

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ
ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение
высшего образования

«Тихоокеанский государственный университет»

Кафедра «Программное обеспечение вычислительной техники и
автоматизированных систем»

Изучение структур данных и алгоритмов работы с ними

Текстовый документ курсовой работы

по дисциплине «Структуры и алгоритмы обработки данных»

КР. 180003518.ТД

Выполнил студент

Чекулаев В. Ю.

Факультет, группа

ФКФН, ПО(аб)-81

Руководитель работы

Бахрушина Г. И.

Виза: _____

(доработать, к защите и т.д.)

Хабаровск – 2020г.

Оглавление

ВВЕДЕНИЕ.....	3
1.Постановка задачи.....	4
2.Описание алгоритмов сортировки, поиска(в соответствии с вариантом).....	5
2.1 Сортировка методом Вильямса-Флойда(быстрая сортировка).....	5
2.2 Двоичный поиск.....	7
3. Описание структуры программы.....	8
3.1 Описание функций и структур данных:.....	8
ЗАКЛЮЧЕНИЕ.....	11
СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ.....	12
ПРИЛОЖЕНИЕ А.....	13
ПРИЛОЖЕНИЕ Б.....	19

ВВЕДЕНИЕ

Структура — это совокупность переменных, объединенных одним именем, предоставляющая общепринятый способ совместного хранения информации.

Структуру стоит использовать, если нам необходима совокупность переменных с разными типами под одним именем. Это делает программу более компактной и более удобной для внесения изменений в нее.

По заданию файл базы данных загружается в память в виде массива.

Массив - это структура данных, представленная в виде группы ячеек одного типа, объединенных под одним единым именем.

В результате поиска формируется очередь.

Очередь - абстрактный тип данных с дисциплиной доступа к элементам «первый пришёл — первый вышел». Добавление элемента возможно лишь в конец очереди, выборка — только из начала очереди, при этом выбранный элемент из очереди удаляется.

1. Постановка задачи (в соответствии с вариантом)

Хранящуюся в текстовом файле базу данных "Населенный пункт" загрузить динамически в оперативную память компьютера в виде массива и вывести на экран по 20 записей (строк) на странице с возможностью отказа от просмотра. Текстовый файл создать самостоятельно.

Упорядочить данные по дате поселения и названию улицы, используя **метод Вильямса-Флойда**. Упорядоченные данные вывести на экран.

Реализовать возможность быстрого поиска по ключу (**ключ – год поселения**) в упорядоченной базе, в результате которого из записей с одинаковым ключом формируется очередь, содержимое очереди выводится на экран.

2. Описание алгоритмов сортировки, поиска (в соответствии с вариантом) /1,2/ 2.1 Сортировка методом Вильямса-Флойда (Пирамидальная сортировка)

Метод сортировки массива, предложенный и развитый Вильямсом и Флойдом, носит название алгоритма "пирамиды". Он основан на специальном представлении массива в форме бинарного дерева, обладающего особыми свойствами и называемого "пирамидой". Алгоритм имеет гарантированную трудоемкость вида $O(n \log n)$ и не требует дополнительной памяти.

Высокая эффективность алгоритма и гарантированная надежность для самого «худшего» случая часто оказываются решающими факторами, заставляющими отдавать предпочтение этому способу сортировки.

Алгоритм включает в себя два этапа:

1. преобразование массива к виду «пирамиды» ;
2. рекурсивная сортировка «пирамиды».

Назовем пирамидой (Неар) бинарное дерево высоты k , в котором:

- все узлы имеют глубину k или $k-1$ - дерево сбалансированное.
- при этом уровень $k-1$ полностью заполнен, а уровень k заполнен слева направо
- выполняется "свойство пирамиды": каждый элемент меньше, либо равен родителю.

