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-180006287-ТД

Выполнил студент Пшеничный Д. О.

Факультет, группаФКФН, ПО(аб)-81

Руководитель работы Бахрушина Г. И.

Виза:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(доработать, к защите и т.д.)

Хабаровск – 2020г.

Оглавление

[ВВЕДЕНИЕ 3](#_Toc28125783)

[1.Постановка задачи 4](#_Toc28125784)

[2.Описание алгоритмов сортировки, поиска(в соответствии с вариантом) 5](#_Toc28125785)

[2.1. Метод цифровой сортировки. 5](#_Toc28125786)

[2.2. Двоичный поиск. 7](#_Toc28125787)

[3. Описание структуры программы. 8](#_Toc28125788)

[ЗАКЛЮЧЕНИЕ 10](#_Toc28125790)

[СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ: 11](#_Toc28125791)

[ПРИЛОЖЕНИЕ А 12](#_Toc28125792)

[ПРИЛОЖЕНИЕ Б 25](#_Toc28125802)

ВВЕДЕНИЕ

Структура — это набор данных, где данные могут быть разного типа.

Структуру стоит использовать, если нам необходимо объединить несколько 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 списка, который сортируется цифровым методом, в результате чего получается индексный массив.

Список - это абстрактный тип данных, представляющий собой упорядоченный набор значений, в котором некоторое значение может встречаться более одного раза. Экземпляр списка является компьютерной реализацией математического понятия конечной последовательности. Экземпляры значений находящихся в списке, называются элементами списка.

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 "Обманутые вкладчики" загрузить динамически в оперативную память компьютера в виде списка вывести на экран по 20 записей (строк) на странице с возможностью отказа от просмотра. Текстовый файл создать самостоятельно.

Упорядочить данные по сумме ФИО и сумме вклада, используя **метод Цифровой сортировки** . Упорядоченные данные вывести на экран.

Реализовать возможность быстрого поиска по ключу (**ключ – первые три буквы фамилии**) 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 Метод цифровой сортировки

Цифровая сортировка - один из алгоритмов сортировки, использующих внутреннюю структуру сортируемых объектов. Здесь числа сортируются по разрядам.

Перед сортировкой необходимо определить 2 величины:

1. width - максимальное количество разрядов в сортируемых величинах.
2. range - количество возможных значений одного разряда ключа(сортируемого элемента), то есть мощность используемого алфавита.

Алгоритм включает в себя три этапа:

1. распределение элементов по «корзинам» в соответствии с текущим сортируемым разрядом;
2. последовательное соединение всех корзин;
3. повторение алгоритма, пока не будет достигнут самый старший разряд.

В разработанной программе сортировка происходит по текстовому и числовому параметру. Так как символы очень легко представляются в виде чисел, их сортировка почти не отличается от сортировки чисел. В виду большой мощности алфавита, от изначального представления корзин было решено отказаться. Вместо этого создается два одинаковых массива, содержащих указатели на элементы несортированного списка. Эти два массива попеременно сортируются «пересыпанием» из одного в другой. В итоге первый массив получается отсортированным, а второй удаляется из памяти.

Для сортировки по второму ключу элементов, имеющих идентичный первый ключ сортировки, производится поиск диапазонов таких элементов. Для каждого диапазона создается два подмассива, содержащих только эти элементы. Подмассивы сортируются аналогично «пересыпанием» одного в другой. Значения указателей из отсортированного подмассива переписываются в диапазон, и после оба подмассива удаляются из памяти.

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 производится поиск одного вхождения введенного ключа, и после находится левая и правая граница диапазона результатов.

3 Описание программы

3.1 Функциональное назначение программы

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

3.2 Программное обеспечение

Программа была составлена на языке C++ с использованием встроенного компилятора g++. Исполняемый файл программы может быть запущен на любом компьютере под управлением операционной системы Linux. При разработке программы была использована литература /3-5/.

3.3 Описание функций и структур данных

**struct Note** - структура записи в базе данных

**struct ListElement** - структура элемента списка

**struct Queue** - структура элемента очереди

**void Fixer()** - исправление ФИО из исходного файла согласно реализации

**void DigitalSorting()** - функция цифровой сортировки

**void ShowUnsorted()** - вывод несортированной базы

**void ShowSorted()** - вывод сортированной базы

**void ISummToChSumm()** - перевод суммы вклада из типа данных **unsigned short int** в тип данных **char[]**

**void BinarySearch()** - функция бинарного поиска

**void ConsoleClean()** - очистка консоли

**int FindBorder()** - поиск границы диапазона

**int MaxLength()** - определение максимальной длины суммы вклада

Исходный код представлен в приложении Б.

