МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«МИРЭА – Российский технологический университет»

Отчет

Практическая работа №9

Дисциплина Структуры и алгоритмы обработки данных

Тема. Алгоритмы поиска в таблице (массиве)

Выполнил студент Дамарад.Д.В.

Фамилия И.О.

Группа ИКБО-13-21

Номер группы

Задание №1

1. Постановка задачи:

Разработать программу поиска записи по ключу в таблице записей с применение двух алгоритмов линейного поиска

2. Описание подхода к решению:

В линейном поиске сравниваем каждый элемент массива с ключом и смотрим за выходом из массива, если элемент массива совпадает с ключом, то возвращаем его объект. Если ключ не найден в массиве, то возвращаем объект с нулевыми полями.

В линейном поиске с барьером добавляем искомый ключ за рамки массива, затем сравниваем текущий элемент массива с ключом, если совпадает, то возвращаем его объект.

Структура записи:

```
struct Patient {
   Patient() {

   }
   Patient(int card_number, int chronic_disease_code, string
surname_of_the_doctor) {
      this->card_number = card_number;
      this->chronic_disease_code = chronic_disease_code;
      this->surname_of_the_doctor = surname_of_the_doctor;
   }
   int card_number=0; //ключ
   int chronic_disease_code=0;
   string surname_of_the_doctor = "-";
};
```

3. Функция линейного поиска:

Предусловие: массив table типа Patient.

Постусловие: объект, поле которого содержит требуемый ключ.

```
Patient FindLineal(int key, Patient* table, long int size) {
   Patient nullpatient;
   for (int i = 0; i < size; i++) {
      if (table[i].card_number == key) {
```

```
            return table[i];
        }
    }
    return nullpatient;
}
```

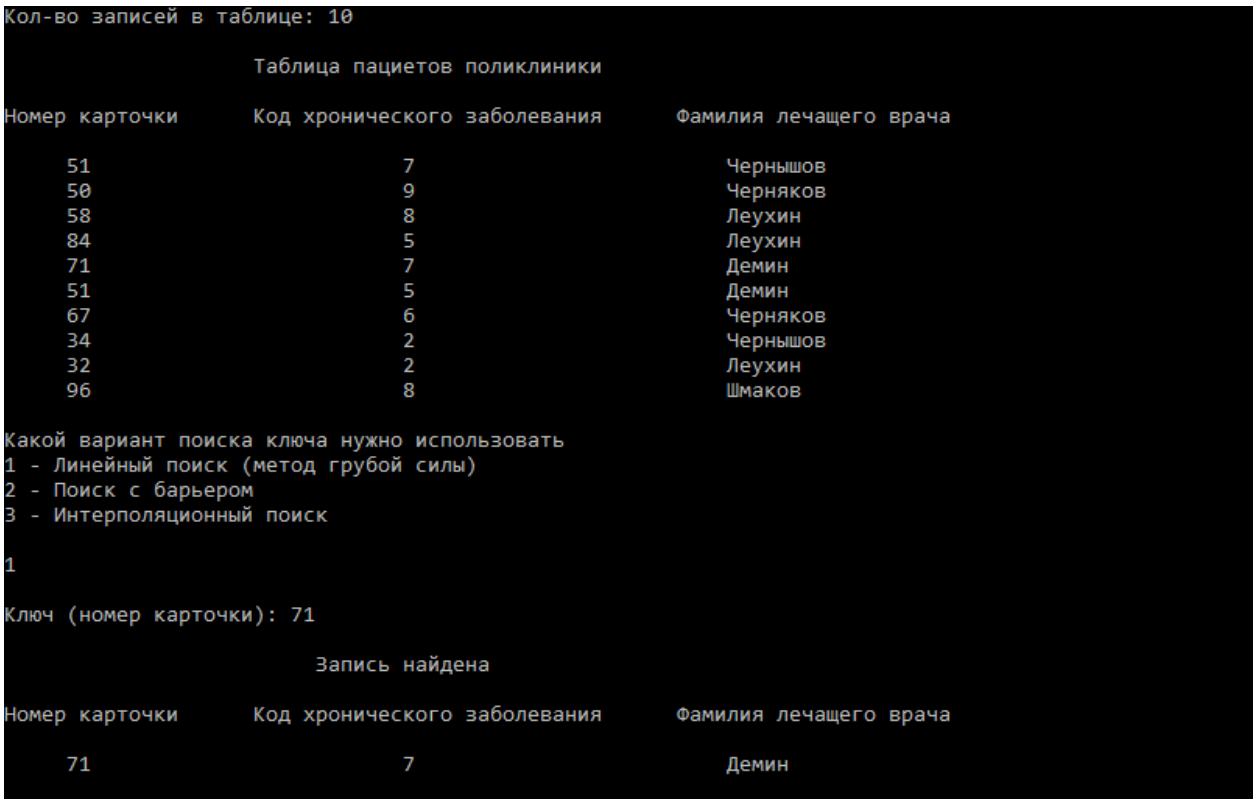
Функция линейного поиска с барьером:

Предусловие: массив table типа Patient.

Постусловие: объект, поле которого содержит требуемый ключ.

```
Patient FindWithBarier(int key, Patient* table, long int size) {
    table[size].card_number = key;
    int i = 0;
    while (table[i].card_number != key) {
        i++;
    }
    return table[i];
}
```

4. Скриншоты работы программы:



```
Кол-во записей в таблице: 10

                    Таблица пациетов поликлиники

Номер карточки      Код хронического заболевания      Фамилия лечащего врача

     51                          7                          Чернышов
     50                          9                          Черняков
     58                          8                          Леухин
     84                          5                          Леухин
     71                          7                          Демин
     51                          5                          Демин
     67                          6                          Черняков
     34                          2                          Чернышов
     32                          2                          Леухин
     96                          8                          Шмаков

Какой вариант поиска ключа нужно использовать
1 - Линейный поиск (метод грубой силы)
2 - Поиск с барьером
3 - Интерполяционный поиск

2

Ключ (номер карточки): 96

                        Запись найдена

Номер карточки      Код хронического заболевания      Фамилия лечащего врача

     96                          8                          Шмаков
```

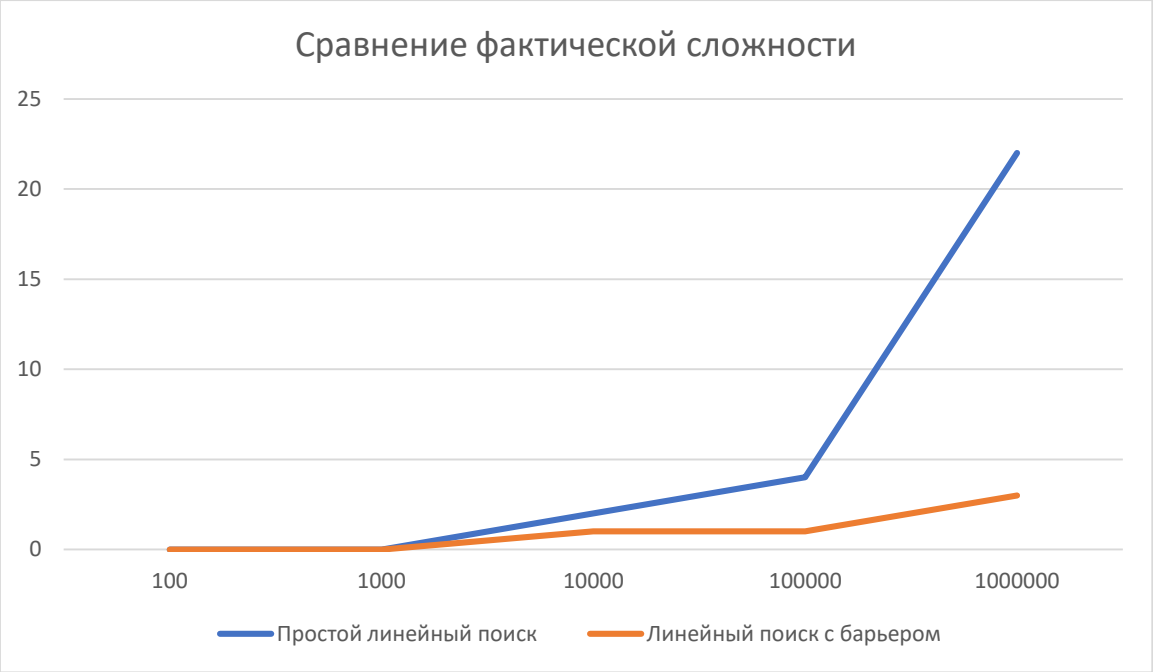
5. Таблица результатов для линейного поиска:

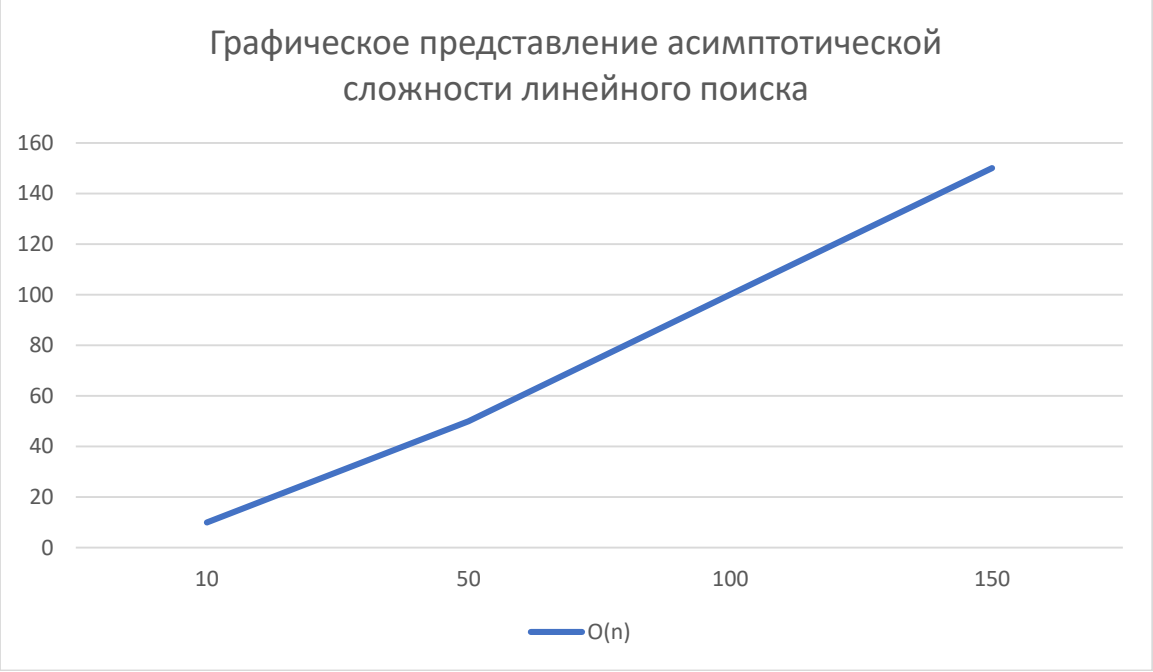
n	**T**	**$T_т$=f(C+M)**	**$T_п$=Cф+Mф**
100	0 мс	203	200
1000	0 мс	2003	2000
10000	2 мс	20003	20000
100000	4 мс	200003	200000
1000000	22 мс	2000003	2000000

Таблица результатов для линейного поиска с барьером:

n	**T**	**$T_т$=f(C+M)**	**$T_п$=Cф+Mф**
100	0 мс	102	100
1000	0 мс	1002	1000
10000	1 мс	10002	10000
100000	1 мс	100002	100000
1000000	3 мс	1000002	1000000

6. Графики зависимости времени выполнения от объема выполняемых данных





7. В ходе работы с линейными поисками была выявлена асимптотическая сложность $O(n)$, показана важность меньшего количества критических операций.

Задание №2

1. Постановка задачи:

Разработать программу поиска записи по ключу в таблице записей с применением алгоритма, определенного в задании варианта.

2. Описание подхода к решению:

На основе данных о ключе, левой и правой позиции, а также их значениях, «предугадывается» возможное местоположение ключа и поиск продолжает производиться с угаданного индекса.