Хранить пирамиду удобнее всего с помощью массива. Соответствие между геометрической структурой пирамиды как дерева и массивом устанавливается по следующей схеме:

- в $a[0]$ хранится корень дерева

- левый и правый сыновья элемента $a[i]$ хранятся, соответственно, в $a[2i+1]$ и $a[2i+2]$

Таким образом, для массива, хранящего в себе пирамиду, выполняется следующее характеристическое свойство: $a[i] \geq a[2i+1]$ и $a[i] \geq a[2i+2]$.

Начать построение пирамиды можно с $a[k]..a[n]$, $k = \lfloor \text{size}/2 \rfloor$. Эта часть массива удовлетворяет свойству пирамиды, так как не существует индексов i, j : $i = 2i+1$ (или $j = 2i+2$). Просто потому, что такие i, j находятся за границей массива.

Далее будем расширять часть массива, добавляя по одному элементу за шаг. Следующий элемент на каждом шаге добавления - тот, который стоит перед уже готовой частью.

Чтобы при добавлении элемента сохранялась пирамидальность, будем использовать следующую процедуру расширения пирамиды $a[i+1]..a[n]$ на элемент $a[i]$ влево:

1. Смотрим на сыновей слева и справа - в массиве это $a[2i+1]$ и $a[2i+2]$ и выбираем наибольшего из них.
2. Если этот элемент больше $a[i]$ - меняем его с $a[i]$ местами и идем к шагу 1, имея в виду новое положение $a[i]$ в массиве. Иначе конец процедуры.

Новый элемент "просеивается" сквозь пирамиду.

После построения пирамиды сортировка осуществляется следующим образом:

1. Берем верхний элемент пирамиды $a[0]..a[n]$ (первый в массиве) и меняем с последним местами. Теперь "забываем" об этом элементе и далее рассматриваем массив $a[0]..a[n-1]$. Для превращения его в пирамиду достаточно просеять лишь новый первый элемент.

2. Повторяем шаг 1, пока обрабатываемая часть массива не уменьшится до одного элемента.

Очевидно, в конец массива каждый раз попадает максимальный элемент из текущей пирамиды, поэтому в правой части постепенно возникает упорядоченная последовательность.

Метод не является устойчивым: по ходу работы массив так перемешивается, что исходный порядок элементов может измениться случайным образом.

Поведение неестественно: частичная упорядоченность массива никак не учитывается.

2.2 Двоичный поиск

Алгоритм двоичного поиска в упорядоченном массиве сводится к следующему. Берём средний элемент отсортированного массива и сравниваем с ключом X . Возможны три варианта:

1. выбранный элемент равен X . Поиск завершён;
2. выбранный элемент меньше X . Продолжаем поиск в правой половине массива;
3. выбранный элемент больше X . Продолжаем поиск в левой половине массива.

Из-за необходимости нахождения всех элементов, соответствующих заданному ключу поиска, в курсовой работе использовалась вторая версия двоичного поиска, которая из необходимых элементов находит самый левый, в результате чего для поиска остальных требуется просматривать лишь оставшуюся правую часть массива.

3. Описание структуры программы

Функциональное назначение программы

Разработанная программа представляет собой приложение по просмотру базы данных и произведению в ней определенных операций, таких как, например, поиск людей по заданной дате поселения. Она позволяет пользователю просматривать отсортированную базу данных людей по дате поселения и названию улицы. Функционирование программы и код функций представлены в приложении А и Б соответственно.

Язык и среда программирования

Программа была составлена на языке C++ с использованием компилятора g++. При разработке программы была использована литература /3-5/.

