**Отчет по лабораторной работе №2**

**Пронина Арсения, ИУ7-32Б**

**Вариант 25**

# Условия задачи

Создать таблицу, содержащую не менее 40-ка записей (тип – запись с вариантами). Упорядочить данные в ней по возрастанию ключей, где ключ – любое невариантное поле (по выбору программиста), используя:

• саму таблицу

• массив ключей

(возможность добавления и удаления записей в ручном режиме обязательна).

Ввести список машин, 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.

# Техническое задание

## Входные данные

1) На вход программе подаётся файл с именем car\_list.txt который содержит таблицу записи в формате: *Brand; Country; price; color; new/old; guarantee/year\_of\_release; probeg; repair\_count; owners\_count;*

2) Целое число от 0 до 12 (для ввода номера команды из меню)

3) Строка или целое число в зависимости от вводимого поля (при вводе новой записи), в том же формате что и в файле

4) Целое число от 0 до 9 при вводе номера поля (для вывода или удаления по полю или его значению)

5) Строка, содержащая только числа от 0 до 9 и пробелы (для вывода таблицы по значениям нескольких полей)

6) Строка или целое число в зависимости от вводимого поля (при вводе значения поля для удаления из списка или вывода в консоли)

## Требования к входным данным

1. Одна запись вводится в одну строчку
2. После каждого поля должен быть разделительный символ ‘;’
3. Строковые поля могут содержать любые символы кроме ‘;’
4. Целочисленные поля должны содержать только цифры и пробелы
5. Поле, отвечающее за new/old воспринимает соответственно только

строчки “new” и “old”

1. Между цифрами в целочисленных полях могут быть введены пробелы (например, “1 000 000” считается как 1000000)
2. Кол-во полей 1ой записи должно соответствовать следующим критериям

* 6 в случае, когда машина новая
* 9 в случае, когда машина старая

1. При вводе номеров полей для вывода таблицы по значениям нескольких полей, строка может содержать только цифры и пробелы.

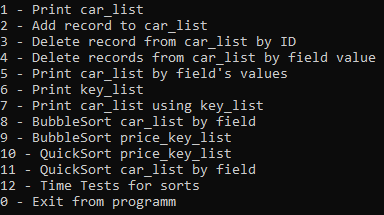
## Выходные данные

Вывод списка машин, таблицы ключей или информации о скорости сортировок в зависимости от команды.

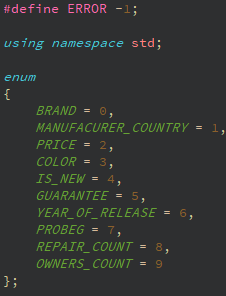
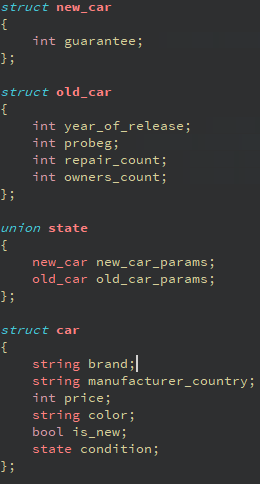
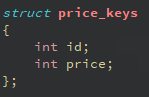
В случае некорректных входных данных или ввода будет выведено соответствующее сообщение.

## Функции программы

При запуске программы она считывает файл с именем car\_list.txt из которого берётся исходный список машин. Вся остальная часть программы представляет из себя консольное приложение исполняющее следующие команды:



## Структуры данных

# Описание алгоритма

Вся информация о таблице записей о машинах хранится в динамическом массиве типа car\* (структура car описана выше). Также информация о таблице ключей соответственно в динамическом массиве типа price\_keys\* (тоже описанного выше). А далее описаны функции, работающие с этими массивами.

