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

ОТЧЁТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №2

Название предмета: Типы и структуры данных

Студент: Варламова Екатерина Алексеевна

Группа: ИУ7-31Б

2020г.

I. Описание условия задачи

Ввести список машин (не менее 40 записей), имеющихся в автомагазине, содержащий: марку автомобиля, страну-производитель, цену, цвет и состояние: новый – гарантия (в годах); нет - год выпуска, пробег, количество ремонтов, количество собственников. Вывести цены не новых машин указанной марки с одним предыдущим собственником, отсутствием ремонта в указанном диапазоне цен. 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). Предоставить возможность добавления записей в конец таблицы и удаления записи по значению указанного поля.

II. Техническое задание

1. Описание исходных данных

Файл, имя которого задано первым аргументом командной строки при вызове программы.

В случае успешного чтения данных из файла программа ожидает номер действия, производимого над данными (фильтрация, добавление, удаление, сортировка, вывод).

Ниже представлено меню программы. Возможным действиям над данными соответствуют их номера в меню:

1. *Фильтрация*

Фильтрация производится в соответствии с условием задачи (не новые машины указанной марки с одним предыдущим собственником, отсутствием ремонта в указанном диапазоне цен).

Дополнительно требуется ввести: марку автомобиля (строка), минимальную цену (вещественное число) и максимальную цену (вещественное число). Допускается ввод сначала максимальной цены, затем минимальной.

2. *Добавление записи*

Добавление записи производится в конец исходной таблицы.

Дополнительно требуется ввести: данные об автомобиле.

То есть:

- марка автомобиля (строка)
- страна-производитель (строка)
- цвет (строка)
- цена (вещественное число)
- состояние (“old” или “new”)

если состояние = “old”:

- год выпуска (целое)
- пробег (целое)
- количество ремонтов (целое)
- количество собственников (целое)

если состояние = “new”:

- гарантия (целое)

3. Удаление записи

Удаляются все записи, значение поля которых совпадает с введённым значением. Предоставляется выбор поля, по которому производится удавление.

Дополнительно требуется ввести: название поля (инструкция по вводу будет доступна после запуска) и значение поля.

41. Сортировка по возрастанию цены автомобилей. Сортируется таблица ключей. Алгоритм: выбором (простой).
42. Сортировка по возрастанию цены автомобилей. Сортируется таблица ключей. Алгоритм: heapsort (ускоренный).
43. Сортировка по возрастанию цены автомобилей. Сортируется таблица ключей. Алгоритм: quicksort (ускоренный).
51. Сортировка по возрастанию цены автомобилей. Сортируется исходная таблица. Алгоритм: выбором (простой).
52. Сортировка по возрастанию цены автомобилей. Сортируется исходная таблица. Алгоритм: heapsort (ускоренный).

53. Сортировка по возрастанию цены автомобилей. Сортируется исходная таблица. Алгоритм: quicksort (ускоренный).
6. Вывод таблицы сравнения времени сортировок.
7. Вывод исходной таблицы записей.
8. Вывод таблицы ключей.
9. Вывод исходной таблицы по таблице ключей.
10. Чтение из файла ещё раз.
0. Завершение программы.

Формат ввода:

1. Данные об автомобилях в файле.

Каждая характеристика автомобиля располагается на новой строке. Характеристиками автомобиля при этом считаются:

- марка автомобиля (строка)
- страна-производитель (строка)
- цвет (строка)
- цена (вещественное число)
- состояние (“old” или “new”)

если состояние = “old”:

- год выпуска (целое)
- пробег (целое)
- количество ремонтов (целое)
- количество собственников (целое)

если состояние = “new”:

- гарантия (целое)

Таким образом, примерами корректного ввода данных автомобиля являются следующие записи:

Новый автомобиль	Старый автомобиль
toyota france 23.70 green	volvo UK 78.25 green

new 8	old 1960 854413 1 2

При этом между записями допускается любое количество символов перевода на новую строку.