ЗАКЛЮЧЕНИЕ

В ходе выполнения курсовой работы были закреплены знания по работе со списками и двоичным поиском, освоен алгоритм цифровой сортировки, приобретены навыки построения очереди.

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ:

1. MAXimal сборник алгоритмов С++. <http://e-maxx.ru/algo/>
2. Алгоритм цифровой сортировки. Университет ИТМО. URL:

[http://neerc.ifmo.ru/wiki/index.php?title=%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC\_%D1%86%D0%B8%D1%84%D1%80%D0%BE%D0%B2%D0%BE%D0%B9\_%D1%81%D0%BE%D1%80%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%BA%D0%B8](http://neerc.ifmo.ru/wiki/index.php?title=Алгоритм_цифровой_сортировки)

1. Очередь (структура данных). Электронный справочник. URL: <http://kvodo.ru/queue.html>
2. Бьерн Страутсруп. Язык программирования С++. 1985.
3. Fundamentals of C++ Programming. Richard L. Halterman. 2019.

ПРИЛОЖЕНИЕ А

Код функций.

#include <iostream>

#include <string>

#include <string.h>

#include <stdlib.h>

#include <stdio.h>

#include <fstream>

using namespace std;

struct Note

{

public:

char contributorData[30];

unsigned short int contributionSumm;

char contributionDate[10];

char lawyerData[22];

Note(char contrData[30], unsigned short int contrSumm, char \*contrDate, char \*lawData)

{

strcpy(contributorData, contrData);

contributionSumm = contrSumm;

strcpy(contributionDate, contrDate);

strcpy(lawyerData, lawData);

}

Note()

{

}

private:

};

struct ListElement

{

public:

Note note;

ListElement \*nextElement = NULL;

ListElement()

{

}

ListElement(Note valueField)

{

note = valueField;

}

void ShowList()

{

cout << note.contributorData << " "

<< note.contributionSumm << " "

<< note.contributionDate << " "

<< note.lawyerData << endl;

if (nextElement)

{

nextElement->ShowList();

}

}

int ListLength(int length = 1) //Returns lenth of list in a human vision

{

if (nextElement)

{

nextElement->ListLength(length + 1);

}

else

{

return length;

}

}

ListElement \*ElementGetter(int ind)

{

if (ind > 0)

nextElement->ElementGetter(ind - 1);

else

return this;

}

private:

};

struct Queue

{

public:

Note \*note = NULL;

Queue \*nextElement = NULL;

Queue(Note \*notePtr)

{

note = notePtr;

}

Queue()

{

}

void ShowList(int i = 0)

{

cout << i + 1 << ") "

<< note->contributorData << " "

<< note->contributionSumm << " "

<< note->contributionDate << " "

<< note->lawyerData << endl;

if (nextElement)

{

nextElement->ShowList(i + 1);

}

}

void Delete()

{

if (nextElement)

{

note = nextElement->note;

nextElement = nextElement->nextElement;

Delete();

}

}

private:

};

int MaxLength(ListElement \*\*, int);

int FindBorder(ListElement \*\*, char \*, bool, int, int, int);

void Fixer(char \*, int);

void DigitalSorting(int, ListElement \*\*, ListElement \*\*, ListElement \*);

void DigitalSorting(ListElement \*\*, ListElement \*);

void ShowUnsorted(ListElement \*);

void ShowSorted(ListElement \*\*, int);

void ISummToChSumm(char \*, ListElement \*\*, int, int);

void BinarySearch(ListElement \*\*, int);

void ConsoleClean();

int main()

{

ListElement \*head = NULL;

ListElement \*tail = NULL;

string inputPath = "";

cout << "Enter input file path: ";

cin >> inputPath;

ifstream file;

file.open(inputPath);

if (!file)

{

cout << "Invalid file!\n";

return 0;

}

while (!file.eof()) //Чтение из файла

{

char contributorData[30] = "";

char fullContributorData[100] = "";

unsigned short int contributionSumm = NULL;

char contributionDate[10] = "";

char lawyerData[22] = "";

file >> fullContributorData;

file >> contributionSumm;

file >> contributionDate;

file >> lawyerData;

if (strlen(fullContributorData) >= 30)

{

strncpy(contributorData, fullContributorData, 29);

