



ТЕХНИЧЕСКИ УНИВЕРСИТЕТ – СОФИЯ

ФАКУЛТЕТ “КОМПЮТЪРНИ СИСТЕМИ И УПРАВЛЕНИЕ”

Катедра „Програмиране и Компютърни Технологии”

Курсова Задача по
Структури от Данни и приложни
алгоритми

Програма тип „меню“ за поддръжка на
каталог за автокъща

Изготвил Задачата:

Стефан Байчев

Email: sbaychev@gmail.com

ФКСУ - КТПП

Факултетен Номер 281611007

Магистър - изравнително обучение

I курс II семестър

Преподавател:.....

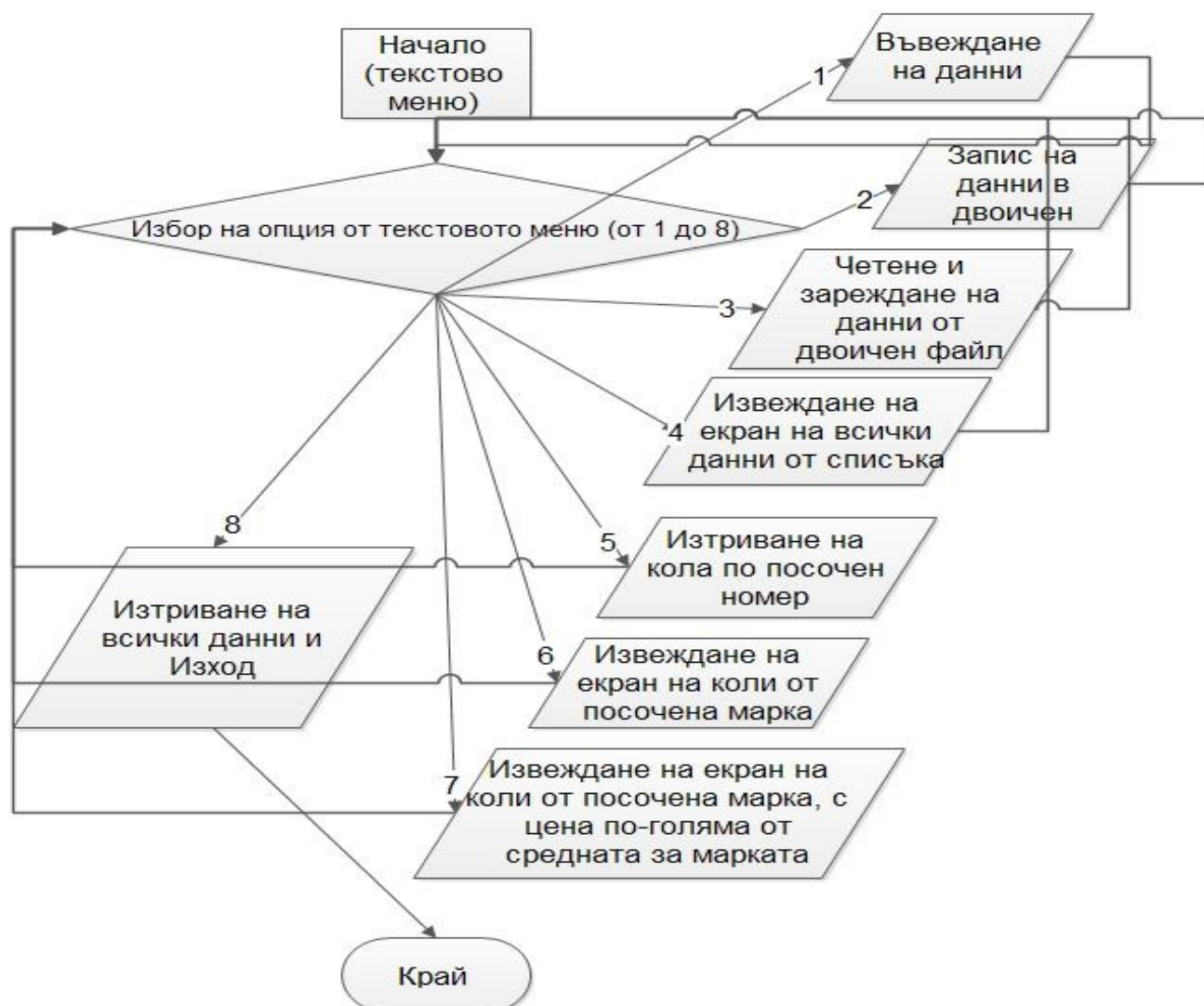
/доц. д-р Юлиана Георгиева/

София, 05.06.2012 г.

Съдържание

Обобщен Блоков Алгоритъм.....	3
Описание на използваните Функции.....	3
Функции използвани във връзка със въвеждането на данни - Опция 1.....	4
Функция за запис на вече въведени данни (при едно и също изпълнение на програмата) в двоичен файл – Опция 2	5
Функция за четене на данни от двоичен файл – Опция 3	6
Функциите извършващи печатането на екрана на конзолата всички заредени в локалната памет данни от свързаният списък – Опция 4	6
Функция за изтриване на елемент от свързаният списък по посочена данна – Опция 5	7
Функции осигоряващи търсенето и принтирането на елементи от свързаният списък по посочена данна – Опция 6	8
Функции осигоряващи търсенето и принтирането на елементи от едносвързаният списък по посочена данна и ако отговарят на зададено условие – Опция 7	9
Функции за изтриване на елементите от свързаният списък и излизане от програмата – Опция 8.....	10
Source Code на програмата	10
myFunc.c	10
myFunc.h.....	15
main.c	15
Контролен Пример от изпълнението на програмата	19

Обобщен Блоков Алгоритъм



При изпълнение на която и да е от възможните за избор Опция (от 1 до 8), при коректно или некоректно такова, програмата се връща в изначалната точка за избор на Опция от текстовото меню – това изключва Опция 8 разбира се, защото чрез нея се осъществява изходът от програмата.

