1. Описание условия задачи.

Ввести репертуар театров, содержащий: название театра, название спектакля, диапазон цен на билеты, тип спектакля: 1. Пьеса 2. Драма 3. Комедия 4. Сказка a. Возраст: 3+, 10+, 16+ 5. Музыкальный: a. Композитор b. Страна c. Вид: балет, опера, мюзикл d. Возраст: 3+, 10+, 16+ e. Продолжительность Вывести список всех балетов для детей указанного возраста, продолжительностью меньше указанной.

2. Описание технического задания.

Создать таблицу, содержащую не менее 40 записей с вариантной частью. Произвести поиск информации по вариантному полю. Упорядочить таблицу, по возрастанию ключей (где ключ – любое невариантное поле по выбору программиста), используя: а) исходную таблицу; б) массив ключей, используя 2 разных алгоритма сортировки (простой, ускоренный). Оценить эффективность этих алгоритмов (по времени и по используемому объему памяти) при различной реализации программы, то есть, в случаях а) и б). Обосновать выбор алгоритмов сортировки. Оценка эффективности должна быть относительной (в %).

Входные данные.

Целое число: пункт меню(от 0 до 11)
Меню:

```
Программа для работы с репертуаром театров
Пункты меню:
 1) Загрузить таблицу из файла
 2) Показать несортированную таблицу
 3) Вывести неупорядоченную таблицу ключей
 4) Добавить запись в конец таблицы
 5) Удалить (первый попавшийся) театр по его названию
 6) Вывести упорядоченную таблицу ключей
 7) Вывести упорядоченную таблицу, к-рая упорядочена сама по
минимальной цене билета
 8) Вывести таблицу с помощью упорядоченных ключей
 9) Вывести все балеты указанного возраста, продолжительностью
меньше указанной
 10) Получить результат сравнения эффективности работы для текущей
таблицы и массива ключей
 11) Получить результат использования различных алгоритмов
сортировок
 12) Сохранить таблицу в файл.
 0) Завершить работу программы
```

Файл: путь к файлу, в котором содержатся данные.

Строка и числа: параметры зависящие от выбранного пункта.

Выходные данные.

Таблица, таблица ключей, успех или ошибка выполнения выбранного пункта, результат сравнения эффективности сортировок, результат сравнения эффективности работы программы при обработке данных в исходной таблице и в таблице ключей.

Способ обращения к программе.

Запускается через кнопку «*run*» в среде разработки «*Clion*»

Возможные аварийные ситуации и ошибки пользователя.

Некорректный ввод данных.

3. Описание внутренних структур данных.

Перечисление возрастных ограничений.

```
// Возрастные ограничения
typedef enum
{
    AGE_3 = 1,
    AGE_10,
    AGE_16
} age_limit_t;
```

Перечисление типа спектакля.

```
// Тип спекатакля
typedef enum
{
    PLAY = 1,
    DRAMA,
    COMEDY,
    FAIRY_TALE,
    MUSICAL
} performance_type_t;
```

PLAY — пьеса, *DRAMA* — драма, *COMEDY* — комедия, *FAIRY_TALE* — сказка, *MUSICAL* — Музыкальный.

Перечисление типа спектакля.

```
// Тип музыыкального спекатакля
typedef enum
{
```

```
    BALLET = 1,
    OPERA,
    MUSICAL_SHOW
} musical_type_t;
```

BALLET — баллет, OPERA — опера, MUSICAL_SHOW — мюзикл.

Структура для музыкального спектакля.

```
// Структура музыкального спектакля
typedef struct
{
    char composer[MAX_STR_LEN + 1];
    char country[MAX_STR_LEN + 1];
    musical_type_t type;
    age_limit_t age;
    int duration;
} musical_t;
```

MAX_STR_LEN = 30.
char composer[MAX_STR_LEN + 1] — имя композитора,
country[MAX_STR_LEN + 1] — название страны,
musical_type_t type — тип музыкального спектакля,
age_limit_t age — возрастное ограничения спекатакля,
int duration — продолжительность спектакля.

Объединение для спекаткля.

```
// Объединение для спектакля
typedef union
{
    age_limit_t fairy_tale_age_limit; // Для сказки
    musical_t musical; // Для музыкального
} performance_union;
```

age_limit_t fairy_tale_age_limit — Возрастное ограничение для сказки,
musical_t musical — данные для музыкального спектакля.