}

else

{

strcpy(contributorData, fullContributorData);

}

Fixer(contributorData, 30);

Fixer(lawyerData, 22);

Note newNote = \*new Note(contributorData, contributionSumm, contributionDate, lawyerData);

if (!head)

{

head = new ListElement(newNote);

}

else if (!tail)

{

tail = new ListElement(newNote);

head->nextElement = tail;

}

else

{

tail->nextElement = new ListElement(newNote);

tail = tail->nextElement;

}

}

ListElement \*tempElement = head;

while (tempElement->nextElement)

{

if (tempElement->nextElement == tail)

{

tempElement->nextElement = NULL;

tail = tempElement;

break;

}

tempElement = tempElement->nextElement;

}

//head->ShowList();

ListElement \*\*indMass = new ListElement \*[head->ListLength()];

ListElement \*\*tempSortMass = new ListElement \*[head->ListLength()];

for (int i = 0; i < head->ListLength(); i++)

{

indMass[i] = head->ElementGetter(i);

tempSortMass[i] = indMass[i];

}

DigitalSorting(2, indMass, tempSortMass, head);

DigitalSorting(indMass, head);

delete (tempSortMass);

char marker[4] = {' ', ' ', ' ', ' '}; //вверх = 65, вниз = 66, лево = 68, право = 67, y = 121

int choise = 0;

while (true)

{

for (int i = 0; i < 4; i++)

{

if (i == choise)

marker[i] = '>';

else

marker[i] = ' ';

}

ConsoleClean();

cout << "Main menu:\n";

cout << marker[0] << "Show unsorted base\n";

cout << marker[1] << "Show sorted base\n";

cout << marker[2] << "Search in base\n";

cout << marker[3] << "Exit\n";

cout << "\n\n\n\n\nUse up/down keys to navigate. To accept your choise press 'y'. After you press the key you want, press \"Enter\"\n";

while (getchar() != '\n')

;

cin.clear();

int keyCode = getchar();

cout << keyCode;

if (keyCode == 27)

{

keyCode = getchar();

keyCode = getchar();

}

switch (keyCode)

{

case 65:

if (choise > 0)

choise--;

break;

case 66:

if (choise < 3)

choise++;

break;

case 121:

switch (choise)

{

case 0:

ShowUnsorted(head);

break;

case 1:

ShowSorted(indMass, head->ListLength());

break;

case 2:

BinarySearch(indMass, head->ListLength() - 1);

break;

case 3:

return 0;

break;

default:

break;

}

break;

default:

break;

}

}

}

void DigitalSorting(int digit, ListElement \*\*mass, ListElement \*\*tempMass, ListElement \*head) //Первый проход с сортирровкой по Фамилии

{

int ind = 0;

if (digit % 2 == 0)

{

for (int i = 65; i < 123; i++)

{

if (i == 91 || i == 92 || i == 93 || i == 94)

{

continue;

}

for (int j = 0; j < head->ListLength(); j++)

{

if (tempMass[j]->note.contributorData[digit] - '0' + 48 == i)

{

mass[ind] = tempMass[j];

ind++;

}

}

}

}

else if (digit % 2 == 1)

{

for (int i = 65; i < 123; i++)

{

if (i == 91 || i == 92 || i == 93 || i == 94)

{

continue;

}

for (int j = 0; j < head->ListLength(); j++)

{

if (mass[j]->note.contributorData[digit] - '0' + 48 == i)

{

tempMass[ind] = mass[j];

ind++;

}

}

}

}

if (digit != 0)

{

DigitalSorting(digit - 1, mass, tempMass, head);

}

}

void DigitalSorting(ListElement \*\*mass, ListElement \*head) //Второй проход с сортировкой по сумме вклада

{

char lastName[15] = "";

int firstEnter = 0, lastEnter = 0, lastNameLength = 0;

for (int i = 0; i < head->ListLength(); i++)

{

if (!strcmp(lastName, ""))

{

for (int j = 0; j < strlen(mass[i]->note.contributorData); j++)

{

if (mass[i]->note.contributorData[j] != '\_')

{

lastName[j] = mass[i]->note.contributorData[j];

lastNameLength++;

}

else

{

break;

}

}

firstEnter = i;

continue;

}

char subStr[15] = "";

strncpy(subStr, mass[i]->note.contributorData, lastNameLength);

if (!strcmp(lastName, subStr))

{

lastEnter = i;

}

if (i == head->ListLength() - 1 && !strcmp(lastName, subStr))

