ТЕХНИЧЕСКИ УНИВЕРСИТЕТ–СОФИЯ

ФАКУЛТЕТ "КОМПЮТЪРНИ СИСТЕМИ И УПРАВЛЕНИЕ"
Катедра „Програмиране и Компютърни Технологии"

Курсова Работа по

Алгоритмизация и Основи на Програмирането

Програма за поддръжка на студентски резултати от изпитна сесия

Изготвил Задачата:
Стефан Байчев
Email: sbaychev@gmail.com
ФКСУ - КТПП
Факултетен Номер 281611007
Магистър - изравнително обучение
I курс II семестър

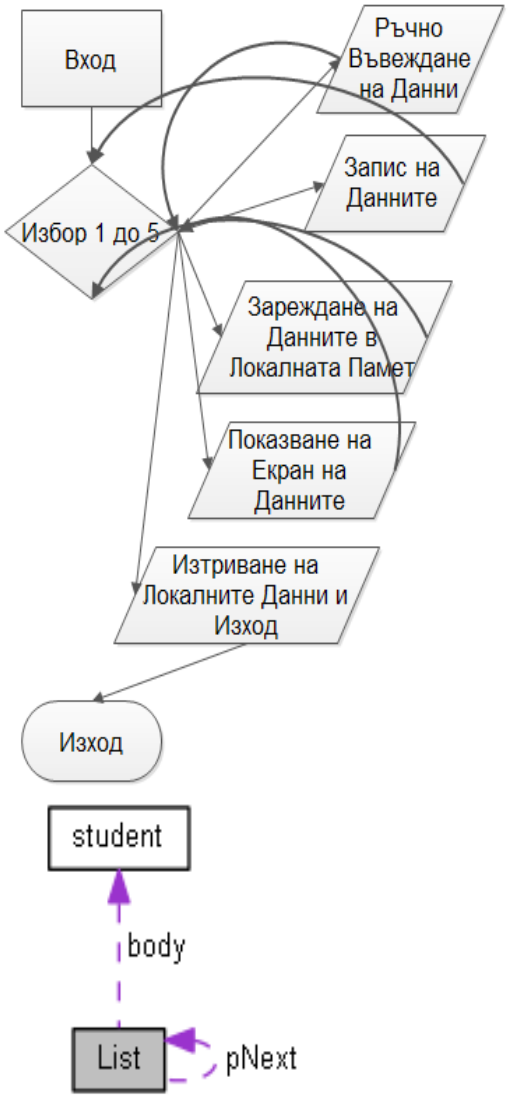
Преподавател:……
/доц. д-р Мариана Горанова/

София, 14.06.2012 г.

Съдържание

Обобщен Блоков Алгоритъм

При изпълнение на която и да е от възможните за избор Опции (от 1 до 5), при коректно или некоректно такова, програмата се връща в изначалната точка за избор на Опция от текстовото меню – това изключва Опция 5 разбира се, защото чрез нея се осъществява изходът от програмата.



фиг.1 **Диаграма на взаимовръзката за List**

Описание на използваните Функции

student Struct Референция

typedef struct student BODY

Полета за Данни:
int facNum
char firstName [NAME_LENGTH+1]
char secondName [NAME_LENGTH+1]
char thirdName [NAME_LENGTH+1]
int groupNum
double markOne
double markTwo
double markThree
double markFour
double avgGrade

където NAME_LENGTH = 12

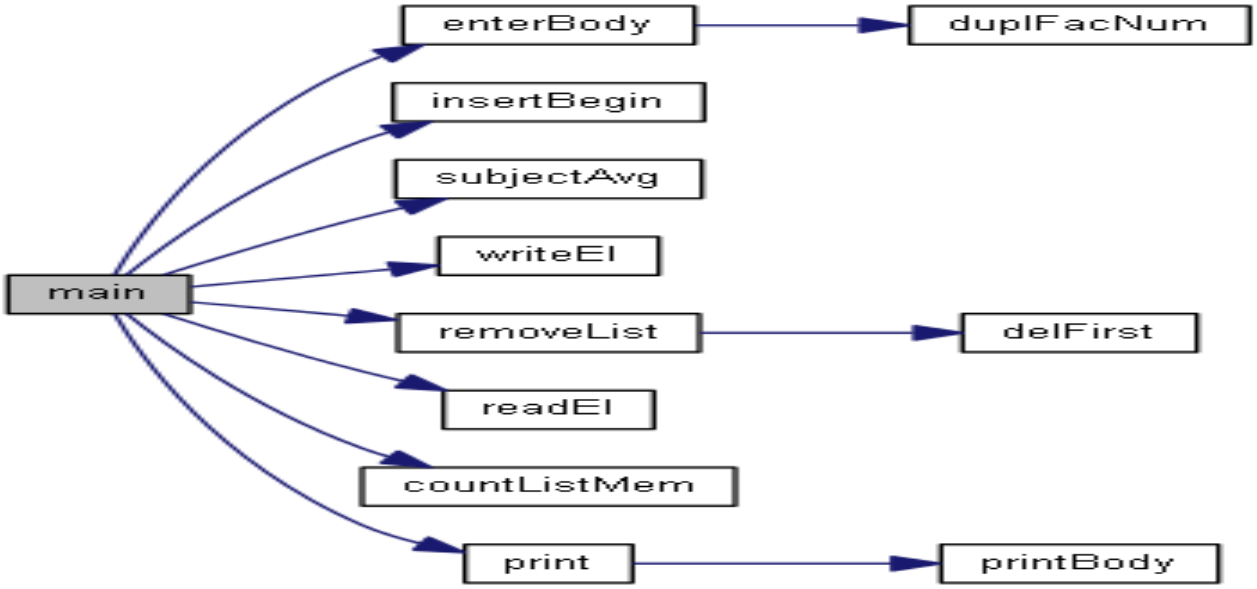
List Struct Референция

typedef struct List LIST

Полета за Данни:
BODY body
struct List * pNext

Общ вид на използваните функции за реализиране на поставената задача:

- ✓ countListMem() : cFunc.c , hFunc.h
- ✓ delFirst() : cFunc.c , hFunc.h
- ✓ duplFacNum() : cFunc.c , hFunc.h
- ✓ enterBody() : cFunc.c , hFunc.h
- ✓ insertBegin() : cFunc.c , hFunc.h
- ✓ main() : mainF.c
- ✓ print() : cFunc.c , hFunc.h
- ✓ printBody() : cFunc.c , hFunc.h
- ✓ readEl() : cFunc.c , hFunc.h
- ✓ removeList() : cFunc.c , hFunc.h
- ✓ subjectAvg() : cFunc.c , hFunc.h
- ✓ writeEl() : cFunc.c , hFunc.h



За съответните Опции в **mainF.c** функцията се правят нужните за тях проверки дали трябва 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. Има осъществени няколко проверки посредством различни флагове за дадени функции и Опции като цяло, а по специфично, колко пъти са били вече извикани и от къде, след това може да има и извикване на други функции които да извършат определени действия в тези случаи или да се извършат други аритметично-логически операции. Това е нужно за да бъде коректното изпълнението на поставената задача, а също и поради това, че имаме 2 опции за визуализиране и моделиране на данните – ръчно и чрез файл.

Функции използвани във връзка със въвеждането на данни - Опция 1

Трябва да се има предвид, че в **mainF.c** функцията, се предоставя възможност на потребителя да въведе броят на елементите/ студентите които би искал да въведе един след друг. Това предоставя на потребителя значителна свобода и най-вече гъвкавост при работата в зависимост от нуждите за даденият момент на обработка на данни за студент/ и. Друга особеност, е че в **mainF.c** функцията бива запомнена и предавана на **int enterBody** стойност на първо-въведената **groupNum** (от student Struct) номер на група и се прави проверка в посочената функция дали ново (второ и тн.) въвежданата отговаря на нея – ако не, се показва съобщение за това коя трябва да е тя и потребителя е подканен посредством съобщение да въведе наново стойност, ако пък отговаря **int enterBody** продължава своето изпълнение – това е нужно за да отговаря на посоченото условие в задачата, че става дума за изпитните резултати на една студентска група.

int enterBody (BODY * ps, int groupNumf, int facNumF, LIST ** pFirst)

Пълната дефиниция на функцията се намира на ред 7, в **cFunc.c**. Приема като параметър указател от тип student Struct, две променливи от integer тип и двоен указател **pFirst** от тип List Struct. Действието на функцията е първо да бъдат декларирани съответните локални променливи нужни за въвеждането на данни за елемента/ студента. Правим проверка дали предаваният указател коректно сочи към съответното място в паметта, тоест дали е заделено съответното адресно пространство за student Struct, и ако не, връща **0**, съобщение за грешка и функцията приключва действието си. При успешно премината проверка, заделя чрез

функцията **memset** памет с големината на student Struct (за да имаме място за всички променливи дефинирани в student 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 нужният контрол за големината на извличаните данни според дефинираният им формат, съответно не се позволява въвеждането на некоректни, за поставените условия, данни – посредством съответните проверки (има повторно извикване на частта от функцията за която това събитие се случва) и/ или „отрязване“ на излишните данни. 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 при първото частта до която е стигнала функцията (Опцията, в случая с EOF) се повтаря, а при второто функцията приключва действието си.

Вътре в самата функция се прави проверка за уникалността на въвежданата данна за Факултетен Номер, посредством функцията **duplFacNum** – тя ще бъде разгледана по-надолу. При успешно, коректно изпълнение за извличане на информация от потребителя, функцията връща флаг **1**, позволявайки ни да продължим нататък към следващата функция извиквана в **mainF.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 206, в **cFunc.c**. Приема като параметри указател сочещ началото на структура от тип свързан списък и променлива от тип student 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 **mainF.c** се преустановява въвеждането на данни за елемента. При коректно заделена памет, декларира, че на отреденото място (за List Struct) body (student 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 **mainF.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 нов студент в едно и също изпълнение на програмата, ние отново ще имаме указателят pFirst, поставящ началото на елемент с данни от динамична структура – едносвързан списък. Следва за Опция 1, в **mainF.c,** няколко проверки на предавани флагове, нулирането им при изпълнение на съответното логическо условие и/ или изпълнение на няколко изчисления посредством извикването на функцията **subjectAvg** (която ще бъде подробно описана по-долу), а и изпълнение на други аритметично - логически действия.

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 запаметен с адреса на предхождащият 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 предхождащият го елемент – това по-нататък ще бъде демонстрирано при обхождането на такъв едносвързан списък почвайки от върхът и стигайки до дъното (началото).

int duplFacNum (int stuID, LIST ** pFirst)

Пълната дефиниция на функцията се намира на ред 223, в **cFunc.c**. Приема като параметри двоен указател сочещ началото на структура от тип свързан списък и променлива от тип integer (Факултетен Номер на Студент). Действието на функцията е да направи обхождане на целият свързаният списък запаметен в локалната памет и при съвпадение на стойността на приеманата integer променлива със същата от полето facNum, за това данново поле от student Struct, връща съобщение към потребителя за наличието на вече въведена (и запаметена) такава стойност и резултат **1** (при който в извикващата функция води до повторно изпълнение на част от нея за да бъде коректно попълнено това данново поле).
При липса на съвпадение, функцията връща резултат **0**, който всъщност указва на извикващата функция, че може да продължи надолу по изпълнение на останалата програмна част от кода за нея – в този случай, проверяваната за уникалност стойност за Факултетен Номер на Студент е такава.
Ето и използваната в **mainF.c**, Опции 1 и 4, функция, която изчислява средният успех за студент:

double subjectAvg (LIST ** pFirst, int numStudents, int times)

Пълната дефиниция на функцията се намира на ред 300, в **cFunc.c**. Приема като параметри двоен указател сочещ началото на структура от тип свързан списък и две променлива от тип integer. Действието на функцията е да обходи списъкът до достигане на неговият край, като в зависимост от подадената променлива **times** се определя за коя Оценка бива сумирана (при обхождането) и след това връщана като Среден Успех за дадената (спрямо броят на студентите).
Като в **mainF.c** поставени съответните аритметично-логически условия и променливи, които да биват приемани/ променяни, за да бъде изпълнено условието на поставената задача.