Структура для театра.

```
typedef struct
{
    char name[MAX_STR_LEN + 1];
    char performance_name[MAX_STR_LEN + 1];
    int price_high;
    int price_low;
    performance_union performance;
    performance_type_t performance_type;
} theatre_t;
```

char name[MAX_STR_LEN + 1] — Название театра,
char performance_name[MAX_STR_LEN + 1] — название спекаткля,
int price_high — максимальная цена билета,
int price_high — минимальная цена билета,
performance_union performance — вариантная часть записи(зависит от выбранного типа),
performance_type_t performance_type — тип спекаткля.

Структура для таблицы театров.

```
typedef struct
{
    theatre_t theatres[MAX_ROW_COUNT];
    size_t rows;
} table_theatre_t;
```

MAX_ROW_COUNT = 10000 — максимальный размер таблицы,
theatre_t theatres[MAX_ROW_COUNT] – массив театров,
size_t rows — количество строк в театре.

Структура для ключей.

```
typedef struct
{
    int price_low;
    size_t theatre_id;
} theatre_key_t;
```

int price_low — ключ, который связан с основной таблицей,
size_t theatre_id — индекс театра основной таблицы.

Интерфейс для работы со структурами.

Для работы с theatre_t

```
// Функция для чтения одного театра
int fread_theatre(theatre_t *theatre, FILE *file_input);

// Функция для вывода тетара
void fprint_theatre(theatre_t theatre, FILE *file_output);
```

Для работы с table_theatre_t и theatre_key_t

```
// Функция чтения табоицы из файла
int fread_table(table_theatre_t *table_theatre, theatre_key_t
table_key[], size_t *keys_size, FILE *file_input);

// Функция вывода таблицы
void fprint_table(table_theatre_t table_theatre, FILE
*file_output);
```

```
// Функция вывода таблицы ключей
void fprint_key_table(theatre_key_t table_key[], size_t keys_size,
FILE *file_output);

// Функция вывода таблицы с использование таблицы ключей
void fprint_table_by_keys(table_theatre_t table_theatre,
theatre_key_t table_key[], FILE *file_output);

// Обновление таблицы ключей с основной таблицей
void update_keys_with_table(table_theatre_t table_theatre,
theatre_key_t table_key[], size_t *keys_size);

// Функция удаления театра по его имени
int delete_theatre_by_name(table_theatre_t *table_theatre);

// Функция добавления театра
int add_theatre(table_theatre_t *table_theatre);

// Функция поиска баллетов по введенным данным
int find_ballets(table_theatre_t table_theatre, age_limit_t
age_limit, int duration);

// Соритровка вставками
void insertion_sort(void *base, size_t nmemb, size_t size,
compar_t cmp);

// Замер времени для таблицы
double time_sort_table(table_theatre_t table_theatre, size_t
count, sort_t sort);

// Замер времени для таблицы клбчей
double time_sort_keys(theatre_key_t table_keys[], size_t
table_size, size_t count, sort_t sort);

// Сохранение таблицы в файл
void save_file(table_theatre_t table_theatre, FILE *file_output);
```

4. Описание алгоритма.

Программа работает следующим образом:

1. Пользователь видит меню при запуске программы.
2. Пользователь выбирает пункт меню.
3. Если выбранный пункт требует ввода данных, программа проверяет их и выполняет соответствующий запрос.
4. Если выбранный пункт требует вывода таблицы, программа сортирует данные методом быстрой сортировки и выводит их на экран.

5. Если возникает ошибка, программа информирует пользователя и продолжает работать.

5. Набор тестов.

Название теста	**Ввод**	**Результат**
Негативные тесты		
Некорректный ввод пункта меню	q	Ошибка: Пожалуйста, введите число.
Неверный пункт меню	12	Выберите 1 из пунктов меню.
Несуществующий файл	non_exist.txt	Ошибка открытия файла или его не существует.
Некорректный ввод тетра	toooooo_long_long_name_of_theatre	Ошибка ввода названия театра. Ошибка ввода театра
Удаление несуществующего театра	non_exist_theatre	Ничего не удалилось
Некорректный ввод для поиска баллета	2 10	Балеты не найдены.
Позитивные тесты.		
Существующий файл	../data/test_40.txt	Таблица успешна загружена
Корректное добавление театра	theatre name spectacle name 200 1200 1	Театр был успешно добавлен.
Корректное удаление театра	theatre name	Театр успешно удален
Корректный ввод данных баллета	2 90	Таблица с найденными баллетами

6. Оценка эффективности.

