Министерство образования Республики Беларусь
Учреждение образования «Белорусский государственный университет информатики и радиоэлектроники»

Факультет информационных технологий и управления

Кафедра вычислительных методов и программирования

Дисциплина: Основы алгоритмизации и программирования

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
к курсовой работе
на тему:

«СИСТЕМА ПЛАНИРОВАНИЯ БЮДЖЕТА РАЗРАБОТКИ ПРОГРАММНОГО ПРОЕКТА»

Выполнил: гр.421701 Неводниченко М.В.

Проверил: Панасик А.А.

Минск 2025

РЕФЕРАТ

При обработке баз данных часто применяются массивы структур. Обычно база данных накапливается и хранится на диске в файле. К ней часто приходится обращаться, обновлять, перегруппировывать.

Объектом изучения являются алгоритмы поиска и сортировки для массивов, так как наиболее частыми операциями при работе с базами данных являются «поиск» и «сортировка». При этом алгоритмы решения этих задач существенно зависят от того, организованы структуры в массивы или размещены на диске.

В процессе написания курсовой работы использовались следующие методы исследования: анализ литературы и интернет-ресурсов по выбранной теме, сравнительный анализ.

В данной курсовой работе детально изложен алгоритм работы и способ использования программы для планирования бюджета разработки программных проектов, включая описание алгоритмов сортировки, поиска, анализа бюджета и статистики по проектам.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В условиях роста сложности программных проектов и увеличения затрат на их разработку эффективное планирование бюджета становится ключевым фактором успеха. Системы планирования бюджета позволяют контролировать финансовые ресурсы, отслеживать сроки, учитывать количество участников и категоризировать проекты для оптимизации процессов. Надёжная система планирования бюджета помогает минимизировать риски перерасхода средств и повысить эффективность управления проектами.

Данный курсовой проект представляет программу для планирования бюджета разработки программных проектов. Программа позволяет хранить информацию о проектах, выполнять поиск, сортировку, анализировать проекты с недостаточным бюджетом и предоставлять статистику по категориям.

Целью курсового проекта является разработка программы, автоматизирующей планирование бюджета программных проектов и предоставляющей аналитические инструменты для управления. Задачи:

- Изучить методы планирования бюджета программных проектов;
- Проанализировать существующие системы управления проектами;
- Разработать функциональную модель программы;
- Реализовать код программы на языке C++;
- Протестировать программу и устранить ошибки.

Исследование имеет практическую значимость, так как способствует оптимизации бюджетного планирования и повышению эффективности управления проектами.

1 СТРУКТУРЫ И ФАЙЛЫ

1.1 Структуры данных

Структуры данных являются неотъемлемой частью разработки алгоритмов, предназначенных для решения сложных задач, и играют важную роль в производительности программного обеспечения. Они предоставляют эффективные методы организации данных, что позволяет оптимизировать использование доступной памяти компьютера. Также структуры очень удобны при сортировках и поисках, так как организация структуры позволяет искать или сортировать по одному или нескольким полям.

Рассмотрим структуры, которые организованы в данной курсовой работе. Основной единицей, с которой происходит работа в коде, является проект. Для этого создается структура `Project`, которая включает несколько полей для характеристик проектов.

Поле `name` используется для хранения названия проекта. Типом данных является строковый массив (`char[50]`), так как название представлено последовательностью символов, а размер 50 символов позволяет хранить достаточно длинные названия.

Поле `startDate` хранит дату начала проекта в формате ГГГГ-ММ-ДД. Типом данных выбран строковый массив (`char[11]`) для удобства работы с текстовым форматом даты, обеспечивая простоту ввода и проверки.

Поле `endDate` предназначено для хранения даты завершения проекта в формате ГГГГ-ММ-ДД. Тип данных строковый массив (`char[11]`), что соответствует текстовому представлению даты и упрощает её обработку.

Поле `budget` хранит бюджет проекта. Тип данных `double`, так как бюджет может быть дробным числом, а `double` обеспечивает необходимую точность для финансовых данных.

Поле `teamSize` используется для хранения количества участников команды. Тип данных `int`, поскольку количество людей является целым числом.

Поле `category` хранит категорию проекта (например, "IT", "Development"). Типом данных является строковый массив (`char[50]`), что позволяет хранить текстовые названия категорий.

1.2 Файлы

Файл представляет собой совокупность данных, организованных и хранящихся на внешнем носителе, которые программы рассматривают как единое целое в процессе обработки. В рамках данной курсовой работы были задействованы два типа файлов: бинарный и текстовый, каждый из которых выполнял определённые функции в системе управления проектами.

Для работы с обоими типами файлов использовалась стандартная библиотека C++ `<fstream>`, обеспечивающая удобные средства для чтения, записи и управления файлами.

Бинарный файл, названный projects.bin, служит для хранения информации о проектах. Данные в него записываются посимвольно в двоичном формате, представляя собой последовательность нулей и единиц, что позволяет компактно и эффективно хранить структурированные записи, такие как объекты структуры `Project`.

Текстовый файл, получивший название result.txt, используется для формирования отчетов о действиях пользователя, таких как результаты поиска, сортировки или статистика по проектам. Запись в текстовый файл осуществляется посимвольно с использованием кодировки ASCII, что обеспечивает читаемость данных для человека и совместимость с текстовыми редакторами.

Оба файла играют ключевую роль в программе, обеспечивая хранение данных и предоставление пользователю удобных отчетов о выполненных операциях.

2 АЛГОРИТМЫ СОРТИРОВКИ

Сортировка представляет собой процесс упорядочивания элементов по определенному критерию. Этот процесс может включать последовательное распределение элементов по возрастанию или убыванию значений, либо разбиение на группы в соответствии с выбранным критерием. Алгоритм сортировки – это набор инструкций, разработанных для систематического переупорядочивания элементов в списке с целью достижения желаемого порядка.

В случае, если элементы списка представляют собой структуры с несколькими полями данных, сортировка обычно выполняется по одному из этих полей, называемому ключом сортировки.

Важно отметить, что существует множество алгоритмов сортировки, каждый из которых имеет свои особенности и подходит для разных целей. Они оцениваются двум главным критериям:

− Память (эффективность использования памяти) – требует ли алгоритм использования дополнительной памяти для временного хранения данных.

− Время (вычислительная сложность) – параметр, определяющий, насколько быстро работает алгоритм;

Кроме того, каждая сортировка характеризуется такими свойствами, как устойчивость (меняются ли элементы с одинаковыми ключами), естественность поведения (эффективность при работе с уже ранее упорядоченными данными), сфера применения алгоритма.

Рассмотрим алгоритмы сортировки, использованные в данной курсовой работе.

2.1 Сортировка вставками

Сортировка вставками (Insertion sort) – алгоритм сортировки, в котором элементы входного массива просматриваются по одному, и каждый следующий просматриваемый элемент размещается сразу на подходящее место относительно ранее отсортированных элементов.

Алгоритм сортировки вставками:

1 Начинаем сортировку с первого элемента, считая его уже отсортированным;

2 Берем следующий элемент и вставляем его на правильное место в уже отсортированной части списка. Это достигается при помощи поэлементного сравнивания, то есть движения нового элемента левее до тех пор, пока элемент слева от данного больше;

3 Повторяем это действие для оставшихся элементов списка, переставляя каждый элемент на правильное место;

4 Сортировка продолжается, пока все элементы массива не окажутся на своем месте.

Стоит отметить, что данный алгоритм хуже всего проявляет себя в случае, когда массив отсортирован обратно нужному (например, если нам нужно отсортировать массив по убыванию, а он отсортирован по возрастанию). Среди же преимуществ этого метода сортировки главным является незначительное использование вспомогательной памяти (только для хранения одной переменной при обмене элементов местами). Данный алгоритм лучше всего использовать, когда массив уже частично отсортирован и размер его небольшой.

В этой курсовой работе сортировка вставками использовалась для сортировки проектов по полю `startDate`.

2.2 Сортировка выбором

Сортировка выбором — это простой алгоритм сортировки, который на каждом шаге находит минимальный (или максимальный, в зависимости от порядка сортировки) элемент в неотсортированной части массива и помещает его в начало этой части. Алгоритм делит массив на две части: отсортированную (в начале) и неотсортированную (в конце), постепенно уменьшая размер неотсортированной части. Алгоритм сортировки выбором:

1 Находим минимальный элемент в неотсортированной части массива.

2 Меняем местами минимальный элемент с первым элементом неотсортированной части (тем самым расширяя отсортированную часть).

3 Повторяем шаги 1–2 для оставшейся неотсортированной части массива, пока весь массив не будет отсортирован.

Особенности и характеристики:

Временная сложность: $O(n^2)$ во всех случаях (лучшем, среднем и худшем), так как алгоритм всегда выполняет полный перебор неотсортированной части для поиска минимума.

Пространственная сложность: O(1), так как сортировка выполняется "на месте" (in-place), используя только одну дополнительную переменную для обмена элементов.

Худший случай: как и сортировка вставками, сортировка выбором плохо проявляет себя на больших массивах или массивах, отсортированных в обратном порядке, так как требует выполнения всех сравнений и перестановок.

2.3 Быстрая сортировка

Быстрая сортировка (QuickSort) считается одним из наиболее эффективных алгоритмов сортировки. Она разбивает массив на меньшие подмассивы до тех пор, пока она не станут размером 1 или 2, сортирует их, а затем обратно объединяет в один. За счет этого намного быстрее можно отсортировать даже очень большие списки, если был правильно выбран опорный элемент (в литературе он также часто называется пивотом).

Алгоритм:

1 Индексы l и r инициализируются минимальным и максимальным значениями для массива;

2 Определяется индекс опорного элемента (например, серединный или последний элемент массива);

3 l увеличивается до m, пока не превысит опорный элемент;

4 r уменьшается до m, пока не будет меньше опорного элемента;

5 Если r и l становятся равными, операция завершена, оба индекса указывают на опорный элемент;

6 Если l меньше r, обмениваются значениями и продолжается операция с новыми l и r. Причем, если граница (l или r) достигает опорного элемента, то при обмене значение m изменяется на элемент с индексом r или l соответственно;

7 Рекурсивно сортируем подмассивы, расположенные слева и справа от опорного элемента.

8 Базовым случаем рекурсии являются подмассивы, содержащие один или два элемента. В первом случае возвращаем подмассив без изменений, во втором случае производим перестановку элементов, если это необходимо.

Таким образом, все такие подмассивы уже упорядочены в процессе разделения.

В этой курсовой работе сортировка вставками использовалась для сортировки товаров по полю `budget`.

3 АЛГОРИТМЫ ПОИСКА

Поиском называется процесс нахождения нужной информации в структуре данных, такой как массив, дерево или список. Он направлен на обнаружение элемента с заданным значением или установление факта его наличия либо отсутствия. Поиск — одна из ключевых операций в программировании, влияющая на производительность программ, особенно при обработке больших объёмов данных. Эффективность алгоритма поиска определяет скорость выполнения задач, что критично для пользовательских запросов и анализа информации.

Существует множество поисковых алгоритмов, различающихся по организации, требованиям к данным и производительности. Выбор алгоритма зависит от размера данных, их упорядоченности и частоты поисков. В программировании применяются линейный, бинарный, интерполяционный, хэш-поиск и другие методы, каждый с уникальными плюсами и минусами, подходящими для конкретных задач.

