

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 ФАКУЛЬТАТИВНЫХ ЗАНЯТИЙ В
УНИВЕРСИТЕТЕ»**

Выполнил: гр.421701 Крамич А.А.

Проверила: Новицкая Л.И.

Минск 2025

РЕФЕРАТ

ТЕМА : Система планирования факультативных занятий в университете / А.А. Крамич. – Минск : БГУИР, 2025, – п.з. – 54 с., рисунков – 19, источников – 11, приложений – 2.

Данная пояснительная записка к курсовой работе содержит описание программы по обработке и анализу данных факультативов, разработанной на языке программирования C++. Работа состоит из текстовой части и программного кода.

В работе представлены ключевые алгоритмы обработки данных: быстрая сортировка, сортировка выбором и вставками, линейный и бинарный поиск, а также статистика по выбранному критерию. Также реализована система ведения отчётности о выполненных действиях в отдельном файле отчета.

Графические материалы (рисунки и таблицы), иллюстрирующие работу алгоритмов и структуру данных, прилагаются по тексту пояснительной записки.

Пояснительная записка включает все основные инструменты, использованные при разработке программного продукта, а также описание интерфейса и логики работы программы.

СОДЕРЖАНИЕ

Введение.....	7
1 Структуры и файлы.....	8
1.1 Структуры данных.....	8
1.2 Файлы.....	9
2 Алгоритмы сортировки.....	10
2.1 Быстрая сортировка	11
2.2 Сортировка выбором	12
2.3 Сортировка вставками.....	13
3 Алгоритмы поиска	15
3.1 Линейный поиск.....	15
3.2 Бинарный поиск	16
4 Пользовательские функции.....	18
4.1 Функция writeToReport().....	18
4.2 Функция viewInformation()	18
4.3 Функция createBinaryFile().....	19
4.4 Функция file_create()	19
4.5 Функция viewOneInformation().....	19
4.6 Функция viewOneInformationReport().....	19
4.7 Функция addInformation()	20
4.8 Функция deleteFacultativ().....	20
4.9 Функция edit()	20
4.10 Функция LineSearchFacultativ()	21
4.11 Функция binarySearchByPrepod()	21
4.12 Функция sortbychoice()	22
4.13 Функция insertionSortByFacultativ().....	22
4.14 Функция getDayIndex().....	23
4.15 Функция quickSortByDay()	23
4.16 Функция quickSortByPrepod().....	23
4.17 Функция SortInformation().....	23
4.18 Функция searchByPrepodAndDay()	24
4.19 Функция printStatistics()	25
4.20 Функция popular_facultative()	25

4.21 Функция menu()	26
4.22 Функция main()	26
5 Описание работы программы	27
Заключение	32
Список использованных источников	33
Приложение А (обязательное) Листинг кода	34
Приложение Б (обязательное) Блок-схема работы программы	53

ВВЕДЕНИЕ

Современные университеты сталкиваются с необходимостью эффективного управления образовательным процессом, что включает в себя планирование факультативных занятий. Автоматизация таких процессов позволяет повысить качество организации учебного времени, упростить взаимодействие между преподавателями и студентами, а также улучшить контроль за наполняемостью факультативов. Именно поэтому направление данной работы может помочь разобраться как именно происходит работа с базами данных в таких системах.

Целью работы является разработка программной системы на языке C++, которая позволяет хранить, обрабатывать и анализировать информацию о факультативных занятиях.

Задачи, которые необходимо выполнить в ходе выполнения курсовой работы для достижения поставленных целей:

- разработать структуру данных для хранения информации о факультативных занятиях, включая название факультатива, преподавателя, день недели и количество записавшихся студентов;
- реализовать функции добавления, редактирования и удаления записей с использованием бинарного файла;
- реализовать алгоритмы сортировки для упорядочивания данных о факультативах по различным критериям, таким как день недели, количество студентов, название факультатива;
- разработать алгоритмы поиска: по преподавателю, по дню недели, линейный поиск по названию и бинарный поиск по преподавателю;
- реализовать вывод статистики по преподавателям и определение самого популярного факультатива;
- провести тестирование всех функциональных возможностей системы, включая тестирование на корректность работы алгоритмов сортировок и поисков.

С увеличением объема данных и возможностей их обработки, использование программного обеспечения для управления факультативами становится необходимым. Создание системы на C++ позволит не только автоматизировать процессы, но и продемонстрировать применение алгоритмов сортировки и поиска в реальных задачах.

1 СТРУКТУРЫ И ФАЙЛЫ

1.1 Структуры данных

Структуры данных – это основа программирования, предназначенная для систематизации и эффективного хранения информации в оперативной памяти компьютера [7].

В данной работе для хранения информации о факультативных занятиях использовалась структура данных, объявленная с помощью ключевого слова `struct` языка программирования C++. Эта структура содержит основные поля, описывающие факультатив: название, преподаватель, день недели и количество записавшихся студентов. Ниже приведён фрагмент кода, иллюстрирующий объявление структуры:

```
struct Information {  
    char facultativ[50];  
    char prepod[50];  
    char day[20];  
    int count_of_students;  
};
```

Тип `char[]` выбран для хранения строк (названия, ФИО преподавателя и дня недели), поскольку он легко преобразуется в последовательность байтов при записи в бинарный файл.

Тип `int` использован для количества студентов, так как это целочисленный числовой параметр, не требующий дробной части.

Использование структуры `Information` позволило логически объединить все сведения о факультативе в единый элемент. Это упростило реализацию алгоритмов сортировки и поиска, а также обеспечило прямую запись и чтение данных в бинарный файл.

Четырьмя основными типами структур данных являются [2]:

- линейные структуры данных;
- древовидные структуры данных;
- хеш-структуры данных;
- графовые структуры данных.

В данной работе применяется линейная структура данных – массив структур, что обусловлено удобством последовательного размещения информации о факультативных занятиях. Такой способ хранения позволяет эффективно реализовать сортировку и поиск данных по различным параметрам.

Структуры данных представляют собой взаимосвязи данных и предоставляют методы эффективной сортировки и доступа к данным.

1.2 Файлы

При работе с файлами важно понимать, что они могут иметь различную структуру хранения данных. В зависимости от способа представления и обработки информации файлы подразделяются на *текстовые* и *бинарные*. Каждый тип имеет свои особенности и применяется в различных задачах программирования. Ниже приведены основные отличия между этими видами файлов:

Текстовый	Бинарный
<ul style="list-style-type: none">– последовательность символов– редактируется с помощью текстовых редакторов	<ul style="list-style-type: none">– последовательность байтов– редактируется с помощью двоичных редакторов

Рисунок 1.1 – Виды файлов

Для хранения данных о факультативах используется бинарный файл, что позволяет более эффективно сохранять и обрабатывать информацию, особенно при работе с большими объёмами данных. Работа с файлом осуществляется с помощью стандартных потоков ввода-вывода `fstream` в режиме `binary`.

В данной системе структура `Information`, содержащая сведения о факультативе, записывается и считывается из бинарного файла как единый блок памяти. Такой подход минимизирует издержки на преобразование данных и обеспечивает высокую скорость обработки.

Основные функции работы с файлами [1]:

- открытие файла (прежде чем можно будет работать с файлом, его необходимо открыть, эта операция устанавливает связь между программой и файлом на диске);
- чтение из файла;
- запись в файл;
- закрытие файла (эта операция освобождает ресурсы, связанные с файлом, и гарантирует, что все данные, записанные в файл, будут сохранены на диске).

2 АЛГОРИТМЫ СОРТИРОВКИ

Сортировкой называется процесс упорядочивания множества объектов по какому-либо признаку. *Алгоритм сортировки* – это алгоритм для упорядочивания элементов в списке. В случае, когда элемент в списке имеет несколько полей, поле, служащее критерием порядка, называется *ключом сортировки*.

В данной курсовой работе было несколько видов сортировок по следующим критериям:

- быстрая сортировка по дню недели;
- сортировка выбором по количеству студентов;
- сортировка вставками по названию факультатива.

Выбор конкретного алгоритма сортировки зависит от нескольких факторов, таких как:

- объём обрабатываемых данных;
- стабильность сортировки (важна, если нужно сохранять относительный порядок элементов с одинаковыми значениями ключа);
- сложность реализации;
- тип данных, которые нужно отсортировать.

Разные алгоритмы имеют различную временную сложность, то есть оцениваются по тому, сколько операций сравнения и перестановки выполняется в зависимости от числа элементов.

Алгоритм	средняя сложность
<ul style="list-style-type: none">• сортировка выбором• сортировка вставками• быстрая сортировка	<ul style="list-style-type: none">• $O(n^2)$• $O(n^2)$• $O(n \log n)$

Рисунок 2.1 – Сложность алгоритмов сортировок

Для реализации сортировки в курсовой работе были использованы несколько классических алгоритмов:

- быстрая сортировка;
- сортировка выбором;
- сортировка вставками.

Каждый из этих методов имеет свои преимущества и недостатки, которые будут рассмотрены далее.

2.1 Быстрая сортировка

Быстрая сортировка (Quicksort) – это один из самых эффективных и широко используемых алгоритмов сортировки. Суть метода заключается в выборе опорного элемента (*pivot*), после чего массив делится на две части: элементы меньше опорного и элементы больше него. Затем процесс повторяется рекурсивно для каждой из частей.

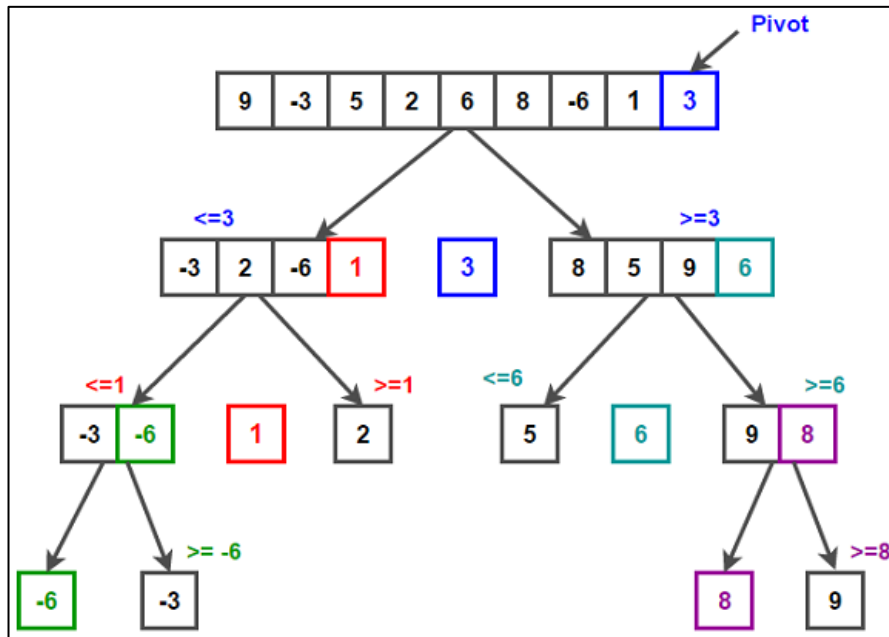


Рисунок 2.2 – Пример работы быстрой сортировки

В рамках данной работы быстрая сортировка была использована для упорядочивания записей факультативов *по дню недели*. Это позволило сгруппировать занятия в логической последовательности от понедельника к воскресенью. Для этого каждая строка поля *day* преобразовывалась в числовой индекс с помощью вспомогательной функции `getDayIndex()`, что обеспечивало корректное сравнение значений. Реализация данной функции находится в приложении А.

