Министерство образования Российской Федерации
Пензенский государственный университет
Кафедра «Вычислительная техника»

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовой работе

по курсу «Программирование»

на тему «Разработка программы сложной структуры методом нисходящего программирования. База данных по столицам стран»

Выполнил:

ОТ.Л. 13.05.26 студент группы 25ВВВ4

Качкуркин Р. Н.

Принял:

к.т.н., Токарев А. Н.

Пенза 2026

ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
Факультет Вычислительной техники

Кафедра "Вычислительная техника"

"УТВЕРЖДАЮ"

Зав. кафедрой ВТ

_____ М.А. Митрохин

«____»_____20__г

ЗАДАНИЕ

на курсовое проектирование по курсу

Программирование

Студенту Калмыкину Роману Николаевичу Группа 25ВВВ4

Тема проекта: ***Разработка программы сложной структуры методом нисходящего программирования.***

Исходные данные (технические требования) на проектирование

База данных по столицам стран. Технические требования:
- Добавление записей
- Удаление записей
- Редактирование записей
- Поиск и сортировка записей
- Фильтрация записей по определённому критерию
- Создание новой базы данных
- Сохранение текущей базы данных
- Загрузка сохранённых баз данных

Обязательные требования к программе:

1. Многомодульность.
2. Использование сложных типов данных – структур, списков, файлов.
3. Режим работы видеосистемы – текстовый / графический.
4. Устройство ввода информации – клавиатура, мышь.
5. Пользовательский интерфейс должен быть построен на основе меню и панели инструментов.
6. Наличие заставки.
7. Операционная система MS Windows.
8. Язык программирования – Си и Ассемблер.
9. Среда разработки ПО – Microsoft Visual Studio.

Объем работы по курсу

1. **Расчетная часть**

Разработка программы.

2. **Графическая часть**

Схема данных, схема ресурсов системы, схема работы системы, иерархическая структура программы, схема взаимодействия программы.

3. **Экспериментальная часть**

Отладка программы.

Срок выполнения проекта по разделам

В соответствии с графиком.

Дата выдачи задания "18" февраля 2026

Дата защиты проекта "____" _____

Руководитель _____

Задание получил "25" февраля 2026 г.

Студент Канкулин Канкулин Роман Николаевич

Содержание

Введение 5

Постановка задачи 7

Выбор решения 11

Описание разработки программы 16

Отладка и тестирование 18

Описание программы 21

Руководство пользователя 35

Заключение 41

Список используемых источников 43

Введение

На сегодняшний день информация является одним из наиболее значимых ресурсов, систематизация и обработка которого позволяют оптимизировать деятельность как отдельных специалистов, так и общества в целом. Компьютерные технологии во многом упрощают нашу жизнь в самых разнообразных сферах деятельности. Именно они позволяют грамотно взаимодействовать с информационными потоками, эффективно обрабатывать их, структурировать и надежно хранить на компактных носителях.

База данных представляет собой упорядоченную, структурированную совокупность данных, подобранных таким образом, чтобы эти данные могли легко и быстро обрабатываться на ЭВМ. Эволюция систем управления базами данных (СУБД) берет свое начало с середины 1950-х годов, развившись от простейших файловых моделей на перфокартах до распределенных высокопроизводительных систем. В наше время базы данных являются неотъемлемой частью любого предприятия, коммерческой организации или государственного института. Они активно применяются в работе банковских сетей, поисковых систем, динамических веб-сайтов и аналитических платформ.

Предметная область данной курсовой работы связана с накоплением, систематизацией и обработкой географических, демографических и политических сведений о государствах мира. Создание специализированной информационной системы для учета стран, их столиц, численности населения и континентальной принадлежности является практически важной задачей. Систематизация подобных данных востребована в справочно-энциклопедических сервисах, образовательных программах, аналитических ведомствах и международной логистике. Разработанная программа представляет собой локальную информационно-поисковую систему. Она автоматизирует учет записей, предоставляет пользователю удобные инструменты навигации и избавляет от ручной обработки больших объемов каталогизированной информации.

Результаты данной курсовой работы могут быть непосредственно использованы в качестве автономного информационного модуля, компонента электронных учебных пособий по географии, а также как концептуальная база для проектирования более масштабных информационных систем. Наработанные алгоритмические решения и модульная архитектура представляют высокую ценность при дальнейшем изучении дисциплин, связанных с системным программированием и архитектурой вычислительных систем.

Постановка задачи

Целью данной курсовой работы является разработка многомодульного программного комплекса на языке Си для автоматизации процессов учета, хранения, корректировки и поиска информации в предметной области «Столицы стран мира». Программа представляет собой локальную систему управления базой данных (СУБД), оперирующую сведениями о государствах, их административных центрах, численности населения и географическом положении.

Для достижения поставленной цели необходимо детально проработать и реализовать ряд обязательных технических и архитектурных требований. Ниже приведено развернутое описание каждого требования с позиции логики его программного воплощения и практической целесообразности.

1. Многомодульность структуры программы.

Разделение всего исходного кода программного комплекса на логически обособленные составные части — заголовочные файлы (с расширением .h) и файлы реализации (с расширением .cpp). В рамках проекта выделяются: модуль описания структур данных (record.h), модуль управления базой данных (database.h, database.cpp), модуль вспомогательных интерфейсных функций (minor_functions.h) и главный управляющий модуль (main.cpp / код курсовой.cpp).

Многомодульный подход избавляет от избыточности кода и предотвращает создание нечитаемых «монолитных» файлов. Он позволяет изолированно тестировать и отлаживать отдельные функции (например, алгоритм сортировки или ассемблерную вставку сравнения строк), не затрагивая логику работы интерфейса. Кроме того, это обеспечивает высокую повторяемость кода и упрощает последующее расширение программного продукта.

2. Использование сложных типов данных (структур, динамических списков, файлов).

Структуры: Объединение разнородных характеристик сущности «Страна» в единый пользовательский тип данных Record. Поля структуры включают:

название страны (char[50]), название столицы (char[50]), численность населения (double) и континент (char[30]).

Динамические списки: Организация хранения записей в оперативной памяти в виде линейного односвязного динамического списка, где каждый элемент (Node) содержит полезную нагрузку (Record data) и указатель на следующий элемент (Node* next).

Файлы: Реализация механизмов сериализации и десериализации данных для долговременного хранения информации на жестком диске в текстовом формате.

Использование структур позволяет абстрагироваться от отдельных массивов под каждое свойство и оперировать объектом целиком. Динамический список необходим для преодоления ограничений статических массивов: он обеспечивает выделение памяти строго под текущее количество записей в реальном времени, позволяя легко добавлять и удалять элементы без сдвига последующих ячеек в памяти. Файловая система необходима для обеспечения персистентности данных, чтобы результаты работы пользователя сохранялись между перезапусками программы.

3. Режим работы видеосистемы — текстовый.

Программа функционирует в текстовом (консольном) режиме отображения информации. Для управления выводом, позиционированием курсора и изменением геометрии графического окна консоли (задание фиксированного размера в 100 столбцов и 30 строк) задействуются низкоуровневые управляющие последовательности терминала VT100 , активируемые через WinAPI функции.

Текстовый режим является наиболее эффективным, легковесным и быстродействующим для утилитарных информационно-поисковых систем. Отказ от тяжелых графических библиотек снижает требования к аппаратным ресурсам ЭВМ, а использование стандартизированных VT100-последовательностей позволяет добиться четкого табличного форматирования данных и контролировать поведение интерфейса при запуске на различных конфигурациях операционной системы.

4. Устройство ввода информации — клавиатура.

Взаимодействие пользователя с системой осуществляется посредством стандартного потока ввода (stdin). Ввод текстовых данных (названий стран, столиц) и численных значений (население) реализуется с помощью защищенных функций (таких как fgets), исключающих переполнение буфера, с последующей программной очисткой строк от символа переноса каретки (\n). Опрос клавиатуры также используется для навигации по функциональным зонам приложения.

Клавиатурный ввод является базовым и наиболее надежным способом структурированного наполнения текстовых баз данных. Он гарантирует высокую скорость ввода оператором текстовых массивов и точное соответствие передаваемых команд выбранным пунктам меню, минимизируя случайные ошибки позиционирования.

5. Построение пользовательского интерфейса на основе меню.

Разработка интерактивного текстового меню, работающего в циклическом режиме. Интерфейс выводит на экран перечень доступных административных и аналитических операций (инструментов), жестко привязанных к числовым индексам (от 1 до 10), и ожидает выбора пользователя. Реализуется строгое разграничение экранов: вывод таблицы, форма добавления, окно фильтрации по континентам и т. д.

Меню-ориентированный интерфейс обеспечивает интуитивно понятное и предсказуемое управление программой. Он защищает целостность данных, блокируя несанкционированные действия и предотвращая переход приложения в неопределенное состояние. Пользователь всегда видит полный спектр возможностей системы (панель доступных функций) и может прозрачно управлять жизненным циклом базы данных.

6. Наличие заставки (стартового экрана).

Интеграция в процесс инициализации программы специализированной заставки (всплывающего загрузочного экрана). Заставка выводит визуальный индикатор выполнения, имитирующий или отражающий реальный процесс

чтения конфигурационных файлов и развертывания динамических структур в памяти при старте СУБД.

Зачем это необходимо: Наличие заставки выполняет важную эргономическую функцию: оно сигнализирует пользователю о корректном запуске исполняемого файла, визуализирует этап подготовки программной среды к работе и улучшает общее восприятие программного продукта, приближая его к стандартам коммерческого ПО.

Выбор решения

Для успешной реализации всех требований, предъявленных к курсовой работе, проектирование программного комплекса строится на принципах нисходящего проектирования. Этот подход подразумевает движение от общего к частному: сначала определяется общая структура приложения и управляющий контур, после чего детально прорабатываются низкоуровневые модули, структуры данных и алгоритмы.

Механизм управления программой и архитектура основного цикла

В рамках разработки данной информационной системы выбран циклический императивный механизм управления на основе текстового меню (меню-ориентированное управление) с синхронным опросом состояния. В отличие от асинхронного событийно-ориентированного управления, применяемого в оконных графических интерфейсах, данный подход является оптимальным для консольных утилит и локальных СУБД. Он обеспечивает строгую детерминированность выполнения, исключает возникновение состояний гонки при одновременном обращении к ресурсам памяти и гарантирует предсказуемое поведение программы при последовательном выполнении операций.

Архитектура управляющего контура берет свое начало в главной функции main() и спускается ниже к функциональным модулям. Основной цикл программы функционирует по следующему алгоритму:

1. Этап инициализации: при запуске программы вызывается низкоуровневая функция настройки видеосистемы setupConsole(), которая переводит стандартный терминал операционной системы в режим обработки управляющих последовательностей VT100. Затем управление передается модулю отрисовки заставки showLoadingScreen(), который визуализирует процесс подготовки среды.