В курсовой работе, посвящённой системе управления проектами, реализованы линейный и бинарный поиски. Линейный поиск, или последовательный, проверяет каждый элемент массива, пока не найдёт искомое или не завершит структуру. В программе он используется для поиска проекта по названию, что удобно для неупорядоченных данных. Однако он медлен на больших массивах, так как может проверять все элементы.

Бинарный поиск требует отсортированных данных и делит массив пополам на каждом шаге, быстро сужая область поиска. В программе он применяется для поиска по бюджету после сортировки. Благодаря логарифмической сложности, бинарный поиск эффективен для больших данных. Эти алгоритмы дополняют друг друга: линейный — для простоты, бинарный — для скорости. Их реализация обеспечивает обработку запросов и создание отчётов в `result.txt`, демонстрируя важность выбора метода поиска для конкретной задачи.

3.1 Линейный поиск

Линейный поиск — это простой и интуитивно понятный алгоритм, предназначенный для нахождения элемента в наборе данных путём последовательной проверки каждого элемента. Он особенно полезен в ситуациях, когда данные не отсортированы или их структура не позволяет применять более сложные методы поиска. В системе управления проектами, например, линейный поиск используется для нахождения проекта по его названию, что делает его удобным инструментом для работы с неупорядоченными массивами.

Процесс линейного поиска начинается с первого элемента массива. Каждый элемент сравнивается с искомым значением, например, названием проекта. Если совпадение найдено, алгоритм возвращает индекс этого элемента, указывая его позицию в массиве. Если текущий элемент не

соответствует искомому, проверка продолжается со следующим элементом. Этот процесс повторяется, пока не будет найден нужный элемент или не будут проверены все элементы массива. В случае, если искомое значение отсутствует, алгоритм возвращает специальный индикатор, обычно -1, сигнализируя, что элемент не найден.

Преимущество линейного поиска заключается в его простоте и универсальности. Он не требует предварительной сортировки данных и легко реализуется даже в небольших программах. Однако его эффективность ограничена, особенно на больших наборах данных, поскольку в худшем случае алгоритм проверяет каждый элемент, что делает его менее подходящим для массивов с тысячами записей. Несмотря на это, для небольших объёмов данных, таких как список проектов в программе, линейный поиск остаётся надёжным и практичным решением. Его понятная логика и минимальные требования к структуре данных обеспечивают удобство использования в задачах, где скорость не является критическим фактором, а важна простота и корректность результата.

В этой курсовой работе сортировка вставками использовалась для поиска товаров по полю `name`, представляющее собой массив `char`.

3.2 Бинарный поиск

Бинарный поиск — это эффективный алгоритм, предназначенный для нахождения элемента в отсортированном наборе данных, который значительно превосходит линейный поиск по скорости на больших массивах. Его основная идея заключается в последовательном делении массива пополам, что позволяет быстро сузить область поиска. В контексте программы управления проектами бинарный поиск используется, например, для нахождения проекта по бюджету, предполагая, что массив проектов предварительно отсортирован по этому критерию.

Процесс бинарного поиска начинается с определения границ массива: левой (первый элемент) и правой (последний элемент). Алгоритм вычисляет средний элемент массива, сравнивая его значение с искомым. Если средний элемент совпадает с искомым значением, поиск завершается, и возвращается индекс этого элемента. Если искомое значение меньше среднего, поиск продолжается в левой половине массива, а правая отбрасывается. Если же искомое значение больше, алгоритм переходит к правой половине, игнорируя левую. Этот процесс повторяется, каждый раз уменьшая область поиска вдвое, пока не будет найден элемент или не станет ясно, что его нет в массиве. В последнем случае возвращается индикатор, например, -1, либо индекс ближайшего элемента, как в случае программы, где может быть возвращён проект с бюджетом, наиболее близким к искомому.

Ключевое требование бинарного поиска — отсортированность массива. В программе перед выполнением поиска по бюджету проекты сортируются с помощью быстрой сортировки `quickSortByBudget`, что обеспечивает необходимый порядок. Это предварительное действие увеличивает временные

затраты, но они окупаются при многократных поисках.

Эффективность бинарного поиска обусловлена его логарифмической сложностью O(log n), где n — количество элементов. Это означает, что даже для массива из миллиона элементов потребуется всего около 20 сравнений, в отличие от линейного поиска, который может проверить все элементы.

Преимущества бинарного поиска делают его идеальным для больших наборов данных, где скорость критически важна. Однако он имеет ограничения: необходимость сортировки данных и применимость только к массивам или структурам с прямым доступом к элементам. В программе управления проектами бинарный поиск реализован для поиска по бюджету, где он возвращает либо точное совпадение, либо ближайший меньший бюджет, что удобно для анализа проектов.

Простота логики бинарного поиска сочетается с его высокой производительностью, что делает его стандартом для задач, требующих быстрого доступа к данным. В сравнении с линейным поиском, который проверяет элементы последовательно, бинарный поиск демонстрирует значительное преимущество, особенно когда массив велик. В программе он интегрирован с функцией записи результатов в `result.txt`, обеспечивая пользователю наглядный отчет о найденном проекте. Таким образом, бинарный поиск является мощным инструментом, оптимально подходящим для задач, где данные можно предварительно упорядочить, обеспечивая баланс между скоростью и точностью.

4 ПОЛЬЗОВАТЕЛЬСКИЕ ФУНКЦИИ

В рамках данного курсового проекта было разработано 15 функций, каждая из которых выполняет специализированные задачи для реализации системы управления проектами. Эти функции обеспечивают полный цикл операций, включая создание, добавление, редактирование, удаление, поиск, сортировку и анализ данных о проектах, а также генерацию подробных отчетов. Реализация программного кода представлена в приложении А, демонстрируя логическую структуру и взаимосвязь компонентов системы. Функции разделены на основные, отвечающие за ключевые операции с данными, и вспомогательные, обеспечивающие поддержку основных процессов.

4.1 Основные функции

Функция `readFromBinaryFile` предназначена для считывания записей из бинарного файла `projects.bin` и возврата их количества. Она принимает три аргумента: имя файла (`filename`), массив для хранения данных (`projects`) и максимальный размер массива (`maxSize`). Функция формирует полный путь к файлу, конкатенируя константу `BASE_PATH` с именем файла, и открывает его в бинарном режиме с помощью `ifstream`. Если файл не открывается, выводится сообщение об ошибке, и возвращается 0. Данные считываются по одной структуре `Project` (содержащей название, даты, бюджет, размер команды и категорию) в цикле, пока не достигнут конец файла или лимит массива. Каждая успешная операция чтения увеличивает счётчик записей. После завершения файл закрывается, а функция возвращает количество считанных записей. Эта функция критически важна для инициализации данных перед их обработкой, обеспечивая доступ к актуальной информации для последующих операций, таких как поиск, сортировка или генерация отчетов.

Функция `initializeBinaryFile` создаёт бинарный файл `projects.bin` с 10 тестовыми записями структуры `Project`. Она принимает имя файла (`filename`) как аргумент. Функция формирует полный путь, открывает файл в бинарном режиме с помощью `ofstream` и проверяет успешность открытия. В случае ошибки (например, из-за отсутствия прав доступа) выводится сообщение, и выполнение завершается. В файл записываются предопределённые записи, каждая из которых включает название (например, "Project A"), даты начала и завершения (в формате ГГГГ-ММ-ДД), бюджет (типа `double`), размер команды (типа `int`) и категорию (например, "IT"). После записи данных файл закрывается, и пользователю отображается сообщение об успешном создании файла с указанием пути. Эта функция используется для начальной настройки системы, предоставляя тестовый набор данных для отладки, тестирования и демонстрации функциональности программы.

Функция `addProjectToFile` позволяет добавлять новую запись о проекте в конец бинарного файла `projects.bin`. Она принимает массив проектов (`projects`), текущую длину массива (`projectCount`), максимальный размер массива (`maxSize`) и имя файла (`filename`). Если массив заполнен, выводится сообщение о достижении лимита. Пользователь вводит данные: название, даты начала и завершения, бюджет, количество участников и категорию. Ввод проверяется на корректность: название и категория не должны быть пустыми, даты — в формате ГГГГ-ММ-ДД с годом ≥ 2000, дата завершения не раньше даты начала, бюджет и размер команды — положительные числа. Новая запись добавляется в массив и записывается в файл в бинарном формате с помощью `ofstream` в режиме ios::app. При успешной записи выводится сообщение с названием проекта, иначе — сообщение об ошибке. Текстовый файл `result.txt` не изменяется, что сохраняет его для отчетов.

Функция `editProjectInFile` предоставляет возможность редактирования существующего проекта в projects.bin по его названию. Она принимает массив проектов (`projects`), их количество (`projectCount`) и имя файла (`filename`). Пользователь вводит название проекта для поиска, которое проверяется на непустоту. С помощью функции `linearSearchByName` определяется индекс проекта. Если проект не найден, выводится сообщение об ошибке. Иначе отображаются текущие данные проекта, и пользователь вводит новые значения для каждого поля (название, даты, бюджет, команда, категория), при этом можно оставить текущее значение, нажав Enter. Ввод проверяется: даты должны быть валидными и хронологически корректными, бюджет и команда — положительными или нулевыми (для сохранения текущих значений). Функция перезаписывает весь файл с обновлённым массивом, используя `ofstream` в бинарном режиме. При успехе выводится сообщение об успешной редакции, иначе — об ошибке. Текстовый файл `result.txt` не затрагивается.

Функция `deleteProjectFromFile` удаляет проект из бинарного файла по названию. Она принимает массив проектов (`projects`), текущую длину массива (`projectCount`) и имя файла (`filename`). Пользователь вводит название проекта, которое проверяется на непустоту. С помощью `linearSearchByName` находится индекс проекта. Если проект не найден, выводится сообщение об ошибке. Иначе элементы массива сдвигаются, исключая найденный проект, а счётчик записей уменьшается. Файл перезаписывается с обновлённым массивом через `ofstream` в бинарном режиме. При успешном удалении выводится сообщение с названием удалённого проекта, при ошибке — соответствующее уведомление. Текстовый файл `result.txt` остаётся неизменным.

Функция `viewResultsFile` отображает содержимое бинарного файла projects.bin в консоли и записывает его в текстовый файл `result.txt`. Она принимает имена бинарного (`filename`) и текстового (`textFile`) файлов. Формируются полные пути, и бинарный файл открывается для чтения через

ifstream. Если файл не открывается, выводится ошибка. Текстовый файл открывается с флагом ios::trunc для перезаписи. Каждая запись считывается как структура Project и выводится построчно в консоль и result.txt с полями: название, даты, бюджет, размер команды, категория. Если файл пуст, записывается сообщение о пустом файле. После завершения файлы закрываются, и пользователю сообщается о записи в result.txt. Функция удобна для общего обзора всех проектов.

Функция linearSearchByName реализует линейный поиск проекта по названию. Она принимает массив проектов (projects), его размер (size) и искомое название (searchName). Функция последовательно сравнивает название каждой записи с искомым. При совпадении возвращается индекс проекта. Если совпадений нет, возвращается -1. В программе она вызывается в editProjectInFile, deleteProjectFromFile и в меню (опция 6), где пользователь вводит название, а результат (данные проекта или сообщение об отсутствии) выводится в консоль и записывается в result.txt с помощью writeProjectToFileAndConsole. Функция проста и эффективна для небольших массивов.