Описание на използваните Функции

car Struct Референция

```
typedef struct car BODY
```

Полета за Данни:

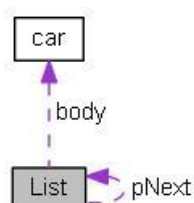
```
char carID [MIN_LENGTH+1]
char color [MAX_LENGTH+1]
float carEngineSize
char carBrand [MAX_LENGTH+1]
double carPrice
```

List Struct Референция

```
typedef struct List LIST
```

Полета за Данни:

```
BODY body
struct List * pNext
```



фиг.1 Диаграма на взаимовръзката за List

Общ вид на използваните функции за реализиране на поставената задача:

- ✓ brandAvg() : myFunc.c , myFunc.h
- ✓ delByCarID() : myFunc.c , myFunc.h
- ✓ delFirst() : myFunc.h , myFunc.c
- ✓ displAllGAvgForBrand() : myFunc.c , myFunc.h
- ✓ displByBrand() : myFunc.h , myFunc.c
- ✓ enterBody() : myFunc.c , myFunc.h
- ✓ insertBegin() : myFunc.c , myFunc.h
- ✓ main() : main.c
- ✓ print() : myFunc.c , myFunc.h
- ✓ printBody() : myFunc.h , myFunc.c
- ✓ printBrand() : myFunc.h , myFunc.c
- ✓ readEl() : myFunc.c , myFunc.h
- ✓ removeList() : myFunc.c , myFunc.h
- ✓ writeEl() : myFunc.c , myFunc.h

Съответно за повечето опции се правят в [main.c](#) функцията нужните за тях проверки дали е нужно да бъдат изпълнявани, ако нямаме записани в локалната памет данни от едносвързаният списък, при положителен резултат програмата се връща в изначалната точка за избор на Опция от текстовото меню, ако е отрицателен продължава изпълнението на записаните дейности в дадената опция.

Функции използвани във връзка със въвеждането на данни - Опция 1

int enterBody (BODY * ps)



Пълната дефиниция на функцията се намира на ред [107](#), в **myFunc.c**. Приема като параметър указател от тип [car Struct](#). Действието на функцията е първо да бъдат декларирани съответните локални променливи нужни за въвеждането на данни за елемента. Правим проверка дали предаваният указател коректно сочи към съответното място в паметта, тоест дали е заделено съответното адресно пространство за car Struct, и ако не връща **0**, съобщение за грешка и функцията приключва действието си. След това, заделя чрез функцията **memset** памет с големината на car Struct (за да имаме място за всички променливи дефинирани в car Struct, съответно ползвана като променлива в [List Struct](#) – осигурявайки частта с данни за динамична структура, едносвързан списък). Тук функцията вече пристъпва към съответното събиране на информация посредством печатане на съобщения на конзолата, искащи от потребителя да въведе данните в изрично упоменат формат. Осигурен е контрол за големината на извличаните данни, или посредством автоматично „отрязване“ на допълнително въведена информация, или изискващо повторното им въвеждане докато отговорят на видимо - конзолно упоменатият формат. При грешки от тип натискане на Enter без да сме въвели каквато/ и то и да е стойност/ и, или ръчно прекъсване EOF (чрез Ctrl + Z/

D) - печата се на екран съобщение за съответната причина за възникналата грешка и функцията приключва действието си. При успешно, коректно изпълнение за извличане на информация от потребителя, функцията връща флаг **1**, позволявайки ни да продължим нататък към следващата функция извиквана в [main.c](#) функцията – за да завършим процеса за въвеждането на данните за елемента.

LIST* insertBegin (LIST * pFirst, BODY newBody)



Пълната дефиниция на функцията се намира на ред [89](#), в **myFunc.c**. Приема като параметри указател сочещ началото на структура от тип свързан списък и променлива от тип **car Struct**. Действието на функцията е да резервира първо памет за участък в паметта с размера на **List Struct**, след това предава на локално-дефинираната променлива началото на този участък. Прави проверка дали наистина е заделена памет, и ако не, връща текстово съобщение за грешка, стойност **NULL** и в [main.c](#) се преустановява въвеждането на данни за елемента. При коректно заделена памет, декларира, че на отреденото място (за **List Struct**) **body (car Struct)**, ще се запази приеманата променлива **newBody** (запазва данните в новия елемент), а за отреденото място (за **List Struct**) на указател **pNext**, запазва предаваният **pFirst** – насочва елемента към началото. След това чрез локално дефиниран указател, приеманият **pFirst** установява началото в новия елемент. Функцията връща указател сочещ началото в даден нов елемент.

В [main.c](#) функцията, указателят **pFirst** от тип **LIST**, локално дефиниран за нея, приема стойността на връщаният от *** insertBegin** функцията, указател – и така всъщност при дефиниране на нова кола в едно и също изпълнение на програмата, ние отново ще имаме указателят **pFirst**, поставящ началото на елемент с данни от динамична структура – едносвързан списък.

Идеята при добавянето на нов елемент към едносвързаният списък е на стеков принцип, където краят на участъкът от резервираната памет, **p** (горе упоменат като локално дефиниран за дадената функция), бива предаван като адресна стойност от нея, като преди това **pNext**, бива запаменен с адреса на предхождащия го елемент. В този случай нуждата от глава - **head** на едносвързаният списък е премахната, началният от нас въведен елемент става един вид „последен“, и ние пазим и предаваме на всеки нов елемент само адреса на предхождащия го елемент – това по-нататък ще бъде демонстрирано при обхождането на такъв едносвързан списък почвайки от върхът и стигайки до дъното (началото).

Функция за запис на вече въведени данни (при едно и също изпълнение на програмата) в двоичен файл – Опция 2

Предварително в [main.c](#) функцията сме дефинирали името на файла в който ще записваме данните, както и нужните ни за целта променливи.

int writeEl (LIST * pL, FILE * pF)



Пълната дефиниция на функцията се намира на ред [58](#), в **myFunc.c**. Приема като параметри указател от тип **List Struct** и указател от тип **File Struct** (дефиниран в **stdio.h**). Действието на функцията е първо да провери за **NULL pointer** грешки, ако има такива връща флаг с цифра като грешка в [main.c](#), като там в функцията има съобщение което да се принтира на екрана

на потребителя заедно с флаг-а и приключва изпълнението на Опция 2. Ако всичко е коректно, взима да записва посредством **fwrite** функцията от началото (в нашия случай стек-овото начало) на свързаният списък, със стъпка размерът на BODY (**car Struct**), във файла – като това действие го изпълнява веднъж. Прави проверка дали върнатият флаг от **fwrite** е единица отговарящ на изпълнението само веднъж, ако не, връща флаг със съответната цифра и в **main.c** има съобщение което да се принтира на екрана на потребителя заедно с флаг-а и приключва изпълнението на Опция 2, ако да функцията **writeEl** връща флаг единица в **main.c**, където посредством дефиниран **for** цикъл обхождащ списъкът и извиква отново и отново **writeEl**, докато не се получи грешка или не е обходен списъкът. В **main.c**, е осигурено списъкът да бъде изтрит, тъй като го имаме вече записан във файла – това става посредством функцията **removeList**, която ще бъде описано по-долу.

