ТЕХНИЧЕСКИ УНИВЕРСИТЕТ – ВАРНА

Факултет по изчислителна техника и автоматизация

Катедра „СИТ“

**СЕМЕСТРИАЛНА ДОМАШНА РАБОТА**

по дисциплината „Базово програмиране”

на тема: **Информационна система „Студенти”**

Вариант 161

**Изготвил:**  **Проверил:**

Петя Георгиева Игнатова проф. Милена Карова

**Специалност:** СИТ

**Група:** I a

**Факултетен номер:** 23621639

2024

Съдържание

[I. Задание на проекта 3](#_Toc154846362)

[II. Анализ на решението 5](#_Toc154846363)

[1. Структура за данните в програмата 5](#_Toc154846364)

[2. Реализация на условие А 5](#_Toc154846365)

[3. Реализация на условие B 8](#_Toc154846366)

[4. Реализация на условие C 10](#_Toc154846367)

[5. Реализация на условие D 11](#_Toc154846368)

[6. Реализация на условие E 13](#_Toc154846369)

[7. Реализация на условие F 14](#_Toc154846370)

[8. Реализация на условие G – допълнение първо 18](#_Toc154846371)

[9. Реализация на условие G (a,b) – допълнение първо 20](#_Toc154846372)

[10. Реализация на условие H (a,b) – допълнение второ 24](#_Toc154846373)

[11. Реализация на условие I – допълнение второ 26](#_Toc154846374)

[12. Реализация на условие J – допълнение трето 28](#_Toc154846375)

[III. Примерно действие на програмата 29](#_Toc154846376)

[1. Условие А 29](#_Toc154846377)

[2. Условие B 31](#_Toc154846378)

[3. Условие C 33](#_Toc154846379)

[4. Условие D 33](#_Toc154846380)

[5. Условие E 35](#_Toc154846381)

[6. Условие F 35](#_Toc154846382)

[7. Условие G - допълнение първо 37](#_Toc154846383)

[8. Условие G (a,b) - допълнение първо 39](#_Toc154846384)

[9. Условие H (a,b) - допълнение второ 41](#_Toc154846385)

[10. Условие I - допълнение второ 45](#_Toc154846386)

[1. Условие J - допълнение трето 47](#_Toc154846387)

[I. Изходен код на програмата 48](#_Toc154846388)

# Задание на проекта

**Информационна система „Студенти“**

Да се напише компютърна програма, реализираща информационна система, която поддържа студентска група (факултетен номер, ЕГН, три имена, списък с дисциплини, пол, възраст, статут-активен/прекъснал/завършил). Програмата съхранява и обработва данни за студентите в групата. Максималния брой студенти в една група е 30. Всеки студент има 5 оценки по дисциплините (име, оценка) от 1 семестър.

**Базова задача**

A. Меню за избор на функциите от програмата.

**Функции от програмата са:**

B. Добавяне на студенти в група:

a. Добавяне на студенти в групата, чрез добавяне в масива, като не трябва да се превишава максималният брой студенти в една група (30).

***Пример:*** *Добавяне на списък от студенти. Въвежда се цяло число* ***n*** *и след него* ***n*** *на брой студенти .* ***n*** *не може да надвишава свободните елементи в масива със студенти.* ***Валидни оценки са 0 – няма оценка, 2, 3, 4, 5, 6.***

C. Извеждане на всички студенти на екрана:

a. Извеждане на всички студенти, чрез подходящо форматиране в таблица

D. Търсене и извеждане на екрана:

a. Търсене и извеждане на студенти с оценка по БП в определен в интервал [min – max]

b. Търсене и извеждане на студенти с минимум един предмет, с оценка Слаб 2.

E. Сортиране и извеждане на студенти на екран:

a. Сортиране на студентите по малко име по азбучен ред.

F. Работа с външен файл(двоичен или текстов):

a. Извеждане на масива със студенти във файл.

b. Въвеждане на масива от файл.

**Допълнение първо (+ базова задача)**

G. Да се създаде подменю към основното с нови функции за:

a. Отделяне на студенти с определен статут (въвежда се от потребителят) и сортиране по факултетен номер.

b. Отделяне на студенти с успех в интервал (мин - макс) и сортиране по ЕГН.

**Допълнение второ (+ базова задача)**

H. Въвеждане/Актуализация на оценки и среден успех за студент:

a. Добавяне в структурата на поле среден успех.

b. Въвеждане/Актуализацията на данни (оценки и среден успех), става по въведен факултетен номер на студент.

i. Ако студентът присъства в списъка(масива):

1. Проверява се статута му:

a. Активен – при въвеждане/актуализация на оценки, се изчислява автоматично средният успех. Всяка промяна на оценка изисква запитване за промяна! Ако няма оценка, тя се актуализира без запитване!

b. Прекъснал/завършил – НЕ могат да се актуализират оценки.

c. Ако студента не присъства в списъка(масива), се извежда пояснително съобщение на екрана.

I. Смяна на статут на студент:

a. Въвежда се факултетният номер на студента (ако го има, се въвежда и новият статут). Ако статутът е активен или прекъснал се сменя с новият. Ако статутът е завършил НЕ СЕ допуска промяна!

**Допълнение трето (+ базова задача)**

J. Данните в програмата да се попълват автоматично от файл при стартиране и да се записват автоматично във файл при затваряне на програмата.

# Анализ на решението

## Структура за данните в програмата

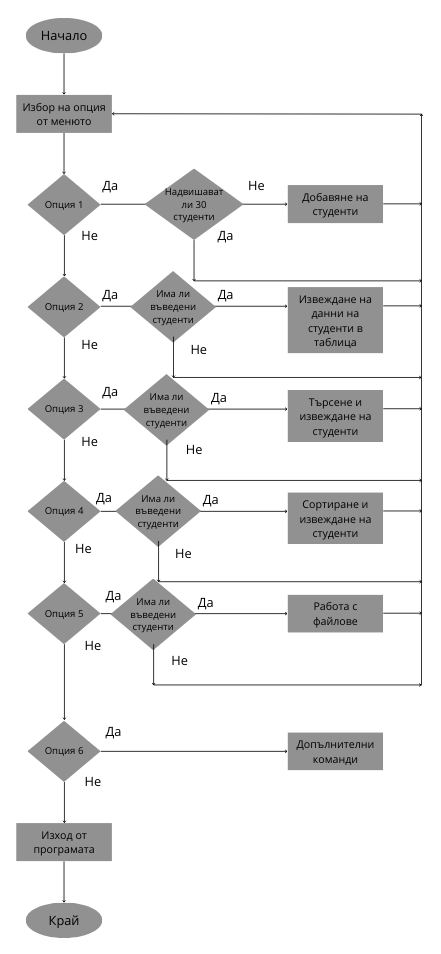
|  |  |  |
| --- | --- | --- |
| Структура | Обяснение | Примерни стойности |
| struct Students  {  string fakNomer;  string egn;  string firstName;  string secondName;  string thirdName;  string discipline[5];  string gender;  int age = 0;  string statut;  int grades[5];  double avarageScore;  }; | -fakNomer – ще съхранява фак. номер на студента  -egn - ще съхранява ЕГН-то на студента  -firstName - ще съхранява името на студента  -secondName - ще съхранява презимето на студента  -thirdName - ще съхранява фамилията на студента  -discipline - ще съхранява масив с до 5 дисциплини на студента  -gender - ще съхранява пола на студента  -age - ще съхранява възрастта на студента  -statut - ще съхранява статута на студента  -grades - ще съхранява масив с до 5 оценки на студента  -avarageScore - ще съхранява средния успех на студента | 23621622  0445161423  Ivaylo  Grigorov  Ivanov  BP  male  19  active  6  3 |

## Реализация на условие А

* 1. Алгоритъм Меню с избор на функциите от програмата

Алгоритъмът за създаване на меню с функции предоставя на потребителя възможност да избира основните функционалности чрез опции, представени в основното меню. Главният алгоритъм извиква подпрограма за изграждане на менюто и проверка на валидността на избраната опция. Главният алгоритъм се реализира чрез функцията int main(), докато подалгоритъмът за менюто се осъществява чрез функцията void Menu().

* 1. Блок схема на алгоритъма (Фиг. 1, Фиг. 2)



*Фиг. 1 Блок схема на Алгоритъм Меню с избор на функции*



*Фиг. 2 Блок схема на Подалгоритъм Избор на опция от менюто*

* 1. Функция, с която е реализиран алгоритъма

int main()

* + 1. Входни данни на функцията

Няма входни данни.

* + 1. Изходни данни на функцията или данни, които се извеждат

Основната задача на тази функция е да координира основния цикъл на програмата чрез техниката на менютата. В зависимост от избраната опция, функцията извиква съответните оператори и функции, които изпълняват необходимите действия.

* 1. Функция, с която е реализиран алгоритъма

void Menu()

* + 1. Входни данни на функцията

Няма входни данни.

* + 1. Изходни данни на функцията или данни, които се извеждат

Функцията извежда на екрана графично представено главното меню. Съответно отново потребителят трябва да избере опция от дадените. Реализирана е валидация на избора в интервала от 1 до последната опция.

## Реализация на условие B

* 1. Алгоритъм Добавяне на студенти в група

Потребителят трябва да въведе данни за броят студенти, които реши да добави, описани в структурата Students. Направена е валидация за бройката на студентите, фак. Номер, ЕГН, броя дисциплини, оценките, пол и статут на студента.

* 1. Блок схема на алгоритъма (Фиг. 3)
  2. Функция, с която е реализиран алгоритъма

void addStudents(Students mass[], int N, int& countAddedStudents);

* + 1. Входни данни на функцията

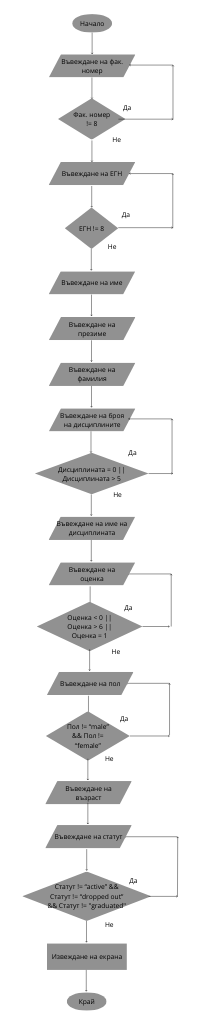
Students mass[] – масива със студентите от тип Students.

Int N – броя студенти, които потребителят иска да въведе.

Int& countAddedStudents – запазване по адрес на променливата на записаните вече студенти в системата.

* + 1. Изходни данни на функцията

Няма. Функцията запазва данните на студентите в масива.

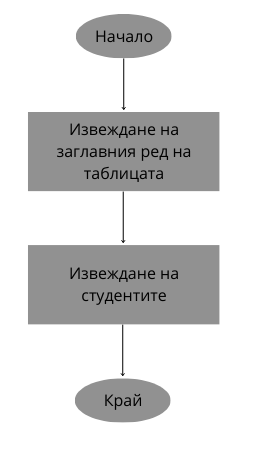
 *Фиг. 3 Блок схема на Алгоритъм Добавяне на студенти в група*

## Реализация на условие C

* 1. Алгоритъм Показване на всички студенти

След като вече са добавени данни за студентите, те се извеждат на екрана в табличен вид. Използван е подалгоритъм за графичното реализиране на таблицата.

void tableDesign();

* 1. Блок схема на алгоритъма (Фиг. 4)

*Фиг. 4 Блок схема на Алгоритъм Показване на всички студенти*

* 1. Функция, с която е реализиран алгоритъма

void print(Students mas[], int& countAddedStudents);

* + 1. Входни данни на функцията

Students mass[] – масива със студентите от тип Students.

Int& countAddedStudents – запазване по адрес на променливата на записаните вече студенти в системата.

* + 1. Изходни данни на функцията

Няма изходни данни. Функцията отпечатва студентите в табличен вид.

* 1. Помощни функции

void tableDesign(); - отпечатва заглавния ред на таблицата

## Реализация на условие D

* 1. Алгоритъм Търсене и извеждане на студенти с оценка по БП и с минимум 1 дисциплина с оценка Слаб 2

Потребителят трябва да въведе интервал (min-max), в който ще се търсят всички стуенти, които имат оценка по дисциплината БП в този интервал. След това с още един алгоритъм ще се извеждат студентите, които имат оценка Слаб 2 по минимум една дисциплина.

* 1. Блок схема на алгоритъма (Фиг. 5, Фиг. 6)
  2. Функция, с която е реализиран алгоритъма

void studentsInInterval(Students mas[], int& countAddedStudents);

void studentsWithFailDiscipline(Students mas[], int& countAddedStudents);

* + 1. Входни данни на функцията

Students mass[] – масива със студентите от тип Students.

Int& countAddedStudents – запазване по адрес на променливата на записаните вече студенти в системата.

firstInterval – Въвеждане на първата оценка от интервала, в който ще търсим.

secondInterval - Въвеждане на втората оценка от интервала, в който ще търсим.

* + 1. Изходни данни на функцията

Няма изходни данни. Функцията извежда първо студентите с оценки по БП в дадения интервал (или се появява съобщение, че няма такива студенти), след което се извеждат и студентите с поне една оценка Слаб 2 по някоя от дисциплините му(съответно – се появява съобщение, че няма такива студенти, ако условието не е изпълнено).

* 1. Помощни функции

printWithoutTable(masCopy, count); - отпечатва студентите, които отговарят на двете условия(оценка в дадения интервал по БП/оценка Слаб 2 по минимум ч дисциплина).

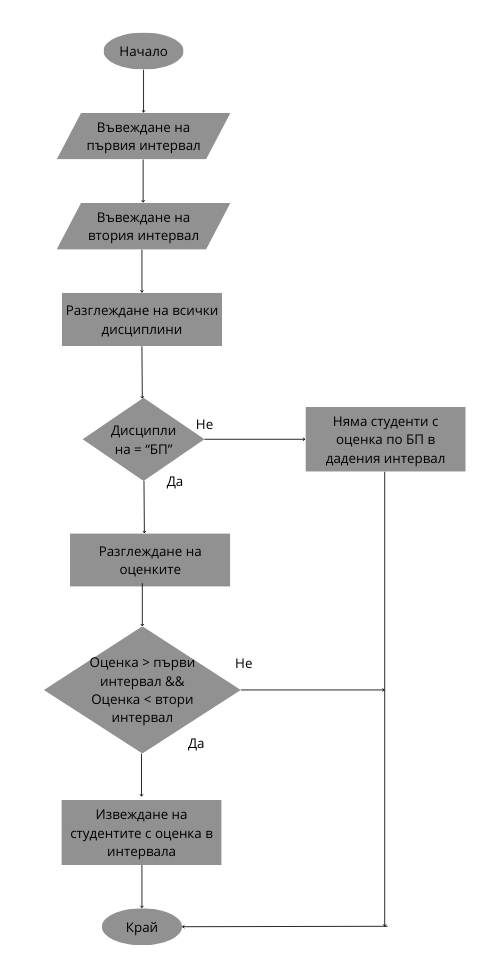
* + 1. Входни данни на функцията

masCopy – копие на масива със студентите.

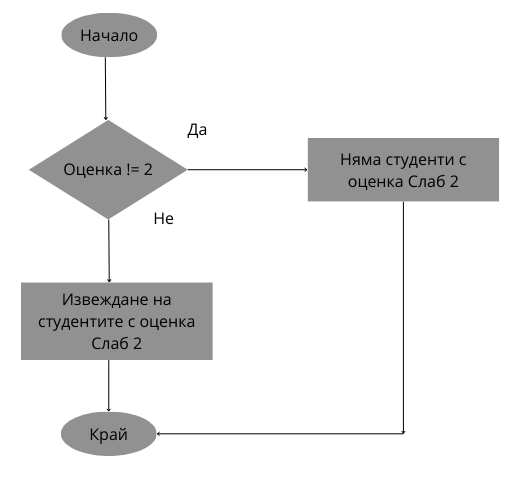
count – брояч, който в реално време да отчита колко студенти отговарят на условията.

* + 1. Изходни данни на функцията

Няма изходни данни. Функцията извежда всички студенти, които отговарят на двете условия.

*Фиг. 5 Блок схема на Алгоритъм Търсене и извеждане на студенти с оценка по БП*

*Фиг. 6 Блок схема на Алгоритъм Търсене и извеждане на студенти с минимум 1 дисциплина с оценка Слаб 2*



## Реализация на условие E

* 1. Алгоритъм Сортиране и извеждане на студенти

Функцията автоматично сортира всички записани студенти по малко име във възходящ ред (азбучен ред), след което извежда цялата информация за всеки отделен студент от групата. Извеждането се осъществява чрез допълнителна помощна функция printSorting(masCopy, countAddedStudents);. Сортирането се извършва по метода на Мехурчето.

* 1. Блок схема на алгоритъма (Фиг. 7)
  2. Функция, с която е реализиран алгоритъма

void sorting(Students mas[], int& countAddedStudents);

* + 1. Входни данни на функцията

Students mass[] – масива със студентите от тип Students.

Int& countAddedStudents – запазване по адрес на променливата на записаните вече студенти в системата.

* + 1. Изходни данни на функцията

Няма изходни данни. Функцията сортира студентите по малко име по азбучен ред.

* 1. Помощни функции

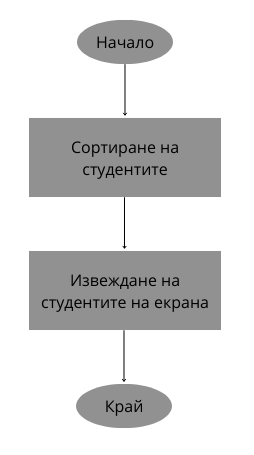
printSorting(masCopy, countAddedStudents); - отпечатва сортираните студенти.

* + 1. Входни данни на функцията

masCopy – копие на масива със студентите.

countAddedStudents – броя на студентите в групата.