Функция binarySearchByBudget выполняет бинарный поиск проекта по бюджету. Она принимает массив проектов (projects), его размер (size) и искомый бюджет (searchBudget). Перед поиском массив сортируется с помощью quickSortByBudget. Функция определяет границы поиска (левая — 0, правая — size-1) и вычисляет средний элемент. Если бюджет среднего элемента совпадает с искомым, возвращается его индекс. Если искомый бюджет меньше, поиск продолжается в левой половине, иначе — в правой. Процесс повторяется, пока границы не сойдутся. Если точного совпадения нет, возвращается индекс ближайшего меньшего бюджета или -1, если такого нет. Результат выводится в консоль и result.txt. Функция эффективна для больших данных благодаря логарифмической сложности.

Функция quickSortByBudget сортирует массив проектов по бюджету с помощью быстрой сортировки. Она принимает массив (projects), индексы левой (low) и правой (high) границ. Опорный элемент выбирается как бюджет последнего элемента. Массив делится так, что элементы с меньшим бюджетом оказываются слева, с большим — справа. Процесс рекурсивно повторяется для подмассивов. Отсортированные данные записываются в result.txt и выводятся в консоль через writeProjectToFileAndConsole. Функция используется для подготовки данных к бинарному поиску и статистике.

Функция selectionSortByTeamSize сортирует проекты по количеству участников с помощью сортировки выбором. Она принимает массив (projects) и его размер (size). На каждой итерации находится элемент с максимальным размером команды, который меняется местами с текущим. Сортировка выполняется по убыванию. Результаты записываются в result.txt и отображаются в консоли. Функция полезна для анализа проектов по численности команды.

Функция `insertionSortByStartDate` сортирует проекты по дате начала с помощью сортировки вставками. Она принимает массив (`projects`) и его размер (`size`). Каждый элемент вставляется в правильную позицию в уже отсортированной части массива, сравнивая даты с помощью `dateLessThan`. Отсортированные данные записываются в `result.txt` и выводятся в консоль. Функция обеспечивает хронологический порядок.

Функция `generateProjectStatistics` формирует статистический отчёт по категориям проектов. Она принимает массив проектов (`projects`), их количество (`projectCount`) и имя текстового файла (`textFile`). Массив сортируется по бюджету с помощью `quickSortByBudget`. Собираются уникальные категории (до 10) в массив `categories`. Для каждой категории выводится список проектов, отсортированных по бюджету, а также самый дешевый, дорогой, короткий и долгий проекты, определяемые по бюджету и длительности (`dateDifference`). Результаты записываются в `result.txt` с флагом `ios::trunc` и отображаются в консоли через `writeProjectToFileAndConsole`. Если файл не открывается, выводится ошибка. Функция предоставляет глубокий анализ данных.

4.2 Вспомогательные функции

Функция `writeProjectToFileAndConsole` записывает данные о проекте в текстовый файл и консоль. Она принимает поток вывода (`ofstream`) и структуру `Project`. Выводятся все поля: название, даты, бюджет, размер команды, категория, разделённые строкой-разделителем. Функция используется в функциях поиска, сортировки и статистики для единообразного форматирования отчетов.

Функция `writeProjectToConsole` выводит данные проекта только в консоль, принимая структуру `Project`. Формат вывода аналогичен `writeProjectToFileAndConsole`, но без записи в файл. Применяется в viewResultsFile для отображения данных без изменения `result.txt`.

Функция `dateDifference` вычисляет разницу между датами начала и конца проекта в днях, основываясь на годах. Она принимает строки дат (start, end) и возвращает целое число, умножая разницу лет на 365. Используется в `generateProjectStatistics` для определения длительности проектов, что помогает выявить самые короткие и долгие проекты.

Функция `isValidDate` проверяет корректность даты в формате ГГГГ-ММ-ДД. Она принимает строку (date) и возвращает true, если длина строки 10 символов, формат соответствует шаблону, год ≥ 2000, месяц 1–12, а день допустим для месяца (например, ≤ 30 для апреля). Используется в `addProjectToFile` и `editProjectInFile` для валидации ввода.

Функция `dateLessOrEqual` сравнивает две даты, возвращая true, если первая дата не позже второй. Она посимвольно сравнивает строки дат, применяясь в `addProjectToFile` и `editProjectInFile` для проверки хронологической корректности.

Функция `dateLessThan` определяет, является ли первая дата строго раньше второй, сравнивая их посимвольно. Используется в `insertionSortByStartDate` для сортировки по датам.

Функция `isNonEmptyString` проверяет, является ли строка непустой, возвращая true, если первый символ не нулевой. Применяется в `addProjectToFile`, `editProjectInFile` и других функциях для валидации текстовых полей.

Функция `strEqual` сравнивает две строки на равенство, возвращая true, если они идентичны. Используется в `linearSearchByName` и `generateProjectStatistics` для сравнения названий и категорий.

5 ОПИСАНИЕ РАБОТЫ ПРОГРАММЫ

Программа, разработанная в рамках курсовой работы, представляет собой систему управления проектами на языке C++. Она позволяет хранить, обрабатывать и анализировать данные о проектах, включая название, даты начала и завершения, бюджет, размер команды и категорию. Данные сохраняются в бинарном файле `projects.bin`, а отчеты записываются в текстовый файл `result.txt`. Программа предоставляет интерактивное меню с 12 опциями (13 это выход) для выполнения операций: создание файла, просмотр, добавление, редактирование, удаление, поиск, сортировка и анализ данных.

При запуске программы перед пользователем отображается основное меню программы, состоящее из 13 пунктов (рисунок 1).

```
Меню:
1. Создание нового бинарного файла
2. Просмотр записей из бинарного файла
3. Добавление записей в бинарный файл
4. Редактирование записей в бинарном файле
5. Удаление записей из бинарного файла
6. Линейный поиск по названию проекта
7. Бинарный поиск по общему бюджету
8. Быстрая сортировка по бюджету
9. Сортировка выбором по количеству участников
10. Сортировка вставками по дате начала разработки
11. Определение проектов с недостаточным бюджетом
12. Статистика по проектам
0. Выход
Выберите действие:
```

Рисунок 1 – Пример вывода меню

Далее продемонстрируем пример работы каждого из выбора на конкретном примере.

При выборе действия 1 программа вызывает функцию `initializeBinaryFile`, которая создаёт новый бинарный файл с 10 тестовыми записями. Если файл не существует, он создаётся в бинарном режиме записи, заполняется предопределёнными данными (название, даты, бюджет, размер команды, категория), и выводится сообщение "Тестовый файл ... успешно создан". В случае ошибки создания файла, например, из-за отсутствия прав доступа, отображается уведомление "Ошибка при создании файла". Если файл успешно загружается в программу, выводится "Бинарный файл успешно создан и загружен". Если файл уже существует, он перезаписывается без дополнительного уведомления о существовании.

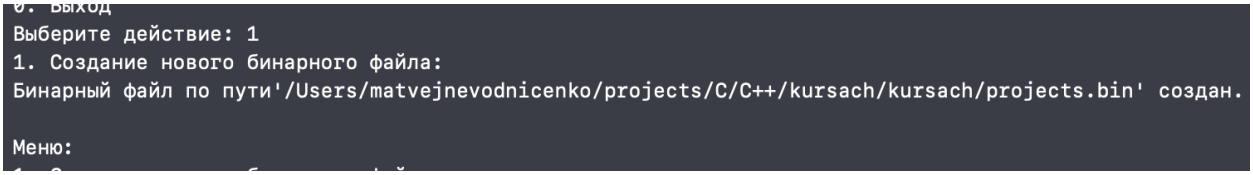

Рисунок 2 – Вывод первой опции

При выборе действия 2 программа активирует функцию `viewResultsFile`, которая считывает все записи из бинарного файла и отображает их в консоли, одновременно записывая в `result.txt`. Каждая запись представлена в структурированном виде с указанием названия, дат начала и завершения, бюджета, размера команды и категории, разделённых строками-разделителями. Если файл пуст, в консоль и отчетный файл выводится сообщение "Файл пуст". В случае, если файл не удаётся открыть (например, из-за его отсутствия), программа отображает уведомление "Ошибка при открытии файла", информируя пользователя о невозможности выполнения операции просмотра.

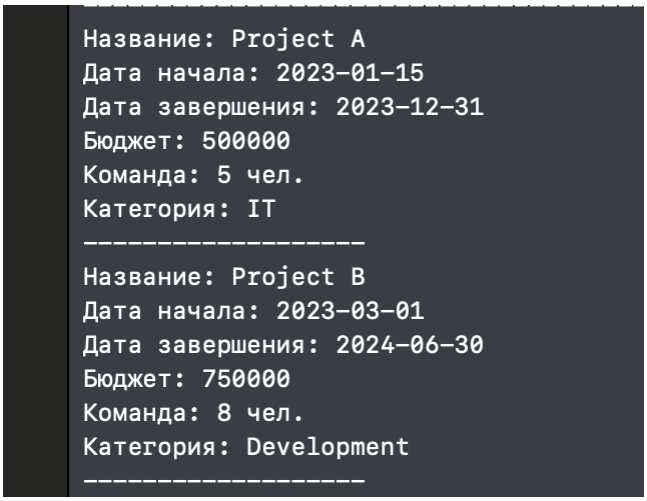

Рисунок 3 – Пример записи в бинарном файле

При выборе действия 3 программа выполняет функцию `addProjectToFile`, запрашивая у пользователя данные нового проекта в следующем порядке: название, дата начала, дата завершения, бюджет, количество участников, категория. Ввод проверяется: название и категория не должны быть пустыми, даты — в формате ГГГГ-ММ-ДД с годом ≥ 2000, дата завершения не раньше начала, бюджет и размер команды — положительные числа. После успешного ввода данные добавляются в массив и записываются в конец бинарного файла, а пользователю отображается сообщение "Проект '[название]' успешно добавлен". Если добавление не удалось (например, массив заполнен), выводится уведомление "Ошибка при добавлении данных в файл". Текстовый файл `result.txt` остаётся неизменным.

