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: «Государство»

Выполнил:

студент группы М3О-209Б-23 Черных А.П.

.

Проверил:  
доцент каф. 304, к.т.н. Дмитриева Е.А.

Москва 2024 г.

Содержание

[Задание 4](#_Toc185407397)

[Введение 5](#_Toc185407398)

[Блок-схемы функций 6](#_Toc185407399)

[Блок-схема функции поиска в списке по параметру 6](#_Toc185407400)

[Блок-схема функции добавления в список 7](#_Toc185407401)

[Блок-схема функции обновления базы данных на основе изменений в очереди 8](#_Toc185407402)

[Блок-схема основной функции 9](#_Toc185407403)

[Процесс разработки 10](#_Toc185407404)

[Описание структур 10](#_Toc185407405)

[Структура хранения данных о стране 10](#_Toc185407406)

[Структура элемента двусвязного списка 11](#_Toc185407407)

[Описание функций 12](#_Toc185407408)

[Функция возвращения поля по индикатору 12](#_Toc185407409)

[Функция возвращения поля по индикатору (вторая) 12](#_Toc185407410)

[Функция поиска по параметру 12](#_Toc185407411)

[Функция добавления записи с данными с клавиатуры 13](#_Toc185407412)

[Функция считывания данных из файла и заполнения базы данных 13](#_Toc185407413)

[Функция создания нового элемента списка (базы данных) 14](#_Toc185407414)

[Функция добавления элемента в конец списка (базы данных) 14](#_Toc185407415)

[Функция удаления записи из базы данных по параметру 15](#_Toc185407416)

[Функция вывода базы данных на экран 16](#_Toc185407417)

[Функция сортировки базы данных по заданному параметру 16](#_Toc185407418)

[Функция изменения очереди 17](#_Toc185407419)

[Функция обновления базы данных 18](#_Toc185407420)

[Функция создания элемента очереди 18](#_Toc185407421)

[Основная функция 19](#_Toc185407422)

[Результаты работы 20](#_Toc185407423)

[Корректные тесты 20](#_Toc185407424)

[Некорректные тесты 23](#_Toc185407425)

[Заключение 25](#_Toc185407426)

[Приложение А. Код программы 26](#_Toc185407427)

# Задание

Разработать базу данных на языке С++.

Тематики базы данных выбираются согласно варианту.

Для хранения данных использовать текстовый файл.

Разработать структуры данных для хранения информации и функции:

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.

Схемы организации данных:

1. Линейные односвязные списки
2. Линейные двусвязные списки
3. Стеки
4. Очереди

Тематика базы данных:

Вариант 18. «Государство» : название страны; столица; государственный язык; население(число жителей); площадь территории; денежная единица; курс валюты относительно рубля(доллара); государственный строй. В программе реализовываются следующие схемы организации данных: линейные односвязные списки, очереди с приоритетом.

# Введение

Целью данной курсовой работы является разработка базы данных для учёта государств и их характеристик с использованием языка программирования C++ и хранения данных в текстовых файлах. В рамках работы планируется создание системы для управления данными государств, включающей функционал для ввода, вывода, сортировки, удаления и записи новых данных, а также для выборки информации по заданным условиям.

Для решения поставленной задачи будет использован ряд технологий и подходов, включая работу с динамическими структурами данных, такими как линейные односвязные списки и очереди с приоритетами. Эти структуры обеспечат гибкость и эффективность при обработке данных, а также позволят организовать взаимодействие с пользователем через систему меню.

Задачи работы включают:

* Разработку структур данных для хранения информации о государствах, таких как название страны, столица, государственный язык, население, площадь территории, денежная единица, курс валюты относительно рубля, государственный строй.
* Реализацию функций для ввода, вывода и сортировки данных, а также для удаления и записи новых записей.
* Обеспечение выполнения запросов по любым полям данных.
* Организацию динамических структур для обработки данных с использованием линейных односвязных списков и очередей с приоритетом.

Работа состоит из нескольких разделов, включая описание структур данных и функций программы, а также блок-схемы, иллюстрирующие основные алгоритмы и кода программы (см. [Приложение А](#_Приложение_А._Код)). В заключении приведены результаты работы программы и её оценка.

Входными данные программы – текстовый файл, содержащий информацию о 10 государствах.

# Блок-схемы функций

## Блок-схема функции поиска в списке по параметру

void searchParam(NodeList\* headList, string param, int indicate)

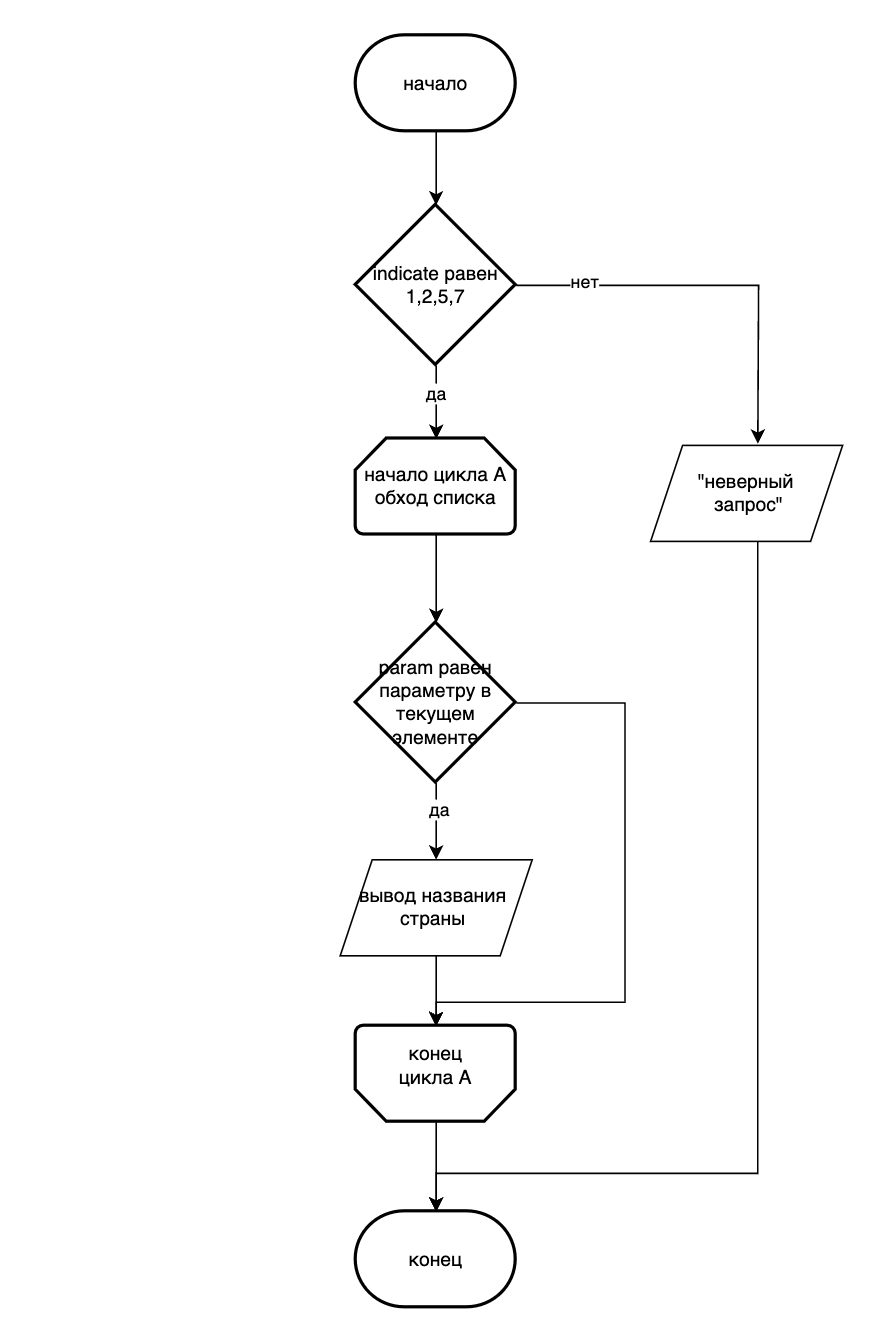


Рисунок 1 - Блок-схема функции поиска по параметру

## Блок-схема функции добавления в список

void appendList(NodeList\*\* headList, listParam newData)

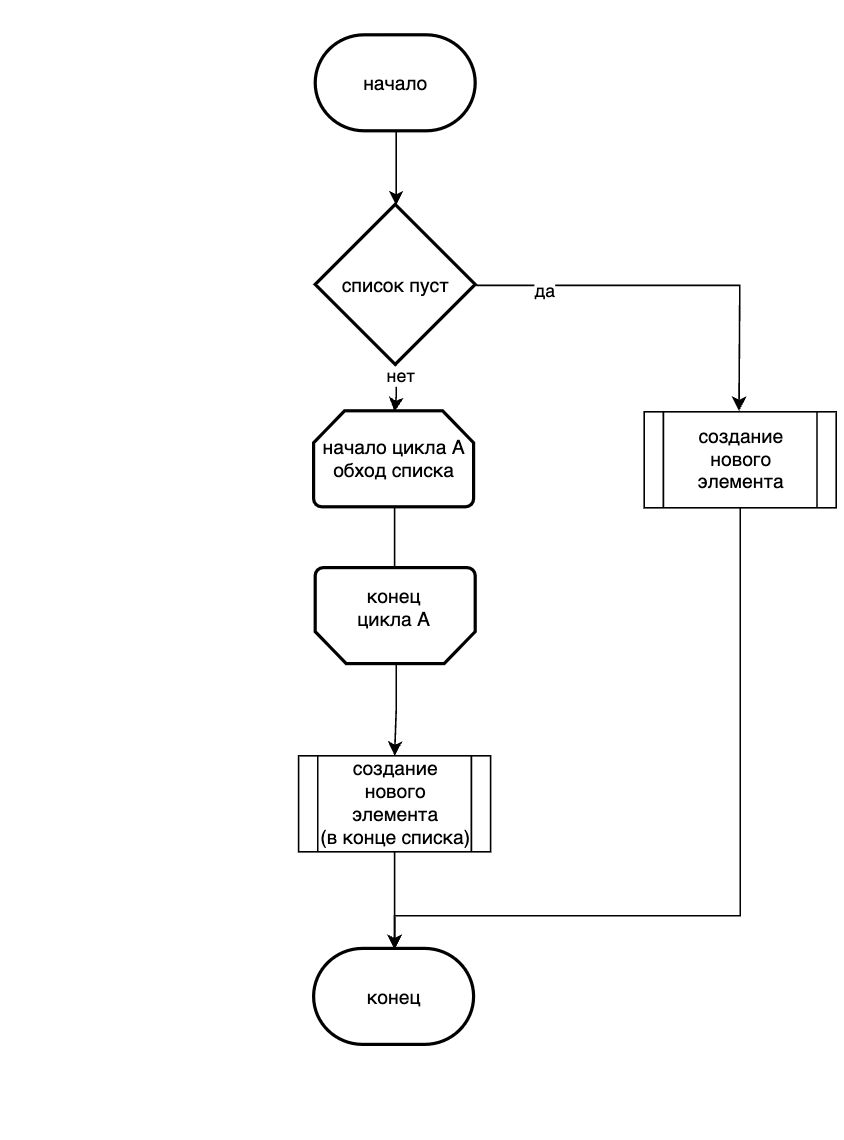


Рисунок 2 - Блок-схема функции добавления элемента в список

## Блок-схема функции обновления базы данных на основе изменений в очереди

void updateBase(NodeList\* headQueue, NodeList\* headList)

