ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

МОСКОВСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Факультет Информационных технологий
Кафедра Информатики и информационных технологий

направление подготовки
09.03.02 «Информационные системы и технологии»

ЛАБОРАТОРНАЯ РАБОТА № 2

Дисциплина: «Основы современных алгоритмов»

Тема: «Разработка программы сохранения, поиска и управления данными с использованием структур и файлового ввода вывода»

Выполнил: студент группы 211-723

Сергеев Станислав Олегович

Дата, подпись_____ _____
(Дата) (Подпись)

Проверил:_____ _____
(Фамилия И.О., степень, звание) **(Оценка)**

Дата, подпись_____ _____
(Дата) (Подпись)

Замечания: _____

_____

_____

Москва

2022

Программное моделирование стеков и очередей

Цель:

Получить практические навыки в использовании структурных типов данных в языке С.

2. Постановка задачи

Разработать программу ведения и использование базы данных с использованием структурных переменных и сохранением данных в плоском (без использования форматирования) файле.

Для этого создать шаблон структуры, создать набор функций для работы с массивом структурных переменных. В структурной переменной предусмотреть способ отметки ее как не содержащей данных (т.е. "пустой"). Разработать функции в соответствии с ниже приводимым перечнем:

1. “Очистка” структурных переменных (отметка переменной как не содержащей данных). Функция должна получать в качестве параметра индекс массива.
2. Поиск свободной структурной переменной. Функция должна возвращать индекс первого свободного элемента в массиве структур.
3. Ввод элементов (полей) структуры с клавиатуры. В качестве параметра в функцию должен передаваться индекс элемента массива.
4. Вывод элементов (полей) структуры на монитор. В функцию должен передаваться индекс элемента массива.
5. Вывод на экран всех заполненных элементов массива структур.
6. Поиск в массиве структур элемента с заданным значением поля или с наиболее близким к нему по значению. Предусмотреть возможность задания одного числового значения и одного строкового.
7. Поиск в массиве структуры с минимальным значением заданного поля. Поле должно передаваться в поле числом, обозначающим номер поля в структуре.
8. Сортировка массива структур в порядке возрастания заданного поля (при сортировке можно использовать тот факт, что в Си++ разрешается присваивание структурированных переменных);
9. Сортировка массива структур в порядке убывания заданного поля (при сортировке можно использовать тот факт, что в Си++ разрешается присваивание структурированных переменных);
10. Функция чтения файла с содержимым базы данных (массива структурных элементов).
11. Функция записи в файл содержимого базы данных (массива структурных элементов).

Функции должны работать с массивом структурных переменных или с отдельной структурной переменной через указатели, а также при необходимости возвращать указатель на структурную переменную или массив структурных переменных.