2. Предварительная загрузка: система осуществляет автоматическое чтение данных из текущего файла базы данных (по умолчанию db.txt) в оперативную память, формируя первичный динамический список.

3. Глава управляющего цикла: программа входит в детерминированный цикл while (choice != 0). На каждой итерации цикла происходит полная очистка экрана консоли, вывод графической панели доступных инструментов (интерфейсного меню) и блокирующий опрос клавиатуры для ввода переменной choice.

4. Диспетчеризация команд: обработка введенного пользователем значения выполняется с помощью многоветвевой управляющей конструкции switch(choice). Каждая ветвь (от 1 до 10) представляет собой вызов соответствующей высокоуровневой функции из абстрактного модуля database.h.

5. Корректное завершение: при вводе значения 0 цикл прерывается, управление возвращается в финальный блок функции main(), где происходит обязательный каскадный дескриптор очистки динамической памяти (free_list) во избежание утечек ресурсов, после чего программа завершает работу.

Разработка схемы данных

Основой логической структуры базы данных является реляционная концепция, адаптированная под структуры языка Си. Схема данных жестко фиксирует формат хранения информации о предметной области и реализуется на двух связанных уровнях: уровне отдельной записи (Record) и уровне динамического контейнера памяти (Node).

1. Структура элементарных данных (Record):

Для описания сущности «Страна мира» создается комбинированный тип данных, объединяющий разнородные атрибуты:

– country (массив char[50]) — уникальный текстовый идентификатор (ключ) государства;

– capital (массив char[50]) — текстовое наименование столицы;

– population (вещественный тип double) — численность населения (выбор типа double обусловлен необходимостью корректного представления и масштабирования больших демографических показателей);

– continent (массив char[30]) — географическая привязка к материку.

Использование статических символьных массивов внутри структуры гарантирует фиксированный размер объекта в памяти, что существенно упрощает процедуры прямого чтения и записи полей.

2. Динамическая схема контейнеризации (Node):

Поскольку объем базы данных динамически изменяется во время работы (пользователь может добавлять и удалять записи), организация хранения в виде статического массива неэффективна. Выбрана схема линейного односвязного динамического списка. Каждое звено схемы представляет собой структуру Node, состоящую из двух полей:

– Информационное поле: data типа Record;

– Связующее поле: next типа Node* (указатель на следующий элемент в цепочке оперативной памяти).

Конец списка однозначно идентифицируется указателем NULL. Данная схема позволяет осуществлять вставку и удаление элементов за константное время $O(1)$ без необходимости физического сдвига соседних областей памяти.

3. Детализация планируемых этапов разработки

Процесс проектирования и кодинга программного комплекса разделяется на три иерархических уровня реализации.

Верхний уровень (Уровень 0 — Управляющий интерфейс)

На данном этапе реализуется файл код курсовой.cpp. Задача уровня — развертывание главного меню, организация защищенного ввода пользовательских команд, циклическая диспетчеризация вызовов и визуальное разграничение экранов приложения. На этом этапе программа видится как высокоуровневый автомат состояний.

Средний уровень (Уровень 1 — Операционный функционал СУБД)

Этот этап включает разработку модулей database.h и database.cpp. Создаются и тестируются основные алгоритмы СУБД:

– Алгоритм линейного обхода списка: применяется для табличного вывода информации (print_table) и кастомной фильтрации данных по континентальному признаку (filter_by_continent).

– Алгоритм модификации связей: реализуется в функциях динамического добавления элемента (Notes) и удаления узла по ключу (delete_from_list) с аккуратным перестроением указателей next и освобождением памяти через free().

– Алгоритм сортировки: модификация классической сортировки методом обмена (или пузырьковой сортировки) для динамических указателей, выполняющая упорядочивание элементов по алфавиту.

– Файловый интерфейс: функции save_to_file и load_from_file, отвечающие за последовательный разбор динамических цепочек, перевод их в текстовые потоки и обратное восстановление структуры при чтении дисковых файлов.

Низший уровень (Уровень 2 — Системные утилиты и аппаратная оптимизация)

Разработка вспомогательного модуля minor_functions.h и низкоуровневых вставок. Данный этап детализирует системное окружение:

– Разрабатываются inline-функции для оптимизации скорости вызовов (очистка строк от мусорных символов trim_newline, инициализация дескрипторов WinAPI для VT100).

– Ассемблерный модуль (asm_strcmp)**:** Самый низкий уровень детализации. Алгоритм посимвольного сравнения строк выносится из стандартной библиотеки в оптимизированный блок __asm архитектуры x86. Планируется использование регистров общего назначения (ESI и EDI для хранения адресов строк, AL и BL для считывания текущих символов). Сравнение завершается либо при обнаружении первого несовпадения кодов символов, либо при достижении терминального нуля \0. Результат возвращается в вызывающую функцию через

регистр EAX. Данный этап критически важен для ускорения выполнения операций сквозного поиска и сортировки в базе данных.

Описание разработки программы

Процесс создания программного комплекса базировался на методе нисходящего проектирования. Выбор данного метода обусловлен необходимостью обеспечения целостности архитектуры на всех этапах разработки. Нисходящий подход позволил сначала определить общую логику взаимодействия модулей и структуру интерфейса, а затем постепенно углубляться в реализацию конкретных алгоритмов, что минимизирует вероятность архитектурных ошибок и упрощает стыковку компонентов.

Деление на модули и иерархия функций

Программа спроектирована как модульная система, что обеспечивает высокую степень инкапсуляции и удобство отладки. Структура проекта представлена следующими логическими блоками:

– Модуль описания данных (record.h): Базовый уровень, определяющий типы данных (Record, Node). Он выступает «фундаментом» для всех остальных модулей.

– Модуль управления базой данных (database.h, database.cpp): Центральный исполнительный блок. Содержит логику работы со списком (добавление, удаление, сортировка, поиск) и файловые операции.

– Модуль системных утилит (minor_functions.h): Сервисный слой. Включает функции для работы с консолью (VT100), очистки ввода (trim_newline) и визуализации заставки.

– Управляющий модуль (код курсовой.cpp): «Вершина» иерархии. Содержит точку входа main(), реализует главный цикл обработки команд и визуальное представление интерфейса.

Порядок реализации функций

Реализация функционала производилась поэтапно:

Этап 1: Создание структур данных и каркаса интерфейса. На первом этапе были описаны структуры данных в record.h и создан главный цикл в main(),

включая инициализацию консоли. Это позволило получить работающую «пустую» программу с меню, к которой можно было подключать функционал.

Этап 2: Базовые манипуляции с памятью. Реализованы функции Notes и free_list. Без этих функций невозможно было бы накопление данных в динамической памяти, поэтому они стали приоритетными. На этом же этапе была внедрена очистка строк trim_newline для корректной обработки ввода.

Этап 3: Файловый ввод-вывод. Реализованы функции save_to_file и load_from_file. Наличие возможности сохранения данных позволило ускорить процесс тестирования: не требовалось каждый раз вводить записи вручную — достаточно было один раз создать файл и загружать его при запуске.

Этап 4: Алгоритмизация и оптимизация. На данном этапе были реализованы функции поиска (find_record), редактирования (edit_record), фильтрации (filter_by_continent) и сортировки (sort_list). Важным шагом стала интеграция ассемблерной вставки asm_strcmp внутри функции поиска. Это позволило протестировать низкоуровневое сравнение строк в изолированном модуле и применить его для ускорения остальных операций.

Этап 5: Финализация и тестирование. Заключительный этап включал создание заставки (showLoadingScreen), которая была интегрирована в процесс инициализации, а также проведение комплексных тестов на корректность перестроения указателей при удалении записей из середины или начала списка, чтобы избежать потери данных (утечек памяти).

Отладка и тестирование

Процесс отладки и тестирования является критически важным этапом жизненного цикла разработки, позволяющим верифицировать корректность работы алгоритмов, стабильность управления динамической памятью и устойчивость интерфейса к некорректным действиям пользователя.

Методы отладки

В ходе работы применялись следующие методы обеспечения качества программного продукта:

1. Модульное тестирование: каждый из модулей (database.cpp, minor_functions.h) тестировался изолированно. Например, функция asm_strcmp проверялась на сравнении строк разной длины, строк с идентичными префиксами и пустых строк, что позволило убедиться в корректности работы ассемблерного кода до его интеграции в общую систему.

2. Пошаговая отладка: с помощью встроенного отладчика среды разработки (MS Visual Studio) проводилось пошаговое выполнение кода. Это позволило отслеживать изменение указателей next в динамическом списке и предотвращать «потерю» узлов при операциях добавления и удаления.

3. Тестирование граничных условий: Программа подвергалась проверкам на «аномальное» поведение:

- Попытка удаления записи из пустого списка.
- Попытка удаления несуществующей страны.
- Ввод сверхдлинных строк (проверка корректности работы с fgets и trim_newline).
- Загрузка несуществующего или пустого файла базы данных.

4. Контроль утечек памяти: Особое внимание уделялось функции free_list. В ходе отладки использовались методы принудительного вызова деструкторов списка, чтобы убедиться, что при завершении работы программы все выделенные через malloc блоки памяти корректно освобождаются.

Выявленные ошибки и их исправление

В процессе отладки были устранены следующие ключевые недочеты:

– Проблема «мусорных» символов: изначально при вводе данных через fgets в структуру попадал символ переноса строки (\n). Это приводило к некорректному отображению таблицы и ошибкам при сравнении строк. Решение: разработка и внедрение функции trim_newline в модуле minor_functions.h, которая принудительно заменяет символ \n на терминальный ноль \0.

– Ошибка сегментации при удалении: при попытке удаления первого элемента списка возникала ошибка, так как функция не обновляла указатель head в вызывающей функции. Решение: передача указателя на указатель (Node head) в функции изменения списка, что позволило корректно перенаправлять корень списка.

– Некорректная работа ассемблерной вставки: на ранних этапах наблюдалась потеря данных в регистрах при переключении контекста. Решение: добавление инструкций push и pop для регистров ESI и EDI в блоке __asm, что обеспечило сохранение целостности стека при сравнении строк.

Результаты тестирования

Итоговое тестирование подтвердило соответствие разработанной системы техническому заданию:

1. Функциональность: Все заявленные операции (добавление, поиск, сортировка, фильтрация) отрабатывают согласно логике СУБД.

2. Целостность: Динамический список корректно перестраивается при любых манипуляциях, утечек памяти при работе программы не зафиксировано.

3. Интерфейс: Окно консоли стабильно удерживает заданный размер 80x30, использование VT100-последовательностей обеспечивает чистоту визуального вывода без «артефактов».

