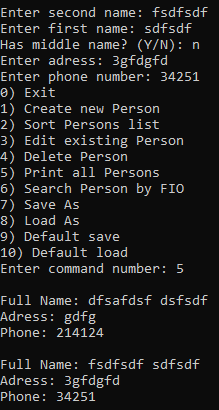
* 1. **Тестирование**

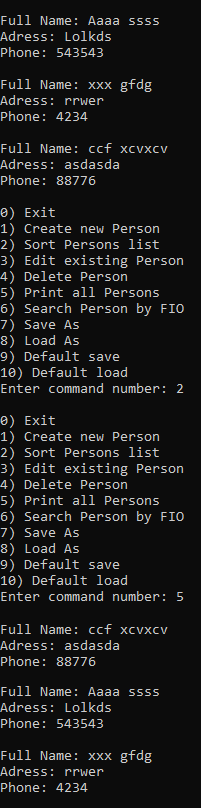
При добавлении записи будет срабатывать конструктор в которые подадутся введённые параметры и объект создастся с уже заполненными полями, а не как это было раньше.

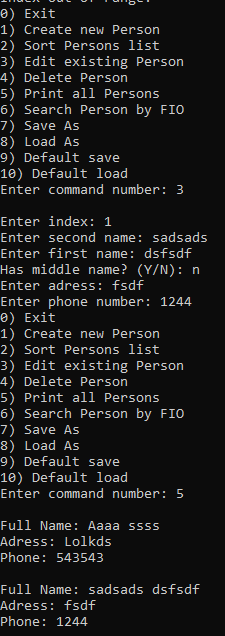


Деструктор срабатывает в конце работы программы, либо при операциях удаления.



Остальной функционал программы работает корректно 





**Заключение**

В данной работе я проверил свои знания и умения в ООП. Также закрепил их переписав прошлую лабораторную работу.