Функция за четене на данни от двоичен файл – Опция 3

При нея данните биват заредени в локалната памет и са достъпни за бъдещи действия от програмата с тях.

В **main.c** функцията е осигурено дефинирането на нужните ни променливи и отварянето на вече дефинираният като име, и стойности от нас файл. Там са дефинирани и опции за улавяне на грешки които да биват извиквани при NULL pointer грешки или при връщани стойности, дефинирани за грешки, от съответните функции, както следват по-долу – осигурено е да има съобщение което да се принтира на екрана на потребителя заедно с флаг-а, а след това да приключва изпълнението на Опция 3.

Самото четене се осъществява от функцията:

int readEl (BODY * pB, FILE * pF)



Пълната дефиниция на функцията се намира на ред [73](#), в **myFunc.c**.

Приема като параметри указател от тип **car Struct** и указател от тип **File Struct** (дефиниран в **stdio.h**). Действието на функцията е първо да провери за NULL pointer грешки, ако има такива връща флаг с цифра като грешка в **main.c**. Ако всичко е коректно, започва да чете посредством **fread** функцията от файла, със стъпка размерът на BODY (**car Struct**) и връща прочетеното чрез **pB** – като това действие го изпълнява веднъж. Прави проверка дали върнатият флаг от **fread** е единица отговарящ на изпълнението само веднъж, ако не, връща флаг със съответната цифра и в **main.c**, ако да, функцията **writeEl** връща флаг единица в **main.c**, където продължаваме истинското зареждане в локалната памет посредством по-горе описаната * **insertBegin** функция – ако всичко е наред и няма NULL pointer грешки или друг тип грешки които да карат програмата да приключи изпълнението на Опция 3, досега описаните в този абзац, се повтарят докато връщаният флаг от **readEl** е цифрата 1 (тоест има данни за четене от файла) – съответно проверката се осъществява в **main.c** функцията, при различна от едно стойност, се приключва изпълнението на Опция 3.

Функциите извършващи печатането на екрана на конзолата всички

заредени в локалната памет данни от свързаният списък – Опция 4

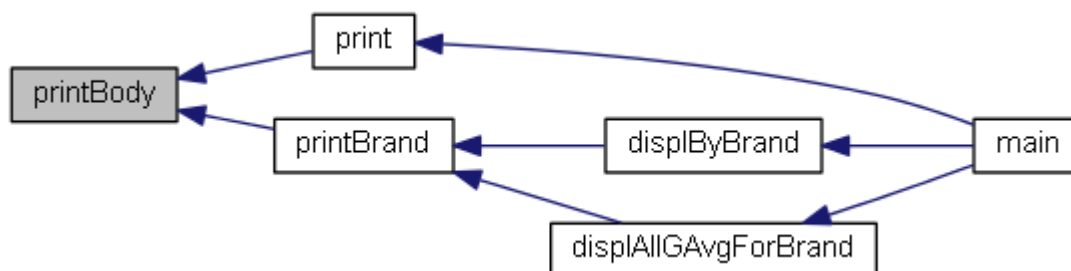
void print (LIST * pFirst)



Пълната дефиниция на функцията се намира на ред [182](#), в **myFunc.c**.

Приема като параметър указател от тип [List Struct](#). Действието на функцията е първо да дефинира локална променлива указател от тип [List Struct](#), който да приема предаваният параметър от същият тип, и посредством **while** цикъл (който не приключва изпълнението на Опция 4 докато не е достигнат края на свързаният списък и локално дефинираната променлива от тип `int` не е 0 – зависи от резултата на функцията **printBody**) да извиква отново и отново функцията **printBody**, като след това променя стойността на локално дефинираният указател на следващата стойност (ако има такава) записана в `pNext` променливата, в [List Struct](#) на даденият елемент.

int printBody (BODY s)



Фигура 2. Графика на Извикванията на Функцията **printBody**

Пълната дефиниция на функцията се намира на ред [201](#), в **myFunc.c**. Приема като параметър указател от тип `car Struct`. Действието на функцията е посредством форматиран за специфичните данни **printf** функция да принтира на екрана на конзолата записаните данни в `car Struct`, като ги достъпва една по една. Запазва резултата от **printf** функцията в локално дефинирана променлива от тип `int`, като при неуспешно изпълнение на операцията връща флаг **0**, а при успешно флаг **1** (съответно води до изпълнение на самата функция още веднъж).

Функция за изтриване на елемент от свързаният списък по посочена данна

– Опция 5

void delByCarID (LIST ** pFirst, char * carID)



Пълната дефиниция на функцията се намира на ред [8](#), в **myFunc.c**. Приема като параметър двоен указател от тип [List Struct](#) и указател от тип `char`. Действието на функцията е първоначално да дефинира локален двоен указател от тип [List Struct](#) и друг единичен от същият тип. Целта е, посредством двойният указател приемащ стойността на другият такъв предаван като параметър да обхождаме в един **for** цикъл (със съответните условия) свързаният списък, пазейки едно назад (адреса на елемента преди сегашния), а пък посредством другият единичен, да достъпваме адреса на следващият елемент. В този **for** цикъл, правим сравнение/ проверка на получената като параметър променлива от тип `char`, със същото по тип и конкретно интересуващо ни данново поле в `car Struct` (където се палят данните за елемента), ако има съвпадение освобождаваме паметта заета от този елемент (извършваме триене) и пренасочваме указателят на предхождащият го елемент да сочи този след изтритият – приключва се изпълнението на Опция 5. Правим друго едно сравнение/ проверка в този **for** цикъл, логически поставено след първото такова, дали указателят за следващият елемент не е `NULL`, ако да - приключва се изпълнението на Опция 5 със съответно съобщение за потребителя принтирано на конзолата.