4. Надежность: Программа устойчива к некорректному вводу и сохраняет работоспособность при работе с файлами данных, что позволяет эффективно использовать её для решения прикладных задач по хранению информации о столицах стран.

Результатом тестирования стала полностью работоспособная версия программы, готовая к эксплуатации в учебных и справочных целях.

Описание программы

Данный раздел содержит описание архитектуры и принципов взаимодействия компонентов разработанного программного комплекса «База данных по столицам стран».

Состав модулей и иерархия

Программа построена на принципе многомодульности, что обеспечивает четкое разделение ответственности:

– main.cpp (код курсовой.cpp): Управляющий модуль. Осуществляет инициализацию консоли, запуск цикла обработки команд (while), диспетчеризацию действий пользователя и управление жизненным циклом списка (head).

– database.h / .cpp: Логическое ядро. Реализует структуру хранения данных и методы манипуляции: CRUD-операции (создание, чтение, обновление, удаление), файловый обмен и сортировку.

– record.h: Описательный модуль. Содержит определение структуры Record (данные страны) и Node (узел динамического списка).

– minor_functions.h: Сервисный модуль. Содержит вспомогательные функции для форматирования консоли (VT100), очистки ввода и реализации заставки.

Спецификация функций

asm_strcmp — функция низкоуровневого посимвольного сравнения двух строк, реализованная с помощью встроенной ассемблерной вставки __asm для архитектуры x86. Принимает два указателя на константные строки const char* a и const char* b. Сравнение выполняется в циклическом режиме с использованием регистров общего назначения ESI и EDI в качестве указателей на текущие адреса, а также регистров AL и BL для считывания и сопоставления кодов текущих символов. Функция завершает работу либо при обнаружении первого несовпадения символов, либо при достижении терминального нуля \0, и возвращает целое число (int) через регистр EAX: значение 0 при полной идентичности строк, или разность ASCII-кодов первых несовпавших элементов

в ином случае. Блок-схема алгоритма низкоуровневого сравнения строк приведена на рисунке 1.

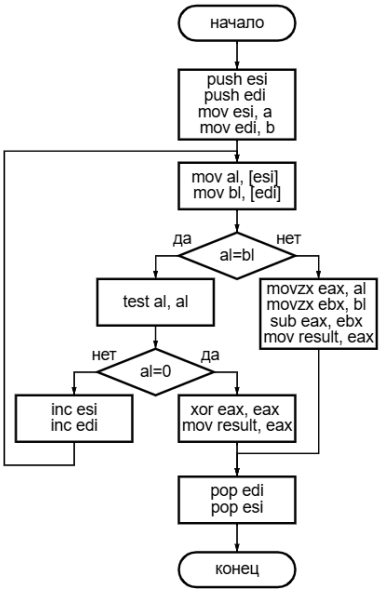

Рисунок 1 – блок-схема функции asm_strcmp

add_to_list — функция динамического добавления новой записи в информационную структуру базы данных. Принимает двойной указатель на начало односвязного списка Node head и заполненную структуру данных Record rec, содержащую сведения о стране. Функция выполняет динамическое выделение блока оперативной памяти под новый узел типа Node с помощью системного вызова sizeof и функции malloc, копирует переданную структуру в информационное поле созданного элемента и осуществляет перестроение указателей для вставки узла в начало цепочки. Функция не возвращает значений (void). Блок-схема функции представлена на рисунке 2.

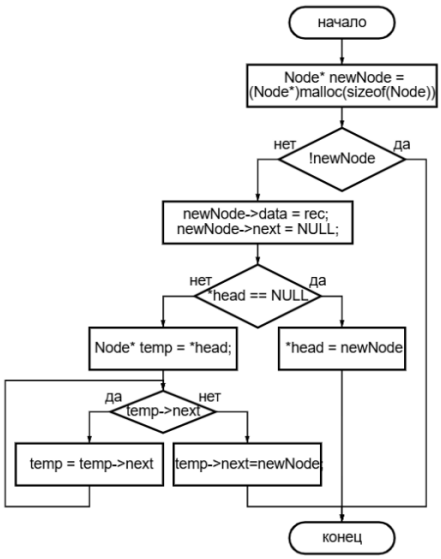

Рисунок 2 – блок-схема функции add_to_list

load_from_file — функция десериализации данных, отвечающая за первичное развертывание динамической структуры из файла на жестком диске. Принимает в качестве параметра строку с именем файла const char* filename. Функция открывает указанный файл в режиме чтения ("r"), в циклическом режиме построчно считывает текстовые и числовые поля (название страны, столицы, население и континент), формирует на их основе структуры типа Record и последовательно наполняет список с помощью функции add_to_list. Возвращает указатель на сформированный корень списка Node* или значение NULL в случае возникновения ошибки открытия файла. Блок-схема функции представлена на рисунке 3.

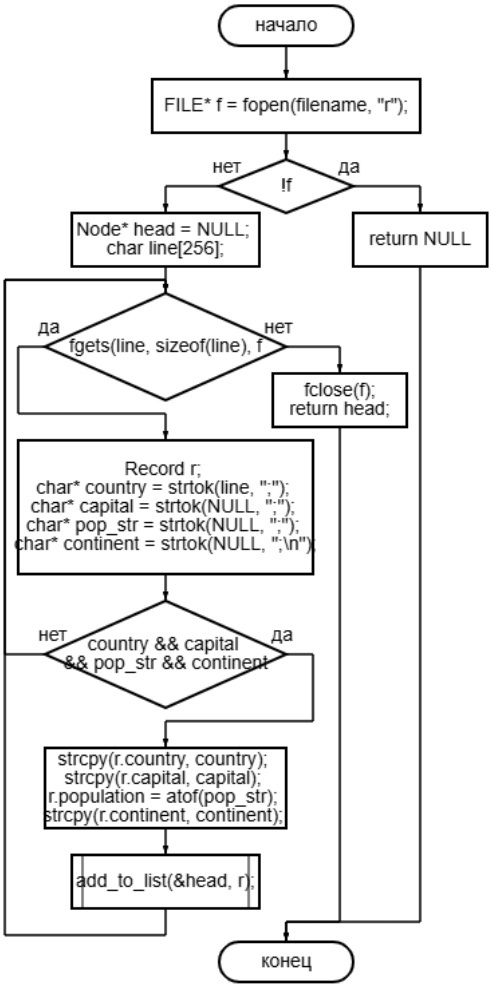

Рисунок 3 – блок-схема функции load_from_file

save_to_file — функция сериализации текущего состояния базы данных из оперативной памяти на внешний носитель. Принимает указатель на первый элемент списка Node* head и строку с именем целевого файла const char* filename. Функция открывает или перезаписывает текстовый файл в режиме

записи ("w"), осуществляет последовательный обход списка от корня до терминального указателя NULL и построчно фиксирует поля каждой структуры Record с сохранением форматированных разделителей. Функция не возвращает значений (void). Графическое представление процесса сохранения связного списка на диск отображено на рисунке 4.

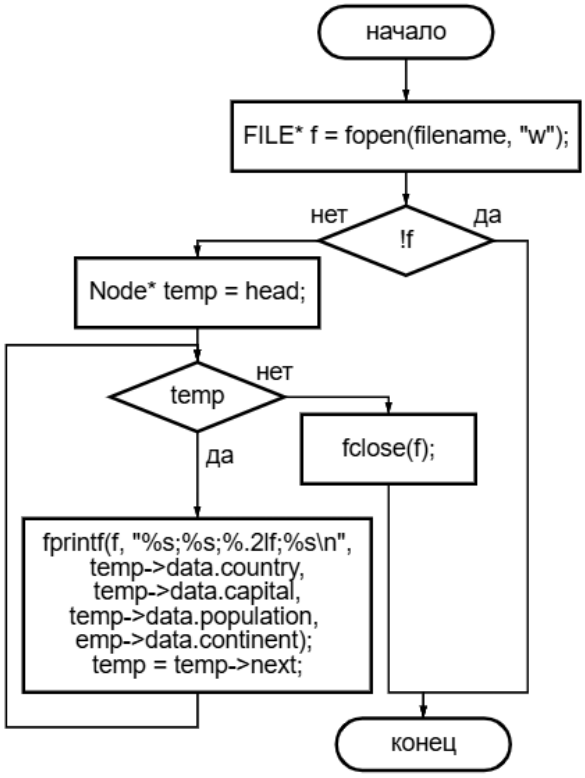

Рисунок 4 – блок-схема функции save_to_file

delete_from_list — функция каскадного удаления записи из базы данных по ключевому текстовому признаку. Принимает двойной указатель на заголовок списка Node head и строку const char* country с названием удаляемого государства. Функция выполняет поиск целевого узла в списке; при обнаружении совпадения ключей с помощью asm_strcmp она перенаправляет указатель next предыдущего элемента в обход удаляемого узла, после чего физически освобождает память через free, корректно обрабатывая граничные случаи удаления первого или единственного элемента. Функция не возвращает значений (void). Схема исключения узла из цепочки списка приведена на рисунке 5.

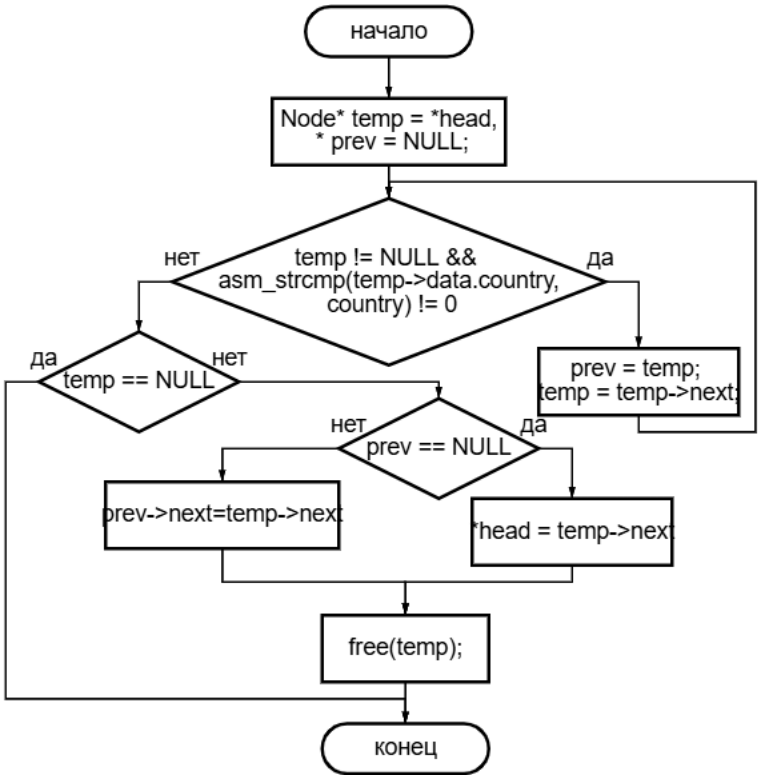

Рисунок 5 – блок-схема функции delete_from_list

free_list — функция полной очистки динамической памяти, распределенной под хранение элементов односвязного списка. Принимает указатель на корень списка Node* head. Функция осуществляет последовательный линейный обход всей динамической структуры, используя временный указатель для сохранения адреса следующего элемента перед вызовом процедуры деаллокации памяти free для текущего узла, что гарантирует полное отсутствие утечек ресурсов при закрытии программы или перезагрузке файлов. Функция не возвращает значений (void). Схема алгоритма пошагового освобождения динамической памяти проиллюстрирована на рисунке 6.