3.1 Описание структур данных и функций

struct person – структура созданной базы данных.

struct list – структура с указателем на следующий элемент и указатель на person. Используется как элемент очереди.

struct queue – структура, используемая для формирования очереди. Указывает на головной и хвостовой элементы очереди.

int noteAmount() - функция подсчета количества записей в базе данных.

void init(queue *q) — процедура инициализации очереди. Параметр процедуры: указатель на очередь.

int isempty(queue *q) — функция проверки очереди на пустоту. Параметр функции: указатель на очередь.

void insert(queue *q, person *data) — процедура вставки элемента очереди. Параметры функции: указатели на очередь и на элемент, который нужно занести в очередь.

void personOutForQueue(queue *q) — процедура вывода на экран одной записи из очереди. Параметр процедуры: указатель на очередь.

void printQueue(queue *q) — процедура вывода всей очереди на экран. Параметр процедуры: указатель на очередь.

void deleteQueue(queue *q) — процедура удаления очереди. Параметр процедуры: указатель на очередь.

void read(person* mass[]) - процедура чтения базы из файла с формированием индексного массива. Параметр процедуры: указатель на двумерный массив, в который считывается база.

void swap(int *a, int *b) — процедура меняющая местами два целых числа. Параметры процедуры: два указателя на числа, которые нужно поменять местами.

int dateCmp(char d1[], char d2[]) - функция сравнения двух дат. Параметры функции: две сравниваемые строки.

int streetCmp(char s1[], char s2[]) - функция сравнения названий двух улиц. Параметры функции: две сравниваемые строки.

void downHeap(int root, int n) — процедура формирования пирамиды. Параметры функции: индекс корневого элемента и размер дерева.

void heapSort() - процедура пирамидальной сортировки.

void personOut(int i) — процедура вывода на экран одной записи базы данных. Параметр функции: индекс элемента, который нужно вывести.

int yearCmp(char s1[], int K) — функция сравнения даты с ключом поиска. Параметры функции: строка с датой и ключ поиска.

void binarySearch(int K) — процедура бинарного поиска по ключу. Параметр процедуры: ключ поиска.

Исходный код представлен в приложении А.

ЗАКЛЮЧЕНИЕ

В ходе выполнения курсовой работы были закреплены знания по работе с массивами и двоичным поиском, освоен алгоритм сортировки методом Вильямса-Флойда, приобретены навыки построения очереди.

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ:

- 1) Структуры данных и алгоритмы. ООО “И.Д. Вильямс”, 2007.
- 2) Ананий Левитин. Алгоритмы: введение в разработку и анализ. ООО “И.Д. Вильямс”, 2006.
- 3) Очередь (структура данных). Электронный справочник. URL: <http://kvodo.ru/queue.html>
- 4) Стенли Б. Липпман, Жози Лажойе, Барбара Э. Му. Язык программирования С++. Базовый курс. Издательство Диалектика, 2014.
- 5) Седжвик Роберт. Фундаментальные алгоритмы на С++. Издательство «ДиаСофт», 2001.

ПРИЛОЖЕНИЕ А

Исходный код программы

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

const char* filename = "baza.txt";

int noteAmount(){
    int n = 0;
    char buf[35];
    FILE* file;
    file = fopen("baza.txt", "r");
    char* ptr = fgets(buf, 50, file);

    while(!feof(file)){
        ptr = fgets(buf, 50, file);
        n+=1;
    }

    fclose(file);
    return n/5;
};int N = noteAmount();

int* index = (int*)malloc(N*sizeof(int)); // Индексный массив

struct person{
    char fio[32];
    char street[18];
    short int building;
    short int flat;
    char date[10];
};

person** mass = (person**)malloc(N*sizeof(person*));

//-----
struct list{
    person* data;
    list* ptr;
};
struct queue{
    list *head, *tail;
};

void init(queue* q){ // Инициализация очереди
    q->head = NULL;
    q->tail = NULL;
}

int isempty(queue* q){ // Проверка очереди на пустоту. 1-пустая, 0 - непустая
    if(q->head == NULL) return 1;
    else return 0;
}

void insert(queue* q, person* data){ // Вставка элемента в очередь
    if(q->head == NULL && q->tail == NULL){
```

