МИНИСТЕРСТВО науки и высшего ОБРАЗОВАНИЯ РОссИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ

УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ

(национальный исследовательский университет)»

Институт №3 «Системы управления, информатика и электроэнергетика»

Кафедра № 304 «Вычислительные машины, системы и сети»

Пояснительная записка к курсовой работе

по дисциплине: «Программирование»

на тему: «Программа телепередач на неделю»

Выполнил студент группы M3О-119БК-22

Закусилов Л.З.

Проверил преподаватель

Ивашенцев И.В.

Москва 2023 г.

Содержание

[Задание 3](#_Toc152859769)

[Входные данные 4](#_Toc152859770)

[Выходные данные 5](#_Toc152859771)

[Описание интерфейса программы 6](#_Toc152859772)

[Структурные схемы алгоритмов 7](#_Toc152859773)

[Функция sort() 7](#_Toc152859774)

[Функция print() 8](#_Toc152859775)

[Функция find() 9](#_Toc152859776)

[Функция add() 10](#_Toc152859777)

[Код программы 11](#_Toc152859778)

[Файл list.h 11](#_Toc152859779)

[Файл list.cpp 11](#_Toc152859780)

[Файл dataBase.h 15](#_Toc152859781)

[Файл dataBase.cpp 16](#_Toc152859782)

[Файл functions.h 24](#_Toc152859783)

[Файл functions.cpp 24](#_Toc152859784)

[Файл main.cpp 27](#_Toc152859785)

[Результаты проверки функционирования программы 29](#_Toc152859786)

[Добавление нового студента в базу данных с клавиатуры 29](#_Toc152859787)

[Добавление новых студентов в базу данных из файла 30](#_Toc152859788)

[Удаление студентов из базы данных 31](#_Toc152859789)

[Сортировка базы данных по определенному полю 32](#_Toc152859790)

[Поиск в базе данных студентов по заданному условию 33](#_Toc152859791)

[Вывод 34](#_Toc152859792)

# Задание

Разработать систему управления базой данных на языке С++.  
Тематики базы данных выбираются согласно варианту.  
Для хранения данных использовать текстовый файл.  
Разработать структуры данных для хранения данных и функции для их обработки:

* для ввода данных,
* выборки значения по заданному условию,
* вывода данных,
* удаления,
* записи новых данных,
* сортировки по определенному пользователем значению.

Обеспечить выдачу запросов по любым полям данных.  
Всю обработку данных вести с помощью динамических структур данных.  
Реализовать систему меню для взаимодействия с пользователем по всем пунктам задания.

# Входные данные

В качестве входных данных используются данные, введенные пользователем с клавиатуры или считанные из файла.

# Выходные данные

Выходными данными программы является таблица с данными о студентах и файл с записанной в него базой данных.

# Описание интерфейса программы

Главное меню программы имеет вид:



Рисунок 1. Главное меню программы.

При вводе help в строке пользователю выведется весь список команд и их описание

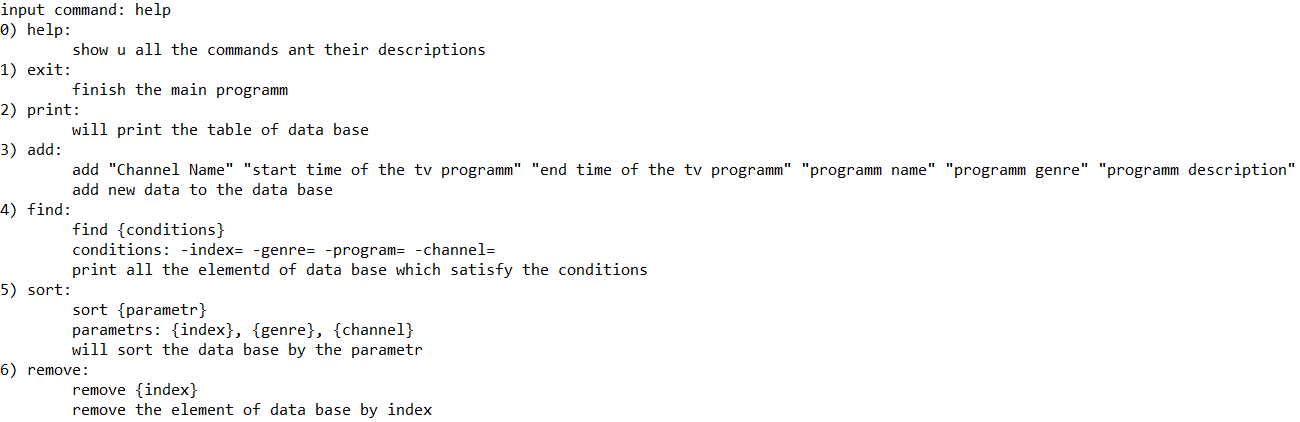


Рисунок 2. Ввод некорректных данных в главном меню.

Если пользователь ввёл неизвестную команду то программа сообщит об этом и предложить использовать команду “help”



# Структурные схемы алгоритмов

## Функция sort()

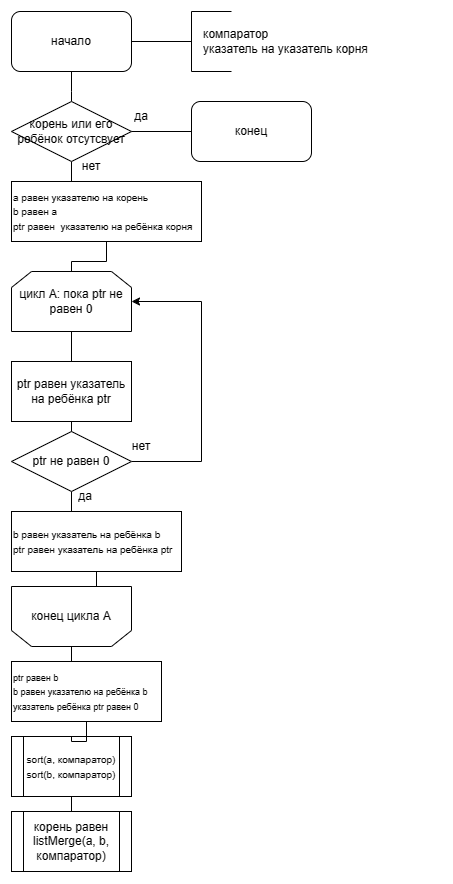


Рисунок 3. Структурная схема функции sort ()

