**Отчет по лабораторной работе №2**

**Пронина Арсения, ИУ7-32Б**

**Вариант 25**

# Условия задачи

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

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). Осуществить поиск информации по варианту.

## Входные данные

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;*

## Требования к входным данным

1. Одна запись вводится в одну строчку
2. После каждого поля должен быть разделительный символ ‘;’
3. Именные части могут содержать любые символы
4. Целочисленные поля должны содержать только цифры и пробелы
5. Поле, отвечающее за new/old воспринимает соответственно только строчки “new” и “old”
6. Между цифрами в целочисленных полях могут быть введены пробелы (например, “1 000 000” считается как 1000000)
7. Кол-во полей 1ой записи должно соответствовать следующим критериям

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

# Структуры данных

*struct* **new\_car**

{

int guarantee;

};

*struct* **old\_car**

{

int year\_of\_release;

int probeg;

int repair\_count;

int owners\_count;

};

*union* **state**

{

new\_car new\_car\_params;

old\_car old\_car\_params;

};

*struct* **car**

{

string brand;

string manufacturer\_country;

int price;

string color;

bool is\_new;

state condition;

};

*struct* **price\_keys**

{

int id;

int price;

};

# 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**()*)*

1. 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*
2. void **price\_keys\_sort\_puz\_test**(price\_keys \*\*mas, int size\_of\_mas, int mn) *Функция* *тестирования* *сортировки* *таблицы* *ключей* *пузырьком* *на* *время*
3. 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*
4. 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*

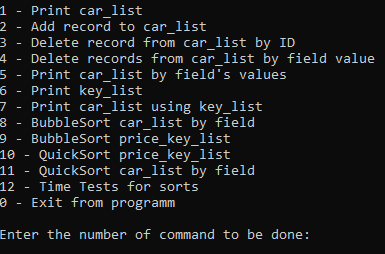
### Функции сравнения:

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*

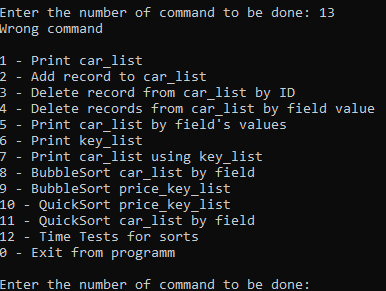
# Тесты

### Вот так выглядит меню:



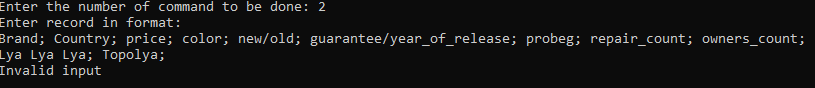
### Некорректный ввод:

1. Некорректная команда





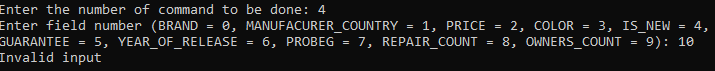
1. Команда 2 (добавление записи)

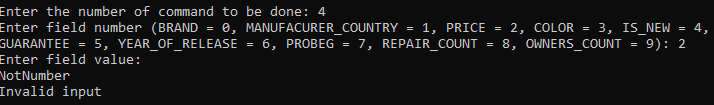


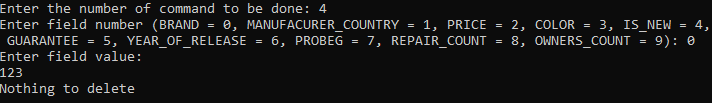
1. Команда 3 (Удаление записи по номеру)



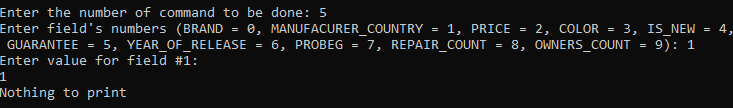
1. Команда 4 (Удаление записи по значению поля)

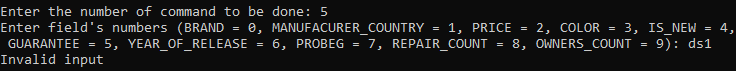




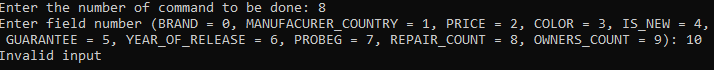


1. Команда 5 (вывод списка машин по значениям полей)

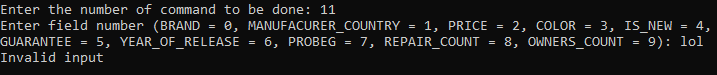




1. Команда 8 (сортировка списка машин пузырьком)



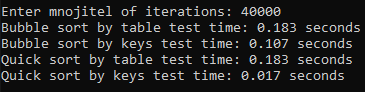
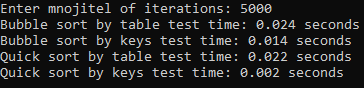
1. Команда 11 (сортировка списка машин быстрой сортировкой)



# Тип “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



(показать в процентах разницу в скорости и памяти (таблицей))

Чем больше таблица, тем наглядней видно, что сортировка с помощью ключей экономит значительное количество времени.

Тем не менее использование дополнительного массива влечёт за собой использование дополнительной памяти.

При всём это необходимо не забывать, что при использовании строкового типа ключа эффективность этого метода снижается из-за необходимости сравнивать поля посимвольно.

# Контрольные вопросы

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

Выделяется общий блок памяти = размеру наибольшего поля

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

Данные будут заменены, но будут нести не корректную информацию.

**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 при большом кол-ве записей. Но следует учитывать, что 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.

# Вывод

(вариатные поля + таблица ключей) /\*Я приобрёл навыки работы с типом данных «запись» (структура, struct), содержащим вариантную часть (объединение, смесь, union), и с данными, хранящимися в таблицах. Произвёл сравнительный анализ реализации разных алгоритмов сортировки (bubble и qsort), при использовании записей с большим числом полей, и тех же алгоритмов, при использовании таблицы ключей. А также оценил эффективность программы по используемому объему памяти при использовании объединений. \*/