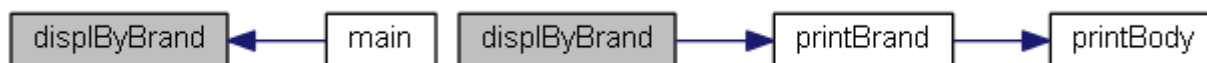
В [main.c](#) функцията където получаваме от потребителя `char * carID` посредством конзолата. Има дефинирани действия които да се извършват при грешки от тип натискане на Enter без

да сме въвели каквато/ ито и да е стойност/ и, или ръчно прекъсване EOF (чрез Ctrl + Z/ D) - печата се на екран съобщение за съответната причина за възникналата грешка и посочената опция приключва действието си. Има осигурен контрол на дължината на въвежданите данни, ако неотговарят, повтаря се искането към потребителя за въвеждане на данни докато не е коректно или при грешка.

Функции осигоряващи търсенето и принтирането на елементи от свързаният списък по посочена данна – Опция 6

В [main.c](#) функцията получаваме от потребителя исканата данна посредством конзолата. Има дефинирани действия които да се извършват при грешки от тип натискане на Enter без да сме въвели каквато/ ито и да е стойност/ и, или ръчно прекъсване EOF (чрез Ctrl + Z/ D) - печата се на екран съобщение за съответната причина за възникналата грешка и посочената опция приключва действието си. Има осигурен контрол на дължината на въвежданите данни, ако неотговарят, повтаря се искането към потребителя за въвеждане на данни докато не е коректно или при грешка.

void displByBrand (char * carBrand, LIST ** pFirst)



Пълната дефиниция на функцията се намира на ред [210](#), в **myFunc.c**. Приема като параметри указател от тип `char` и двоен указател от тип [List Struct](#). Действието на функцията е идентично с това при **delByCarID**, само където тук имаме за първото сравнение/ проверка, за всяко такова съвпадение се извиква функцията **PrintBrand**, но при върнат от **int printBrand** флаг 0 - приключва се изпълнението на Опция 6. При другото сравнение/ проверка където, ако указателят за следващият елемент е NULL, ако да - приключва се изпълнението на Опция 6 със съответно съобщение за потребителя принтирано на конзолата.

int printBrand (LIST * pFirst)



Пълната дефиниция на функцията се намира на ред [230](#), в **myFunc.c**.

Приема като параметър указател от тип [List Struct](#). Действието на функцията е идентично с това на вече описаната функция **void print** ([За Опция 4](#)), с тези две разлики:

Първо нямаме **while** цикъл който да извиква отново и отново **printBody**, многократното и извикване зависи от **displByBrand** функцията, но тук отново следим за грешка в **printf** функцията.

Опция 6 приключва при положителен резултат от второто съвпадение за **void displByBrand**, съответно ако указателят за следващият елемент е NULL - със съответно съобщение за потребителя принтирано на конзолата.

Функции осигуряващи търсенето и принтирането на елементи от едносвързаният списък по посочена данна и ако отговарят на зададено условие – Опция 7

В случаят става дума за принтиране на елементи от свързаният списък, коли, по посочена от потребителя марка, но само ако цената им е по-голяма от средната за марката. В [main.c](#) функцията получаваме от потребителя исканата данна посредством конзолата. Има дефинирани действия които да се извършват при грешки от тип натискане на Enter без да сме въвели каквато/ и то да е стойност/ и, или ръчно прекъсване EOF (чрез Ctrl + Z/ D) - печата се на екран съобщение за съответната причина за възникналата грешка и посочената опция приключва действието си. Има осигурен контрол на дължината на въвежданите данни, ако неотговарят, повтаря се искането към потребителя за въвеждане на данни докато не е коректно или при грешка. Друга специфична проверка, преди да извикаме функция за да покажем тези коли, е да видим дали търсената марка коли въобще се намира в едносвързаният списък – ако не е, приключва се изпълнението на Опция 7. Ако вече сме получили исканата данна, няма грешки и действията в тази опция продължават, искаме да получим средната цена за дадената марка, ползвайки функцията:

double brandAvg (LIST ** pFirst, char * carBrand)

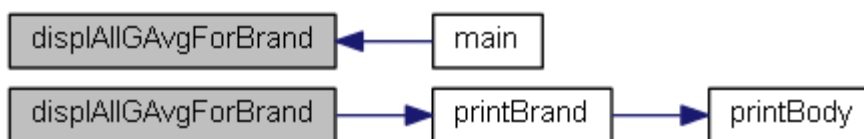


Пълната дефиниция на функцията се намира на ред [260](#), в **myFunc.c**.

Приема като параметри указател от тип `char` и двоен указател от тип [List Struct](#).

Действието на функцията е сходно с това на **delByCarID**, но тук използваме двете сравнение/ проверка за да изчислим средната цена за автомобил от дадената марка. При първото изчисляваме общата сума като събрана единична цена на всички и броят коли, като това става посредством обхождане на списъкът и сравнение по марка, **carBrand**. Изходът от тази функция е при второто сравнение/ проверка, където ако указателят за следващият елемент е NULL, извършваме аритметични действия за намиране на средната цена за марката и връщаме резултатът в [main.c](#) функцията – съответно той бива обработен от следната функция:

void displAllGAvgForBrand (LIST ** pFirst, char * carBrand, double average)



Пълната дефиниция на функцията се намира на ред [239](#), в **myFunc.c**. Приема като параметри указател от тип `char`, двоен указател от тип [List Struct](#) и променлива от тип `double`.