3. Алгоритм на псевдокоде:

```
interpolationSearch(int key, Patient* table, int size) {
   Patient nullpatient
   int low ← 0
   int high ← size - 1
   int mid
   пока table[low].card_number < key и table[high].card_number >= key {
      mid ← low + ((key - table[low].card_number) * (high - low)) /
(table[high].card_number - table[low].card_number)
      if table[mid].card_number < key
         low ← mid + 1
      else if table[mid].card_number > key
         high ← mid - 1
      else
         возврат table[mid]
   }
   if table[low].card_number = key
      возврат table[low]
   else
      возврат объекта с нулевыми полями
}
```

4. Код функции:

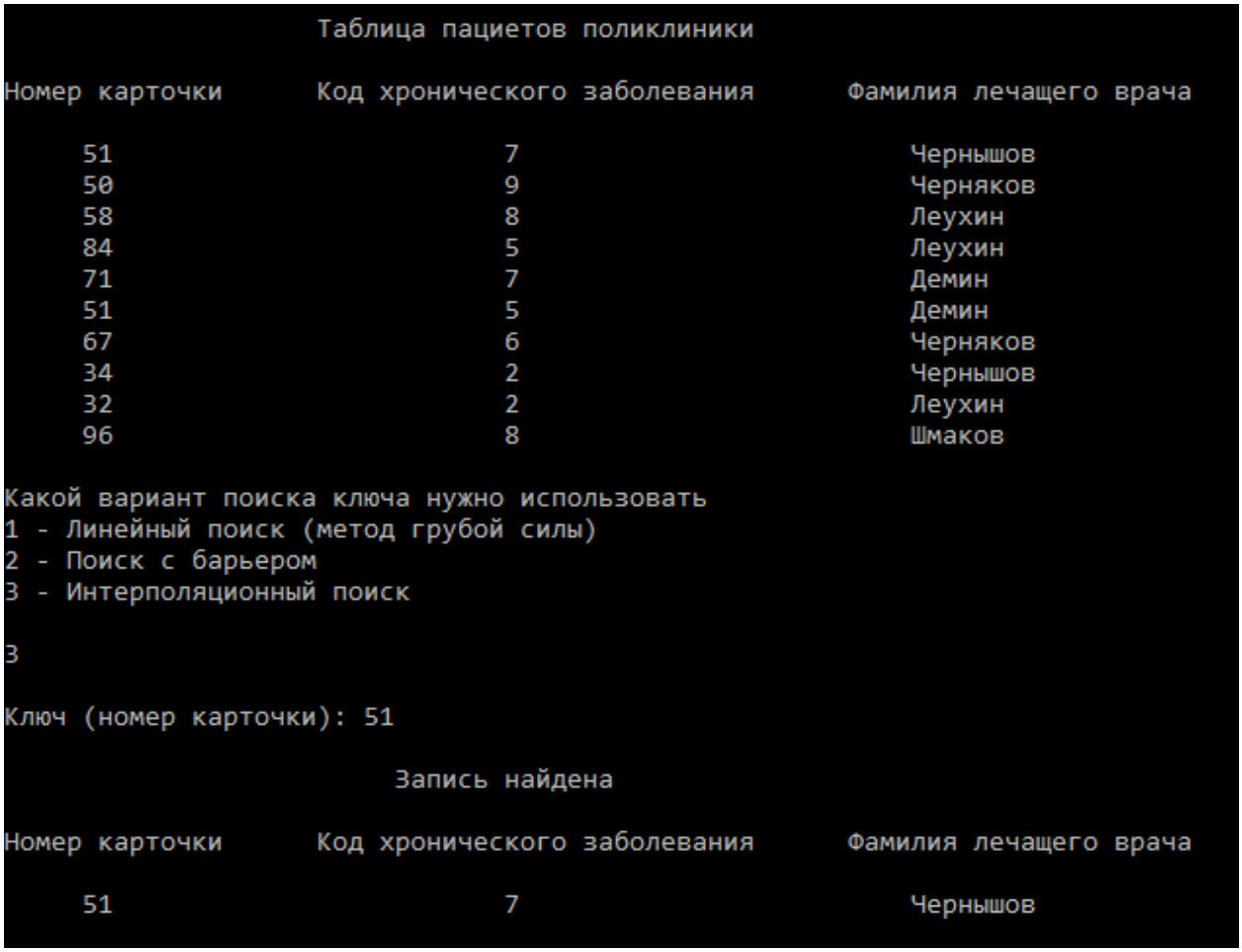
Предусловие: массив table типа Patient.