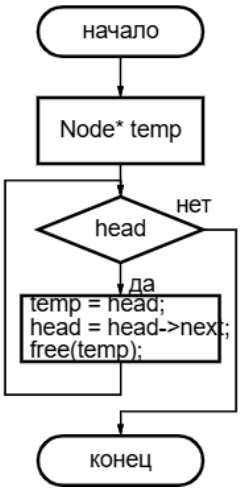

Рисунок 6 – блок-схема функции free_list

save_database_as — функция сохранения текущих данных в текстовый файл под новым именем с переназначением рабочего файла СУБД. Принимает указатель на список Node* head. Функция запрашивает у пользователя новое имя файла, обрабатывает его через trim_newline, инициирует вызов save_to_file для записи текущего списка и обновляет глобальную переменную current_file, делая новый файл активным для последующих автосохранений. Функция не возвращает значений (void). Логика переназначения активного файла базы данных приведена на рисунке 7.

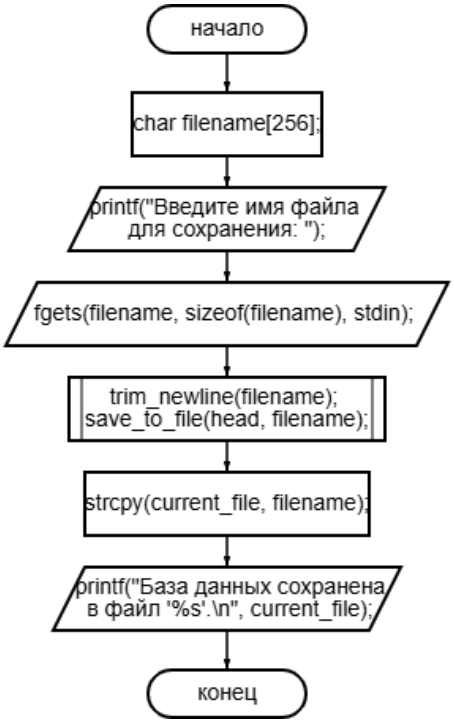

Рисунок 7 – блок-схема функции save_database_as

print_table — функция вывода содержимого базы данных на экран в виде строго структурированной таблицы. Принимает указатель на начало списка Node* head. Функция осуществляет обход списка, форматирует вывод текстовых массивов и вещественных чисел (население стран) с заданием фиксированной ширины столбцов и элементов псевдографики, обеспечивая просматриваемость и наглядность отображения информации в окне консоли. Функция не возвращает значений (void). Схема формирования табличного интерфейса отображена на рисунке 8.

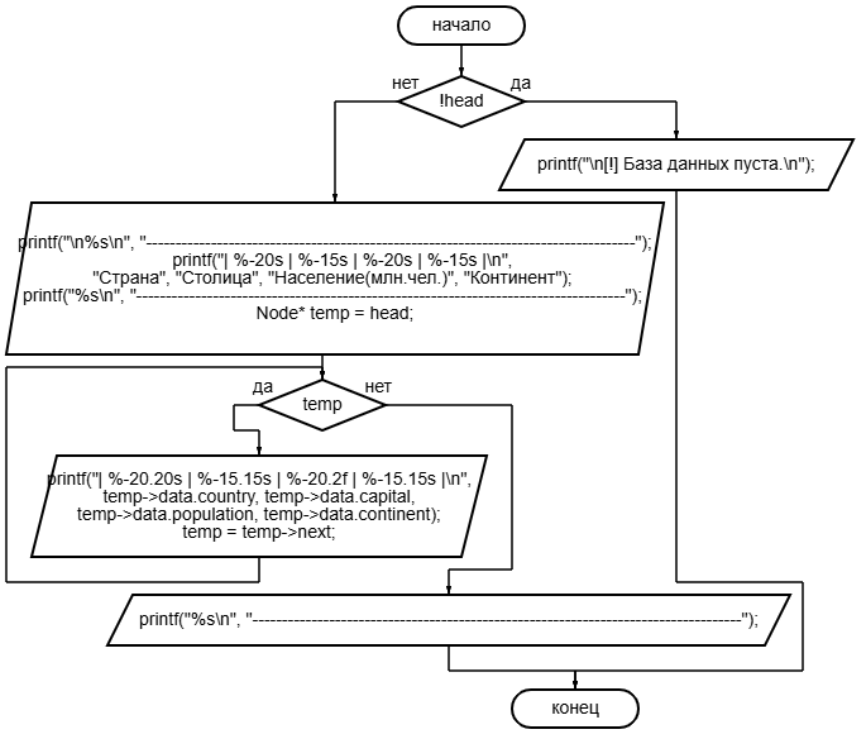

Рисунок 8 – блок-схема функции print_table

trim_newline — встраиваемая (inline) функция предварительной обработки строк, введенных пользователем с клавиатуры. Принимает указатель на символьный массив char* str. Функция вычисляет длину строки с помощью strlen и, если на конце строки обнаруживается символ перевода каретки (\n), принудительно заменяет его на нулевой символ \0 для обеспечения корректной работы файловой системы и строковых компараторов. Функция не возвращает значений (void). Схема работы функции очистки строк приведена на рисунке 9.

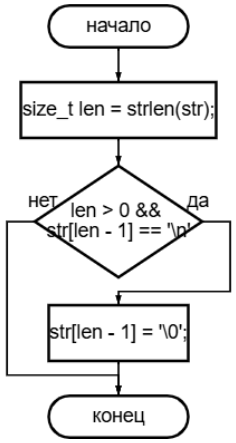

Рисунок 9 – блок-схема функции trim_newline

find_record — функция поиска информации о конкретном государстве в оперативной памяти. Принимает указатель на корень списка Node* head и искомую текстовую строку const char* country. Функция производит

последовательный линейный перебор элементов списка, вызывая на каждой итерации ассемблерный блок asm_strcmp для сопоставления полей, и возвращает прямой указатель на найденный узел Node* для его последующей обработки, либо NULL, если совпадений в базе данных не обнаружено. Схема логики ветвления при поиске записи приведена на рисунке 10.

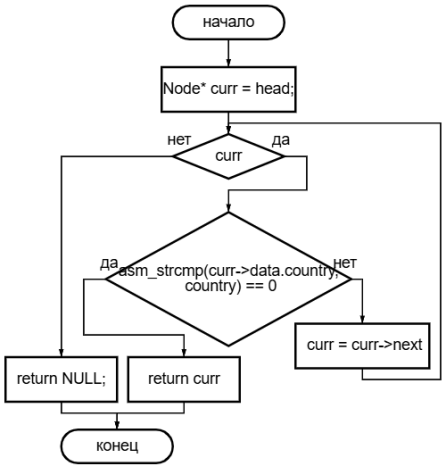

Рисунок 10 – блок-схема функции find_record

create_new_database — функция инициализации новой пустой базы данных. Принимает двойной указатель на заголовок списка Node head. При вызове функция полностью уничтожает текущий динамический список в оперативной памяти через free_list, обнуляет указатель *head, запрашивает у пользователя новое имя файла с клавиатуры, очищает его от мусорных символов и создает на диске пустой файл через fopen в режиме "w". Функция не возвращает значений (void). Схема последовательности сброса контекста и создания файла представлена на рисунке 11.

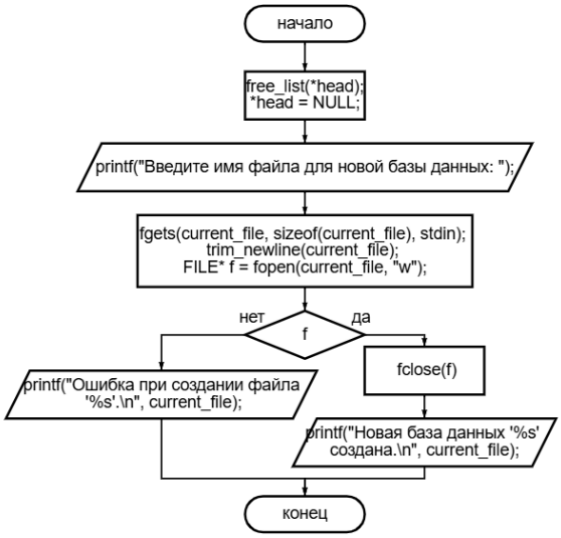

Рисунок 11 – блок-схема функции create_new_database

edit_record — функция интерактивного изменения полей существующей записи. Принимает указатель на начало списка Node* head и строку const char* country. Функция вызывает find_record для локализации целевого элемента; если элемент успешно найден, программа выводит текущие параметры и запрашивает у пользователя новые текстовые и числовые значения для столицы, населения и континента, перезаписывая информационные поля структуры Record. Функция не возвращает значений (void). Порядок изменения полей записи в памяти показан на рисунке 12.

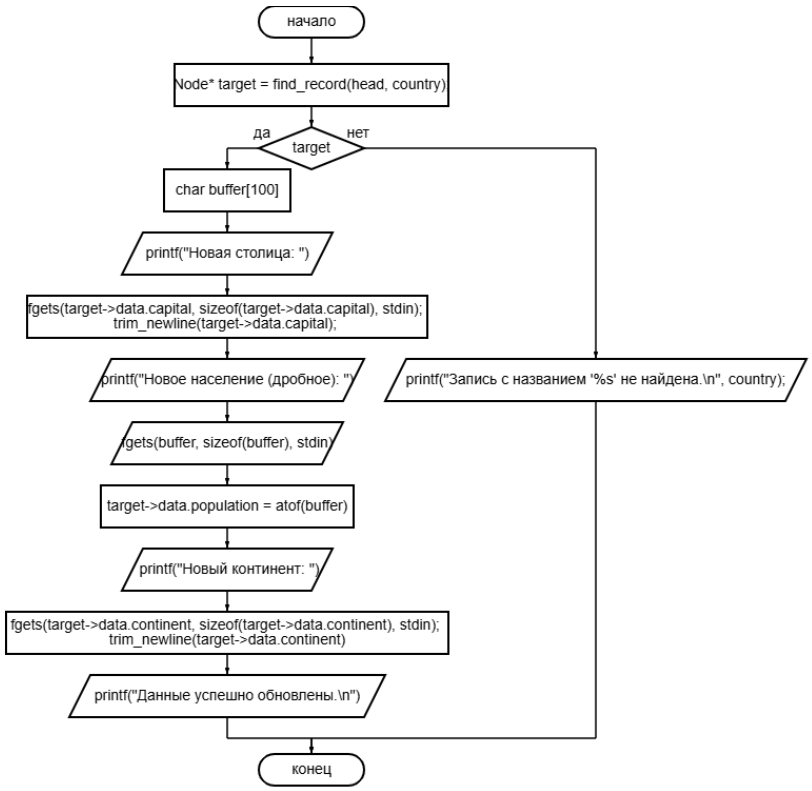

Рисунок 12 – блок-схемы функции edit_record

filter_by_continent — функция выборки и фильтрации данных по географическому признаку для формирования аналитических отчетов. Принимает указатель на список Node* head и строку-критерий const char* continent. Функция обходит всю динамическую цепочку, сравнивает символьное поле континента каждой записи с заданным пользователем критерием через asm_strcmp и выводит в консоль строки таблицы только для тех государств, которые полностью удовлетворяют условию выборки. Функция не возвращает значений (void). Схема генерации отчета по фильтру приведена на рисунке 13.