### Набор функций, используемых в меню:

1. void **print\_car\_list**(car \*cars\_list, int size\_of\_list)

*Функция* *вывода* *текущего* *списка* *машин*

1. int **read\_record** (string input, car \*car\_record\_rez)

*Функция* *считывания* *записи* *о* *машине* *из* *строки*

1. int **add\_record** (car \*\*car\_list, int size\_of\_list, car car\_record)

*Функция* *добавления* *одной* *записи* *в* *конец* *списка* *машин*

1. int **delete\_record**(car \*\*car\_list, int size\_of\_list, int id)

*Функция* *удаления* *записи* *из* *списка* *машин* *по* *номеру*

1. int **delete\_car\_record\_by\_pole**(int pole, string znach, car \*\*car\_list, int \*size\_of\_list) *Функция* *удаления* *записи* *из* *списка* *машин* *по* *значению* *конкретного* *поля (использует* **delete\_record**()*)*
2. int **print\_car\_record\_by\_poles**(int pole[10], string znach[10], car \*\*car\_list, int size\_of\_list) *Функция* *вывода* *списка* *машин* *по* *значениям* *одного* *или* *нескольких* *полей*
3. void **print\_price\_key\_list**(price\_keys \*key\_list, int size\_of\_list)

*Функция* *вывода* *таблицы* *ключей*

1. void **print\_car\_list\_by\_price\_keys**(car \*cars\_list, int size\_of\_list, price\_keys \*key\_list) *Функция* *вывода* *списка* *машин* *с* *использованием* *таблицы* *ключей*
2. void **car\_sort\_puz**(car \*\*mas, int size\_of\_mas, int (\*comparator)(*const* car\*, *const* car\*)) *Функция* *сортировки* *списка* *машин* *пузырьком*
3. void **price\_keys\_sort\_puz**(price\_keys \*\*mas, int size\_of\_mas)

*Функция* *сортировки* *таблицы* *ключей* *пузырьком*

1. void **price\_keys\_sort\_qsort**(price\_keys \*a, int l, int r)

*Функция* *сортировки* *таблицы* *ключей* *быстрой* *сортировкой*

1. void **car\_sort\_qsort**(car \*a, int l, int r, int (\*comparator)(*const* car\*, *const* car\*)) *Функция* *сортировки* *списка* *машин* *быстрой* *сортировкой*
2. int **read\_file**(string file\_name, car \*\*car\_list, int \*size\_of\_list) *Функция* *считывания* *списка* *машин* *из* *файла (Использует* **read\_record**() и **add\_record**()*)*
3. void **car\_sort\_puz\_test**(car \*\*mas, int size\_of\_mas, int (\*comparator)(*const* car\*, *const* car\*), int mn) *Функция* *тестирования* *сортировки* *списка* *машин* *пузырьком* *на* *время*
4. void **price\_keys\_sort\_puz\_test**(price\_keys \*\*mas, int size\_of\_mas, int mn) *Функция* *тестирования* *сортировки* *таблицы* *ключей* *пузырьком* *на* *время*
5. void **car\_sort\_qsort\_test**(car \*a, int l, int r, int (\*comparator)(*const* car\*, *const* car\*), int mn) *Функция* *тестирования* *сортировки* *списка* *машин* *быстрой* *сортировкой* *на* *время*
6. void **price\_keys\_sort\_qsort\_test**(price\_keys \*a, int l, int r, int mn) *Функция* *тестирования* *сортировки* *таблицы* *ключей* *быстрой* *сортировкой* *на* *время*

### Вспомогательные функции:

1. void **print\_car\_record**(car car\_record)

*Функция* *вывода* *отдельной* *записи* *о* *машине*

1. int **string\_to\_int**(string input, int \*rez)

*Функция* *перевода* *строки* *в* *число*

1. void **car\_list\_copy** (car \*car\_list1, car \*car\_list2, int size\_of\_list)

*Функция* *копирования* *списка* *машин*

1. void **swap\_car\_records**(car \*car\_list1, car \*car\_list2)

*Функция* *обмена* *записей* *о* *машинах* *местами*

1. void **swap\_price\_keys**(price\_keys \*key1, price\_keys \*key2)