Алгоритм сортировки [6]:

- выбираем опорный элемент из массива по своему усмотрению. Для простоты предположим, что последний элемент — опорный элемент;
- массив разбивается на две части: элементы, меньшие *pivot*, и элементы, большие или равные *pivot*;
- используются два указателя (один движется по массиву, второй отслеживает позицию для следующего меньшего элемента);
- при проходе по массиву, если элемент *меньше pivot*, он обменивается местами с элементом на позиции второго указателя;
- после завершения прохода, *pivot* меняется местами с элементом, стоящим после последнего меньшего элемента. Таким образом, *pivot* встанет на

своё отсортированное место;

- алгоритм вызывается рекурсивно для левого и правого подмассивов (относительно позиции *pivot*).

Эффективен при работе с большими наборами данных.

2.2 Сортировка выбором

Сортировка выбором (Selection Sort) – это простой алгоритм сортировки, работающий по следующему принципу: на каждом шаге из неотсортированной части массива выбирается минимальный (или максимальный) элемент и меняется местами с первым элементом этой части. Процесс повторяется, сдвигая границу отсортированной части массива на один элемент вправо, пока весь массив не будет упорядочен.

Алгоритм сортировки:

- Находим минимальный элемент в неотсортированной части массива;
- Меняем местами найденный минимальный элемент с первым элементом неотсортированной части массива.
- Повторяем шаги 1 и 2 для оставшейся неотсортированной части массива.

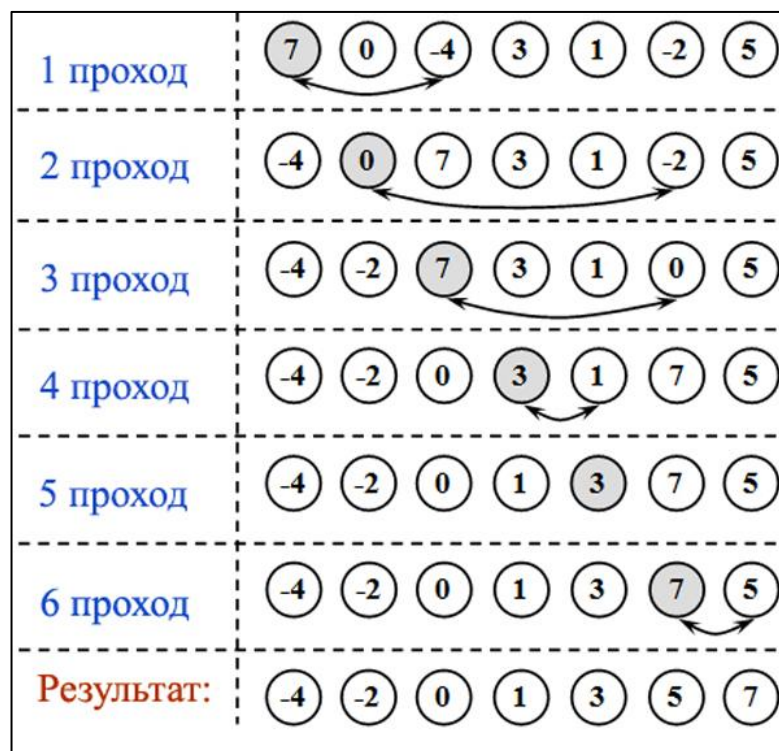


Рисунок 2.3 –Пример работы сортировки выбором

В рамках курсовой работы сортировка выбором применялась для упорядочивания факультативов по количеству записавшихся студентов. Это позволило легко выделять факультативы с наибольшей и наименьшей посещаемостью.

Реализация метода сортировки выбора на C++:

```
void SelectionSort(int k, int x[max])
{
    int i, j, min, temp;
    for (i = 0; i < k - 1; i++)
    {
        min = i;
        for (j = i + 1; j < k; j++)
        {
            if (x[j] < x[min])
                min = j;
        }
        temp = x[i];
        x[i] = x[min];
        x[min] = temp;
    }
}
```

Сортировка выбором проста в понимании и реализации, что делает ее хорошим выбором для новичков или в ситуациях, когда простота важнее эффективности [11].

2.3 Сортировка вставками

Сортировка вставками (Insertion Sort) – это простой алгоритм сортировки, работающий по принципу, аналогичному сортировке карт в руке: каждый новый элемент сравнивается с уже отсортированными и вставляется на своё место. Стоит отметить, что массив из 1-го элемента считается отсортированным [3].

Алгоритм проходит по массиву, начиная со второго элемента. Для каждого элемента определяется позиция, на которую он должен быть вставлен среди уже упорядоченных элементов слева.

Алгоритм:

- начинаем сортировку с первого элемента, считая его уже отсортированным;
- сравниваем текущий элемент с предыдущим;
- сдвигаем элементы, большие текущего, вправо;
- вставляем текущий элемент на подходящее место;
- повторяем это действие для оставшихся элементов списка, переставляя каждый элемент на правильное место;

В рамках работы сортировка вставками использовалась для упорядочивания записей по названию факультатива. При сортировке по названиям факультативов особенно важно обеспечить корректную работу с текстовыми данными, так как даже незначительное отличие в написании (например, пробел или различие в регистрах) может повлиять на порядок

элементов. Сравнение строк осуществлялось с помощью стандартной функции `strcmp()`, которая возвращает значение, определяющее лексикографический порядок строк. Функция `strcmp()` чувствительна к регистру, поэтому это стоит учитывать при вводе данных.



Рисунок 2.4 –Пример работы сортировки вставками

Реализация метода сортировки вставками на C++:

```
void InsertionSort(int arr[], int n) {
    int i, key, j;
    for (i = 1; i < n; i++) {
        key = arr[i];
        j = i - 1;
        while (j >= 0 && arr[j] > key) {
            arr[j + 1] = arr[j];
            j = j - 1;
        }
        arr[j + 1] = key;
    }
}
```

Сортировка вставками эффективна для сортировки небольших наборов данных или списков с небольшим количеством элементов.

3 АЛГОРИТМЫ ПОИСКА

Поиск – это один из ключевых алгоритмических процессов, позволяющий находить нужную информацию в массиве данных по заданному критерию. Эффективность поиска особенно важна в системах, работающих с большими объёмами данных, таких как система учёта факультативных занятий. Выбор конкретного алгоритма поиска напрямую зависит от структуры и состояния данных – были ли они отсортированы или нет.

3.1 Линейный поиск

Линейный поиск – это базовый метод, при котором последовательно проверяется каждый элемент массива [4]. Слово «последовательный» содержит в себе основную идею метода. Начиная с первого, все элементы массива последовательно просматриваются и сравниваются с искомым. Если на каком-то шаге текущий элемент окажется равным искомому, тогда элемент считается найденным, и в качестве результата возвращается номер этого элемента, либо другая информация о нем. Далее, в качестве выходных данных будет выступать номер элемента. Иначе, следуют вернуть что-то, что может оповестить о его отсутствии в пройденной последовательности.

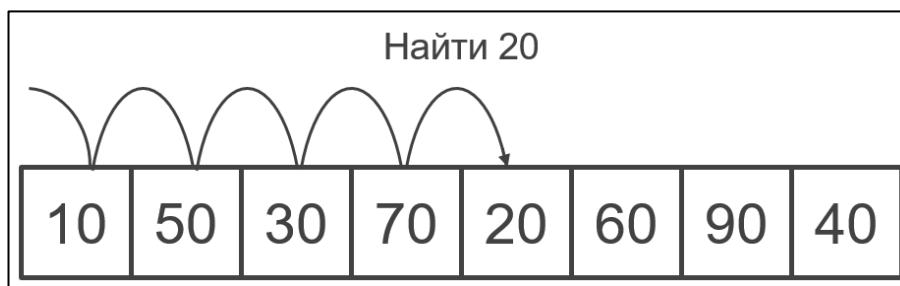


Рисунок 3.1 –Пример работы линейного поиска

В рамках работы реализован линейный поиск факультатива по названию в бинарном файле. Этот алгоритм перебирает каждую запись, сравнивая поле `facultativ` с искомым значением. Сравнение строк осуществлялось с помощью стандартной функции `strcmp()`, которая возвращает значение, определяющее лексикографический порядок строк. Реализация данной функции находится в приложении А.

Преимущество этого подхода – простота реализации и независимость от предварительной сортировки данных [4]. Однако его основным недостатком является линейная сложность – в худшем случае требуется перебрать все записи в файле, что при больших объёмах данных может быть неэффективно.

Несмотря на это, линейный поиск остаётся актуальным для небольших объёмов данных, а также в тех случаях, когда необходимо выполнить однократную проверку без дополнительных затрат на организацию структуры

хранения или индексирования. Он также может применяться как базовый алгоритм в комбинированных решениях — например, для предварительной фильтрации данных перед более сложной обработкой.

3.2 Бинарный поиск

Бинарный поиск — это эффективный алгоритм поиска элемента в отсортированном массиве. В отличие от линейного поиска, он работает по принципу деления массива пополам на каждом этапе, что позволяет значительно сократить количество сравнений.

Алгоритм работает только с *отсортированными данными*. В контексте данной работы бинарный поиск применяется для поиска факультатива по фамилии преподавателя, что требует предварительной сортировки записей по этому полю [4].

Алгоритм:

- выбирается середина массива;
- сравнивается ключ поиска с элементом в середине: если совпадает — элемент найден, если искомый элемент меньше — поиск продолжается в левой половине, если больше — в правой половине.
- процесс повторяется до нахождения элемента или пока границы не сомкнутся.