* + 1. Изходни данни на функцията

Няма изходни данни. Функцията сортира студентите по малко име по азбучен ред.

*Фиг. 7 Блок схема на Алгоритъм Сортиране и извеждане на студенти*

## Реализация на условие F

* 1. Алгоритъм Записване на студенти във файл

Данните със студентите се записват в текстов файл чрез избор от главното меню. Данните за студентите се съхраняват всяка на отделен ред. Записването се реализира чрез функцията void SaveStudentsToFile(Students mas[], int& countAddedStudents);.

* 1. Блок схема на алгоритъма (Фиг. 8)
  2. Функция, с която е реализиран алгоритъма

void SaveStudentsToFile(Students mas[], int& countAddedStudents);

* + 1. Входни данни на функцията

Students mass[] – масива със студентите от тип Students.

Int& countAddedStudents – запазване по адрес на променливата на записаните вече студенти в системата.

* + 1. Изходни данни на функцията

Няма изходни данни. Функцията записва всички данни на студентите в текстов файл. Ако файлът не бъде намерен или не работи (не може да бъде отворен) ще изведе съобщение за грешка и програмата ще спре.

* 1. Алгоритъм Въвеждане на студенти от файл

Данните със студентите се прочитат от текстов файл чрез избор от главното меню. Данните за студентите се четат ред по ред. Четенето се реализира чрез функцията void ReadStudentsFromFile(Students mas[], int& countAddedStudents);

* 1. Блок схема на алгоритъма (Фиг. 9)
  2. Функция, с която е реализиран алгоритъма

void ReadStudentsFromFile(Students mas[], int& countAddedStudents);

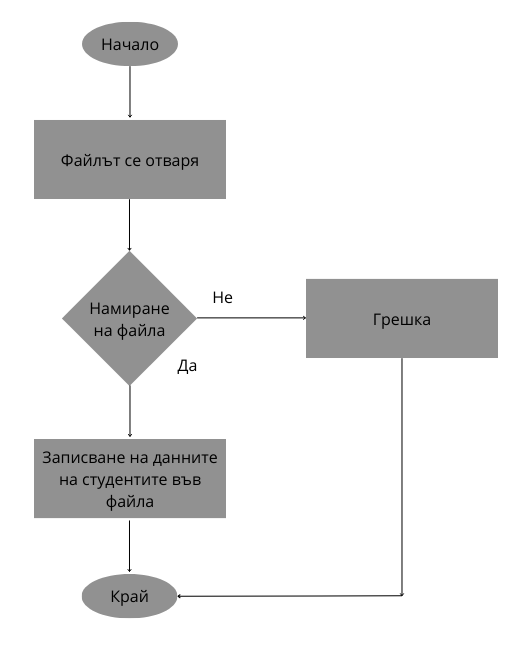
* + 1. Входни данни на функцията

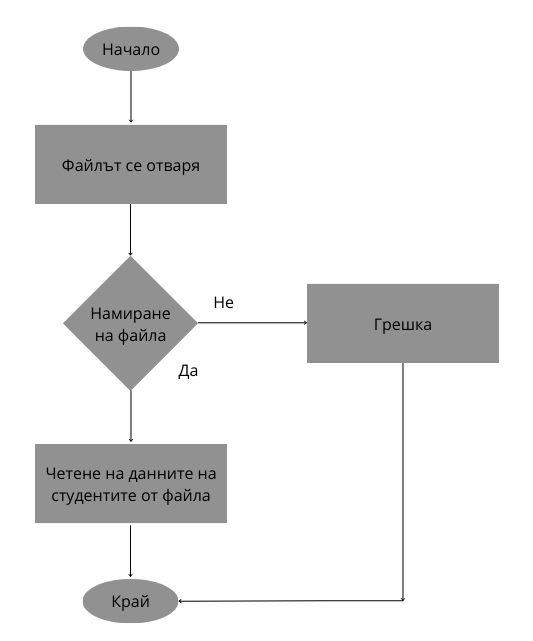
Students mass[] – масива със студентите от тип Students.

Int& countAddedStudents – запазване по адрес на променливата на записаните вече студенти в системата.

* + 1. Изходни данни на функцията

Няма изходни данни. Функцията чете всички данни на студентите от текстов файл. Ако файлът не бъде намерен или не работи (не може да бъде отворен) ще изведе съобщение за грешка и програмата ще спре.

*Фиг. 8 Блок схема на Алгоритъм Записване на студенти във файл*

*Фиг. 9 Блок схема на Алгоритъм Въвеждане на студенти от файл*

## Реализация на условие G – допълнение първо

* 1. Алгоритъм Създаване на подменю към основното

Алгоритъмът за създаване на подменю към основното с функции предоставя на потребителя възможност да избира допълнителни функционалности чрез опции. Главният алгоритъм извиква подпрограма за изграждане на менюто и проверка на валидността на избраната опция. Главният алгоритъм се реализира чрез функцията int main() в един от случаите в цикъла switch, докато подалгоритъмът за менюто се осъществява чрез функцията void SubMenu().

* 1. Блок схема на алгоритъма (Фиг. 10)
  2. Функция, с която е реализиран алгоритъма

int main();

* + 1. Входни данни на функцията

Няма входни данни.

* + 1. Изходни данни на функцията

Подменюто служи за структуриране и групиране на функционалността, улеснявайки навигацията и поддържайки чист и прегледен дизайн. Позволява организиране на опции, оптимизиране на екранното пространство и предоставят контекстуална видимост.

* 1. Помощни функции

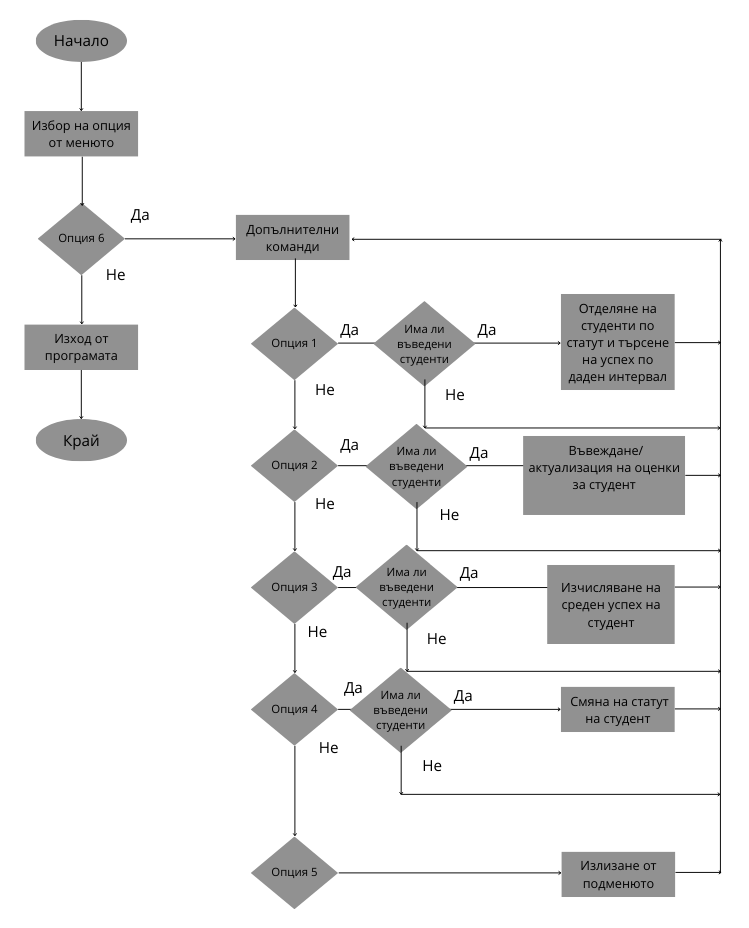
void SubMenu(); - графично представяне на подменюто.

* + 1. Входни данни на функцията

Няма входни данни.

* + 1. Изходни данни на функцията

Функцията извежда на екрана графично представено подменюто. Съответно отново потребителят трябва да избере опция от дадените. Реализирана е валидация на избора в интервала от 1 до последната опция.

*Фиг. 10 Блок схема на Алгоритъм Създаване на подменю към основното* 

## Реализация на условие G (a,b) – допълнение първо

* 1. Алгоритъм Отделяне на студенти по статут и търсене на успех по даден интервал

Потребителят трябва да въведе един от трите валидни статута, след което на екрана ще се изведат всички студенти с този статут, сортирани по факултетен номер. След това потребителят ще има възможност да напише интервал (min-max), в който да се търсят всички студенти със среден успех в този интервал. При извеждането им на екрана ще бъдат сортирани по ЕГН.

* 1. Блок схема на алгоритъма (Фиг. 11, Фиг. 12)
  2. Функция, с която е реализиран алгоритъма

void separateStudents(Students mas[], int size, int& countAddedStudents);

* + 1. Входни данни на функцията

Students mass[] – масива със студентите от тип Students.

Int size – променлива с начална стойност 0, в която ще се отчита броя на студентите, които ще се извеждат на екрана.

Int& countAddedStudents – запазване по адрес на променливата на записаните вече студенти в системата.

* + 1. Изходни данни на функцията

Няма изходни данни. Функцията съдържа валидация, която проверява за правилно изписания статут. Съответно се прави и проверка дали изобщо са запазени студенти със съответния статут, въведен от потребителя. Ако няма – на екрана се извежда подходящо съобщение, което ни уведомява, че няма студенти с този статут. Ако съществуват студенти със дадения статут, се сортират в помощна функция по факултетен номер и се изписват на нашия екран.

* 1. Помощни функции

sortByFakNomer(masCopy, size); - сортира студентите по факултетен номер

* + 1. Входни данни на функцията

masCopy – копие на масива със студентите.

size – броя на студентите, които са с дадения статут.

* + 1. Изходни данни на функцията

Няма изходни данни. Функцията извежда на екрана сортираните студенти по факултетен номер с всичките им данни.

* 1. Функция, с която е реализиран алгоритъма

void separateStudentsByScore(Students mas[], int size, int& countAddedStudents);

* + 1. Входни данни на функцията

Students mass[] – масива със студентите от тип Students.

Int size – променлива с начална стойност 0, в която ще се отчита броя на студентите, които ще се извеждат на екрана.

Int& countAddedStudents – запазване по адрес на променливата на записаните вече студенти в системата.

* + 1. Изходни данни на функцията

Няма изходни данни. Функцията дава възможност на потребителя да отдели всички студенти със среден успех в интервал, въведен от него. Ако няма студенти, които да попадат в този интервал, на екрана се изписва подходящо съобщение. Ако има, се извеждат на екрана всички данни на дадените студенти, сортирани по ЕГН.

* 1. Помощни функции

sortByEGN(masCopy, size); - графично представяне на подменюто.

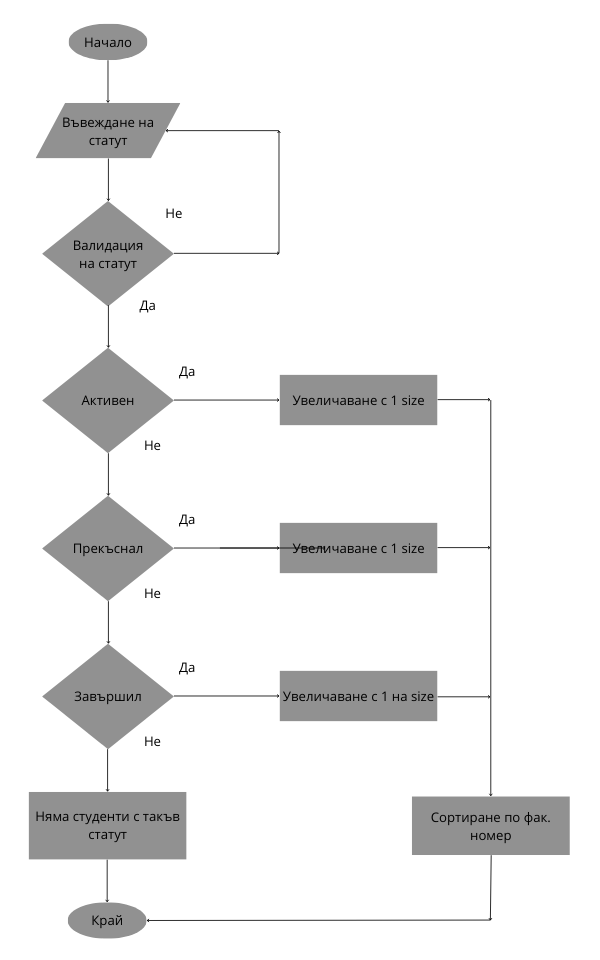
* + 1. Входни данни на функцията

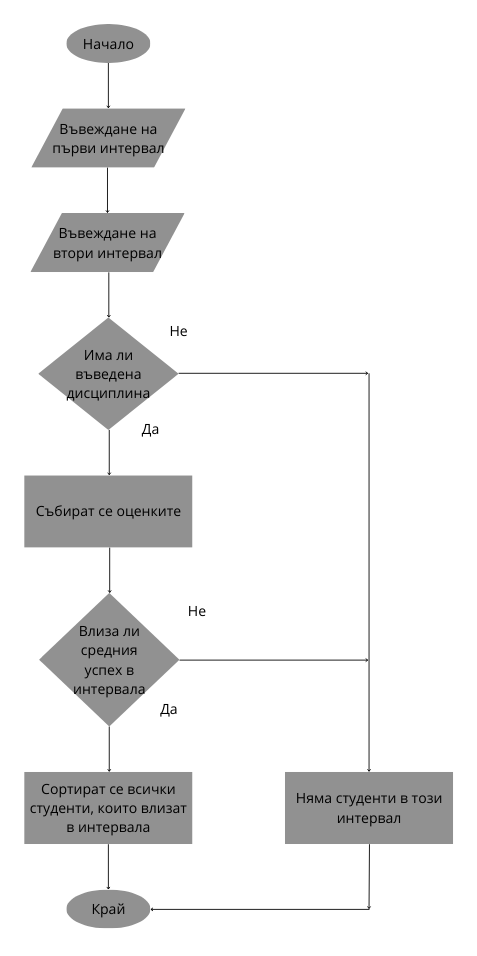
masCopy – копие на масива със студентите.

size – броя на студентите, които са с дадения статут.

* + 1. Изходни данни на функцията

Няма изходни данни. Функцията извежда на екрана сортираните студенти по ЕГН с всичките им данни.

*Фиг. 11 Блок схема на Алгоритъм Отделяне на студенти по статут*

*Фиг. 12 Блок схема на Алгоритъм Търсене на успех по даден интервал*

## Реализация на условие H (a,b) – допълнение второ

* 1. Алгоритъм Въвеждане/Актуализация на оценки и среден успех за студент

Чрез алгоритъма за въвеждане и актуализация на оценки на студент, потребителят има възможността да въвежда оценки по дадена дисциплина (оценката преди това е била 0) или да актуализира вече съществуваща оценка. Като, за да се случи това, трябва да въведе факултетен номер на студента, чиято оценка ще променя. С алгоритъма за среден успех автоматично се пресмята средния успех на дадения студент(дори след актуализация на оценка) и се извеждат данните на студента на екрана. Отново по факултетен номер се търси студента, на който потребителят иска да види средния успех.

* 1. Блок схема на алгоритъма (Фиг. 13, Фиг. 14)
  2. Функция, с която е реализиран алгоритъма

void newGrade(Students mas[], int& countAddedStudents);

* + 1. Входни данни на функцията

Students mass[] – масива със студентите от тип Students.

Int& countAddedStudents – запазване по адрес на променливата на записаните вече студенти в системата.

* + 1. Изходни данни на функцията

Няма изходни данни. Функцията съдържа валидация, която проверява дали съществува студент с написания от потребителя факултетен номер. Ако не съществува такъв – се изписва подходящо съобщение. Ако съществува, се проверява статутът на студента – ако е активен, се проверява дали оценката е 0(Няма оценка) или от 2-6. При оценка 0 автоматично се въвежда новата оценка. При оценка варираща от 2-6 първо се изисква запитване дали потребителят е сигурен, че иска да промени оценката. Ако статутът на студента е прекъснал или завършил програмата не ни позволява да променяме неговите оценки – това разбираме като на екрана се изписва подходящо съобщение.

* 1. Функция, с която е реализиран алгоритъма

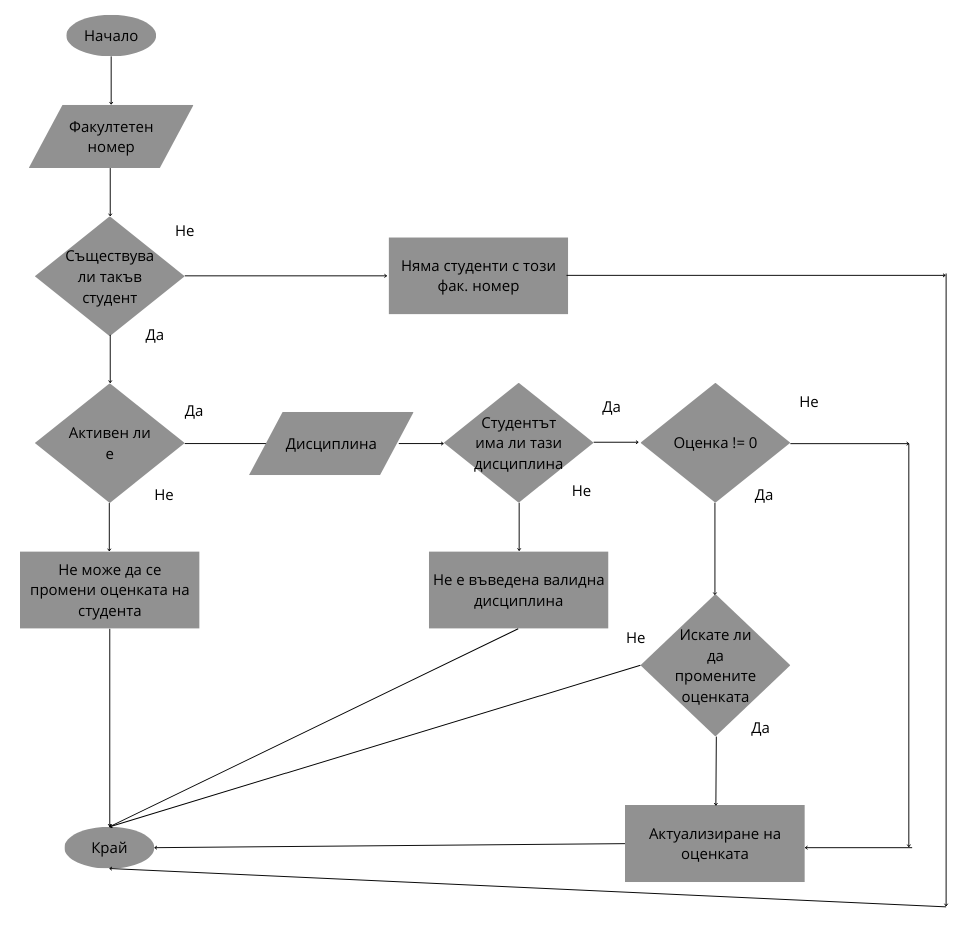
void avarageGrades(Students mas[], int& countAddedStudents);

* + 1. Входни данни на функцията

Students mass[] – масива със студентите от тип Students.

