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

Вариант 7

Студент **Ильченко Е. А.**

Группа **ИУ7-34Б**

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

Студент _____ **Ильченко Е. А.**

Преподаватель _____ **Силантьева А. В.**

2024 г.

Описание условия задачи

Создать таблицу, содержащую не менее 40-ка записей. Упорядочить данные в ней по возрастанию ключей, двумя алгоритмами сортировки, где ключ – любое невариантное поле (по выбору программиста), используя: а) саму таблицу, б) массив ключей. Возможность добавления и удаления записей в ручном режиме, просмотр таблицы, просмотр таблицы в порядке расположения таблицы ключей обязательна. Осуществить поиск информации по варианту.
Ввести список абонентов, содержащий фамилию, имя, номер телефона, адрес (улица, дом), статус абонента:

1. Друзья:
    a. Дата рождения: день, месяц, год
2. Коллеги:
    a. должность
    b. организация

Вывести список всех друзей, которых необходимо поздравить с днем рождения в ближайшую неделю. (текущая дата вводится пользователем)

Описание ТЗ

1. Описание исходных данных и результатов работы программы

Входные данные:

Пользовательская команда из доступных, содержимое файла и необходимые аргументы определенного сценария

`0` - Выход
`1` - Чтение таблицы из файла
`2` - Ввод таблицы абонентов
`3` - Вывод списка друзей, которых необходимо поздравить
`4` - Добавить запись в конец таблицы
`5` - Удалить запись по значению указанного поля
`6` - Отсортировать таблицу ключей
`7` - Отсортировать таблицу абонентов
`8` - Вывод таблицы ключей
`9` - Вывод таблицы абонентов
`10` - Вывод таблицы абонентов, по таблице ключей

11 - Замерить эффективность сортировок

2. Описание задачи, реализуемой в программе

Задача, реализуемая в программе, заключается в реализации инструмента для работы со списком абонентов. Это включает в себя: ввод, хранение, обработку и вывод данных, связанных с абонентами

4. Способ обращения к программе

Запуск исполняемого файла

```
./app.exe
```

Далее выбирается, какой пункт меню выполнить

5. Описание возможных аварийных ситуаций и ошибок пользователя

Список аварийных ситуаций:

- Ошибка ввода `IO_ERROR`
- Переполнение буфера `BUFFER_OVERFLOW`
- Переполнение массива абонентов `ERROR_TOO_MANY_ENTRIES`
- Ошибка открытия файла `ERROR_FILE_OPEN`
- Ошибка ввода комманды `COMMAND_ERROR`
- Ошибка ввода метода сортировки `WRONG_SORT_METHOD`

6. Описание внутренних структур данных

Структура данных абонента:

```
#define MAX_NAME_LEN 50
#define MAX_PHONE_LEN 20
#define MAX_STREET_LEN 50
#define MAX_POSITION_LEN 50
#define MAX_ORG_LEN 50

typedef struct
{
   int day;
   int month;
   int year;
} birth_date_t;
```

```
typedef struct
{
   char position[MAX_POSITION_LEN];
   char organization[MAX_ORG_LEN];
} colleague_info_t;

typedef struct
{
   birth_date_t birth_date;
} friend_info_t;

typedef union
{
   friend_info_t friend_info;
   colleague_info_t colleague_info;
} subscriber_details_t;

typedef enum
{
   FRIEND,
   COLLEAGUE
} subscriber_status_t;

typedef struct
{
   char surname[MAX_NAME_LEN];
   char name[MAX_NAME_LEN];
   char phone[MAX_PHONE_LEN];
   char street[MAX_STREET_LEN];
   int house_number;
   subscriber_status_t status;
   subscriber_details_t details;
} subscriber_t;
```

Структура данных ключей:

```
typedef struct
{
   char surname[MAX_NAME_LEN];
   int index;
} key_t;
```

7. Описание функций

```
int load_subscribers_from_file(const char *filename, subscriber_t
subscribers[], int *num_subscribers)
```

Функция для чтения файла, содержащего данные об абонентах

```
int save_to_csv(subscriber_t *subscribers, int num_entries, const char
*filename)
```

Функция для сохранения массива абонентов в csv-файл

```
int enter_subscribers_array(subscriber_t subscribers[], int num_subscribers)
```