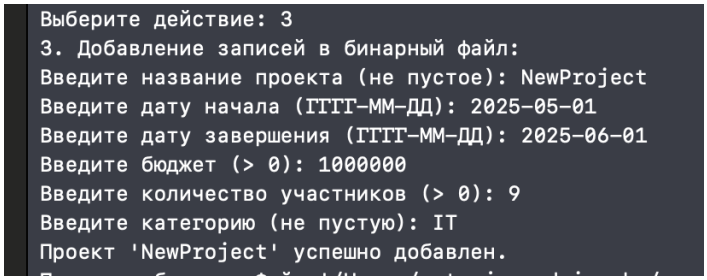

Рисунок 4 – Пример добавления проекта

При выборе действия 4 программа вызывает функцию `editProjectInFile`, которая позволяет изменить данные проекта по его названию. Пользователь вводит название проекта, которое проверяется на непустоту. Программа выполняет линейный поиск для нахождения проекта. Если проект найден, отображаются его текущие данные, и пользователь вводит новые значения для каждого поля (название, даты, бюджет, команда, категория), оставляя старые при пустом вводе. Даты проверяются на формат и хронологическую корректность, бюджет и команда — на положительность. Файл перезаписывается с обновлённым массивом, и выводится сообщение "Проект '[название]' успешно отредактирован". Если проект не найден или произошла ошибка, отображается уведомление "Ошибка при редактировании данных". Текстовый файл не изменяется

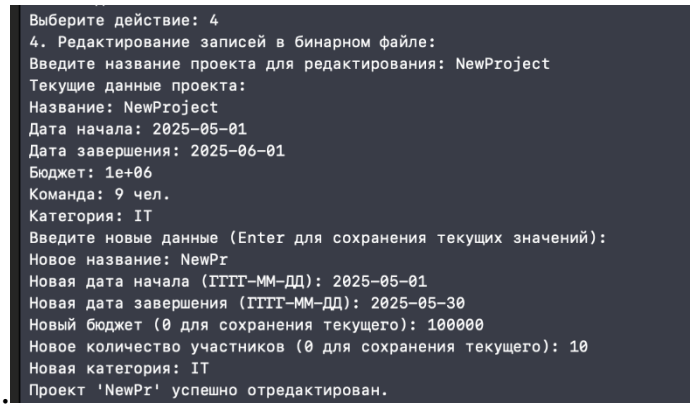

Рисунок 5 – Пример редактирования проекта

При выборе действия 5 программа выполняет функцию `deleteProjectFromFile`, которая удаляет проект по указанному названию. Пользователь вводит название, которое проверяется на непустоту. Программа использует линейный поиск для нахождения проекта. Если проект найден, он исключается из массива путём сдвига элементов, а бинарный файл перезаписывается с обновлённым массивом. Пользователю отображается сообщение "Проект '[название]' успешно удален". Если проект не найден или произошла ошибка записи, выводится уведомление "Ошибка при удалении данных". Текстовый файл остаётся неизменным.

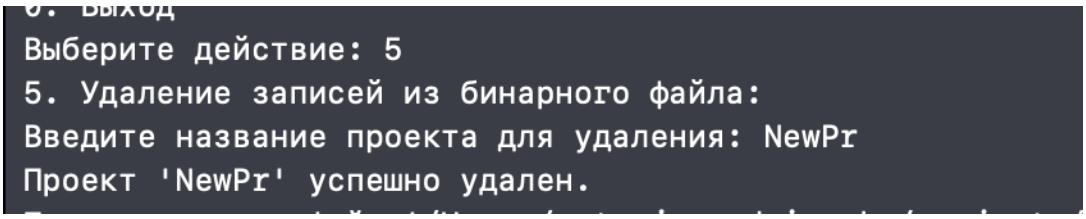

Рисунок 6 – Пример удаления проекта

При выборе действия 6 программа вызывает функцию `linearSearchByName`, которая выполняет линейный поиск проекта по названию. Пользователь вводит название, которое проверяется на непустоту. Программа последовательно сравнивает название с каждым проектом в массиве. При совпадении данные проекта (название, даты, бюджет, команда, категория) выводятся в консоль и записываются в отчетный файл. Если проект не найден, отображается сообщение "Проект не найден". В случае ошибки доступа к данным выводится уведомление "Ошибка при выполнении поиска".

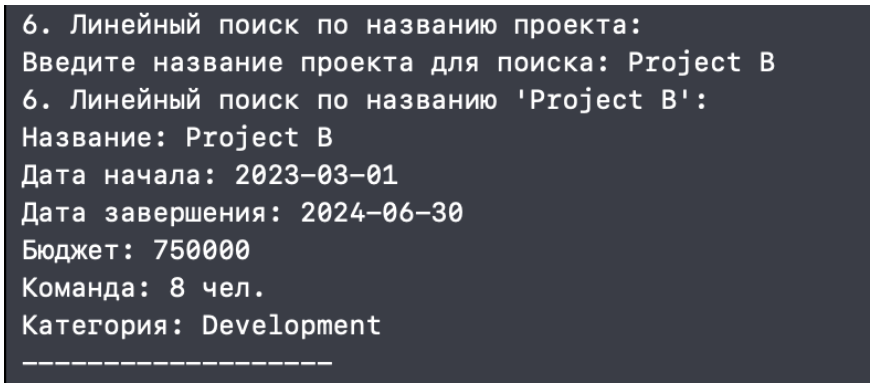

Рисунок 7 – Линейный поиск

При выборе действия 7 программа активирует функцию `binarySearchByBudget`, которая выполняет бинарный поиск проекта по бюджету. Массив предварительно сортируется по бюджету с помощью `quickSortByBudget`. Пользователь вводит бюджет, который проверяется на положительность. Программа делит массив пополам, сравнивая средний элемент с искомым бюджетом, и продолжает поиск в нужной половине. При нахождении точного совпадения или ближайшего меньшего бюджета данные проекта выводятся в консоль и записываются в отчет. Если подходящий проект не найден, отображается сообщение "Проект с указанным бюджетом не найден". При ошибке доступа к данным выводится "Ошибка при выполнении поиска".

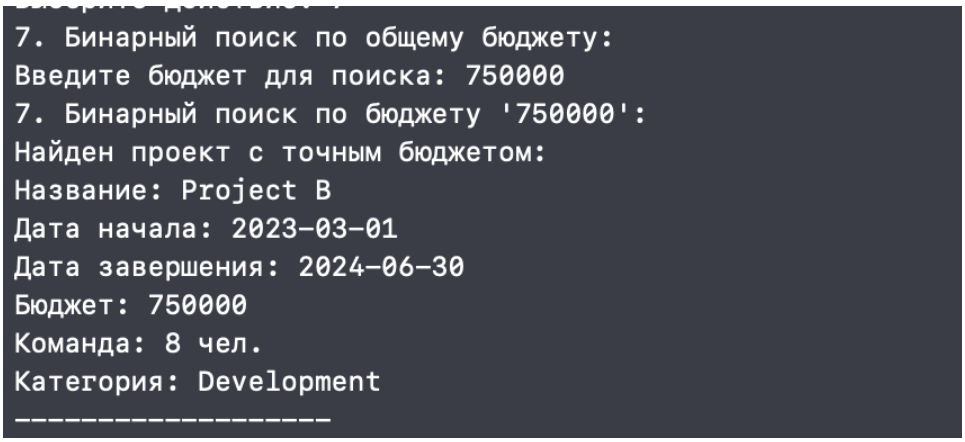

Рисунок 8 – Бинарный поиск

При выборе действия 8 программа вызывает функцию `quickSortByBudget`, которая сортирует проекты по возрастанию бюджета с помощью быстрой сортировки. Массив делится рекурсивно с использованием опорного элемента. Отсортированный список проектов выводится в консоль и записывается в отчетный файл в структурированном виде с полями: название, даты, бюджет, команда, категория. Если массив пуст или файл не открывается, выводится сообщение "Ошибка при выполнении сортировки" или "Файл пуст". При успешной сортировке отображается уведомление о записи результата. В последующих примерах сортировок будут отображаться только первые записи.

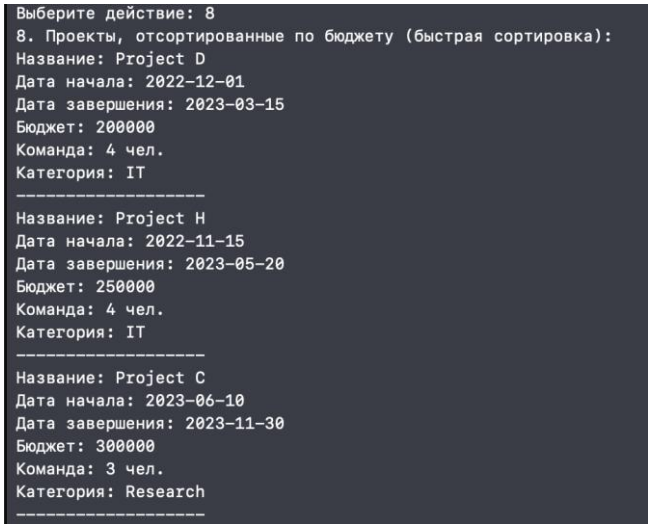

Рисунок 9 – Быстрая сортировка по бюджету

При выборе действия 9 программа выполняет функцию `selectionSortByTeamSize`, которая сортирует проекты по убыванию размера команды с помощью сортировки выбором. На каждой итерации находится проект с максимальным количеством участников.

Отсортированный список выводится в консоль и записывается в отчет. Если массив пуст или доступ к данным невозможен, отображается сообщение "Ошибка при выполнении сортировки" или "Файл пуст". При успешной сортировке выводится уведомление о записи результата.

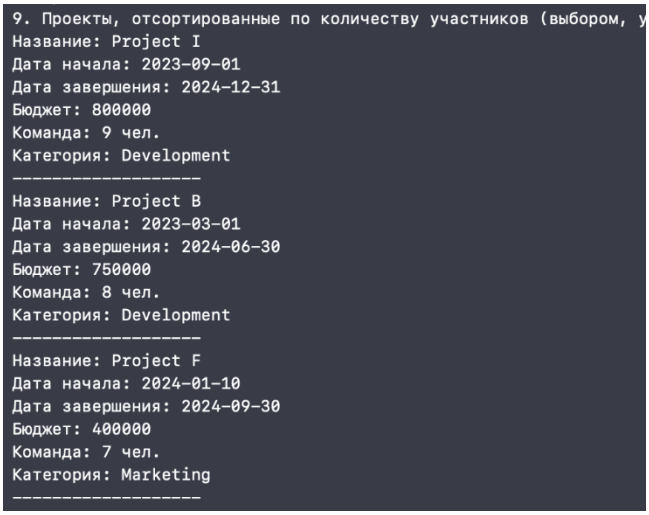

Рисунок 10 – Сортировка выбором по кол-ву участников

При выборе действия 10 программа вызывает функцию `insertionSortByStartDate`, которая сортирует проекты по возрастанию даты начала с помощью сортировки вставками. Каждый проект вставляется в правильную позицию в отсортированной части массива с использованием функции сравнения дат. Отсортированный список выводится в консоль и записывается в отчет. Если массив пуст или файл не открывается, отображается сообщение "Ошибка при выполнении сортировки" или "Файл пуст". При успешной сортировке выводится уведомление о записи результата.

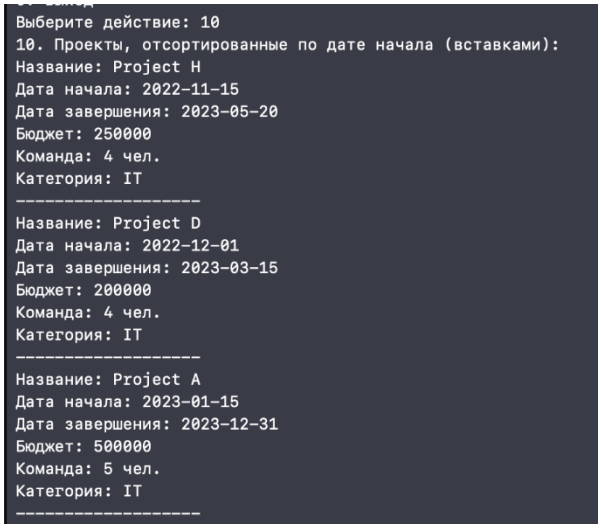

Рисунок 11 – Сортировка вставками по дате начала

При выборе действия 11 программа запрашивает минимальный бюджет и выполняет поиск проектов, чей бюджет ниже указанного значения. Массив сортируется по количеству участников с помощью `selectionSortByTeamSize`. Совпадающие проекты выводятся в консоль и записываются в `result.txt` в структурированном виде. Если подходящих проектов нет, отображается сообщение "Проекты с бюджетом менее [значение] не найдены". В случае ошибки доступа к данным выводится уведомление "Ошибка при выполнении операции". Ввод бюджета проверяется на положительность.