## Функция print()

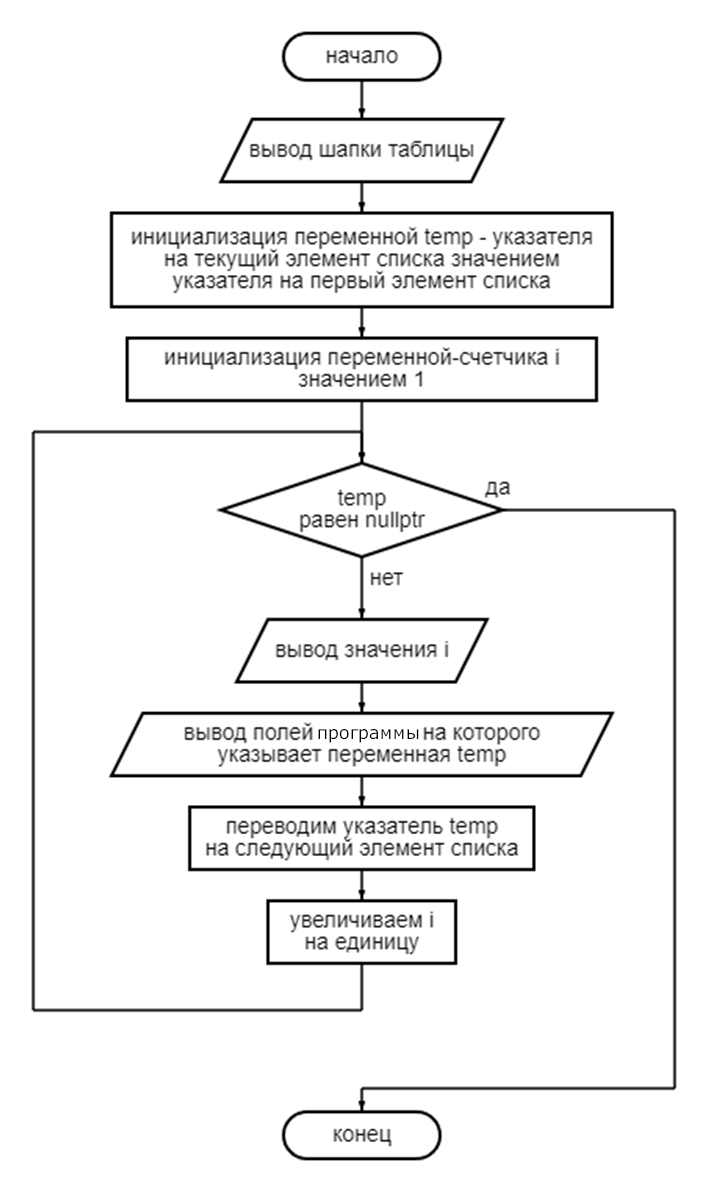


Рисунок 4. Структурная схема функции show()

## Функция find()

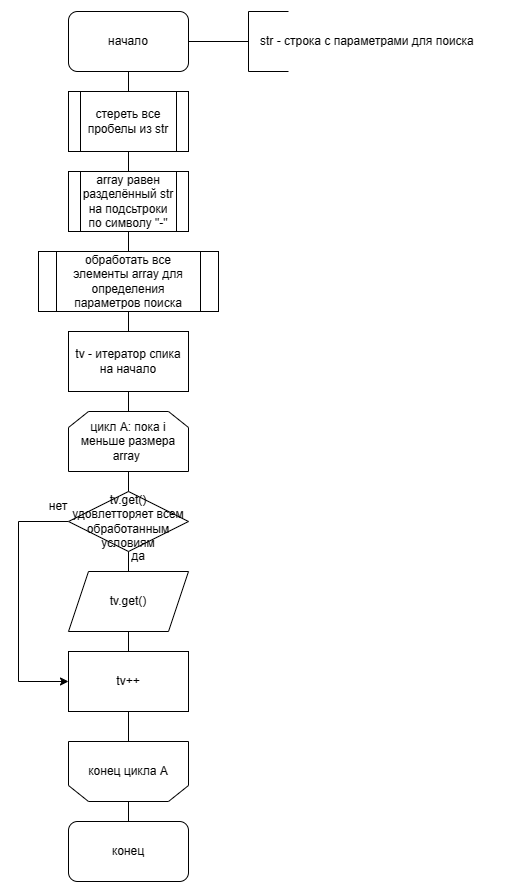


Рисунок 5. Структурная схема функции find()

## Функция add()

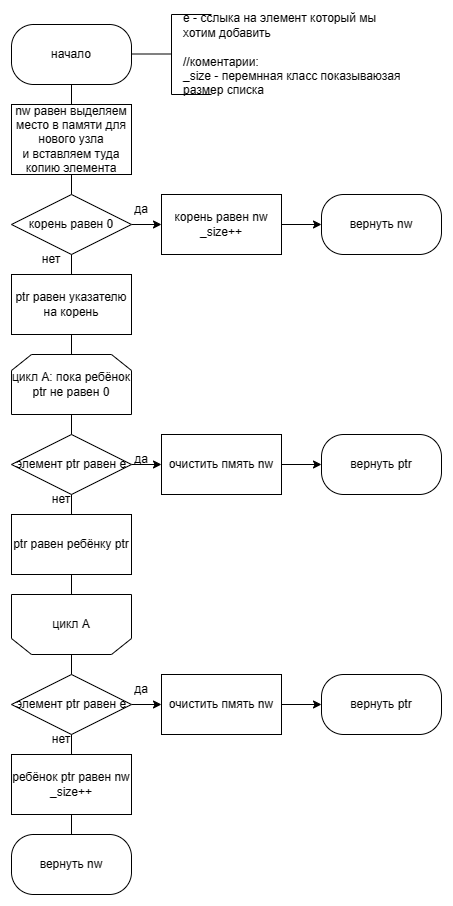


Рисунок 6. Структурная схема функции add()

# Код программы

## Файл list.h

#pragma once

#include "Prerequisites.h"

template<class T>

class list;

template<class T>

class listElement

{

public:

listElement(T\* element);

~listElement();

T\* get();

private:

listElement<T>\* next = nullptr;

T\* element;

friend class list<T>;

};

template<class T>

class list

{

public:

list();

~list();

void pop(uint index);

void sort(bool (\*comarator)(T&, T&));

void begin();

uint size();

T\* add(const T& element);