12 - ВАРИАНТ

12. Фамилия И.О., количество переговоров (для каждого - дата и продолжительность).

```
#include <iostream>
#include <stdio.h>
#include <string>
#include <cmath>
using namespace std;

struct Data
{
    char surname[20];
```

```
    char name[15];
    char lastname[25];
    int count;
    int day;
    int month;
    int year;
    int duration;
    bool isDeleted;
};

class Database
{
private:
    Data* _dataArray;
    long _databaseSize;
    char* _databaseName;

    void swap(Data* xp, Data* yp)
    {
        Data temp = *xp;
        *xp = *yp;
        *yp = temp;
    }

public:
    Database(string databaseName, long length)
    {
        _databaseName = new char[databaseName.length() + 1];
        strcpy_s(_databaseName, databaseName.size()+1, databaseName.c_str());
        _dataArray = new Data[length];
        _databaseSize = length;

        for (int i = 0; i < _databaseSize; i++)
            _dataArray[i].isDeleted = true;
    }

    ~Database()
    {
        delete[] _dataArray;
    }

    void Delete(int index) // Очистка структурных переменных (отметка переменной как не
содержащей данных). Функция должна получать в качестве параметра индекс массива.
    {
        _dataArray[index].isDeleted = true;
    }

    int FindEmpty() // Поиск свободной структурной переменной. Функция должна возвращать
индекс первого свободного элемента в массиве структур.
    {
        for (int i = 0; i < _databaseSize; i++)
            if (_dataArray[i].isDeleted)
                return i;
        return -1;
    }

    void Add(int index, Data data) // Ввод элементов (полей) структуры с клавиатуры. В
качестве параметра в функцию должен передаваться индекс элемента массива.
    {
        _dataArray[index] = data;
        _dataArray[index].isDeleted = false;
    }

    Data Get(int index) // Вывод элементов (полей) структуры на монитор. В функцию должен
передаваться индекс элемента массива.
    {
```

```
        return _dataArray[index];
    }

    Data* FindNearestDay(int day) //Поиск в массиве структур элемента с заданным значением поля или с наиболее близким к нему по значению. Предусмотреть возможность задания одного числового значения и одного строкового.
    {
        int indexOfNearestElement = -1;
        int minimalDiff = INT_MAX;

        for (int i = 0; i < _databaseSize; i++)
            if (!_dataArray[i].isDeleted) {
                int valueDiff = abs(_dataArray[i].day - day);
                if (valueDiff < minimalDiff)
                {
                    minimalDiff = valueDiff;
                    indexOfNearestElement = i;
                }
            }
        if (indexOfNearestElement == -1)
            return (Data*)NULL;
        else
            return &_dataArray[indexOfNearestElement];
    }

    Data* FindNearestMonth(int month)
    {
        int indexOfNearestElement = -1;
        int minimalDiff = INT_MAX;

        for (int i = 0; i < _databaseSize; i++)
            if (!_dataArray[i].isDeleted) {
                int valueDiff = abs(_dataArray[i].month - month);
                if (valueDiff < minimalDiff)
                {
                    minimalDiff = valueDiff;
                    indexOfNearestElement = i;
                }
            }
        if (indexOfNearestElement == -1)
            return (Data*)NULL;
        else
            return &_dataArray[indexOfNearestElement];
    }

    Data* FindNearestYear(int year)
    {
        int indexOfNearestElement = -1;
        int minimalDiff = INT_MAX;

        for (int i = 0; i < _databaseSize; i++)
            if (!_dataArray[i].isDeleted) {
                int valueDiff = abs(_dataArray[i].year - year);
                if (valueDiff < minimalDiff)
                {
                    minimalDiff = valueDiff;
                    indexOfNearestElement = i;
                }
            }
        if (indexOfNearestElement == -1)
            return (Data*)NULL;
        else
            return &_dataArray[indexOfNearestElement];
    }

    Data* FindNearestCount(int count)
```

```
    {
        int indexOfNearestElement = -1;
        int minimalDiff = INT_MAX;

        for (int i = 0; i < _databaseSize; i++)
            if (!_dataArray[i].isDeleted) {
                int valueDiff = abs(_dataArray[i].count - count);
                if (valueDiff < minimalDiff)
                {
                    minimalDiff = valueDiff;
                    indexOfNearestElement = i;
                }
            }