2. Для действительных чисел допускается ввод в обычном и научном форматах.

Ограничения:

- Количество символов во всех строках не превышает 30 и не меньше 1;
- Количество записей об автомобилях не превышает 100 и не меньше 1;

2. Описание результата программы

В зависимости от введённого номера действия будет выведена разная информация:

- при запросе на фильтрацию (номер действия = 1) будет выведен список автомобилей, подходящих под заданные ограничения, или строка “No results”;
- при запросе на добавление (номер действия = 2) будет выведено сообщение об успешности операции или об ошибке при добавлении (превышено ограничение на количество записей, формат строк не удовлетворяет ограничениям, некорректный ввод);
- при запросе на удаление (номер действия = 3) будет выведен список автомобилей, оставшихся после удаления;
- при запросах на сортировку с таблицей ключей (номера действий = 41, 42, 43) будет отсортирована таблица ключей;
- при запросах на сортировку исходной таблицы (номера действий = 51, 52, 53) будет отсортирована исходная таблица;

- при запросе на вывод результатов использования различных алгоритмов сортировок (номер действия = 6) будет выведена таблица сравнения (все изменения таблиц будут произведены в копиях);
- при запросе на вывод текущего состояния таблицы (номер действия = 7) будет выведена таблица;
- при запросе на вывод текущего состояния таблицы ключей (номер действия = 8) будет выведена таблица ключей;
- при запросе на вывод исходной таблицы по ключам будет (номер действия = 9) будет выведена исходная таблица;
- при запросе на повторную загрузку данных из файла, данные будут загружены (номер действия = 10);

Формат вывода:

1. Таблица ключей выводится в следующем формате:

position: <pos1>; value: <val1>

position: <pos2>; value: <val2>

…

position: <posN>; value: <valN> , где :

pos1…posN – номера записей в исходной таблице

val1…valN – значение ключевого поля

2. Записи об автомобилях выводятся в следующем формате:

Новый автомобиль	Старый автомобиль
number: <num> trademark: < trademark > country: <country> price: <price> color: <color> guarantee: < guarantee >	number: <num> trademark: < trademark > country: <country> price: <price> color: <color> year: <year> mileage: <mileage> repairs: <repairs> owners: <owners>

Таким образом, пример вывода автомобиля:

Новый автомобиль	Старый автомобиль
number: 54 trademark: toyota country: france price: 23.700000 color: green guarantee: 8	number: 1 trademark: mazda country: belarus price: 16.570000 color: gray year: 1969 mileage: 515736 repairs: 1 owners: 3

3. *Описание задачи, реализуемой программой*

Программа в зависимости от запросов пользователя производит ряд действий (фильтрация, добавление, удаление, сортировка, вывод) над исходной таблицей записей и предоставляет возможность сравнить по времени сортировки при различных способах хранения данных и алгоритмах.

4. Способ обращения к программе

Вызов программы осуществляется с использованием аргументов командной строки (в качестве аргумента подаётся имя файла с данными, соответственно, вызов программы может выглядеть следующим образом: app.exe file.txt). Дальнейшие инструкции будут выведены после запуска.

5. Описание возможных аварийных ситуаций и ошибок пользователя

- ошибки, связанные с файлом (не указан аргумент командной строки, файл не существует);
- данные в файле выходят за рамки ограничений;
- данные в файле не соответствуют указанному выше формату;
- введённые данные выходят за рамки ограничений;
- введённые данные не соответствуют указанному выше формату;
- номер действия не существует;
- попытка добавления записи с выходом за количественное ограничение;

Во всех указанных случаях программа сообщит об ошибке.