T\* get();

public:

list<T> operator++(int);

private:

listElement<T>\* listmerge(listElement<T>\* a, listElement<T>\* b, bool (\*comarator)(T&, T&));

void merge(listElement<T>\*\* head, bool (\*comarator)(T&, T&));

private:

listElement<T>\* head = nullptr;

uint \_size = 0;

listElement<T>\* pointer = head;

};

#include "List.imp"

## Файл list.cpp

#include "List.h"

#include <stdio.h>

template<class T>

listElement<T>::listElement(T\* element)

{

next = nullptr;

this->element = element;

}

template<class T>

listElement<T>::~listElement()

{

}

template<class T>

T\* listElement<T>::get()

{

return element;

}

template<class T>

list<T>::list()

{

\_size = 0;

head = nullptr;

pointer = nullptr;

}

template<class T>

list<T>::~list()

{

}

template<class T>

T\* list<T>::add(const T& element)

{

listElement<T>\* new\_element = new listElement<T>(new T(element));

if(\_size == 0)

{

\_size = 1;

head = new\_element;

pointer = head;

return new\_element->element;

}

listElement<T>\* ptr = head;

while(ptr->next != nullptr)

{

if(\*ptr->element == element)

{

delete new\_element;

return ptr->element;

}

ptr = ptr->next;

}

if(\*ptr->element == element)

{

delete new\_element;

return ptr->element;

}

\_size++;

ptr->next = new\_element;

return new\_element->element;

}

template<class T>

T\* list<T>::get()

{

if(pointer == nullptr)

{

return nullptr;

}

return pointer->element;

}

template<class T>

uint list<T>::size()

{

return \_size;

}

template<class T>

void list<T>::begin()

{

pointer = head;

}

template<class T>

void list<T>::pop(uint index)

{

if(\_size <= index)

{

return;

}

listElement<T>\* temp;

listElement<T>\* ptr = head;

if(index == 0)

{

ptr = head->next;

delete head;

head = ptr;

\_size--;

return;

}

index--;

uint i = 0;

while(i < index)

{

i++;

ptr = ptr->next;

}

temp = ptr->next->next;

delete ptr->next;

ptr->next = temp;

\_size--;

}

template<class T>

listElement<T>\* list<T>::listmerge(listElement<T>\* a, listElement<T>\* b, bool (\*comarator)(T&, T&))

{

if(a == nullptr)

{

return b;

}

if(b == nullptr)

{

return a;

}

listElement<T>\* ptr = nullptr;

if(comarator(\*(a->element), \*(b->element)))

{

ptr = a;

ptr->next = listmerge(a->next, b, comarator);

}

else

{

ptr = b;

ptr->next = listmerge(a, b->next, comarator);

}

return ptr;

}

template<class T>

void list<T>::merge(listElement<T>\*\* head, bool (\*comarator)(T&, T&))

{

if((\*head) == nullptr || (\*head)->next == nullptr)

{

return;

}

listElement<T>\* a = \*head;

listElement<T>\* ptr = a->next;

listElement<T>\* b = a;

while(ptr != nullptr)

{

ptr = ptr->next;

if(ptr != nullptr)

{

b = b->next;

ptr = ptr->next;

}

}

ptr = b;

b = b->next;

ptr->next = nullptr;

merge(&a, comarator);

merge(&b, comarator);

\*head = listmerge(a, b, comarator);

}

template<class T>

void list<T>::sort(bool (\*comarator)(T&, T&))

{

if(\_size < 2)

{

return;

}

merge(&head, comarator);

}

template<class T>

list<T> list<T>::operator++(int)

{

list<T> list = \*this;

if(this->\_size < 2)

{

this->pointer = this->head;

return list;

}

if(this->pointer->next != nullptr)

{

this->pointer = this->pointer->next;

}

return list;

}

## Файл dataBase.h

#pragma once

#include <string>

#include <stdio.h>

#include <vector>

#include "Prerequisites.h"

#include "List.h"

struct Genre

{

uint index;

std::string name;

bool operator==(const Genre& g1) const

{

if(this->name == g1.name)

{

return true;

}

return false;

}

};

struct TVprogram

{

uint index;

std::string name;

Genre\* indexOfGenre;

std::string description;

bool operator==(const TVprogram& p1) const

{

if(this->name == p1.name && this->indexOfGenre == p1.indexOfGenre)

{

return true;

}

return false;

}

};

struct Chanel

{

uint index;

std::string name;

bool operator==(const Chanel& c1) const

{

if(this->name == c1.name)

{

return true;

}

return false;

}

};

struct TVshowsProgram

{

uint index;

struct tm startTime;

struct tm endTime;

TVprogram\* indexOfTVprogram;

Chanel\* indexOfChanel;

bool operator==(const TVshowsProgram& t1) const

{

if(this->indexOfTVprogram == t1.indexOfTVprogram && this->indexOfChanel == t1.indexOfChanel)

{

return true;

}

return false;

}

};

class DataBase

{

public:

DataBase();

~DataBase();

char find(std::string& str);

char add(std::string& str);

void remove(uint index);

void sort(std::string command);

void print();

void save(const char\* filename);

void load(const char\* filename);

private:

char addAnalizer(std::vector<char\*> splited);

void printTittle();

void printElement(TVshowsProgram\* tv);

void calculateSpacesArray();

private:

list<Genre> genres;

list<TVprogram> TVprograms;

list<Chanel> chanels;

list<TVshowsProgram> TVshowsPrograms;

//print data

int namesArray [7] = {5, 7, 7, 5, 10, 8, 0};

int spacesArray[7] = {5, 7, 7, 5, 16, 16, 0};

const char\* tableNames[7] = {

"index",

"channel",

"program",

"genre",

"start time",

"end time",

"description",

};

uint distBetween = 5;

};

## Файл dataBase.cpp

#include "dataBase.h"

#include <time.h>

#include "functions.h"

DataBase::DataBase()

{

}

DataBase::~DataBase()

{

}

char DataBase::find(std::string& str)

{

str.erase(std::remove(str.begin(), str.end(), ' '), str.end());

int indexToFind = -1;

std::string genreToFind = "\0";

std::string channelToFind = "\0";

std::string programToFind = "\0";

std::vector<char\*> miniSplited;

std::vector<char\*> splited = split(str, '-');

for(uint i = 0; i < splited.size(); i++)

{

std::string text = splited[i];

miniSplited = split(text, '=');

if(miniSplited.size() != 2)

{

printf("%s is not correct form\n", splited[i]);

return -1;

}

text = miniSplited[0];

if(text == "index")

