|  |  |
| --- | --- |
| **Gerb-BMSTU_01** | **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**

**«ЗАПИСИ С ВАРИАНТАМИ. ОБРАБОТКА ТАБЛИЦ»**

Студент Артемьев Илья Олегович

Группа ИУ7 – 33Б

*2020 г.*

**Цель работы**

Цель работы – приобрести навыки работы с типом данных «запись» (структура), содержащим вариантную часть (объединение, смесь), и с данными, хранящимися в таблицах, произвести сравнительный анализ реализации алгоритмов сортировки и поиска информации в таблицах, при использовании записей с большим числом полей, и тех же алгоритмов, при использовании таблицы ключей; оценить эффективность программы по времени и по используемому объему памяти при использовании различных структур и эффективность использования различных алгоритмов сортировок.

**Условие задачи**

Создать таблицу, содержащую не менее 40-ка записей (тип – запись с вариантами (объединениями)). 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 обязательна).

Ввести 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.

**Техническое задание**

Приобрести навыки работы с типом данных «запись» (структура), содержащим вариантную часть (объединение, смесь), и с данными, хранящимися в таблицах, произвести сравнительный анализ реализации алгоритмов сортировки и поиска информации в таблицах, при использовании записей с большим числом полей, и тех же алгоритмов, при использовании таблицы ключей; оценить эффективность программы по времени и по используемому объему памяти при использовании различных структур и эффективность использования различных алгоритмов сортировок.

### Требования к функциональным характеристикам

Программа должна выполнять следующие функции:

* Производить поиск по таблице
* Сортировать таблицу
* Добавлять новую строку в таблицу
* Удалять строку из таблицы
* Выводить информацию о затраченном времени на сортировку
* Сообщить об ошибке ввода
* Сообщать о отсутствии элементов, удовлетворяющих поиску
* Сообщать о проблемах с открытием файлов

**Входные данные**

Пункт меню (число от 0 до 10), файл с записями абонентов, параметры добавляемого / удаляемого абонента.

**Выходные данные**

Текущее состояние таблицы, результаты сравнения эффективности сортировок, результаты поиска по заданным полям.

**Возможные аварийные ситуации**

1. Запуск программы с неправильными аргументами командной строки

Программа выведет сообщение «Ошибка: не удалось прочитать файл»

1. Ввод строки вместо числа

Программа выведет сообщение «Ошибка: вы ввели некорректный номер пункта меню»

1. Неправильный ввод статуса человека при занесении новой записи

Программа выведет сообщение «Ошибка: не удалось прочитать статус человека»

1. Неправильный ввод личной информации о человеке

Программа выведет сообщение «Ошибка: не удалось добавить новую запись»

1. При удалении не нашло человека с веденной фамилией

Программа выведет сообщение «Ошибка: не было найдено ни одного человека с данной фамилией»

## Способ обращения к программе

Программа представляет собой файл main.exe. Запускается в консоли. Для запуска достаточно команды ./main.exe + название файла.

**Описание структур данных**

**Структура основной таблицы**

typedef struct

{

char surname[SURNAME\_LENGHT]; // Фамилия, SURNAME\_LENGHT = 40

char name[NAME\_LENGHT]; // Имя, NAME\_LENGHT = 40

char phone\_number[PHONE\_NUMBER\_LENGTH]; // Номер телефона, PHONE\_NUMBER\_LENGTH = 15

char address[ADDRESS\_LENGHT]; // Адрес, ADDRESS\_LENGHT = 20

char status[STATUS\_LENGHT]; // Статус, STATUS\_LENGHT = 10

union // Объединение двух структур - статусов

{

struct

{

char post[POST\_LENGHT]; // Должность, POST\_LENGHT = 40

char service\_name[SERVICE\_LENGHT]; // Организация, SERVICE\_LENGHT = 40

} service; // Служебный статус

struct

{

int day; // День

int month; // Месяц

int year; // Год

} personal; // Личный статус

} type\_user;

} user;

**Структура таблицы ключей**

typedef struct

{

int index; // Индекс элемента в таблице

char surname[SURNAME\_LENGHT]; // Фамилия, SURNAME\_LENGHT = 40

} key;

**Описание алгоритма**

1. Пользователю выводится меню программы
2. Пока пользователь не введет 0, ему будет предложено вводить номера команд и выполнять различные действия

**Тестовые данные**

1. Тесты для главного меню

|  |  |
| --- | --- |
| **Входные данные** | **Выходные данные** |
| 3 | Вывод текущей таблицы |
| g | Ошибка: вы ввели некорректный номер пункта меню |
| 11 | Ошибка: вы ввели некорректный номер пункта меню |