III. Описание внутренних структур данных

1. Основная структура данных

Основной структурой данных в программе является статический массив записей с вариантной частью (запись содержит все характеристики автомобиля). Поскольку язык С не предоставляет встроенную СД “запись с вариантной частью”, её приходится реализовать с помощью использования в качестве поля записи union. Объявление массива записей на языке С выглядит следующим образом:

```
automobile_t automobiles[100]
```

При этом тип `automobile_t` определён так:

```
typedef struct
{
    string_t trademark;
    string_t country;
    double price;
    string_t color;
    condition_type_t condition_type;
    condition_t condition;
} automobile_t;
```

Вводятся так же и вспомогательные типы данных:

- Объединение (union), которое позволяет реализовать вариантную часть записи;

```
typedef union
{
    old_auto_t old;
    long guarantee;
} condition_t;
```

- Строка, реализованная как массив символьного типа;

```
typedef char string_t[30];
```

- Структура, которая является одним из вариантов в вариантной части основной записи;

```
typedef struct
{
    long year;
    long mileage;
    long repairs;
    long owners;
} old_auto_t;
```

- Перечисление;

```
typedef enum
{
    old, new
} condition_type_t;
```

Выбор описанной структуры данных (статический массив записей с вариантной частью) обусловлен:

- удобством обращения к данным;
- эффективной организацией хранения данных за счёт использования union: поскольку union объединяет вариантные поля в одну область памяти, нам не приходится хранить в каждой записи лишние (пустые) поля, что даёт выигрыш по памяти. Однако стоит заметить, что при использовании union возникает необходимость хранить информацию о том, какие данные хранятся в данной области памяти (в нашем случае эту роль играет перечисление). В общем случае затраты на это дополнительное поле компенсируются использованием объединения.

2. *Вспомогательная структура данных и сравнительный анализ подходов к сортировке*

По заданию необходимо отсортировать массив записей (таблицу) по определённому полю. В качестве такого поля было выбрано поле price, которое содержит информацию о цене автомобиля (имеет вещественный тип), Выбор обусловлен тем, что остальные поля имеют тип строка, а сравнение строк (операция, которая многократно совершается при сортировке) занимает больше времени, чем сравнение вещественных чисел.

К сортировке таблицы есть 2 подхода:

- отсортировать исходную таблицу;
- отсортировать массив ключей;

Главный недостаток 1-ого подхода заключается в том, что будет затрачено много времени на копирование элементов из одной области памяти в другую (при перестановке), особенно при большой размерности таблицы.

Именно эту проблему призван решить массив ключей, элементами которого являются структуры из 2 полей: номер элемента в исходном массиве и значение ключа. Его объявление на языке С выглядит следующим образом:

```
typedef struct
{
    size_t num;
    double value;
} keys_t;
keys_t keys[100];
```

С использованием этой вспомогательной структуры данных будет значительно ускорен процесс копирования элементов (поскольку каждый из них будет занимать меньше памяти). Однако общее количество затрачиваемой памяти будет увеличено.

Чтобы понять, насколько оправдано использование дополнительной памяти, измерим время сортировки исходной таблицы и таблицы ключей. При этом используем разные алгоритмы сортировки. Для 100 записей результаты сравнения эффективности работы программы получились следующие:

	heapsort	minmax	quicksort
с ключами	19.03	31.25	17.22
без ключей	209.40	59.40	66.85

Из полученных результатов очевидно, что выигрыш во времени составил:

- для простой сортировки (выбором) в ~2 раза;
- для qsort в ~ 4 раза;
- для heapsort в ~11 раз;

При этом в исходной таблице запись состоит из фиксированной (по длине) части: 3 строковых поля, 1 вещественное и 1 целое; и вариантной: 4 целочисленных (берём по максимуму). Даже без учёта выравнивания (!) получим, что 1 запись на 64-битной машине займёт: 30 * 3 + 1 * 8 + 1 * 4 + 4 * 8 = 134 байта, а 100 записей 13400 байт (с учётом выравнивания на конкретной машине размер занимаемой памяти составил 14400). В то же время массив ключей займёт 100 * (8 + 8) = 1600 байт. То есть мы увеличим память примерно в 1.1 раз.