Действието на функцията е сходно с това на **delByCarID**, но тук използваме двете сравнение/ проверка за да принтираме на екрана съответните елементи. При първото извикваме функцията **printBrand** (но при върнат от **int printBrand** флаг 0 - приключва се изпълнението на Опция 7), тя от своя страна за да изпълни истинското визуализиране извиква друга също вече описана функция **printBody** – като до тук казаното става посредством обхождане на списъкът и сравнение по марка, **carBrand**. При коректно изпълнение, изходът от тази функция и Опция 7 е при второто сравнение/ проверка, където ако указателят за следващият елемент е NULL


```

24 break;
25 }
26 }
27 }
28 /* It releases the memory reserved by the list. It gets the beginning of the
   list
29 via a pointer to the beginning of the list pFirst. */
30 void removeList(LIST **pFirst)
31 {
32     BODY first;
33     while(*pFirst != NULL)
34         *pFirst = delFirst(*pFirst, &first);
35 }
36 //=====
37 /* Deletes an element at the beginning of the list. Returns the data of deleted
   element
38 via the delBody pointer and a pointer to the beginning of the list or NULL if
   unsuccessful */
39 LIST *delFirst(LIST *pFirst, BODY *delBody)
40 { if (pFirst == NULL)
41   { printf("Empty list!\n");
42     return NULL;
43   }
44   else
45   { LIST *p; // temp link
46     *delBody = pFirst->body; // saves the data
47     p = pFirst->pNext; // it points the link to the second element
48     if(p != NULL)
49         free (pFirst); // releases the allocated mem
50     pFirst = p; // it points the beginning to the link
51     return pFirst;
52   }
53 }
54 //=====
55 /* Saves the data of an element from a list that the pL pointer points to, into
   a binary file
56 with a pointer pF. Returns a flag: 1 (succesful save),
57 -1 (zero pointer to the data), -2(zero pointer to the file), -3 (Error upon a
   save into the file). */
58 int writeEl(LIST *pL, FILE *pF)
59 {
60     int res;
61     if (pL == NULL) return -1;
62     if (pF == NULL) return -2;
63     res = fwrite(&(pL->body), sizeof(BODY), 1, pF);
64     if (res == 1)
65         return 1; // Success
66     else
67         return -3; // Error upon a save into the file
68 }
69 //=====
70 /*Reads the data from a binary file with a pF pointer. Returns data via the
   pointer pB and flag: 1 (successful reading),
71 -1 (zero pointer to the data), -2 (zero pointer to the file), -3 (Error upon
   reading from the file),
72 -4 (end of File has been reached). */
73 int readEl(BODY *pB, FILE *pF)
74 {
75     int res;
76     if (pB == NULL) return -1;
77     if (pF == NULL) return -2;
78     res = fread(pB, sizeof(BODY), 1, pF);
79     if (res == 1) return 1; // Success
80     else
81     {

```

```

82  if (feof(pF) ) return -4; // End of File has been reached
83  return -3; // Error upon reading from the file
84  }
85  }
86  //=====
87  /* Includes element with data newBody at the beginning of pFirst in the list.
88  Returns a pointer of the list's beginning or NULL upon an unsuccessful operation
89  */
89  LIST *insertBegin(LIST *pFirst, BODY newBody)
90  {
91      LIST *p;
92      p = (LIST *)malloc(sizeof(LIST)); // saves a mem location
93      if (p == NULL)
94      { printf("\nError! There is not enough memory!\n");
95        return NULL;
96      }
97      else
98      { p->body = newBody; // saves the data into the new element
99        p->pNext = pFirst; // points the element at the beginning
100        pFirst = p; // sets the beginning at the new element
101        return p;
102      }
103  }
104  //=====
105  /* The entering function for cars. It does the car input into a list
106  via a pointer ps and flag: 0 (error upon data entry), 1 (correct data entry). */
107  int enterBody(BODY *ps)
108  {
109      LIST *pFirst = NULL;
110      char carIDnum [MIN_LENGTH+1];
111      int res, c, t;
112      if(ps == NULL)
113          return 0;
114      memset(ps, 0, sizeof(BODY));
115
116      do
117      {
118          fflush(stdin);
119          printf("\nCar ID |It is an alphameric sequence of 10 symbols|: ");
120          if(gets(ps->carID) == NULL)
121              {/*the Ctrl/Z combination
122               */
123              return 0;
124              }
125          if(ps->carID[0] == '\0')
126              {/*if nothing has been entered
127               */
128              printf("\nError, nothing has been entered, try again!!\n");
129              return 0;
130              }
131          //if(pFirst != NULL){ if( alreadyIn(&pFirst, carIDnum) ){ printf("\nThere is
132          already a car with ID:%s in the CAR CATALOGUE!\n", carIDnum); break;} }
133          while( strlen(ps->carID) != MIN_LENGTH );
134
135          fflush(stdin);
136          printf("\nCar color |Character set of up to 20 symbols|: ");
137          if(gets(ps->color) == NULL)
138              return 0; // CTRL/Z has been pressed
139          if(ps->color[0] == '\0')
140              {/*if nothing has been entered
141               */
142              printf("\nError, nothing has been entered, try again\n");
143              return 0;
144              }
145          if(strlen(ps->color) > MAX_LENGTH)
146              ps->color[MAX_LENGTH] = '\0';
147      }
148      do

```

```

145 {
146     fflush(stdin);
147     printf("\nCar Engine Size |A real number up to 100|: ");
148     c = getchar();
149     if ( c == '\n' ){ ungetc(c, stdin); printf("\nError, nothing has been entered,
try again!!\n"); return 0;}
150     if ( c == EOF ){ ungetc(c, stdin); return 0;}
151     ungetc(c, stdin);
152     scanf("%f", &(ps->carEngineSize));
153     while(ps->carEngineSize < 1 || ps->carEngineSize > 100);
154
155     fflush(stdin);
156     printf("\nCar Brand |Character set of up to 20 symbols|: ");
157     if(gets(ps->carBrand) == NULL)
158         return 0; // CTRL/Z has been pressed
159     if(ps->carBrand[0] == '\0')
160         { //if nothing has been entered
161             printf("\nError, nothing has been entered, try again!\n");
162             return 0;
163         }
164     if(strlen(ps->carBrand) > MAX_LENGTH)
165         ps->carBrand[MAX_LENGTH] = '\0';
166
167     do
168     {
169         fflush(stdin);
170         printf("\nCar Price |A real number up to 10,000,000|: ");
171         c = getchar();
172         if ( c == '\n' ){ ungetc(c, stdin); printf("\nError, nothing has been entered,
try again!\n"); return 0;}
173         if ( c == EOF ){ ungetc(c, stdin); return 0;}
174         ungetc(c, stdin);
175         scanf("%lf", &(ps->carPrice));
176     }
177     while(ps->carPrice < 1 || ps->carPrice >10000000);
178     return 1;
179 }
180 //=====
181 /* Prints out the elements of list that has a pointer start pFirst. */
182 void print(LIST *pFirst)
183 {
184     int res;
185
186
187
188     LIST *p;
189     p=pFirst;
190     while(p!=NULL || res == 0)
191     {
192         res = printBody(p->body);
193         p=p->pNext;
194     }
195     printf("\n");
196
197 }
198 //=====
199 /* Displays the Car Data on the Screen.
200 Returns a flag: 0 (error upon print), 1 (successful operation). */
201 int printBody(BODY s)
202 {
203     int res;
204     fflush(stdin);
205     res = printf("%-11s%-21s%-12.3f%-21s%-8.5f\n", s.carID, s.color,
s.carEngineSize, s.carBrand, s.carPrice);
206     if(res < 0) return 0;