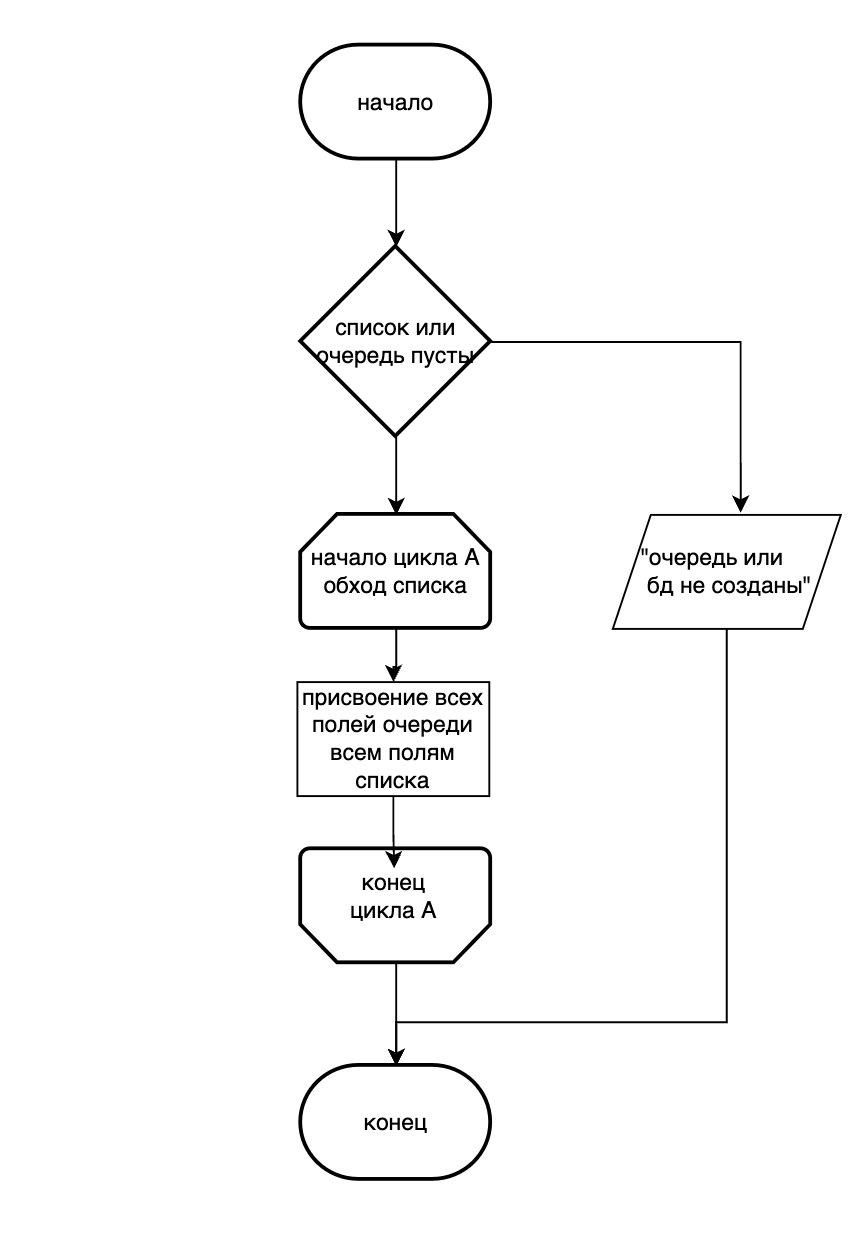


Рисунок 3 – Блок-схема функции обновления базы данных на основе изменений в очереди

## Блок-схема основной функции

int main()

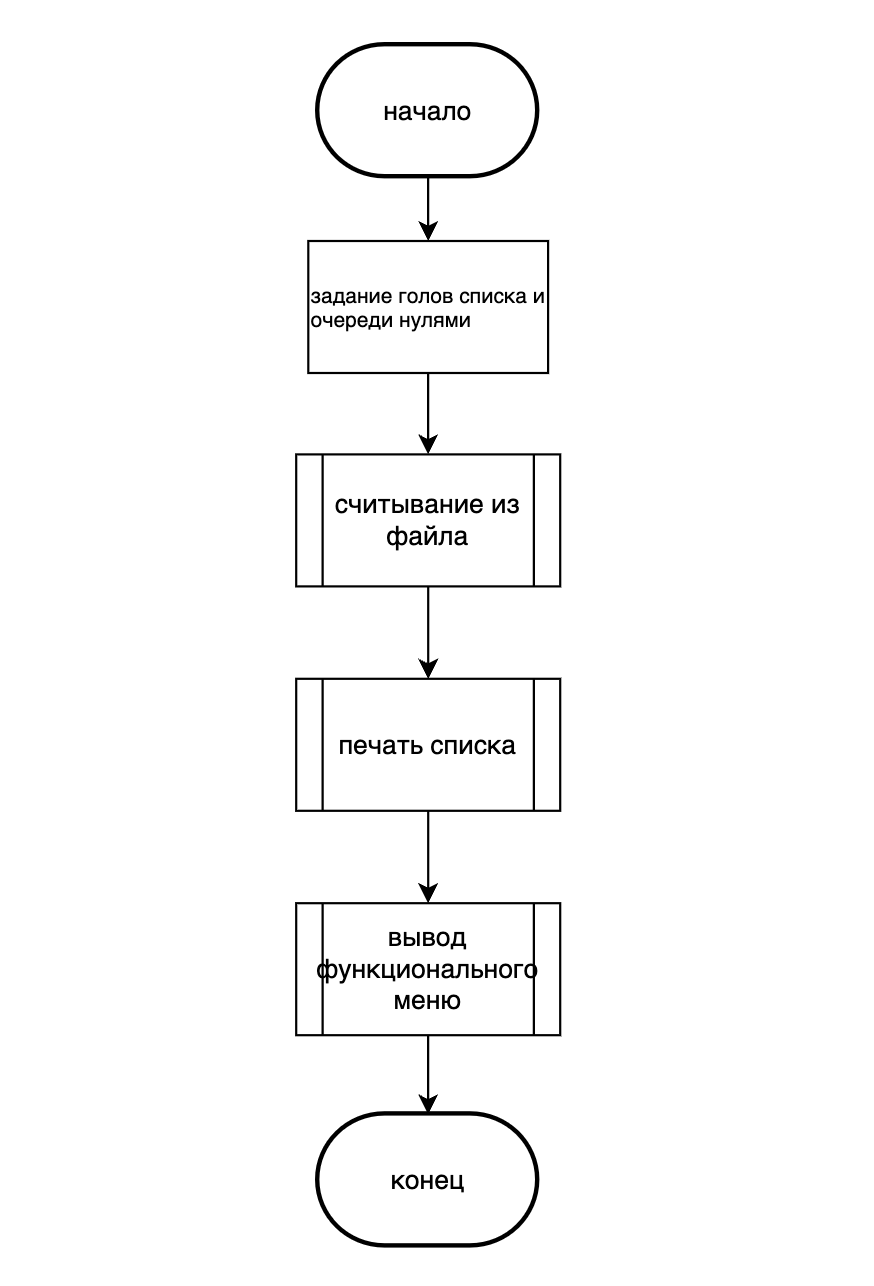


Рисунок 4 - Блок-схема основной функции

# Процесс разработки

## Описание структур

### Структура хранения данных о стране

Назначение: структура предназначена для хранения данных о стране

Прототип структуры:

struct listParam

{

string country;

string capital;

string language;

int population;

int square;

string currency;

int currencyRateRuble;

string politics;

};

Описание полей:

Таблица 1 – Описание полей структуры listParam

|  |  |  |
| --- | --- | --- |
| Имя параметра | Тип параметра | Назначение |
| country | string | Название страны |
| capital | string | Столица страны |
| language | string | Язык страны |
| population | int | Население страны |
| square | int | Площадь территории страны |
| currency | string | Валюта страны |
| currencyRateRuble | int | Курс валюты к рублю |
| politics | string | Политика страны |

### Структура элемента двусвязного списка

Назначение: структура для реализации элемента двусвязного списка

Прототип структуры:

struct NodeList {

NodeList\* next;

listParam paramsData;

};

Описание полей:

Таблица 3 – Описание полей структуры NodeList

|  |  |  |
| --- | --- | --- |
| Имя параметра | Тип параметра | Назначение |
| paramsData | listParam | Данные о стране |
| next | NodeList \* | Указатель на следующий элемент списка |

## Описание функций

### Функция возвращения поля по индикатору

Назначение: по индикатору возвращает нужное поле базы данных

Прототип функции:

string indicParam1(NodeList\* curNodeList, int indicate)

Код функции:

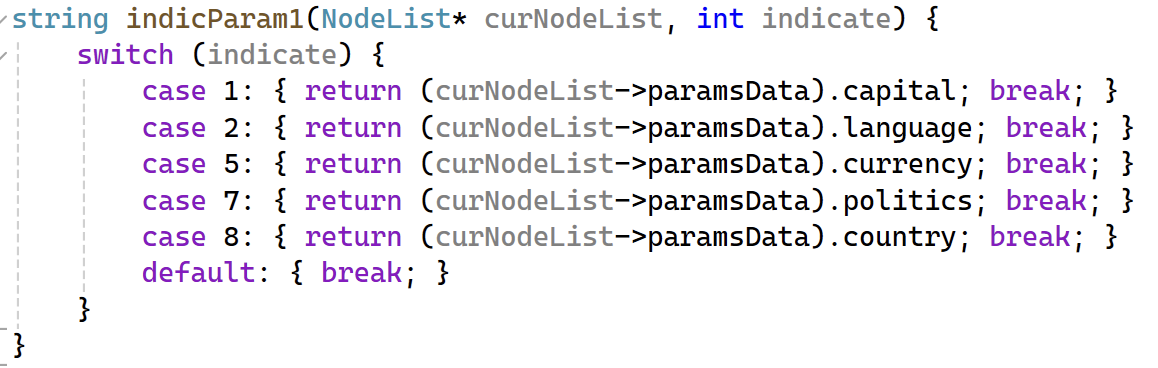


Рисунок 5 - Код функции indicParam1

### Функция возвращения поля по индикатору (вторая)

Назначение: по индикатору возвращает нужное поле базы данных

Прототип функции:

int indicParam2(NodeList\* curNodeList, int indicate)

Код функции:

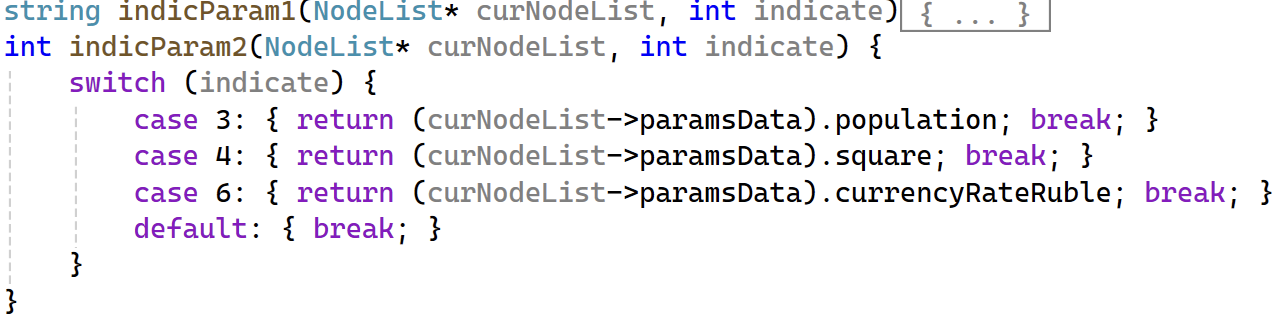


Рисунок 6 - Код функции indicParam2

### Функция поиска по параметру

Назначение: выполняет поиск по параметру

Прототип функции:

void searchParam(NodeList\* headList, string param, int indicate)

Код функции:

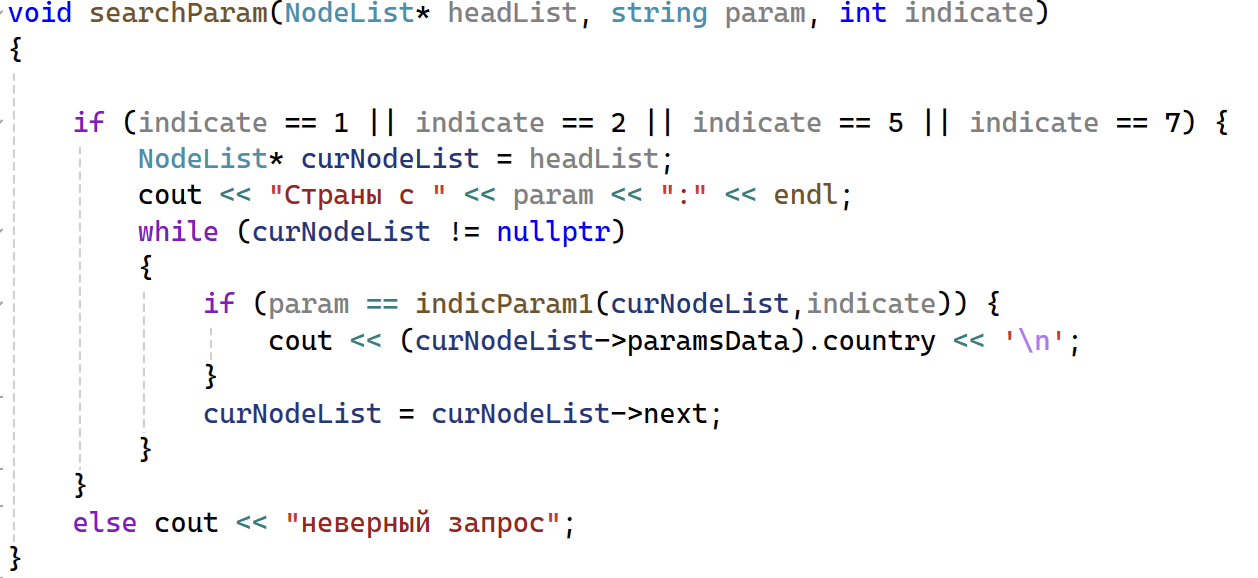


Рисунок 7 - Код функции searchParam

### Функция добавления записи с данными с клавиатуры

Назначение: добавляет новую запись в базу данных

Прототип функции:

void appendBase(NodeList\*\* headList)

Код функции:

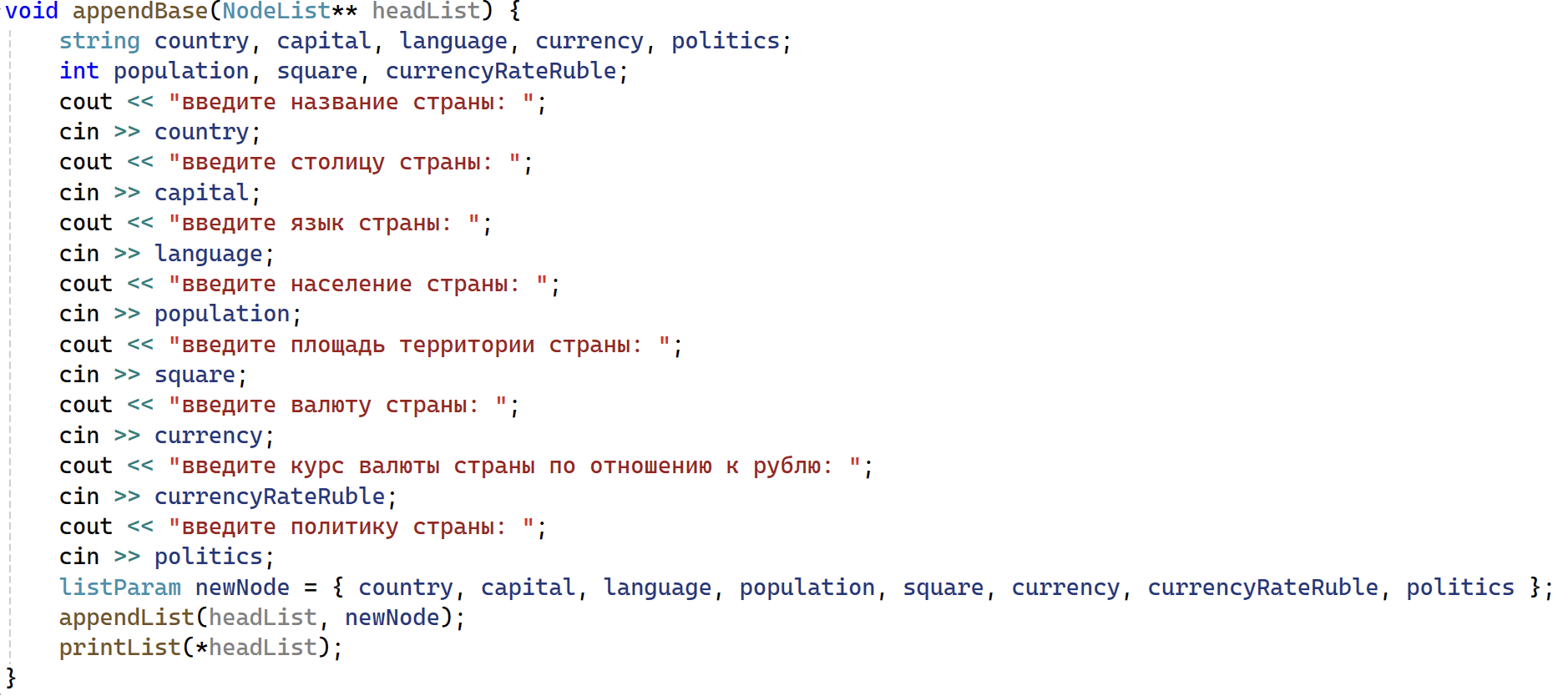


Рисунок 8 - Код функции appendBase

### Функция считывания данных из файла и заполнения базы данных

Назначение: считывает данные из файла и заполняет базу данных

Прототип функции:

void inputFile(NodeList\*\* headList)

Код функции:



Рисунок 8 - Код функции inputFile

### Функция создания нового элемента списка (базы данных)

Назначение: создает запись

Прототип функции:

NodeList\* createNodeList(listParam newData)

Код функции:

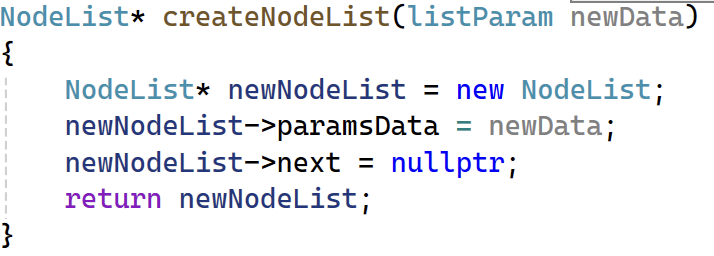


Рисунок 9 - Код функции createNodeList

### Функция добавления элемента в конец списка (базы данных)

Назначение: добавляет запись в конец списка

Прототип функции:

void appendList(NodeList\*\* headList, listParam newData)

Код функции:

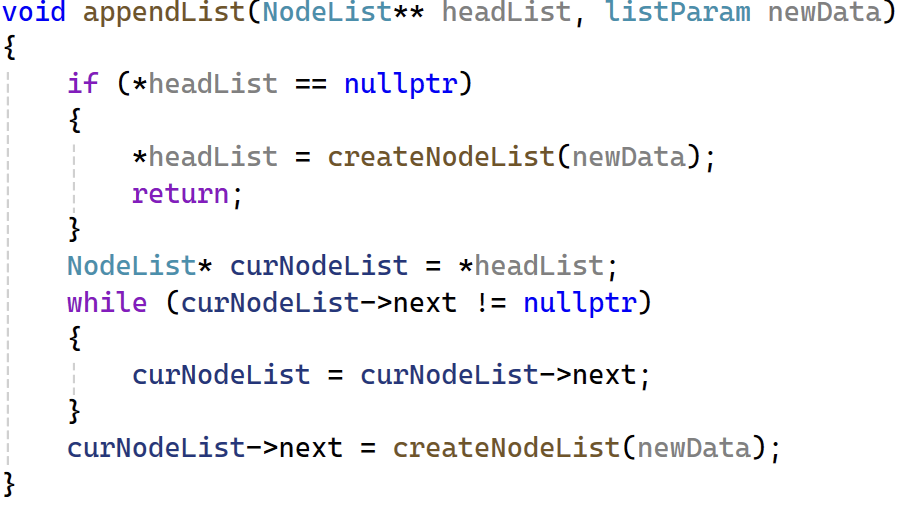


Рисунок 10 - Код функции appendList

### Функция удаления записи из базы данных по параметру

Назначение: удаляет запись по параметру

Прототип функции:

void deleteNodeListParam(NodeList\*\* headList, string param, int indicate)

Код функции:

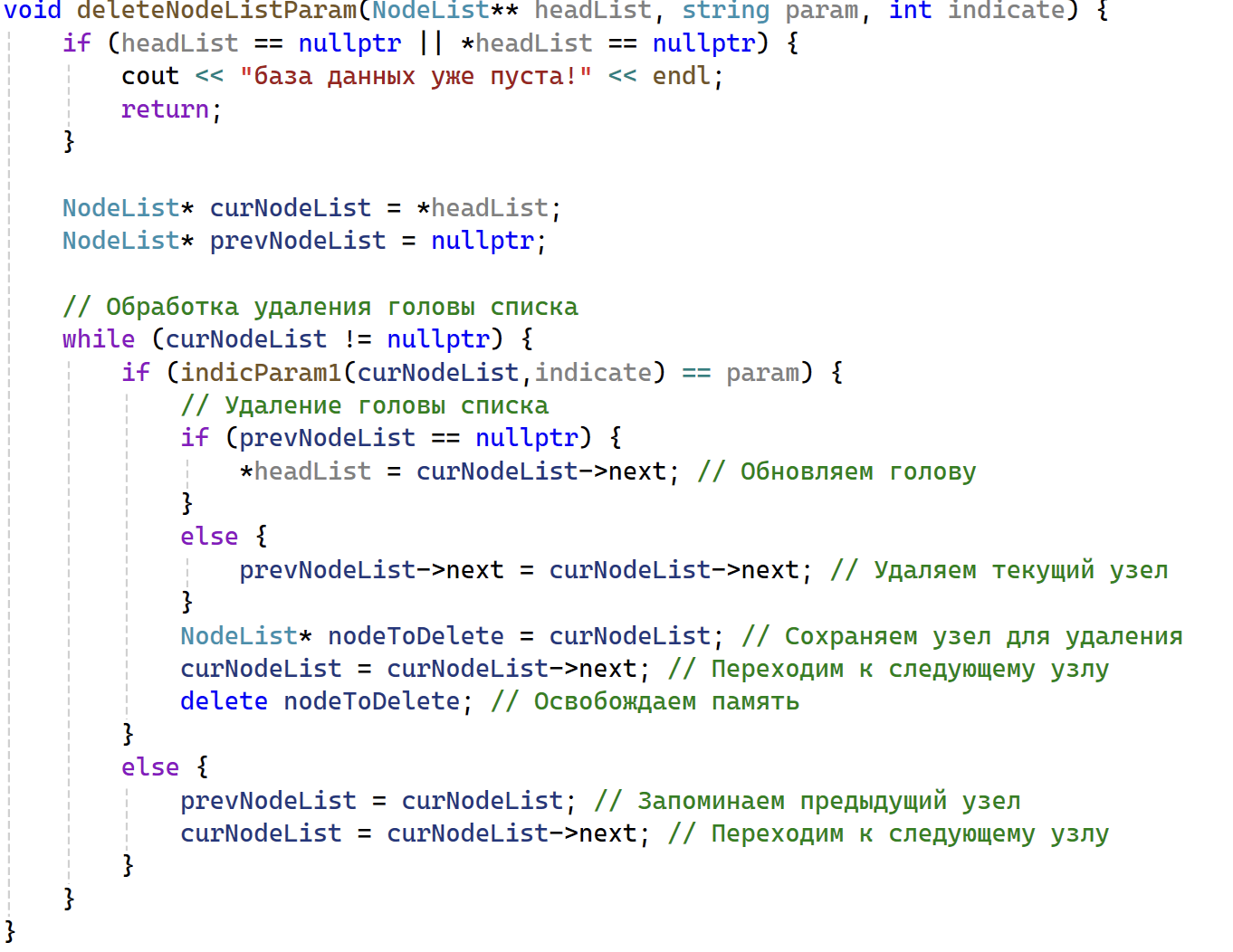


Рисунок 11 - Код функции deleteNodeListParam

### Функция вывода базы данных на экран

Назначение: выводит на экран базу данных

Прототип функции:

void printList(NodeList\* headList)

Код функции:



Рисунок 12 - Код функции printList

### Функция сортировки базы данных по заданному параметру

Назначение: сортирует базу данных по параметру

Прототип функции:

void sortList(NodeList\*& headList, int indicate)

Код функции:

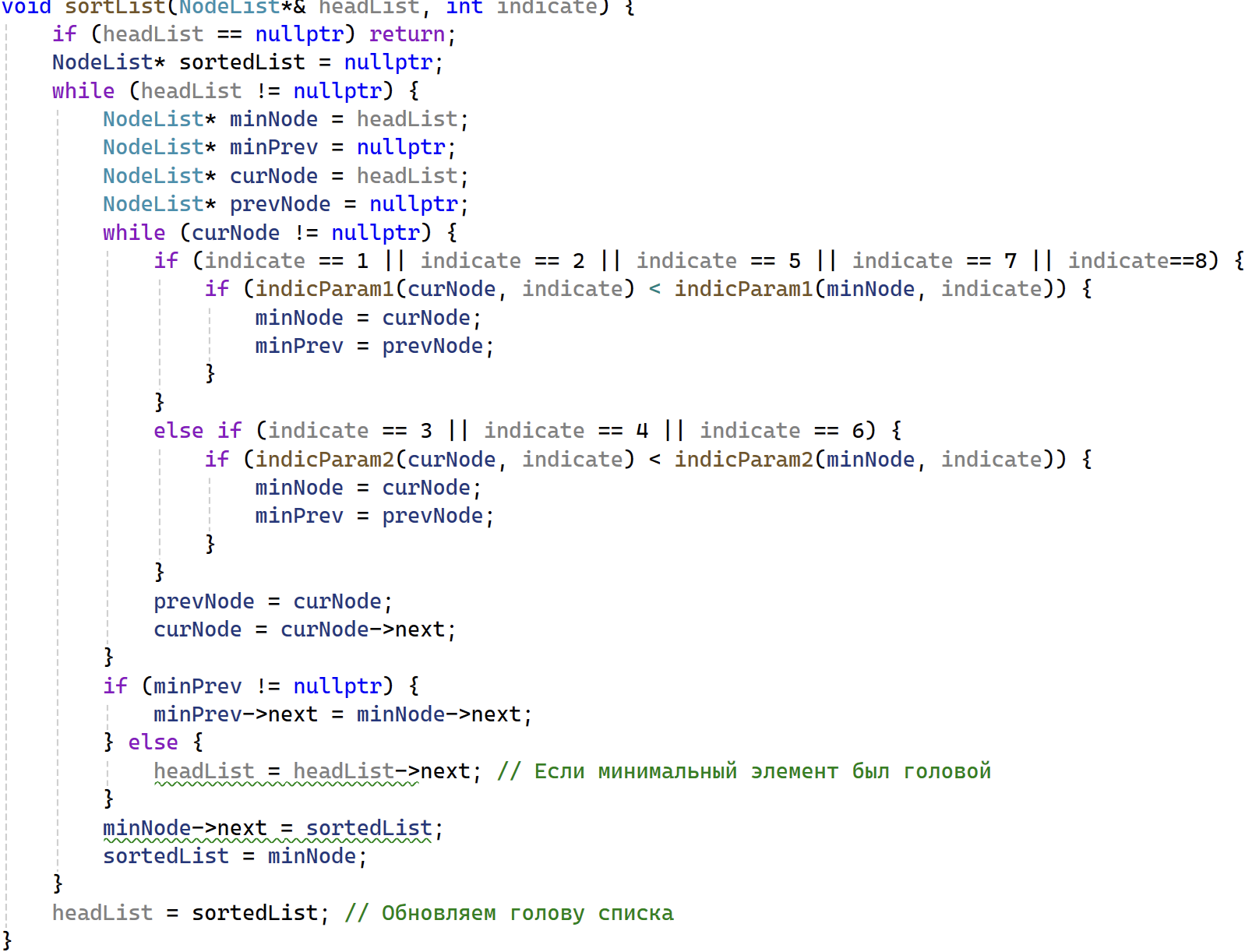


Рисунок 13 - Код функции sortList

### Функция изменения очереди

Назначение: изменяет очередь

Прототип функции:

void editQueue(NodeList\*\* headQueue)

Код функции:

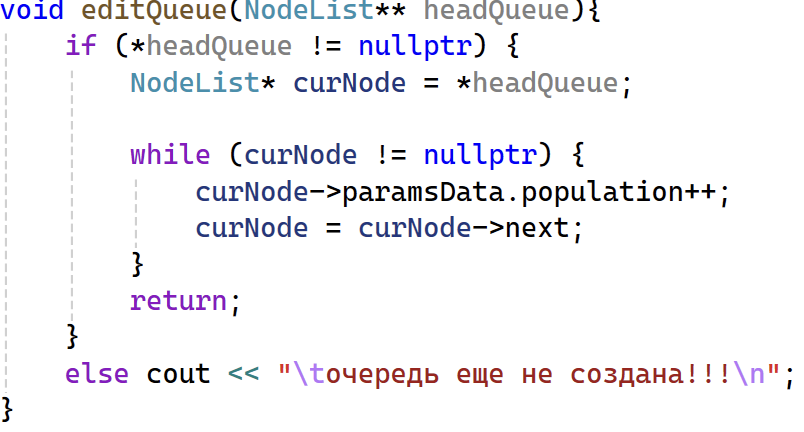


Рисунок 14 - Код функции editQueue

### Функция обновления базы данных

Назначение: обновляет базу данных на основе изменений в очереди

Прототип функции:

void updateBase(NodeList\* headQueue, NodeList\* headList)