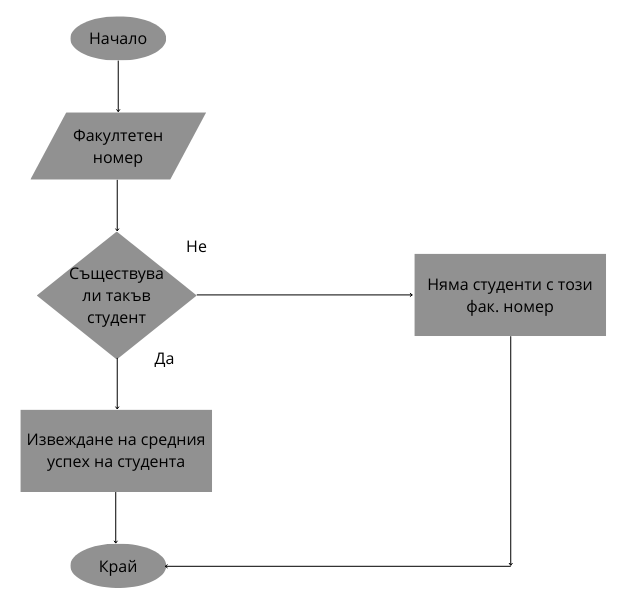
Int& countAddedStudents – запазване по адрес на променливата на записаните вече студенти в системата.

* + 1. Изходни данни на функцията

Няма изходни данни. Няма изходни данни. Функцията съдържа валидация, която проверява дали съществува студент с написания от потребителя факултетен номер. Ако не съществува такъв – се изписва подходящо съобщение. Ако съществува, автоматично се пресмята средният успех на студента и се извежда на екрана.

*Фиг. 13 Блок схема на Алгоритъм Въвеждане/Актуализация на оценки*

*Фиг. 14 Блок схема на Алгоритъм Извеждане на среден успех за студент*



## Реализация на условие I – допълнение второ

* 1. Алгоритъм Смяна на статут на студент

Чрез алгоритъма за смяна на статут на студент, потребителят има възможността да актуализира статута на студент. Като, за да се случи това, трябва да въведе факултетен номер на студента, чийто статут ще променя.

* 1. Блок схема на алгоритъма (Фиг.15)
  2. Функция, с която е реализиран алгоритъма

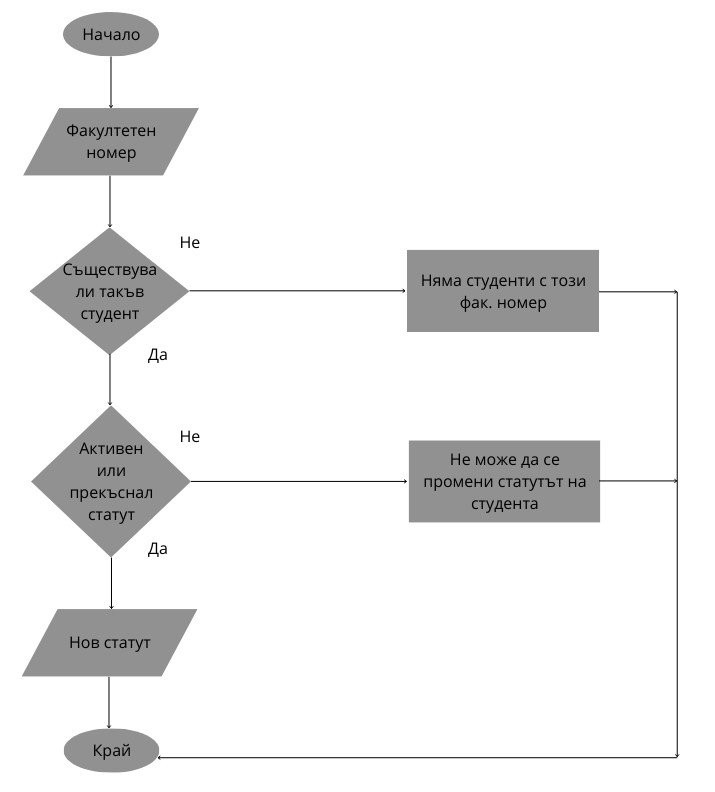
void newStatut(Students mas[], int& countAddedStudents);

* + 1. Входни данни на функцията

Students mass[] – масива със студентите от тип Students.

Int& countAddedStudents – запазване по адрес на променливата на записаните вече студенти в системата.

* + 1. Изходни данни на функцията

Няма изходни данни. Функцията съдържа валидация, която проверява дали съществува студент с написания от потребителя факултетен номер. Ако не съществува такъв – се изписва подходящо съобщение. Ако съществува, се проверява статутът на студента – ако е активен или прекъснал се променя статутът му с написания от потребителя. Ако е вече завършил не се допуска промяна на неговия статут и на екрана се изписва подходящо съобщение.

*Фиг. 15 Блок схема на Алгоритъм Смяна на статут на студент*

## Реализация на условие J – допълнение трето

* 1. Алгоритъм Автоматично попълване на данни от файл

Данните със студентите се прочитат автоматично от текстов файл със стартирането на програмата. Данните за студентите се четат ред по ред.

* 1. Блок схема на алгоритъма (Фиг. 16)
  2. Функция, с която е реализиран алгоритъма

void ReadStudentsFromFile(Students mas[], int& countAddedStudents);

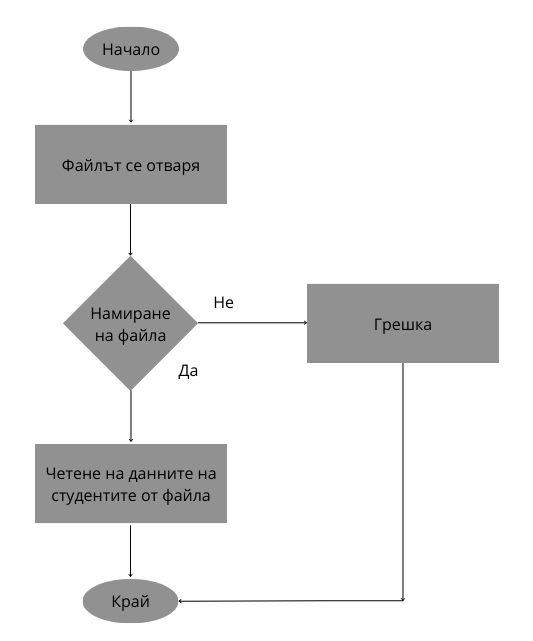
* + 1. Входни данни на функцията

Students mass[] – масива със студентите от тип Students.

Int& countAddedStudents – запазване по адрес на променливата на записаните вече студенти в системата.

* + 1. Изходни данни на функцията

Няма изходни данни. Функцията чете всички данни на студентите от текстов файл. Ако файлът не бъде намерен или не работи (не може да бъде отворен) ще изведе съобщение за грешка и програмата ще спре.



*Фиг. 16 Блок схема на Алгоритъм Смяна на статут на студент*

# Примерно действие на програмата

## Условие А

Входни данни:

Въвеждане на опция от главното меню. При въвеждане на опция са възможни следните ситуации:

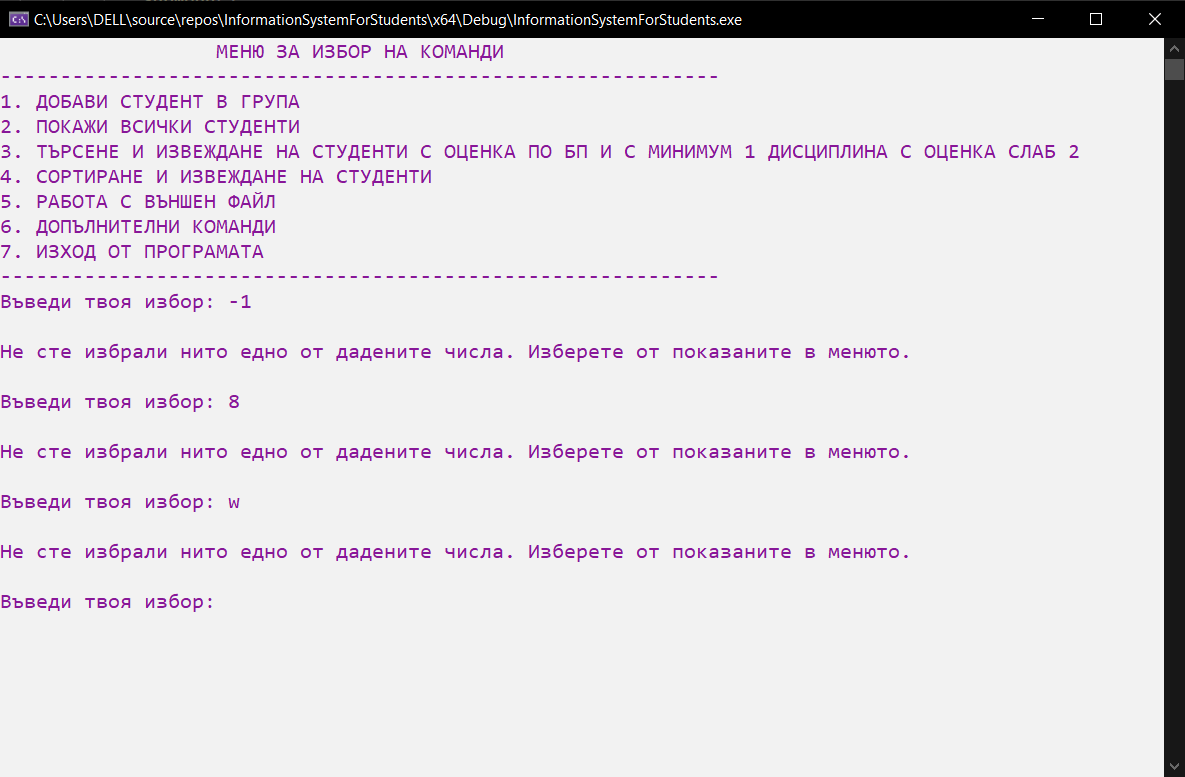
А) Въвеждане на опция < 1 – в този случай програмата ще поиска от потребителя да въведе отново опция, има предвидена валидация на въвеждането.

Б) Въвеждане на опция > последният номер на опция – в този случай, програмата ще поиска от потребителя да въведе отново опция, има предвидена валидация на въвеждането.

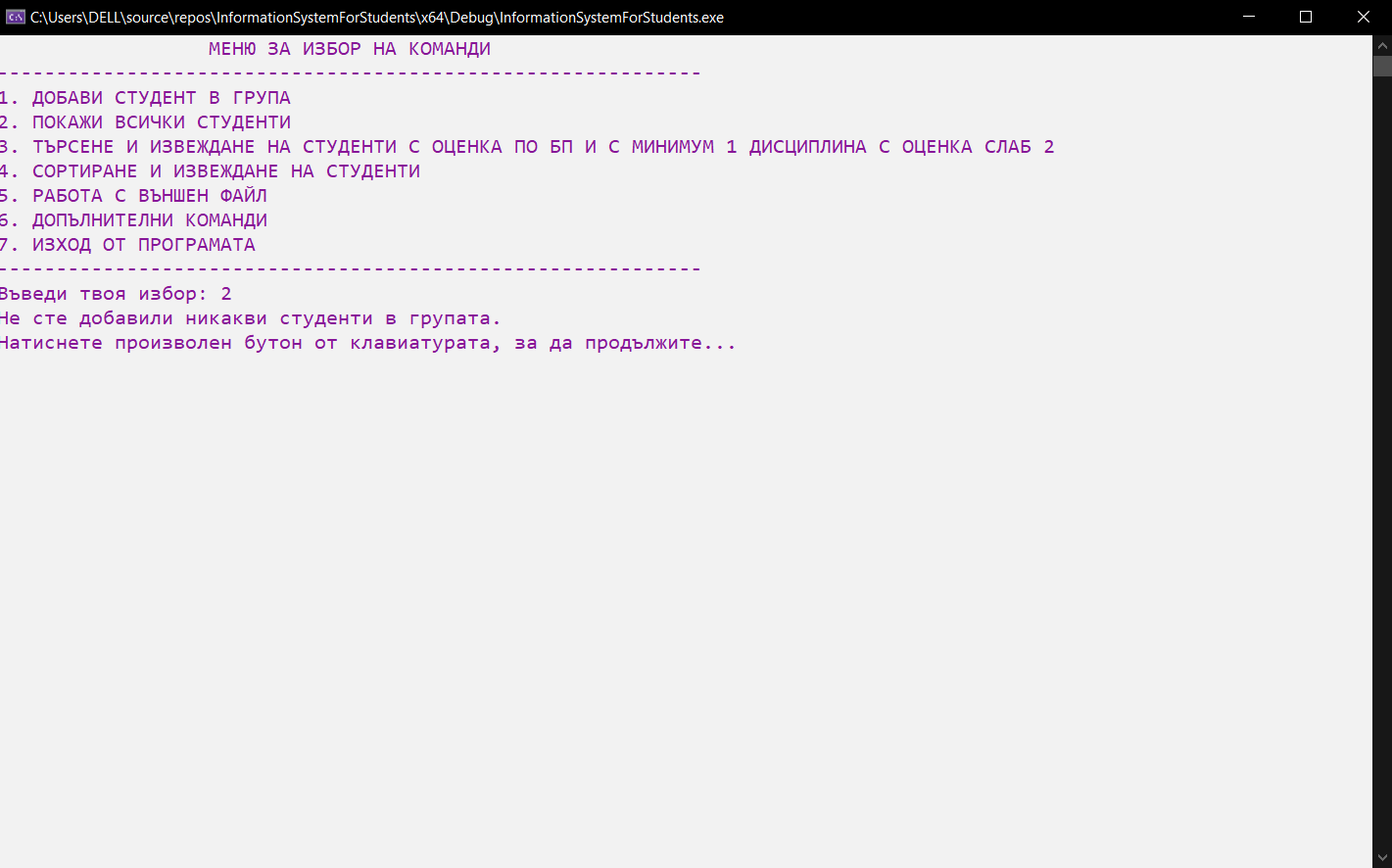
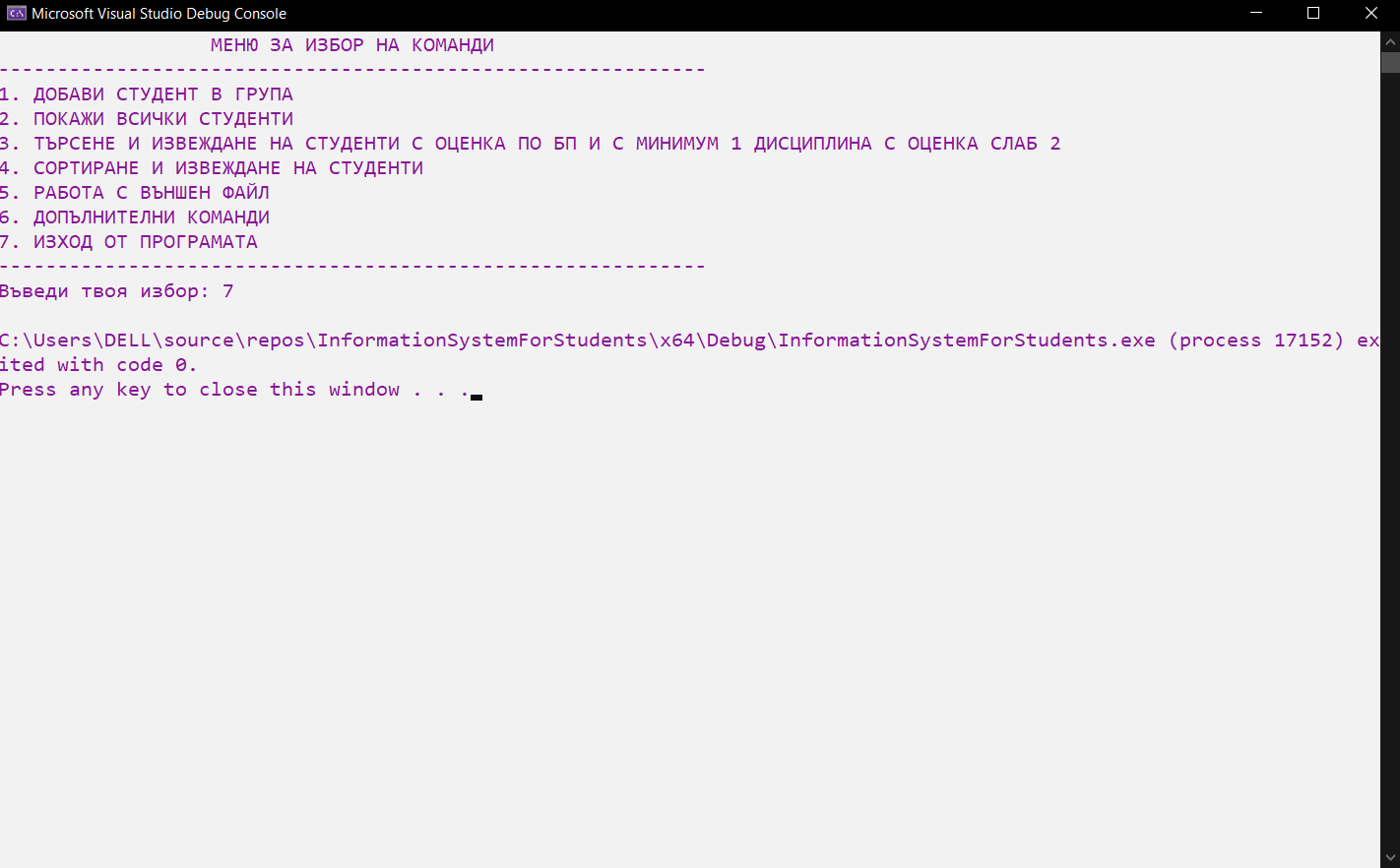
В) Въвеждане на символ, различен от цифра - в този случай, програмата ще поиска от потребителя да въведе отново опция, има предвидена валидация на въвеждането.