{

lastEnter = i;

}

if (strcmp(lastName, subStr) || lastEnter == head->ListLength() - 1)

{

if (lastEnter - firstEnter >= 1)

{

ListElement \*\*subMass = new ListElement \*[lastEnter - firstEnter + 1];

ListElement \*\*tempSubMass = new ListElement \*[lastEnter - firstEnter + 1];

for (int j = 0; j <= lastEnter - firstEnter; j++)

{

subMass[j] = mass[firstEnter + j];

tempSubMass[j] = subMass[j];

}

int cycleAmount = MaxLength(subMass, lastEnter - firstEnter + 1);

for (int j = cycleAmount - 1; j >= 0; j--)

{

int index = 0;

for (int k = 0; k <= 9; k++)

{

if (j % 2 == 0)

{

for (int h = 0; h <= lastEnter - firstEnter; h++)

{

char chSumm[15];

ISummToChSumm(chSumm, tempSubMass, h, cycleAmount);

if (chSumm[j] - '0' == k)

{

subMass[index] = tempSubMass[h];

index++;

}

}

}

else

{

for (int h = 0; h <= lastEnter - firstEnter; h++)

{

char chSumm[15];

ISummToChSumm(chSumm, subMass, h, cycleAmount);

if (chSumm[j] - '0' == k)

{

tempSubMass[index] = subMass[h];

index++;

}

}

}

}

}

for (int j = 0; j <= lastEnter - firstEnter; j++)

{

mass[j + firstEnter] = subMass[j];

}

delete (subMass);

delete (tempSubMass);

}

firstEnter = 0;

lastEnter = 0;

lastNameLength = 0;

for (int j = 14; j >= 0; j--)

{

if (j != 0)

{

lastName[j] = NULL;

}

else

{

lastName[j] = '\0';

}

}

i--;

}

}

}

void Fixer(char \*input, int length)

{

for (int i = length - strlen(input); i > 1; i--)

{

input[length - i] = '\_';

}

input[strlen(input)] = '\0';

}

int MaxLength(ListElement \*\*mass, int massLength)

{

int maxLength = 0;

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

{

char chSumm[15];

sprintf(chSumm, "%d", mass[i]->note.contributionSumm);

if (strlen(chSumm) > maxLength)

{

maxLength = strlen(chSumm);

}

}

return maxLength;

}

void ISummToChSumm(char \*chSumm, ListElement \*\*mass, int ind, int maxLength)

{

sprintf(chSumm, "%d", mass[ind]->note.contributionSumm);

while (strlen(chSumm) < maxLength)

{

for (int i = 14; i >= 0; i--)

{

if (i != 0)

{

chSumm[i] = chSumm[i - 1];

}

else

{

chSumm[i] = '0';

}

}

}

}

void ConsoleClean()

{

printf("\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n");

}

void ShowUnsorted(ListElement \*head)

{

int firstEnter = 0;

int lastEnter, pagesAmount, currentPage = 1;

int listLength = head->ListLength();

pagesAmount = listLength / 20;

if (listLength % 20 > 0)

pagesAmount++;

if (listLength <= 20)

lastEnter = listLength;

else

lastEnter = 20;

char pages[20] = "";

while (true)

{

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

{

if (i + 1 == currentPage)

pages[i] = currentPage + '0';

else

pages[i] = '\*';

}

ConsoleClean();

cout << "Unsorted base:\n\n";

for (int i = firstEnter; i < lastEnter; i++)

{

cout << i + 1 << ")" << head->ElementGetter(i)->note.contributorData << " "

<< head->ElementGetter(i)->note.contributionSumm << " "

<< head->ElementGetter(i)->note.contributionDate << " "

<< head->ElementGetter(i)->note.lawyerData << endl;

}

cout << "\n\n\n<" << pages << ">\n";

cout << "Use left/right keys to navigate. To quit press 'q'. After you press the key you want, press \"Enter\"\n";

while (getchar() != '\n')

;

cin.clear();

int keyCode = getchar();

if (keyCode == 27)

{

keyCode = getchar();

keyCode = getchar();

}

switch (keyCode)

{

case 68: //left

if (currentPage > 1)

{

currentPage--;

firstEnter = 20 \* (currentPage - 1);

lastEnter = 20 \* currentPage;

if (lastEnter > listLength)

lastEnter = listLength;

}

break;

case 67:

if (currentPage < pagesAmount)

