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 ЛАБОРОТОРНОЙ РАБОТЕ № 1 «ОБРАБОТКА ТАБЛИЦ»

Студент Байрамгалин Ярослав Ринатович
Группа ИУ7-33Б

Москва 2021
МГТУ им. Н.Э. Баумана

1. Описание условия задачи

Создать таблицу, содержащую не меньше сорока записей (тип – запись с вариантами). Упорядочить данные в ней по возрастанию ключей, двумя алгоритмами сортировки, где ключ – любое невариантное поле (по выбору программиста), используя:

- саму таблицу;
- массив ключей.

Возможность удаления и добавления записей обязательна.

Вывести список абонентов, содержащий фамилию, имя, телефон, адрес, статус (личный – дата рождения: день, месяц, год; служебный – должность, организация). Найти всех друзей, которых необходимо поздравить с днем рождения в ближайшую неделю.

2. Описание технического задания

Входные данные:

Любые входные данные могут содержать только символы латинского алфавита, цифры и знаки пунктуации, если не указано иное. При вводе кириллических символов поведение программы не определено.

Имя файла: строка, содержащая не более 29 символов. При существовании файла с указанным именем он будет открыт. При отсутствии будет создан новый файл. При открытии файла неверной структуру (созданного не этой программой) поведение программы не определено.

Пункт меню: строка, не превышающая 29 символов. Для взаимодействия с программой необходимо указать число соответствующее пункту меню:

- 1 – для добавления записи в таблицу;
- 2 – для удаления записи из таблицы;
- 3 – для вывода всех абонентов из таблицы, не сортируя;
- 4 – для вывода всех абонентов, отсортированных по фамилии;
- 5 – для вывода всех ключей при сортировке абонентов по имени;
- 6 – для вывода абонентов, имеющих день рождения в течение 7 дней;
- 0 – для завершения работы программы.

При выборе иного пункта меню будет выведено сообщение об ошибке.

Добавление записи в таблицу:

- ввод имени (строка не более 29 символов)
- ввод фамилии (строка не более 29 символов)
- ввод номера телефона (строка не более 29 символов)
- ввод города (строка не более 29 символов)
- ввод улицы (строка не более 29 символов)
- ввод дома (строка не более 29 символов)
- ввод квартиры (строка не более 29 символов)
- ввод статуса (строка не более 29 символов) – может принимать значение *personal* или *company*, при вводе иного будет выведено сообщение об ошибке
- если при вводе статуса указано personal:
  - ввод дня даты рождения (целое число от 1 до 31)
  - ввод месяца даты рождения (целое число от 1 до 12)
  - ввод года даты рождения (целое число от 1900 до 2021)
- если при вводе статуса указано company:
  - ввод названия компании (строка не более 29 символов)
  - ввод должности (строка не более 29 символов)

При успешном добавлении записи в таблицы будет выведено соответствующее сообщение.

Удаление записи из таблицы: необходимо указать индекс соответствующей записи (можно посмотреть при выводе без сортировки).

При успешном удалении записи из таблицы будет выведено соответствующее сообщение.

Просмотр записи по ключам: для просмотра отсортированных по ключам записей после выбора пункта 6 меню будет предложено показать отсортированные по ключам позиции. Для показа необходимо ввести «Д» или «д».

Выходные данные:

При открытии файла: сообщение о успешно выполненном открытии или сообщение об ошибке.

При удалении: сообщение о успешно выполненном удалении или сообщение об ошибке.

При добавлении записи: сообщение о успешно добавлении записи или сообщение об ошибке.

При сортировке по фамилии: список абонентов и всех данных о них, отсортированный по фамилии.

При сортировке по имени: значения массива ключей, при сортировке исходного списка абонентов по имени. Далее будет выведено предложение вывести абонентов в порядке, соответствующем ключам. При положительном ответе будет выведен список абонентов и всех данных о них, отсортированный по имени.

При поиске ближайших дней рождения: будет выведен список абонентов, у которых день рождения выпадает на ближайшие 7 дней.

Действие программы:

Организация работы с таблицей абонентов с полями, указанными выше.

Обращение к программе:

Запускается командой $./app.exe$ через терминал, находясь в директории, содержащей программу.

Аварийные ситуации

1. Введенная строка превышает 29 символов.
2. При указании даты рождения введены числа, выходящие за границы значений, указанных в спецификации.
3. Превышена допустимая длина строки.
4. Пустой ввод.
5. Файл не может быть открыт.
6. Ввод пустой.

3. Описание структуры данных

Для хранения данных об абонентах была разработана структура данных `subscriber_t`.

Примечание: `MAX_NAME_LNG = 30, MAX_PHONE_LNG = 15.`

Ниже приведено описание типа `subscriber_t`.

```
typedef struct {
    char first_name[MAX_NAME_LNG];
    char last_name[MAX_NAME_LNG];
    char phone[MAX_PHONE_LNG];
    address_t address;
    status_t status;
    subscriber_info_t subscriber_info;
} subscriber_t;
```

Поля структуры:

- `first_name` – имя;
- `last_name`– фамилия;
- `phone` – телефон;
- `address` – адрес;
- `status` – статус;
- `subscriber_info` – информация об абоненте;
- `key` – индекс.