*Функция* *обмена* *ключей* *местами* *в* *таблице*

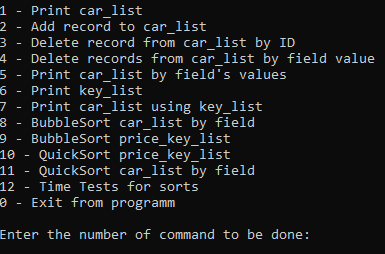
### Функции сравнения:

1. int **comp\_string**(*const* string \*str1, *const* string \*str2)
2. int **comp\_car\_brand**(*const* car \*i, *const* car \*j)
3. int **comp\_car\_manufacturer\_country**(*const* car \*i, *const* car \*j)
4. int **comp\_car\_price**(*const* car \*i, *const* car \*j)
5. int **comp\_car\_color**(*const* car \*i, *const* car \*j)
6. int **comp\_car\_is\_new**(*const* car \*i, *const* car \*j)
7. int **comp\_car\_guarantee**(*const* car \*i, *const* car \*j)
8. int **comp\_car\_year\_of\_release**(*const* car \*i, *const* car \*j)
9. int **comp\_car\_probeg**(*const* car \*i, *const* car \*j)
10. int **comp\_car\_repair\_count**(*const* car \*i, *const* car \*j)
11. int **comp\_car\_owners\_count**(*const* car \*i, *const* car \*j)

*Описание функции вытекает из её названия*

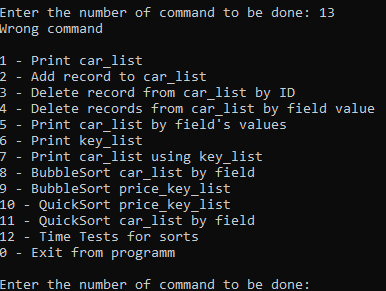
# Тесты

### Вот так выглядит меню:



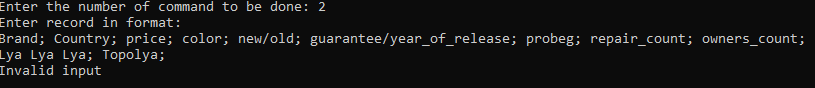
### Некорректный ввод:

1. Некорректная команда





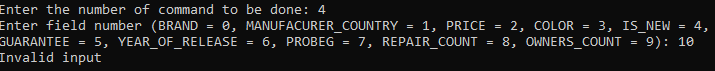
1. Команда 2 (добавление записи)

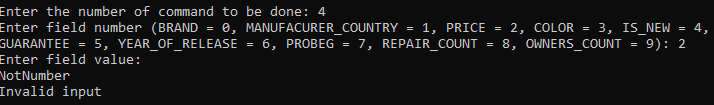


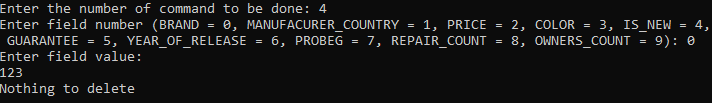
1. Команда 3 (Удаление записи по номеру)



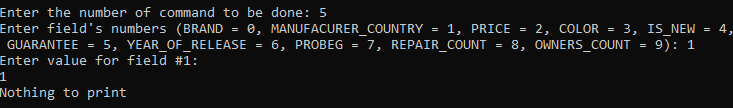
1. Команда 4 (Удаление записи по значению поля)

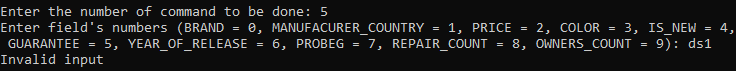




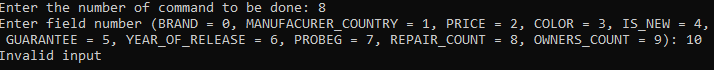


1. Команда 5 (вывод списка машин по значениям полей)

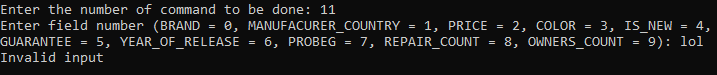




1. Команда 8 (сортировка списка машин пузырьком)



1. Команда 11 (сортировка списка машин быстрой сортировкой)



# Использование типа “запись” с вариантной частью

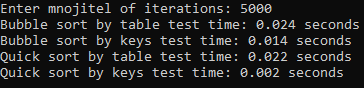
Поскольку нет необходимости хранить оба поля объединения **state** одновременно, то можно сэкономить память на хранении этой информации используя *union*.