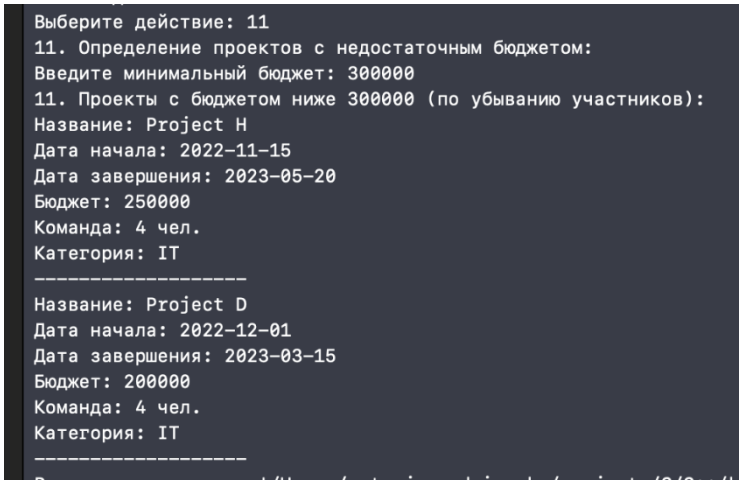

Рисунок 12 – Проекты с недостаточным бюджетом

При выборе действия 12 программа выполняет функцию `generateProjectStatistics`, которая группирует проекты по категориям, сортирует их по бюджету и формирует статистический отчёт. Для каждой категории выводится: список проектов, самый дешевый, самый дорогой, самый короткий и самый долгий проекты, определяемые по бюджету и длительности (с помощью dateDifference). Данные отображаются в консоли и записываются в result.txt. Если массив пуст или файл не открывается, выводится сообщение "Ошибка при формировании

статистики" или "Файл пуст". При успешном выполнении отображается уведомление о записи результата.

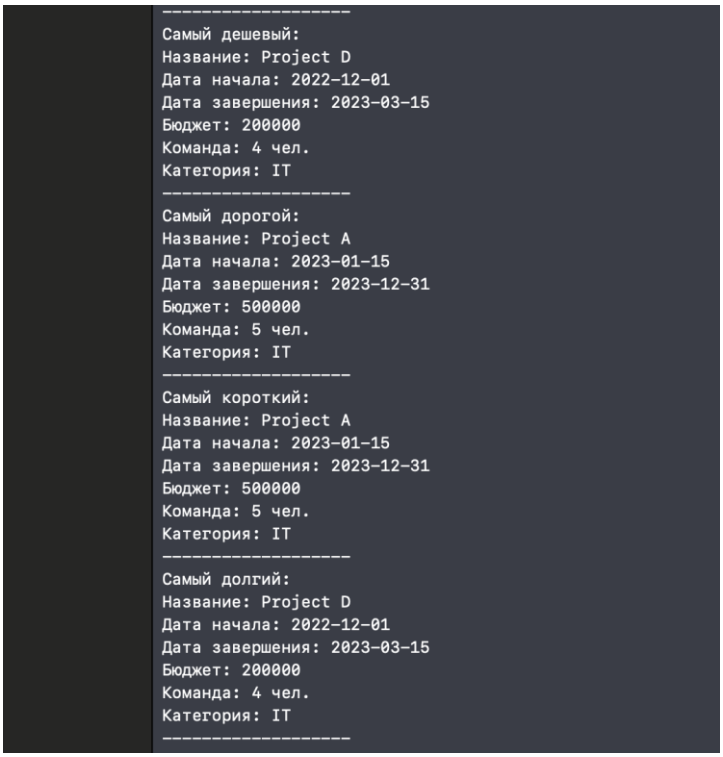

Рисунок 13– Вывод первой опции

Таким образом, функционал программы интуитивно понятен и легок при использовании. Программа достаточно удобна для работы с большими объемами данных. Она помогает реализовать эффективное системой бронирования столиков в ресторане.

ЗАКЛЮЧЕНИЕ

Разработанная в рамках курсовой работы программа управления проектами представляет собой эффективный и удобный инструмент для работы с данными о проектах. Благодаря интуитивно понятному интерфейсу, реализованному через консольное меню с 12 опциями, пользователи могут легко выполнять широкий спектр задач: от создания и заполнения бинарного файла `projects.bin` тестовыми данными до сложного анализа проектов по категориям.

Особое внимание в программе уделено обработке данных: линейный и бинарный поиски обеспечивают быстрый доступ к информации, а три метода сортировки (быстрая, выбором и вставками) позволяют гибко упорядочивать проекты по различным критериям. Работа с файлами организована оптимально: бинарный формат `projects.bin` гарантирует компактное хранение, а текстовый `result.txt` предоставляет структурированные отчёты для дальнейшего анализа. Программа демонстрирует высокую степень автоматизации, упрощая управление проектами и экономя время пользователя.

Несмотря на свои сильные стороны, система имеет потенциал для дальнейшего совершенствования. Например, можно реализовать динамическое выделение памяти для снятия ограничения в 20 проектов, добавить проверку уникальности названий или улучшить расчёт длительности проектов. Эти улучшения сделают программу ещё более универсальной.

В заключение, данная программа успешно решает задачи управления проектами, сочетая простоту использования, функциональность и надёжность. Она является ценным инструментом для организации данных и может быть адаптирована для реальных бизнес-процессов, демонстрируя практическую значимость разработанных алгоритмов и подходов.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Бхаргава А. Грокаем алгоритмы. Иллюстрированное пособие для программистов и любопытствующих. - СПб.: Питер, 2017. - 288 с.: ил. - (Серия «Библиотека программиста»)

[2] Васильев, А. Н. Программирование на C++ в примерах и задачах /А.Н. Васильев. - Москва : Издательство «Э», 2017. - 368 с. - (Российский компьютерный бестселлер).

[3] Беспалов, С. А Основы алгоритмизации и программирования (язык С/С++). лабораторный практикум в 2 ч. Ч. 2 : учеб. – метод. пособие / С. А. Беспалов [и др.] . – Минск: БГУИР, 2018. – 114 с.: ил.

[4] Синицын, А. К. Программирование алгоритмов в среде Builder C++: в 2 ч. / А. К. Синицын. – Минск : БГУИР, 2004 – 2005. – 2 ч.

[5] Алгоритмы: построение и анализ, 2-е издание. : Пер. с англ. — М. :Издательский дом “Вильямс”, 2011 — 1296 с. : ил. — Парал. тит. англ.

[6] Структуры данных и алгоритмы. : Пер. с англ. : Уч. дос. — М. : Издательский дом "Вильямс", 2000. — 384 с. : ил. — Парел. тит. англ.

[7] Макконнелл С. Совершенный код. Мастер-класс / Пер. с англ. — М. : Издательство «Русская редакция», 2010 — 896 стр. : ил.

[8] Кнут, Д. Э. Искусство программирования, том 1. Основные алгоритмы, 3-е изд. : Пер. с англ. — М. : ООО "И.Д. Вильямс", 2018. — 720 с. : ил. — Парал. тит. англ.

ПРИЛОЖЕНИЕ А
(обязательное)
Листинг кода

```
#include <iostream>
#include <fstream>

using namespace std;
const char* BASE_PATH =
"/Users/matvejnevodnicenko/projects/C/C++/kursach/kursach/";

struct Project {
    char name[50];
    char startDate[11];
    char endDate[11];
    double budget;
    int teamSize;
    char category[50];
};

bool isValidDate(const char* date) {
    if (strlen(date) != 10) return false;
    if (date[4] != '-' || date[7] != '-') return false;

    for (int i = 0; i < 10; i++) {
        if (i == 4 || i == 7) continue;
        if (date[i] < '0' || date[i] > '9') return false;
    }

    int year = (date[0] - '0') * 1000 + (date[1] - '0') * 100 +
(date[2] - '0') * 10 + (date[3] - '0');
    int month = (date[5] - '0') * 10 + (date[6] - '0');
    int day = (date[8] - '0') * 10 + (date[9] - '0');

    if (year < 2000 || year > 9999) return false;
    if (month < 1 || month > 12) return false;
    if (day < 1 || day > 31) return false;

    if (month == 4 || month == 6 || month == 9 || month == 11) {
        if (day > 30) return false;
    } else if (month == 2) {
        if (day > 29) return false;
    }
    return true;
}
bool dateLessOrEqual(const char* date1, const char* date2) {
    for (int i = 0; i < 10; i++) {
        if (date1[i] < date2[i]) return true;
        if (date1[i] > date2[i]) return false;
    }
    return true;
}
bool isNonEmptyString(const char* str) {
    return str[0] != '\0';
}
```

Продолжение приложения А

```
bool strEqual(const char* str1, const char* str2) {
    int i = 0;
    while (str1[i] != '\0' && str2[i] != '\0') {
        if (str1[i] != str2[i]) return false;
        i++;
    }
    return str1[i] == str2[i];
}
bool dateLessThan(const char* date1, const char* date2) {
    for (int i = 0; i < 10; i++) {
        if (date1[i] < date2[i]) return true;
        if (date1[i] > date2[i]) return false;
    }
    return false;
}
void quickSortByBudget(Project projects[], int low, int high) {
    if (low < high) {
        double pivot = projects[high].budget;
        int i = low - 1;
        for (int j = low; j < high; j++) {
            if (projects[j].budget <= pivot) {
                i++;
                Project temp = projects[i];
                projects[i] = projects[j];
                projects[j] = temp;
            }
        }
        Project temp = projects[i + 1];
        projects[i + 1] = projects[high];
        projects[high] = temp;
        int pi = i + 1;

        quickSortByBudget(projects, low, pi - 1);
        quickSortByBudget(projects, pi + 1, high);
    }
}

void selectionSortByTeamSize(Project projects[], int size) {
    for (int i = 0; i < size - 1; i++) {
        int maxIdx = i;
        for (int j = i + 1; j < size; j++) {
            if (projects[j].teamSize > projects[maxIdx].teamSize) {
                maxIdx = j;
            }
        }
        if (maxIdx != i) {
            Project temp = projects[i];
            projects[i] = projects[maxIdx];
            projects[maxIdx] = temp;
        }
    }
}

void insertionSortByStartDate(Project projects[], int size) {
```