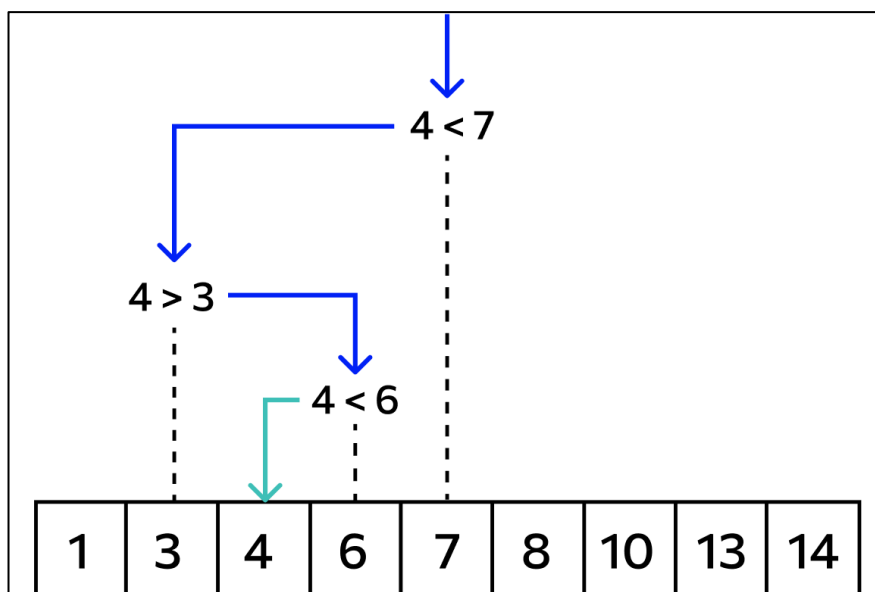


Рисунок 3.2 –Пример работы бинарного поиска

В курсовой работе бинарный поиск реализован для поиска преподавателя. Перед его выполнением массив структур сортируется по полю `prepod`, после чего проводится итеративный бинарный поиск. Сравнение строк осуществлялось с помощью стандартной функции `strcmp()`, которая возвращает значение, определяющее лексикографический порядок строк.

Реализация бинарного поиска на C++:

```
int binarySearch(int arr[], int size, int key) {  
    int left = 0, right = size - 1;  
    while (left <= right) {  
        int mid = (left + right) / 2;  
        if (arr[mid] == key)  
            return mid;  
        else if (arr[mid] < key)  
            left = mid + 1;  
        else  
            right = mid - 1;  
    }  
    return -1;  
}
```

Этот алгоритм особенно эффективен при большом объёме данных, где линейный поиск становится слишком затратным по времени [4].

4 ПОЛЬЗОВАТЕЛЬСКИЕ ФУНКЦИИ

В рамках данного курсовой работы была реализована совокупность пользовательских функций, обеспечивающих полноценную работу с данными о факультативных занятиях. Эти функции отвечают за ключевые операции: создание и инициализацию файлов, добавление новых записей, просмотр всей информации, а также вывод отдельных элементов на экран и в отчёт.

4.1 Функция `writeToReport()`

Функция `writeToReport()` предназначена для форматированной записи информации о факультативе в текстовый отчёт. Она принимает параметры:

- `ofstream& report` – ссылка на выходной файловый поток;
- `const char* facultativ, prepod, day` – строки, содержащие соответствующую информацию;
- `int count_of_students` – количество записавшихся студентов.

Внутри функции используется форматированный вывод с выравниванием (через `setw()` и `left`) для наглядного отображения данных в таблице отчёта. Это улучшает читаемость отчёта и упрощает анализ информации. Функция возвращает `void`, то есть не возвращает значений, но выполняет только вывод.

4.2 Функция `viewInformation()`

Функция `viewInformation()` принимает в качестве аргумента строку `filename`, содержащую имя бинарного файла, и открывает этот файл для чтения. Для работы с файлом используется объект `ifstream`, созданный с флагом `ios::binary`, так как данные в файле представлены в бинарном формате. Параллельно открывается файл `report.txt` для дополнения отчета – для этого используется объект `ofstream` с флагом `ios::app`, позволяющим добавлять данные в конец файла без перезаписи уже существующего содержимого. Если файл с переданным именем невозможно открыть, программа выводит сообщение об ошибке в консоль и завершает выполнение функции. При успешном открытии файла выводится заголовок таблицы, представляющей структуру данных о факультативах. Основная часть функции реализована в виде цикла, в котором из бинарного файла последовательно считываются записи типа `Information`. Каждая считанная структура отображается в консоли в виде строки таблицы с использованием манипуляторов форматирования `setw` и `left`, которые обеспечивают выравнивание данных по столбцам. Для дублирования информации в отчет используется вспомогательная функция `writeToReport()`, которая получает параметры текущей записи и сохраняет их в текстовом файле. По завершении чтения всех записей оба файла закрываются вызовом метода `close()` для

каждого из потоков. Функция возвращает `void`, то есть не возвращает значений, но выполняет вывод нужных полей.

4.3 Функция `createBinaryFile()`

Функция `createBinaryFile()` создаёт новый бинарный файл для хранения информации о факультативах. Принимает один параметр: `const char* filename` – имя создаваемого файла. Для реализации используется объект `ofstream`, открываемый с флагами `ios::out | ios::binary`, что позволяет создать файл в бинарном режиме. Сначала файл открывается и сразу закрывается, что инициализирует его как пустой. Параллельно открывается файл `report.txt` с флагом `ios::app`, позволяющим добавлять записи в конец файла без удаления уже имеющихся данных. Далее в `report.txt` добавляется запись о создании нового файла. Функция не возвращает значения, но выводит пользователю сообщение о завершении операции.

4.4 Функция `file_create()`

Функция `file_create()` создаёт или очищает файл отчёта `report.txt`. Это вспомогательная функция, вызываемая один раз при запуске программы для подготовки чистого отчётного файла. Она открывает `report.txt` в режиме `ios::out`, что приводит к удалению его прежнего содержимого, и сразу закрывает файл. Функция не возвращает значения, но выводит пользователю сообщение о завершении операции.

4.5 Функция `viewOneInformation()`

Функция `viewOneInformation()` выводит на экран информацию об одном факультативе в отформатированном виде. Параметры: `Information info` – структура с информацией о факультативе; `int in` – флаг, указывающий, нужно ли выводить заголовки таблицы (если `in == 0`, заголовки выводятся). Функция предназначена как для одиночного вывода, так и для повторного использования внутри других алгоритмов (например, поиска). Функция не возвращает значения.

4.6 Функция `viewOneInformationReport()`

Эта функция записывает в `report.txt` данные об одном факультативе в текстовом формате. Используется, например, после успешного поиска факультатива. Принимает: `Information info` – структура с информацией; `ofstream& report` – поток отчётного файла. Запись осуществляется с поясняющими подписями для каждого поля. Функция не возвращает значения.

4.7 Функция `addInformation()`

Функция `addInformation()` позволяет пользователю добавить новую запись о факультативе в бинарный файл. Функция не возвращает значения. Параметры: `const char* filename` – имя файла для добавления; `Information records[]` – массив структур для хранения текущих записей; `int& count` – текущее количество записей (изменяется в процессе). Перед выполнением добавления производится проверка: если количество записей достигло максимального предела (`max_records`), функция выводит сообщение и прекращает выполнение. В случае допустимого количества записей пользователь поочередно вводит данные о факультативе: название, имя преподавателя, день недели и количество студентов. Эти данные сохраняются в структуру `plan`. Открытие бинарного файла происходит с флагами `ios::app` | `ios::binary`, что позволяет дописывать записи в конец файла без удаления существующих. После успешной записи новой структуры в файл, она также добавляется в массив `records`, и счетчик `count` увеличивается на единицу. После этого в `report.txt` фиксируется факт добавления новой записи.

4.8 Функция `deleteFacultativ()`

Функция `deleteFacultativ()` предназначена для удаления записи о факультативе из бинарного файла. Функция не возвращает значения. В качестве параметра она принимает строку `filename`, указывающую имя файла, из которого следует удалить запись. После открытия исходного файла и временного файла, в который будут сохраняться все записи, кроме удаляемой, пользователь вводит название факультатива, который необходимо удалить. Далее происходит поочередное считывание записей из исходного файла. Каждая считанная запись сравнивается по названию факультатива с введенным пользователем значением. Если название не совпадает, запись копируется во временный файл. Если совпадает – она не записывается, тем самым удаляется из данных. По завершении чтения исходный файл закрывается. Если целевая запись была найдена и удалена, исходный файл удаляется, а временный файл переименовывается в его имя. При этом в `report.txt` добавляется сообщение об удалении факультатива. Если запись не найдена, временный файл удаляется без изменений, и выводится соответствующее сообщение.

4.9 Функция `edit()`

Функция `edit()` предназначена для редактирования существующей записи о факультативе в бинарном файле. Функция не возвращает значения. Принимает один параметр: `const char* filename` – имя файла. Сначала пользователь вводит название факультатива, данные о котором требуется изменить. Функция открывает основной файл для чтения и временный файл

для записи. Затем происходит последовательное считывание всех записей. Если имя текущего факультатива совпадает с введенным пользователем, функция предлагает ввести новые значения для преподавателя, дня недели и количества студентов. Эти новые данные сохраняются в структуру `plan`, которая затем записывается во временный файл. Если совпадение не найдено, текущая запись копируется во временный файл без изменений. По завершении обработки всех записей исходный файл закрывается и удаляется. Временный файл переименовывается в исходное имя, только если редактируемый факультатив был найден. При этом в файл отчёта `report.txt` заносится сообщение об успешном редактировании.

4.10 Функция `LineSearchFacultativ()`

Функция `LineSearchFacultativ()` реализует линейный поиск записи о факультативе в бинарном файле по названию. Функция возвращает `void`, то есть не возвращает значений, а выполняет поиск. В качестве параметра она принимает строку `filename`, содержащую имя файла, в котором осуществляется поиск. Сначала пользователь вводит название факультатива, которое требуется найти. Функция открывает указанный файл для чтения в бинарном режиме, а также файл отчёта `report.txt` в режиме дозаписи. Если открыть хотя бы один из файлов не удастся, выполнение функции завершается с сообщением об ошибке. Затем осуществляется последовательное считывание записей. Каждая запись сравнивается с введенным названием факультатива. В случае совпадения информация о найденной записи выводится пользователю и сохраняется в файл отчёта через вспомогательную функцию `viewOneInformationReport()`. Если совпадений не обнаружено, в отчёт записывается соответствующее уведомление. Функция завершается закрытием всех файлов.

4.11 Функция `binarySearchByPrepod()`

Функция `binarySearchByPrepod()` предназначена для выполнения бинарного поиска записей о факультативных занятиях по фамилии преподавателя. Функция возвращает `void`, то есть не возвращает значений, а выполняет поиск. Принимает один параметр: `const char* filename` – имя файла. Сначала функция открывает файл в бинарном режиме и считывает данные в массив структур `arr`. Чтение выполняется до достижения конца файла или максимально допустимого количества записей. Если файл пуст, функция завершает выполнение с соответствующим сообщением. После этого пользователь вводит фамилию преподавателя, которую требуется найти. Далее осуществляется бинарный поиск по массиву записей, предполагая, что данные уже *отсортированы по фамилии преподавателя*. Если нужная запись найдена, сохраняется индекс найденного элемента. Функция также записывает результаты поиска в файл отчета `report.txt`. Если запись с указанной

фамилией не найдена, об этом выводится сообщение как в консоль, так и в отчет. В случае успеха осуществляется перебор всех записей в массиве, совпадающих по фамилии преподавателя, и для каждой выводится подробная информация на экран и в отчет с помощью вспомогательной функции `viewOneInformationReport()`. Функция завершается закрытием файла отчета.