Функция для ввода массива абонентов

```
int input_subscriber(subscriber_t *subscriber)
```

Функция для ввода одного абонента

```
void create_key_array(subscriber_t subscribers[], key_t keys[], int
num_subscribers)
```

Функция для создания массива ключей на основе массива абонентов

```
int save_keys_to_csv(const char *filename, key_t keys[], int num_keys)
```

Функция для сохранения массива ключей в csv-файл

```
void print_keys(key_t keys[], int num_keys)
```

Функция для вывода массива ключей

```
void bubble_sort_keys(key_t keys[], int num_subscribers)
```

Функция для сортировки пузырьком массива ключей

```
void quick_sort_keys(key_t keys[], int low, int high)
```

Функция для быстрой сортировки массива ключей

```
void print_upcoming_birthdays(subscriber_t *subscribers, int num_subscribers,
int current_day, int current_month, int current_year)
```

Функция для вывода предстоящих дней рождений

```
int add_new_subscriber(subscriber_t subscribers[], int *num_subscribers,
key_t keys[], const char *file, const char *key_file)
```

Функция для добавления нового абонента в конец массива

```
int delete_subscriber(subscriber_t subscribers[], key_t keys[], int
*num_subscribers, const char *surname, const char *file, const char
*key_file)
```

Функция для удаления абонента по фамилии

```
void print_subscribers(subscriber_t subscribers[], int num_subscribers)
```

Функция для вывода массива абонентов

```
void bubble_sort_subscribers(subscriber_t subscribers[], int num_subscribers)
```

Функция для сортировки пузырьком абонентов

```
void quick_sort_subscribers(subscriber_t subscribers[], int low, int high)
```

Функция для быстрой сортировки абонентов

```
void print_subscribers_by_keys(subscriber_t subscribers[], key_t keys[], int
num_keys)
```

Функция для вывода массива абонентов по ключам

```
void generate_random_subscriber(subscriber_t *subscriber)
```

Функция для генерации случайного абонента

```
void generate_dataset(subscriber_t subscribers[], int dataset_size)
```

Функция для генерации массива случайных абонентов

```
void measure_sorting_times_bubble(subscriber_t subscribers[], key_t keys[],
int num_subscribers)
```

Функция для замера времени сортировки пузырьком

```
void measure_sorting_times_quick(subscriber_t subscribers[], key_t keys[],
int num_subscribers)
```

Функция для замера времени быстрой сортировки

8. Описание алгоритма

Алгоритм создания таблицы абонентов

1. Инициализация:

Пусть S — основная таблица абонентов, содержащая n записей. Каждая запись включает в себя поля, такие как фамилия, имя, телефон, адрес, статус и дополнительная информация

2. Шаги создания таблицы абонентов:

Запросите у пользователя количество абонентов n.

Для каждой записи i от 0 до n-1:

1. Вводите данные абонента, включая фамилию, имя, телефон, адрес, статус и, если это необходимо, дополнительные детали (например, дату рождения или информацию о коллеге).
2. Сохраните введенные данные в S[i].

3. Итог:

После завершения ввода у вас будет заполненная таблица абонентов S, готовая к использованию.

Алгоритм создания массива ключей

1. Инициализация:

Пусть T — основная таблица абонентов, содержащая n записей.

Пусть K — массив ключей, состоящий из структур, содержащих ключ (например, фамилию) и индекс записи в основной таблице T.

2. Шаги создания массива ключей:

Для каждой записи T[i] в основной таблице, где i = 0, 1, ..., n-1:

1. Извлеките значение ключевого поля T[i].surname.
2. Создайте элемент массива ключей K[i], который содержит:
    a. Значение ключа: K[i].key = T[i].surname.
    b. Индекс записи: K[i].index = i.

3. Итог:

Массив K будет содержать n элементов, каждый из которых соответствует одной записи в основной таблице T, но включает только значение ключа и индекс.

Алгоритм быстрой сортировки массива ключей

1. Инициализация:

Пусть K — массив ключей длины n

Алгоритм QuickSort вызывается для сортировки массива ключей по возрастанию.