Продолжение приложения А

```
for (int i = 1; i < size; i++) {
        Project key = projects[i];
        int j = i - 1;
        while (j >= 0 && dateLessThan(key.startDate,
projects[j].startDate)) {
            projects[j + 1] = projects[j];
            j--;
        }
        projects[j + 1] = key;
    }
}

void writeProjectToFileAndConsole(ofstream& file, const Project& p) {
    file << "Название: " << p.name << "\n";
    cout << "Название: " << p.name << "\n";
    file << "Дата начала: " << p.startDate << "\n";
    cout << "Дата начала: " << p.startDate << "\n";
    file << "Дата завершения: " << p.endDate << "\n";
    cout << "Дата завершения: " << p.endDate << "\n";
    file << "Бюджет: " << p.budget << "\n";
    cout << "Бюджет: " << p.budget << "\n";
    file << "Команда: " << p.teamSize << " чел.\n";
    cout << "Команда: " << p.teamSize << " чел.\n";
    file << "Категория: " << p.category << "\n";
    cout << "Категория: " << p.category << "\n";
    file << "--------------------\n";
    cout << "--------------------\n";
}
void writeProjectToConsole(const Project& p) {
    cout << "Название: " << p.name << "\n";
    cout << "Дата начала: " << p.startDate << "\n";
    cout << "Дата завершения: " << p.endDate << "\n";
    cout << "Бюджет: " << p.budget << "\n";
    cout << "Команда: " << p.teamSize << " чел.\n";
    cout << "Категория: " << p.category << "\n";
    cout << "--------------------\n";
}
int readFromBinaryFile(Project projects[], int maxSize, const char*
filename) {
    char fullPath[256];
    int i = 0;
    while (BASE_PATH[i] != '\0') { fullPath[i] = BASE_PATH[i]; i++; }
    int j = 0;
    while (filename[j] != '\0'){ fullPath[i + j] = filename[j]; j++; }
    fullPath[i + j] = '\0';

    ifstream inFile(fullPath, ios::binary);
    if (!inFile) {
        cout << "Не удалось открыть бинарный файл '" << fullPath <<
"'!\n";
       return 0;
    }
    int count = 0;
    while (count < maxSize && inFile.read((char*)&projects[count],
sizeof(Project))) {
```

Продолжение приложения А

```
 count++;
    }
    inFile.close();
    return count;
}

void initializeBinaryFile(const char* filename) {
    char fullPath[256];
    int i = 0;
    while (BASE_PATH[i] != '\0') { fullPath[i] = BASE_PATH[i]; i++; }
    int j = 0;
    while (filename[j] != '\0') { fullPath[i + j] = filename[j]; j++;
}
    fullPath[i + j] = '\0';

    ofstream outFile(fullPath, ios::binary);
    if (!outFile) {
        cout << "Не удалось создать бинарный файл '" << fullPath <<
"'!\n";
        return;
    }
    Project sampleData[10] = {
 {"Project A", "2023-01-15", "2023-12-31", 500000.0, 5, "IT"},
 {"Project B", "2023-03-01", "2024-06-30", 750000.0, 8,"Development"},
 {"Project C", "2023-06-10", "2023-11-30", 300000.0, 3,"Research"},
 {"Project D", "2022-12-01", "2023-03-15", 200000.0, 4, "IT"},
 {"Project E", "2023-04-01", "2025-01-01", 900000.0, 6,"Development"},
 {"Project F", "2024-01-10", "2024-09-30", 400000.0, 7,"Marketing"},
 {"Project G", "2023-07-01", "2024-03-31", 600000.0, 5, "Research"},
 {"Project H", "2022-11-15", "2023-05-20", 250000.0, 4, "IT"},
 {"Project I", "2023-09-01", "2024-12-31", 800000.0, 9,"Development"},
 {"Project J", "2024-02-01", "2025-06-30", 550000.0, 6, "Marketing"}
    };
    for (int i = 0; i < 10; i++) {
        outFile.write((char*)&sampleData[i], sizeof(Project));
    }
    outFile.close();
    cout << "Бинарный файл по пути'" << fullPath << "' создан.\n";
}

int dateDifference(const char* start, const char* end) {
    int startYear = (start[0] - '0') * 1000 + (start[1] - '0') * 100 +
(start[2] - '0') * 10 + (start[3] - '0');
    int endYear = (end[0] - '0') * 1000 + (end[1] - '0') * 100 +
(end[2] - '0') * 10 + (end[3] - '0');
    return (endYear - startYear) * 365;
}
int linearSearchByName(Project projects[], int size, const char*
searchName) {
    for (int i = 0; i < size; i++) {
        if (strEqual(projects[i].name, searchName)) {
            return i;
        }
    }
    return -1;
}
```

Продолжение приложения А

```
int binarySearchByBudget(Project projects[], int size, double
searchBudget) {
    int left = 0;
    int right = size - 1;

    while (left <= right) {
        int mid = left + (right - left) / 2;
        if (projects[mid].budget == searchBudget) {
            return mid;
        }
        if (projects[mid].budget < searchBudget) {
            left = mid + 1;
        } else {
            right = mid - 1;
        }
    }

    return (right >= 0) ? right : -1;
}

void addProjectToFile(Project projects[], int& projectCount, int
maxSize, const char* filename) {
    if (projectCount >= maxSize) {
        cout << "Достигнуто максимальное количество проектов (" <<
maxSize << ")!\n";
        return;
    }
    Project newProject;
    bool validInput = false;
    do {
        cout << "Введите название проекта (не пустое): ";
        cin.getline(newProject.name, 50);
        if (!isNonEmptyString(newProject.name)) {
            cout << "Ошибка: Название не может быть пустым.\n";
            continue;
        }
        cout << "Введите дату начала (ГГГГ-ММ-ДД): ";
        cin.getline(newProject.startDate, 11);
        if (!isValidDate(newProject.startDate)) {
            cout << "Ошибка: Неверный формат даты (ожидается ГГГГ-ММ-
ДД, год >= 2000).\n";
            continue;
        }

        cout << "Введите дату завершения (ГГГГ-ММ-ДД): ";
        cin.getline(newProject.endDate, 11);
        if (!isValidDate(newProject.endDate)) {
            cout << "Ошибка: Неверный формат даты (ожидается ГГГГ-ММ-
ДД, год >= 2000).\n";
            continue;
        }
        if (!dateLessOrEqual(newProject.startDate,
newProject.endDate)) {
            cout << "Ошибка: Дата завершения должна быть не раньше
даты начала.\n";
            continue;}
```

Продолжение приложения А

```
      cout << "Введите бюджет (> 0): ";
         cin >> newProject.budget;
         if (cin.fail() || newProject.budget <= 0) {
           cout << "Ошибка: Бюджет должен быть положительным числом\n";
             cin.clear();
             cin.ignore(10000, '\n');
             continue;
         }

         cout << "Введите количество участников (> 0): ";
         cin >> newProject.teamSize;
         if (cin.fail() || newProject.teamSize <= 0) {
             cout << "Ошибка: Количество участников должно быть
положительным числом.\n";
             cin.clear();
             cin.ignore(10000, '\n');
             continue;
         }

         cin.ignore(10000, '\n');
         cout << "Введите категорию (не пустую): ";
         cin.getline(newProject.category, 50);
         if (!isNonEmptyString(newProject.category)) {
             cout << "Ошибка: Категория не может быть пустой.\n";
             continue;
         }

         validInput = true;
     } while (!validInput);

     projects[projectCount] = newProject;
     projectCount++;
     char fullPath[256];
     int i = 0;
     while (BASE_PATH[i] != '\0') { fullPath[i] = BASE_PATH[i]; i++; }
     int j = 0;
     while (filename[j] != '\0') { fullPath[i + j] =filename[j]; j++; }
     fullPath[i + j] = '\0';
     ofstream outFile(fullPath, ios::binary | ios::app);
     if (!outFile) {
         cout << "Не удалось открыть бинарный файл '" << fullPath << "'
для добавления!\n";
         return;
     }
     outFile.write((char*)&newProject, sizeof(Project));
     outFile.close();
     cout << "Проект '" << newProject.name << "' успешно добавлен.\n";
}

void editProjectInFile(Project projects[], int projectCount, const
char* filename) {
     char searchName[50];
     cout << "Введите название проекта для редактирования: ";
     cin.getline(searchName, 50);
     if (!isNonEmptyString(searchName)) {
```

Продолжение приложения А

```
      cout << "Ошибка: Название не может быть пустым.\n";
         return;
    }

    int index = linearSearchByName(projects, projectCount,
searchName);
    if (index == -1) {
        cout << "Проект с названием '" << searchName << "' не
найден.\n";
        return;
    }

    cout << "Текущие данные проекта:\n";
    cout << "Название: " << projects[index].name << "\n";
    cout << "Дата начала: " << projects[index].startDate << "\n";
    cout << "Дата завершения: " << projects[index].endDate << "\n";
    cout << "Бюджет: " << projects[index].budget << "\n";
    cout << "Команда: " << projects[index].teamSize << " чел.\n";
    cout << "Категория: " << projects[index].category << "\n";
    cout << "Введите новые данные (Enter для сохранения текущих
значений):\n";
    char newName[50] = "";
    cout << "Новое название: ";
    cin.getline(newName, 50);
    if (isNonEmptyString(newName)) {
        for (int i = 0; i < 50; i++) projects[index].name[i] =
newName[i];
    }
    char newStartDate[11] = "";
    cout << "Новая дата начала (ГГГГ-ММ-ДД): ";
    cin.getline(newStartDate, 11);
    if (isNonEmptyString(newStartDate)) {
        if (!isValidDate(newStartDate)) {
            cout << "Ошибка: Неверный формат даты. Сохранено текущее
значение.\n";
        } else {
            for (int i = 0; i < 11; i++) projects[index].startDate[i]
= newStartDate[i];
        }
    }
    char newEndDate[11] = "";
    cout << "Новая дата завершения (ГГГГ-ММ-ДД): ";
    cin.getline(newEndDate, 11);
    if (isNonEmptyString(newEndDate)) {
        if (!isValidDate(newEndDate)) {
            cout << "Ошибка: Неверный формат даты. Сохранено текущее
значение.\n";
        } else {
            for (int i = 0; i < 11; i++) projects[index].endDate[i] =
newEndDate[i];
        }
    }

    if (isNonEmptyString(newStartDate) ||
isNonEmptyString(newEndDate)) {
```