```

    q->head = (list*)malloc(sizeof(list));
    q->tail = (list*)malloc(sizeof(list));
    q->head->data = data;
    q->head->ptr = q->tail;
    q->tail = q->head;
    q->tail->ptr = NULL;
} else{
    list* tmp = (list*)malloc(sizeof(list));
    tmp->data = data;
    tmp->ptr = NULL;
    q->tail->ptr = tmp;
    q->tail = tmp;
}
}

void personOutForQueue(person* p){ // Вывод для очереди
    cout << p->fio << " " << p->street << " " << p->building << " " << p->flat << " " <<
p->date;
}

void printQueue(queue* q){ // Вывод очереди на экран
    list* h = q->head;

    if(isempty(q) == 1){
        cout << "Queue is empty!" << "\n";
        return;
    }
    while(h){
        personOutForQueue(h->data);
        h = h->ptr;
    }
}

void deleteQueue(queue* q){ // Удаление очереди
    if(isempty(q) == 1){
        free(q);
        return;
    }
    bool flag = true;

    while(flag){
        list* tmp;
        tmp = q->head;
        q->head = q->head->ptr;
        free(tmp);
        if(isempty(q) == 1){
            flag = false;
        }
    }
    free(q);
}
//-----

void read(person* mass[]){ // Чтение из файла с формированием индексного массива
    FILE* file = fopen(filename, "r");
    if(!file){
        cout << "Не удалось открыть файл! Выход из программы!" << "\n";
        exit(1);
    }
    char buf[33];
    int ind = 0;

```

```

char* ptr;

while(!feof(file)){
    mass[ind] = (person*)malloc(sizeof(person));
    fgets(mass[ind]->fio, 32, file);
    mass[ind]->fio[29] = '\0';
    fgets(mass[ind]->street, 18, file);
    mass[ind]->street[15] = '\0';
    fscanf(file, "%hu", &mass[ind]->building);
    fscanf(file, "%hu", &mass[ind]->flat);
    getc(file);
    fgets(mass[ind]->date, 12, file);
    mass[ind]->date[10] = '\0';
    index[ind] = ind;

    ind++;
}
fclose(file);
}

void swap(int* a, int* b){
    int* c = a;
    a = b;
    b = c;
}

int dateCmp(char d1[], char d2[]){ // Сравнение даты. 1-первая больше, 2-вторая больше, 0-равны
    char year1[3], year2[3], month1[3], month2[3], day1[3], day2[3];
    int y1, y2, m1, m2, day11, day22;

    year1[0] = d1[6]; year1[1] = d1[7]; year1[2] = '\0';
    year2[0] = d2[6]; year2[1] = d2[7]; year2[2] = '\0';
    month1[0] = d1[3]; month1[1] = d1[4]; month1[2] = '\0';
    month2[0] = d2[3]; month2[1] = d2[4]; month2[2] = '\0';
    day1[0] = d1[0]; day1[1] = d1[1]; day1[2] = '\0';
    day2[0] = d2[0]; day2[1] = d2[1]; day2[2] = '\0';
    y1 = atoi(year1); y2 = atoi(year2);
    m1 = atoi(month1); m2 = atoi(month2);
    day11 = atoi(day1); day22 = atoi(day2);

    if(y1 > y2) return 1;
    else if(y1 < y2) return 2;
    else if(m1 > m2) return 1;
    else if(m1 < m2) return 2;
    else if(day11 > day22) return 1;
    else if(day11 < day22) return 2;
    else return 0;
}

int streetCmp(char s1[], char s2[]){ // Сравнение названия улиц. 1-первое больше, 2-второе больше, 0-одинаковые

    for(int i = 0; i < 16; i++){
        if(s1[i] > s2[i]) return 1;
        else if(s1[i] < s2[i]) return 2;
    }
    return 0;
}

void downHeap(int root, int n){ // Формирование пирамиды

```

```

int max_child;
bool flag = true;