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

Предварително в **mainF.c** функцията сме поискали от потребителя 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 270, в **cFunc.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 **mainF.c,** като там в логическата операция има съобщение което да се принтира на екрана на потребителя заедно с флаг-а, както и приключва изпълнението на Опция 2. Ако всичко е коректно, започва да записва посредством **fwrite** функцията от началото (в нашия случай стек-овото начало) на свързаният списък, със стъпка размерът на BODY (student 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 **mainF.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 **mainF.c**, където посредством дефиниран **for** цикъл обхождащ списъкът, извиква отново и отново **writeEl**, докато не се получи грешка или не е обходен

списъкът. В **mainF.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, като предварително в **mainF.c** сме поискали от потребителя името на файла от който ще четем данните.
В **mainF.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.
Трябва да се има предвид, че данните въвеждани от файл трябва да отговарят на формата зададен от student Struct, List Struct и на този на функциите извеждащи на екран данните (Опция 4).
Самото четене се осъществява от функцията:

int readEl (BODY * pB, FILE * pF)

Пълната дефиниция на функцията се намира на ред 285, в **cFunc.c**.
Приема като параметри указател от тип student 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 **mainF.c**. Ако всичко е коректно, започва да чете посредством **fread** функцията от файла, със стъпка размерът на BODY (student 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 в **mainF.c**, ако да, функцията **readEl** връща флаг единица в **mainF.c**, където продължаваме истинското зареждане в локалната памет посредством вече 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 **mainF.c** функцията, при различна от едно стойност, се приключва изпълнението на Опция 3.
Тъй като данните за програмата биват зареждани от файл, е нужно да знаем броят на заредените в локалната памет елементи/ студенти, за да може програмата да работи коректно като цяло, а по конкретно да показва правилни резултати при аритметично – логическите операции засягащи целият списък от студенти и техните резултати от изпитната сесия. Както следва функцията извършваща броенето:

int countListMem (LIST ** pFirst)

Пълната дефиниция на функцията се намира на ред 332, в **cFunc.c**. Приема като параметър двоен указател сочещ началото на структура от тип свързан списък. Действието на функцията е да обходи зареденият в локалната памет списък, а когато се достигне краят му, да върне натрупваният в локално декларираната променлива резултат.
Накрая на Опция 3, имаме вдигането на флаг от тип integer в 1, чиято цел е да покаже на друга опция/ и, че сме зареждали данни от файл и е нужно да бъде изпълнена съответната заложена аритметично-логическа операция.

Функциите извършващи печатането на екрана на конзолата всички заредени в локалната памет данни от свързаният списък – Опция 4

В **mainF.c** функцията правим проверка дали директно след като сме заредили данни от файл, сме влезнали в Опция 4, в зависимост от стойността на локално дефинираната променлива за функцията, се пристъпва към извършване на аритметично-логически операции (извикване на функцията **subjectAvg** и др.) и/ или показване на конзолният екран на наличните в локалната памет данни за резултатите от студентската изпитната сесия – като последните подлежат на специфично форматиране, според изискванията на заданието. Както следва това са функциите извършващи показването на данните на екрана:

void print (LIST * pFirst)

Пълната дефиниция на функцията се намира на ред 242, в **cFunc.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 неприключва изпълнението на Опция 4 докато не е достигнат края на свързаният списък и локално дефинираната променлива от тип integer 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)

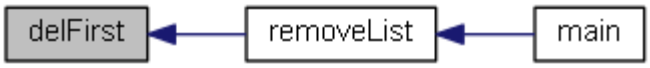
Пълната дефиниция на функцията се намира на ред 257, в **cFunc.c**. Приема като параметър указател от тип student Struct. Действието на функцията е посредством форматиран за специфичните данни **printf** функция да принтира на екрана на конзолата записаните данни в student Struct, като ги достъпва една по една. Запазва резултата от **printf** функцията в локално дефинирана променлива от тип integer, като при неуспешно изпълнение на операцията връща флаг **0**, а при успешно флаг **1** (съответно води до изпълнение на самата функция още веднъж).

Функции за изтриване на елементите от свързаният списък и излизане от програмата – Опция 5

void removeList (LIST ** pFirst)

Пълната дефиниция на функцията се намира на ред 349, в **cFunc.c**. Приема като параметър двоен указател **pFirst** от тип List Struct. Действието на функцията е първо да дефинира локална променлива от тип student Struct и да предаде единичен указател **pFirst** и адреса на локално дефинираната променлива на функцията **delFirst**. Това предаване на стойности се повтаря докато **delFirst** връща **pFirst** адресна стойност различна от NULL.

LIST* delFirst (LIST * pFirst, BODY * delBody)



Пълната дефиниция на функцията се намира на ред 358, в **cFunc.c**. Приема като параметри указател **pFirst** от тип List Struct и указател от тип student Struct. Действието на функцията е първо да провери дали **pFirst** е NULL, ако да, връща съобщение към потребителя на програмата и стойността, ако не, освобождава заетата от съответният елемент памет и връща указател към следващият елемент от едносвързаният списък.