{

if(stringToInt(indexToFind, miniSplited[1]) == true)

{

return -1;

}

}

else if(text == "genre")

{

genreToFind = miniSplited[1];

}

else if(text == "channel")

{

channelToFind = miniSplited[1];

}

else if(text == "program")

{

programToFind = miniSplited[1];

}

else

{

printf("'%s' is unknow form\n", miniSplited[0]);

return -1;

}

}

uint counter = 0;

TVshowsProgram\* tv;

TVshowsPrograms.begin();

for(uint i = 0; i < TVshowsPrograms.size(); i++)

{

tv = TVshowsPrograms.get();

if(

(indexToFind == tv->index || indexToFind == -1) &&

(genreToFind == "\0" || genreToFind == tv->indexOfTVprogram->indexOfGenre->name) &&

(programToFind == "\0" || programToFind == tv->indexOfTVprogram->name) &&

(channelToFind == "\0" || channelToFind == tv->indexOfChanel->name))

{

counter++;

}

TVshowsPrograms++;

}

if(counter == 0)

{

printf("nothing was found by your request\n");

return 0;

}

printTittle();

TVshowsPrograms.begin();

for(uint i = 0; i < TVshowsPrograms.size(); i++)

{

tv = TVshowsPrograms.get();

if(

(indexToFind == tv->index || indexToFind == -1) &&

(genreToFind == "\0" || genreToFind == tv->indexOfTVprogram->indexOfGenre->name) &&

(programToFind == "\0" || programToFind == tv->indexOfTVprogram->name) &&

(channelToFind == "\0" || channelToFind == tv->indexOfChanel->name))

{

printElement(tv);

}

TVshowsPrograms++;

}

return 0;

}

char DataBase::addAnalizer(std::vector<char\*> splited)

{

if(splited.size() < 5)

{

printf("too few values\n");

return -1;

}

struct tm startTime;

struct tm endTime;

char\* channel = splited[0];

if(dataHandler(startTime, splited[1]) == false) {printf("incorrect start time\n"); return -1;};

if(dataHandler(endTime, splited[2]) == false) {printf("incorrect end time\n"); return -1;};

char\* name = splited[3];

char\* genr = splited[4];

const char\* description = "-";

if(splited.size() == 6)

{

description = splited[5];

}

Genre\* gr = genres.add({genres.size(), genr});

TVprogram\* pr = TVprograms.add({TVprograms.size(), name, gr, description});

Chanel\* ch = chanels.add({chanels.size(), channel});

TVshowsProgram tv = {TVshowsPrograms.size(), startTime, endTime, pr, ch};

TVshowsPrograms.add(tv);

return 0;

}

char DataBase::add(std::string& str)

{

std::vector<char\*> splited = split2(str, '\"');

if(splited.size() == 1)

{

FILE\* f = fopen(splited[0], "r");

if(f == nullptr)

{

printf("file path is incorrect\n");

return -1;

}

char res = 0;

fseek(f, 0, SEEK\_END);

uint lSize = ftell(f);

rewind(f);

fseek(f, 0, SEEK\_SET);

char\* buffer = new char[lSize];

fread(buffer, 1, lSize, f);

std::string bufferStr = buffer;

std::vector<char\*> full = split(bufferStr, 10);

for(uint i = 0; i < full.size(); i++)

{

bufferStr = full[i];

std::vector<char\*> spliter = split2(bufferStr, '\"');

res = addAnalizer(spliter);

if(res != 0)

{

printf("error\n");

break;

}

}

fclose(f);

calculateSpacesArray();

return 0;

}

calculateSpacesArray();

return addAnalizer(splited);

}

void DataBase::remove(uint index)

{

uint indexInList = 0;

TVshowsProgram\* tv;

TVshowsPrograms.begin();

for(uint i = 0; i < TVshowsPrograms.size(); i++)

{

tv = TVshowsPrograms.get();

if(index == tv->index)

{

break;

}

indexInList++;

TVshowsPrograms++;

}

TVshowsPrograms.pop(indexInList);

TVshowsPrograms.begin();

for(uint i = 0; i < TVshowsPrograms.size(); i++)

{

tv = TVshowsPrograms.get();

if(tv->index > index)

{

tv->index--;

}

TVshowsPrograms++;

}

}

void DataBase::sort(std::string command)

{

if(command == "channel")

{

TVshowsPrograms.sort([](TVshowsProgram &x, TVshowsProgram &y) { return x.indexOfChanel->name < y.indexOfChanel->name;});

}

else if(command == "genre")

{

TVshowsPrograms.sort([](TVshowsProgram &x, TVshowsProgram &y) { return x.indexOfTVprogram->indexOfGenre->name < y.indexOfTVprogram->indexOfGenre->name;});

}

else if(command == "index")

{

TVshowsPrograms.sort([](TVshowsProgram &x, TVshowsProgram &y) { return x.index < y.index;});

}

else

{

printf("sort parametr unknow\n");

}

}

void DataBase::printTittle()

{

for(uint i = 0; i < 7; i++)

{

printf("%s", tableNames[i]);

for(uint j = 0; j < spacesArray[i] - namesArray[i] + distBetween; j++)

{

printf(" ");

}

}

printf("\n");

}

void DataBase::printElement(TVshowsProgram\* tv)

{

int temp;

temp = numberToStringSize(tv->index);

printf("%d", tv->index);

for(uint i = 0; i < spacesArray[0] - temp + distBetween; i++) { printf(" "); }

temp = numberToStringSize(tv->indexOfChanel->index);

printf("%s(%d)", tv->indexOfChanel->name.c\_str(), tv->indexOfChanel->index);

for(uint i = 0; i < spacesArray[1] - tv->indexOfChanel->name.size() + distBetween - 2 - temp; i++) { printf(" "); }

printf("%s", tv->indexOfTVprogram->name.c\_str());

for(uint i = 0; i < spacesArray[2] - tv->indexOfTVprogram->name.size() + distBetween; i++) { printf(" "); }

printf("%s", tv->indexOfTVprogram->indexOfGenre->name.c\_str());

for(uint i = 0; i < spacesArray[3] - tv->indexOfTVprogram->indexOfGenre->name.size() + distBetween; i++) { printf(" "); }

printDate(&tv->startTime);

for(uint i = 0; i < distBetween; i++) { printf(" "); }

printDate(&tv->endTime);

for(uint i = 0; i < distBetween; i++) { printf(" "); }

printf("%s\n", tv->indexOfTVprogram->description.c\_str());