Для хранения адреса была разработана структура `address_t`.

Примечание: `MAX_ADDRESS_FIELD_LNG = 30.`

Ниже приведено описание типа `address_t`.

```
typedef struct {
    char city[MAX_ADDRESS_FIELD_LNG];
    char street[MAX_ADDRESS_FIELD_LNG];
    char house[MAX_ADDRESS_FIELD_LNG];
    char apartment[MAX_ADDRESS_FIELD_LNG];
} address_t;
```

Поля структуры:

- `city` – город;
- `street` – улица;
- `house` –дом;
- `apartment` – квартира.

Для хранения данных об статусе абонента была разработана структура данных `status_t`.

Ниже приведено описание типа `subscriber_t`.

```
typedef enum {
    personal,
    company
} status_t;
```

Поля структуры:

- `personal` – статус - личный;
- `company` – статус - служебный.

Для хранения вариативных данных информации о абонементе было реализовано объединение `subscriber_info_t`.

Ниже приведено описание типа `subscriber_info_t`.

```
typedef union {
    personal_info_t person_info;
    company_info_t company_info;
} subscriber_info_t;
```

Поля объединения:

- `person_info` – информация об абоненте, если его статус - личный;
- `company_info` – информация об абоненте, если его статус - служебный.

Для хранения данных о компании была реализована структура `company_info_t`.

Ниже приведено описание типа `company_info_t`.

```
typedef struct {
    char name[MAX_COMPANY_INFO_LNG];
    char position[MAX_COMPANY_INFO_LNG];
} company_info_t;
```

Поля структуры:

- `name` – название компании;
- `position` – должность.

Для хранения личных данных абонента была разработана структура данных `personal_info_t`.

Ниже приведено описание типа `subscriber_t`.

```
typedef struct {
    date_t birth_date;
} personal_info_t;
```

Поля структуры:

- `date_t` – дата рождения.

Для хранения данных о дате рождения была реализована структура `date_t`.

Ниже приведено описание типа `date_t`.

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

Поля структуры:

- `day`– день;
- `month`–месяц;
- `year` –год.

4. Описание алгоритма

1. Пользователь вводит пункт меню.
2. Пользователь выбирает соответствующий его желанию пункт меню.
3. При вводе «0» программа завершается.

5. Набор тестов

	Описание теста	**Ввод**	**Ожидаемый вывод**
1	Выбран некорректный пункт меню	uuuu	Неверный ввод
2	Превышено наибольшее возможное число записей в файле	../too_many_elems	Слишком много элементов
3	Превышена допустимая длина строки (в любом месте)	Aaaaa37189371293712937193719237912379123791237921731982371293172837129321877 9	Превышена допустимая длина строки
4	Недопустимое значение дня рождения	35	Ошибка ввода
5	Недопустимое значение месяца рождения	14	Ошибка ввода
6	Недопустимое значение года рождения	1000	Ошибка ввода

6. Оценка эффективности

При сортировке массива с массивом ключей требуется дополнительная память в размере sizeof(int) * sz, где sz – размер массива. Однако выигрыш по времени является существенным и увеличивается с увеличением размера сортируемой структуры. Таким образом сортировка по ключам может помочь оптимизировать работу с памятью.

COUNT	QSORT, мс	BUBBLE, мс	QSORT KEY, мс	BUBBLE KEY, мс
45	16.8	31	3.3	37
100	69.6	157	6.3	150
500	1850	4547	55	1978
1000	7826	18379	145	7689

7. Выводы

Использование массива ключей для сортировки требует некоторое количество дополнительной памяти, однако при большом размере сортируемой структуры размер дополнительной памяти sizeof(int) / sizeof(struct my_struct)

8. Контрольные вопросы

a. Как выделяется память под вариантную часть записи?

Память под вариативную часть в Си выделяется так, чтобы помещалось поле, занимающее наибольшее место. Наиболее удобный способ реализации – используя объединения (union).

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

Поведение программы не будет определенным. Именно поэтому необходимо контролировать любые данные, поступающее от пользователя.

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

Ответственность за правильность проведения операций возлагается на программиста.

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

Таблица ключей представляет собой массив, содержащий индексы в исходной таблице. Для получения выигрыша по времени (оптимизации объема переставляемой местами информации) при необходимости сортировки таблица ключей представляет более оптимальное решение.

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

   В самой таблице эффективнее обрабатывать данные, если необходимо минимизировать затраты по памяти. Использование таблицы ключей же дает выигрыш во времени, но задействует O(n) дополнительной памяти.

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

   Выбор сортировки зависит от конкретной задачи. Одним из самых быстрых по среднему времени работы в Си является встроенный алгоритм qsort. Однако и он не лишен недостатков, так например, он не является устойчивым. При необходимости можно (непример, если в данных встречается много повторяющихся элементов) имплементировать собственную реализацию qsort, которая будет являться устойчивой.