Код функции:

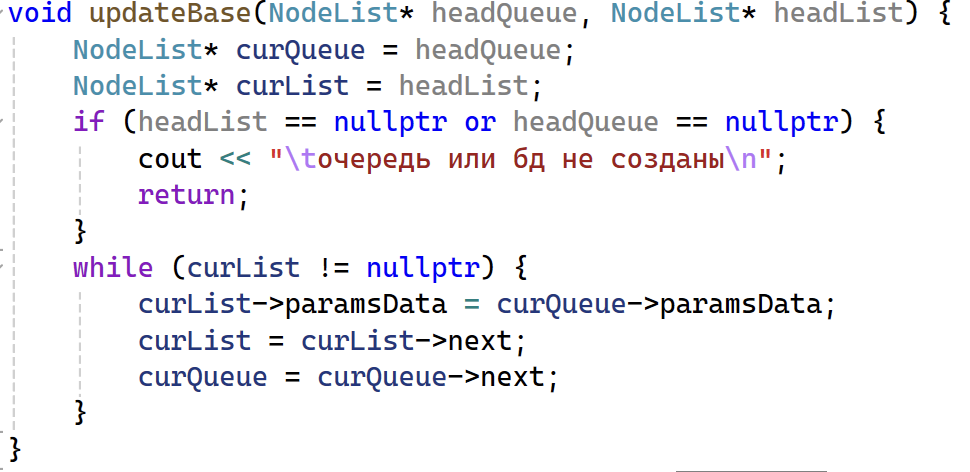


Рисунок 15 - Код функции updateBase

### Функция создания элемента очереди

Назначение: создает очередь с приоритетом

Прототип функции:

NodeList\* createQueue(NodeList\* headList)

Код функции:

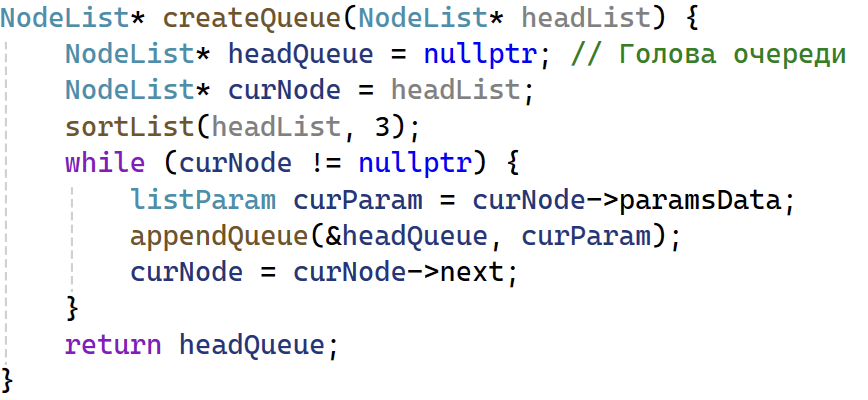


Рисунок 16 - Код функции createQueue

### 

### Основная функция

Назначение: запускает другие функции (в том числе функцию menu)

Прототип функции:

int main()

Код функции:

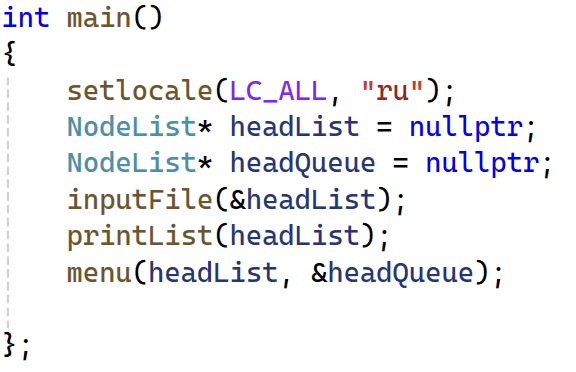


Рисунок 17 - Код функции main

# Результаты работы

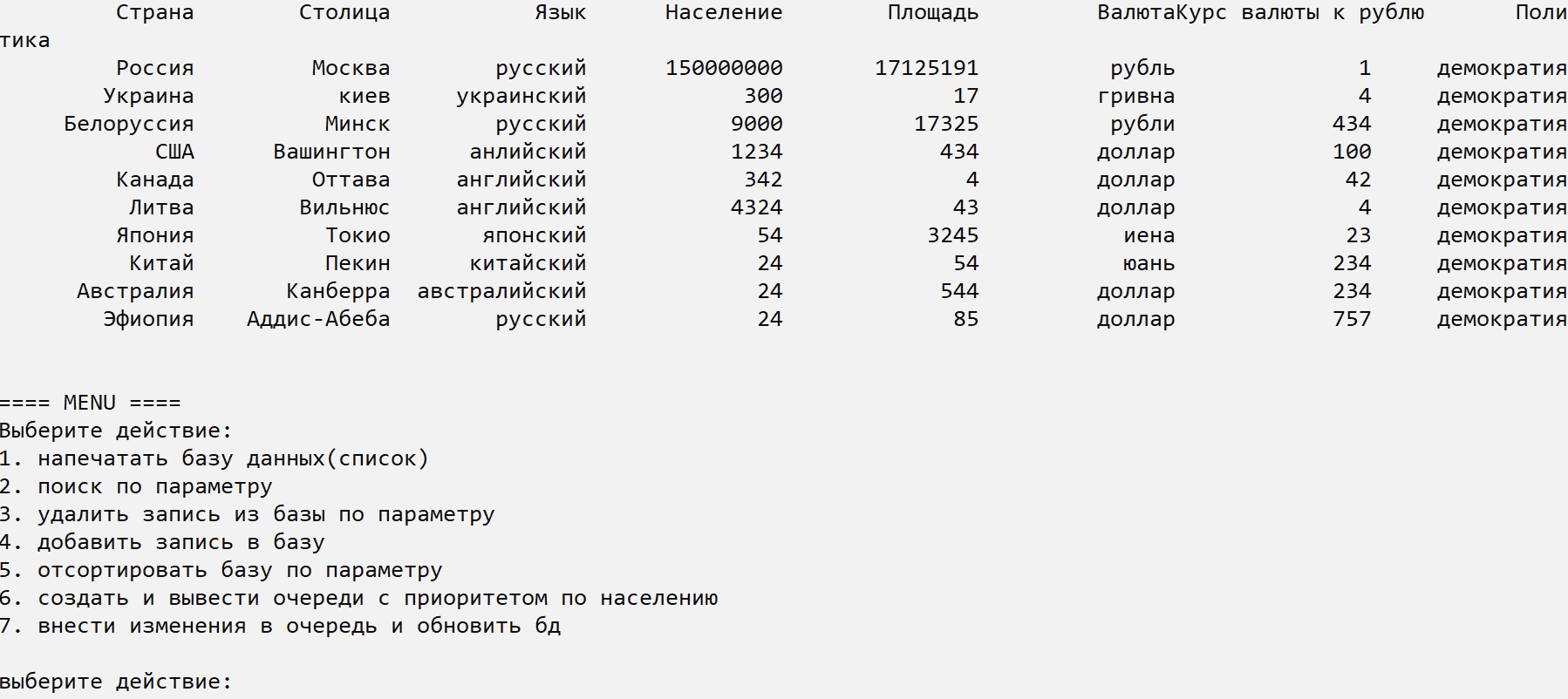


Рисунок 18 – Начальное меню

## Корректные тесты

1) напечатать базу данных(список)