}

void DataBase::calculateSpacesArray()

{

TVshowsProgram\* tv;

int temp;

TVshowsPrograms.begin();

for(uint i = 0; i < TVshowsPrograms.size(); i++)

{

tv = TVshowsPrograms.get();

temp = numberToStringSize(tv->index);

if(spacesArray[0] < temp) { spacesArray[0] = temp;}

temp = numberToStringSize(tv->indexOfChanel->index);

if(spacesArray[1] < tv->indexOfChanel->name.size() + 2 + temp) { spacesArray[1] = tv->indexOfChanel->name.size() + 2 + temp;}

if(spacesArray[2] < tv->indexOfTVprogram->name.size()) { spacesArray[2] = tv->indexOfTVprogram->name.size();}

if(spacesArray[3] < tv->indexOfTVprogram->indexOfGenre->name.size()) { spacesArray[3] = tv->indexOfTVprogram->indexOfGenre->name.size();}

TVshowsPrograms++;

}

}

void DataBase::print()

{

TVshowsProgram\* tv;

int temp;

printTittle();

TVshowsPrograms.begin();

for(uint j = 0; j < TVshowsPrograms.size(); j++)

{

tv = TVshowsPrograms.get();

printElement(tv);

TVshowsPrograms++;

}

}

void DataBase::save(const char\* filename)

{

FILE\* f = fopen(filename, "wb");

if(f == NULL)

{

printf("file couldn't create\n");

return;

}

uint size = 0;

uint sizeStr = 0;

genres.begin();

size = genres.size();

fwrite(&size, sizeof(uint), 1, f);

for(uint i = 0; i < size; i++)

{

sizeStr = genres.get()->name.size();

fwrite(&genres.get()->index, sizeof(uint), 1, f);

fwrite(&sizeStr, sizeof(uint), 1, f);

fwrite(genres.get()->name.c\_str(), sizeof(char), genres.get()->name.size(), f);

genres++;

}

chanels.begin();

size = chanels.size();

fwrite(&size, sizeof(uint), 1, f);

for(uint i = 0; i < size; i++)

{

sizeStr = chanels.get()->name.size();

fwrite(&chanels.get()->index, sizeof(uint), 1, f);

fwrite(&sizeStr, sizeof(uint), 1, f);

fwrite(chanels.get()->name.c\_str(), sizeof(char), chanels.get()->name.size(), f);

chanels++;

}

TVprograms.begin();

size = TVprograms.size();

fwrite(&size, sizeof(uint), 1, f);

for(uint i = 0; i < size; i++)

{

sizeStr = TVprograms.get()->name.size();

fwrite(&TVprograms.get()->index, sizeof(uint), 1, f);

fwrite(&sizeStr, sizeof(uint), 1, f);

fwrite(TVprograms.get()->name.c\_str(), sizeof(char), TVprograms.get()->name.size(), f);

sizeStr = TVprograms.get()->description.size();

fwrite(&sizeStr, sizeof(uint), 1, f);

fwrite(TVprograms.get()->description.c\_str(), sizeof(char), TVprograms.get()->description.size(), f);

fwrite(&TVprograms.get()->indexOfGenre->index, sizeof(uint), 1, f);

TVprograms++;

}

TVshowsPrograms.begin();

size = TVshowsPrograms.size();

fwrite(&size, sizeof(uint), 1, f);

for(uint i = 0; i < size; i++)

{

fwrite(&TVshowsPrograms.get()->index, sizeof(uint), 1, f);

fwrite(&TVshowsPrograms.get()->startTime, sizeof(struct tm), 1, f);

fwrite(&TVshowsPrograms.get()->endTime, sizeof(struct tm), 1, f);

fwrite(&TVshowsPrograms.get()->indexOfTVprogram->index, sizeof(uint), 1, f);

fwrite(&TVshowsPrograms.get()->indexOfChanel->index, sizeof(uint), 1, f);

TVshowsPrograms++;

}

fclose(f);

}

void DataBase::load(const char\* filename)

{

FILE\* f = fopen(filename, "rb");

if(f == nullptr)

{

printf("file couldn't get save file\n");

return;

}

uint indexToFind = 0;

char\* str;

uint str\_size = 0;

uint size = 0;

fread(&size, sizeof(uint), 1, f);

for(uint i = 0; i < size; i++)

{

Genre ge;

fread(&ge.index, sizeof(uint), 1, f);

fread(&str\_size, sizeof(uint), 1, f);

str = new char[str\_size + 1];

str[str\_size] = 0;

fread(str, sizeof(char), str\_size, f);

ge.name = std::string(str);

genres.add(ge);

}

fread(&size, sizeof(uint), 1, f);

for(uint i = 0; i < size; i++)

{

Chanel ch;

fread(&ch.index, sizeof(uint), 1, f);

fread(&str\_size, sizeof(uint), 1, f);

str = new char[str\_size + 1];

str[str\_size] = 0;

fread(str, sizeof(char), str\_size, f);

ch.name = std::string(str);

chanels.add(ch);

}

fread(&size, sizeof(uint), 1, f);

for(uint i = 0; i < size; i++)

{

TVprogram pr;

fread(&pr.index, sizeof(uint), 1, f);

fread(&str\_size, sizeof(uint), 1, f);

str = new char[str\_size + 1];

str[str\_size] = 0;

fread(str, sizeof(char), str\_size, f);

pr.name = std::string(str);

fread(&str\_size, sizeof(uint), 1, f);

str = new char[str\_size + 1];

str[str\_size] = 0;

fread(str, sizeof(char), str\_size, f);

pr.description = std::string(str);

fread(&indexToFind, sizeof(uint), 1, f);

genres.begin();

for(uint i = 0; i < genres.size(); i++)

{

if(genres.get()->index == indexToFind)

{

pr.indexOfGenre = genres.get();

}

genres++;

}

TVprograms.add(pr);

}

fread(&size, sizeof(uint), 1, f);

for(uint i = 0; i < size; i++)

{

TVshowsProgram tv;

fread(&tv.index, sizeof(uint), 1, f);

fread(&tv.startTime, sizeof(struct tm), 1, f);

fread(&tv.endTime, sizeof(struct tm), 1, f);

fread(&indexToFind, sizeof(uint), 1, f);

TVprograms.begin();

for(uint i = 0; i < TVprograms.size(); i++)

{

if(TVprograms.get()->index == indexToFind)

{

tv.indexOfTVprogram = TVprograms.get();

}

TVprograms++;