4.12 Функция `sortbychoice()`

Функция `sortbychoice()` предназначена для сортировки записей факультативов по количеству записавшихся студентов методом выбора. В качестве аргумента принимает строку `filename`, содержащую имя бинарного файла, из которого загружаются данные. Сначала функция открывает указанный файл и считывает все записи в массив структур `records[]`. После успешного считывания выполняется сортировка методом выбора: на каждой итерации находится элемент с минимальным количеством студентов и меняется местами с текущим. Это гарантирует, что записи будут упорядочены по возрастанию количества студентов. После завершения сортировки данные записываются обратно в тот же файл, перезаписывая его содержимое. Если запись в файл проходит успешно, на экран выводится соответствующее сообщение. Также функция записывает отсортированные данные в файл `report.txt` с помощью функции `viewOneInformationReport()`, чтобы зафиксировать изменения в отчете. Если файл отчета не удастся открыть, пользователю выводится сообщение об ошибке. Функция возвращает `void`, то есть не возвращает значений, а выполняет сортировку.

4.13 Функция `insertionSortByFacultativ()`

Функция `insertionSortByFacultativ()` реализует сортировку записей факультативов по названию с использованием метода сортировки вставками. Принимает один параметр – строку `filename`. Сначала функция открывает указанный файл для чтения и считывает все записи в массив `records[]`. Если файл открыть не удалось, пользователю выводится сообщение об ошибке. После успешного считывания данных применяется алгоритм сортировки вставками: каждый элемент вставляется в отсортированную часть массива на нужную позицию на основе лексикографического сравнения названий факультативов с помощью функции `strcmp`. Затем исходный файл открывается заново в режиме перезаписи. Отсортированные данные записываются обратно в файл, полностью заменяя его содержимое. Дополнительно функция открывает файл отчета `report.txt` и фиксирует в нем факт сортировки. Функция завершается закрытием всех файлов. Функция возвращает `void`, то есть не возвращает значений, а выполняет сортировку.

4.14 Функция `getDayIndex()`

Функция `getDayIndex()` предназначена для определения номера дня недели на основе переданной строки с его названием. Принимает один параметр – строку `day` (в виде `const char*`). Эта функция используется как вспомогательная для осуществления корректной работы быстрой сортировки. Функция поочередно сравнивает значение `day` с предопределёнными строками, соответствующими дням недели, и возвращает значения согласно порядку дня. Если строка не совпадает ни с одним из предусмотренных значений, функция возвращает `-1`, сигнализируя о том, что день недели не был распознан.

4.15 Функция `quickSortByDay()`

Функция `quickSortByDay()` реализует алгоритм быстрой сортировки (Quick Sort) для массива структур `Information` по дням недели. Принимает три параметра: массив записей `arr[]`, а также индексы начала (`left`) и конца (`right`) диапазона сортировки. Функция использует рекурсивный подход. В качестве опорного элемента (`p`) берется первый элемент текущего диапазона. Далее выполняется разбиение массива: элементы с днями недели, предшествующими опорному (по порядку дней недели), перемещаются влево, а элементы, следующие за опорным – вправо. Для определения порядка дней недели используется вспомогательная функция `getDayIndex()`, которая преобразует название дня в целочисленный индекс. Функция не возвращает значения, только выполняет сортировку.

4.16 Функция `quickSortByPrepod()`

Функция `quickSortByPrepod()` реализует алгоритм быстрой сортировки (Quick Sort) для массива структур `Information` по фамилии преподавателя. Принимает три параметра: массив записей `arr[]`, а также индексы начала (`left`) и конца (`right`) диапазона сортировки. Функция использует рекурсивный подход. В качестве опорного элемента (`pivot`) берется средний элемент текущего диапазона. Далее выполняется разбиение массива: все элементы, чьи фамилии преподавателей лексикографически меньше опорной, перемещаются в левую часть массива, а большие – в правую. Функция не возвращает значения, только выполняет сортировку. Эта функция была *реализована для выполнения бинарного поиска* по преподавателю, т.к. массив должен быть отсортирован перед поиском.

4.17 Функция `SortInformation()`

Функция `SortInformation()` выполняет сортировку записей, содержащихся в бинарном файле, по дням недели или по фамилии

преподавателя, в зависимости от переданного флага. Она принимает два аргумента: строку `filename`, содержащую имя файла, и логическое значение `sortByDay`, определяющее тип сортировки. Если значение `sortByDay` истинно (`true`), сортировка выполняется по дню недели, в противном случае – по фамилии преподавателя. Сортировка производится над массивом `records`, в котором содержатся считанные из файла данные. После сортировки в консоль выводится сообщение о том, какой тип сортировки был выполнен. В начале функция пытается открыть указанный файл в бинарном режиме для чтения. Если файл не удастся открыть, выводится сообщение об ошибке, и выполнение прекращается. Далее содержимое файла считывается в массив структур `Information`, пока не достигнут конец файла или максимально допустимое количество записей. Если файл пустой и в массив ничего не считано, функция завершает выполнение, предварительно уведомив об этом пользователя. Также функция формирует отчет об отсортированных данных в текстовом файле `report.txt`. Каждая запись передаётся во вспомогательную функцию `viewOneInformationReport()`, которая оформляет и записывает информацию о ней в файл отчета. По завершении все файлы закрываются. Функция не возвращает значения.

4.18 Функция `searchByPrepodAndDay()`

Функция `searchByPrepodAndDay` предназначена для поиска и отображения факультативов по фамилии преподавателя и дню недели. Параметры: `const char* filename` – имя бинарного файла, содержащего записи факультативов. Функция возвращает `void`, то есть не возвращает значений, а выполняет поиск, выводит информацию на экран и записывает результаты в отчет. Открывается бинарный файл `filename` для чтения с помощью потока `ifstream`. Если открыть файл не удастся, выводится сообщение об ошибке, и функция завершается. Содержимое файла считывается в массив `records[]` до достижения конца файла. Пользователю предлагается ввести фамилию преподавателя и день недели для поиска соответствующих факультативов. Ввод обрабатывается через функции `cin.getline()` для строк `teacher` и `day`. Для каждой записи из массива `records[]` проверяется, совпадают ли фамилия преподавателя и день недели с введенными значениями. Все подходящие записи сохраняются в массив `filteredRecords[]`. Если подходящих записей не найдено, выводится сообщение об этом, и функция завершает выполнение. Все подходящие записи сортируются по количеству студентов в факультативе в убывающем порядке с использованием алгоритма сортировки пузырьком (по количеству студентов – `count_of_students`). Открывается файл отчета `report.txt` для добавления данных. Если открыть файл отчета не удастся, выводится сообщение об ошибке, и функция завершает выполнение. В отчет записываются результаты поиска, включая фамилию преподавателя и день недели. Также выводится таблица с названиями факультативов, фамилиями преподавателей, днями

недели и количеством студентов. По завершении записи отчета файл закрывается, и пользователю сообщается, что результаты поиска записаны в отчет.

4.19 Функция `printStatsStatistics()`

Функция `printStatsStatistics` предназначена для вывода статистики по факультатам, сгруппированной по преподавателям, а также для записи этой информации в файл отчета `report.txt`. Параметры: `const char* filename` – имя бинарного файла, содержащего записи факультативов. Функция возвращает `void`, то есть не возвращает значений. Сначала функция пытается открыть указанный бинарный файл на чтение. Если файл открыть не удастся, пользователю выводится сообщение об ошибке, и функция завершает выполнение. Если файл успешно открыт, его содержимое считывается в массив `records[]` структур типа `Information` до достижения конца файла. После этого файл закрывается. Затем открывается файл `report.txt` в режиме добавления. Если файл отчета открыть не удастся, выводится соответствующее сообщение, и функция также завершает выполнение. В случае успешного открытия в начало отчета записывается заголовок таблицы со статистикой, включающей поля: фамилия преподавателя, название факультатива, день недели и количество записавшихся студентов. Основная часть функции заключается в группировке данных по преподавателям. Для каждого преподавателя, еще не обработанного ранее, формируется отдельный блок статистики. Для этого осуществляется двойной цикл: внешний цикл проходит по всем записям, а внутренний проверяет, встречался ли преподаватель раньше. Если преподаватель не встречался, формируется список всех его факультативов. Эти записи копируются во временный массив `facultyList[]`, а затем сортируются по количеству студентов с помощью функции `sortByStudents`. После сортировки каждая запись выводится в консоль и одновременно записывается в файл отчета с помощью функции `writeToReport`. Таким образом, для каждого преподавателя выводится отсортированный список его факультативов. По завершении всей обработки файл отчета закрывается. В результате работы функция предоставляет пользователю сгруппированные и отсортированные данные по преподавателям как в консоли, так и в отчете.

4.20 Функция `popular_facultative()`

Функция `popular_facultative()` предназначена для определения самых популярных факультативов на основе общего количества студентов, записавшихся на каждый из них. В качестве параметра она принимает строку `filename`, указывающую имя бинарного файла, в котором хранятся записи о факультативах. Функция не возвращает значения. Сначала функция открывает указанный файл в бинарном режиме для чтения и файл отчета `report.txt` в

режиме добавления. Если входной файл не удастся открыть, пользователю выводится сообщение об ошибке, и выполнение прерывается. Далее из файла последовательно считываются записи типа `Information` и сохраняются в массив `arr`. Подсчитывается количество успешно прочитанных записей, не превышающее `max_records`. Если файл пуст, программа сообщает об этом и завершает работу. Затем формируется список уникальных названий факультативов в массиве `facultatives`, параллельно с массивом `students`, в котором подсчитывается общее количество студентов для каждого уникального факультатива. Если факультатив уже встречался ранее, к его количеству студентов добавляется новое значение. Если нет – он добавляется в список. После завершения подсчета определяется максимальное значение среди всех факультативов. В консоль и файл отчета выводится сводка по каждому факультативу и числу студентов. Затем отдельно перечисляются самые популярные факультативы, имеющие максимальное количество студентов. Функция завершает выполнение, закрыв файл отчета, если он был успешно открыт.

4.21 Функция `menu()`

Функция `menu()` реализует пользовательский интерфейс для взаимодействия с программой, работающей с данными о факультативах. Она предлагает пользователю выбрать одно из действий через числовое меню. Все действия выполняются в бесконечном цикле `do...while`, который продолжается до тех пор, пока пользователь не выберет пункт 14 – выход из программы. Функция не возвращает значения.

4.22 Функция `main()`

Функция `main()` просто инициализирует локаль и кодировки для поддержки русского языка и запускает меню, откуда уже выполняются все остальные функции.