Из полученных цифр очевидно, что использование массива ключей целесообразно всегда, когда объём занимаемой памяти не критичен (критичен он может быть в случае программирования, например, под какие-то микроконтроллеры, где память сильно ограничена). В частности, при решении нашей задачи использование массива ключей целесообразно. Примечание: сказанное верно при большой размерности таблицы.

IV. Описание алгоритма

По заданию необходимо отсортировать массив записей (таблицу), поэтому встаёт вопрос о выборе алгоритма сортировки.

Оценка разных алгоритмов

1. Простая сортировка выбором (minmax)

Суть: выбираем максимальный элемент из массива длины k, меняем элемент с последним и повторяем всё для массива длины k-1. Таких повторений следует провести k-2.

Таким образом, сортировка имеет оценку $\Theta(N^2)$. При этом количество перестановок в худшем случае равно N.

2. Сортировка кучей (heapsort)

Суть: предполагаем, что массив представляет собой полное бинарное дерево.

- Первым шагом выстраиваем кучу: начиная от первого «родителя», индекс которого можно найти по формуле N div 2 – 1 (индексация элементов с 0) и до 0-ого элемента, решаем, является ли элемент максимальным среди своих ближайших потомков. Если да, то он остается на месте, если нет, то он меняется местами с бОльшим его потомком, а для потомка производится та же операция (и дальше вниз по дереву производятся такие же операции в случае необходимости). Оценка сверху этого шага составляет O(N*log N).
- После первого шага максимальный элемент точно находится на вершине дерева (нулевой элемент). Поэтому меняем его местами с

последним элементом в массиве и укорачиваем рассматриваемый массив на единицу. Теперь для нулевого элемента решаем, является ли он максимальным среди своих ближайших потомков. Если да, то он остается на месте, если нет, то он меняется местами с бОльшим его потомком, а для потомка производится та же операция (и дальше вниз по дереву производятся такие же операции в случае необходимости). Оценка сверху этого шага составляет O(N*log N).

После описанных операций мы получим отсортированный по возрастанию массив. Таким образом, сортировка имеет оценку O(N*logN). Алгоритм реализован не рекурсивно, что также влияет на скорость работы. При этом количество перестановок в общем случае превышает N.

3. Быстрая сортировка (qsort)

Суть:

- выберем элемент из массива (возьмем тот, который посередине), назовём его опорным;
- перераспределим элементы в массиве таким образом, что элементы меньше опорного поместятся перед ним, а больше или равные после;
- рекурсивно применим первые два шага к двум подмассивам слева и справа от опорного элемента (рекурсия не применяется к массиву, в котором только один элемент или отсутствуют элементы);

После описанных операций мы получим отсортированный по возрастанию массив. Таким образом, сортировка имеет оценку в среднем и лучшем случае O(N*logN), $O(N^2)$ в худшем. При этом количество перестановок в общем случае превышает N.

Выводы по оценкам

Мы получаем, что при сортировке исходной большой таблицы (где перестановка элементов занимает много времени) будет наиболее эффективен алгоритм простой сортировки, поскольку количество перестановок в нем имеет верхнюю границу N. При этом, как мы выяснили в предыдущем пункте (описание СД), при решении нашей задачи целесообразно использование

таблицы ключей при сортировке, а значит перестановка элементов (в таблице ключей) не является фактором, который более всего влияет на время выполнения. Соответственно, мы можем приближённо считать, что перестановка выполняется за O(1), поэтому выбор эффективного алгоритма основывается на оценке сложности алгоритмов. Так как heapsort и qsort в среднем выполняются за O(N*logN), а простая за $O(N^2)$, то очевидно, что первые 2 выполнятся значительно быстрее (qsort немного быстрее).