Г) Въвеждане на опция в интервала [1 – предпоследната опция] – в този случай програмата изпълнява действията, заложени за изпълнение при дадената опция.

Д) Въвеждане на последната опция – в този случай това е опция „Изход“, което води до прекратяване на изпълнението на цялата програма.

Снимка на изгледа с примерни изходни данни

*Фиг. 17 Въвеждане на невалидна опция А), Б). В)*

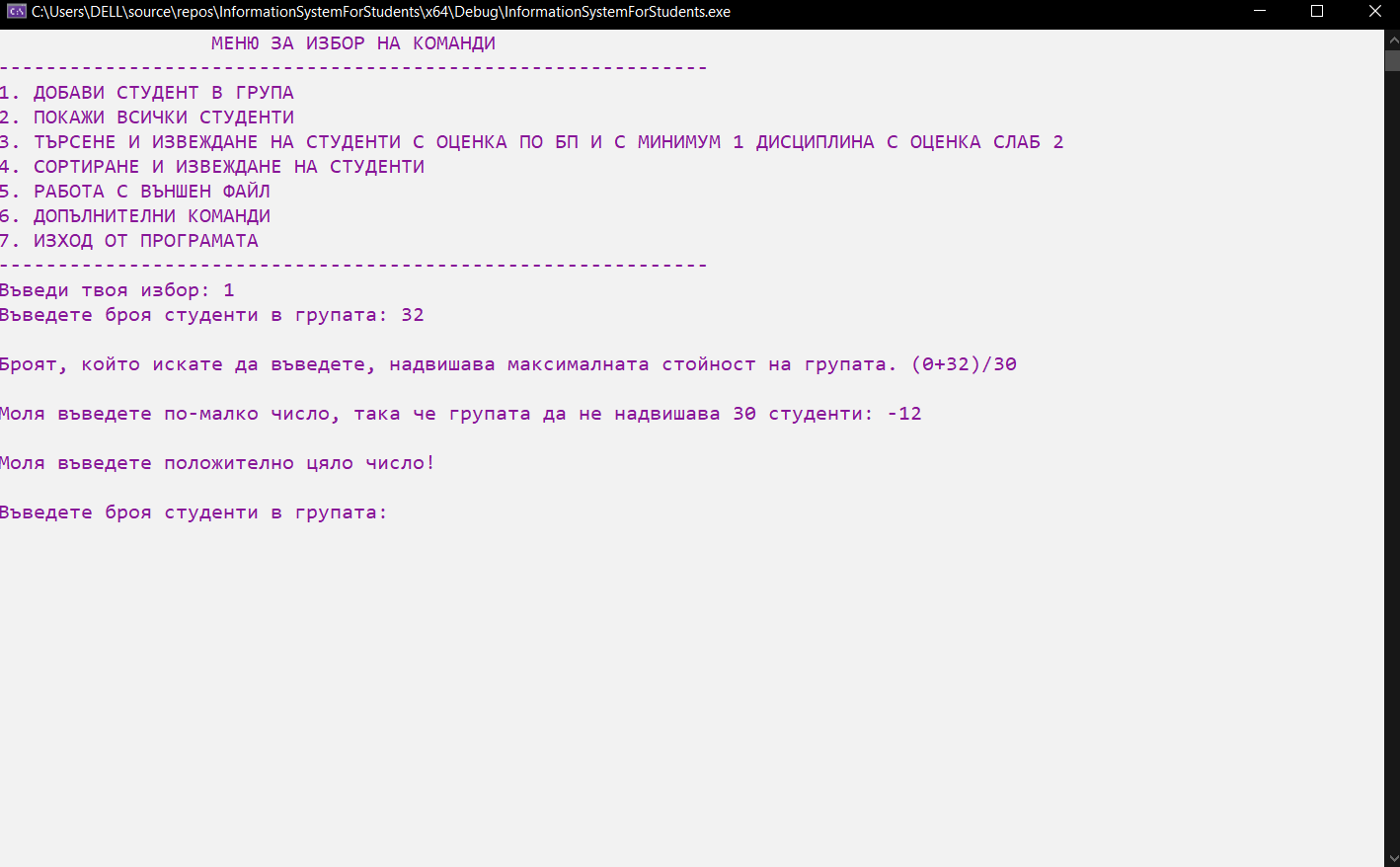
*Фиг. 18 Въвеждане на валидна опция Г)*

*Фиг. 19 Въвеждане на последната опция от менюто – Изход – случай Д)*

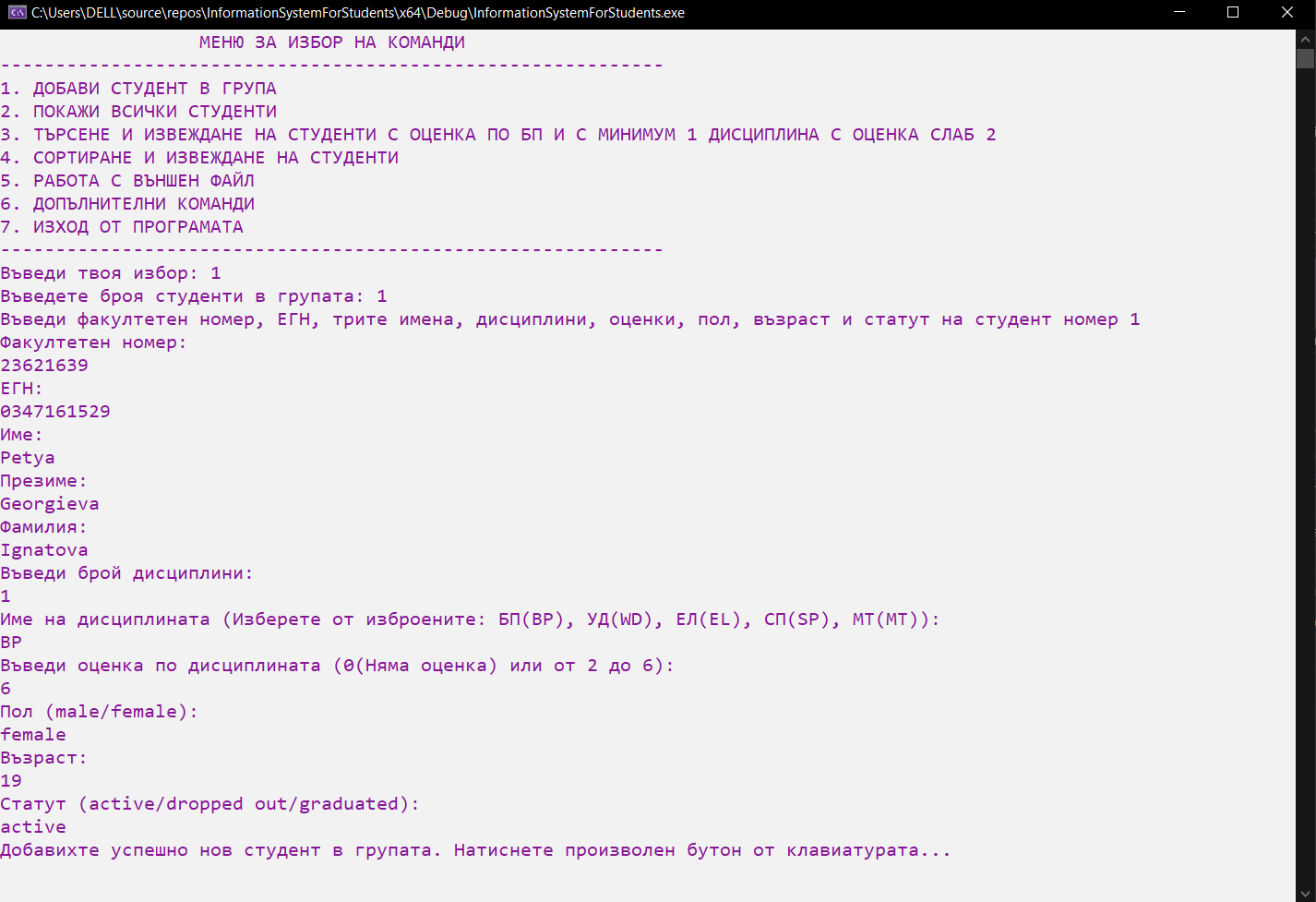
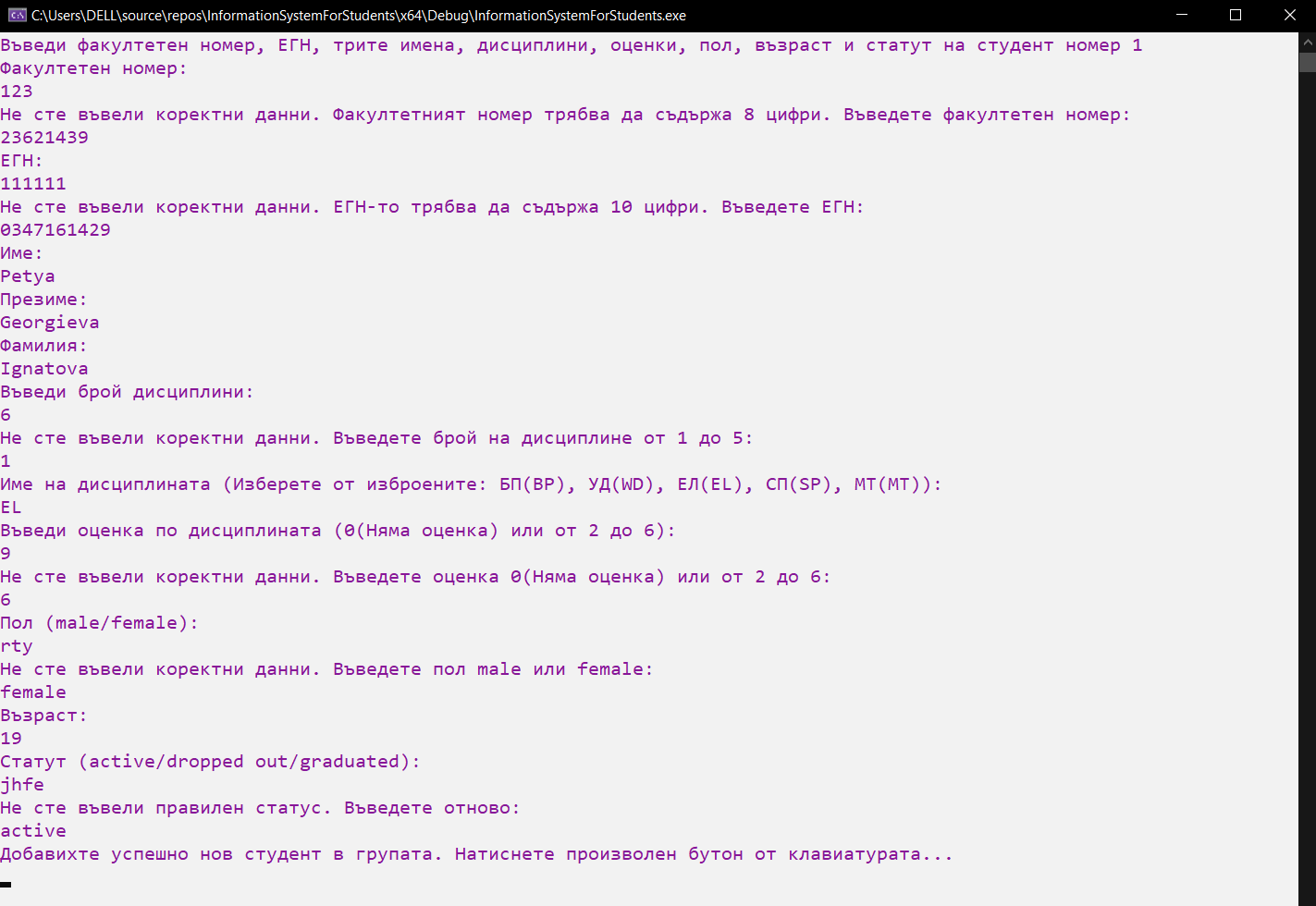
## Условие B

Входни данни:

Потребителят трябва да въведе първо броя студенти, които ще добавя в групата. След това последователно се въвеждат останалите данни за студента. Направена е валидация за бройката на студентите, фак. Номер, ЕГН, броя дисциплини, оценките, пол и статут на студента.

Снимка на изгледа с примерни изходни данни

*Фиг. 20 Въвеждане на невалидни стойности за бройката на студентите в групата*

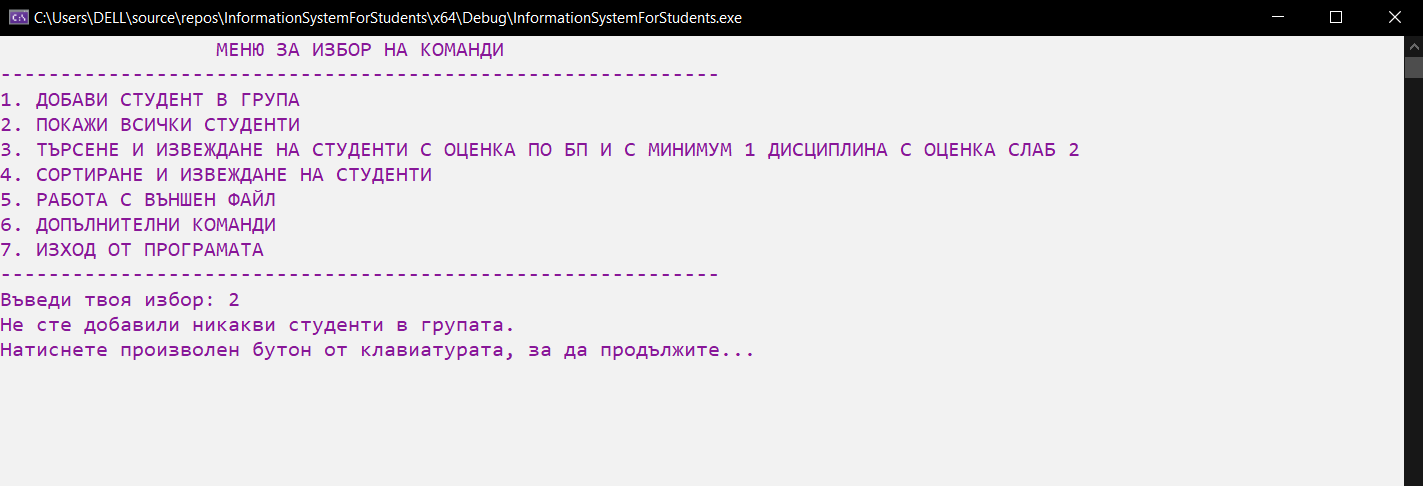
*Фиг. 21 Въвеждане на коректни данни за студента при добавяне на студент*

*Фиг. 22 Въвеждане на невалидни данни за студента при добавяне на студент*

## Условие C

Входни данни:

Извеждането на студенти на екрана (отпечатване) не изисква въвеждане на входни данни. Предвидена е проверка за празен масив (списък) със студенти и извеждане на подходящо съобщение. При наличие на стойности се отпечатват студентите в таблица.