}

fread(&indexToFind, sizeof(uint), 1, f);

chanels.begin();

for(uint i = 0; i < chanels.size(); i++)

{

if(chanels.get()->index == indexToFind)

{

tv.indexOfChanel = chanels.get();

}

chanels++;

}

TVshowsPrograms.add(tv);

}

fclose(f);

calculateSpacesArray();

}

## Файл functions.h

#pragma once

#include <time.h>

#include <vector>

#include <string>

std::vector<char\*> split(std::string& str, const char splitSymbol);

std::vector<char\*> split2(std::string& str, const char splitSymbol);

bool dataHandler(struct tm& date, char\* dataStr);

void printDate(struct tm\* date);

int numberToStringSize(int number);

char stringToInt(int& number, char\* str);

bool is\_digit(unsigned char symbol);

int checkDayByMouthAndYear(int m, int y);

## Файл functions.cpp

#include "functions.h"

#include <stdio.h>

std::vector<char\*> split(std::string& str, const char splitSymbol)

{

str[str.size()] = splitSymbol;

int firstLetterAfterSplit = 0;

std::vector<char\*> charList;

for(int i = 0; i < str.size() + 1; i++)

{

if(str[i] == splitSymbol)

{

int size = i - firstLetterAfterSplit;

if(size > 0)

{

char\* new\_word = new char[size + 1];

for(int j = firstLetterAfterSplit; j < i; j++)

{

new\_word[j - firstLetterAfterSplit] = str[j];

}

new\_word[size] = '\0';

charList.push\_back(new\_word);

}

firstLetterAfterSplit = i + 1;

continue;

}

}

str[str.size()] = '\n';

return charList;

}

std::vector<char\*> split2(std::string& str, const char splitSymbol)

{

int firstLetterAfterSplit = 0x00'FF'FF'FF;

std::vector<char\*> charList;

for(int i = 0; i < str.size() + 1; i++)

{

if(str[i] == splitSymbol)

{

int size = i - firstLetterAfterSplit;

if(size > 0)

{

char\* new\_word = new char[size + 1];

for(int j = firstLetterAfterSplit; j < i; j++)

{

new\_word[j - firstLetterAfterSplit] = str[j];

}

new\_word[size] = '\0';

charList.push\_back(new\_word);

firstLetterAfterSplit = 0x00'FF'FF'FF;

continue;

}

firstLetterAfterSplit = i + 1;

continue;

}

}

return charList;

}

bool dataHandler(struct tm& date, char\* dataStr)

{

std::string big = dataStr;

std::vector<char\*> bigOne = split(big, ' ');

if(bigOne.size() != 2)

{

return false;

}

std::string time = bigOne[0];

std::string data = bigOne[1];

std::vector<char\*> timeOne = split(time, ':');

std::vector<char\*> dateOne = split(data, '.');

if(timeOne.size() != 2 || dateOne.size() != 3)

{

return false;

}

if(stringToInt(date.tm\_hour, timeOne[0])) { return false;};

if(stringToInt(date.tm\_min, timeOne[1])) { return false;};

if(stringToInt(date.tm\_mday, dateOne[0])) { return false;};

if(stringToInt(date.tm\_mon, dateOne[1])) { return false;};

if(stringToInt(date.tm\_year, dateOne[2])) { return false;};

if(date.tm\_hour > 23 || date.tm\_min > 59 || date.tm\_mon > 12 || date.tm\_mday > checkDayByMouthAndYear(date.tm\_mon, date.tm\_year)) { return false;}

date.tm\_mon -= 1;

date.tm\_year -= 1900;

return true;

}

void printDate(struct tm\* date)

{

if(date->tm\_hour < 10)

{

printf("0");

}

printf("%d:", date->tm\_hour);

if(date->tm\_min < 10)

{

printf("0");

}

printf("%d ", date->tm\_min);

if(date->tm\_mday < 10)

{

printf("0");

}

printf("%d.", date->tm\_mday);

if(1 + date->tm\_mon < 10)

{

printf("0");

}

printf("%d.%d", 1 + date->tm\_mon, 1900 + date->tm\_year);

}

bool is\_digit(unsigned char symbol)

{

if(47 < symbol && symbol < 58)

{

return true;

}

return false;

}

int numberToStringSize(int number)

{

int size = 0;

if(number == 0)

{

return 1;

}

while(number > 0)

{

number/=10;

size++;

}

return size;

}

char stringToInt(int& number, char\* str)

{

number = 0;

while(\*str != '\0')

{

if(is\_digit(\*str) == false)

{

return 1;

}

number \*= 10;

number += \*str - 48;

str++;

}

return 0;

}

int checkDayByMouthAndYear(int m, int y)

{

int daysCount[12] = {

31, //january

28, //february

31, //march

30, //april

31, //may

30, //june

31, //july

31, //august

30, //september

31, //october

30, //november

31 //december

};

if(m == 2)

{

return daysCount[m - 1] + (y % 4 == 0 && ((y % 100 == 0 && y % 400 == 0) || y % 100 != 0));

}

return daysCount[m - 1];

}

## Файл main.cpp

#include <iostream>

#include <string>

#include <vector>

#include "dataBase.h"

#include "functions.h"

#include <algorithm>

using namespace std;

const char\* commands[] = {

"help", //0

"exit", //1

"print", //2

"add ", //3

"find ", //4

"sort ", //5

"remove ", //6

};

const char\* commandsDescriptions[] = {

":\n\tshow u all the commands ant their descriptions", //0

":\n\tfinish the main programm", //1

":\n\twill print the table of data base", //2

"\b:\n\tadd \"Channel Name\" \"start time of the tv programm\" \"end time of the tv programm\" \"programm name\" \"programm genre\" \"programm description\"\n\tadd new data to the data base", //3

"\b:\n\tfind {conditions}\n\tconditions: -index="" -genre="" -program="" -channel=""\n\tprint all the elementd of data base which satisfy the conditions", //4

"\b:\n\tsort {parametr}\n\tparametrs: {index}, {genre}, {channel}\n\twill sort the data base by the parametr", //5

"\b:\n\tremove {index}\n\tremove the element of data base by index", //6

};

int main()

{

system("cls");

const char\* saveFile = "./save.txt";

uint commands\_size = sizeof(commands) / sizeof(const char\*);

string command;

DataBase dataBase;

dataBase.load(saveFile);

while(1)

{

printf("input command: ");

getline(cin, command);

if(command == "0" ||command == commands[0])