Продолжение приложения А

```
     if (!dateLessOrEqual(projects[index].startDate,
projects[index].endDate)) {
            cout << "Ошибка: Дата завершения должна быть не раньше
даты начала. Сохранены текущие значения дат.\n";
            for (int i = 0; i < 11; i++) projects[index].startDate[i]
= projects[index].startDate[i];
            for (int i = 0; i < 11; i++) projects[index].endDate[i] =
projects[index].endDate[i];
        }
    }

    double newBudget;
    cout << "Новый бюджет (0 для сохранения текущего): ";
    cin >> newBudget;
    if (cin.fail() || (newBudget < 0 && newBudget != 0)) {
        cout << "Ошибка: Бюджет должен быть положительным или 0.
Сохранено текущее значение.\n";
        cin.clear();
        cin.ignore(10000, '\n');
    } else if (newBudget > 0) {
        projects[index].budget = newBudget;
    }

    int newTeamSize;
    cout << "Новое количество участников (0 для сохранения текущего):
";
    cin >> newTeamSize;
    if (cin.fail() || (newTeamSize < 0 && newTeamSize != 0)) {
        cout << "Ошибка: Количество участников должно быть
положительным или 0. Сохранено текущее значение.\n";
        cin.clear();
        cin.ignore(10000, '\n');
    } else if (newTeamSize > 0) {
        projects[index].teamSize = newTeamSize;
    }

    cin.ignore(10000, '\n');
    char newCategory[50] = "";
    cout << "Новая категория: ";
    cin.getline(newCategory, 50);
    if (isNonEmptyString(newCategory)) {
        for (int i = 0; i < 50; i++) projects[index].category[i] =
newCategory[i];
    }
    char fullPath[256];
    int i = 0;
    while (BASE_PATH[i] != '\0') { fullPath[i] = BASE_PATH[i]; i++; }
    int j = 0;
    while (filename[j] != '\0') { fullPath[i + j] = filename[j]; j++;
}
    fullPath[i + j] = '\0';
    ofstream outFile(fullPath, ios::binary);
    if (!outFile) {
        cout << "Не удалось открыть бинарный файл '" << fullPath << "'
для редактирования!\n";
```

```
      return;
    }
    for (int i = 0; i < projectCount; i++) {
        outFile.write((char*)&projects[i], sizeof(Project));
    }
    outFile.close();
    cout << "Проект '" << projects[index].name << "' успешно
отредактирован.\n";
}

void deleteProjectFromFile(Project projects[], int& projectCount,
const char* filename) {
    char searchName[50];
    cout << "Введите название проекта для удаления: ";
    cin.getline(searchName, 50);
    if (!isNonEmptyString(searchName)) {
        cout << "Ошибка: Название не может быть пустым.\n";
        return;
    }

    int index = linearSearchByName(projects, projectCount,
searchName);
    if (index == -1) {
        cout << "Проект с названием '" << searchName << "' не
найден.\n";
        return;
    }

    for (int i = index; i < projectCount - 1; i++) {
        projects[i] = projects[i + 1];
    }
    projectCount--;
    char fullPath[256];
    int i = 0;
    while (BASE_PATH[i] != '\0') { fullPath[i] = BASE_PATH[i]; i++; }
    int j = 0;
    while (filename[j] != '\0'){ fullPath[i + j] = filename[j]; j++; }
    fullPath[i + j] = '\0';
    ofstream outFile(fullPath, ios::binary);
    if (!outFile) {
        cout << "Не удалось открыть бинарный файл '" << fullPath << "'
для удаления!\n";
        return;
    }
    for (int i = 0; i < projectCount; i++) {
        outFile.write((char*)&projects[i], sizeof(Project));
    }
    outFile.close();
    cout << "Проект '" << searchName << "' успешно удален.\n";
}
void viewResultsFile(const char* filename, const char* textFile) {
    char fullPath[256];
    int i = 0;
    while (BASE_PATH[i] != '\0') { fullPath[i] = BASE_PATH[i]; i++; }
    int j = 0;
```

Продолжение приложения А

```
     while (filename[j] != '\0') { fullPath[i + j] = filename[j]; j++;
}
    fullPath[i + j] = '\0';

    char fullTextPath[256];
    i = 0;
    while (BASE_PATH[i] != '\0'){ fullTextPath[i] =BASE_PATH[I]; i+; }
    j = 0;
    while(textFile[j] != '\0'){ fullTextPath[i + j]= textFile[j];j+; }
    fullTextPath[i + j] = '\0';

    ifstream inFile(fullPath, ios::binary);
    if (!inFile) {
        cout << "Не удалось открыть бинарный файл '" << fullPath << "'
для чтения!\n";
        return;
    }

    ofstream outFile(fullTextPath, ios::out | ios::trunc);
    if (!outFile) {
        cout << "Не удалось открыть текстовый файл '" << fullTextPath
<< "' для записи!\n";
        inFile.close();
        return;
    }
    outFile << "2. Просмотр записей из бинарного файла:\n";
    cout << "\nСодержимое файла '" << fullPath << "':\n";
    cout << "----------------------------------------\n";
    Project p;
    bool isEmpty = true;
    while (inFile.read((char*)&p, sizeof(Project))) {
        outFile << "Название: " << p.name << "\n";
        outFile << "Дата начала: " << p.startDate << "\n";
        outFile << "Дата завершения: " << p.endDate << "\n";
        outFile << "Бюджет: " << p.budget << "\n";
        outFile << "Команда: " << p.teamSize << " чел.\n";
        outFile << "Категория: " << p.category << "\n";
        outFile << "------------------\n";
        writeProjectToConsole(p);
        isEmpty = false;
    }
    if (isEmpty) {
        outFile << "Файл пуст.\n";
        cout << "Файл пуст.\n";
    }
    cout << "----------------------------------------\n";
    inFile.close();
    outFile.close();
}
void generateProjectStatistics(Project projects[], int projectCount,
const char* textFile) {
    char fullTextPath[256];
    int i = 0;
    while (BASE_PATH[i] != '\0') { fullTextPath[i] = BASE_PATH[i];
i++; }
    int j = 0;
```

Продолжение приложения А

```
      while (textFile[j] != '\0') { fullTextPath[i + j] = textFile[j];
j++; }
    fullTextPath[i + j] = '\0';

    ofstream outFile(fullTextPath, ios::out | ios::trunc);
    if (!outFile.is_open()) {
        cout << "Не удалось открыть текстовый файл '" << fullTextPath
<< "' для записи!\n";
        return;
    }

    outFile << "12. Статистика по категориям:\n";
    cout << "12. Статистика по категориям:\n";
    quickSortByBudget(projects, 0, projectCount - 1);
    const int MAX_CATEGORIES = 10;
    char categories[MAX_CATEGORIES][50];
    int catCount = 0;

    for (int i = 0; i < projectCount; i++) {
        bool exists = false;
        for (int j = 0; j < catCount; j++) {
            if (strEqual(projects[i].category, categories[j])) {
                exists = true;
                break;
            }
        }
        if (!exists && catCount < MAX_CATEGORIES) {
            int k = 0;
            while (projects[i].category[k] != '\0') {
                categories[catCount][k] = projects[i].category[k];
                k++;
            }
            categories[catCount][k] = '\0';
            catCount++;
        }
    }
    for (int c = 0; c < catCount; c++) {
        outFile << "\nКатегория: " << categories[c] << "\n";
        cout << "\nКатегория: " << categories[c] << "\n";
        outFile << "Список проектов по возрастанию бюджета:\n";
        cout << "Список проектов по возрастанию бюджета:\n";
        for (int i = 0; i < projectCount; i++) {
            if (strEqual(projects[i].category, categories[c])) {
                writeProjectToFileAndConsole(outFile, projects[i]);
            }
        }
        double minBudget = 1e9, maxBudget = 0;
        int minDuration = 1e9, maxDuration = 0;
        int minIdx = -1, maxIdx = -1, shortIdx = -1, longIdx = -1;

        for (int i = 0; i < projectCount; i++) {
            if (strEqual(projects[i].category, categories[c])) {
                if (projects[i].budget < minBudget) {
                    minBudget = projects[i].budget;
                    minIdx = i;}
```

Продолжение приложения А

```
                if (projects[i].budget > maxBudget) {
                    maxBudget = projects[i].budget;
                    maxIdx = i;
                }
                int duration = dateDifference(projects[i].startDate,
projects[i].endDate);
                if (duration < minDuration) {
                    minDuration = duration;
                    shortIdx = i;
                }
                if (duration > maxDuration) {
                    maxDuration = duration;
                    longIdx = i;
                }
            }
        }

        if (minIdx != -1) {
            outFile << "Самый дешевый:\n"; cout << "Самый дешевый:\n";
            writeProjectToFileAndConsole(outFile, projects[minIdx]);
            outFile << "Самый дорогой:\n"; cout << "Самый дорогой:\n";
            writeProjectToFileAndConsole(outFile, projects[maxIdx]);
            outFile << "Самый короткий:\n"; cout << "Самый короткий:
                \n»;
            writeProjectToFileAndConsole(outFile, projects[shortIdx]);
            outFile << "Самый долгий:\n"; cout << "Самый долгий:\n";
            writeProjectToFileAndConsole(outFile, projects[longIdx]);
        }
    }
    outFile.close();
    cout << "Результат записан в '" << fullTextPath << "'.\n";
}

int main() {
    const int MAX_SIZE = 20;
    Project projects[MAX_SIZE];
    const char* binFile = "projects.bin";
    const char* textFile = "result.txt";
    char fullTextPath[256];
    int i = 0;
    while (BASE_PATH[i] != '\0') { fullTextPath[i] = BASE_PATH[i];
i++; }
    int j = 0;
    while (textFile[j] != '\0') { fullTextPath[i + j] = textFile[j];
j++; }
    fullTextPath[i + j] = '\0';
    int projectCount = readFromBinaryFile(projects, MAX_SIZE,
binFile);

    int choice;
    do {
        cout << "\nМеню:\n";
        cout << "1. Создание нового бинарного файла\n";
        cout << "2. Просмотр записей из бинарного файла\n";
```

Продолжение приложения А

```
        cout << "3. Добавление записей в бинарный файл\n";
        cout << "4. Редактирование записей в бинарном файле\n";
        cout << "5. Удаление записей из бинарного файла\n";
        cout << "6. Линейный поиск по названию проекта\n";
        cout << "7. Бинарный поиск по общему бюджету\n";
        cout << "8. Быстрая сортировка по бюджету\n";
        cout << "9. Сортировка выбором по количеству участников\n";
        cout << "10. Сортировка вставками по дате начала
разработки\n";
        cout << "11. Определение проектов с недостаточным бюджетом\n";
        cout << "12. Статистика по проектам\n";
        cout << "0. Выход\n";
        cout << "Выберите действие: ";
        cin >> choice;
        cin.ignore(10000, '\n');
        ofstream outFile;
        switch (choice) {
            case 1: {
                cout << "1. Создание нового бинарного файла:\n";
                initializeBinaryFile(binFile);
                projectCount = readFromBinaryFile(projects, MAX_SIZE,
binFile);
                if (projectCount == 0) {
                    cout << "Не удалось загрузить данные после
создания файла!\n";
                }
                break;
            }
            case 2: {
                cout << "2. Просмотр записей из бинарного файла:\n";
                viewResultsFile(binFile, textFile);
                cout << "Результат записан в '" << fullTextPath <<
"'.\n";
                break;
            }
            case 3: {
                cout << "3. Добавление записей в бинарный файл:\n";
                addProjectToFile(projects, projectCount, MAX_SIZE,
binFile);
                cout << "Проект добавлен. Файл '" << fullTextPath <<
"' не изменен.\n";
                break;
            }
            case 4: {
                cout << "4. Редактирование записей в бинарном
файле:\n";
                editProjectInFile(projects, projectCount, binFile);
                cout << "Проект отредактирован. Файл '" <<
fullTextPath << "' не изменен.\n";
                break;}
            case 5: {
                cout << "5. Удаление записей из бинарного файла:\n";
                deleteProjectFromFile(projects, projectCount,
binFile);
                cout << "Проект удален. Файл '" << fullTextPath << "'
не изменен.\n";
```