5 ОПИСАНИЕ РАБОТЫ ПРОГРАММЫ

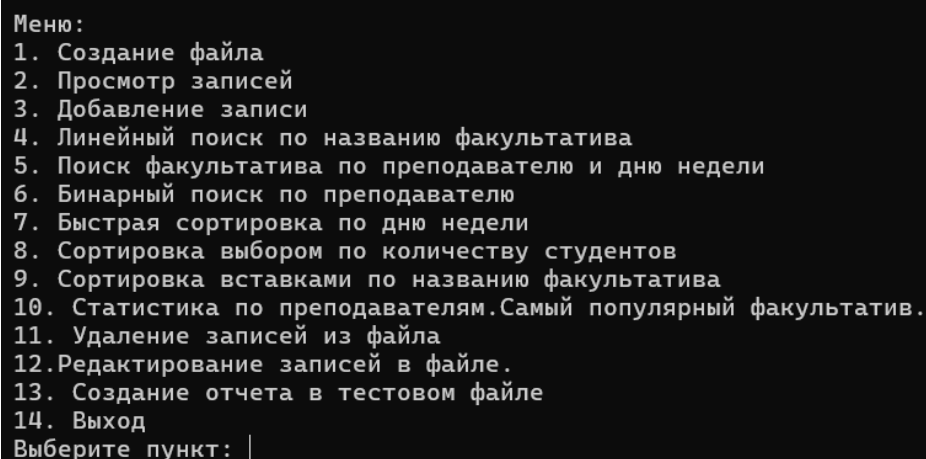
Данная программа предназначена для ведения и обработки данных о факультативах. Она реализует функции создания и редактирования записей, поиска, сортировки, анализа и формирования отчётной информации. После запуска пользователю отображается текстовое меню с номерами пунктов и кратким описанием каждой функции.

Основные функции программы:

- создание файла;
- просмотр записей;
- добавление записи;
- линейный поиск по названию факультатива;
- поиск по преподавателю и дню недели;
- бинарный поиск по преподавателю;
- быстрая сортировка по дню недели;
- сортировка выбором по количеству студентов;
- сортировка вставками по названию факультатива;
- статистика по преподавателям и самый популярный факультатив;
- удаление записей;
- редактирование записей;
- создание отчета в текстовом файле;
- выход из программы.

В этой главе будет разобрана программа с точки зрения пользователя.

Первое, что увидит пользователь – это меню программы, ход программы будет зависеть от выбора пункта в меню.



```
Меню:
1. Создание файла
2. Просмотр записей
3. Добавление записи
4. Линейный поиск по названию факультатива
5. Поиск факультатива по преподавателю и дню недели
6. Бинарный поиск по преподавателю
7. Быстрая сортировка по дню недели
8. Сортировка выбором по количеству студентов
9. Сортировка вставками по названию факультатива
10. Статистика по преподавателям.Самый популярный факультатив.
11. Удаление записей из файла
12.Редактирование записей в файле.
13. Создание отчета в тестовом файле
14. Выход
Выберите пункт: |
```

Рисунок 5.1 –Меню программы

При выборе второго пункта на экран будет выведен список факультативов. Вот пример, как он будет выглядеть:

Система планирования факультативных занятий в университете:			
Факультатив	Преподаватель	День недели	Кол-во записавшихся студентов
лаиаг	князюк	понедельник	15
матем	князюк	понедельник	24
твимс	гуревич	пятница	4

Рисунок 5.2 –Вывод списка факультативов

Третий пункт меню ведет к добавлению нового факультатива в список. Программа запросит необходимые данные. Вот как это будет выглядеть:

```

Выберите пункт: 3
Введите название факультатива:оаип
Введите фамилию преподавателя:новицкая
Введите день недели:среда
Введите количество записавшихся студентов:12
Информация добавлена успешно.
Информация о добавлении факультатива записана в отчет.

```

Рисунок 5.3 –Добавление факультатива

При выборе пункта «редактирование записей в файле» пользователю необходимо будет ввести название факультатива, который нужно отредактировать. Далее вводим информацию так же, как и при добавлении. Если факультатив с таким названием не единственный, то придется редактировать все записи с таким названием.

```

Выберите пункт: 12
Введите факультатив для редактирования: матем
Редактирование факультатива: матем
Введите преподавателя: князюк
Введите день недели: вторник
Введите кол-во записавшихся студентов: 13
Информация о факультативе обновлена.

```

Рисунок 5.4 –Редактирование записей

При выборе пунктов 7–9 в консоль не будет выводиться отсортированный список, а лишь сообщение, по какому параметру была выполнена сортировка. Также будет выведено сообщение о том, что данные были перезаписаны в файл и отчет. Для того чтобы получить полные результаты сортировки, пользователю будет необходимо воспользоваться функцией просмотра данных в файле. Это позволяет избежать перегрузки консоли лишней информацией, особенно при работе с большими наборами данных. Отчет сохраняется в текстовом формате, что обеспечивает его совместимость с большинством программ для обработки данных.

Система планирования факультативных занятий в университете:			
Факультатив	Преподаватель	День недели	Кол-во записавшихся студентов
твимс	гуревич	пятница	4
оаип	новицкая	среда	12
матем	князюк	вторник	13
лаиаг	князюк	понедельник	15

Рисунок 5.5 –Сортировка выбором по количеству студентов

Система планирования факультативных занятия в университете:			
Факультатив	Преподаватель	День недели	Кол-во записавшихся студентов
лаиаг	князюк	понедельник	15
матем	князюк	вторник	13
оаип	новицкая	среда	12
твимс	гуревич	пятница	4

Рисунок 5.6 –Быстрая сортировка по дню недели

Система планирования факультативных занятий в университете:			
Факультатив	Преподаватель	День недели	Кол-во записавшихся студентов
лаиаг	князюк	понедельник	15
матем	князюк	вторник	13
оаип	новицкая	среда	12
твимс	гуревич	пятница	4

Рисунок 5.7 –Сортировка вставками по названию факультатива

При выборе пунктов 4-6 будет выполнен поиск. Программа запросит ввести параметр, по которому будет искать. После выполнения, она выведет все сведения о факультативе. Если в списке несколько факультативов, с одинаковым параметром, то программа выведет все искомые факультативы. Если в списке нет искомого факультатива, то программа выведет предупреждающее сообщение. Также в консоль будет выведено сообщение, что результаты поиска записаны в отчет, а сам файл отчета будет отредактирован.

```

Введите название факультатива для поиска: твимс
Найден факультатив: твимс
Преподаватель: гуревич
День недели: пятница
Количество записавшихся студентов: 4

```

Рисунок 5.8 –Линейный поиск по названию факультатива

Перед выполнением бинарного поиска по преподавателю предварительно осуществляется сортировка данных по полю «преподаватель». Это необходимо для корректной работы алгоритма, так как бинарный поиск требует упорядоченной последовательности. Сразу после сортировки пользователю выводится отсортированный список, что позволяет визуально убедиться в правильности подготовки данных и точности работы программы.

```
Введите фамилию преподавателя для поиска: князюк
Преподаватель князюк найден.
Преподаватель: князюк
Предмет: матем
День: вторник
Количество студентов: 13

Преподаватель: князюк
Предмет: лаиаг
День: понедельник
Количество студентов: 15
```

Рисунок 5.9 –Бинарный поиск по преподавателю

```
Введите фамилию преподавателя: князюк
Введите день недели: понедельник
Факультативы преподавателя князюк в день понедельник:
Факультатив: лаиаг, количество студентов: 15
Результаты поиска записаны в отчет.
```

Рисунок 5.10 –Поиск факультатива по преподавателю и дню недели

Выбор 10 пункта приведет к выводу статистики по факультативам. Здесь факультативы будут сгруппированы по преподавателям и выведен весь список факультативов для каждого преподавателя в порядке убывания студентов, а также будет найден самый популярный факультатив (или несколько). Вот, как это будет выглядеть:

```
Преподаватель: гуревич
Факультатив: твимс, День: пятница, Количество студентов: 4
Преподаватель: князюк
Факультатив: лаиаг, День: понедельник, Количество студентов: 15
Факультатив: матем, День: вторник, Количество студентов: 13
Преподаватель: новицкая
Факультатив: оаип, День: среда, Количество студентов: 12
Статистика по факультативам:
-----
Факультатив: твимс, Количество студентов: 4
Факультатив: матем, Количество студентов: 13
Факультатив: лаиаг, Количество студентов: 15
Факультатив: оаип, Количество студентов: 12

Самые популярные факультативы (с максимальным количеством студентов: 15):
Факультатив: лаиаг, Количество студентов: 15
```

Рисунок 5.11 –Статистика по преподавателям и поиск самого популярного факультатива

При выборе пункта удаление записей из файла программа попросит ввести название для факультатива, который необходимо удалить, если факультативов с таким названием несколько, то программа удалит их все. После удаления будет выведено сообщение об окончании операции. Вот пример:



Выберите пункт: 11
Введите название факультатива для удаления: оаип
Факультатив удален.

Рисунок 5.12 –Удаление факультатива по названию

При выборе таких функций как «создание файла», «создание отчета в текстовом файле» и «выход» не требуют объяснений, т.к. при вызове этих функций программа лишь предупредит о их выполнении.

ЗАКЛЮЧЕНИЕ

В ходе выполнения курсовой работы была разработана консольная программа на языке C++ для автоматизации хранения, поиска, сортировки и анализа данных о факультативах, проводимых преподавателями. Основная цель работы заключалась в создании удобного инструмента для ведения и обработки информации о факультативах, позволяющего быстро находить нужные записи, сортировать их по различным критериям, а также формировать отчёт.

Программа реализует все необходимые функции: создание и редактирование бинарного файла с данными, добавление и удаление записей, поиск по различным параметрам (в том числе линейный и бинарный), сортировки по названию, преподавателю, дню недели и количеству студентов. Также реализованы функции статистического анализа, позволяющие определить наиболее популярные факультативы и построить сводные отчёты по преподавателям.

Работа программы была протестирована с различными наборами данных. В процессе тестирования были проверены все основные сценарии использования, в том числе граничные и исключительные ситуации – такие как отсутствие записей, некорректный ввод, попытка поиска несуществующих данных и другие. Программа показала стабильную работу, корректно обрабатывая все проверенные случаи.

В результате выполнения работы были решены задачи структурирования данных с использованием пользовательских структур, организации эффективного поиска и сортировки, работы с бинарными файлами, а также формирования текстовых отчётов.