{

for(uint i = 0; i < commands\_size; i++)

{

printf("%d) %s%s\n", i, commands[i], commandsDescriptions[i]);

}

}

else if(command == "1" ||command == commands[1])

{

break;

}

else if(command == "2" ||command == commands[2])

{

dataBase.print();

}

else if(command.find("3") == 0)

{

command[0] = ' ';

dataBase.add(command);

}

else if(command.find(commands[3]) == 0)

{

command[0] = ' ';

command[1] = ' ';

command[2] = ' ';

dataBase.add(command);

}

else if(command.find("4") == 0 || command.find(commands[4]) == 0)

{

command[0] = '-';

command[1] = '-';

command[2] = '-';

command[3] = '-';

dataBase.find(command);

}

else if(command.find("5") == 0 || command.find(commands[5]) == 0)

{

std::vector<char\*> l = split(command, ' ');

dataBase.sort(l[1]);

}

else if(command.find("6") == 0 || command.find(commands[6]) == 0)

{

int number = 0;

for(uint i = 0; i < 6; i++) {command[i] = ' ';};

command.erase(std::remove(command.begin(), command.end(), ' '), command.end());

if(stringToInt(number, const\_cast<char\*>(command.c\_str())) == true)

{

printf("'%s' is not number\n", command.c\_str());

}

dataBase.remove(number);

}

else

{

printf("command: \"%s\" is unknown.\nuse \"help\" to know more\n", command.c\_str());

}

printf("\n");

}

dataBase.save(saveFile);

return 0;

}

# Результаты проверки функционирования программы

## Добавление в базу данных с клавиатуры

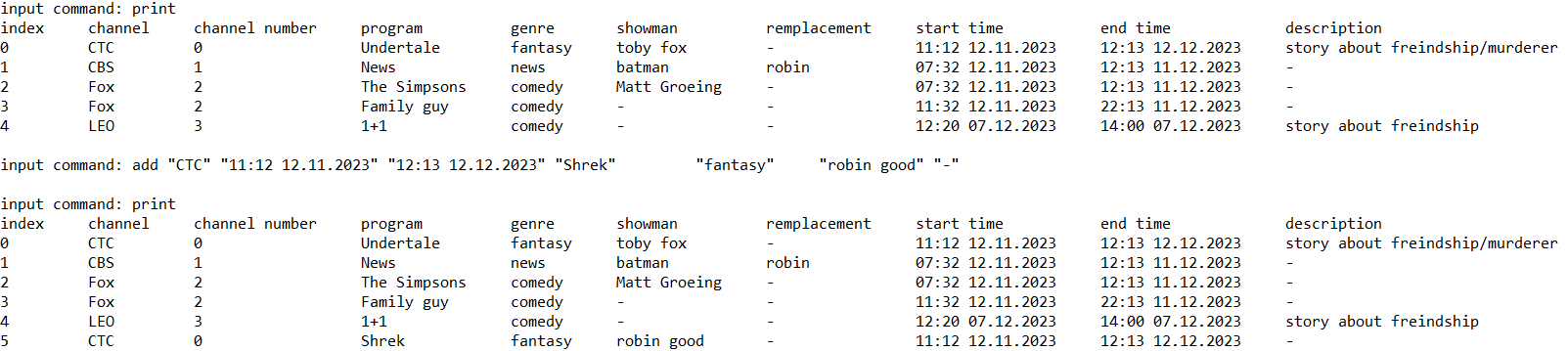


Рисунок 8. Процесс добавления в базу данных с клавиатуры.

## Добавление в базу данных из файла

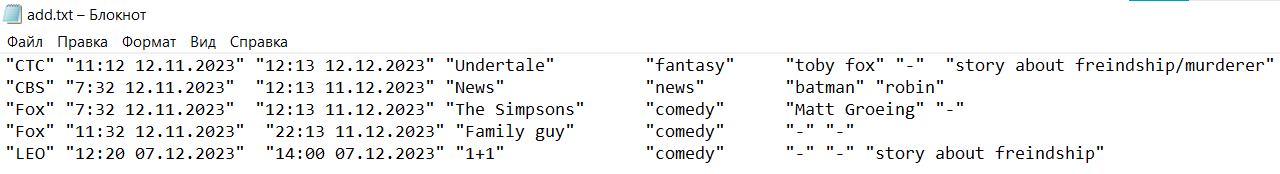


Рисунок 9. Исходные данные.



Рисунок 10. Процесс добавления из файла.

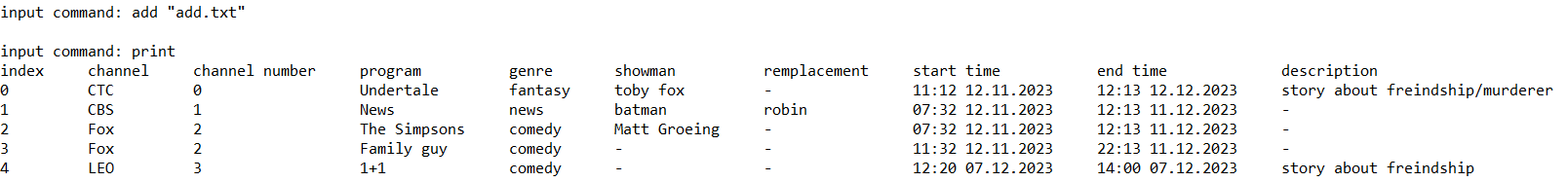


Рисунок 11. Вывод в консоль данных добавленных из файла.

## Удаление из базы данных

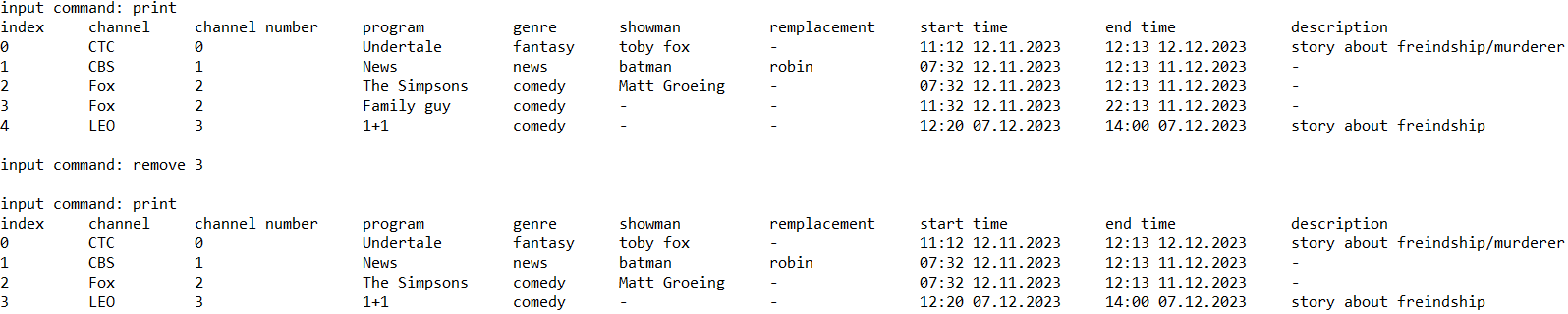


Рисунок 12. Процесс удаления из базы данных.