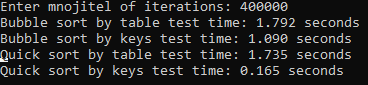
# Сравнительный анализ

С помощью sizeof определим память занимаемую переменными типа **car** и **price\_keys**.

|  |  |  |
| --- | --- | --- |
|  | Список машин | Таблица ключей |
| Память (N – кол-во элементов) | N\*128 байт | N\*8 байт |

Из таблицы памяти мы видим, что таблица ключей увеличивает кол-во используемой памяти на ~6% (8 байт за каждую запись).





|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Кол-во зап. | Список Bubble | Ключи Bubble | Список Qsort | Ключи Qsort |
| 5000\*40 | 0.024 sec | 0.014 sec | 0.022 sec | 0.002 sec |
| 400000\*40 | 1.792 sec | 1.090 sec | 1.735 sec | 0.165 sec |

* Список Bubble – void **car\_sort\_puz**(…)
* Ключи Bubble – void **price\_keys\_sort\_puz**(…)
* Список Qsort – void **car\_sort\_qsort**(…)
* Ключи Qsort – void **price\_keys\_sort\_qsort**(…)

Из таблицы времени выполнения сортировок мы видим, что при не очень большом списке эффективность таблицы ключей на ~71% (в 1.71 раз) больше при использовании BubbleSort, и на 1000% (в 11 раз) больше при использовании QuickSort. При значительном увеличении списка (в 80 раз больше чем в первом примере) эффективность таблицы ключей возрастает на ~64% (в 1.64 раз) при использовании BubbleSort, и на ~950% (в 10.5 раз) при использовании QuickSort. (Очевидно, что при большем кол-ве записей точность вычисления эффективности выше). Также можем сравнить эффективность сортировок по отдельности:

1. При не большом списке:

* QuickSort эффективней BubbleSort на 9% (в 1.09 раз) при использовании самого списка
* QuickSort эффективней BubbleSort на 600% (в 7 раз) при использовании таблицы ключей

1. При большом списке:

* QuickSort эффективней BubbleSort на 3% (в 1.03 раз) при использовании самого списка
* QuickSort эффективней BubbleSort на 560% (в 6.6 раз) при использовании самого списка

# Контрольные вопросы

**1.Как выделяется память под вариантную часть записи?**

Выделяется общий блок памяти = размеру наибольшего поля. Она является общей для каждого поля.

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

Данные будут заменены, но будут нести не корректную информацию. Поскольку тип данных в вариантной части при компиляции не проверяется поведение программы не определенно.

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

Программист (При обработке записей с вариантами программисту необходимо следить за правильностью хранения и обработки данных, содержащихся в вариантной части.)

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

Матрица индексов ключевого поля и соответствующих им значений. Нужна для экономии времени.

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

Таблицу ключей следует использовать при большом кол-ве записей. Но следует учитывать, что для хранения этой таблицы требуется дополнительная память, а также если в качестве ключа используется символьное поле записи, то это влечет за собой необходимость посимвольной обработки данного поля в цикле, и, следовательно, приводит к увеличению времени выполнения любых операций. Выбор данных из основной таблицы в порядке, определенном таблицей ключей, так же замедляет вывод этих данных. Если же исходная таблица содержит небольшое число полей, то выгоднее обрабатывать данные в самой таблице.

# Вывод

Если нет необходимости использовать обе переменные одновременно, то для экономии памяти можно воспользоваться объединениями (*union*). При сортировке больших таблиц выгодно использовать таблицу ключей для экономии времени, при этом не забывая о том, что для её хранения используется дополнительная память.