Временные замеры.

Количество итераций = 1000

Количество записей	**Алгоритм сортировки**	**Структура данных**	**Среднее время(мс)**
40	Быстрая сортировка	Таблица	2.822
		Ключи	1.313
	Сортировка выбором	Таблица	20.583
		Ключи	0.679
100	Быстрая сортировка	Таблица	8.386
		Ключи	5.89
	Сортировка выбором	Таблица	87.194
		Ключи	1.086
500	Быстрая сортировка	Таблица	64.024
		Ключи	33.319
	Сортировка выбором	Таблица	2183.254
		Ключи	4.772
1000	Быстрая сортировка	Таблица	162.934
		Ключи	55.438

	Сортировка выбором	Таблица	9249.56
		Ключи	7.004

Оценка эффективности.

Количество записей	**Отношение времени *быстрой сортировки* таблицы к таблице ключей**	**Отношение времени *сортировки выбором* таблицы к таблице ключей**
40	2.15	30.31
100	1.42	80.29
500	1.92	457.51
1000	2.94	1320.61

Объем занимаемой памяти (байт)

Количество записей	**Таблица**	**Ключи**
40	6080	640
100	15200	1600
500	76000	8000
1000	152000	16000

7. Выводы по проделанной работе.

Объединения позволяют хранить разные типы данных в одной области памяти, но только одно значение может быть актуальным в любой момент времени. Это может быть полезно для экономии памяти, но требует осторожного использования, чтобы избежать ошибок. По проделанной работе можно сделать вывод, что сортировка таблицы ключей выполняется за меньшее количество времени, на малых количествах записей это может быть не так заметно, но с добавлением данных прирост времени сортировки сильно увеличивается. Также можно заметить, что быстрая сортировка работает быстрее сортировки выбором, особенно на больших объемах данных.

Контрольные вопросы.

1.Как выделяется память под вариантную часть записи?

Память под вариантную часть записи в данном случае выделяется в объединении, где все варианты записи имеют одинаковую область памяти, а размер области памяти определяется размером наибольшего варианта (в моем случае размеру музыкального спекаткля).

2.Что будет, если в вариантную часть ввести данные, несоответствующие описанным?

Это приведет к неопределенному поведению программы, поскольку компилятор не проверяет типы данных в объединениях.

3.Кто должен следить за правильностью выполнения операций с вариантной частью записи?

Сам программист должен следить за правильностью выполнения операций с вариантной частью записи в Си, поскольку компилятор не обеспечивает автоматической проверки типов данных в объединениях.

4.Что представляет собой таблица ключей, зачем она нужна?

Таблица ключей — это таблица, содержащая в себе некоторое ключевое поле исходной таблицы, а также индекс исходной этого поля в исходной таблице. Нужна она для того, чтобы не работать со всей таблице во время выполнения разных операций с таблицей, а лишь с ключевым полем таблицы.

5.В каких случаях эффективнее обрабатывать данные в самой таблице, а когда – использовать таблицу ключей?

Обрабатывать данные в самой таблице эффективнее, когда время обработки не так важно, как занимаемая память, а таблицу ключей, когда время обработки в приоритете по сравнению с занимаемой дополнительной памятью.

6. Какие способы сортировки предпочтительнее для обработки таблиц и почему?

Те сортировки, чья сложность выполнения измеряется логарифмически или линейна O(n*log(n) или O(log(n)) или O(n). Обычно это quick sort или merge sort. Они предпочтительны, так как самая затратная по времени операция в сортировке — обмен элементов, то использование сортировко с меньшим кол-вом обменов наиболее предпочтительно.

Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «ИНФОРМАТИКА И СИСТЕМЫ УПРАВЛЕНИЯ»

КАФЕДРА «ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ ЭВМ И ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ»

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №2 ПО ДИСЦИПЛИНЕ:
ТИПЫ И СТРУКТУРЫ ДАННЫХ

Записи с вариантами, обработка таблиц

Студент **Павлов ДВ**

Группа **ИУ7-33Б**

Вариант 5

Название предприятия **НУК ИУ МГТУ им. Н. Э. Баумана**

Студент _____ **Павлов Д. В.**

Преподаватель _____**Никульшина Т. А.**

2024