1. Тесты для вывода таблицы

|  |  |
| --- | --- |
| **Входные данные** | **Выходные данные** |
| В таблице нет абонентов | Таблица пустая |
| В таблице один абонент | Один абонент |
| Таблица с абонентами | Таблица |

1. Тесты для ввода абонента

|  |  |
| --- | --- |
| **Входные данные** | **Выходные данные** |
| d, d, d, d | Ошибка: не удалось прочитать статус человека |
| d, d, d, d, 0, 0 | Ошибка: проверьте введенные данные |
| d, d, d, d, 0, 20, 1, 2001 | Занесение абонента в таблицу |
| d, d, d, d, 1, d, d | Занесение абонента в таблицу |

1. Тесты для удаления абонента из таблицы

|  |  |
| --- | --- |
| **Входные данные** | **Выходные данные** |
| Фамилия, которой нет в таблице | Ошибка: не было найдено ни одного человека с данной фамилией |
| Фамилия, которая имеется в таблице | Удаление людей с данной фамилией |

1. Тесты для нахождения друзей, которых нужно поздравить в ближайшую неделю

|  |  |
| --- | --- |
| **Входные данные** | **Выходные данные** |
| f | Ошибка: проверьте введенные данные |
| 20 5 1977 | Информация о людях |

**Замеры времени**

Измерение эффективности сортировок будет производиться в миллисекундах.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Количество записей | Пузырек | | Быстрая сортировка | |
|  | Основная таблица | Таблица ключей | Основная таблица | Таблица ключей |
| 50 | 29 | 29 | 11 | 10 |
| 100 | 149 | 122 | 18 | 17 |
| 200 | 419 | 409 | 49 | 35 |
| 500 | 2500 | 2468 | 88 | 64 |
| 1000 | 26621 | 22128 | 445 | 287 |
| 2000 | 102333 | 94367 | 617 | 486 |

Объем занимаемой памяти (длина массива структур = 2000):

|  |  |  |
| --- | --- | --- |
| **Количество записей** | **Основная таблица** | **Таблица ключей** |
| 50 | 20800000 | 4400000 |
| 100 | 41600000 | 8800000 |
| 200 | 83200000 | 17600000 |
| 500 | 208000000 | 44000000 |
| 1000 | 416000000 | 88000000 |
| 2000 | 832000000 | 176000000 |

|  |  |  |
| --- | --- | --- |
| **Количество записей** | **Отношение времени сортировки основной таблицы и таблицы ключей для метода пузырька** | **Отношение времени сортировки основной таблицы и таблицы ключей для быстрой сортировки** |
| 50 | 1 | 1,1 |
| 100 | 1,2 | 1,1 |
| 200 | 1 | 1,4 |
| 500 | 1 | 1,4 |
| 1000 | 1,2 | 1,6 |
| 2000 | 1,1 | 1,3 |

|  |  |  |  |
| --- | --- | --- | --- |
| Количество записей | % памяти, занимаемый таблицей ключей от всей таблицы | На сколько процентов сортировка таблицы ключей быстрее сортировки основной таблицы (“bubble sort”) | На сколько процентов сортировка таблицы ключей быстрее сортировки основной таблицы (“qsort”) |
| 50 | ~ 21% | 0% | ~10% |
| 100 | ~ 21% | ~22% | ~6% |
| 200 | ~ 21% | ~2% | 40% |
| 500 | ~ 21% | ~1% | ~37% |