Подтвердим сказанное экспериментально, обратившись к уже приведённой таблице:

	heapsort	minmax	quicksort
с ключами	19.03	31.25	17.22
без ключей	209.40	59.40	66.85

V. Тестирование

1. Позитивные тесты

Во всех позитивных тестах исходный список машин такой:

```
volvo
UK
78.25
green
old
1960
854413
1
2

toyota
france
23.70
green
new
8

renault
italy
45.25
blue
```

new
15

mazda
belarus
16.57
gray
old
1969
515736
1
3

honda
germany
53.52
purple
old
1960
152259
3
1

toyota
germany
99.61
black
old
1959
901288
1
0

kia
china
47.78
white
new
9

audi
china
89.91
red
old

1981
673610
0
1

toyota
russia
88.08
green
old
1980
423185
4
2

mazda
spain
57.97
blue
new
17

Входные данные (делимое, затем делитель)	Описание теста	Результат
1 audi 80.98 90 1 toyota 80.98 90 0	Проверка фильтрации	trademark: audi country: china price: 89.910000 color: red year: 1981 mileage: 673610 repairs: 0 owners: 1 No results
3 trademark toyota 0	Проверка удаления	trademark: volvo country: UK price: 78.250000 color: green year: 1960 mileage: 854413 repairs: 1 owners: 2 trademark: renault country: italy

		price: 45.250000 color: blue guarantee: 15 trademark: mazda country: belarus price: 16.570000 color: gray year: 1969 mileage: 515736 repairs: 1 owners: 3 trademark: honda country: germany price: 53.520000 color: purple year: 1960 mileage: 152259 repairs: 3 owners: 1 trademark: kia country: china price: 47.780000 color: white guarantee: 9 trademark: audi country: china price: 89.910000 color: red year: 1981 mileage: 673610 repairs: 0 owners: 1 trademark: mazda country: spain price: 57.970000 color: blue guarantee: 17

41 0	Проверка простой сортировки с таблицей ключей	position: 3; value: 16.570000 position: 1; value: 23.700000 position: 2; value: 45.250000 position: 6; value: 47.780000 position: 4; value: 53.520000 position: 9; value: 57.970000 position: 0; value: 78.250000 position: 8; value: 88.080000 position: 7; value: 89.910000 position: 5; value: 99.610000 trademark: mazda country: belarus price: 16.570000 color: gray year: 1969 mileage: 515736 repairs: 1 owners: 3 trademark: toyota country: france price: 23.700000 color: green guarantee: 8 trademark: renault country: italy price: 45.250000 color: blue guarantee: 15 trademark: kia country: china price: 47.780000 color: white

		guarantee: 9 trademark: honda country: germany price: 53.520000 color: purple year: 1960 mileage: 152259 repairs: 3 owners: 1 trademark: mazda country: spain price: 57.970000 color: blue guarantee: 17 trademark: volvo country: UK price: 78.250000 color: green year: 1960 mileage: 854413 repairs: 1 owners: 2 trademark: toyota country: russia price: 88.080000 color: green year: 1980 mileage: 423185 repairs: 4 owners: 2 trademark: audi country: china price: 89.910000 color: red year: 1981 mileage: 673610 repairs: 0 owners: 1 trademark: toyota

		country: germany price: 99.610000 color: black year: 1959 mileage: 901288 repairs: 1 owners: 0
42 0	Проверка heapsort сортировки с таблицей ключей	То же, что в предыдущем
43 0	Проверка qsort сортировки с таблицей ключей	То же, что в предыдущем
51 0	Проверка простой сортировки без таблицы ключей	trademark: mazda country: belarus price: 16.570000 color: gray year: 1969 mileage: 515736 repairs: 1 owners: 3 trademark: toyota country: france price: 23.700000 color: green guarantee: 8 trademark: renault country: italy price: 45.250000 color: blue guarantee: 15 trademark: kia country: china price: 47.780000 color: white guarantee: 9 trademark: honda country: germany price: 53.520000 color: purple year: 1960

		mileage: 152259 repairs: 3 owners: 1 trademark: mazda country: spain price: 57.970000 color: blue guarantee: 17 trademark: volvo country: UK price: 78.250000 color: green year: 1960 mileage: 854413 repairs: 1 owners: 2 trademark: toyota country: russia price: 88.080000 color: green year: 1980 mileage: 423185 repairs: 4 owners: 2 trademark: audi country: china price: 89.910000 color: red year: 1981 mileage: 673610 repairs: 0 owners: 1 trademark: toyota country: germany price: 99.610000 color: black year: 1959 mileage: 901288 repairs: 1 owners: 0