Source Code на програмата

cFunc.c

```
#include "hFunc.h"
#include "windows.h"
/*****************************************FUNCTIONS
 ************************************************************/

/* The data entry function for students. It does the student input into a list
via a pointer ps and flag: 0 (error upon data entry), 1 (correct data entry - if
 no errors are encountered in the process). */
int enterBody(BODY *ps, int groupNumf, int facNumF, LIST **pFirst)
{
 int c, i = 0, k = 0, z = 0, x = 0, n = 0, m = 0, ret, fGrNum = 0;
 char strFNUM [LINE], strGrNUM [LINE], mOne[LINE], mTwo[LINE], mThree[LINE],
 mFour[LINE]; // making them all the standard windows
 if(ps == NULL) // cmd screen buffer size, just to be sure
 return 0;
 memset(ps, 0, sizeof(BODY));

stuFacNum:
 do
 {
 fflush(stdin);
 printf("\nStudent Faculty Number |An integer value of 9 digits,NO 0 at start|:
 ");
 c = getchar();
 if ( c == '\n' ){ ungetc(c, stdin); printf("\nError, nothing has been entered,
 try again!!\n"); goto stuFacNum;}
 if ( c == EOF ){ ungetc(c, stdin); return 0;}
 ungetc(c, stdin);
 scanf("%s", &strFNUM);
 if(strlen(strFNUM) > LINE)
 strFNUM[LINE] = '\0';
 }while ( (strlen(strFNUM) > 9) || (strlen(strFNUM) < 9) );
 if(strFNUM [0] == '0'){printf("\nError, you can NOT have a ZERO -> 0 entered as
 a \nbeginning of this set of integer values, try again!!\n"); goto stuFacNum;}
 for (z; z < 9; z++)
 {
 if( ispunct(strFNUM [z]) )
 {
 printf("\nError, you can NOT have a punctuation character entered in this
 set\nof integer values, try again!!\n"); goto stuFacNum;
 }
 }
 for (x; x < 9; x++)
 {
 if( isalpha(strFNUM [x]) )
 {
```

```
40  printf("\nError, you can NOT have an alphabetic character entered in this
    set\nof integer values, try again!!\n"); goto stuFacNum;
41  }
42  }
43  for (i; i < 9; i++)
44  {
45  if ( !isdigit(strFNUM [i]) )
46  {
47  printf("\nError, you have not entered a set of integer values, try again!!\n");
    goto stuFacNum;
48  }
49  }
50  strFNUM[10] = '\0';
51  ps->facNum = atoi (strFNUM);
52
53  if (facNumF == 0){ ret = duplFacNum(ps->facNum, pFirst); if (ret == 1) { goto
    stuFacNum; } if(ret == 0){ goto stuFirstName;} }
54
55 stuFirstName:
56  fflush(stdin);
57  printf("\nStudent, First Name |Character set of up to 12 symbols|: ");
58  if(gets(ps->firstName) == NULL)
59  return 0; // CTRL/Z has been pressed
60  if(ps->firstName[0] == '\0')
61  {//if nothing has been entered
62  printf("\nError, nothing has been entered, try again\n");
63  return 0;
64  }
65  if(strlen(ps->firstName) > NAME_LENGTH)
66  ps->firstName[NAME_LENGTH] = '\0';
67
68  fflush(stdin);
69  printf("\nStudent, Second Name |Character set of up to 12 symbols|: ");
70  if(gets(ps->secondName) == NULL)
71  return 0; // CTRL/Z has been pressed
72  if(ps->secondName[0] == '\0')
73  {//if nothing has been entered
74  printf("\nError, nothing has been entered, try again\n");
75  return 0;
76  }
77  if(strlen(ps->secondName) > NAME_LENGTH)
78  ps->secondName[NAME_LENGTH] = '\0';
79
80  fflush(stdin);
81  printf("\nStudent, Third Name |Character set of up to 12 symbols|: ");
82  if(gets(ps->thirdName) == NULL)
83  return 0; // CTRL/Z has been pressed
84  if(ps->thirdName[0] == '\0')
85  {//if nothing has been entered
86  printf("\nError, nothing has been entered, try again\n");
87  return 0;
88  }
89  if(strlen(ps->thirdName) > NAME_LENGTH)
90  ps->thirdName[NAME_LENGTH] = '\0';
91
92 stuGroupNum:
93  do
94  {
95  fflush(stdin);
96  printf("\nStudent Group Number |It is an integer value of 3 digits,NO 0 at
    start|: ");
97  c = getchar();
98  if ( c == '\n' ){ ungetc(c, stdin); printf("\nError, nothing has been entered,
    try again!!\n"); goto stuGroupNum;}
99  if ( c == EOF ){ ungetc(c, stdin); return 0;}
```

```
100  ungetc(c, stdin);
101  scanf("%s", &strGrNUM);
102  if(strlen(strGrNUM) > LINE)
103  strGrNUM[LINE] = '\0';
104  }while ( strlen(strGrNUM) > 3 || strlen(strGrNUM) < 3);
105  if(strGrNUM [0] == '0'){printf("\nError, you can NOT have a ZERO -> 0 entered
     as a \nbeginning of this set of integer values, try again!!\n"); goto
     stuGroupNum;}
106  for (n; n < 3; n++)
107  {
108  if( ispunct(strGrNUM [n]) )
109  {
110  printf("\nError, you can NOT have a punctuation character entered in this
     set\nof integer values, try again!!\n"); goto stuGroupNum;
111  }
112  }
113  for (m; m < 3; m++)
114  {
115  if( isalpha(strGrNUM [m]) )
116  {
117  printf("\nError, you can NOT have an alphabetic character entered in this
     set\nof integer values, try again!!\n"); goto stuGroupNum;
118  }
119  }
120  for (k; k < 3; k++)
121  {
122  if ( !isdigit(strGrNUM [k]) )
123  {
124  printf("\nError, you have not entered a set of integer values, try again!!\n");
     goto stuGroupNum;
125  }
126  }
127  strGrNUM[4] = '\0';
128  ps->groupNum = atoi (strGrNUM);
129  if (groupNumf != 0){ if( (ps->groupNum != groupNumf) ) {printf("\nError, you