**Функции**

**void menu(void)**

Функция печатает меню программы

**Аргументы**

**-**

**Возвращаемые значения**

**-**

**void print\_table\_header(int8\_t table\_number)**

Функция печатает заголовок таблицы

**Аргументы**

Table\_number – номер заголовка, который нужно выводить

**Возвращаемые значения**

**-**

**int8\_t print\_data(const user \*const data\_array, const int data\_len)**

Функция выводит в консоль данные таблицы

**Аргументы**

Data\_array – массив структур

Data\_len – длина массива структур

**Возвращаемые значения**

Успешное завершение функции

**void print\_error(const int error\_code)**

Функция печатает сообщение об ошибке по коду ошибки

**Аргументы**

Error\_code – код ошибки

**Возвращаемые значения**

-

**int8\_t add\_entry(user \*data\_array, int \*const data\_len)**

Функция добавляет запись в конец таблицы

**Аргументы**

Data\_array – массив структур

Data\_len – длина массива структур

**Возвращаемые значения**

Код ошибки или успешное завершение функции

**void delete\_entry(user \*data\_array, int \*const data\_len)**

Функция удаляет записи из таблицы по введенной фамилии

**Аргументы**

Data\_array – массив структур

Data\_len – длина массива структур

**Возвращаемые значения**

**-**

**int8\_t data\_fill(FILE \*f, user \*data\_array, int \*const data\_len)**

Функция формирует массив структур по данным из файла

**Аргументы**

F – файл из которого ведется считывание

Data\_array – массив структур

Data\_len – длина массива структур

**Возвращаемые значения**

Код ошибки или успешное завершение функции

**void make\_temp\_file(const char \*const temp\_file\_name)**

Функция формирует временный файл

**Аргументы**

Temp\_file\_name – название файла

**Возвращаемые значения**

**-**

**void clear\_file(FILE \*temp\_file, const char \*const temp\_file\_name)**

Функция очищает файл

**Аргументы**

Temp\_file – файл, который нужно очистить

Temp\_file\_name – название файла, который нужно очистить

**Возвращаемые значения**

**-**

**void temp\_file\_fill(FILE \*temp\_file, const user \*const data\_array, const int data\_len)**

Функция заполняет временный файл

**Аргументы**

Temp\_file – файл, который нужно заполнить

Data\_array – массив структур, с помощью которого нужно заполнить файл

Data\_len – длина массива структур

**Возвращаемые значения**

**-**

**int8\_t user\_find(const user \*const data\_array, const int data\_len)**

Функция ищет абонента, у которого день рождения в ближайшую неделю

**Аргументы**

Data\_array – массив структур, с помощью которого нужно заполнить файл

Data\_len – длина массива структур

**Возвращаемые значения**

**-**

**int8\_t parse\_args(const int argc, const char \*\*argv, int \*number\_args, char \*file\_name)**

Функция парсит аргументы командной строки

**Аргументы**

Argc – количество аргументов командной строки

Argv – аргументы командной строки

Number\_args – количество аргументов командной строки

File\_name – название файла, откуда считываются данные

**Возвращаемые значения**

Код ошибки или успешное завершение функции

**void all\_files\_close(FILE \*main\_file, FILE \*temp\_file)**

Функция закрывает все файлы

**Аргументы**

Main\_file – основной файл

Temp\_file – дополнительный файл

**Возвращаемые значения**

-

**void key\_table\_fill(key \*table, const user \*const data\_array, int \*const key\_len, const int data\_len)**

Функция заполняет массив структур для таблицы ключей

**Аргументы**

Table – массив структур таблицы ключей

Data\_array – массив структур основной таблицы

Key\_len – длина массива структур для таблицы ключей

Data\_len – длина массива структур для основной таблицы

**Возвращаемые значения**

-

**Ответы на контрольные вопросы**

1. Как выделяется память под вариантную часть записи?

Память под вариативную часть выделяется таким образом, чтобы ее хватало на самую большую возможную структуру.

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

Результат будет системно-зависимым и трудно предсказуемым. Возможно, произойдет приведение типов.

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

Программист.

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

Таблица ключей представляет из себя некую структуру, содержащую индекс и значение некоторого выбранного поля таблицы (исходной). Используется для реализации более эффективного выполнения перемещения (сортировок).

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

Когда много данных с большим количеством полей, эффективнее обрабатывать данные с использованием ключей. Иначе таблицу.

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

Для таблиц из большого количества записей предпочтительно использовать стандартные и устойчивые способы сортировки, со средним временем обработки O(n\*log n), такие как QuickSort, MergeSort и т.д. Если же в таблице не так много записей, то предпочтительнее использовать простые алгоритмы сортировки, например, сортировку пузырьком.

**Вывод по проделанной работе**

В результате работы были реализованы функции сортировки таблицы (исходной) и таблицы ключей (QuickSort и BubbleSort). Также поиск и добавление абонентов в структуре с вариантными полями. Если сортировать по строкам, то разница в приросте времени при сравнении сортировки ключей и таблицы будет незначительной по сравнению с сортировкой по числам. Для сортировки таблицы ключей малого размера удобнее использовать bubble sort, как видно из таблицы на малом количестве записей (100) сортировка таблицы ключей на 20% быстрее сортировки основной таблицы, а для сортировки таблицы ключей большего размера удобнее использовать qsort, так как видно из таблицы, на (500) количествах записи сортировка таблицы ключей на 37% быстрее сортировки основной таблицы. Для сортировки гораздо выгоднее использовать таблицу ключей, так как на ее обработку уходит меньше времени, но программе потребуется больше памяти.