while(root*2 <= n && flag){
    if(root*2 == n){ // Нахождение максимального сына
        max_child = root*2;
    } else if(dateCmp(mass[index[root*2]]->date, mass[index[root*2+1]]->date) == 1){
        max_child = root*2;
    } else if(dateCmp(mass[index[root*2]]->date, mass[index[root*2+1]]->date) == 0){
        if(streetCmp(mass[index[root*2]]->street, mass[index[root*2+1]]->date) == 1){
            max_child = root*2;
        } else max_child = root*2+1;
    } else max_child = root*2+1;

    if(dateCmp(mass[index[root]]->date, mass[index[max_child]]->date) == 2){ //
        Проверка условия пирамиды (корень больше листьев)
        swap(index[max_child], index[root]);
        root = max_child;
    } else if(dateCmp(mass[index[root]]->date, mass[index[max_child]]->date) == 0){
        if(streetCmp(mass[index[root]]->street, mass[index[max_child]]->street) == 2){
            swap(index[max_child], index[root]);
            root = max_child;
        } else flag = false;
    } else flag = false;
}
}

void heapSort(){ // Пирамидальная сортировка по возрастанию
    for(int i = (N-1)/2; i >= 0; i--){
        downHeap(i, N-1);
    }
    for(int i = N-1; i >= 1; i--){
        swap(index[0], index[i]);
        downHeap(0, i-1);
    }
}

void personOut(int i){
    cout << mass[i]->fio << " " << mass[i]->street << " " << mass[i]->building << " " <<
    mass[i]->flat << " " << mass[i]->date;
}

int yearCmp(char s1[], int K){ // Сравнение года с ключом. 1-строка больше ключа, 2-
    ключ больше строки, 0-
    char year[3];
    int y;
    year[0] = s1[6]; year[1] = s1[7]; year[2] = '\0';
    y = atoi(year);

    if(y > K) return 1;
    else if(y < K) return 2;
    else return 0;
}

void binarySearch(int K){ // Бинарный поиск по отсортированному массиву
    queue* q = (queue*)malloc(sizeof(queue));
    init(q);
    int left = 0, right = N-1, m, j;
    while(left < right){
        m = (left+right)/2;
        if(yearCmp(mass[index[m]]->date, K) == 2) left = m+1;
    }
}

```



```

    else right = m;
}

if(right != 0){
    if(yearCmp(mass[index[right-1]]->date, K) == 0){
        for(j = right-1; j >= 0; j--){
            if(yearCmp(mass[index[j]]->date, K) != 0){
                j++;
                break;
            }
        }
    } else j = right;
} else j = right;

for(int i = j; i < N; i++){
    if(yearCmp(mass[index[i]]->date, K) == 0){
        insert(q, mass[index[i]]);
    } else break;
}

printQueue(q);
deleteQueue(q);
}

void clearScreen(){// Процедура очистки экрана
    cout << "\033[2J\033[1;1H";
}

int main(){
    int ansMain, temp, i, K;
    char ans;
    bool flag = true, flag1 = true;

    read(mass);
    heapSort();

    while(flag)
    {
        clearScreen();
        cout << "Данные считаны из файла \"" << filename << "\"\n";
        cout << "Количество элементов в базе данных: " << N << "\n";
        cout << "-----\n";
        cout << " 1. Вывести базу данных на экран.\n";
        cout << " 2. Сортировать базу данных и вывести ее на экран.\n";
        cout << " 3. Найти в базе данных записи по ключу поиска.\n\n";
        cout << " 0. Выйти из программы.\n\n  >";
        cin >> ansMain;

        switch (ansMain)
        {
        case 1:
            clearScreen();
            for(i = 0; i < N; i++){
                if((i+1) < 10) cout << " ";
                cout << " " << i+1 << ". ";
                personOut(i);

                if((i+1) % 20 == 0)
                {
                    m1:

```

```

        cout << "Продолжить? (у или n)... ";
        cin >> ans;