     have not entered the correct Group Number %d, try again!!\n", groupNumf); goto
     stuGroupNum; } }
130
131
132 marOne:
133  do
134  {
135  fflush(stdin);
136  printf("\nEnter a Student Grade for Subject A |A real or an integer number from
     2.00 to 6.00|: ");
137  c = getchar();
138  if ( c == '\n' ){ ungetc(c, stdin); printf("\nError, nothing has been entered,
     try again!!\n"); goto marOne;}
139  if ( c == EOF ){ ungetc(c, stdin); return 0;}
140  ungetc(c, stdin);
141  scanf("%s", &mOne);
142  if(strlen(mOne) > LINE)
143  mOne[LINE] = '\0';
144  }while ( strlen(mOne) != 1 );
145  if ( !isdigit(mOne[0]) ){ printf("\nError, you have not entered a digit, try
     again!!\n"); goto marOne; }
146  ps->markOne = atof(mOne);
147  if( (ps->markOne < 2 || ps->markOne > 6) ){ printf("\nError, you have not
     entered an integer number from 2.00 to 6.00, try again!!\n"); goto marOne;}
148
149 marTwo:
150  do
151  {
152  fflush(stdin);
```

```
153  printf("\nEnter a Student Grade for Subject B |A real or an integer number from
     2.00 to 6.00|: ");
154  c = getchar();
155  if ( c == '\n' ){ ungetc(c, stdin); printf("\nError, nothing has been entered,
     try again!!\n"); goto marTwo;}
156  if ( c == EOF ){ ungetc(c, stdin); return 0;}
157  ungetc(c, stdin);
158  scanf("%s", &mTwo);
159  if(strlen(mTwo) > LINE)
160  mTwo[LINE] = '\0';
161  }while ( strlen(mTwo) != 1 );
162  if ( !isdigit(mTwo[0]) ){ printf("\nError, you have not entered a digit, try
     again!!\n"); goto marTwo; }
163  ps->markTwo = atof(mTwo);
164  if( (ps->markTwo < 2 || ps->markTwo > 6) ){ printf("\nError, you have not
     entered an integer number from 2.00 to 6.00, try again!!\n"); goto marTwo; }
165
166 marThree:
167  do
168  {
169  fflush(stdin);
170  printf("\nEnter a Student Grade for Subject C |A real or an integer number from
     2.00 to 6.00|: ");
171  c = getchar();
172  if ( c == '\n' ){ ungetc(c, stdin); printf("\nError, nothing has been entered,
     try again!!\n"); goto marThree;}
173  if ( c == EOF ){ ungetc(c, stdin); return 0;}
174  ungetc(c, stdin);
175  scanf("%s", &mThree);
176  if(strlen(mThree) > LINE)
177  mThree[LINE] = '\0';
178  }while ( strlen(mThree) != 1 );
179  if ( !isdigit(mThree[0]) ){ printf("\nError, you have not entered a digit, try
     again!!\n"); goto marThree; }
180  ps->markThree = atof(mThree);
181  if( (ps->markThree < 2 || ps->markThree > 6) ){ printf("\nError, you have not
     entered an integer number from 2.00 to 6.00, try again!!\n"); goto marThree; }
182
183 marFour:
184  do
185  {
186  fflush(stdin);
187  printf("\nEnter a Student Grade for Subject D |A real or an integer number from
     2.00 to 6.00|: ");
188  c = getchar();
189  if ( c == '\n' ){ ungetc(c, stdin); printf("\nError, nothing has been entered,
     try again!!\n"); goto marFour;}
190  if ( c == EOF ){ ungetc(c, stdin); return 0;}
191  ungetc(c, stdin);
192  scanf("%s", &mFour);
193  if(strlen(mFour) > LINE)
194  mFour[LINE] = '\0';
195  }while ( strlen(mFour) != 1 );
196  if ( !isdigit(mFour[0]) ){ printf("\nError, you have not entered a digit, try
     again!!\n"); goto marFour; }
197  ps->markFour = atof(mFour);
198  if( (ps->markFour < 2 || ps->markFour > 6) ){ printf("\nError, you have not
     entered an integer number from 2.00 to 6.00, try again!!\n"); goto marFour; }
199
200  ps->avgGrade = (ps->markOne + ps->markTwo + ps->markThree + ps->markFour)/4;
201 return 1;
202 }

203 //=============================================================================
    ====================================
```

```
204 /* Includes element with data newBody at the beginning of pFirst in the list.
205 Returns a pointer of the list's beginning or NULL upon an unsuccessful operation
    */
206 LIST *insertBegin(LIST *pFirst, BODY newBody)
207 {
208  LIST *p;
209  p = (LIST *)malloc(sizeof(LIST)); // saves a mem location
210  if (p == NULL)
211  { printf("\nError! There is not enough memory!\n");
212  return NULL;
213  }
214  else
215  { p->body = newBody; // saves the data into the new element
216  p->pNext = pFirst; // points the element at the beginning
217  pFirst = p; // sets the beginning at the new element
218  return p;
219  }
220 }

    221 //================================================================================
    ========================================
222 /* Searches the student list members for duplicate Student Faculty members, if
    true returns 1, else just breaks the search and returns 0 */
223 int duplFacNum (int stuID, LIST **pFirst)
224 {
225  LIST **current;
226  LIST *pNext;
227  for (current = pFirst, pNext = (*current)->pNext; *current; current =
    &(*current)->pNext, pNext = (*current)->pNext)
228  {
229  if((*current)->body.facNum == stuID)
230  {
231  printf("\nThere is already a student with Faculty Number: %d, try again!!\n",
    stuID);
232  return 1;
233  }
234  if(pNext == NULL)
235  {