Постусловие: объект, поле которого содержит требуемый ключ.

```
Patient interpolationSearch(int key, Patient* table, int size) {
   Patient nullpatient;
   int low = 0;
   int high = size - 1;
   int mid;
```

```
    while (table[low].card_number < key && table[high].card_number >= key) {
        mid = low + ((key - table[low].card_number) * (high - low)) /
(table[high].card_number - table[low].card_number);
        if (table[mid].card_number < key)
            low = mid + 1;
        else if (table[mid].card_number > key)
            high = mid - 1;
        else
            return table[mid];
    }
    if (table[low].card_number == key)
        return table[low];
    else
        return nullpatient;
}
```

5. Скриншоты работы программы:

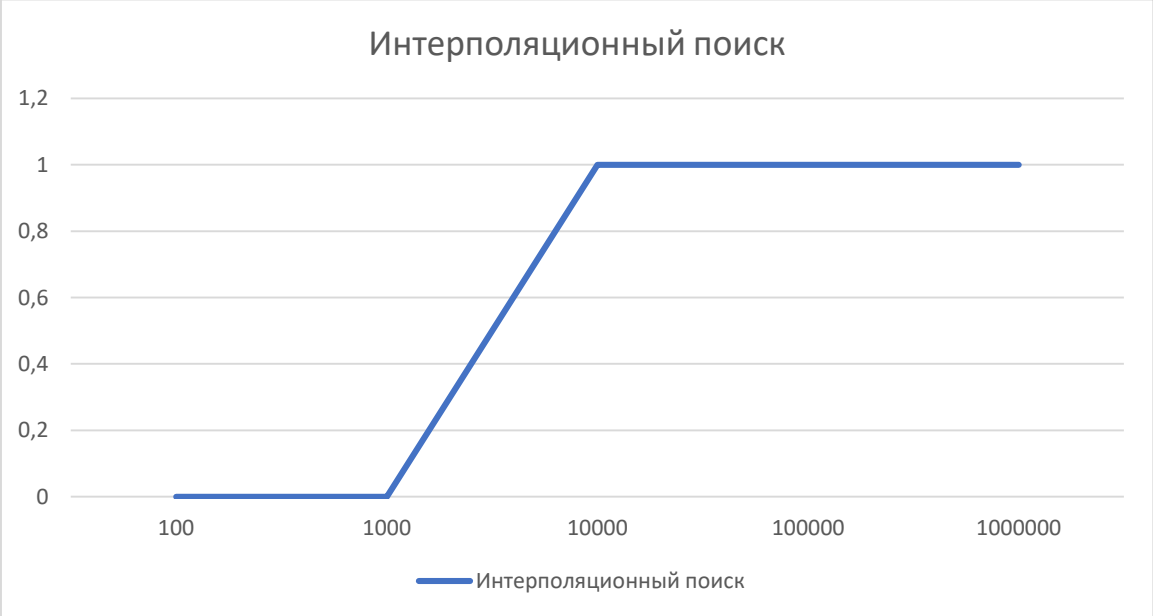


```
                    Таблица пациетов поликлиники

Номер карточки      Код хронического заболевания      Фамилия лечащего врача

     51                          7                          Чернышов
     50                          9                          Черняков
     58                          8                          Леухин
     84                          5                          Леухин
     71                          7                          Демин
     51                          5                          Демин
     67                          6                          Черняков
     34                          2                          Чернышов
     32                          2                          Леухин
     96                          8                          Шмаков

Какой вариант поиска ключа нужно использовать
1 - Линейный поиск (метод грубой силы)
2 - Поиск с барьером
3 - Интерполяционный поиск

3

Ключ (номер карточки): 1

                        Запись не найдена
```