        if(ans == 'y') clearScreen();
        else if(ans == 'n') break;
        else goto m1;
    }
}

if(i == N){
    cout << "\n-----КОНЕЦ БАЗЫ ДАННЫХ-----\n\n";
    cout << "Для продолжения введите любую цифру... ";
    cin >> temp;
}
break;

case 2:
    clearScreen();
    for(i = 0; i < N; i++){
        if((i+1) < 10) cout << " ";
        cout << " " << i+1 << ". ";
        personOut(index[i]);

        if((i+1) % 20 == 0)
        {
            m2:
            cout << "Продолжить? (у или n)... ";
            cin >> ans;

            if(ans == 'y') clearScreen();
            else if(ans == 'n') break;
            else goto m2;
        }
    }

    if(i == N){
        cout << "\n-----КОНЕЦ БАЗЫ ДАННЫХ-----\n\n";
        cout << "Для продолжения введите любую цифру... ";
        cin >> temp;
    }
    break;

case 3:
    clearScreen();

    cout << "Введите год поселения (две последние цифры года): ";
    cin >> K;
    binarySearch(K);

    cout << "\n-----КОНЕЦ ОЧЕРЕДИ-----\n\n";

    cout << "Для продолжения введите любую цифру... ";
    cin >> temp;
    break;

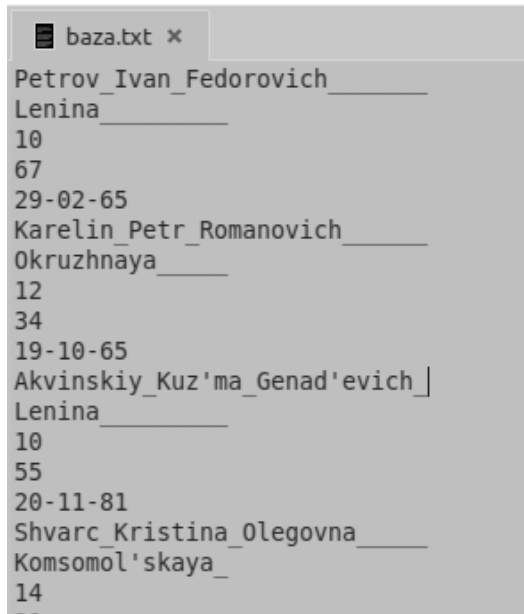
case 0:
    flag = false;
    break;

default:
    break; }clearScreen();return 0;}

```

ПРИЛОЖЕНИЕ Б

Функционирование программы.



```

baza.txt x
Petrov_Ivan_Fedorovich_____
Lenina_____
10
67
29-02-65
Karelin_Petr_Romanovich_____
Okruzhnaya_____
12
34
19-10-65
Akviskiy_Kuz'ma_Genad'evich_|
Lenina_____
10
55
20-11-81
Shvarc_Kristina_Olegovna_____
Komsomol'skaya_
14
21

```

Рисунок А.1. База данных
«Населенный пункт» в исходном
текстовом файле.