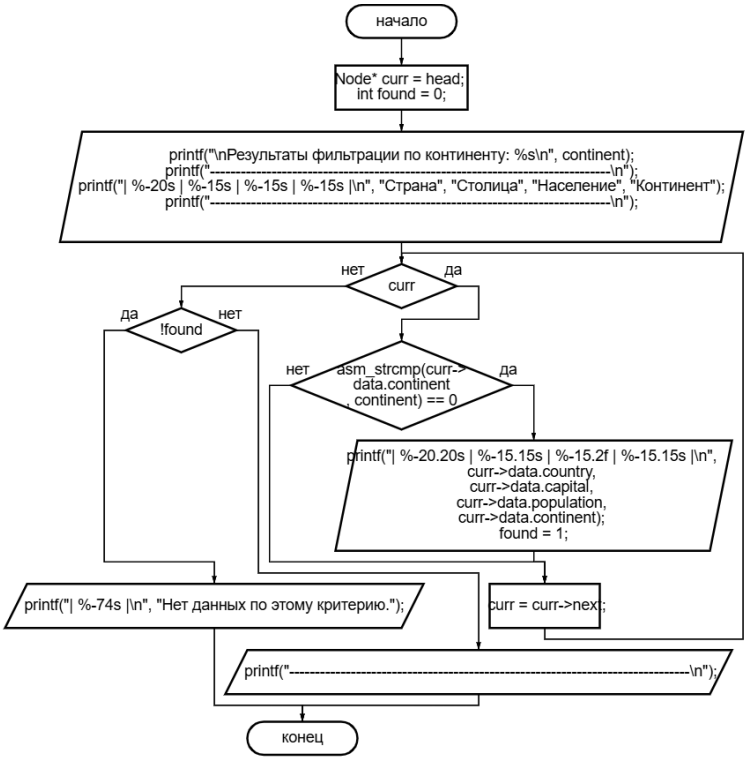

Рисунок 13 – блок-схема функции filter_by_continent

load_database — функция интерактивной смены и загрузки альтернативного файла базы данных. Принимает двойной указатель на корень списка Node head. Функция запрашивает имя нового файла, проверяет его физическое существование на диске, и при успешном открытии сначала освобождает текущую память через free_list, а затем вызывает load_from_file, обновляя указатель *head и имя текущего файла. Функция не возвращает значений (void). Схема переключения контекста баз данных отображена на рисунке 14.

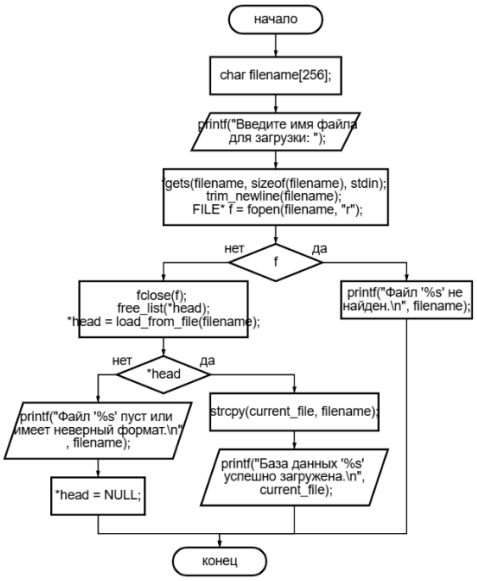

Рисунок 14 – блок-схема функции load_database

sort_list — функция алфавитной сортировки записей базы данных по названию стран. Принимает указатель на начало списка Node* head. Внутри функции реализован модифицированный алгоритм сортировки методом обмена (пузырьковая сортировка), адаптированный для работы с линейными связными структурами. Посимвольное сравнение ключевых полей выполняется через вызов ассемблерной функции asm_strcmp, а при нарушении алфавитного порядка осуществляется безопасный взаимный обмен информационными полями data (структур Record) внутри узлов без физического изменения адресов самих элементов Node в памяти. Функция не возвращает значений (void). Схема обмена данными между узлами при сортировке представлена на рисунке 15.

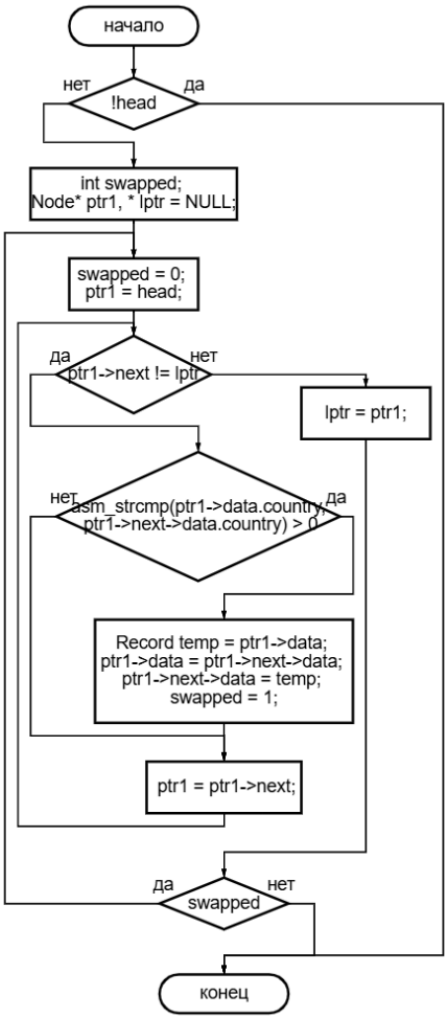

Рисунок 15 – блок-схема функции sort_list

setupConsole — встраиваемая (inline) функция низкоуровневой инициализации видеосистемы приложения. Принимает параметры ширины и высоты окна терминала в символах (int cols, int lines). С помощью функций

WinAPI запрашивает дескриптор стандартного вывода, активирует низкоуровневый режим обработки ENABLE_VIRTUAL_TERMINAL_ PROCESSING для полноценной поддержки ANSI/VT100-последовательностей и отправляет в поток специальный управляющий код \x1b[8;H;Wt для жесткой фиксации размеров окна. Функция не возвращает значений (void). Схема настройки системных флагов терминала показана на рисунке 16.

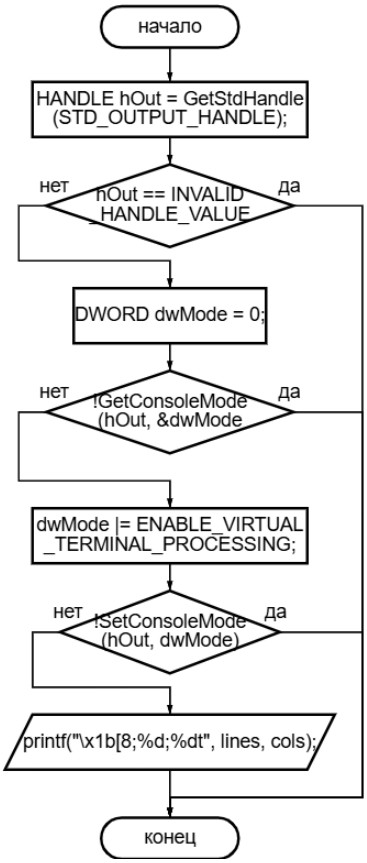

Рисунок 16 – блок-схема функции setupConsole

showLoadingScreen — встраиваемая (inline) функция отрисовки стартового экрана (заставки) приложения. Не имеет входных параметров и возвращаемого значения. Функция выполняет очистку экрана консоли (system("cls")), выводит на экран академические метаданные о курсовой работе (название, дисциплину, тему) и в фиксированном цикле отрисовывает псевдографический индикатор выполнения (progress bar) с использованием временных задержек, визуально маскируя процесс развертывания программной среды. Схема последовательности вывода заставки представлена на рисунке 17.

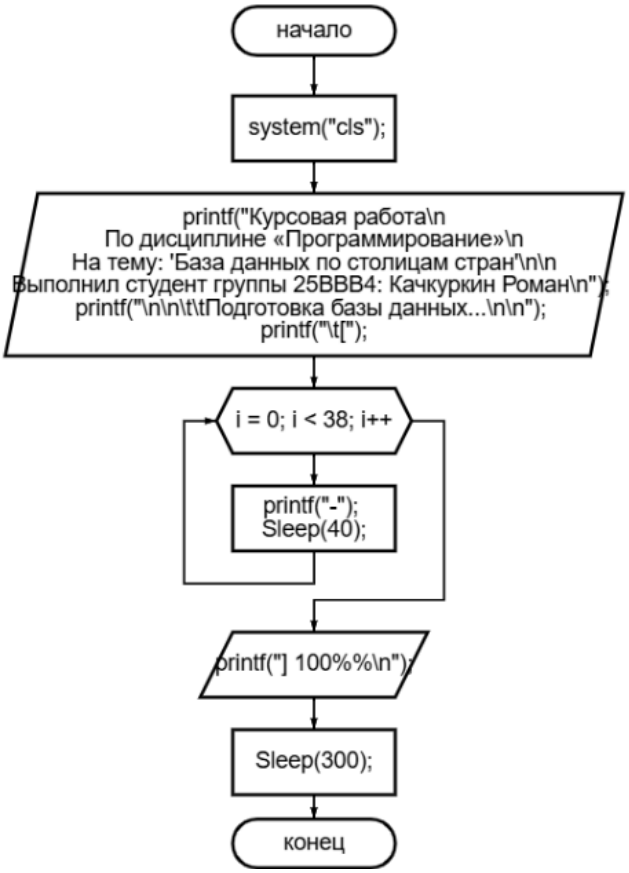

Рисунок 17 – блок-схема функции showLoadingScreen

Схемы работы системы

Ниже представлена схема основной функции main программы. При запуске будет показано меню, в котором пользователь выбирает функцию. Блок-схема функции main представлена на рисунке 18.