```

```

207 else return 1;
208 }
209 //=====
210 void displByBrand(char *carBrand, LIST **pFirst)
211 {
212     LIST **current;
213     LIST *pNext; int t;
214     printf("\nCarID: Color: EngineSize: Brand: Price:\n");
215     for (current = pFirst, pNext = (*current)->pNext; *current; current =
        &(*current)->pNext, pNext = (*current)->pNext)
216     {
217         if(strcmp((*current)->body.carBrand, carBrand) == 0)
218         {
219             t = printBrand(*current); if (t == 0) {return;}
220         }
221         if(pNext == NULL)
222         {
223             printf("\n\nEither the car brand you have entered is NOT in the Car Catalogue
                or the above\nis ALL of them!\n\n");
224             return;
225         }
226     }
227     return;
228 }
229 //=====
230 int printBrand(LIST *pFirst)
231 {
232     LIST *t;
233     int res;
234     t=pFirst;
235     res = printBody((t)->body);
236     t = (t)->pNext; if (res == 0){return 0;}
237 }
238 //=====
239 void displAllGAvgForBrand(LIST **pFirst, char *carBrand, double average)
240 {
241     LIST **current;
242     LIST *pNext; int t;
243     printf("\nCarID: Color: EngineSize: Brand: Price:\n");
244     for (current = pFirst, pNext = (*current)->pNext; *current; current =
        &(*current)->pNext, pNext = (*current)->pNext)
245     {
246         if( ( strcmp((*current)->body.carBrand, carBrand) ) == 0)
247         {
248             if((*current)->body.carPrice > average)
249             {
250                 t = printBrand(*current); if(t == 0) {return;}
251             }
252         }
253         if(pNext == NULL)
254         {
255             return;
256         }
257     }
258 }
259 //=====
260 double brandAvg(LIST **pFirst, char *carBrand)
261 {
262     LIST **current;
263     LIST *pNext;
264     int count = 0;
265     double sum = 0, avg = 0;
266     for (current = pFirst, pNext = (*current)->pNext; *current; current =
        &(*current)->pNext, pNext = (*current)->pNext)
267     {

```

```

268 if( ( strcmp( (*current)->body.carBrand, carBrand ) ) == 0)
269 {
270 ++count;
271 sum += ((*current)->body.carPrice);
272 }
273 if(pNext == NULL)
274 {
275 avg = sum/(double)count;
276 return avg;
277 }
278 }
279 }

```

myFunc.h

```

1 #include <stdio.h>
2 #include <malloc.h> //modify the behavior of malloc, realloc, and free by
   specifying appropriate hook functions
3 #include "windows.h"
4 #define LOOPS 1
5 #define MAX_LENGTH 20
6 #define MIN_LENGTH 10
7 struct car
8 {
9 char carID[MIN_LENGTH+1]; //10 symbols
10 char color[MAX_LENGTH+1]; //20 symbols
11 float carEngineSize;
12 char carBrand[MAX_LENGTH+1];
13 double carPrice;
14 };
15 typedef struct car BODY;

16 struct List
17 {
18 BODY body;
19 struct List *pNext;
20 };
21 typedef struct List LIST;
22
23 void delByCarID(LIST **pFirst, char *carID);
24 void removeList(LIST **pFirst);
25 LIST *delFirst(LIST *pFirst, BODY *delBody);
26 int writeEl( LIST *pL, FILE *pF);
27 int readEl( BODY *pB, FILE *pF);
28 LIST *insertBegin(LIST *pFirst, BODY newBody);
29 int enterBody(BODY *ps);
30 void print(LIST *pFirst);
31 int printBody(BODY s);
32 void displByBrand(char *carBrand, LIST **pFirst);
33 int printBrand(LIST *pFirst);
34 void displAllGAvgForBrand(LIST **pFirst, char *carBrand, double average);
35 double brandAvg(LIST **pFirst, char *carBrand);

```

main.c

```

1 #include "myFunc.h"
2 #include <stdlib.h>
3 #include "windows.h"
4 int main()
5 {
6 LIST *pFirst = NULL, *p;
7 int res, i, mode, c, t;
8 double average = 0;

```



```

9 FILE *pOut = NULL, *pIn = NULL;
10 char FName[]="Car_List_bin.dat";
11 BODY car;
12 char carIDnum [MIN_LENGTH+1], carBran [MAX_LENGTH+1];
13 char *menu[] = {"USED CAR DEALERSHIP CATALOGUE\n\tSUPPORT DATA MENU",
14 "1-Enter data for a new car",
15 "2-Write the data into a binary file",
16 "3-Read the data from a binary file",
17 "4-Display all available cars",
18 "5-Delete a car by a car identification number",
19 "6-Search and Display available cars by a specified brand",
20 "7-Display all cars from a specific brand that have a price\nlarger than the
    average for the brand",
21 "8-Destroy the car data and Exit"};
22 do
23 { system("cls");
24 for(i=0; i < 9; i++)
25 printf("\n%s\n",menu[i]);
26 do
27 {
28 fflush(stdin);
29 printf ("\n\nChoose mode[1-8]: ");
30 res = scanf("%d", &mode);
31 }while(res !=1);
32 switch(mode)
33 {
34 case 1: //data entry from the keyboard
35 for(i = 0; i < LOOPS; i++)
36 {
37 res = enterBody(&car);
38 if (res != 1 ) //the function returns 0 or 1
39 {
40 printf("\nError in initialization %d!\n", res);
41 break;
42 }
43 p = insertBegin(pFirst, car);
44 if(p == NULL)
45 {
46 printf("\n\nNot enough memory!\n");
47 break;
48 }
49 pFirst = p;
50 }
51 system("pause");
52 break;
53 case 2: // opening the file and writing on it the list
54 {
55 if (pFirst == NULL)
56 { printf("\nThe CAR CATALOGUE is Empty, there are no car records to be
    saved!\n");system("pause");break;}
57 pOut = fopen(FName, "wb");
58 if(pOut == NULL)
59 {
60 printf("\nCan't open file for writing!\n");
61 break;
62 }
63 for(p = pFirst; p != NULL ; p = p->pNext)
64 {
65 res = writeEl(p, pOut);
66 if(res != 1)
67 {
68 printf("\nWriting error %d !\n\n", res);
69 break;
70 }
71 }