```

alway@alway-Lenovo-B590: ~/Документы/СТАЛ/КУРСОВАЯ
Файл  Правка  Вид  Поиск  Терминал  Справка
1. Petrov_Ivan_Fedorovich_____ Lenina_____ 10 67 29-02-65
2. Karelin_Petr_Romanovich_____ Okruzhnaya_____ 12 34 19-10-65
3. Akviskiy_Kuz'ma_Genad'evich_| Lenina_____ 10 55 20-11-81
4. Shvarc_Kristina_Olegovna_____ Komsomol'skaya_ 14 21 01-03-91
5. Prut'ko_Viktor_Artemovich_____ Sadovaya_____ 11 43 06-12-64
6. Lobanov_Semen_Fedorovich_____ Lenina_____ 43 12 18-11-87
7. Radchenko_Natal'ya_Romanovna_ Moskovskaya_____ 16 21 30-07-81
8. Tatarchenko_Yutiy_L'vovich_____ Bol'shaya_____ 75 44 15-03-95
9. Ryabinina_Mariya_Dmitrievna_ Okruzhnaya_____ 89 41 30-07-81
10. Karasik_Andrey_Denisovich_____ Sadovaya_____ 36 26 31-12-79
11. Odincova_Mariya_Yakimovna_____ Lenina_____ 46 35 22-01-97
12. Kirienko_Artem_Vladimirovic_ Sadovaya_____ 65 11 10-08-68
13. Pivovarov_Kiril_Valer'evich_ Abricosovaya_____ 34 11 14-01-87
14. Sidorov_Valentin_Genad'evich_ Okruzhnaya_____ 75 44 19-11-65
15. Petrenko_Mariya_Viktorovna_ Polyarnaya_____ 37 92 12-08-64
16. Chekulaev_Valentin_Yur'evich_ Okruzhnaya_____ 38 54 09-03-99
17. Lastochkin_Mark_Yur'evich_____ Lenina_____ 25 95 26-02-81
18. Balabanova_Alisa_Genad'evna_ Polyarnaya_____ 35 11 12-11-91
19. Ryabova_Margarita_Fedorovna_ Abricosovaya_____ 35 12 12-01-98
20. Grigor'ev_Sergey_Sergeevich_ Abricosovaya_____ 22 32 26-07-82
Продолжить? (y или n)...

```

Рисунок А.2. Вывод базы данных на экран.



Рисунок А.3. Вывод отсортированной по дате поселения и названию улицы базы данных на экран.

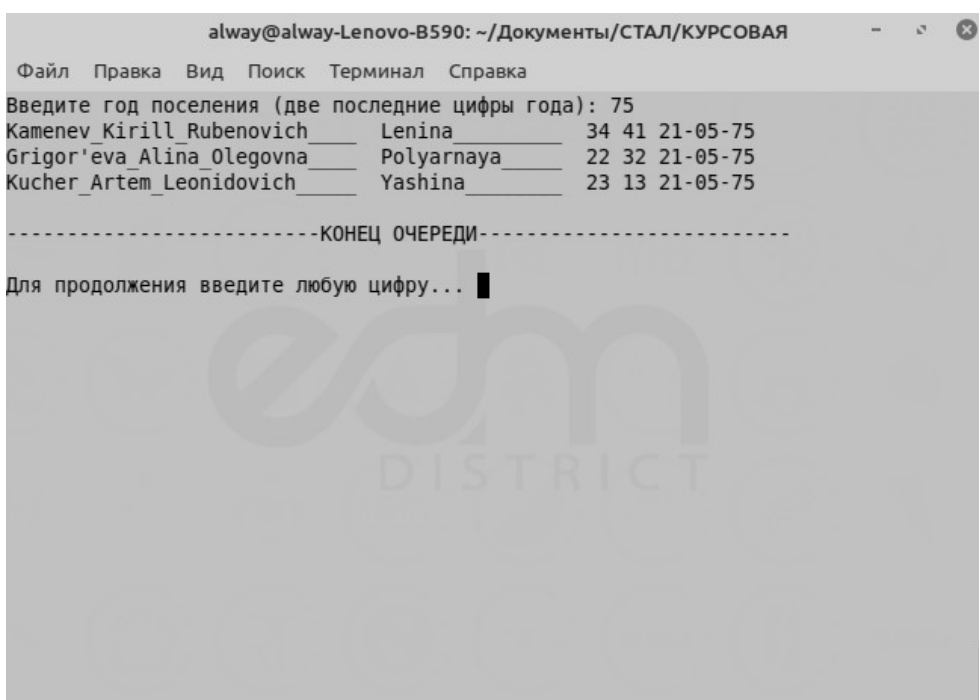


Рисунок А.4. Поиск человека по году поселения и вывод очереди на экран.