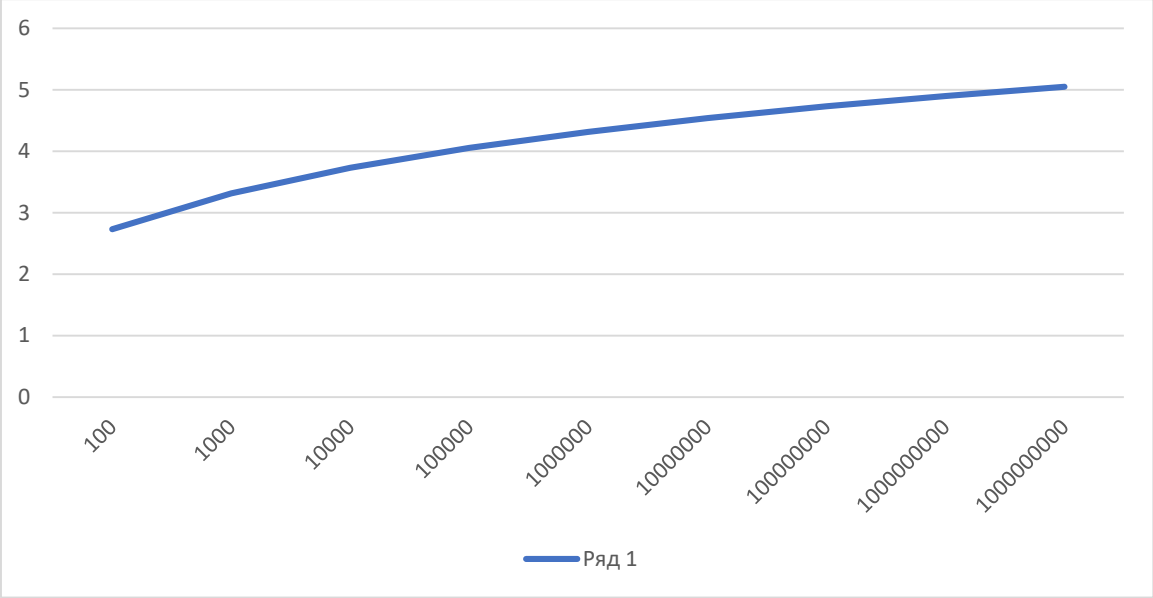
6. Таблица результатов для интерполяционного поиска:

n	T	$T_т=f(C+M)$	$T_п=Cф+Mф$
100	0 мс	5	2
1000	0 мс	6	2
10000	1 мс	6	2
100000	1 мс	7	3
1000000	1 мс	7	4

7. Графическое представление функции:



Графическое представление асимптотической сложности алгоритма:



8. Вывод: данный алгоритм очень эффективен при работе с большим количеством данных, распределенными равномерно.

9. Результаты сравнительного анализа:

Наиболее выгодным оказывается интерполяционный поиск, средним является линейный поиск с барьером, самым худшим является линейный поиск.