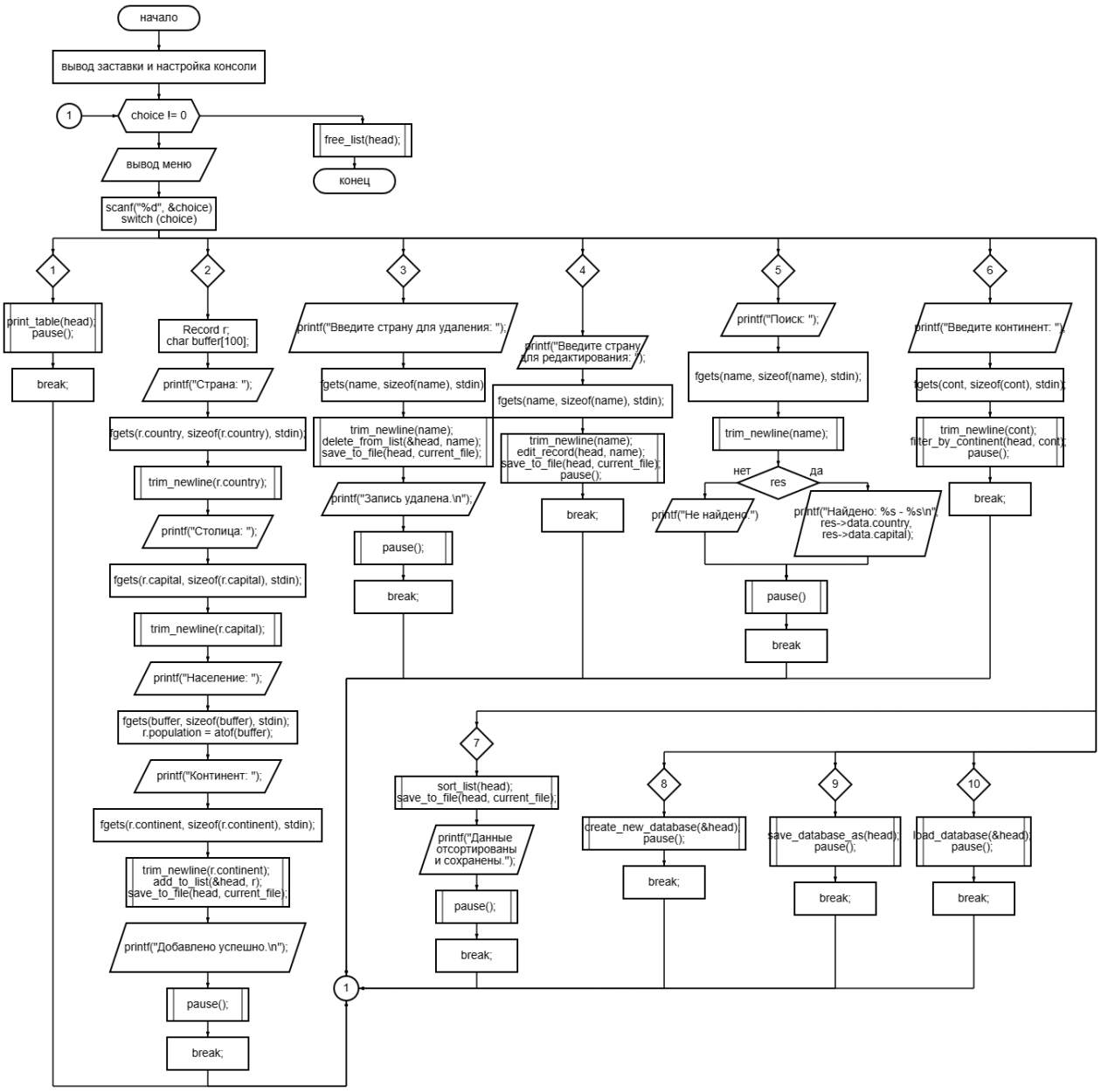

Рисунок 18 – блок-схема функции main

Руководство пользователя

Программа код курсовой.cpp предназначена для хранения информации о столицах стран. Программа имеет интуитивно понятный интерфейс и поддерживает такие операции как добавление новой записи в базу данных, удаление записи из базы данных, изменение записи в базе данных, поиск записи в базе данных, сохранение существующей базы данных в файл, импорт базы данных из файла, сортированный вывод записей базы данных и фильтрация записей в базе данных. Для открытия и вывода на экран существующего каталога необходимо выбрать пункт меню «Показать записи». На рисунке 19 и рисунке 20 изображен скриншот терминала в главном меню программы и записи базы данных.

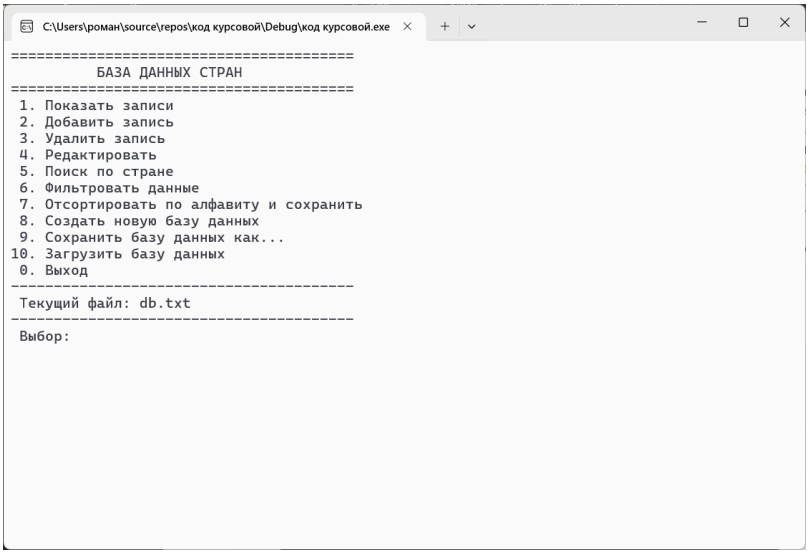

Рисунок 19 – главное меню

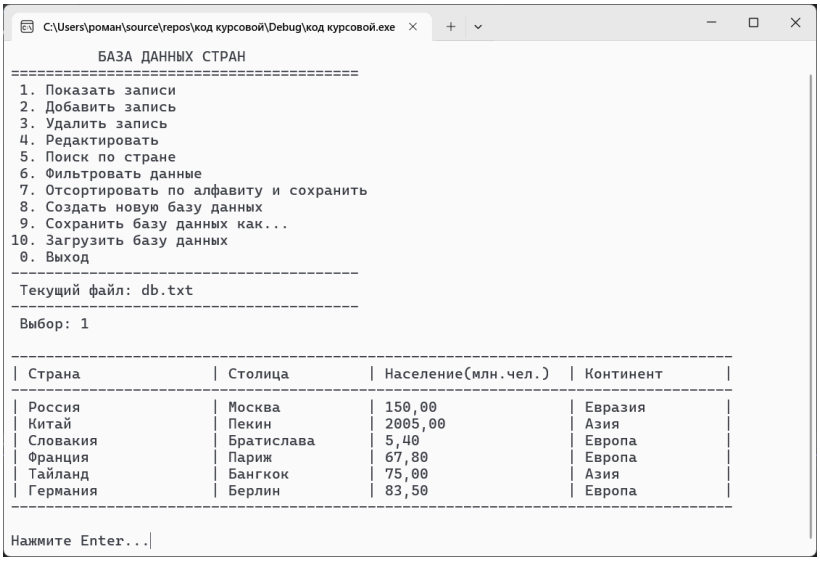

Рисунок 20 – имеющиеся записи

Для удаления записи из каталога необходимо выбрать пункт меню «Удалить». Далее необходимо ввести название страны удаляемой записи и нажать клавишу Enter. В случае если не возникли ошибки на экране появится надпись «Запись удалена.». На рисунке 21 изображен скриншот терминала при удалении записи.

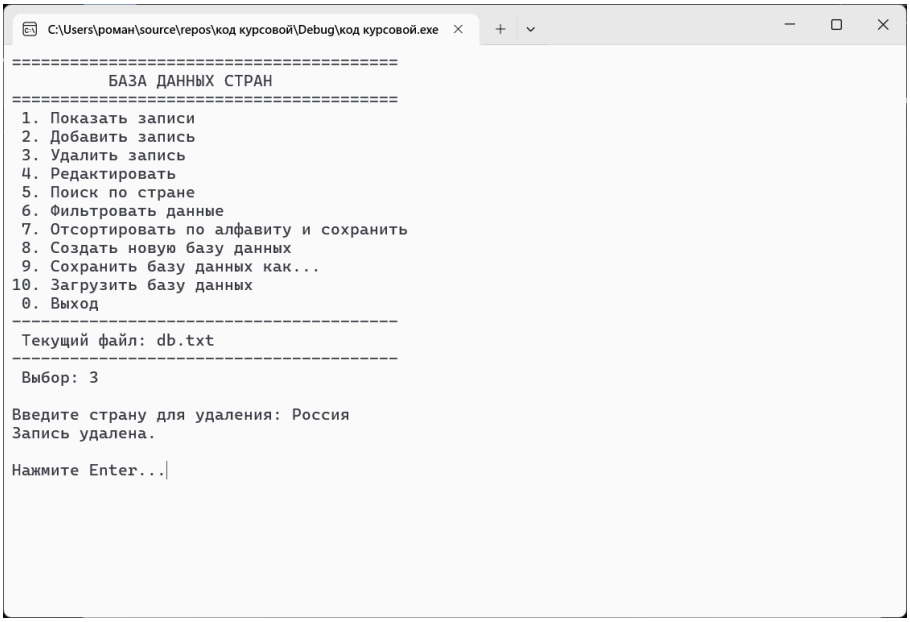

Рисунок 21 – демонстрация работы функции удаления

Для добавления записи в каталог необходимо выбрать пункт меню 2 «Добавить запись». Далее надо вписать данные записи о названии страны, столице, населении и континенте. При успешном добавлении будет выведено сообщение «Добавлено успешно». На рисунке 22 изображен скриншот терминала после добавления записи.