```

```

72  fclose(pOut);
73  removeList(&pFirst);
74  printf("\nThe Catalogue data has been recorded on the Car_List_bin.dat
file\n");
75  }
76  system("pause");
77  break;
78  case 3: // opening the file and reading the list
79  {
80  pIn = fopen(Fname, "rb");
81  if( pIn == NULL)
82  {
83  printf("\nCan't open file for reading!\n");
84  break;
85  }
86  do
87  {
88  res = readEl(&car, pIn);
89  if (res != 1 && res != -4 )
90  {
91  printf("\nReading error %d !\n", res);
92  break;
93  }
94  if (res != -4)
95  { p = insertBegin(pFirst, car);
96  if ( p == NULL )
97  { printf("\nNot enough memory!\n");
98  break;
99  }
100  pFirst = p;
101  }
102  }while(res == 1);
103  fclose(pIn);
104  printf("\nThe file has been read and the catalogue is available for Display via
Option 4.\n");
105  }
106  system("pause");
107  break;
108  case 4: // Displaying all available cars on the screen
109  if (pFirst!=NULL)
110  {
111  printf("\nCarID: Color: EngineSize: Brand: Price:\n");
112  print(pFirst);
113  }
114  else
115  printf("\nThe CAR CATALOGUE is Empty, there are no cars to be Displayed!\n");
116  system("pause");
117  break;
118  case 5:
119  if (pFirst == NULL)
120  { printf("\nThe CAR CATALOGUE is Empty, there are no cars to be
deleted!!\n");system("pause");break;}
121  do{
122  fflush(stdin);
123  printf("\nDelete a car by entering carID|It's an alphameric sequence of upto 10
symbols|: ");
124  if(gets(carIDnum) == NULL)//the Ctrl/Z combination
125  {
126  printf("Error in initialization!\n");
127  break;
128  }
129  if(carIDnum[0] == '\0')//if nothing has been entered
130  {
131  printf("\nError, nothing has been entered, try again!\n");
132  break;

```

```

133 }
134 else
135 {
136     delByCarID(&pFirst, carIDnum);
137 }
138
139 }while( strlen(carIDnum) != MIN_LENGTH );
140 system("pause");
141 break;
142 case 6:
143     if (pFirst == NULL)
144     { printf("\nThe CAR CATALOGUE is Empty, there are no cars to be searched/
displayed!\n");system("pause");break;}
145     do{
146         fflush(stdin);
147         printf("\nSearch and Display all cars by a specified by you car brand
|Character set of up to 20 symbols|: ");
148         if(gets(carBran) == NULL)//the Ctrl/Z combination
149         {
150             printf("\nError in initialization!\n");
151             break;
152         }
153         if(carBran[0] == '\0')//if nothing has been entered
154         {
155             printf("\nError, nothing has been entered, try again!\n");
156             break;
157         }
158         else
159         {
160             displByBrand(carBran, &pFirst);
161         }
162     }while( !( strlen(carBran) < MAX_LENGTH && strlen(carBran) > 0 ) );
163     system("pause");
164     break;
165 case 7:
166     if (pFirst == NULL)
167     { printf("\nThe CAR CATALOGUE is Empty, there are no cars to be searched/
displayed!\n");system("pause");break;}
168     do{
169         fflush(stdin);
170         printf("\nEnter a car brand |Character set of up to 20 symbols|: ");
171         if(gets(carBran) == NULL)//the Ctrl/Z combination
172         {
173             printf("\nError in initialization!\n");
174             break;
175         }
176         if(carBran[0] == '\0')//if nothing has been entered
177         {
178             printf("\nError, nothing has been entered, try again!\n");
179             break;
180         }
181         else
182         {
183             average = brandAvg(&pFirst, carBran);
184             if(average != average){ printf("\n\nThe car brand you have entered is NOT in
the CAR CATALOGUE!\n\n"); break;}
185             printf("\nThe Average Car Price for brand %s is: %f\n",carBran ,average);
186             displAllGAvgForBrand(&pFirst, carBran, average);
187         }
188     }while( !( strlen(carBran) < MAX_LENGTH && strlen(carBran) > 0 ) );
189     system("pause");
190     break;
191 case 8:
192     if (pFirst!=NULL)
193     removeList(&pFirst);

```

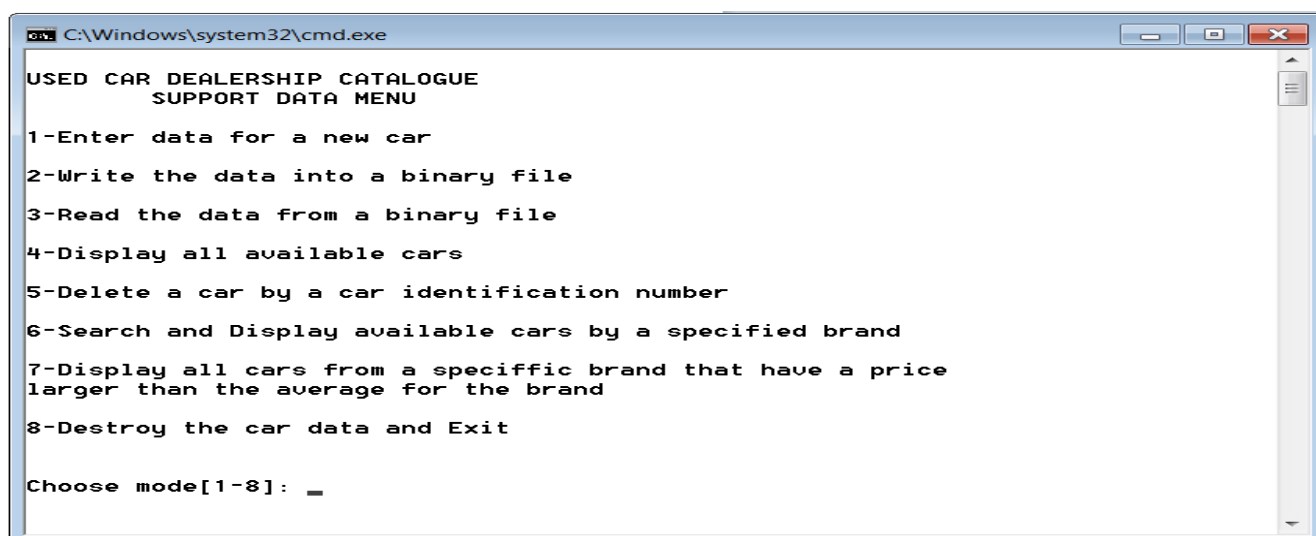
```

194 printf("\nThe CAR LIST is Empty!\nProgram exiting...\n");
195 break;
196 default:
197 printf("\nBad choice! \n");
198 system("pause");
199 }
200 }while(mode != 8); // eight because the list gets destroyed and we exit the
program
201 return 0;
202 }