        if (indexOfNearestElement == -1)
            return (Data*)NULL;
        else
            return &_dataArray[indexOfNearestElement];
    }

    Data* FindNearestDuration(int duration)
    {
        int indexOfNearestElement = -1;
        int minimalDiff = INT_MAX;

        for (int i = 0; i < _databaseSize; i++)
            if (!_dataArray[i].isDeleted) {
                int valueDiff = abs(_dataArray[i].duration - duration);
                if (valueDiff < minimalDiff)
                {
                    minimalDiff = valueDiff;
                    indexOfNearestElement = i;
                }
            }
        if (indexOfNearestElement == -1)
            return (Data*)NULL;
        else
            return &_dataArray[indexOfNearestElement];
    }

    Data* FindNearestName(string name)
    {
        for (int i = 0; i < _databaseSize; i++)
            if (!_dataArray[i].isDeleted && string(_dataArray[i].name).find(name) !=
string::npos)
                return &_dataArray[i];
        return NULL;
    }

    Data* FindNearestSurname(string surname)
    {
        for (int i = 0; i < _databaseSize; i++)
            if (!_dataArray[i].isDeleted && string(_dataArray[i].surname).find(surname) !=
string::npos)
                return &_dataArray[i];
        return NULL;
    }

    Data* FindNearestLastName(string lastname)
    {
        for (int i = 0; i < _databaseSize; i++)
            if (!_dataArray[i].isDeleted && string(_dataArray[i].lastname).find(lastname) !=
string::npos)
                return &_dataArray[i];
        return NULL;
    }
```

```
    Data* findMin(bool (*f)(Data, Data)) //Поиск в массиве структуры с минимальным значением заданного поля. Поле должно передаваться в поле числом, обозначающим номер поля в структуре.
    {
        int indexOfMin = -1;
        Data min = _dataArray[0];

        for (int i = 0; i < _databaseSize; i++)
        {
            if (!_dataArray[i].isDeleted && f(min, _dataArray[i]))
            {
                min = _dataArray[i];
                indexOfMin = i;
            }
        }

        if (indexOfMin == -1)
            return (Data*)NULL;
        else
            return &_dataArray[indexOfMin];
    }

    void SortBy(bool (*f)(Data, Data))
    {
        for (int i = 0; i < _databaseSize - 1; i++)
            for (int j = 0; j < _databaseSize - i - 1; j++)
                if (f(_dataArray[j], _dataArray[j + 1]))
                    swap(&_dataArray[j], &_dataArray[j + 1]);
    }

    void Restore()
    {
        FILE* file;
        fopen_s(&file, _databaseName, "rb");

        for (int i = 0; i < _databaseSize; i++)
            fread(&_dataArray[i], sizeof(Data), 1, file);

        fclose(file);
    }

    void Save()
    {
        FILE* file;
        fopen_s(&file, _databaseName, "wb");

        for (int i = 0; i < _databaseSize; i++)
            fwrite(&_dataArray[i], sizeof(Data), 1, file);

        fclose(file);
    }

    Data print(int index);

};

Data Database :: print(int index)
{
    return _dataArray[index];
}

void Get(Data data)
{
    cout << data.surname << " " << data.name << " " << data.lastname << " " << data.count << " " << data.day << "." << data.month << "." << data.year << " " << data.duration << endl;
}
```

```
void Get(Data* data)
{
    cout << data->surname << " " << data->name << " " << data->lastname << " " << data->count
<< " " << data->day << "." << data->month << "." << data->year << " " << data->duration <<
endl;
}

int main()
{
    setlocale(LC_ALL, "Russian");
    int databaseSize = 3;
    string databaseName = "12var.bin";

    Database* database = new Database(databaseName, databaseSize);

    for (int i = 0; i < databaseSize; i++) {
        char ch = '.';
        Data data;
        cout << "ФИО: ";
        cin >> data.surname >> data.name >> data.lastname;
        cout << "Количество переговоров: ";
        cin >> data.count;
        cout << "Дата переговоров через точки: ";
        cin >> data.day >> ch >> data.month >> ch >> data.year;
        cout << "Длительность переговоров: ";
        cin >> data.duration;
        database->Add(i, data);
    }
    database->Save();

    delete database;
    database = new Database(databaseName, databaseSize);
    database->Restore();

    for (int i = 0; i < databaseSize; i++) {
        Data data = database->print(i);
        Get(data);
    }

    int nearestInt;
    string nearestString;
    Data* nString;
    Data* nInt;
    int field;
    cout << "\nВведите номер поля: ";