236  return 0;
237  }
238  }
239 }

    240 //================================================================================
    ========================================
241 /* Prints out the elements of list that has a pointer start pFirst. */
242 void print(LIST *pFirst)
243 {
244  int res;
245  LIST *p;
246  p=pFirst;
247  while(p!=NULL || res == 0)
248  {
249  res = printBody(p->body);
250  p=p->pNext;
251  }
252  printf("\n");
253 }

    254 //================================================================================
    ========================================================
255 /* Displays the Student Data on the Screen.
256 Returns a flag: 0 (error upon print), 1 (successful operation). */
257 int printBody(BODY s)
258 {
```

```
 int res;
 fflush(stdin);
 res = printf("%d %-12s %-12s %-12s %d %-.2f %-.2f %-.2f %-.2f %.4lf\n",
s.facNum, s.firstName, s.secondName, s.thirdName,
 s.groupNum, s.markOne, s.markTwo, s.markThree, s.markFour, s.avgGrade);
 if(res < 0) return 0;
 else return 1;
}

//=============================================================================
=================================================
/* Saves the data of an element from a list that the pL pointer points to, into
 a binary file
with a pointer pF. Returns a flag: 1 (succesful save),
-1 (zero pointer to the data), -2(zero pointer to the file), -3 (Error upon a
 save into the file). */
int writeEl(LIST *pL, FILE *pF)
{
 int res;
 if (pL == NULL) return -1;
 if (pF == NULL) return -2;
 res = fwrite(&(pL->body), sizeof(BODY), 1, pF);
 if (res == 1)
 return 1; // Success
 else
 return -3; // Error upon a save into the file
}

//=============================================================================
=========================================
/*Reads the data from a binary file with a pF pointer. Returns data via the
 pointer pB and flag: 1 (successful reading),
-1 (zero pointer to the data), -2 (zero pointer to the file), -3 (Error upon
 reading from the file),
-4 (end of File has been reached). */
int readEl(BODY *pB, FILE *pF)
{
int res;
if (pB == NULL) return -1;
if (pF == NULL) return -2;
res = fread(pB, sizeof(BODY), 1, pF);
 if (res == 1) return 1; // Success
 else
 {
 if (feof(pF) ) return -4; // End of File has been reached
 return -3; // Error upon reading from the file
 }
}

//=============================================================================
=========================================
/* Calculates the Grade Point Average per subject via a for cycle that goes
 through the list of student members*/
double subjectAvg(LIST **pFirst, int numStudents, int times)
{
 LIST **current;
 LIST *pNext;
 double sum = 0, avg = 0;
 for (current = pFirst, pNext = (*current)->pNext; *current; current =
&(*current)->pNext, pNext = (*current)->pNext)
 {
 if( times == 0 )
 {
 sum += ((*current)->body.markOne);
 }
```

```
311  if( times == 1 )
312  {
313  sum += ((*current)->body.markTwo);
314  }
315  if( times == 2 )
316  {
317  sum += ((*current)->body.markThree);
318  }
319  if( times == 3 )
320  {
321  sum += ((*current)->body.markFour);
322  }
323  if(pNext == NULL)
324  {
325  avg = sum/(double)numStudents;
326  return avg;
327  }
328  }
329 }

330 //=============================================================================
    ============================================
331 /* A simple function that returns the number of students; it uses a for cycle to
    go through the list*/
332 int countListMem(LIST **pFirst)
333 {
334  LIST **current;
335  LIST *pNext;
336  int count = 0;
337  for (current = pFirst, pNext = (*current)->pNext; *current; current =
    &(*current)->pNext, pNext = (*current)->pNext)
338  {
339  ++count;
340  if(pNext == NULL)
341  {
342  return count;
343  }
344  }
345 }

346 //=============================================================================
    ============================================
347 /* It releases the memory reserved by the list. It gets the beginning of the
    list
348 via a pointer to the beginning of the list pFirst. */
349 void removeList(LIST **pFirst)
350 {
351  BODY first;
352  while(*pFirst != NULL)
353  *pFirst = delFirst(*pFirst, &first);
354 }

355 //=============================================================================
    ========================
356 /* Deletes an element at the beginning of the list. Returns the data of deleted
    element
357 via the delBody pointer and a pointer to the beginning of the list or NULL if
    unsuccesful */
358 LIST *delFirst(LIST *pFirst, BODY *delBody)
359 { if (pFirst == NULL)
360  { printf("\nEmpty list!\n");
361  return NULL;
362  }
363  else
364  { LIST *p; // temp link
```

```
 *delBody = pFirst->body; // saves the data
 p = pFirst->pNext; // it points the link to the second element
 if(p != NULL)
 free (pFirst); // releases the alocated mem
 pFirst = p; // it points the beginning to the link
 return pFirst;
 }
}
```