```

Контролен Пример от изпълнението на програмата

Началното меню



```

C:\Windows\system32\cmd.exe

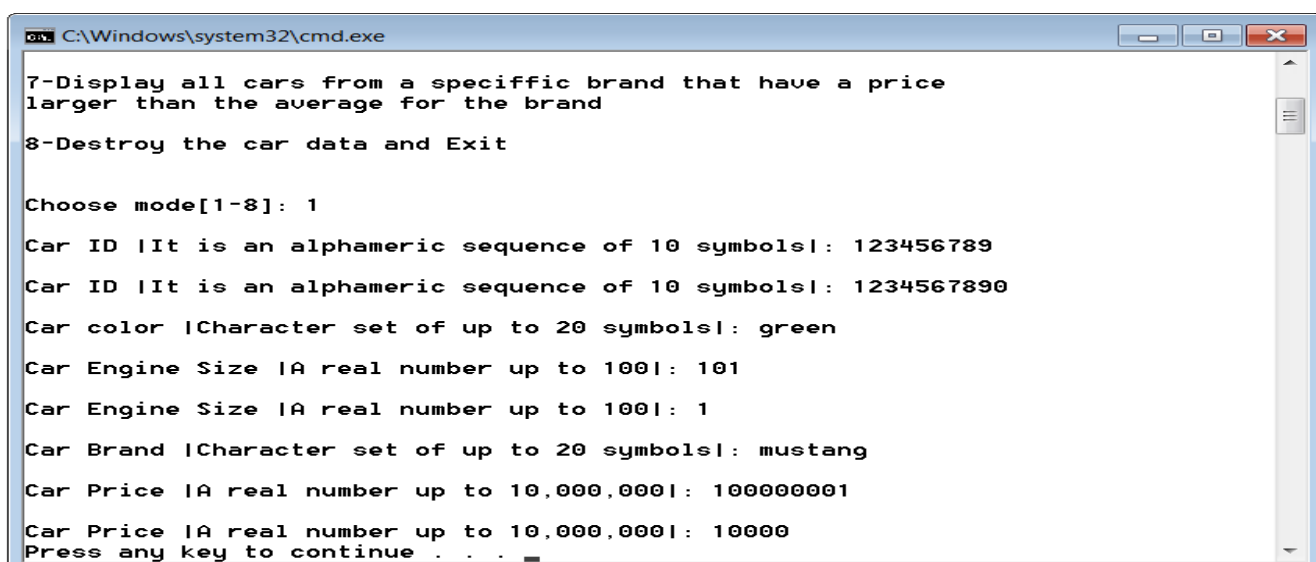
USED CAR DEALERSHIP CATALOGUE
SUPPORT DATA MENU

1-Enter data for a new car
2-Write the data into a binary file
3-Read the data from a binary file
4-Display all available cars
5-Delete a car by a car identification number
6-Search and Display available cars by a specified brand
7-Display all cars from a specific brand that have a price
larger than the average for the brand
8-Destroy the car data and Exit

Choose mode[1-8]: _

```

Опция 1



```

C:\Windows\system32\cmd.exe

7-Display all cars from a specific brand that have a price
larger than the average for the brand
8-Destroy the car data and Exit

Choose mode[1-8]: 1
Car ID |It is an alphaneric sequence of 10 symbols|: 123456789
Car ID |It is an alphaneric sequence of 10 symbols|: 1234567890
Car color |Character set of up to 20 symbols|: green
Car Engine Size |A real number up to 100|: 101
Car Engine Size |A real number up to 100|: 1
Car Brand |Character set of up to 20 symbols|: mustang
Car Price |A real number up to 10,000,000|: 100000001
Car Price |A real number up to 10,000,000|: 10000
Press any key to continue . . . _

```

Опция 2

```
Choose mode[1-8]: 2
```

```
The Catalogue data has been recorded on the Car_List_bin.dat file
```

```
Press any key to continue . . .
```

Опция 3

```
Choose mode[1-8]: 3
```

```
The file has been read and the catalogue is available for Display via Option 4.
```

```
Press any key to continue . . .
```

Опция 4

```
Choose mode[1-8]: 4
```

CarID:	Color:	EngineSize:	Brand:	Price:
1234567890	red	10.000	mustang	10000.00000
123456789A	blue	1.400	volvo	12000.00000
123456789B	black	1.800	volvo	12000.00000
123456789C	pueple	13.000	mustang	5000.00000
1234567890	red	10.000	mustang	10000.00000
123456789A	blue	1.400	volvo	12000.00000
123456789B	black	1.800	volvo	12000.00000

```
Press any key to continue . . .
```

Опция 5

```
Choose mode[1-8]: 5
```

```
Delete a car by entering carID|It's an alphameric sequence of upto 10 symbols|:  
1234567890
```

```
The car with ID:1234567890 has been deleted!
```

```
Press any key to continue . . .
```

Результат от избор на Опция 5

```
Choose mode[1-8]: 4
```

CarID:	Color:	EngineSize:	Brand:	Price:
123456789A	blue	1.400	volvo	12000.00000
123456789B	black	1.800	volvo	12000.00000
123456789C	pueple	13.000	mustang	5000.00000
1234567890	red	10.000	mustang	10000.00000
123456789A	blue	1.400	volvo	12000.00000
123456789B	black	1.800	volvo	12000.00000

```
Press any key to continue . . .
```

Опция 6

```
Choose mode[1-8]: 6
Search and Display all cars by a specified by you car brand |Character set of up
to 20 symbols|: mustang

CarID:      Color:      EngineSize: Brand:      Price:
123456789C pueple      13.000      mustang      5000.00000
1234567890 red         10.000      mustang      10000.00000

Either the car brand you have entered is NOT in the Car Catalogue or the above
is ALL of them!

Press any key to continue . . .
```

Опция 7

```
Choose mode[1-8]: 7

Enter a car brand |Character set of up to 20 symbols|: mustang

The Average Car Price for brand mustang is: 7500.000000

CarID:      Color:      EngineSize: Brand:      Price:
1234567890 red         10.000      mustang      10000.00000

Press any key to continue . . .
```

Опция 8

```
Choose mode[1-8]: 8

The CAR LIST is Empty!
Program exiting...
Press any key to continue . . .
```