Таким образом, поставленные в начале работы цели были успешно достигнуты, а задачи – выполнены.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

- [1] LabEx [Электронный ресурс]. – Режим доступа: <https://labex.io/ru/tutorials/cpp/>
- [2] Built in [Электронный ресурс]. – Режим доступа: <https://builtin.com/>
- [3] Habr [Электронный ресурс]. – Режим доступа: <https://habr.com/en/>
- [4] CodeLessons [Электронный ресурс]. – Режим доступа: <https://codelessons.dev/ru/>
- [5] CyberForum [Электронный ресурс]. – Форум программистов C++. – Электронные данные. – Режим доступа: <http://www.cyberforum.ru/cpp/>
- [6] КОД [Электронный ресурс]. – Режим доступа: <https://thecode.media/qsort/>
- [7] METANIT.COM C++ [Электронный ресурс]. – Режим доступа: <https://metanit.com/cpp/>
- [8] MSDN Microsoft Docs [Электронный ресурс]. – Режим доступа: <https://learn.microsoft.com/en-us/cpp/>
- [9] Структуры данных и алгоритмы. : Пер. с англ. : Уч. пос. – М. : Издательский дом "Вильямс", 2007. - 400 с. : ил. – Парал. тит. англ.
- [10] Кормен Т.Х., Лейзерсон Ч.И., Ривест Р.Л., Штайн К. Алгоритмы: построение и анализ / Кормен Т.Х., Лейзерсон Ч.И., Ривест Р.Л., Штайн К. – М.: МГТУ им. Н.Э. Баумана, 2016.
- [11] ScholarHat [Электронный ресурс]. – Режим доступа: <https://www.scholarhat.com/>

ПРИЛОЖЕНИЕ А
(обязательное)
Листинг кода

```
#include <iostream>
#include <fstream>
#include <windows.h>
#include <iomanip>

using namespace std;
const int max_records = 100;

struct Information {
    char facultativ [50];
    char prepod [50];
    char day [20];
    int count_of_students;
};

void writeToReport(ofstream& report, const char* facultativ,
const char* prepod, const char* day, int count_of_students) {
    report << left << setw(25) << facultativ << setw(25)
        << prepod << setw(25)
        << day << setw(45)
        << count_of_students << endl;
}

void viewInformation(const char* filename) {
    ifstream file(filename, ios::binary);
    ofstream report("report.txt", ios::app);
    Information plan;

    if (!file.is_open()) {
        cout << "Ошибка при открытии файла.\n";
        return;
    }

    cout << "\nСистема планирования факультативных занятий в
университете:\n";
    cout << "-----\n";
    cout << "-----\n";
    cout << left << setw(25) << "Факультатив" << setw(25)
        << "Преподаватель" << setw(25)
        << "День недели" << setw(45)
        << "Кол-во записавшихся студентов" << endl;
    cout << "-----\n";
    cout << "-----\n";
    report << "\nСистема планирования факультативных занятий в
университете:\n";
```


Продолжение приложения А

```
report << "-----\n";
report << left << setw(25) << "Факультатив" << setw(25)
    << "Преподаватель" << setw(25)
    << "День недели" << setw(45)
    << "Кол-во записавшихся студентов" << endl;
report << "-----\n";
while (file.read((char*)&plan, sizeof(plan))) {
    cout << left << setw(25) << plan.facultativ <<
setw(25)
    << plan.prepod << setw(25)
    << plan.day << setw(45)
    << plan.count_of_students << endl;
    writeToReport(report, plan.facultativ, plan.prepod,
plan.day, plan.count_of_students);
}
cout << "-----\n";
file.close();
report.close();
}
void createBinaryFile(const char* filename) {
    ofstream file, report;
    file.open(filename, ios::out | ios::binary);
    file.close();
    report.open("report.txt", ios::out | ios::app);
    report << " Вы создали новый файл\n";
    cout << "Файл создан\n";
}
void file_create() {
    ofstream out;
    out.open("report.txt", ios::out);
    out.close();
    cout << "Файл создан\n";
}
// Функция для вывода одного факультатива на экран
void viewOneInformation(Information info, int in) {
    if (in == 0) {
        cout << left << setw(20) << "Факультатив" << setw(20)
<< "Преподаватель" << setw(20)
        << "День недели" << setw(15) << "Количество
студентов" << endl;
    }
    cout << left << setw(20) << info.facultativ << setw(20)
<< info.prepod << setw(20) << info.day
    << setw(15) << info.count_of_students << endl; }
```

Продолжение приложения А

```
void viewOneInformationReport(Information info, ofstream&
report) {
    report << "\nФакультатив: " << info.facultativ <<
"\nПреподаватель: " << info.prepod
    << "\nДень недели: " << info.day << "\nКоличество
студентов: " << info.count_of_students << "\n";
}
void addInformation(const char* filename, Information
records[], int& count) {
    Information plan;
    if (count >= max_records) {
        cout << "Достигнуто максимальное количество
записей.\n";
        return;
    }
    cout << "Введите название факультатива:";
    cin.ignore();
    cin.getline(plan.facultativ, 50);
    cout << "Введите фамилию преподавателя:";
    cin.getline(plan.prepod, 50);
    cout << "Введите день недели:";
    cin.getline(plan.day, 20);
    cout << "Введите количество записавшихся студентов:";
    cin >> plan.count_of_students;
    ofstream file(filename, ios::app | ios::binary);
    if (file.is_open()) {
        file.write((char*)&plan, sizeof(plan));
        file.close();
        cout << "Информация добавлена успешно.\n";
        records[count++] = plan;
        ofstream report("report.txt", ios::app);
        if (report.is_open()) {
            report << "\nФакультатив \"" << plan.facultativ <<
"\n" << "был добавлен.\n";
            report.close();
            cout << "Информация о добавлении факультатива
записана в отчет.\n";
        }
        else {
            cout << "Ошибка при записи в отчет.\n";
        }
    }
    else {
        cout << "Ошибка при добавлении информации.\n";
        return;
    }
}
void LineSearchFacultativ(const char* filename) {
    ifstream file(filename, ios::binary);
    ofstream report("report.txt", ios::app);
```

Продолжение приложения А

```
if (!file.is_open() || !report.is_open()) {
    cout << "Ошибка при открытии файла." << endl;
    return;
}

Information plan;
char search[50];
bool found = false;

cout << "Введите название факультатива для поиска: ";
cin.ignore();
cin.getline(search, 50);
report << "\nПоиск факультатива по названию: " << search
<< "\n";
while (file.read(reinterpret_cast<char*>(&plan),
sizeof(plan))) {
    if (strcmp(plan.facultativ, search) == 0) {
        cout << "Найден факультатив: " << plan.facultativ
<< endl;

        cout << "Преподаватель: " << plan.prepod << endl;
        cout << "День недели: " << plan.day << endl;
        cout << "Количество записавшихся студентов: " <<
plan.count_of_students << endl;
        viewOneInformationReport(plan, report);
        found = true;
    }
}
if (!found) {
    cout << "Факультатив не найден." << endl;
    report << "Факультатив не найден.\n";
}

file.close();
report.close();
}

// Функция сортировки по количеству студентов и записи в
отчет
void sortbychoice(const char* filename) {
    ifstream file(filename, ios::binary);
    if (!file.is_open()) {
        cout << "Ошибка при открытии файла.";
        return;
    }

    Information records[max_records];
    int count = 0;
    while (file.read((char*)&records[count],
sizeof(Information)))
        count++;
}
file.close();
```

Продолжение приложения А

```
for (int i = 0; i < count - 1; i++) {
    int min_index = i;
    for (int j = i + 1; j < count; j++) {
        if (records[min_index].count_of_students >
records[j].count_of_students) {
            min_index = j;
        }
    }
    if (min_index != i) {
        Information temp = records[min_index];
        records[min_index] = records[i];
        records[i] = temp;
    }
}

ofstream outfile(filename, ios::binary);
if (outfile.is_open()) {
    for (int i = 0; i < count; i++) {
        outfile.write((char*)&records[i],
sizeof(Information));
    }
    outfile.close();
    cout << "Записи отсортированы по количеству
записавшихся студентов.\n";
}
else {
    cout << "Ошибка при записи в файл.\n";
}
ofstream report("report.txt", ios::app);
if (report.is_open()) {
    report << "\nЗаписи после сортировки по количеству
студентов:\n";

    for (int i = 0; i < count; i++) {
        viewOneInformationReport(records[i], report);
    }
    report.close();
    cout << "Данные перезаписаны в отчет.\n";
}
else {
    cout << "Ошибка при записи в отчет.\n";
}
}

int getDayIndex(const char* day) {
    if (strcmp(day, "понедельник") == 0) return 0;
    if (strcmp(day, "вторник") == 0) return 1;
    if (strcmp(day, "среда") == 0) return 2;

    if (strcmp(day, "четверг") == 0) return 3;
    if (strcmp(day, "пятница") == 0) return 4;
    if (strcmp(day, "суббота") == 0) return 5;
```

Продолжение приложения А

```
if (strcmp(day, "воскресенье") == 0) return 6;
    return -1; // Ошибка, если день недели не распознан
}

// Быстрая сортировка по дням недели
void quickSortByDay(Information arr[], int left, int right)
{
    if (left >= right) return;
    Information p = arr[left];
    int i = left, j = right;
    while (i <= j) {
        while (getDayIndex(arr[i].day) < getDayIndex(p.day))
            i++;
        while (getDayIndex(arr[j].day) > getDayIndex(p.day))
            j--;
        if (i <= j) {
            Information temp = arr[i];
            arr[i] = arr[j];
            arr[j] = temp;
            i++;
            j--;
        }
    }
    if (left < j) quickSortByDay(arr, left, j);
    if (i < right) quickSortByDay(arr, i, right);
}

void quickSortByPrepod(Information arr[], int left, int
right) {
    if (left >= right) return;
    Information pivot = arr[(left + right) / 2];
    int i = left, j = right;

    while (i <= j) {
        while (strcmp(arr[i].prepod, pivot.prepod) < 0) i++;
        while (strcmp(arr[j].prepod, pivot.prepod) > 0) j--;
        if (i <= j) {
            Information temp = arr[i];
            arr[i] = arr[j];
            arr[j] = temp;
            i++;
            j--;
        }
    }
    if (left < j) quickSortByPrepod(arr, left, j);
    if (i < right) quickSortByPrepod(arr, i, right);
}

void SortInformation(const char* filename, bool sortByDay) {
    ifstream file(filename, ios::binary);
    if (!file) {
        cout << "Ошибка при открытии файла.\n";
        return;
    }
}
```