Снимка на изгледа с примерни изходни данни

*Фиг. 23 Въвеждане на опция, когато няма записани студенти*

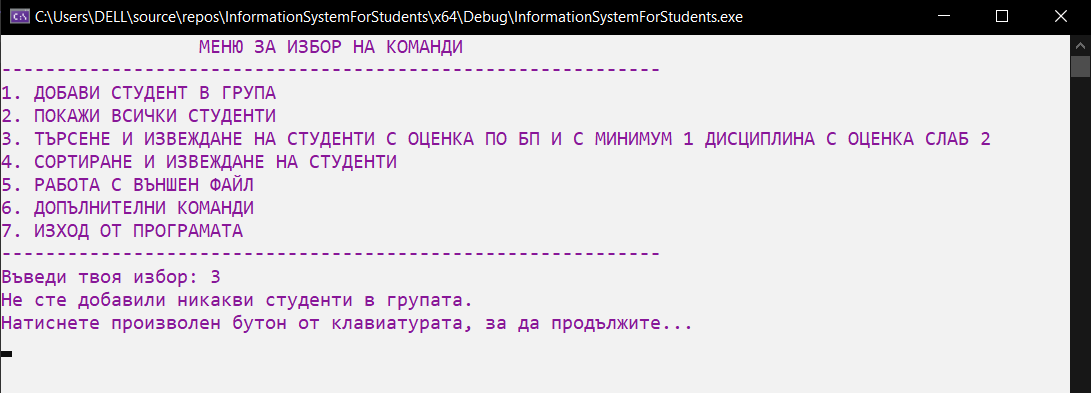
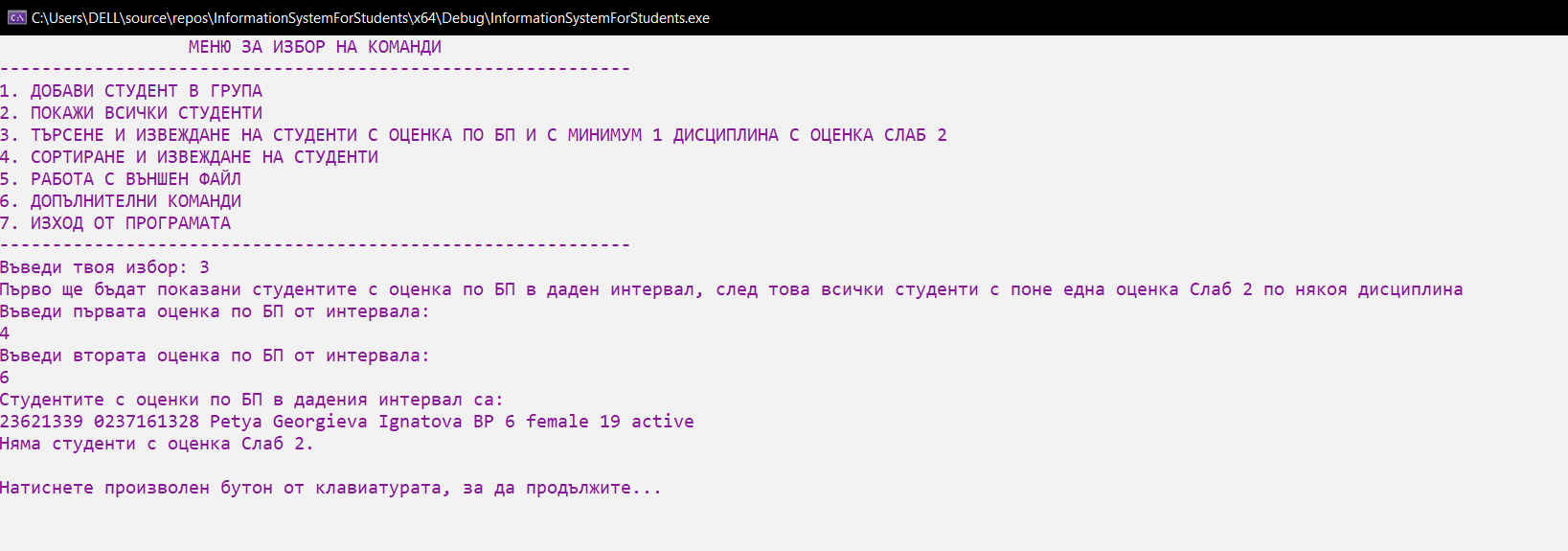
*Фиг. 24 Извеждане на данните за студентите в таблица*

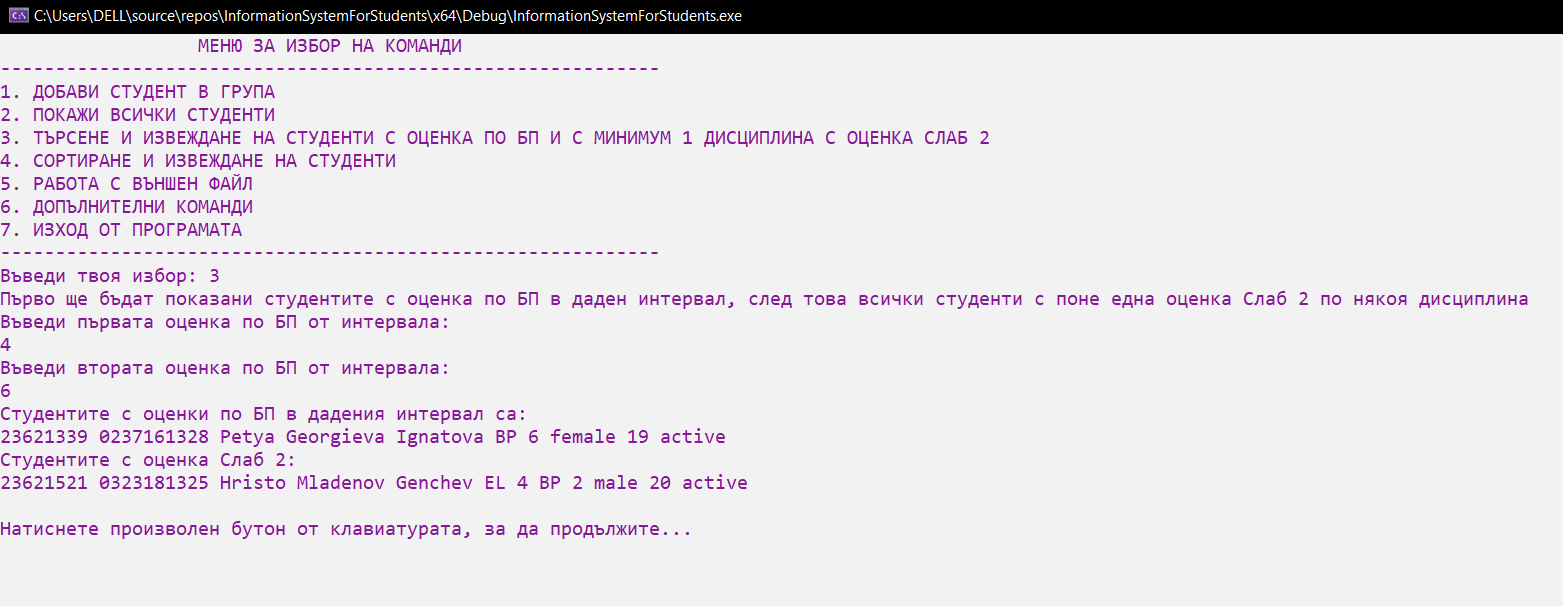
## Условие D

Входни данни:

Потребителят трябва да въведе интервал (min-max), в който ще се търсят всички стуенти, които имат оценка по дисциплината БП в този интервал. След това на екрана ще се изведат студентите, които имат оценка Слаб 2 по минимум една дисциплина.

Снимка на изгледа с примерни изходни данни

*Фиг. 25 Въвеждане на опция, когато няма записани студенти*

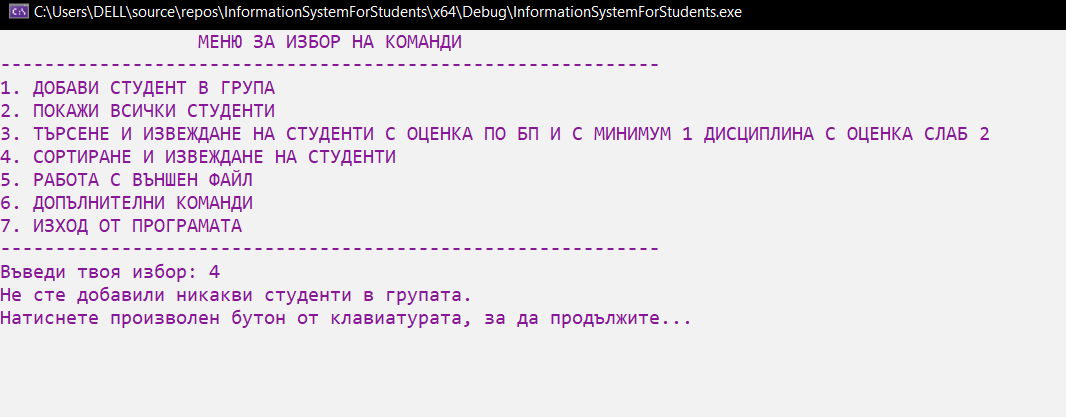
*Фиг. 26 Извеждане на студентите в дадения интервал*

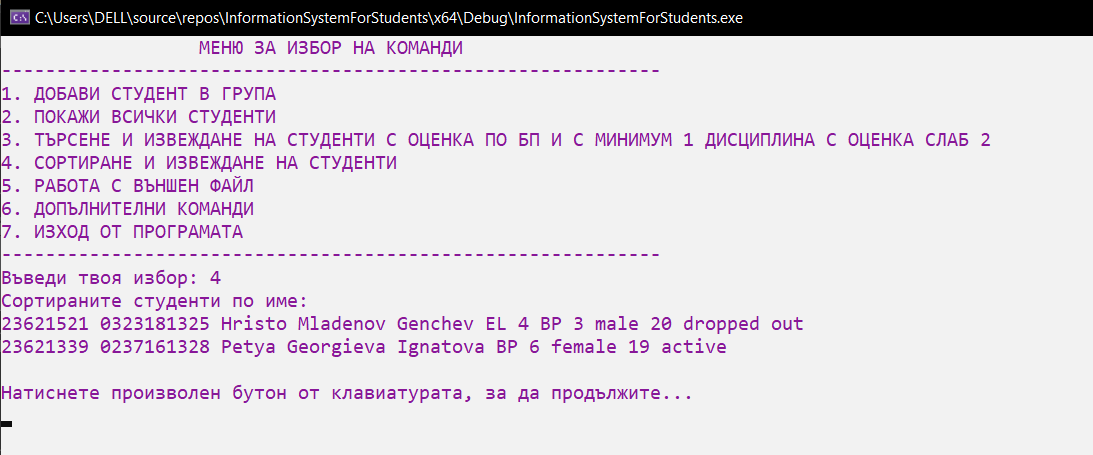
*Фиг. 27 Извеждане на студентите със Слаб 2*

## Условие E

Входни данни:

Функцията автоматично сортира всички записани студенти по малко име във възходящ ред (азбучен ред), след което извежда цялата информация за всеки отделен студент от групата.

Снимка на изгледа с примерни изходни данни

*Фиг. 28 Въвеждане на опция, когато няма записани студенти*

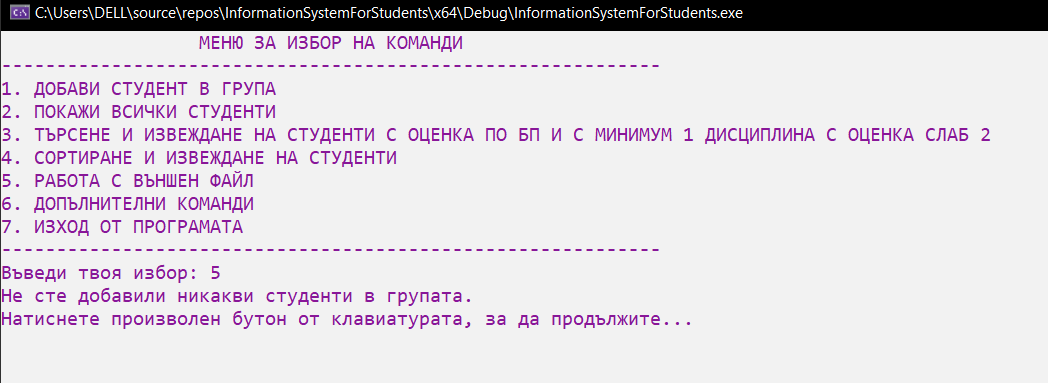
*Фиг. 29 Извеждане на студентите сортирани по малко име*

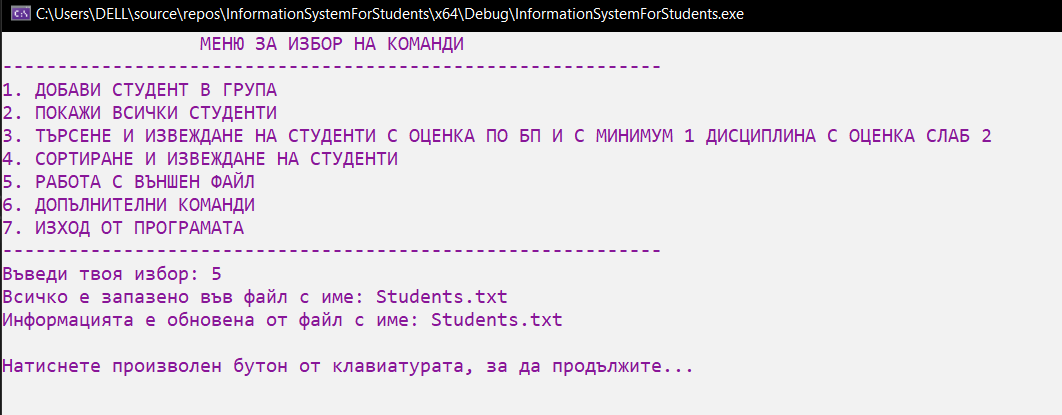
## Условие F

Входни данни:

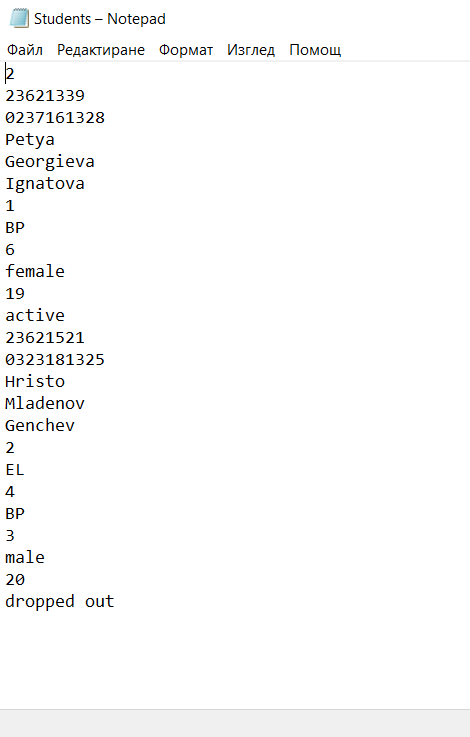
Данните със студентите се записват в текстов файл чрез избор от главното меню. На екрана се извежда подходящо съобщение, че всичко се е запазило успешно.

Снимка на изгледа с примерни изходни данни

*Фиг. 30 Въвеждане на опция, когато няма записани студенти*



*Фиг. 31 Успешно записване на данните на студентите във файл*



*Фиг. 32 Данните в текстовия файл*

## Условие G - допълнение първо

Входни данни:

Когато потребителят избере опция 6 на екрана се извежда подменю с допълнителни опции, от които потребителят да избира. При въвеждане на опция са възможни следните ситуации:

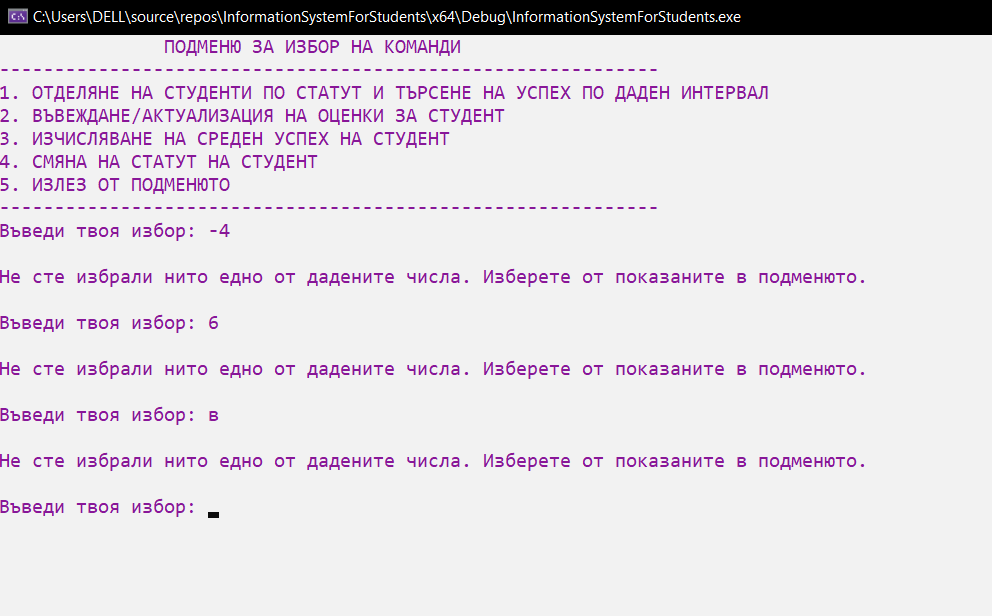
А) Въвеждане на опция < 1 – в този случай програмата ще поиска от потребителя да въведе отново опция, има предвидена валидация на въвеждането.

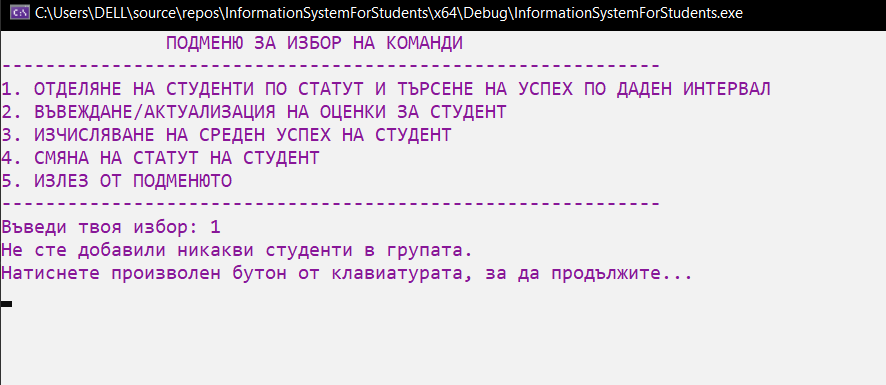
Б) Въвеждане на опция > последният номер на опция – в този случай, програмата ще поиска от потребителя да въведе отново опция, има предвидена валидация на въвеждането.

В) Въвеждане на символ, различен от цифра - в този случай, програмата ще поиска от потребителя да въведе отново опция, има предвидена валидация на въвеждането.