hFunc.h

```
#include <stdio.h>
#include <malloc.h> //modify the behavior of malloc, realloc, and free by specifying appropriate hook functions
#include <string.h>
#include "windows.h"
#define NAME_LENGTH 12
#define LINE 80

struct student
{
 int facNum;
 char firstName [NAME_LENGTH + 1];
 char secondName [NAME_LENGTH + 1];
 char thirdName [NAME_LENGTH + 1];
 int groupNum;
 double markOne;
 double markTwo;
 double markThree;
 double markFour;
 double avgGrade;
};typedef struct student BODY;
struct List
{
BODY body;
struct List *pNext;
};typedef struct List LIST;

LIST *insertBegin(LIST *pFirst, BODY newBody);
int enterBody(BODY *ps, int groupNumf, int facNumF, LIST **pFirst);
int duplFacNum (int stuID, LIST **pFirst);
void print(LIST *pFirst);
int printBody(BODY s);
int writeEl(LIST *pL, FILE *pF);
int readEl(BODY *pB, FILE *pF);
double subjectAvg(LIST **pFirst, int numStudents, int times);
int countListMem(LIST **pFirst);
void removeList(LIST **pFirst);
LIST *delFirst(LIST *pFirst, BODY *delBody);
```

mainF.c

```
#include "hFunc.h"
#include <stdlib.h>
#include "windows.h"
int main()
{
 LIST *pFirst = NULL, *p;
 int res, i, mode, c, fGroupNum = 1, fGrNum = 0, fiFacNum = 1, k, o, loops, actStudC = 0 , fileStudCount = 0, flag = 0;
 double avgMarkOne = 0, avgMarkTwo = 0, avgMarkThree = 0, avgMarkFour = 0, avgMarkAll = 0;
```

```
 9  FILE *pOut = NULL, *pIn = NULL;
10  char fOutName [30], fInName [30];
11  BODY student;
12
13  char *menu[] = {"STUDENT EXAM SESSION DATA\n SUPPORT DATA MENU",
14  "1-Enter manually the data for a new student",
15  "2-Write the student data into a binary file",
16  "3-Read student data from a binary file",
17  "4-Display all available students' information",
18  "5-Destroy the student information and Exit"};
19
20 do
21 {
22  system("cls");
23  for(i=0; i < 6; i++)
24  printf("\n%s\n",menu[i]);
25 do
26 {
27 fflush(stdin);
28 printf ("\n\nChoose mode[1-5]: ");
29 res = scanf("%d", &mode);
30 }while(res !=1);
31  switch(mode)
32  {
33  case 1: //data entry from the keyboard
34  fflush(stdin);
35  printf("\nEnter the number of students' information you wish to enter: ");
36  c = getchar();
37  if ( c == '\n' ){ ungetc(c, stdin); printf("\nError, nothing has been entered,
    try again!!\n"); system("pause"); break;}
38  if ( c == EOF ){ ungetc(c, stdin); system("pause"); break;}
39  ungetc(c, stdin);
40  scanf("%d", &loops);
41  fflush(stdin);
42  for(i = 0; i < loops; i++)
43  {
44  system("cls");
45  fflush(stdin);
46  res = enterBody(&student, fGrNum, fiFacNum, &pFirst);
47  if (res != 1 ) //the function returns 0 or 1
48  {
49  printf("\nError in initialization %d!\n", res);
50  break;
51  }
52  if (fGroupNum == 1) {fGrNum = student.groupNum;}
53  p = insertBegin(pFirst, student);
54  if(p == NULL)
55  {
56  printf("\n\nNot enough memory!\n");
57  system("pause");
58  break;
59  }
60  pFirst = p;
61  ++actStudC;
62  fGroupNum = 0;
63  fiFacNum = 0;
64  }
65  if(actStudC > 0)
66  {
67
68  if (flag == 1){actStudC = (actStudC + fileStudCount); flag = 0;}
69  for (k = 0; k < 4; k++)
70  {
71  if(k == 0) avgMarkOne = subjectAvg(&pFirst, actStudC, k);
72  if(k == 1) avgMarkTwo = subjectAvg(&pFirst, actStudC, k);
```

```
 73  if(k == 2) avgMarkThree = subjectAvg(&pFirst, actStudC, k);
 74  if(k == 3) avgMarkFour = subjectAvg(&pFirst, actStudC, k);
 75  }
 76  avgMarkAll = (avgMarkOne + avgMarkTwo + avgMarkThree + avgMarkFour)/4;
 77  }
 78  system("pause");
 79  break;
 80
 81  case 2: // openning the file and writing on it the list
 82  fflush(stdin);
 83  if (pFirst == NULL)
 84  { printf("\nThe STUDENT EXAM SESSION DATA is Empty,\nthere is no student data
     to be saved!\n\n");system("pause");break;}
 85  printf("\nEnter the Name of the File you would like to record the data to\n|It
     should be an alphameric sequence of up 25 symbols ending with .dat|:\n");
 86  c = getchar();
 87  if ( c == '\n' ){ ungetc(c, stdin); printf("\nError, nothing has been entered,
     try again!!\n"); system("pause"); break;}
 88  if ( c == EOF ){ ungetc(c, stdin); system("pause"); break;}
 89  ungetc(c, stdin);
 90  scanf("%s", &fOutName);
 91  pOut = fopen(fOutName, "wb");
 92  if(pOut == NULL)
 93  {
 94  printf("\nCan't open file for writing!\n");
 95  system("pause");
 96  break;
 97  }
 98  for(p = pFirst; p != NULL ; p = p->pNext)
 99  {
100  res = writeEl(p, pOut);
101  if(res != 1)
102  {
103  printf("\nWriting error %d !\n\n", res);
104  system("pause");
105  break;
106  }
107  }
108  fclose(pOut);
109  removeList(&pFirst);
110  printf("\nThe STUDENT EXAM SESSION DATA has been recorded on the %s file\n",
     fOutName);
111  system("pause");
112  break;
113
114  case 3: // openning the file and reading the list
115  fflush(stdin);
116  {
117  printf("\nEnter the Name of the File you would like to read the data from\n| It
     should be an alphameric sequence of up 25 symbols ending with .dat |:\n");
118  c = getchar();
119  if ( c == '\n' ){ ungetc(c, stdin); printf("\nError, nothing has been entered,
     try again!!\n"); system("pause"); break;}
120  if ( c == EOF ){ ungetc(c, stdin); system("pause"); break;}
121  ungetc(c, stdin);
122  scanf("%s", &fInName);
123  pIn = fopen(fInName, "rb");
124  if( pIn == NULL)
125  {
126  printf("\nCan't open file for reading!\n");
127  system("pause");
128  break;
129  }
130  do
131  {
```

```
132  res = readEl(&student, pIn);
133  if (res != 1 && res != -4 )
134  {
135  printf("\nReading error %d !\n", res);
136  system("pause");
137  break;
138  }
139  if (res != -4)
140  {
141  p = insertBegin(pFirst, student);
142  if ( p == NULL )
143  {
144  printf("\nNot enough memory!\n");
145  system("pause");
146  break;
147  }
148  pFirst = p;
149  }
150  }while(res == 1);
151  fclose(pIn);
152  printf("\nThe file has been read and the STUDENT EXAM SESSION DATA\nis
     available for Display via Option 4.\n");
153  }
154  fileStudCount = countListMem(&pFirst);
155  flag = 1;
156  system("pause");
157  break;
158
159  case 4: // Displaying all
160  fflush(stdin);
161  if (pFirst!=NULL)
162  {
163  if(flag == 1)
164  {
165  for (o = 0; o < 4; o++)
166  {
167  if(o == 0) avgMarkOne = subjectAvg(&pFirst, fileStudCount, o);
168  if(o == 1) avgMarkTwo = subjectAvg(&pFirst, fileStudCount, o);
169  if(o == 2) avgMarkThree = subjectAvg(&pFirst, fileStudCount, o);
170  if(o == 3) avgMarkFour = subjectAvg(&pFirst, fileStudCount, o);
171  }
172  avgMarkAll = (avgMarkOne + avgMarkTwo + avgMarkThree + avgMarkFour)/4;
173  //flag = 0;
174  }
175  printf("\nRESULTS from a Winter Exam session\n\n");
176  printf("FacNum: Name,First: Name,Second: Name,Third: GrNum: sA: sB: sC: sD:
     AvgGrd:\n");
177  print(pFirst);
178  printf(" Grade Point Average %-.2f %-.2f %-.2f %-.2f %.4lf\n", avgMarkOne,
     avgMarkTwo, avgMarkThree, avgMarkFour, avgMarkAll);
179  }
180  else
181  printf("\nThe STUDENT EXAM SESSION DATA is Empty,\nthere are no student data to
     be Displayed!\n\n");
182  system("pause");
183  break;
184
185  case 5:
186  fflush(stdin);
187  if (pFirst!=NULL)
188  removeList(&pFirst);
189  printf("\nThe STUDENT EXAM SESSION DATA is Empty!\nProgram exiting...\n");
190  break;
191
192  default:
```

```
193  printf("\nBad choice! \n");
194  system("pause");
195  }
196  }while(mode != 5); // because the list gets destroyed and we exit the program
197  return 0;
198 }
```