2. Шаги сортировки:

1. Разбиение: Выберите опорный элемент (например, последний элемент массива K).
2. Сравнение и перестановка:
   a. Разделите массив на две части: одна часть будет содержать элементы, меньшие опорного, другая — большие.
   b. Обменяйте элементы так, чтобы все элементы, меньшие опорного, оказались слева, а большие — справа.
3. Рекурсия: Рекурсивно применяйте сортировку для каждой из получившихся частей массива.
4. Процесс продолжается, пока каждая часть массива не будет отсортирована.

3. Итог:

После завершения работы алгоритма QuickSort массив K будет отсортирован по ключам. Индексы K[i].index можно использовать для доступа к соответствующим записям в основной таблице T.

Алгоритм сортировки пузырьком

1. Инициализация:

Пусть T — основная таблица абонентов, содержащая n записей.

2. Шаги сортировки:

Для i от 0 до n-1:

Для j от 0 до n-i-2:

Если T[j].surname > T[j+1].surname, обменяйте T[j] и T[j+1].

3. Итог:

После завершения работы алгоритма сортировки пузырьком таблица T будет отсортирована по фамилиям абонентов.

Алгоритм замера времени работы сортировки

1. Инициализация:

Сгенерируйте массив абонентов заданной длины.

Сохраните текущее время до начала сортировки.

2. Запуск сортировки:

Выполните алгоритм сортировки (например, быструю сортировку или сортировку пузырьком) для таблицы абонентов или массива ключей.

3. Завершение и замер:

Сохраните текущее время после завершения сортировки.

Вычислите разницу между начальным и конечным временем, чтобы получить время выполнения сортировки.

4. Итог:

Выведите время работы сортировки на экран для анализа производительности.

Тесты

Тест	**Описание теста**	**Шаги теста**	**Ожидаемый результат**
1	Проверка правильности ввода и вывода данных	1. Ввести корректные данные для абонента (фамилия, имя, телефон, статус). 2. Выполнить вывод данных на экран.	Введенные данные корректно отображаются на экране.
2	Ввод неверных данных в вариантную часть записи	1. Ввести абонента со статусом "Friend". 2. В поля даты рождения ввести строки вместо чисел (например, "abc" вместо "01-01-2000").	Программа выводит сообщение об ошибке типа данных и запрашивает повторный ввод.

3	Пустой ввод	1. Попробовать создать абонента без ввода данных (нажимая Enter без ввода).	Программа выводит сообщение, что ввод данных пуст, и запрашивает корректные данные.
4	Проверка операций добавления и удаления абонентов	1. Добавить несколько абонентов. 2. Удалить одного абонента по фамилии. 3. Проверить, что оставшиеся абоненты корректно отображаются.	Добавленные абоненты корректно отображаются, удаленный абонент исключен из списка.
5	Отслеживание переполнения таблицы	1. Заполнить таблицу до максимальной емкости (100 записей). 2. Попытаться добавить еще одного абонента.	Программа выводит сообщение о переполнении таблицы и предотвращает добавление нового абонента.
6	Проверка правильности сортировки таблицы абонентов	1. Ввести несколько абонентов с разными фамилиями. 2. Выполнить сортировку абонентов по фамилиям. 3. Вывести отсортированную таблицу.	Абоненты отображаются в правильном алфавитном порядке по фамилии.
7	Проверка	1. Создать таблицу	Таблица ключей

	правильности сортировки таблицы ключей	абонентов. 2. Сгенерировать таблицу ключей (по фамилии). 3. Выполнить сортировку таблицы ключей и вывести её.	отображается в правильном алфавитном порядке по фамилии.
8	Проверка вывода таблицы абонентов по отсортированной таблице ключей	1. Ввести несколько абонентов. 2. Создать таблицу ключей. 3. Выполнить вывод абонентов в порядке, соответствующем отсортированной таблице ключей.	Абоненты выводятся в том же порядке, что и в отсортированной таблице ключей.
9	Проверка ввода некорректного номера телефона	1. Ввести абонента с некорректным номером телефона (например, меньше 11 цифр или с буквами).	Программа выводит сообщение об ошибке и запрашивает повторный ввод номера телефона.
10	Проверка вывода друзей с предстоящим днем рождения	1. Ввести несколько абонентов со статусом "Friend". 2. Ввести текущую дату и проверить, что программа корректно выводит список друзей, у которых день рождения в ближайшую неделю.	Программа корректно выводит друзей с предстоящим днем рождения.