{

currentPage++;

firstEnter = 20 \* (currentPage - 1);

lastEnter = 20 \* currentPage;

if (lastEnter > listLength)

lastEnter = listLength;

}

break;

case 113:

return;

break;

default:

break;

}

}

}

void ShowSorted(ListElement \*\*mass, int length)

{

int firstEnter = 0;

int lastEnter, pagesAmount, currentPage = 1;

int listLength = length;

pagesAmount = listLength / 20;

if (listLength % 20 > 0)

pagesAmount++;

if (listLength <= 20)

lastEnter = listLength;

else

lastEnter = 20;

char pages[20] = "";

while (true)

{

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

{

if (i + 1 == currentPage)

pages[i] = currentPage + '0';

else

pages[i] = '\*';

}

ConsoleClean();

cout << "Sorted base:\n\n";

for (int i = firstEnter; i < lastEnter; i++)

{

cout << i + 1 << ")" << mass[i]->note.contributorData << " "

<< mass[i]->note.contributionSumm << " "

<< mass[i]->note.contributionDate << " "

<< mass[i]->note.lawyerData << endl;

}

cout << "\n\n\n<" << pages << ">\n";

cout << "Use left/right keys to navigate. To quit press 'q'. After you press the key you want, press \"Enter\"\n";

while (getchar() != '\n')

;

cin.clear();

int keyCode = getchar();

if (keyCode == 27)

{

keyCode = getchar();

keyCode = getchar();

}

switch (keyCode)

{

case 68: //left

if (currentPage > 1)

{

currentPage--;

firstEnter = 20 \* (currentPage - 1);

lastEnter = 20 \* currentPage;

if (lastEnter > listLength)

lastEnter = listLength;

}

break;

case 67:

if (currentPage < pagesAmount)

{

currentPage++;

firstEnter = 20 \* (currentPage - 1);

lastEnter = 20 \* currentPage;

if (lastEnter > listLength)

lastEnter = listLength;

}

break;

case 113:

return;

break;

default:

break;

}

}

}

void BinarySearch(ListElement \*\*mass, int lastInd)

{

ConsoleClean();

int leftBorder = 0, rightBorder = lastInd;

int searchPointer = -1;

char input[10] = "";

char key[4] = "";

char substr[4] = "";

bool goingLeft = true;

cout << "Enter 3 first letters from last name.(Register is important!): ";

cin >> input;

if (strlen(input) < 3)

{

cout << "You must enter at least 3 letters!\n";

cout << "Press any key to continue";

while (getchar() != '\n')

;

cin.clear();

searchPointer = getchar();

return;

}

strncpy(key, input, 3);

key[3] = '\0';

while (leftBorder <= rightBorder)

{

int midInd = (rightBorder + leftBorder) / 2;

strncpy(substr, mass[midInd]->note.contributorData, 3);

substr[3] = '\0';

if (strcmp(substr, key) > 0)

{

rightBorder = midInd - 1;

}

else if (strcmp(substr, key) < 0)

{

leftBorder = midInd + 1;

}

else if (strcmp(substr, key) == 0)

{

searchPointer = midInd;

break;

}

}

if (searchPointer > 0)

{

leftBorder = FindBorder(mass, key, goingLeft, searchPointer, 0, lastInd);

rightBorder = FindBorder(mass, key, !goingLeft, searchPointer, 0, lastInd);

Queue \*head = new Queue(&mass[leftBorder]->note);

Queue \*tail = head;

for (int i = leftBorder + 1; i <= rightBorder; i++)

{

tail->nextElement = new Queue(&mass[i]->note);

tail = tail->nextElement;

}

head->ShowList();

head->Delete();

free(head);

free(tail);

// cout << mass[searchPointer]->note.contributorData << endl;

}

else

{

cout << "Such an element doesn't exist!\n";

}

cout << "Press any key to continue";

while (getchar() != '\n')

;

cin.clear();

searchPointer = getchar();

}

int FindBorder(ListElement \*\*mass, char \*key, bool goingLeft, int pointer, int leftRamp, int rightRamp)

{

char subStr[4] = "";

if (goingLeft)

{

int leftBorder = pointer;

while (true)

{

if (pointer == leftRamp)

{

return leftRamp;

}

strncpy(subStr, mass[pointer]->note.contributorData, 3);

subStr[3] = '\0';

if (!strcmp(key, subStr))

{

leftBorder = pointer;

}

else

{

return leftBorder;

}

pointer--;

}

}

else

{

int rightBorder = pointer;

while (true)

{

if (pointer == rightRamp)

{

return rightRamp;

}

strncpy(subStr, mass[pointer]->note.contributorData, 3);

subStr[3] = '\0';

if (!strcmp(key, subStr))

{

rightBorder = pointer;

}

else

{

return rightBorder;

}

pointer++;

}

}

}

ПРИЛОЖЕНИЕ Б

Функционирование программы.