Г) Въвеждане на опция в интервала [1 – предпоследната опция] – в този случай програмата изпълнява действията, заложени за изпълнение при дадената опция.

Д) Въвеждане на последната опция – в този случай това е опция „Изход“, която ни връща към главното меню.

Снимка на изгледа с примерни изходни данни

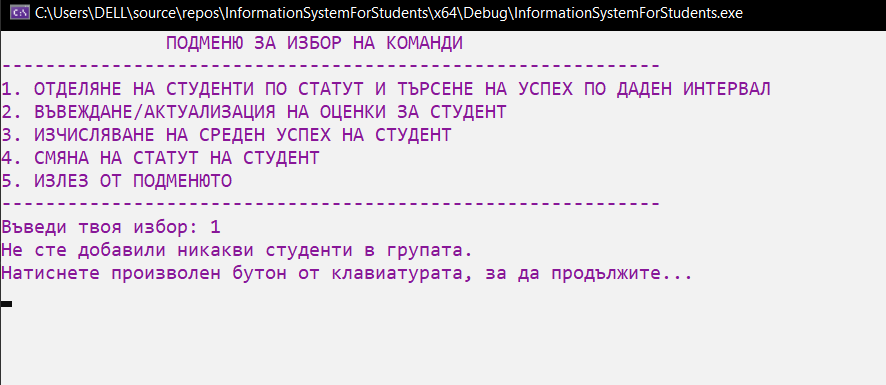
*Фиг. 33 Въвеждане на невалидна опция А), Б), В)*

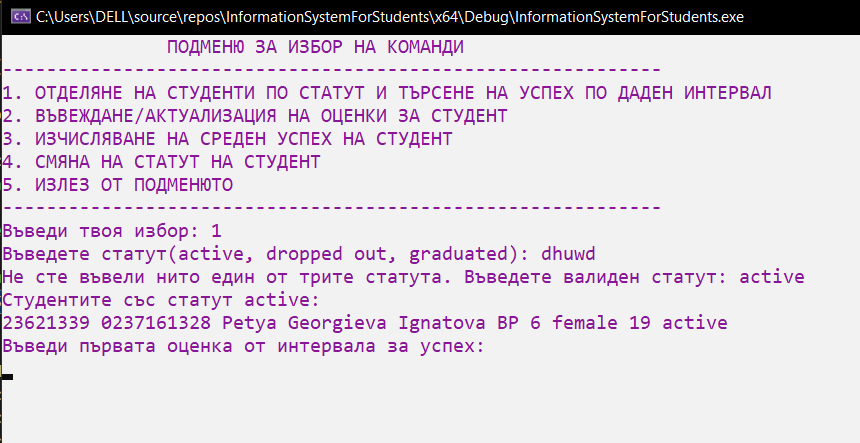
*Фиг. 34 Въвеждане на валидна опция без добавени студенти*

## Условие G (a,b) - допълнение първо

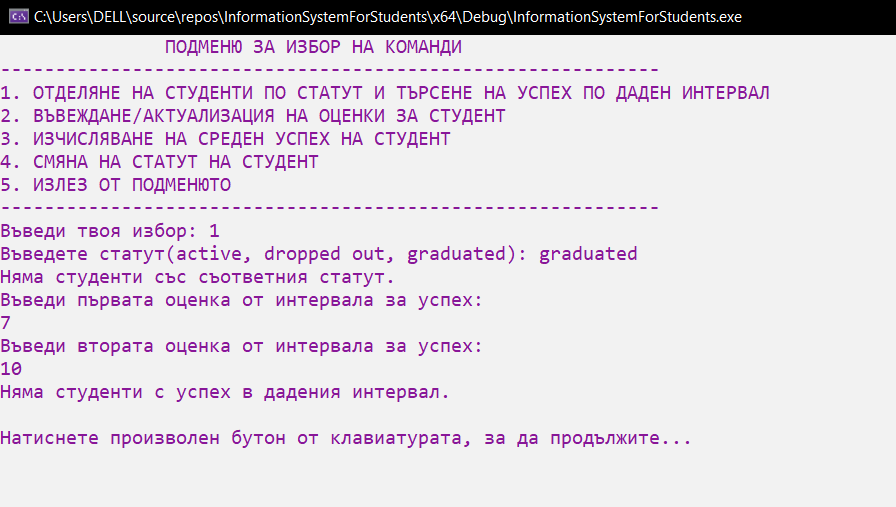
Входни данни:

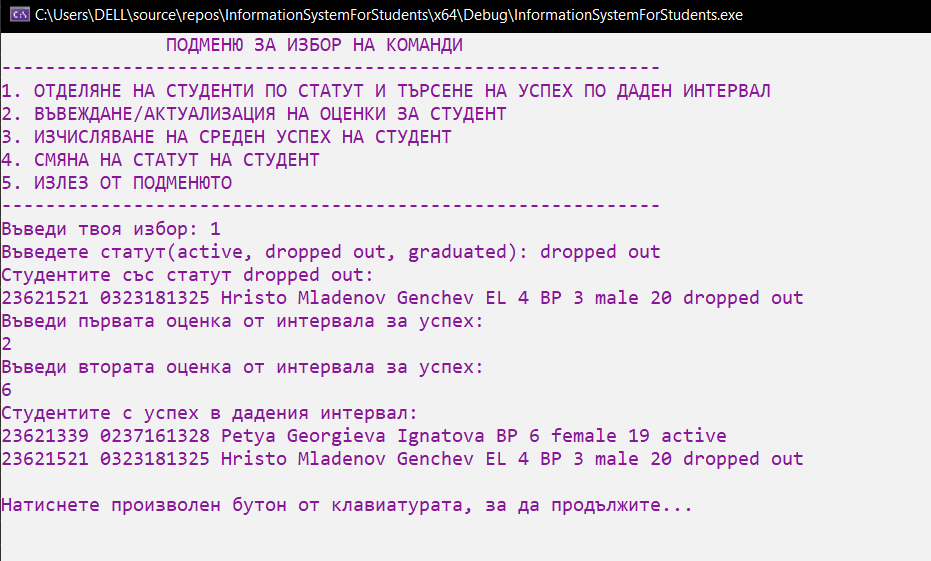
Потребителят трябва да въведе един от трите валидни статута, след което на екрана ще се изведат всички студенти с този статут, сортирани по факултетен номер. След това потребителят ще има възможност да напише интервал (min-max), в който да се търсят всички студенти със среден успех в този интервал. При извеждането им на екрана ще бъдат сортирани по ЕГН.

Снимка на изгледа с примерни изходни данни

*Фиг. 35 Въвеждане на опция без добавени студенти*

*Фиг. 36 Въвеждане на невалиден статут*

* Фиг. 37 Извеждане на екрана, че няма студенти с този статут/в този интервал*

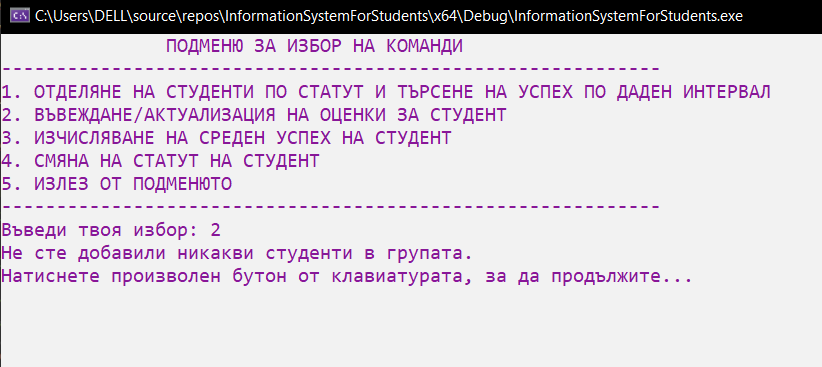
**

*Фиг. 38 Въвеждане на валидни данни и извеждане на екрана*

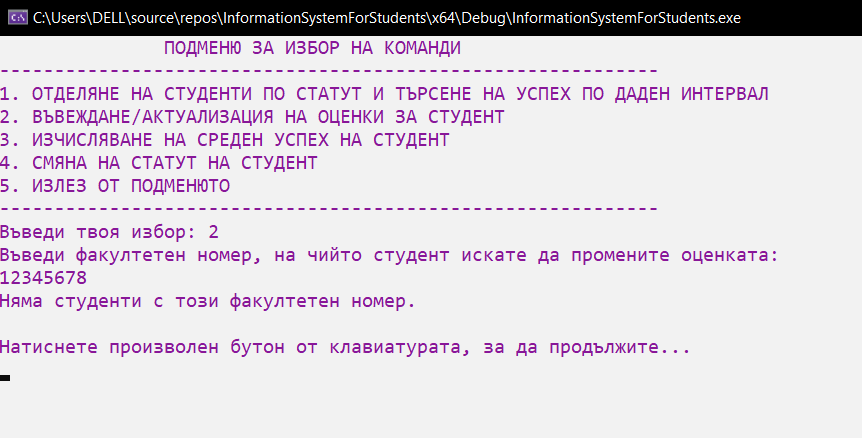
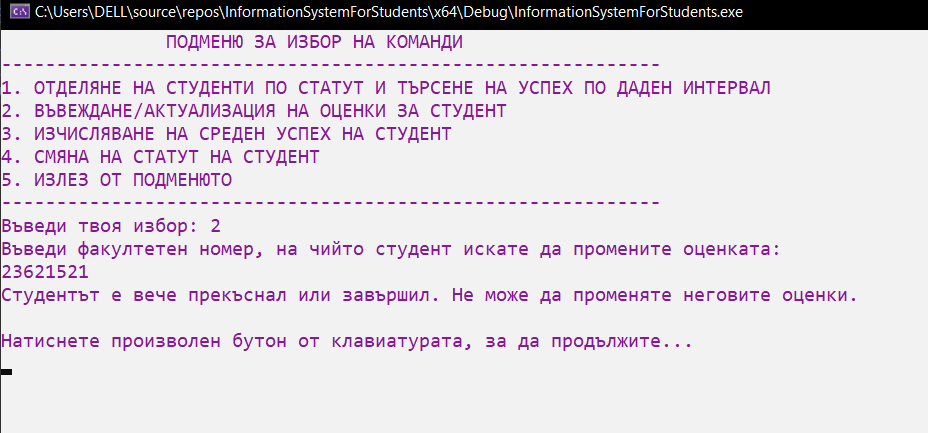
## Условие H (a,b) - допълнение второ

Входни данни:

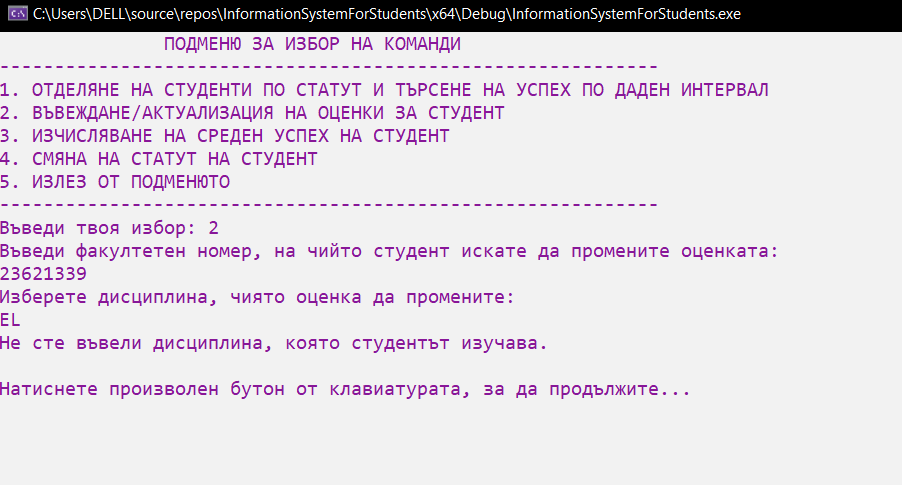
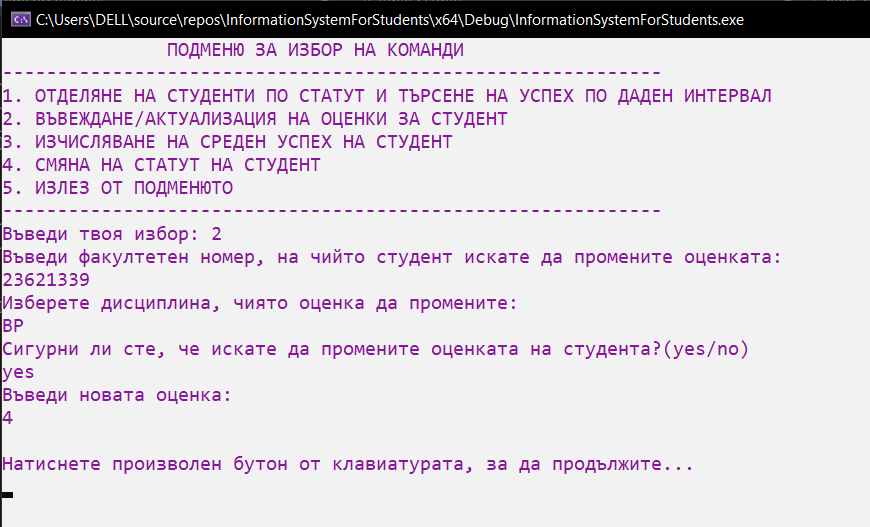
Чрез алгоритъма за въвеждане и актуализация на оценки на студент, потребителят има възможността да въвежда оценки по дадена дисциплина(оценката преди това е била 0) или да актуализира вече съществуваща оценка. Като, за да се случи това, трябва да въведе факултетен номер на студента, чиято оценка ще променя. С алгоритъма за среден успех автоматично се пресмята средния успех на дадения студент (дори след актуализация на оценка) и се извеждат данните на студента на екрана. Отново по факултетен номер се търси студента, на който потребителят иска да види средния успех.

Снимка на изгледа с примерни изходни данни

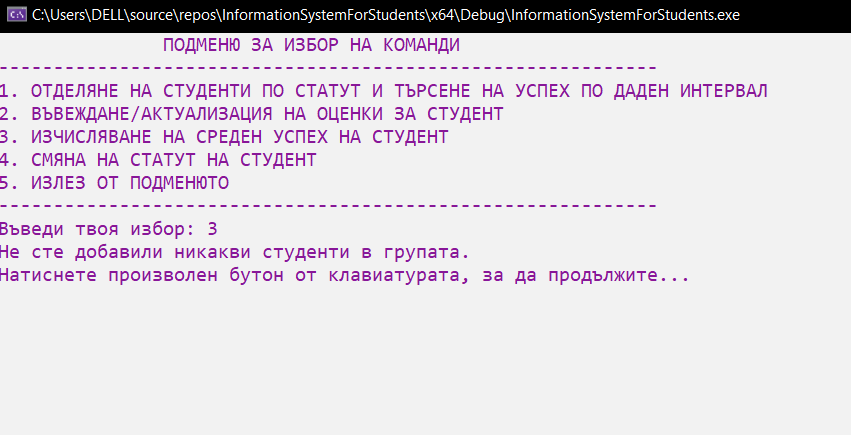
*Фиг. 39 Въвеждане на опция без добавени студенти*

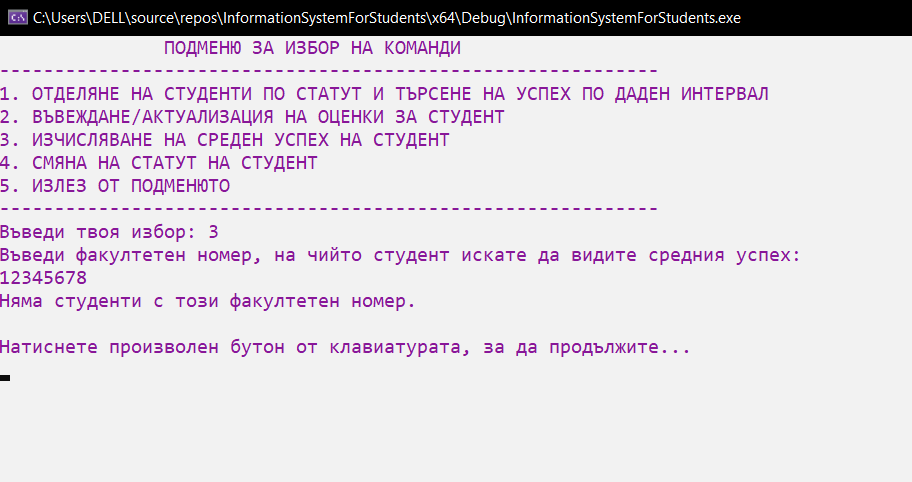
 *Фиг. 40 Въвеждане на невалиден фак. номер*

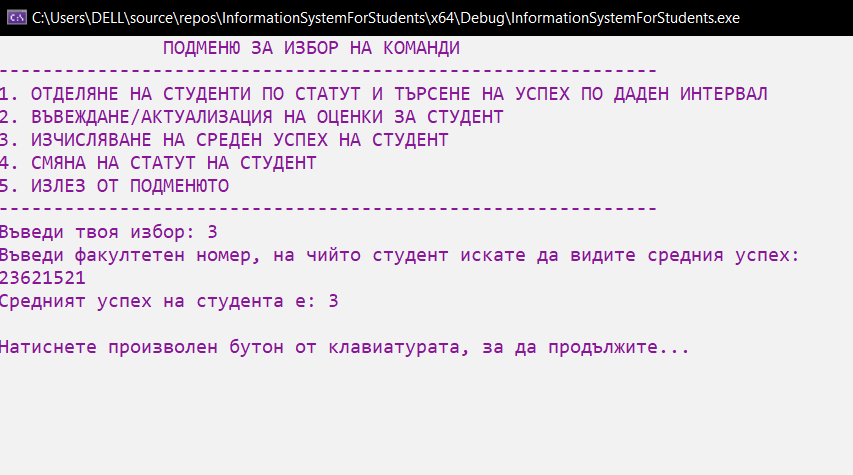
*Фиг. 41 Промяна на оценка на студент, който е прекъснал или вече завършил*