Оценка эффективности

Эффективность по времени/памяти считается, как среднее арифметическое разных от эфективности одного метода сортировки с ключами и без ключей. Эффективность по времени/памяти считалась, как

$$\frac{SortTable - SortKeys}{SortKeys} \cdot 100$$

Измерения проводились на MacBook Pro 13 2019

Количество	**Таблица абонентов**			**Таблица ключей**			**Выгода по времени, %**		**Выгода по памяти, %**
	Пузырек, сек$^{-6}$	QSort, сек$^{-6}$	Память, байт	Пузырек, сек$^{-6}$	QSort, сек$^{-6}$	Память, байт	Пузырек	QSort	Память
100	167	79	27800	149	55	33200	12,1	43,6	-16,3
1000	12970	3808	278000	7612	1626	332000	70,4	134,2	-16,3
10000	1134252	290773	2780000	618520	108981	3320000	83,4	166,8	-16,3

Вывод

В данной лабораторной работе можно выделить несколько ключевых моментов:

1. Использование типа данных `union` позволяет более эффективно расходовать память программы, так как оно обеспечивает совместное хранение различных типов данных. Однако это требует от разработчика более внимательного подхода.
2. Применение индексов в ситуациях, когда количество столбцов таблицы значительно превышает количество столбцов, по которым производится сортировка, значительно увеличивает эффективность сортировки. Это связано с тем, что такой подход снижает объем копируемой памяти во время выполнения сортировки. Однако это также приводит к увеличению использования дополнительной памяти примерно на 11% в рамках базы данных данной лабораторной работы.

Ответы на контрольные вопросы

1. **Как выделяется память под вариантную часть записи?**
   Без использования типа данных union необходимо выделять память для каждой из возможных вариантных структур, что увеличивает общий объем используемой памяти. Применение union позволяет более эффективно распределять память, так как для вариантной части записи выделяется пространство, достаточное для хранения любого из возможных вариантов. Поскольку в каждый момент времени используется только один вариант, память выделяется исходя из размера самого большого варианта. Это значительно оптимизирует использование памяти.

2. **Что будет, если в вариантную часть ввести данные, несоответствующие описанным?**
   Это может привести к неопределенному поведению программы, так как в языке Си тип данных union не проверяется во время компиляции. Из-за этого программист должен контролировать ввод данных

3. **Кто должен следить за правильностью выполнения операций с вариантной частью записи?**
   За правильностью выполнения операций с вариантной частью записи должен следить программист, так как в языке Си нет функционала для проверки корректности ввода данных в вариантную запись

4. **Что представляет собой таблица ключей, зачем она нужна?**
   Таблица ключей – это вспомогательная структура данных, содержащая ключевые поля записей и индексы этих записей в основной таблице. Она используется для ускорения поиска и сортировки, позволяя избежать перемещения самих записей. Вместо этого сортировке и обработке подвергаются ключи, а доступ к записям осуществляется через их индексы.

5. **В каких случаях эффективнее обрабатывать данные в самой таблице, а когда – использовать таблицу ключей?**
   Обработка данных в таблице достигает максимальной эффективности, когда она компактна или требуется выполнить лишь несколько простых операций, таких как поиск или обновление. В ситуациях, когда таблица имеет значительные размеры и требуется многократное выполнение сортировок или поисковых операций, использование таблицы ключей становится более

оправданным. Это также позволяет сократить количество перемещений записей, что помогает экономить время и ресурсы, особенно если записи содержат множество полей.

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

   Выбор методов сортировки зависит от размера таблицы. Для небольших таблиц могут оказаться эффективными простые алгоритмы, такие как сортировка вставками или пузырьковая сортировка, так как они проще в реализации для разработчика. В то же время для больших таблиц лучше использовать более сложные и эффективные алгоритмы, такие как быстрая сортировка или сортировка слиянием, поскольку они обладают лучшей асимптотической сложностью, равной O(n log n). Эти методы обеспечивают более быстрое выполнение при сортировке больших объемов данных и таблиц с множеством записей.