52 0	Проверка heapsort сортировки без таблицы ключей	То же, что в предыдущем
53 0	Проверка qsort сортировки без таблицы ключей	То же, что в предыдущем

Негативные тесты:

Входные данные (делимое, затем делитель)	Описание теста	Результат
	Список машин в файле пуст	error while reading file
файл с 101 записью (все записи корректны)	Размер списка машин в файле превышает максимально дпустимый	error while reading file
	Файл с данными не существует	no such file
	После успешного чтения данных из файла не выбрано ни одно действие над данными	nothing was chosen
2 audi rus red 13.4 new 12 0	При добавлении в исходную таблицу новой записи произошла ошибка чтения записи (вместо вещественного значения подана строка)	error while reading
2 audi rus red 0	При добавлении в исходную таблицу новой записи произошла ошибка чтения записи (введены не все данные записи)	error while reading
3 audi 0	При удалении не указано название поля, по которому будет произведён поиск	error while reading type of field or value of field
1 audi price 0	При фильтрации неверно заданы вещественные поля	error while reading filter values
3 trademark	При удалении удалятся все элементы	all elements were deleted

audi 0		
1 aaaaaaaaaaaaaaa aaaaaaaaaaaaaaaa 1.1 1.2 0	При фильтрации у строкового поля превышена максимальная длина	error while reading filter values

VI. Выводы по проделанной работе

В ходе работы был проведён сравнительный анализ подходов к обработке больших таблиц с данными: были реализованы разные алгоритмы и организации хранения данных. Установилось, что если каждая запись в исходной таблице содержит большое количество полей, а для обработки используется лишь несколько из них, то эффективнее работать с данными в отдельной таблице – таблице ключей. Иначе, если в обработке используются почти все поля (например, нужно отсортировать по всем полям) или количество полей не столь большое, то имеет смысл обрабатывать данные в самой таблице. При этом в качестве ключа лучше выбирать такое поле, которое можно быстро сравнить, то есть вещественные или целые числа (например, сравнение строк требует внутренний цикл, что увеличивает время сравнения). Кроме того, при небольшом количестве записей в исходной таблице так же нет смысла создавать дополнительную таблицу ключей.

VII. Ответы на вопросы

1. Память под вариантную часть записи выделяется по максимальному полю вариантной части.

2. Если в вариантную часть ввести данные, несоответствующие описанным, то при обращении к данным они будут интерпретироваться как описанные.

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

4. Таблица ключей представляет собой массив структур, состоящих из значения, по которому происходит сортировка, и индекса записи с этим

значением в исходной таблице. Таблица ключей нужна для ускорения работы с исходной таблицей (например, для её сортировки).

5. Если каждая запись в исходной таблице содержит большое количество полей, а для обработки используется лишь несколько из них, то эффективнее работать с данными в отдельной таблице – таблице ключей. Иначе, если в обработке используются почти все поля (например, нужно отсортировать по всем полям) или количество полей не столь большое, то имеет смысл обрабатывать данные в самой таблице. Кроме того, при небольшом количестве записей в исходной таблице так же нет смысла создавать дополнительную таблицу ключей.

6. Если сортировать исходную таблицу, то предпочтительнее использовать такие алгоритмы сортировки, в которых в среднем совершается наименьшее количество перестановок. Если сортировать массив ключей, то предпочтительнее использовать любой ускоренный алгоритм (heapsort, treesort, qsort, mergesort и тд).