*Фиг. 42 Въвеждане на дисциплина, която студентът не изучава*

*Фиг. 43 Въвеждане на валидни данни и извеждане на екран*

*Фиг. 44 Въвеждане на опция без добавени студенти*

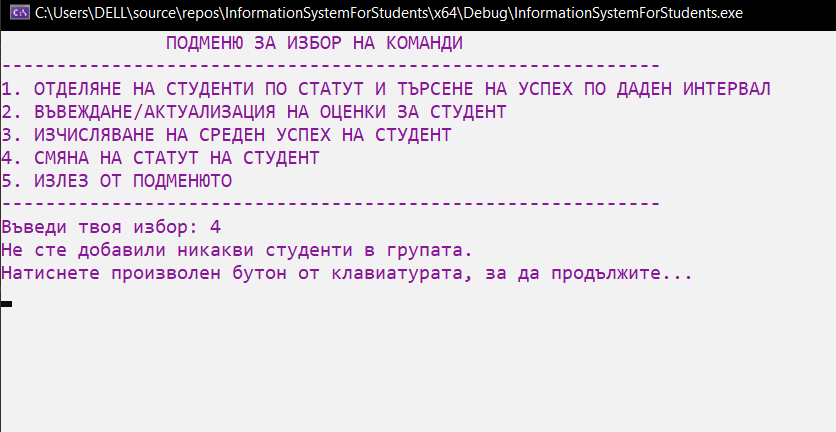
*Фиг. 45 Въвеждане на невалиден фак. номер*

*Фиг. 46 Въвеждане на валидни данни и извеждане на екран* 

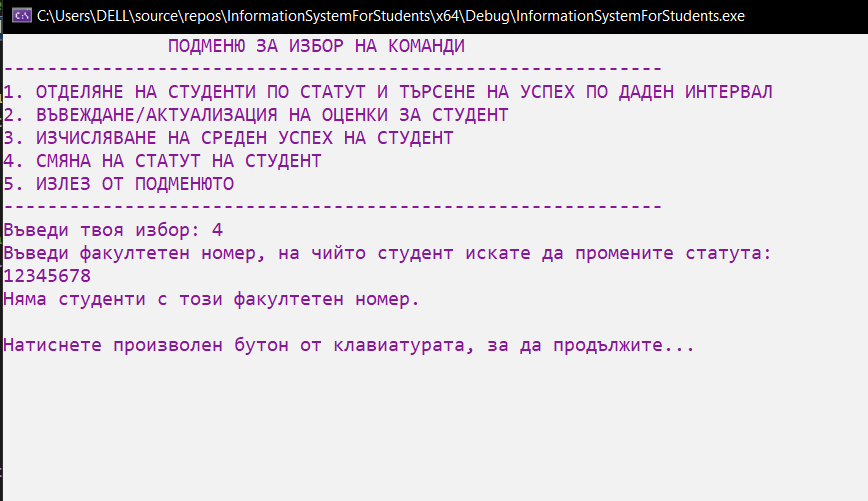
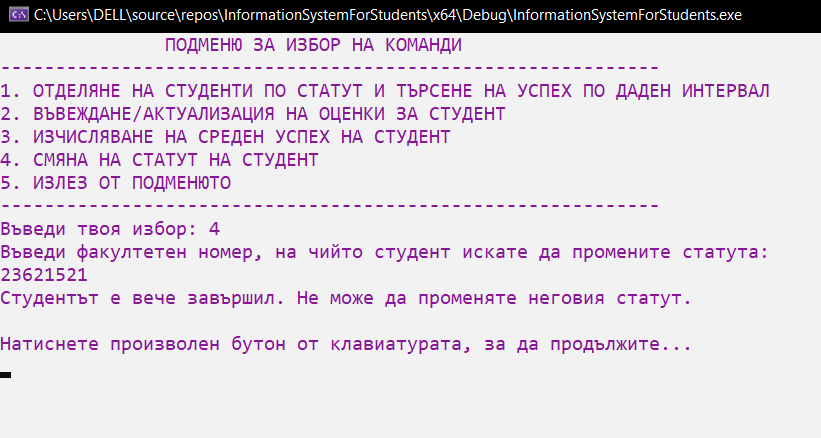
## Условие I - допълнение второ

Входни данни:

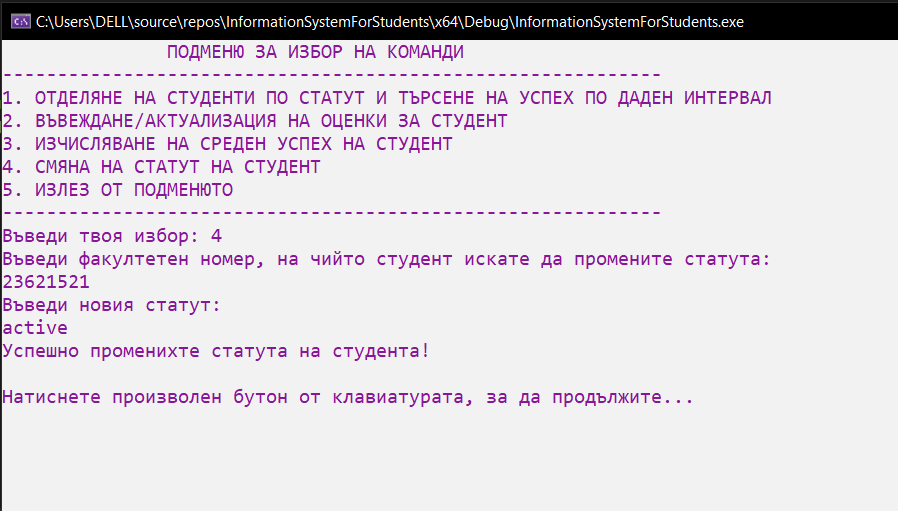
Чрез алгоритъма за смяна на статут на студент, потребителят има възможността да актуализира статута на студент. Като, за да се случи това, трябва да въведе факултетен номер на студента, чийто статут ще променя.

Снимка на изгледа с примерни изходни данни

*Фиг. 47 Въвеждане на опция без добавени студенти*

*Фиг. 48 Въвеждане на невалиден фак. номер*

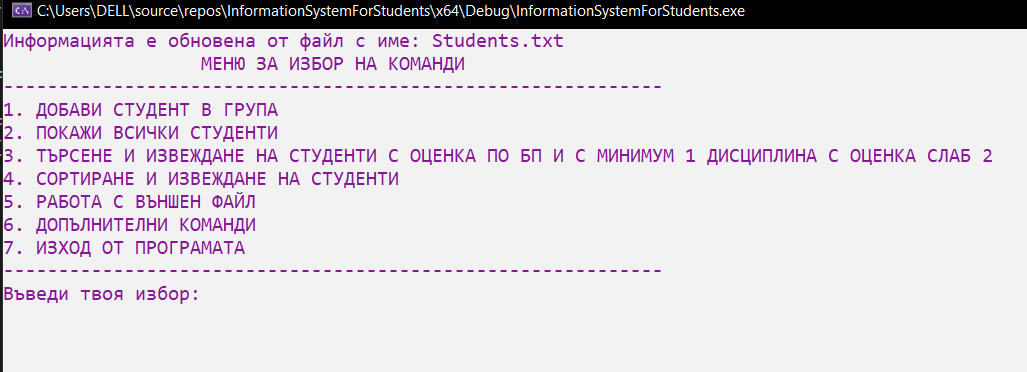
*Фиг. 49 Въвеждане на фак. Номер на студент, който вече е завършил*

*Фиг. 50 Въвеждане на валидни данни и извеждане на екран*

## Условие J - допълнение трето

Входни данни:

Данните със студентите се прочитат автоматично от текстов файл със стартирането на програмата. Данните за студентите се четат ред по ред. Със стартирането на програмата се извежда подходящо съобщение, което ни показва, че има студенти в нашата база данни.

Снимка на изгледа с примерни изходни данни

*Фиг. 51 Успешно прочитане на файла с данни за студентите*

# Изходен код на програмата

#include <iostream>

#include <string>

#include <iomanip>

#include <fstream>

#include <windows.h>

using namespace std;

//Структура с информация за студентите

struct Students

{

string fakNomer;

string egn;

string firstName;

string secondName;

string thirdName;

string discipline[5];

string gender;

int age = 0;

string statut;

int grades[5];

double avarageScore; //допълнение второ

};

//Дизайн на главното меню

void Menu()

{

cout << setw(42) << "МЕНЮ ЗА ИЗБОР НА КОМАНДИ" << endl;

cout << setfill('-') << setw(60) << "" << endl;

cout << setfill(' ');

cout << "1. ДОБАВИ СТУДЕНТ В ГРУПА \n"

<< "2. ПОКАЖИ ВСИЧКИ СТУДЕНТИ \n"

<< "3. ТЪРСЕНЕ И ИЗВЕЖДАНЕ НА СТУДЕНТИ С ОЦЕНКА ПО БП И С МИНИМУМ 1 ДИСЦИПЛИНА С ОЦЕНКА СЛАБ 2 \n"

<< "4. СОРТИРАНЕ И ИЗВЕЖДАНЕ НА СТУДЕНТИ \n"

<< "5. РАБОТА С ВЪНШЕН ФАЙЛ \n"

<< "6. ДОПЪЛНИТЕЛНИ КОМАНДИ \n"

<< "7. ИЗХОД ОТ ПРОГРАМАТА" << endl;

cout << setfill('-') << setw(60) << "" << endl;

cout << setfill(' ');

}

//Дизайн на подменюто

void SubMenu()

{

cout << setw(42) << "ПОДМЕНЮ ЗА ИЗБОР НА КОМАНДИ" << endl;

cout << setfill('-') << setw(60) << "" << endl;

cout << setfill(' ');

cout << "1. ОТДЕЛЯНЕ НА СТУДЕНТИ ПО СТАТУТ И ТЪРСЕНЕ НА УСПЕХ ПО ДАДЕН ИНТЕРВАЛ\n"

<< "2. ВЪВЕЖДАНЕ/АКТУАЛИЗАЦИЯ НА ОЦЕНКИ ЗА СТУДЕНТ \n"

<< "3. ИЗЧИСЛЯВАНЕ НА СРЕДЕН УСПЕХ НА СТУДЕНТ \n"

<< "4. СМЯНА НА СТАТУТ НА СТУДЕНТ \n"

<< "5. ИЗЛЕЗ ОТ ПОДМЕНЮТО \n";

cout << setfill('-') << setw(60) << "" << endl;

cout << setfill(' ');

}

//Функция за добавяне на студенти

void addStudents(Students mass[], int N, int& countAddedStudents)

{

//С тези три променливи се запазва броя дисциплини, които има студентът, и се изчислява средният успех

int disciplina;

int grade;

int allGrades = 0;

//Тази променлива се използва за брояч, който всеки път се увеличава с 1, когато се добави нов студент

int currentCount = countAddedStudents;

//for-цикъл, който върти до посочения от потребителя брой добавени студенти. Чрез него се въвеждат данните за всеки студент

for (int i = countAddedStudents; i < N + countAddedStudents; i++)

{

cout << "Въведи факултетен номер, ЕГН, трите имена, дисциплини, оценки, пол, възраст и статут на студент номер " << i + 1 << endl;

cout << "Факултетен номер: \n";

getline(cin, mass[i].fakNomer);

cin.clear();

while (mass[i].fakNomer.length() != 8)

{

cout << "Не сте въвели коректни данни. Факултетният номер трябва да съдържа 8 цифри. Въведете факултетен номер: \n";

getline(cin, mass[i].fakNomer);

cin.clear();

}

cout << "ЕГН: \n";

getline(cin, mass[i].egn);

cin.clear();

while (mass[i].egn.length() != 10)

{

cout << "Не сте въвели коректни данни. ЕГН-то трябва да съдържа 10 цифри. Въведете ЕГН: \n";

getline(cin, mass[i].egn);

cin.clear();

}

cout << "Име: \n";

getline(cin, mass[i].firstName);

cin.clear();

cout << "Презиме: \n";

getline(cin, mass[i].secondName);

cin.clear();

cout << "Фамилия: \n";

getline(cin, mass[i].thirdName);

cin.clear();

cout << "Въведи брой дисциплини: \n";

cin >> disciplina;

cin.ignore();

string listDisciplini[5];

while (disciplina == 0 || disciplina > 5)

{

cout << "Не сте въвели коректни данни. Въведете брой на дисциплине от 1 до 5: \n";

cin >> disciplina;

cin.ignore();

}

//Втори цикъл for, с който записваме всички дисциплини и оценките към тях

for (int j = 0; j < disciplina; j++)

{

cout << "Име на дисциплината (Изберете от изброените: БП(BP), УД(WD), ЕЛ(EL), СП(SP), МТ(MT)): \n";

getline(cin, listDisciplini[j]);

mass[i].discipline[j] = listDisciplini[j];

cin.clear();

cout << "Въведи оценка по дисциплината (0(Няма оценкa) или от 2 до 6): \n";

cin >> grade;

allGrades += grade;

while (grade < 0 || grade > 6 || grade == 1)

{

cout << "Не сте въвели коректни данни. Въведете оценка 0(Няма оценкa) или от 2 до 6: \n";

cin >> grade;

}

mass[i].grades[j] = grade;

cin.ignore();

}

mass[i].avarageScore = allGrades / disciplina;

cout << "Пол (male/female): \n";

getline(cin, mass[i].gender);

cin.clear();

while (mass[i].gender != "male" && mass[i].gender != "female")

{

cout << "Не сте въвели коректни данни. Въведете пол male или female: \n";

getline(cin, mass[i].gender);

cin.clear();

}

cout << "Възраст: \n";

cin >> mass[i].age;

cin.ignore();

cout << "Статут (active/dropped out/graduated): \n";

getline(cin, mass[i].statut);

cin.clear();

while (mass[i].statut != "active" && mass[i].statut != "dropped out" && mass[i].statut != "graduated")

{

cout << "Не сте въвели правилен статус. Въведете отново: \n";

getline(cin, mass[i].statut);

cin.clear();

}

cin.clear();

currentCount++;

}

//Присвояваме стойността на currentCount на countAddedStudents, за да имаме точната бройка в момента на студентите вътре в групата

//Извикали сме countAddedStudents по адрес във функцията, за да може стойността и да се запази.

countAddedStudents = currentCount;

}

//Дизайн на таблицата с информация за всички студенти

void tableDesign()

{

const int width = 170;

const int columnWidth = 20;

cout << endl << setw(width) << setfill('-') << "" << setfill(' ');

cout << endl << "|"

<< setw(columnWidth) << left << "ФАК. NO" << "|"

<< setw(columnWidth) << left << "ЕГН" << "|"

<< setw(columnWidth) << right << "ИМЕ" << "|"

<< setw(columnWidth) << right << "ПРЕЗИМЕ" << "|"

<< setw(columnWidth) << right << "ФАМИЛИЯ" << "|"

<< setw(columnWidth) << right << "ПОЛ" << "|"

<< setw(columnWidth) << right << "ВЪЗРАСТ" << "|"

<< setw(columnWidth) << right << "СТАТУТ" << "|";

cout << endl << setw(width) << setfill('-') << "" << setfill(' ') << endl;

}

//Извеждане на всички студенти в таблица

void print(Students mas[], int& countAddedStudents)

{

const int columnWidth = 20;

cout << "Студентите в групата: \n";

tableDesign();

for (int i = 0; i < countAddedStudents; i++)

{

cout << endl << "|"

<< setw(columnWidth) << left << mas[i].fakNomer << "|" << setfill(' ')

<< setw(columnWidth) << left << mas[i].egn << "|"

<< setw(columnWidth) << right << mas[i].firstName << "|"

<< setw(columnWidth) << right << mas[i].secondName << "|"

<< setw(columnWidth) << right << mas[i].thirdName << "|"

<< setw(columnWidth) << right << mas[i].gender << "|"

<< setw(columnWidth) << right << mas[i].age << "|"

<< setw(columnWidth) << right << mas[i].statut << "|" << endl;

}

}

//Извеждане на студенти с цялата им информация

void printWithoutTable(Students mas[], int& countAddedStudents)

{

for (int i = 0; i < countAddedStudents; i++)

{

//В arrayLength се изчислява колко елемента има в discipline, като разделя броя байтове от общия масив на байтовете на един елемент от масива с дисциплини

int arrayLength = sizeof(mas[i].discipline) / sizeof(mas[0].discipline[0]);

cout << mas[i].fakNomer << " " << mas[i].egn << " " << mas[i].firstName << " " << mas[i].secondName << " " << mas[i].thirdName << " ";

for (int j = 0; j < arrayLength; j++)

{

if (mas[i].discipline[j] == "")

{

continue;

}

cout << mas[i].discipline[j] << " " << mas[i].grades[j] << " ";

}

cout << mas[i].gender << " " << mas[i].age << " " << mas[i].statut << endl;

}

}

//Извеждане на всички данни на сортираните студенти

void printSorting(Students mas[], int& countAddedStudents)

{

for (int i = 0; i < countAddedStudents; i++)

{

int arrayLength = sizeof(mas[i].discipline) / sizeof(mas[0].discipline[0]);

cout << mas[i].fakNomer << " " << mas[i].egn << " " << mas[i].firstName << " " << mas[i].secondName << " " << mas[i].thirdName << " ";

for (int j = 0; j < arrayLength; j++)

{

if (mas[i].discipline[j] == "")

{

continue;

}

cout << mas[i].discipline[j] << " " << mas[i].grades[j] << " ";

}

cout << mas[i].gender << " " << mas[i].age << " " << mas[i].statut << endl;

}

}

//Сортиране на студентите до първо име във възходящ ред

void sorting(Students mas[], int& countAddedStudents)

{

//Променлива от типа на структурата, в която временно да запаметяване данни

Students temp;