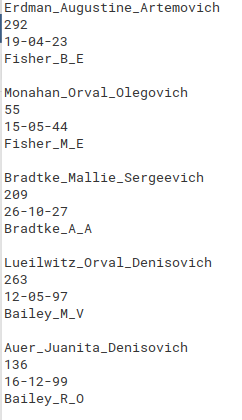


Рисунок Б.1. База данных «Обманутые вкладчики» в исходном текстовом файле.

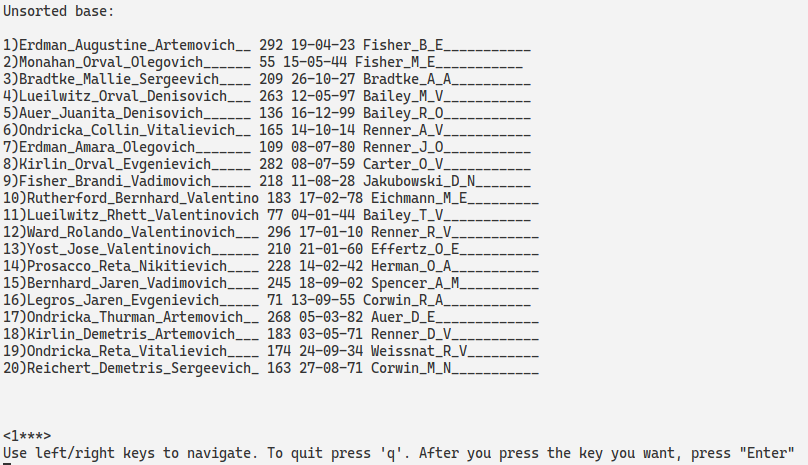


Рисунок Б.2. Вывод базы данных на экран.

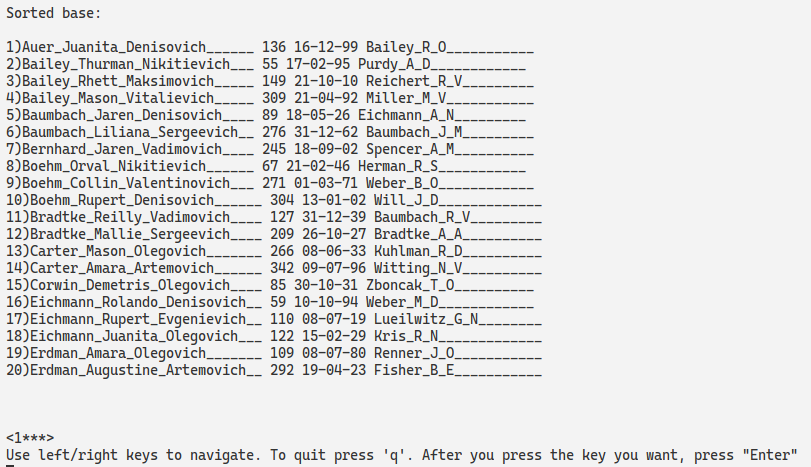


Рисунок Б.3. Вывод отсортированной базы данных на экран.

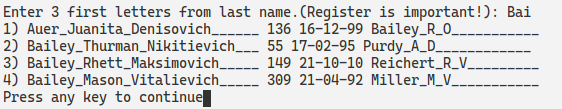


Рисунок Б.4. Поиск человека по трём буквам фамилии.

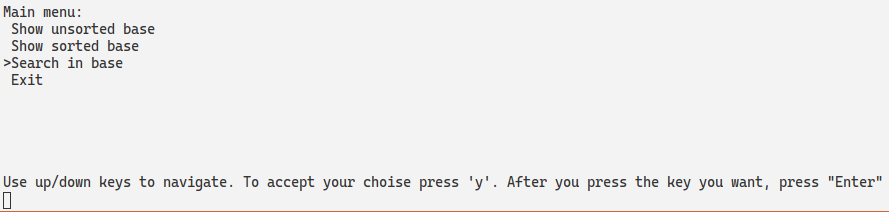


Рисунок Б.5. Главное меню программы.