Рисунок 19 – Вывод базы данных

2) поиск по параметру

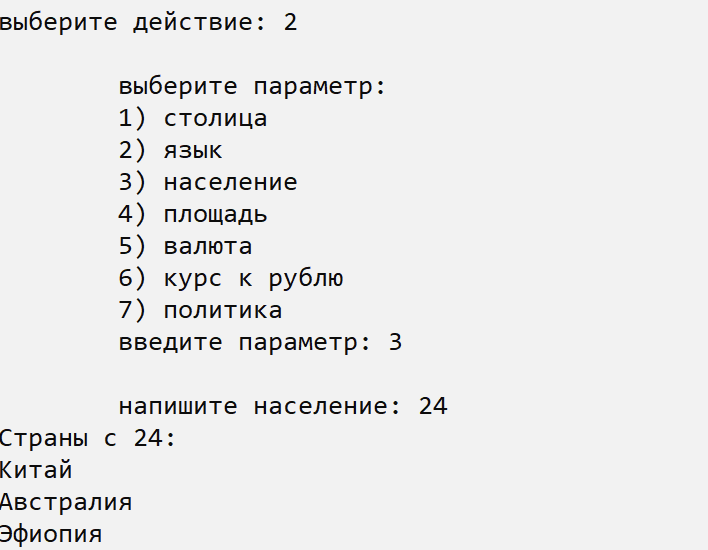


Рисунок 20 – Поиск стран по параметру

3) удаление записей по параметру

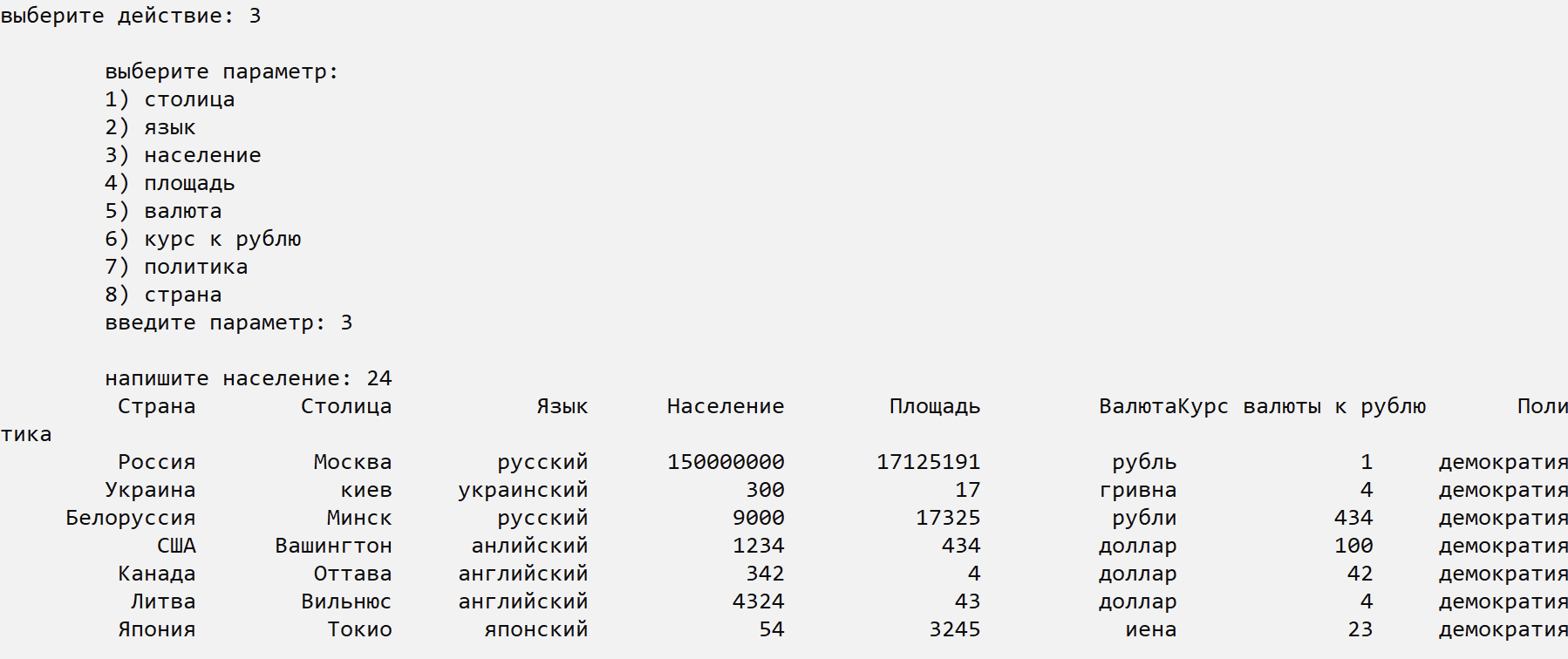


Рисунок 21 – Удаление по параметру

4) добавление записи с клавиатуры

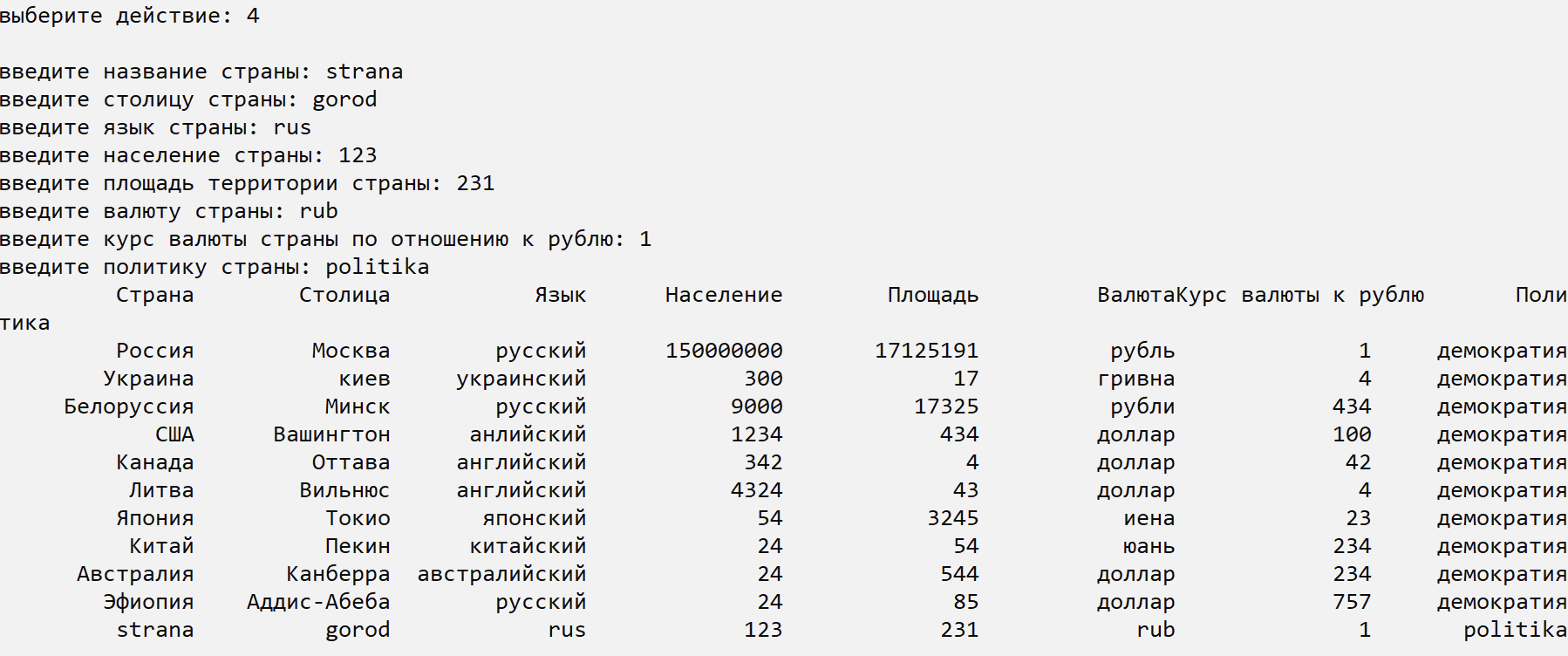


Рисунок 21 – Добавление новой записи с клавиатуры

5) сортировка по параметру

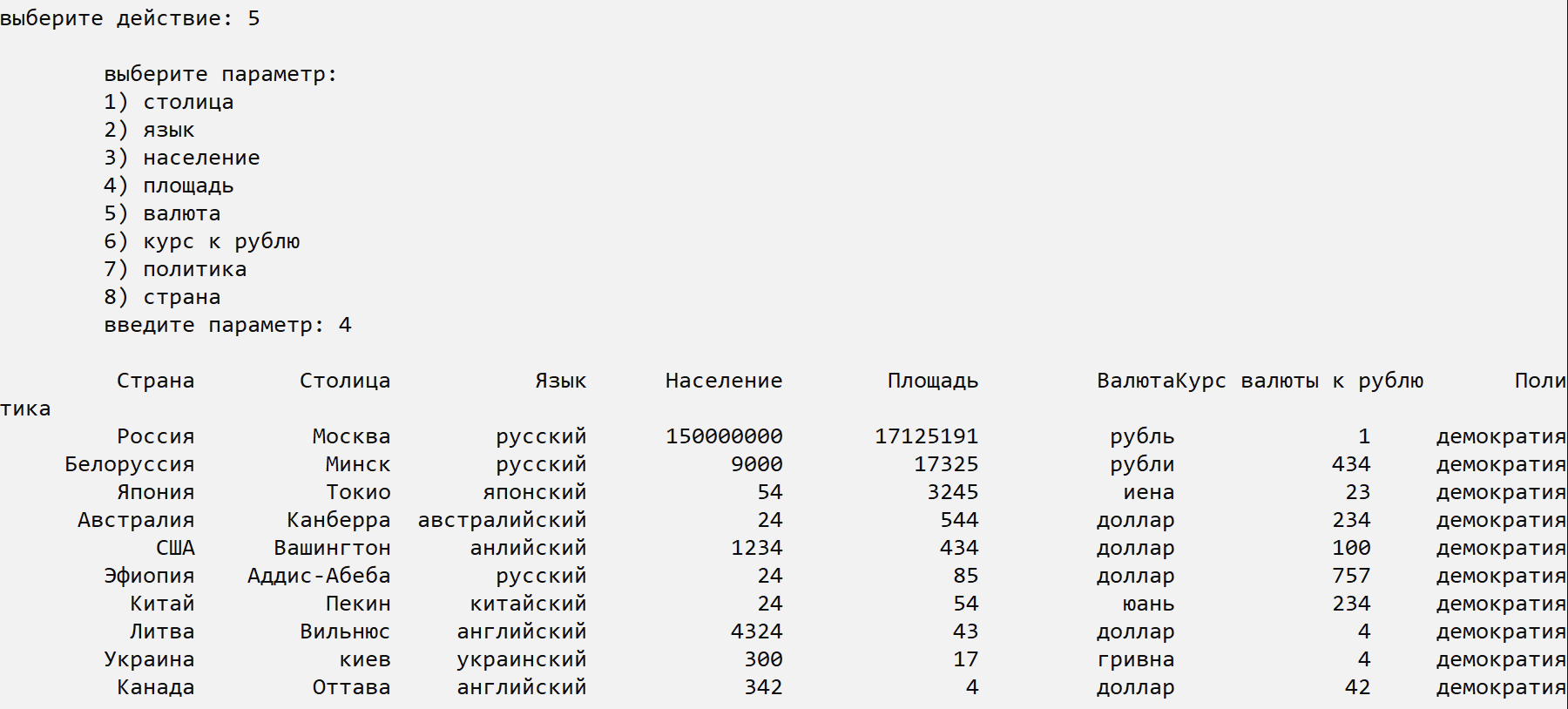


Рисунок 22 – Сортировка по параметру

6) создание очереди по приоритету населения



Рисунок 23 – Создание очередь по приоритету населения

7) обновление базы данных



Рисунок 24 – Обновление базы данных

## Некорректные тесты

1) Некорректный выбор пункта меню

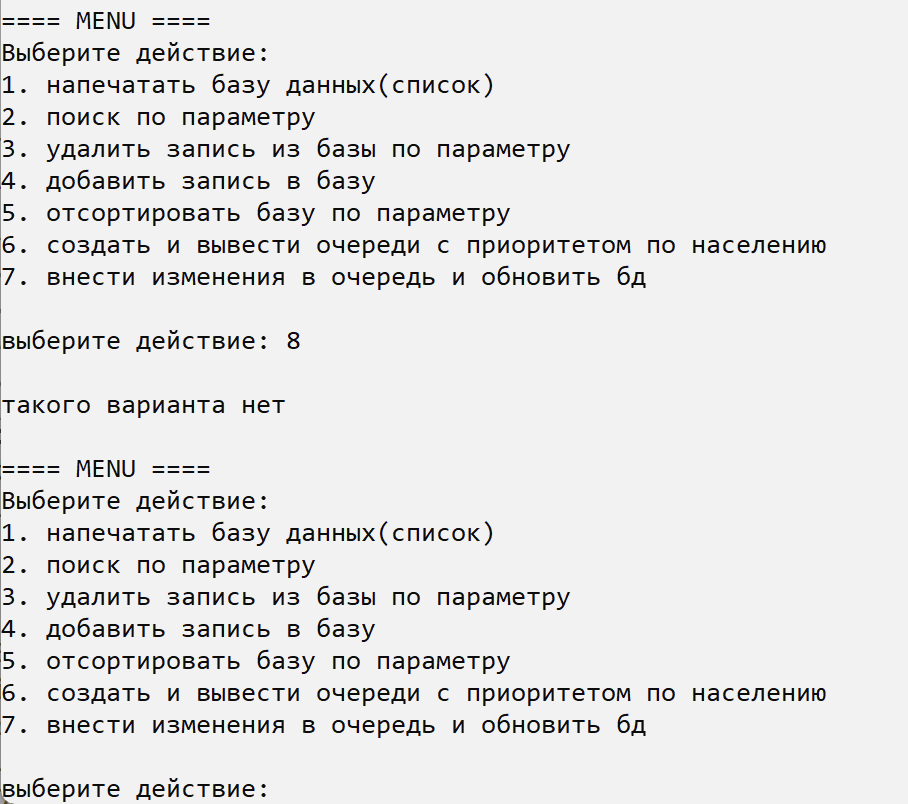


Рисунок 25 – Некорректный выбор действия

2) некорректный выбор параметра для удаления

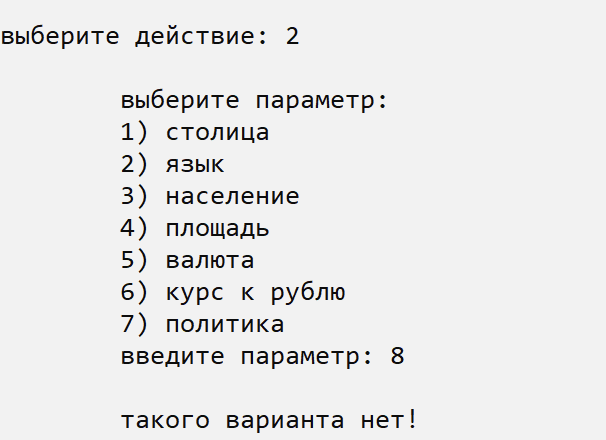


Рисунок 26 – Некорректный выбор параметра