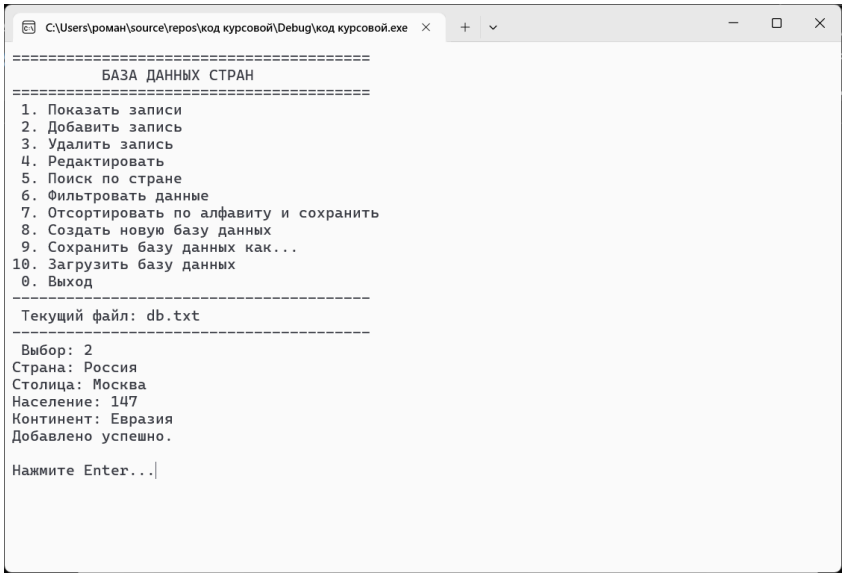

Рисунок 22 – демонстрация работы функции добавления записи

Для изменения записи необходимо выбрать пункт 4 «Редактировать» в меню. Далее необходимо ввести название страны изменяемой записи и ввести новые данные для записи из трех пунктов: ввести название страны, нажать клавишу Enter, ввести столицу страны, нажать клавишу Enter, ввести население, нажать клавишу Enter, ввести континент, нажать клавишу Enter. В случае если не возникли ошибки на экране появится надпись «Данные успешно обновлены». На рисунке 23 изображен скриншот терминала после редактирования записи.

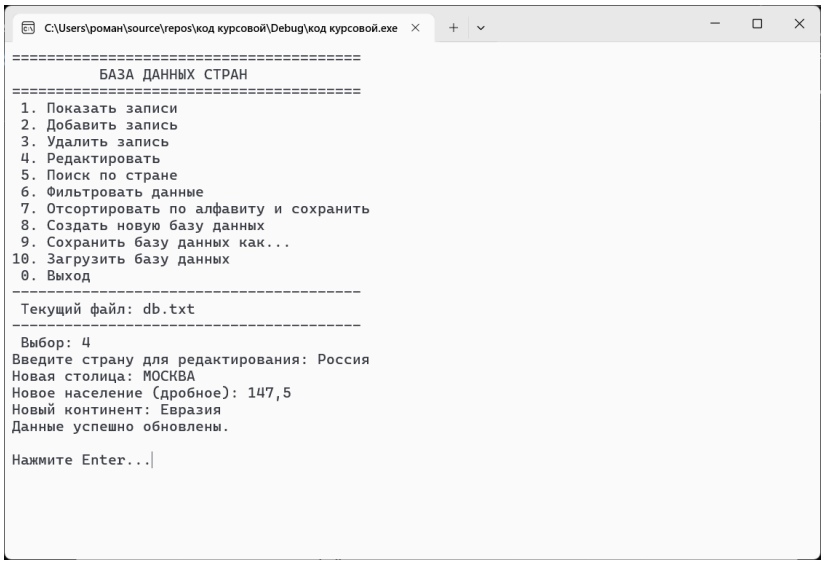

Рисунок 23 – демонстрация работы функции редактирования

Для поиска по названию страны необходимо выбрать пункт 5 «Поиск по стане». Далее необходимо ввести название страны. При успешном нахождении записи будет выведено название страны и её столица. На рисунке 24 изображен скриншот терминала после выполнения поиска.

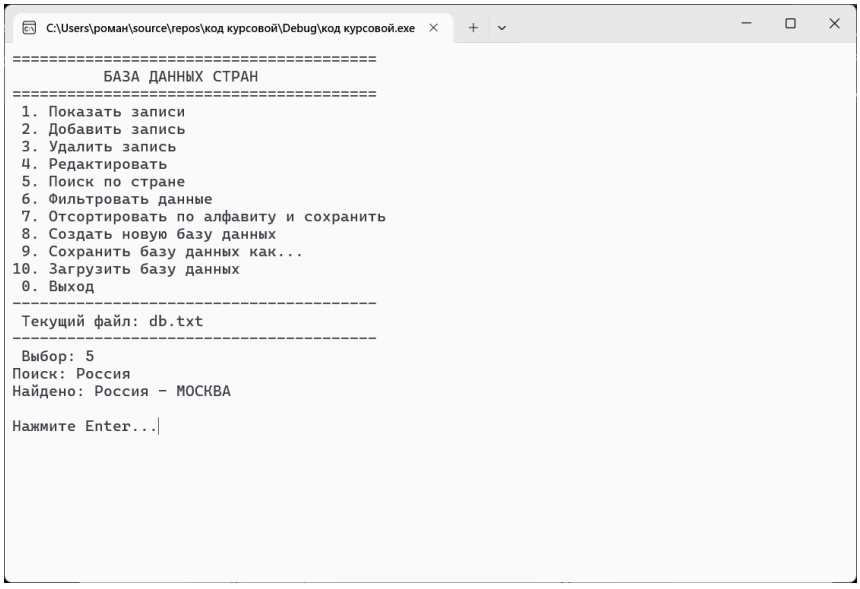

Рисунок 24 – демонстрация работы поиска

Чтобы отфильтровать записи пользователь должен выбрать пункт 6 «Фильтровать данные». Далее необходимо ввести название континента и нажать клавишу Enter. Будет выведена таблица с отфильтрованными данными по критерию континент. На рисунке 25 изображен скриншот терминала после фильтрации.

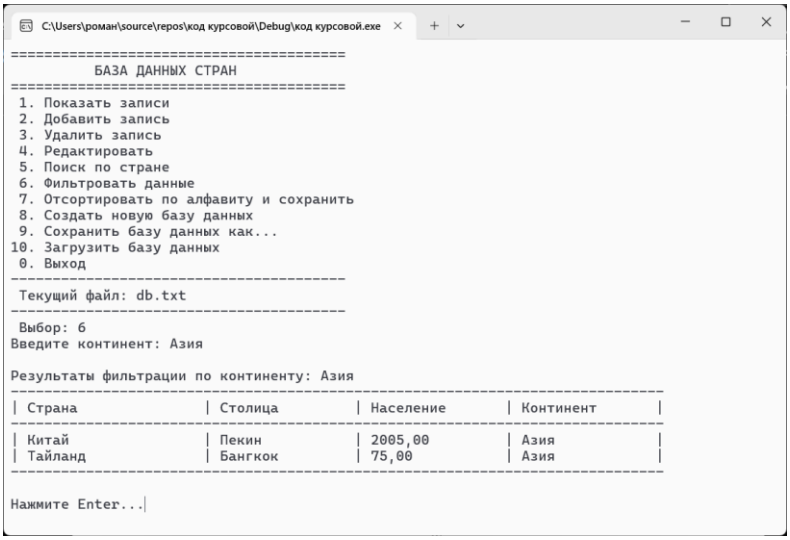

Рисунок 25 – демонстрация работы функции фильтрации

Чтобы отсортировать данные необходимо выбрать пункт 7 «Отсортировать по алфавиту и сохранить». Будет произведена автоматическая сортировка списка по алфавиту с сохранением в файл. На рисунке 26 изображен скриншот терминала после выполнения сортировки.

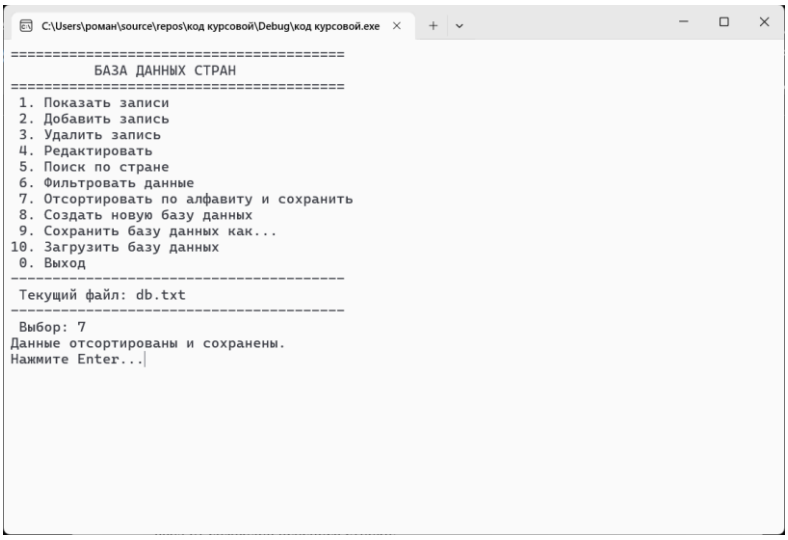

Рисунок 26 – демонстрация работы функции сортирования

Для создания новой БД необходимо выбрать пункт 8 «Создать новую базу данных». Далее нужно будет ввести название файла с расширением. Будет создан новый файл. На рисунке 27 изображен скриншот терминала после создания БД.

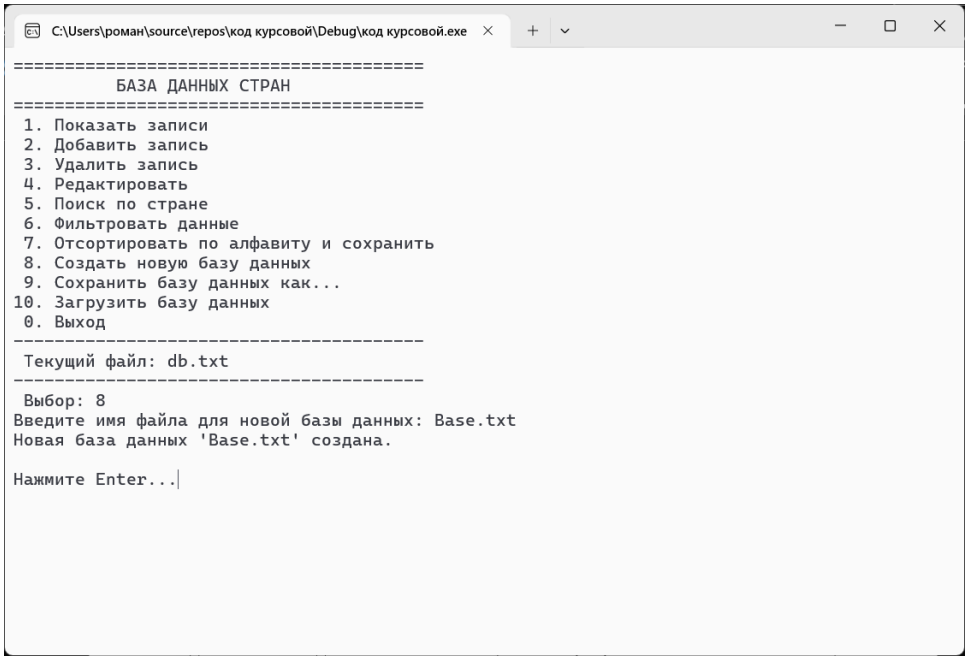

Рисунок 27 – демонстрация работы функции создания новой БД

Для того чтобы сохранить базу данных как необходимо выбрать пункт 6 «Сохранить базу данных как…». Будет необходимо ввести новое название файла и нажать клавишу Enter. Будет выведено сообщение «База данных сохранена в файл ‘название файла’.». На рисунке 28 изображен скриншот терминала после сохранения базы данных.