Контролен Пример от изпълнението на програмата

Началното меню

```
C:\Windows\system32\cmd.exe

STUDENT EXAM SESSION DATA
    SUPPORT DATA MENU

1-Enter manually the data for a new student

2-Write the student data into a binary file

3-Read student data from a binary file

4-Display all available students' information

5-Destroy the student information and Exit


Choose mode[1-5]:
```

Опция 1

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

Enter the number of students' information you wish to enter: _
```

```
Student Faculty Number |An integer value of 9 digits,NO 0 at start|: ^Z

Error in initialization 0!
Press any key to continue . . . _
```

```
Student Faculty Number |An integer value of 9 digits,NO 0 at start|: 012345678

Error, you can NOT have a ZERO -> 0 entered as a
beginning of this set of integer values, try again!!

Student Faculty Number |An integer value of 9 digits,NO 0 at start|: a12345678

Error, you can NOT have an alphabetic character entered in this set
of integer values, try again!!

Student Faculty Number |An integer value of 9 digits,NO 0 at start|: !12345678

Error, you have not entered a set of integer values, try again!!
```

```
Student Faculty Number |An integer value of 9 digits,NO 0 at start|: abcdefghi

Error, you have not entered a set of integer values, try again!!

Student Faculty Number |An integer value of 9 digits,NO 0 at start|:
```

```
Student Faculty Number |An integer value of 9 digits,NO 0 at start|: 123456789

There is already a student with Faculty Number: 123456789, try again!!

Student Faculty Number |An integer value of 9 digits,NO 0 at start|: _
```

```
Student Faculty Number |An integer value of 9 digits,NO 0 at start|: 123456788

Student, First Name |Character set of up to 12 symbols|: Stefan

Student, Second Name |Character set of up to 12 symbols|: Stefanov

Student, Third Name |Character set of up to 12 symbols|: Baychev

Student Group Number |It is an integer value of 3 digits,NO 0 at start|: 1a1

Error, you can NOT have an alphabetic character entered in this set
of integer values, try again!!

Student Group Number |It is an integer value of 3 digits,NO 0 at start|: 111

Enter a Student Grade for Subject A |A real or an integer number from 2.00 to 6.
00|: 6

Enter a Student Grade for Subject B |A real or an integer number from 2.00 to 6.
00|: 6

Enter a Student Grade for Subject C |A real or an integer number from 2.00 to 6.
00|: 6

Enter a Student Grade for Subject D |A real or an integer number from 2.00 to 6.
00|: 6
```

Опция 2

```
Choose mode[1-5]: 2

Enter the Name of the File you would like to record the data to
|It should be an alphameric sequence of up 25 symbols ending with .dat|:
green.dat

The STUDENT EXAM SESSION DATA has been recorded on the green.dat file
Press any key to continue . . .
```

Опция 3

```
Choose mode[1-5]: 3

Enter the Name of the File you would like to read the data from
| It should be an alphameric sequence of up 25 symbols ending with .dat |:
green.dat

The file has been read and the STUDENT EXAM SESSION DATA
is available for Display via Option 4.
Press any key to continue . . .
```

Опция 4

```
Choose mode[1-5]: 4

RESULTS from a Winter Exam session

FacNum:   Name,First:  Name,Second: Name,Third:  GrNum: sA: sB: sC: sD:  AvgGrd:

987654329 zlatina      zlatineva    zlatineva    111 4.00 5.00 5.00 6.00 5.0000
987654328 teodora      teodoroa     teodorova    111 5.00 5.00 6.00 6.00 5.5000
987654327 qna          qneva        qneva        111 4.00 3.00 5.00 3.00 3.7500
987654326 grigor       grigorev     grirevski    111 2.00 3.00 4.00 2.00 2.7500
987654325 stancho      stanchev     stanchev     111 4.00 4.00 5.00 6.00 4.7500
987654324 pencho       penchev      penchev      111 3.00 4.00 5.00 5.00 4.2500
987654323 georgi       georgiev     georgiev     111 6.00 6.00 3.00 3.00 4.5000
987654322 stefan       stefanov     stefanov     111 3.00 4.00 5.00 6.00 4.5000
987654320 petar        petrov       petrov       111 6.00 6.00 4.00 4.00 5.0000
987654321 iavn         ivanov       ivanov       111 6.00 5.00 4.00 3.00 4.5000

                                 Grade Point Average 4.30 4.50 4.60 4.40 4.4500
Press any key to continue . . .
```

Опция 5

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

The STUDENT EXAM SESSION DATA is Empty!
Program exiting...
Press any key to continue . . .
```