Продолжение приложения А

```
}
Information records[max_records];
int count = 0;
while (file.read((char*)&records[count],
sizeof(Information))) {
    count++;
    if (count >= max_records) break;
}
file.close();
if (count == 0) {
    cout << "Файл пуст.\n";
    return;
}
ofstream report("report.txt", ios::app);
if (!report.is_open()) {
    cout << "Ошибка при открытии файла отчета.\n";
    return;
}
report << "\nЗаписи после сортировки: \n";
if (sortByDay) {
    quickSortByDay(records, 0, count - 1); // Сортировка
по дням недели
    cout << "Записи отсортированы по дням недели.\n";
}
else {
    quickSortByPrepod(records, 0, count - 1); //
Сортировка по фамилиям преподавателей
    cout << "Записи отсортированы по фамилиям
преподавателя.\n";
}
// Запись отсортированных данных как в файл, так и в
отчет
ofstream outfile(filename, ios::binary);
if (outfile.is_open()) {
    for (int i = 0; i < count; i++) {
        outfile.write((char*)&records[i],
sizeof(Information));
        viewOneInformationReport(records[i], report);
    }
    outfile.close();
}
else {
    cout << "Ошибка при записи в файл.\n";
}
report.close();
cout << "Данные перезаписаны в файл и отчет.\n";
}

void binarySearchByPrepod(const char* filename) {
    ifstream file(filename, ios::in | ios::binary);
    if (!file) {
        cout << "Ошибка при открытии файла.\n";
    }
}
```

Продолжение приложения А

```
return;
}
Information arr[max_records];
int count = 0;
while (file.read((char*)&arr[count],
sizeof(Information))) {
    count++;
    if (count >= max_records) break;
}
file.close();
if (count == 0) {
    cout << "Файл пуст.\n";
    return;
}
char searchPrepod[50];
cout << "Введите фамилию преподавателя для поиска: ";
cin >> searchPrepod;
int left = 0, right = count - 1;
int foundIndex = -1;
while (left <= right) {
    int mid = left + (right - left) / 2;
    int cmp = strcmp(arr[mid].prepod, searchPrepod);
    if (cmp == 0) {
        foundIndex = mid;
        break;
    }
    else if (cmp < 0) {
        left = mid + 1;
    }
    else {
        right = mid - 1;
    }
}
ofstream report("report.txt", ios::app);
if (!report.is_open()) {
    cout << "Ошибка при открытии файла отчета.\n";
    return;
}
report << "\nРезультаты поиска преподавателя: " <<
searchPrepod << "\n";
if (foundIndex == -1) {
    cout << "Преподаватель с такой фамилией не найден.\n";
    report << "Преподаватель с фамилией " << searchPrepod
<< " не найден.\n";
}
else {
    cout << "Преподаватель " << searchPrepod << "
найден.\n";
    report << "Преподаватель " << searchPrepod << "
найден:\n";
}
```

Продолжение приложения А

```
// Запись всех факультативов для найденного преподавателя
for (int i = 0; i < count; i++) {
    if (strcmp(arr[i].prepod, searchPrepod) == 0) {
        cout << "Преподаватель: " << arr[i].prepod <<
"\n";
        cout << "Предмет: " << arr[i].facultativ <<
"\n";
        cout << "День: " << arr[i].day << "\n";
        cout << "Количество студентов: " <<
arr[i].count_of_students << "\n\n";
        viewOneInformationReport(arr[i], report);
    }
}
report.close();
}

// Функция для сортировки вставками по факультативу
void insertionSortByFacultativ(const char* filename) {
    ifstream in;
    ofstream out, file_out;
    Information info, records[100];
    int counter = 0;
    in.open(filename, ios::binary | ios::in);
    if (!in.is_open()) {
        cout << "Ошибка при открытии файла для чтения.\n";
        return;
    }
    file_out.open("report.txt", ios::out | ios::app);
    if (!file_out.is_open()) {
        cout << "Ошибка при открытии файла отчета.\n";
        return;
    }
    file_out << "\n\nВы осуществили сортировку вставками по
факультативам:\n";
    while (in.read((char*)&info, sizeof(Information))) {
        records[counter++] = info;
    }
    in.close();
    for (int i = 1; i < counter; i++) {
        for (int j = i; j > 0 && strcmp(records[j -
1].facultativ, records[j].facultativ) > 0; j--) {
            Information temp = records[j];
            records[j] = records[j - 1];
            records[j - 1] = temp;
        }
    }
    out.open(filename, ios::binary | ios::trunc);
    if (!out.is_open()) {
        cout << "Ошибка при открытии файла для записи.\n";
        return;
    }
}
```


Продолжение приложения А

```
}
    for (int i = 0; i < counter; i++) {
        out.write((char*)&records[i], sizeof(Information));

        // Запись в отчет информации о факультативе
        file_out << left << setw(25) << records[i].facultativ
            << setw(25) << records[i].prepod
            << setw(25) << records[i].day
            << setw(45) << records[i].count_of_students <<
endl;
    }
    out.close();
    file_out.close();
    cout << "Сортировка завершена и данные перезаписаны в
файл.\n";
}

void searchByPrepodAndDay(const char* filename) {
    Information records[max_records];
    int count = 0;
    ifstream infile(filename, ios::binary);
    if (!infile) {
        cout << "Не удалось открыть файл.\n";
        return;
    }
    while (infile.read((char*)&records[count],
sizeof(Information))) {
        count++;
    }
    infile.close();
    char teacher[50];
    char day[20];
    cout << "Введите фамилию преподавателя: ";
    cin.ignore();
    cin.getline(teacher, 50);
    cout << "Введите день недели: ";
    cin.getline(day, 20);
    Information filteredRecords[max_records];
    int filteredCount = 0;
    for (int i = 0; i < count; i++) {
        if (strcmp(records[i].prepod, teacher) == 0 &&
strcmp(records[i].day, day) == 0) {
            filteredRecords[filteredCount] = records[i];
            filteredCount++;
        }
    }
    if (filteredCount == 0) {
        cout << "Факультативы не найдены.\n";
        return;
    }
    for (int i = 0; i < filteredCount - 1; i++) {
        for (int j = i + 1; j < filteredCount; j++) {
```

Продолжение приложения А

```
        if (filteredRecords[i].count_of_students <
filteredRecords[j].count_of_students) {
            Information temp = filteredRecords[i];
            filteredRecords[i] = filteredRecords[j];
            filteredRecords[j] = temp;
        }
    }
}
ofstream report("report.txt", ios::app);
if (!report) {
    cout << "Не удалось открыть файл отчета.\n";
    return;
}
report << "\nРезультаты поиска по преподавателю " <<
teacher << " в день " << day << ":\n";
report << "-----\n";
report << left << setw(25) << "Факультатив" << setw(25)
<< "Преподаватель" << setw(25)
<< "День недели" << setw(45)
<< "Кол-во записавшихся студентов" << endl;
report << "-----\n";

    for (int i = 0; i < filteredCount; i++) {
        writeToReport(report, filteredRecords[i].facultativ,
filteredRecords[i].prepod, filteredRecords[i].day,
filteredRecords[i].count_of_students);
    }

    report.close();
    cout << "Факультативы преподавателя " << teacher << " в
день " << day << ":\n";
    for (int i = 0; i < filteredCount; i++) {
        cout << "Факультатив: " <<
filteredRecords[i].facultativ << ", "
<< "количество студентов: " <<
filteredRecords[i].count_of_students << endl;
    }

    cout << "Результаты поиска записаны в отчет.\n";
}
void deleteFacultativ(const char* filename) {
    ifstream file(filename, ios::binary);
    ofstream tempfile("temp.bin", ios::binary);
    ofstream report("report.txt", ios::app);
    Information plan;
    char facultativ_delete[50];
    bool found = false;
```

Продолжение приложения А

```
cout << "Введите название факультатива для удаления: ";
cin >> facultativ_delete;

if (file.is_open() && tempfile.is_open() &&
report.is_open()) {
    while (file.read((char*)&plan, sizeof(plan))) {
        bool equal = true;
        // Сравниваем название факультатива
        for (int i = 0; i < 50; i++) {
            if (plan.facultativ[i] != facultativ_delete[i])
{
                if (plan.facultativ[i] == '\0' ||
facultativ_delete[i] == '\0') {
                    break;
                }
                equal = false;
                break;
            }
        }
        if (!equal) {
            tempfile.write((char*)&plan, sizeof(plan));
        }
        else {
            found = true;
        }
    }
    file.close();
    tempfile.close();
    if (found) {
        remove(filename);
        rename("temp.bin", filename);
        report << "\nФакультатив '" << facultativ_delete
<< "' был удален.\n";
        report << "-----\n";
        cout << "Факультатив удален.\n";
    }
    else {
        remove("temp.bin");
        cout << "Факультатив не найден.\n";
    }
    report.close();
}
else {
    cout << "Ошибка при чтении или записи в файлы.\n";
}
}

void edit(const char* filename) {
    ifstream file(filename, ios::binary);
    ofstream tempfile("temp.bin", ios::binary);
```

Продолжение приложения А

```
ofstream report("report.txt", ios::app);
Information plan;
char edit[50];
bool found = false;
cout << "Введите факультатив для редактирования: ";
cin >> edit;
if (file.is_open() && tempfile.is_open()) {
    while (file.read(reinterpret_cast<char*>(&plan),
sizeof(plan))) {
        if (strcmp(plan.facultativ, edit) == 0) {
            found = true;
            cout << "Редактирование факультатива: " <<
plan.facultativ << endl;
            cout << "Введите преподавателя: ";
            cin.ignore();
            cin.getline(plan.prepod, 50);
            cout << "Введите день недели: ";
            cin.getline(plan.day, 20);
            cout << "Введите кол-во записавшихся студентов:
";
            cin >> plan.count_of_students;
        }
        tempfile.write(reinterpret_cast<char*>(&plan),
sizeof(plan));
    }
    file.close();
    tempfile.close();
    if (found) {
        if (remove(filename) != 0) {
            cout << "Ошибка при удалении оригинального
файла.\n";
            return;
        }

        if (rename("temp.bin", filename) != 0) {
            cout << "Ошибка при переименовании временного
файла.\n";
            return;
        }

        cout << "Информация о факультативе обновлена.\n";
        if (report.is_open()) {
            report << "\nФакультатив \"" << edit << "\" был
отредактирован.\n";
        }
    }
    else {
        remove("temp.bin");
        cout << "Факультатив не найден.\n";
    }
}
```

Продолжение приложения А

```
else {
    cout << "Ошибка при открытии файла.\n";
}
if (report.is_open()) {
    report.close();
}
}
// Функция для сортировки факультативов преподавателя по
// количеству студентов (по убыванию)
void sortByStudents(Information arr[], int n) {
    for (int i = 0; i < n - 1; i++) {
        int max_index = i;
        for (int j = i + 1; j < n; j++) {
            if (arr[max_index].count_of_students <
arr[j].count_of_students) {
                max_index = j;
            }
        }
        if (max_index != i) {
            Information temp = arr[max_index];
            arr[max_index] = arr[i];
            arr[i] = temp;
        }
    }
}
// Функция для вывода статистики по преподавателям
void printStatistics(const char* filename) {
    ifstream file(filename, ios::binary);
    ofstream report("report.txt", ios::app);
    if (!file.is_open()) {
        cout << "Ошибка при открытии файла.";
        return;
    }
    Information records[max_records];
    int count = 0;
    while (file.read((char*)&records[count],
sizeof(Information))) {
        count++;
    }
    file.close();
    bool printedPrepod[max_records] = { false };
    if (report.is_open()) {
        report << "\nСтатистика по факультативам:\n";
        report << "-----\n";
        report << left << setw(25) << "Преподаватель" <<
setw(25)
        << "Факультатив" << setw(25)
        << "День недели" << setw(45)
        << "Кол-во записавшихся студентов" << endl;
```