    cin >> field;
    switch (field)
    {
    case 1:
        cout << "Введите фамилию: ";
        cin >> nearestString;
        nString = database->FindNearestSurname(nearestString);
        if (nString == NULL)
            cout << "Ошибка" << endl;
        else
        {
            cout << "Ближайшее по фамилии" << endl; Get(nString);
        }
        break;
    case 2:
        cout << "Введите Имя: ";
        cin >> nearestString;
        nString = database->FindNearestName(nearestString);
        if (nString == NULL)
            cout << "Ошибка" << endl;
```

```
    else
    {
        cout << "Ближайшее по имя" << endl; Get(nString);
    }
    break;
case 3:
    cout << "Введите отчество: ";
    cin >> nearestString;
    nString = database->FindNearestLastName(nearestString);
    if (nString == NULL)
        cout << "Ошибка" << endl;
    else
    {
        cout << "Ближайшее по отчеству" << endl; Get(nString);
    }
    break;
case 4:
    cout << "Введите количество переговоров: ";
    cin >> nearestInt;
    nInt = database->FindNearestCount(nearestInt);
    if (nInt == NULL)
        cout << "Ошибка" << endl;
    else
    {
        cout << "Ближайшее по количеству" << endl; Get(nInt);
    }
    break;
case 5:
    cout << "Введите день: ";
    cin >> nearestInt;
    nInt = database->FindNearestDay(nearestInt);
    if (nInt == NULL)
        cout << "Ошибка" << endl;
    else
    {
        cout << "Ближайшее по дню" << endl; Get(nInt);
    }
    break;
case 6:
    cout << "Введите месяц: ";
    cin >> nearestInt;
    nInt = database->FindNearestMonth(nearestInt);
    if (nInt == NULL)
        cout << "Ошибка" << endl;
    else
    {
        cout << "Ближайшее по месяцу" << endl; Get(nInt);
    }
    break;
case 7:
    cout << "Введите год: ";
    cin >> nearestInt;
    nInt = database->FindNearestYear(nearestInt);
    if (nInt == NULL)
        cout << "Ошибка" << endl;
    else
    {
        cout << "Ближайшее по году" << endl; Get(nInt);
    }
    break;
case 8:
    cout << "Введите длительность: ";
    cin >> nearestInt;
    nInt = database->FindNearestDuration(nearestInt);
    if (nInt == NULL)
        cout << "Ошибка" << endl;
```

```
        else
        {
            cout << "Ближайшее по длительности" << endl; Get(nInt);
        }
        break;
    }

    int min;
    cout << "\n Введите номер поля: ";
    cin >> min;
    Data* minData;
    switch (min)
    {
    case 1:
        cout << "Min по фамилии" << endl;
        minData = database->findMin([](Data p1, Data p2) { return string(p1.surname) >
string(p2.surname); });
        Get(minData);
        break;
    case 2:
        cout << "Min по имени" << endl;
        minData = database->findMin([](Data p1, Data p2) { return string(p1.name) >
string(p2.name); });
        Get(minData);
        break;
    case 3:
        cout << "Min по отчеству" << endl;
        minData = database->findMin([](Data p1, Data p2) { return string(p1.lastname) >
string(p2.lastname); });
        Get(minData);
        break;
    case 4:
        cout << "Min по количеству" << endl;
        minData = database->findMin([](Data p1, Data p2) { return p1.count > p2.count; });
        Get(minData);
        break;
    case 5:
        cout << "Min по дню" << endl;
        minData = database->findMin([](Data p1, Data p2) { return p1.day > p2.day; });
        Get(minData);
        break;
    case 6:
        cout << "Min по месяцу" << endl;
        minData = database->findMin([](Data p1, Data p2) { return p1.month > p2.month; });
        Get(minData);
        break;
    case 7:
        cout << "Min по году" << endl;
        minData = database->findMin([](Data p1, Data p2) { return p1.year > p2.year; });
        Get(minData);
        break;
    case 8:
        cout << "Min по длительности" << endl;
        minData = database->findMin([](Data p1, Data p2) { return p1.duration > p2.duration;
});
        Get(minData);
        break;
    }

    cout << "\nСортировка по возрастанию по количеству" << endl;
    database->SortBy([](Data p1, Data p2) { return p1.count > p2.count; });
    for (int i = 0; i < databaseSize; i++) {
        Data a = database->print(i);
        Get(a);
    }
```

```
    cout << "\nСортировка по убыванию по длительности" << endl;
    database->SortBy([](Data p1, Data p2) { return p1.duration < p2.duration; });
    for (int i = 0; i < databaseSize; i++) {
        Data a = database->print(i);
        Get(a);
    }

    return 0;
}
```