10. Полный код программы:

```
#include <iostream>
#include <string>
#include <cstring>
#include <chrono>
#include <iomanip>
using namespace std;

struct Patient {
    Patient() {

    }
    Patient(int card_number, int chronic_disease_code, string
surname_of_the_doctor) {
        this->card_number = card_number;
        this->chronic_disease_code = chronic_disease_code;
        this->surname_of_the_doctor = surname_of_the_doctor;
    }
    int card_number=0; //ключ
    int chronic_disease_code=0;
    string surname_of_the_doctor = "-";
};

void Fill(Patient* table, long int size) {
    //Фамилии врачей
    string surnames[7] = { "Демин","Шмаков","Черняков", "Ботоев",
"Хоров", "Леухин", "Чернышов" };
    for (int i = 0; i < size; i++) {
        table[i].card_number = 10+rand() % 90;
        table[i].chronic_disease_code = rand() % 10;
        table[i].surname_of_the_doctor = surnames[rand() % 7];
    }
}

void Print(Patient* table, long int size) {
    cout << endl << setw(20) << "  " << "Таблица пациетов поликлиники" <<
setw(25) << "  " << endl << endl;
    cout << "Номер карточки " << setw(5) << " " << "Код хронического
заболевания " << setw(5) << " " << "Фамилия лечащего врача" << endl <<
endl;
    for (long int i = 0; i < size; i++) {
        cout << setw(5) << " "<<table[i].card_number << setw(25) << " " <<
table[i].chronic_disease_code << setw(25) << " " <<
table[i].surname_of_the_doctor << endl;
```

```
    }
}

void PrintOne(Patient obj) {
    if (obj.card_number == 0) {
        cout << setw(25) << "  " << "Запись не найдена" << setw(25) << "  " <<
endl << endl;
    }
    else {
        cout << setw(25) << "  " << "Запись найдена" << setw(25) << "  " <<
endl << endl;
        cout << "Номер карточки " << setw(5) << " " << "Код хронического
заболевания " << setw(5) << " " << "Фамилия лечащего врача" << endl <<
endl;
        cout << setw(5) << " " << obj.card_number << setw(25) << " " <<
obj.chronic_disease_code << setw(25) << " " << obj.surname_of_the_doctor
<< endl;
    }
}

Patient FindLineal(int key, Patient* table, long int size) {
    Patient nullpatient;
    for (int i = 0; i < size; i++) {
        if (table[i].card_number == key) {
            return table[i];
        }
    }
    return nullpatient;
}

Patient FindWithBarier(int key, Patient* table, long int size) {
    table[size].card_number = key;
    int i = 0;
    while (table[i].card_number != key) {
        i++;
    }
    return table[i];

}

Patient interpolationSearch(int key, Patient* table, int size) {
    Patient nullpatient;
    //объявляем необходимые локальные переменные
    //изначально устанавливаем нижний индекс на начало массива, а верний
на конец массива
```

```
    int low = 0;
    int high = size - 1;
    int mid;
    //цикл интерполирующего поиска
    while (table[low].card_number < key && table[high].card_number >= key) {
        //интерполирующий поиск производит оценку новой области поиска
        //по расстоянию между ключом поиска и текущим значение элемента
        mid = low + ((key - table[low].card_number) * (high - low)) /
(table[high].card_number - table[low].card_number);
        //если значение в ячейке с индексом mid меньше, то смещаем
нижнюю границу
        if (table[mid].card_number < key)
            low = mid + 1;
        //в случае, если значение больше, то смещаем верхнюю границу
        else if (table[mid].card_number > key)
            high = mid - 1;
        //если равны, то возвращаем индекс
        else
            return table[mid];
    }
    //если цикл while не вернул индекс искомого значения,
    //то проверяем не находится ли оно в ячейке массива с индексом low,
    //иначе возвращаем -1 (значение не найдено)
    if (table[low].card_number == key)
        return table[low];
    else
        return nullpatient;
}

int main() {
    system("chcp 1251 > null");
    long int size; cout << "Кол-во записей в таблице: "; cin >> size;
    Patient* table=new Patient[size];
    Fill(table,size);
    Print(table, size);
    cout << endl;
    int num; cout << "Какой вариант поиска ключа нужно использовать" <<
endl;
    cout << "1 - Линейный поиск (метод грубой силы)" << endl;
    cout << "2 - Поиск с барьером" << endl;
    cout << "3 - Интерполяционный поиск" << endl << endl;
    cin >> num;
    int key;
    switch (num) {
    case 1:{
```

```
            cout << endl << "Ключ (номер карточки): "; cin >> key; cout << endl;
            PrintOne(FindLineal(key, table, size));
            break;
        }
        case 2: {
            cout << endl << "Ключ (номер карточки): "; cin >> key; cout << endl;
            PrintOne(FindWithBarier(key, table, size));
            break;
        }
        case 3: {
            cout << endl << "Ключ (номер карточки): "; cin >> key; cout << endl;
            PrintOne(interpolationSearch(key, table, size));
            break;
        }
        default: {
            cout << "Нет такого номера варианта поиска ключа" << endl;
            break;
        }
        }
        return 0;
}
```