Продолжение приложения А

```
                break;
            }
            case 6: {
                char searchName[50];
                cout << "6. Линейный поиск по названию проекта:\n";
                cout << "Введите название проекта для поиска: ";
                cin.getline(searchName, 50);
                if (!isNonEmptyString(searchName)) {
                    cout << "Ошибка: Название не может быть
пустым.\n";
                    break;
                }
                outFile.open(fullTextPath, ios::out | ios::trunc);
                if (!outFile.is_open()) {
                    cout << "Не удалось открыть текстовый файл '" <<
fullTextPath << "' для записи!\n";
                    break;
                }
                outFile << "6. Линейный поиск по названию '" <<
searchName << "':\n";
                cout << "6. Линейный поиск по названию '" <<
searchName << "':\n";
                int index = linearSearchByName(projects, projectCount,
searchName);
                if (index != -1) {
                    writeProjectToFileAndConsole(outFile,
projects[index]);
                } else {
                    outFile << "Проект с названием '" << searchName <<
"' не найден.\n";
                    cout << "Проект с названием '" << searchName << "'
не найден.\n";
                }
                outFile.close();
                cout << "Результат записан в '" << fullTextPath <<
"'.\n";
                break; }
            case 7: {
                double searchBudget;
                cout << "7. Бинарный поиск по общему бюджету:\n";
                cout << "Введите бюджет для поиска: ";
                cin >> searchBudget;
                if (cin.fail() || searchBudget <= 0) {
                    cout << "Ошибка: Бюджет должен быть положительным
числом.\n";
                    cin.clear();
                    cin.ignore(10000, '\n');
                    break; }
                outFile.open(fullTextPath, ios::out | ios::trunc);
                if (!outFile.is_open()) {
                    cout << "Не удалось открыть текстовый файл '" <<
fullTextPath << "' для записи!\n";
                    break;
                }
                outFile << "7. Бинарный поиск по бюджету '" <<
searchBudget << «':\n";
```

Продолжение приложения А

```
                cout << "7. Бинарный поиск по бюджету '" <<
searchBudget << "':\n";
                quickSortByBudget(projects, 0, projectCount - 1);
                int index = binarySearchByBudget(projects,
projectCount, searchBudget);
                if (index != -1) {
                    if (projects[index].budget == searchBudget) {
                        outFile << "Найден проект с точным
бюджетом:\n";
                        cout << "Найден проект с точным бюджетом:\n";
                    } else {
                        outFile << "Точного совпадения нет. Ближайший
меньший бюджет:\n";
                        cout << "Точного совпадения нет. Ближайший
меньший бюджет:\n";
                    }
                    writeProjectToFileAndConsole(outFile,
projects[index]);
                } else {
                    outFile << "Проект с бюджетом около '" <<
searchBudget << "' не найден.\n";
                    cout << "Проект с бюджетом около '" <<
searchBudget << "' не найден.\n";
                }
                outFile.close();
                cout << "Результат записан в '" << fullTextPath <<
"'.\n";
                break; }
            case 8: {
                outFile.open(fullTextPath, ios::out | ios::trunc);
                if (!outFile.is_open()) {
                    cout << "Не удалось открыть текстовый файл '" <<
fullTextPath << "' для записи!\n";
                    break; }
                outFile << "8. Проекты, отсортированные по бюджету
(быстрая сортировка):\n";
                cout << "8. Проекты, отсортированные по бюджету
(быстрая сортировка):\n";
                quickSortByBudget(projects, 0, projectCount - 1);
                for (int i = 0; i < projectCount; i++)
writeProjectToFileAndConsole(outFile, projects[i]);
                outFile.close();
                cout << "Результат записан в '" << fullTextPath <<
"'.\n";
                break; }
            case 9: {
                outFile.open(fullTextPath, ios::out | ios::trunc);
                if (!outFile.is_open()) {
                    cout << "Не удалось открыть текстовый файл '" <<
fullTextPath << "' для записи!\n";
                    break;
                }
                outFile << "9. Проекты, отсортированные по количеству
участников (выбором, убывание):\n»;
```

Продолжение приложения А

```
                cout << "9. Проекты, отсортированные по количеству
участников (выбором, убывание):\n";
                selectionSortByTeamSize(projects, projectCount);
                for (int i = 0; i < projectCount; i++)
writeProjectToFileAndConsole(outFile, projects[i]);
                outFile.close();
                cout << "Результат записан в '" << fullTextPath <<
"'.\n";
                break;
            }
            case 10: {
                outFile.open(fullTextPath, ios::out | ios::trunc);
                if (!outFile.is_open()) {
                    cout << "Не удалось открыть текстовый файл '" <<
fullTextPath << "' для записи!\n";
                    break;
                }
                outFile << "10. Проекты, отсортированные по дате
начала (вставками):\n";
                cout << "10. Проекты, отсортированные по дате начала
(вставками):\n";
                insertionSortByStartDate(projects, projectCount);
                for (int i = 0; i < projectCount; i++)
writeProjectToFileAndConsole(outFile, projects[i]);
                outFile.close();
                cout << "Результат записан в '" << fullTextPath <<
"'.\n";
                break;
            }
            case 11: {
                double minBudget;
                cout << "11. Определение проектов с недостаточным
бюджетом:\n";
                cout << "Введите минимальный бюджет: ";
                cin >> minBudget;
                if (cin.fail() || minBudget <= 0) {
                    cout << "Ошибка: Бюджет должен быть положительным
числом.\n";
                    cin.clear();
                    cin.ignore(10000, '\n');
                    break; }
                outFile.open(fullTextPath, ios::out | ios::trunc);
                if (!outFile.is_open()) {
                    cout << "Не удалось открыть текстовый файл '" <<
fullTextPath << "' для записи!\n";
                    break; }
                outFile << "11. Проекты с бюджетом ниже " << minBudget
<< " (по убыванию участников):\n";
                cout << "11. Проекты с бюджетом ниже " << minBudget <<
" (по убыванию участников):\n";
                selectionSortByTeamSize(projects, projectCount);
                bool found = false;
                for (int i = 0; i < projectCount; i++) {
                    if (projects[i].budget < minBudget) {
                        writeProjectToFileAndConsole(outFile,
projects[I]);
```

Продолжение приложения А

```
                        found = true;
                    }
                }
                if (!found) {
                    outFile << "Таких проектов нет.\n";
                    cout << "Таких проектов нет.\n";
                }
                outFile.close();
                cout << "Результат записан в '" << fullTextPath <<
"'.\n";
                break;
            }
            case 12: {
                cout << "12. Статистика по проектам:\n";
                generateProjectStatistics(projects, projectCount,
textFile);
                break;
            }
            case 0: {
                cout << "Выход.\n";
                break;
            }
            default: {
                cout << "Неверный выбор! Попробуйте снова.\n";
                break;
            }
        }
    } while (choice != 0);

    cout << "Программа завершена. Файлы 'projects.bin' и 'result.txt'
находятся по пути: " << BASE_PATH << "\n";
    return 0;
}
```

ПРИЛОЖЕНИЕ Б
(обязательное)
Блок-схема работы программы

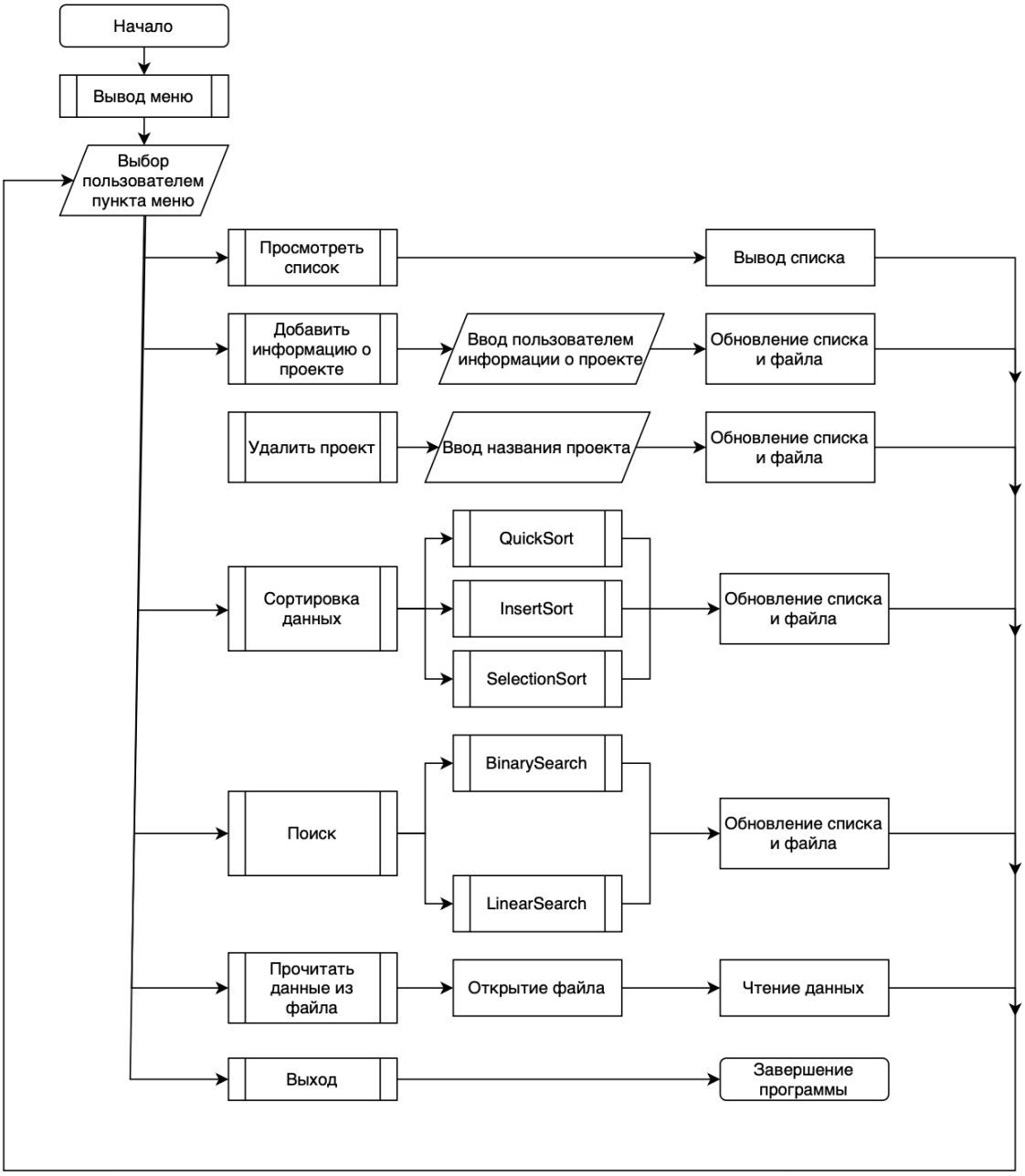

Рисунок Б.1 – Блок-схема работы программы

Обозначение	Наименование	Дополнительные сведения
	Текстовые документы	
БГУИР КП 6-05-0611-03 058 ПЗ	Пояснительная записка	48 с.

					БГУИР КП 6-05-0611-03 058 ВД			
Изм.	Л.	№ докум.	Подп.	Дата	СИСТЕМА ПЛАНИРОВАНИЯ БЮДЖЕТА РАЗРАБОТКИ ПРОГРАММНОГО ПРОЕКТА	Лит	Лист	Листов
Разраб.		Неводниченко				У	48	48
Проверил		Панасик				Кафедра ВМиП гр. 421701		