3) некорректный выбор параметра при удалении

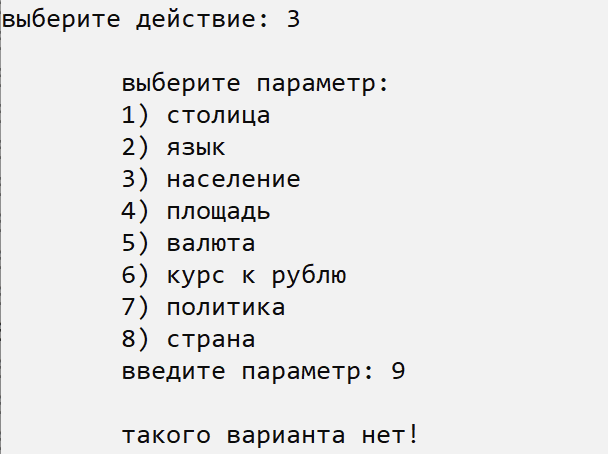


Рисунок 27 - Некорректный выбор параметра при удалении

# Заключение

В ходе разработки были реализованы динамические структуры данных, такие как линейные односвязные списки и очереди с приоритетом, что позволило обеспечить гибкость и эффективность при работе с данными. Программа поддерживает функции ввода, вывода, сортировки и удаления данных, а также выполнение запросов по любым полям информации. Также была реализована система меню, которая позволяет пользователю взаимодействовать с программой и управлять данными без необходимости глубоких знаний в области программирования.

Одной из ключевых задач было обеспечение гибкости работы с данными, что удалось достичь благодаря использованию динамических структур и реализации функций для работы с ними. Все поставленные задачи были успешно выполнены, и результат работы соответствует заявленным требованиям.