## Сортировка базы данных по определенному полю



Рисунок 13 Сортировка базы данных по ТВ каналам.

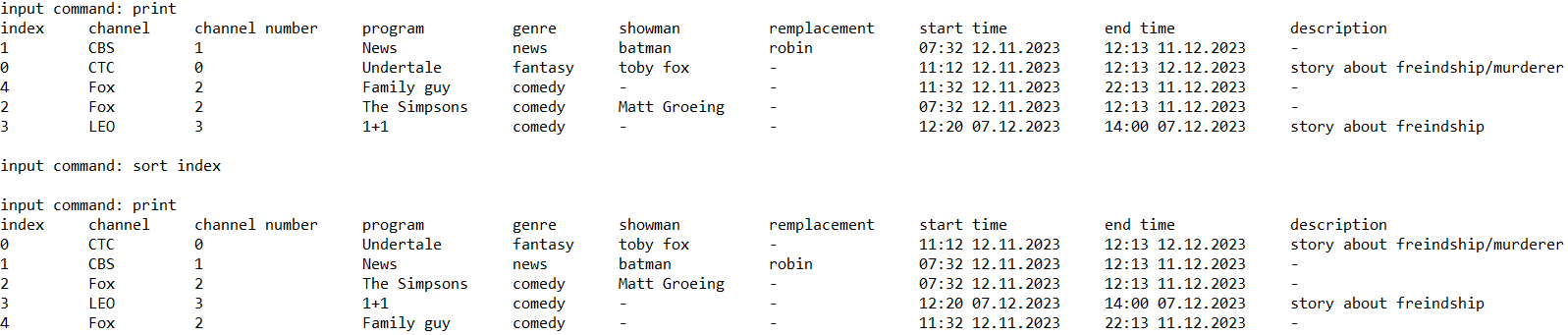


Рисунок 14. Сортировка базы данных по индексу программы телеперадачи.

## Поиск в базе данных по заданным условиям

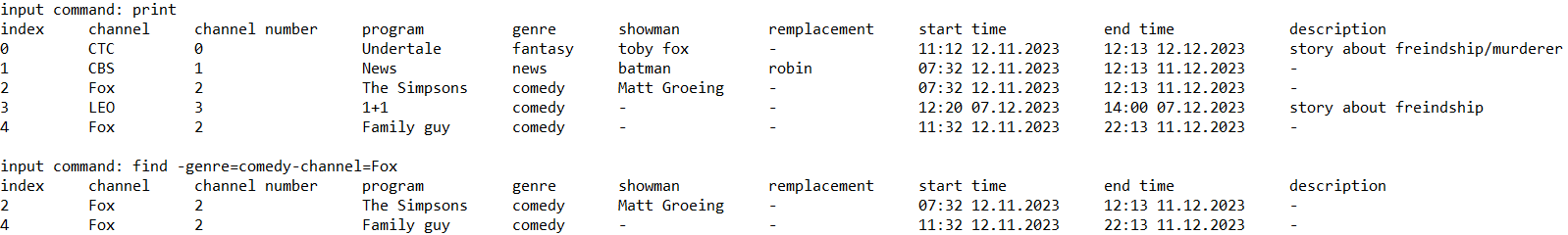


Рисунок 15. Поиск в базе данных по жанру Комедия и каналу Fox.

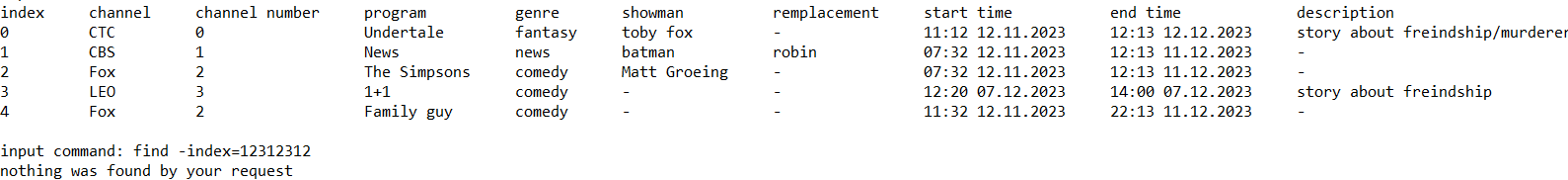


Рисунок 16. Поиск в базе данных по несуществующему индексу

# Вывод

В результате выполнения курсовой работы была изучена и написана такая структура данных, как линейный односвязный список, а также функции для ее обработки.

Выбор линейного односвязного списка основывался на его оптимальности для написания работы в сравнении с другими видами связных списков, что было изучено и подтверждено в ходе выполнения курсовой работы.

Основными критериями оптимальности линейного односвязного списка являются:

1. самый простой тип связанного списка, в котором каждый узел содержит некоторые данные и указатель на следующий узел с тем же типом данных. Узел, содержащий указатель на следующий узел, означает, что узел хранит адрес следующего узла в последовательности. Единый связанный список допускает обход данных только одним способом.
2. Односвязные списки экономят место, поскольку им нужно хранить только указатель на следующий узел в каждом элементе, а не на предыдущие элементы списка как в двухсвязном списке
3. Т.к односвязный линейный список содержит только указатель на след элемент, то многие операции над односвязными линейными списками (например, объединение двух списков) часто имеют очень простые рекурсивные алгоритмы, намного более простые, чем любое решение, использующее итеративные команды. Хотя эти рекурсивные решения могут быть адаптированы для двусвязных и циклически связанных списков, процедурам обычно требуются дополнительные аргументы и более сложные базовые случаи.

В ходе выполнения курсовой работы использовались такие средства языка C++, как классы файловые и строковые потоки ввода-вывода, регулярные выражения.