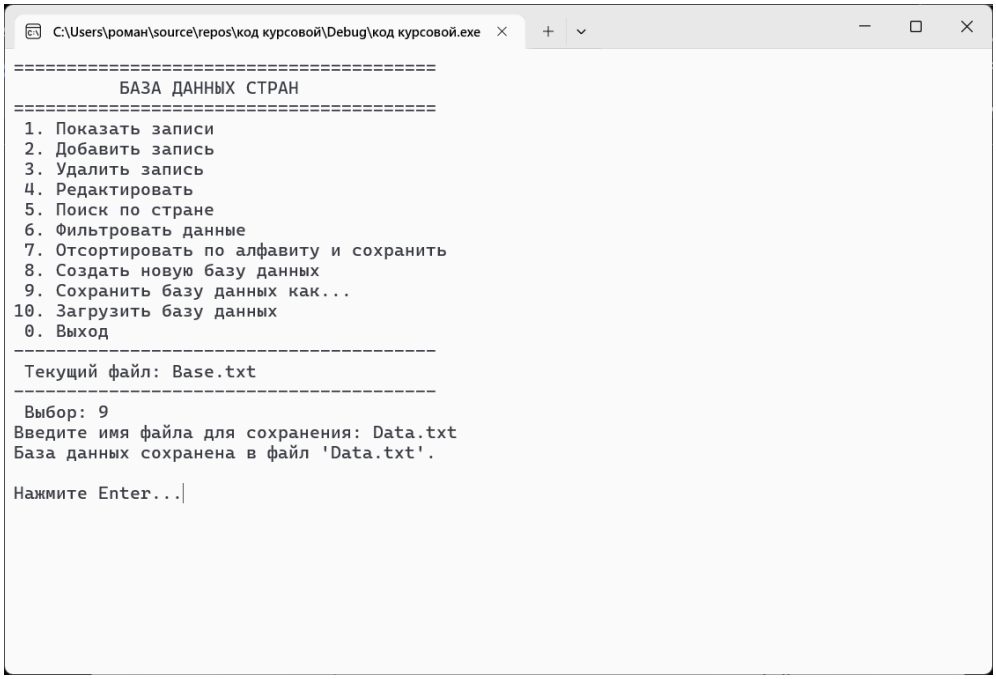

Рисунок 28 – демонстрация работы функции сохранения базы данных

Для того чтобы загрузить базу данных их папки проекта необходимо выбрать пункт 10 «Загрузить базу данных». Далее необходимо ввести название файла с расширением. При успешном нахождении файла будет выведено

сообщение «База данных 'название файла' успешно загружена.». На рисунке 29 изображен скриншот терминала после загрузки базы данных.

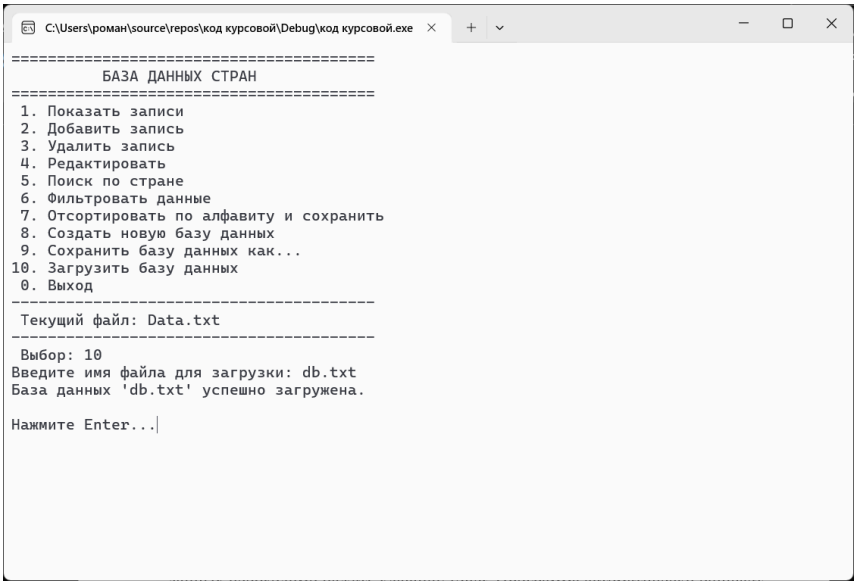

Рисунок 29 – демонстрация работы функции загрузки базы данных

Для выхода из программы необходимо выбрать пункт 0 «Выход». Завершение работы программы с очисткой оперативной памяти. На рисунке 30 изображен скриншот терминала после выбора пункта 0 и нажатия клавиши Enter.

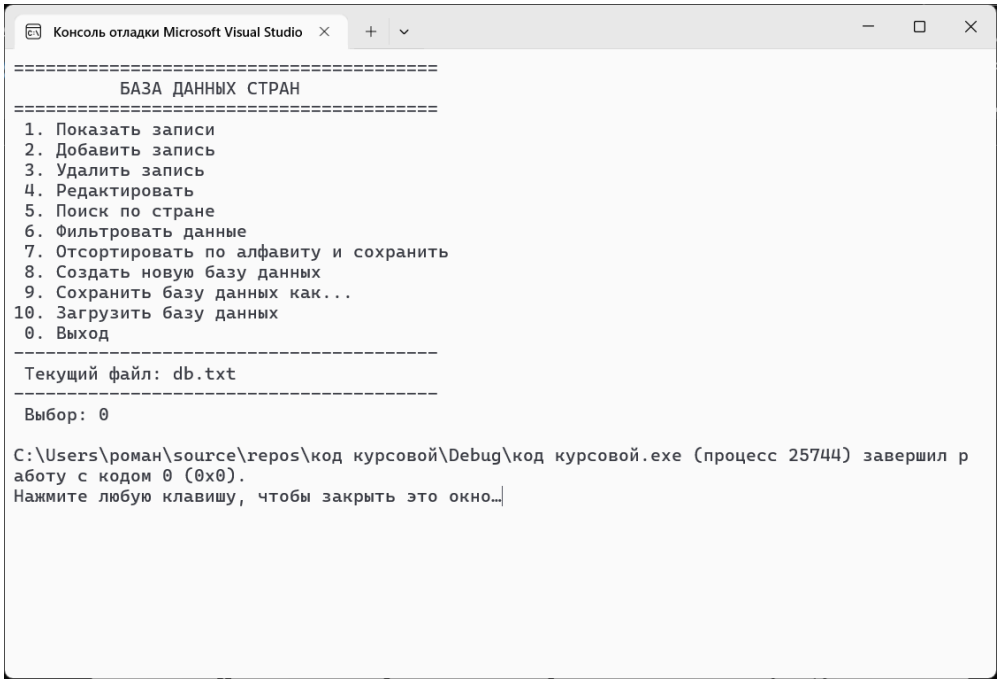

Рисунок 30 – демонстрация работы функции выхода

Заключение

В ходе выполнения курсовой работы была спроектирована и реализована локальная информационно-поисковая система «База данных по столицам стран». Работа над проектом позволила комплексно подойти к процессу разработки программного обеспечения: от первичной постановки задачи до финальной отладки и тестирования готового продукта.

Приобретенные навыки и компетенции

В процессе работы над курсовым проектом были закреплены и развиты следующие профессиональные навыки:

1. Работа с динамическими структурами данных: было достигнуто глубокое понимание принципов работы с односвязными списками, процедур выделения и освобождения памяти в «куче» (heap), а также методов предотвращения утечек памяти, что является критически важным для надежного системного программирования.

2. Архитектурное проектирование: Освоена техника нисходящего проектирования и модульной организации кода. Разделение проекта на логические единицы (database, record, minor_functions) позволило систематизировать разработку и упростить процесс отладки.

3. Оптимизация низкого уровня: Получен уникальный опыт интеграции ассемблерного кода (__asm) в проект на языке высокого уровня. Написание функции сравнения строк на языке ассемблера позволило на практике разобраться в работе регистров процессора, стека и механизмов управления памятью архитектуры x86.

4. Взаимодействие с системными средствами ОС: Освоена работа с функциями WinAPI для настройки параметров консоли, а также применение управляющих последовательностей VT100, что позволило создать эстетически оформленный и функциональный пользовательский интерфейс в текстовом режиме.

5. Алгоритмическая подготовка: Реализация механизмов сортировки, поиска и фильтрации данных закрепила навыки работы с итераторами, указателями и сложными алгоритмами обработки массивов.

Общий вывод

Курсовая работа продемонстрировала высокую эффективность выбранного метода нисходящего проектирования. Программа представляет собой целостный, масштабируемый комплекс, который может быть использован как база для дальнейшего изучения дисциплин «Системное программирование» и «Алгоритмы и структуры данных». Практический результат работы подтверждает, что даже в рамках консольных приложений можно реализовать мощные функциональные инструменты управления данными, обладающие высокой скоростью работы и низким потреблением системных ресурсов.

Работа над проектом сформировала понимание того, что качественный программный продукт строится на строгой дисциплине написания кода, внимании к деталям (таким как обработка ошибок) и тщательном тестировании каждого модуля. Полученные навыки будут использованы в дальнейшем при проектировании более сложных автоматизированных информационных систем.

Список используемых источников

1. Керниган, Б. Ритчи, Д. Язык программирования С / Б. Керниган, Д. Ритчи. – 1985. – 279 с.

2. Прата, С. Язык программирования С++ / С. Прата. – 2019. – 960 с. – ISBN 978-5-4461-1135-2.

3. Иванова, Г. С. Программирование на ассемблере ПЭВМ : методические указания / Г. С. Иванова. – 1999. – 32 с.

4. Финогенов, К. Г. Основы языка Ассемблера / К. Г. Финогенов. – 1999. – 288 с. – ISBN 5-256-01451-X.

5. Тюгашев, А. А. Языки программирования: учебное пособие / А. А. Тюгашев. – 2018. – 136 с. – ISBN 978-5-7782-3520-5.