# Приложение А. Код программы

|  |
| --- |
| #include <iostream> // Для работы с вводом и выводом  #include <string> // Для использования строк  #include <fstream> // Для работы с файлами  #include <stdlib.h> // Для функции atoi (преобразования строки в число)  #include <iomanip> // Для форматированного вывода  **using** **namespace** std;  // Глобальное имя файла, из которого читается база данных  string fileName = "countries.txt";  // Структура для хранения параметров страны  **struct** listParam {  string country; // Название страны  string capital; // Столица страны  string language; // Язык страны  **int** population; // Население страны  **int** square; // Площадь территории  string currency; // Валюта страны  **int** currencyRateRuble; // Курс валюты относительно рубля  string politics; // Политическая система  };  // Структура узла односвязного списка  **struct** NodeList {  NodeList\* next; // Указатель на следующий узел  listParam paramsData; // Параметры страны  };  // Прототипы функций  string indicParam1(NodeList\* curNodeList, **int** indicate);  **int** indicParam2(NodeList\* curNodeList, **int** indicate);  **void** sortList(NodeList\*& headList, **int** indicate);  **void** appendList(NodeList\*\* headList, listParam newData);  **void** inputFile(NodeList\*\* headList);  NodeList\* createNodeList(listParam newData);  **void** printList(NodeList\* headList);  **void** menu(NodeList\* headList, NodeList\*\* heagQueue);  **void** appendBase(NodeList\*\* headList);  **void** deleteNodeListParam(NodeList\*\* headList, string param, **int** indicate);  **void** deleteNodeListParam(NodeList\*\* headList, **int** param, **int** indicate);  **void** searchParam(NodeList\* headList, string param, **int** indicate);  **void** searchParam(NodeList\* headList, **int** param, **int** indicate);  NodeList\* createQueue(NodeList\* headList);  **void** appendQueue(NodeList\*\* headQueue, listParam newData);  **void** editQueue(NodeList\*\* headQueue);  **void** updateBase(NodeList\* headQueue, NodeList\* headList);  // Функция для получения строкового параметра по индексу  string indicParam1(NodeList\* curNodeList, **int** indicate) {  **switch** (indicate) {  **case** 1: **return** curNodeList->paramsData.capital; // Столица  **case** 2: **return** curNodeList->paramsData.language; // Язык  **case** 5: **return** curNodeList->paramsData.currency; // Валюта  **case** 7: **return** curNodeList->paramsData.politics; // Политическая система  **case** 8: **return** curNodeList->paramsData.country; // Страна  **default**: **return** ""; // В случае неверного индекса возвращаем пустую строку  }  }  // Функция для получения числового параметра по индексу  **int** indicParam2(NodeList\* curNodeList, **int** indicate) {  **switch** (indicate) {  **case** 3: **return** curNodeList->paramsData.population; // Население  **case** 4: **return** curNodeList->paramsData.square; // Площадь  **case** 6: **return** curNodeList->paramsData.currencyRateRuble; // Курс валюты  **default**: **return** 0; // В случае неверного индекса возвращаем 0  }  }  // Функция для поиска по строковому параметру  **void** searchParam(NodeList\* headList, string param, **int** indicate) {  **if** (indicate == 1 || indicate == 2 || indicate == 5 || indicate == 7) {  NodeList\* curNodeList = headList;  cout << "Страны с " << param << ":" << endl;  **while** (curNodeList != **nullptr**) {  **if** (param == indicParam1(curNodeList, indicate)) {  cout << curNodeList->paramsData.country << '\n';  }  curNodeList = curNodeList->next; // Переход к следующему узлу  }  } **else** {  cout << "неверный запрос";  }  }  // Функция для поиска по числовому параметру  **void** searchParam(NodeList\* headList, **int** param, **int** indicate) {  **if** (indicate == 3 || indicate == 4 || indicate == 6) {  NodeList\* curNodeList = headList;  cout << "Страны с " << param << ":" << endl;  **while** (curNodeList != **nullptr**) {  **if** (param == indicParam2(curNodeList, indicate)) {  cout << curNodeList->paramsData.country << '\n';  }  curNodeList = curNodeList->next; // Переход к следующему узлу  }  } **else** {  cout << "неверный запрос";  }  }  // Главное меню программы  **void** menu(NodeList\* headList, NodeList\*\* headQueue)  {  cout << "\n==== MENU ====\nВыберите действие:\n" <<  "1. напечатать базу данных(список)\n" <<  "2. поиск по параметру\n" <<  "3. удалить запись из базы по параметру\n"<<  "4. добавить запись в базу\n"<<  "5. отсортировать базу по параметру\n"<<  "6. создать и вывести очереди с приоритетом по населению\n"<<  "7. внести изменения в очередь и обновить бд\n";  **int** inp;  cout << "\nвыберите действие: ";  cin >> inp;  cout << '\n';  **switch** (inp)  {  **case** 1:  {  printList(headList);  **return** menu(headList, headQueue);  }  **case** 2: {  cout << "\tвыберите параметр:\n" <<  "\t1) столица\n" <<  "\t2) язык\n" <<  "\t3) население\n" <<  "\t4) площадь\n" <<  "\t5) валюта\n" <<  "\t6) курс к рублю\n" <<  "\t7) политика\n";  **int** inp2;  cout << "\tвведите параметр: ";  cin >> inp2;  cout << endl;  **switch** (inp2) {  **case** 1: {  cout << "\tнапишите название столицы:\n";  string str1;  cin >> str1;  searchParam(headList, str1, 1);  **break**;  }  **case** 2: {  cout << "\tнапишите язык: ";  string str2;  cin >> str2;  searchParam(headList, str2, 2);  **break**;  }  **case** 3: {  cout << "\tнапишите население: ";  **int** str3;  cin >> str3;  searchParam(headList, str3, 3);  **break**;  }  **case** 4: {  cout << "\tнапишите площадь: ";  **int** str4;  cin >> str4;  searchParam(headList, str4, 4);  **break**;  }  **case** 5: {  cout << "\tнапишите валюту: ";  string str5;  cin >> str5;  searchParam(headList, str5, 5);  **break**;  }  **case** 6: {  cout << "\tнапишите курс в рублях: ";  **int** str6;  cin >> str6;  searchParam(headList, str6, 6);  **break**;  }  **case** 7: {  cout << "\tнапишите вид политики: ";  string str7;  cin >> str7;  searchParam(headList, str7, 7);  **break**;  }  **default**: {  cout << "\tтакого варианта нет!";  }  }  printList(headList);  **return** menu(headList, headQueue);  }  **case** 3: {  cout << "\tвыберите параметр:\n" <<  "\t1) столица\n" <<  "\t2) язык\n" <<  "\t3) население\n" <<  "\t4) площадь\n" <<  "\t5) валюта\n" <<  "\t6) курс к рублю\n" <<  "\t7) политика\n"<<  "\t8) страна\n";  **int** inp2;  cout << "\tвведите параметр: ";  cin >> inp2;  cout << endl;  **switch** (inp2) {  **case** 1: {  cout << "\tнапишите название столицы:\n";  string str1;  cin >> str1;  deleteNodeListParam(&headList, str1, 1);  **break**;  }  **case** 2: {  cout << "\tнапишите язык: ";  string str2;  cin >> str2;  deleteNodeListParam(&headList, str2, 2);  **break**;  }  **case** 3: {  cout << "\tнапишите население: ";  **int** str3;  cin >> str3;  deleteNodeListParam(&headList, str3, 3);  **break**;  }  **case** 4: {  cout << "\tнапишите площадь: ";  **int** str4;  cin >> str4;  deleteNodeListParam(&headList, str4, 4);  **break**;  }  **case** 5: {  cout << "\tнапишите валюту: ";  string str5;  cin >> str5;  deleteNodeListParam(&headList, str5, 5);  **break**;  }  **case** 6: {  cout << "\tнапишите курс в рублях: ";  **int** str6;  cin >> str6;  deleteNodeListParam(&headList, str6, 6);  **break**;  }  **case** 7: {  cout << "\tнапишите вид политики: ";  string str7;  cin >> str7;  deleteNodeListParam(&headList, str7, 7);  **break**;  }  **case** 8: {  cout << "\tнапишите страну: ";  string str8;  cin >> str8;  deleteNodeListParam(&headList, str8, 8);  **break**;  }  **default**: {  cout << "\tтакого варианта нет!";  **break**;  }  }  printList(headList);  **return** menu(headList, headQueue);  }  **case** 4: {  appendBase(&headList);  **return** menu(headList, headQueue);  }  **case** 5: {  cout << "\tвыберите параметр:\n" <<  "\t1) столица\n" <<  "\t2) язык\n" <<  "\t3) население\n" <<  "\t4) площадь\n" <<  "\t5) валюта\n" <<  "\t6) курс к рублю\n" <<  "\t7) политика\n" <<  "\t8) страна\n";  **int** inp2;  cout << "\tвведите параметр: ";  cin >> inp2;  cout << endl;  sortList(headList, inp2);  printList(headList);  **return** menu(headList, headQueue);  }  **case** 6: {  \*headQueue = createQueue(headList);  cout << "созданная очередь:\n";  printList(\*headQueue);  **return** menu(headList ,headQueue);  }  **case** 7: {  editQueue(headQueue);  updateBase(\*headQueue, headList);  cout << "измененная очередь:\n";  printList(\*headQueue);  cout << "измененная бд:\n";  printList(headList);  **return** menu(headList, headQueue);  }  }  }  // Добавление новой записи в базу данных  **void** appendBase(NodeList\*\* headList) {  string country, capital, language, currency, politics;  **int** population, square, currencyRateRuble;  // Получаем данные от пользователя  cout << "введите название страны: "; cin >> country;  cout << "введите столицу страны: "; cin >> capital;  cout << "введите язык страны: "; cin >> language;  cout << "введите население страны: "; cin >> population;  cout << "введите площадь страны: "; cin >> square;  cout << "введите валюту страны: "; cin >> currency;  cout << "введите курс валюты: "; cin >> currencyRateRuble;  cout << "введите политическую систему: "; cin >> politics;  // Создаём новый узел и добавляем его в список  listParam newNode = { country, capital, language, population, square, currency, currencyRateRuble, politics };  appendList(headList, newNode);  printList(\*headList); // Печатаем обновлённую базу данных  }  // Чтение данных из файла и заполнение списка  **void** inputFile(NodeList\*\* headList) {  ifstream fin; // Создаем объект для чтения файла  fin.open(fileName); // Открываем файл  string country, capital, language, population, square, currency, currencyRateRuble, politics;  **int** pop, sq, cuRate;  // Читаем файл построчно до конца  **while** (!fin.eof()) {  getline(fin, country);  getline(fin, capital);  getline(fin, language);  getline(fin, population);  getline(fin, square);  getline(fin, currency);  getline(fin, currencyRateRuble);  getline(fin, politics);  pop = atoi(population.c\_str()); // Преобразуем строки в числа  sq = atoi(square.c\_str());  cuRate = atoi(currencyRateRuble.c\_str());  // Создаем новый узел и добавляем его в список  listParam curCountry = { country, capital, language, pop, sq, currency, cuRate, politics };  appendList(headList, curCountry);  }  fin.close(); // Закрываем файл  }  // Функция для создания нового узла списка  NodeList\* createNodeList(listParam newData) {  NodeList\* newNode = **new** NodeList; // Выделяем память для нового узла  newNode->paramsData = newData; // Заполняем параметры нового узла  newNode->next = **nullptr**; // Изначально следующий элемент отсутствует  **return** newNode; // Возвращаем указатель на созданный узел  }  // Функция для добавления узла в конец списка  **void** appendList(NodeList\*\* headList, listParam newData) {  NodeList\* newNode = createNodeList(newData); // Создаем новый узел  **if** (\*headList == **nullptr**) { // Если список пуст  \*headList = newNode; // Новый узел становится головой списка  } **else** { // Иначе добавляем в конец  NodeList\* temp = \*headList; // Временный указатель для прохода по списку  **while** (temp->next != **nullptr**) { // Переходим к последнему узлу  temp = temp->next;  }  temp->next = newNode; // Присоединяем новый узел к последнему  }  }  // Функция для печати всего списка  **void** printList(NodeList\* headList) {  NodeList\* current = headList; // Начинаем с головы списка  **while** (current != **nullptr**) { // Проходим по всем узлам  cout << "Страна: " << current->paramsData.country << '\n';  cout << "Столица: " << current->paramsData.capital << '\n';  cout << "Язык: " << current->paramsData.language << '\n';  cout << "Население: " << current->paramsData.population << '\n';  cout << "Площадь: " << current->paramsData.square << '\n';  cout << "Валюта: " << current->paramsData.currency << '\n';  cout << "Курс валюты: " << current->paramsData.currencyRateRuble << '\n';  cout << "Политическая система: " << current->paramsData.politics << '\n';  cout << "--------------------------\n";  current = current->next; // Переходим к следующему узлу  }  }  // Функция для удаления узла по строковому параметру  **void** deleteNodeListParam(NodeList\*\* headList, string param, **int** indicate) {  NodeList\* temp = \*headList; // Временный указатель для текущего узла  NodeList\* prev = **nullptr**; // Указатель на предыдущий узел  // Проходим по списку  **while** (temp != **nullptr**) {  **if** (indicParam1(temp, indicate) == param) { // Если параметр совпадает  **if** (prev == **nullptr**) { // Если удаляем голову списка  \*headList = temp->next; // Головой становится следующий элемент  } **else** { // Иначе удаляем текущий узел  prev->next = temp->next; // Пропускаем текущий узел  }  **delete** temp; // Освобождаем память  **return**; // Завершаем функцию  }  prev = temp; // Обновляем предыдущий узел  temp = temp->next; // Переходим к следующему узлу  }  }  // Функция для удаления узла по числовому параметру  **void** deleteNodeListParam(NodeList\*\* headList, **int** param, **int** indicate) {  NodeList\* temp = \*headList; // Временный указатель для текущего узла  NodeList\* prev = **nullptr**; // Указатель на предыдущий узел  **while** (temp != **nullptr**) { // Проходим по списку  **if** (indicParam2(temp, indicate) == param) { // Если параметр совпадает  **if** (prev == **nullptr**) { // Если удаляем голову списка  \*headList = temp->next; // Обновляем голову списка  } **else** { // Удаляем текущий узел  prev->next = temp->next; // Пропускаем текущий узел  }  **delete** temp; // Освобождаем память  **return**; // Завершаем функцию  }  prev = temp; // Обновляем предыдущий узел  temp = temp->next; // Переходим к следующему узлу  }  }  // Функция для создания очереди из списка  NodeList\* createQueue(NodeList\* headList) {  NodeList\* headQueue = **nullptr**; // Изначально очередь пуста  NodeList\* current = headList; // Указатель на текущий узел списка  **while** (current != **nullptr**) { // Проходим по списку  appendQueue(&headQueue, current->paramsData); // Добавляем узел в очередь  current = current->next; // Переходим к следующему узлу  }  **return** headQueue; // Возвращаем голову очереди  }  // Функция для добавления узла в очередь  **void** appendQueue(NodeList\*\* headQueue, listParam newData) {  appendList(headQueue, newData); // Используем функцию appendList для добавления узла  }  // Функция для редактирования данных в очереди  **void** editQueue(NodeList\*\* headQueue) {  NodeList\* current = \*headQueue; // Указатель на голову очереди  **while** (current != **nullptr**) { // Проходим по всем элементам очереди  cout << "Редактирование страны: " << current->paramsData.country << '\n';  cout << "Введите новое население: ";  cin >> current->paramsData.population; // Обновляем население страны  current = current->next; // Переходим к следующему элементу  }  }  // Функция для обновления данных базы из очереди  **void** updateBase(NodeList\* headQueue, NodeList\* headList) {  NodeList\* queueNode = headQueue; // Указатель на начало очереди  **while** (queueNode != **nullptr**) { // Проходим по очереди  NodeList\* listNode = headList; // Начинаем с начала списка  **while** (listNode != **nullptr**) {  // Находим соответствие по названию страны и обновляем данные  **if** (listNode->paramsData.country == queueNode->paramsData.country) {  listNode->paramsData.population = queueNode->paramsData.population;  **break**;  }  listNode = listNode->next; // Переходим к следующему узлу списка  }  queueNode = queueNode->next; // Переходим к следующему узлу очереди  }  cout << "База данных обновлена!\n";  }  **int** main()  {  setlocale(LC\_ALL, "ru");  NodeList\* headList = **nullptr**;  NodeList\* headQueue = **nullptr**;  inputFile(&headList);  printList(headList);  menu(headList, &headQueue);    }; |