Продолжение приложения А

```
report << "-----\n";
-----\n";
    }
    else {
        cout << "Ошибка при открытии отчета.\n";
        return;
    }

    for (int i = 0; i < count; i++) {
        bool isPrinted = false;
        for (int j = 0; j < i; j++) {
            if (strcmp(records[i].prepod, records[j].prepod)
== 0) {
                isPrinted = true;
                break;
            }
        }
        if (!isPrinted) {
            cout << "Преподаватель: " << records[i].prepod <<
endl;

            if (report.is_open()) {
                report << "\nПреподаватель: " <<
records[i].prepod << endl;
            }
            Information facultyList[max_records];
            int facultyCount = 0;
            for (int j = i; j < count; j++) {
                if (strcmp(records[j].prepod,
records[i].prepod) == 0) {
                    facultyList[facultyCount] = records[j];
                    facultyCount++;
                }
            }
            sortByStudents(facultyList, facultyCount);
            for (int j = 0; j < facultyCount; j++) {
                cout << " Факультатив: " <<
facultyList[j].facultativ
                << ", День: " << facultyList[j].day
                << ", Количество студентов: " <<
facultyList[j].count_of_students << endl;
                if (report.is_open()) {
                    writeToReport(report,
facultyList[j].facultativ, facultyList[j].prepod,
facultyList[j].day, facultyList[j].count_of_students);
                }
            }
        }
    }
    if (report.is_open()) {
        report.close();
    }
}
```

Продолжение приложения А

```
}
}
void popular_facultative(const char* filename) {
    ifstream file(filename, ios::in | ios::binary);
    ofstream report("report.txt", ios::app);
    if (!file) {
        cout << "Ошибка при открытии файла.\n";
        return;
    }
    Information arr[max_records];
    int count = 0;
    while (file.read((char*)&arr[count],
sizeof(Information))) {
        count++;
        if (count >= max_records) {
            cout << "Максимальное количество записей
достигнуто.\n";
            break;
        }
    }
    file.close();
    if (count == 0) {
        cout << "Файл пуст.\n";
        return;
    }
    char facultatives[max_records][50];
    int students[max_records] = { 0 };
    int uniqueCount = 0;
    for (int i = 0; i < count; i++) {
        bool found = false;
        for (int j = 0; j < uniqueCount; j++) {
            if (strcmp(arr[i].facultativ, facultatives[j]) ==
0) {
                students[j] += arr[i].count_of_students;
                found = true;
                break;
            }
        }
        if (!found) {
            strcpy_s(facultatives[uniqueCount],
arr[i].facultativ);
            students[uniqueCount] = arr[i].count_of_students;
            uniqueCount++;
        }
    }
    int maxStudents = 0;
    for (int i = 0; i < uniqueCount; i++) {
        if (students[i] > maxStudents) {
            maxStudents = students[i];
        }
    }
    cout << "Статистика по факультатавам:\n";
```

Продолжение приложения А

```
cout << "-----\n";
if (report.is_open()) {
    report << "\nСтатистика по факультатам:\n";
    report << "-----\n";
\n";
}

for (int i = 0; i < uniqueCount; i++) {
    cout << "Факультатив: " << facultatives[i] << ",
Количество студентов: " << students[i] << "\n";
    if (report.is_open()) {
        report << "Факультатив: " << facultatives[i] << ",
Количество студентов: " << students[i] << "\n";
    }
}

cout << "\nСамые популярные факультативы (с максимальным
количеством студентов: " << maxStudents << "):\n";
if (report.is_open()) {
    report << "\nСамые популярные факультативы (с
максимальным количеством студентов: " << maxStudents << "):\n";
}

for (int i = 0; i < uniqueCount; i++) {
    if (students[i] == maxStudents) {
        cout << " Факультатив: " << facultatives[i] << ",
Количество студентов: " << students[i] << "\n";
        if (report.is_open()) {
            report << " Факультатив: " << facultatives[i]
<< ", Количество студентов: " << students[i] << "\n";
        }
    }
}

if (report.is_open()) {
    report.close();
}

}

void menu()
{
    Information records[max_records];
    int count = 0;
    const char* filename = "facultative.txt";
    int choice;
    do {
        cout << "Меню:\n";
        cout << "1. Создание файла\n";
        cout << "2. Просмотр записей\n";
        cout << "3. Добавление записи\n";
        cout << "4. Линейный поиск по названию
факультатива\n";
        cout << "5. Поиск факультатива по преподавателю и дню
недели\n";\n";
    }
```


Продолжение приложения А

```
cout << "6. Бинарный поиск по преподавателю\n";

cout << "7. Быстрая сортировка по дню недели\n";
cout << "8. Сортировка выбором по количеству
студентов\n";
cout << "9. Сортировка вставками по названию
факультатива\n";
cout << "10. Статистика по преподавателям.Самый
популярный факультатив.\n";
cout << "11. Удаление записей из файла\n";
cout << "12.Редактирование записей в файле.\n";
cout << "13. Создание отчета в тестовом файле\n";
cout << "14. Выход\n";
cout << "Выберите пункт: ";
cin >> choice;
switch(choice){
case 1:
    createBinaryFile(filename);
    break;
case 2:
    viewInformation(filename);
    break;
case 3:
    addInformation(filename, records, count);
    break;
case 4:
    LineSearchFacultativ(filename);
    break;
case 5:
    searchByPrepodAndDay(filename);
    break;
case 6:
    SortInformation(filename, false);
    viewInformation(filename);
    binarySearchByPrepod(filename);
    break;
case 7:
    SortInformation(filename, true);
    break;
case 8:
    sortbychoice(filename);
    break;
case 9:
    insertionSortByFacultativ(filename);
    break;
case 10:
    printStatistics(filename);
    popular_facultative(filename);
    break;
case 11:
    deleteFacultativ(filename);
```

Продолжение приложения А

```
        break;
    case 12:
edit(filename);
        break;
    case 13:
        file_create();
        break;
    case 14:
        cout << "Выход из программы.\n";
        break;
    }
} while (choice !=14 );
}
int main() {
    SetConsoleCP(1251);
    SetConsoleOutputCP(1251);
    setlocale(LC_ALL, "ru");

    menu();
    return 0;
}
```

Приложение Б (ОБЯЗАТЕЛЬНОЕ) БЛОК-СХЕМА РАБОТЫ ПРОГРАММЫ

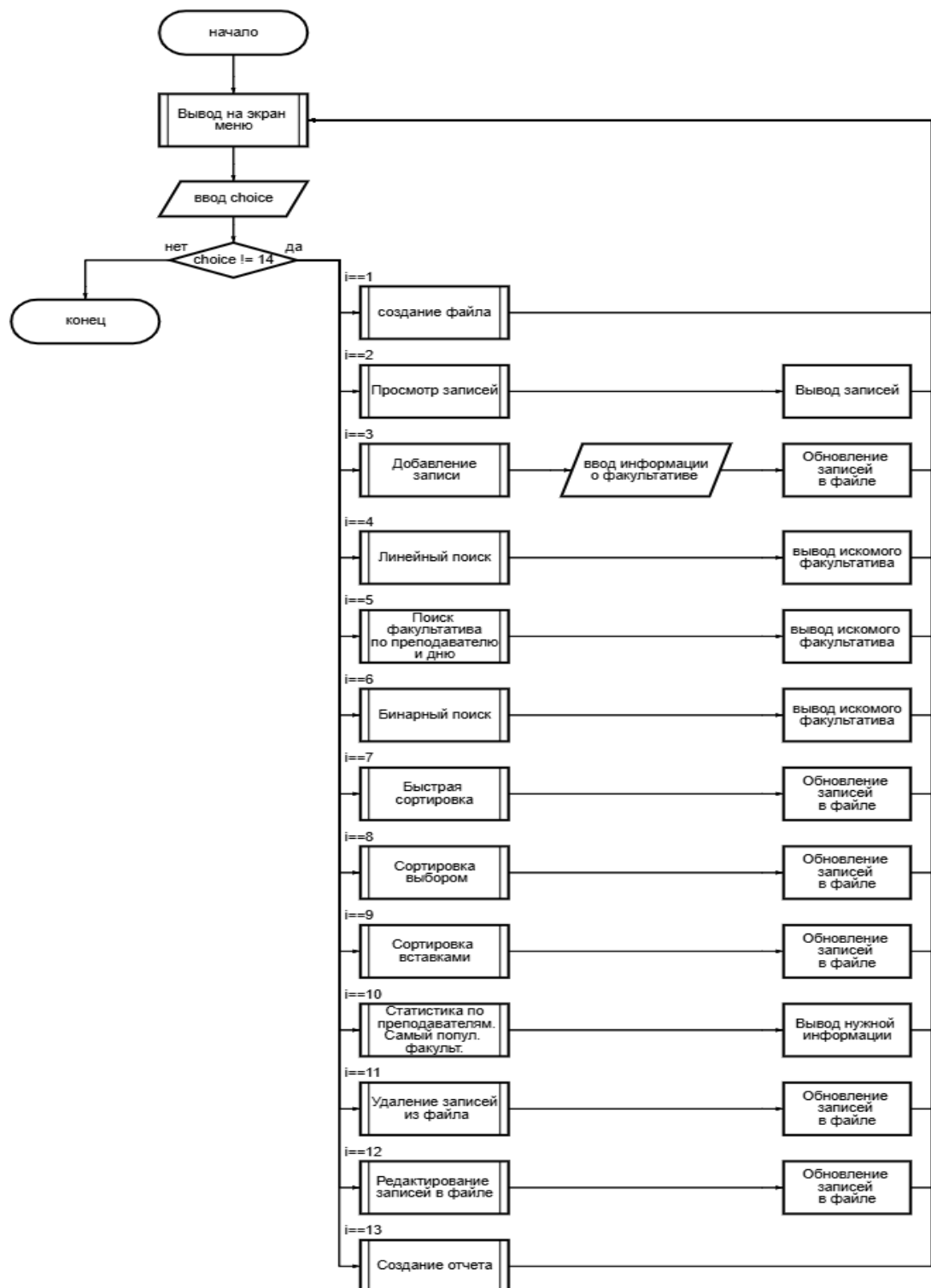


Рисунок Б.1 – Блок-схема работы программы

ВЕДОМОСТЬ ДОКУМЕНТОВ

Обозначение					Наименование					Дополнительные сведения		
					<u>Текстовые документы</u>							
БГУИР КП 6-05-0611-03 042 ПЗ					Пояснительная записка					54 с.		