//Създаване на нов масив, в който да запазим сортираните студенти

Students masCopy[30];

for (int i = 0; i < countAddedStudents; i++)

{

masCopy[i] = mas[i];

}

//Намираме дължината на масива като делим байтовете от целия масив на байтовете на един от елементите в него

int arrayLength = sizeof(masCopy) / sizeof(masCopy[0]);

//Сортиране чрез метод на мехурчето

for (int i = 0; i < countAddedStudents; i++)

{

for (int j = countAddedStudents - 1; j > i; j--)

{

if (masCopy[j - 1].firstName > masCopy[j].firstName)

{

temp = masCopy[j - 1];

masCopy[j - 1] = masCopy[j];

masCopy[j] = temp;

}

}

}

cout << "Сортираните студенти по име: " << endl;

printSorting(masCopy, countAddedStudents);

}

//Сортиране на студентите по факултетен номер

void sortByFakNomer(Students mas[], int& countAddedStudents)

{

//Променлива от типа на структурата, в която временно да запаметяване данни

Students temp;

//Създаване на нов масив, в който да запазим сортираните студенти

Students masCopy[30];

for (int i = 0; i < countAddedStudents; i++)

{

masCopy[i] = mas[i];

}

//Намираме дължината на масива като делим байтовете от целия масив на байтовете на един от елементите в него

int arrayLength = sizeof(masCopy) / sizeof(masCopy[0]);

for (int j = 0; j < countAddedStudents; j++)

{

if (masCopy[j].fakNomer > masCopy[j + 1].fakNomer && j + 1 < countAddedStudents)

{

temp = masCopy[j];

masCopy[j] = masCopy[j + 1];

masCopy[j + 1] = temp;

}

}

printSorting(masCopy, countAddedStudents);

}

//Сортиране на студентите по ЕГН

void sortByEGN(Students mas[], int& countAddedStudents)

{

//Променлива от типа на структурата, в която временно да запаметяване данни

Students temp;

//Създаване на нов масив, в който да запазим сортираните студенти

Students masCopy[30];

for (int i = 0; i < countAddedStudents; i++)

{

masCopy[i] = mas[i];

}

//Намираме дължината на масива като делим байтовете от целия масив на байтовете на един от елементите в него

int arrayLength = sizeof(masCopy) / sizeof(masCopy[0]);

for (int j = 0; j < countAddedStudents; j++)

{

if (masCopy[j].egn > masCopy[j + 1].egn && j + 1 < countAddedStudents)

{

temp = masCopy[j];

masCopy[j] = masCopy[j + 1];

masCopy[j + 1] = temp;

}

}

printSorting(masCopy, countAddedStudents);

}

//Отделяне на студентите по определен статут, въведен от потребителя и сортиране по фак. номер

void separateStudents(Students mas[], int size, int& countAddedStudents)

{

//Създаваме променлива от тип string, с която потребителя ще въведете желания от него статут, по който да ги отдели

string statut2 = "";

cout << "Въведете статут(active, dropped out, graduated): ";

getline(cin, statut2);

//Нов масив, в който да ги запаметява

Students masCopy[30];

//Валидация на статута

while (statut2 != "active" && statut2 != "dropped out" && statut2 != "graduated")

{

cout << "Не сте въвели нито един от трите статута. Въведете валиден статут: ";

cin >> statut2;

cin.clear();

}

for (int i = 0; i < countAddedStudents; i++)

{

//Проверка дали потребителя е избрал активен статут и дали съвпада със статута на студентът в момента на итерацията на цикъла

if (statut2 == "active" && mas[i].statut == "active")

{

for (int j = 0; j < 30; j++)

{

//Проверява дали в момента на итерацията няма записан студент. Ако firstName е празен стринг - вътре записва дадения човек

if (masCopy[j].firstName == "")

{

masCopy[j] = mas[i];

break;

}

}

//Броячът size следи за бройката на студентите, които са с въведения от потребителя статут

size++;

}

else if (statut2 == "dropped out" && mas[i].statut == "dropped out")

{

for (int j = 0; j < 30; j++)

{

if (masCopy[j].firstName == "")

{

masCopy[j] = mas[i];

break;

}

}

size++;

}

else if (statut2 == "graduated" && mas[i].statut == "graduated")

{

for (int j = 0; j < 30; j++)

{

if (masCopy[j].firstName == "")

{

masCopy[j] = mas[i];

break;

}

}

size++;

}

}

//Проверява дали масивът е празен - ако е празен, извежда даденото съобщение

if (masCopy[0].firstName == "")

{

cout << "Няма студенти със съответния статут. \n";

}

else

{

cout << "Студентите със статут " << statut2 << ": " << endl;

sortByFakNomer(masCopy, size);

}

}

//Отделяне на студенти с успех в интервал (мин -макс) и сортиране по ЕГН.

void separateStudentsByScore(Students mas[], int size, int& countAddedStudents)

{

int firstInterval(0), secondInterval(0), score(0), count(0);

cout << "Въведи първата оценка от интервала за успех: \n";

cin >> firstInterval;

cout << "Въведи втората оценка от интервала за успех: \n";

cin >> secondInterval;

Students masCopy[30];

for (int i = 0; i < countAddedStudents; i++)

{

//Зануляваме успеха и брояча след всяка итерация на цикъла

score = 0;

count = 0;

int arrayLength = sizeof(mas[i].discipline) / sizeof(mas[0].discipline[0]);

for (int j = 0; j < arrayLength; j++)

{

//Проверка дали има въведена дисциплина, ако няма продължава за следващата

if (mas[i].discipline[j] == "")

{

continue;

}

//Ако има записана дисциплина, събира вси1ките му оценки и се увеличава count

score += mas[i].grades[j];

count++;

}

//Проверка дали успеха му съвпада с двата интервала

if (score / count >= firstInterval && score / count <= secondInterval)

{

for (int k = 0; k < 30; k++)

{

if (masCopy[k].firstName == "")

{

masCopy[k] = mas[i];

size++;

break;

}

}

}

}

//Проверява дали масивът е празен - ако е празен, извежда даденото съобщение

if (masCopy[0].firstName == "")

{

cout << "Няма студенти с успех в дадения интервал. \n";

}

else

{

cout << "Студентите с успех в дадения интервал: \n";

sortByEGN(masCopy, size);

}

}

//Търсене и извеждане на студенти с оценка по БП в определен в интервал [min – max]

void studentsInInterval(Students mas[], int& countAddedStudents)

{

int firstInterval(0), secondInterval(0), count(0);

cout << "Първо ще бъдат показани студентите с оценка по БП в даден интервал, след това всички студенти с поне една оценка Слаб 2 по някоя дисциплина \n";

cout << "Въведи първата оценка по БП от интервала: \n";

cin >> firstInterval;

cout << "Въведи втората оценка по БП от интервала: \n";

cin >> secondInterval;

Students masCopy[30];

for (int i = 0; i < countAddedStudents; i++)

{

int arrayLength = sizeof(mas[i].discipline) / sizeof(mas[0].discipline[0]);

for (int j = 0; j < arrayLength; j++)

{

//Проверка дали в дадената итерация има въведена дисциплина

if (mas[i].discipline[j] == "")

{

continue;

}

//Проверка дали текущата дисциплина е БП

if (mas[i].discipline[j] == "BP")

{

//Проверка дали оценката попада в дадения интервал

if (mas[i].grades[j] >= firstInterval && mas[i].grades[j] <= secondInterval)

{

for (int k = 0; k < 30; k++)

{

if (masCopy[k].firstName == "")

{

masCopy[k] = mas[i];

count++;

break;

}

}

}

}

else

{

continue;

}

}

}

if (masCopy[0].firstName == "")

{

cout << "Няма студенти с оценки по БП в дадения интервал. \n";

}

else

{

cout << "Студентите с оценки по БП в дадения интервал са: \n";

printWithoutTable(masCopy, count);

}

}

//Търсене и извеждане на студенти с минимум един предмет, с оценка Слаб 2

void studentsWithFailDiscipline(Students mas[], int& countAddedStudents)

{

int countStudent(0), count(0);

Students masCopy[30];

for (int i = 0; i < countAddedStudents; i++)

{

//Зануляваме countStudent за всяка итерация на цикъла

countStudent = 0;

int arrayLength = sizeof(mas[i].discipline) / sizeof(mas[0].discipline[0]);

for (int j = 0; j < arrayLength; j++)

{

//Проверка дали оценката е различна от 2

if (mas[i].grades[j] != 2)

{

continue;

}

else

{

for (int k = 0; k < 30; k++)

{

//Проверка дали firstName е празно и дали вече този студент не е запазен в масива

if (masCopy[k].firstName == "" && countStudent == 0)

{

masCopy[k] = mas[i];

count++;

countStudent++;

break;

}

}

}

}

}

if (masCopy[0].firstName == "")

{

cout << "Няма студенти с оценка Слаб 2. \n";

}

else

{

cout << "Студентите с оценка Слаб 2: \n";

printWithoutTable(masCopy, count);

}

}

//Смяна на статут на студент

void newStatut(Students mas[], int& countAddedStudents)

{

cout << "Въведи факултетен номер, на чийто студент искате да промените статута: \n";

string fakNomerStudent, newStatut;

//Създаваме булева променлива, с която ще правим проверка за факултетния номер

bool fakNomerExists = false;

getline(cin, fakNomerStudent);

cin.clear();

for (int i = 0; i < countAddedStudents; i++)

{

//Проверка дали съществува студент с въведения факултетен номер

if (mas[i].fakNomer == fakNomerStudent)

{

fakNomerExists = true;

break;

}

}

//Влизаме в цикъла, ако съществува студент с такъв факултетен номер

if (fakNomerExists)

{

for (int i = 0; i < countAddedStudents; i++)

{

//Проверка дали фак. номер на студента в този момент съвпада с въведения от потребителя

if (fakNomerStudent != mas[i].fakNomer)

{

continue;

}

if (mas[i].statut == "active" || mas[i].statut == "dropped out")

{

cout << "Въведи новия статут: \n";

getline(cin, newStatut);

mas[i].statut = newStatut;

cout << "Успешно променихте статута на студента! \n";

}

else

{

cout << "Студентът е вече завършил. Не може да променяте неговия статут. \n";

}

}

}

else

{

cout << "Няма студенти с този факултетен номер. \n";

}

}

//Въвеждане/Актуализиране на оценка

void newGrade(Students mas[], int& countAddedStudents)

{

cout << "Въведи факултетен номер, на чийто студент искате да промените оценката: \n";

int allGrades = 0;

int count = 0;

string fakNomerStudent;

string disciplineGrade;

bool fakNomerExists = false;

getline(cin, fakNomerStudent);

cin.clear();

for (int i = 0; i < countAddedStudents; i++)

{

//Проверка дали съществува студент с въведения факултетен номер

if (mas[i].fakNomer == fakNomerStudent)

{

fakNomerExists = true;

break;

}

}

if (fakNomerExists)

{

for (int i = 0; i < countAddedStudents; i++)

{

int arrayLength = sizeof(mas[i].discipline) / sizeof(mas[0].discipline[0]);

//Проверка дали фак. номер на студента в този момент съвпада с въведения от потребителя

if (fakNomerStudent != mas[i].fakNomer)

{

continue;

}

//Ако студентът е активен

if (mas[i].statut == "active")

{

//Булева променлива, с която ще проверим дали е въведена дисциплина, която студентът изучава

bool disciplineExists = true;

cout << "Изберете дисциплина, чиято оценка да промените: \n";

getline(cin, disciplineGrade);

cin.clear();

for (int j = 0; j < arrayLength; j++)

{

if (mas[i].discipline[j] == disciplineGrade)

{

disciplineExists = true;

//Когато оценката е различна от 0(Тоест няма оценка)

if (mas[i].grades[j] != 0)

{

cout << "Сигурни ли сте, че искате да промените оценката на студента?(yes/no) \n";

string check;

getline(cin, check);

cin.clear();

if (check == "yes")

{

int newGrade = 0;

cout << "Въведи новата оценка: \n";

cin >> newGrade;

mas[i].grades[j] = newGrade;

break;

}

else if (check == "no")

{

cout << "Вие отказахте да промените оценката на студента." << endl;

break;

}

}

else if (mas[i].grades[j] == 0)

{

int newGrade = 0;

cout << "Въведи новата оценка: \n";

cin >> newGrade;

mas[i].grades[j] = newGrade;

break;

}

}

else

{

disciplineExists = false;

}

}

if (disciplineExists == false)

{

cout << "Не сте въвели дисциплина, която студентът изучава. \n";

}

//for-цикъл, с който да съберем всички оценки на студента, след промените по тях

for (int j = 0; j < arrayLength; j++)

{

//Отделяме само оценките, които е въвел студента - там, където няма въведена оценка, се запазват отрицателни числа

if (mas[i].grades[j] < 0)

{

break;

}

allGrades += mas[i].grades[j];

count++;

}

}

else

{

cout << "Студентът е вече прекъснал или завършил. Не може да променяте неговите оценки. \n";

break;

}

//Пресмятаме новия среден успех

mas[i].avarageScore = allGrades / count;

}

}

else

{

cout << "Няма студенти с този факултетен номер. \n";

}

}

//Извеждане на средния успех на студент

void avarageGrades(Students mas[], int& countAddedStudents)

{

cout << "Въведи факултетен номер, на чийто студент искате да видите средния успех: \n";

string fakNomerStudent;

bool fakNomerExists = false;

getline(cin, fakNomerStudent);

cin.clear();

for (int i = 0; i < countAddedStudents; i++)

{

if (mas[i].fakNomer == fakNomerStudent)

{

fakNomerExists = true;

break;

}

}

if (fakNomerExists)

{

for (int i = 0; i < countAddedStudents; i++)

{

int arrayLength = sizeof(mas[i].discipline) / sizeof(mas[0].discipline[0]);

if (fakNomerStudent != mas[i].fakNomer)

{

continue;

}

else

{

cout << "Средният успех на студента е: " << mas[i].avarageScore << endl;

}

}

}

else

{

cout << "Няма студенти с този факултетен номер. \n";

}

}

//Записване на информация във файл

void SaveStudentsToFile(Students mas[], int& countAddedStudents)

{

const string FILE\_NAME = "Students.txt";

ofstream fp(FILE\_NAME, ios::out);

fp << countAddedStudents << endl;

if (fp.fail())

{

cout << "Грешка!" << endl;

exit(1);

}

for (int i = 0; i < countAddedStudents; i++)

{

int arrayLength = 0;

fp << mas[i].fakNomer << endl << mas[i].egn << endl << mas[i].firstName << endl << mas[i].secondName << endl << mas[i].thirdName << endl;

for (int j = 0; j < 5; j++)

{

if (mas[i].discipline[j] != "")

{

arrayLength++;

}

}

fp << arrayLength << endl;

for (int j = 0; j < arrayLength; j++)

{

fp << mas[i].discipline[j] << endl << mas[i].grades[j] << endl;

}

fp << mas[i].gender << endl << mas[i].age << endl << mas[i].statut << endl;

}

cout << "Всичко е запазено във файл с име: " << FILE\_NAME << endl;

fp.close();

}

//Четене на информация от файл

void ReadStudentsFromFile(Students mas[], int& countAddedStudents)

{

int allGrades = 0;

const string FILE\_NAME = "Students.txt";

ifstream fp(FILE\_NAME, ios::in);

if (fp.fail())

{

cout << "Грешка!" << endl;

exit(1);

}

fp >> countAddedStudents;

fp.ignore();

for (int i = 0; i < countAddedStudents; i++)

{

allGrades = 0;

int arrayLength = 0;

getline(fp, mas[i].fakNomer);

getline(fp, mas[i].egn);

getline(fp, mas[i].firstName);

getline(fp, mas[i].secondName);

getline(fp, mas[i].thirdName);

fp >> arrayLength;

fp.ignore();

for (int j = 0; j < arrayLength; j++)

{

getline(fp, mas[i].discipline[j]);

fp >> mas[i].grades[j];

fp.ignore();

allGrades += mas[i].grades[j];

}

mas[i].avarageScore = allGrades / arrayLength;

getline(fp, mas[i].gender);

fp >> mas[i].age;

fp.ignore();

getline(fp, mas[i].statut);

}

cout << "Информацията е обновена от файл с име: " << FILE\_NAME << endl;

fp.close();

}

//Функция за изтриване на текст след всяка изпълнена команда

void nextCommand()

{

string next = "";

cout << "\nНатиснете произволен бутон от клавиатурата, за да продължите...\n";

cin.ignore();

getline(cin, next);

system("CLS");

}

int main()

{

//Изпълнение на конзолната програма на цял екран

::SendMessage(::GetConsoleWindow(), WM\_SYSKEYDOWN, VK\_RETURN, 0x20000000);

//Изпълнение на конзолната програма на цял екран с бутони за затваряне и минимизиране

//ShowWindow(GetConsoleWindow(), SW\_MAXIMIZE);

setlocale(LC\_ALL, "");

int choice, secondChoice, N(0), size(0), countAddedStudents(0);

string next = "";

Students student[30];

//Четене на файл с данни

ReadStudentsFromFile(student, countAddedStudents);

do

{

Menu();

cout << "Въведи твоя избор: ";

cin >> choice;

cin.clear();

cin.ignore(1000, '\n');

//Валидация за входа

while (choice < 1 || choice > 7)

{

cout << endl << "Не сте избрали нито едно от дадените числа. Изберете от показаните в менюто." << endl;

cout << "\nВъведи твоя избор: ";

cin >> choice;

cin.clear();

cin.ignore(1000, '\n');

}

switch (choice)

{

case 1:

cout << "Въведете броя студенти в групата: ";

while (N <= 0 || N + countAddedStudents > 30) //Валидира броят добавяни книги

{

cin >> N;

if (N <= 0) //за отрицателни числа

{

cout << endl << "Моля въведете положително цяло число!" << endl

<< "\nВъведете броя студенти в групата: ";

}

if (N + countAddedStudents > 30) //за числа препълващи масива

{

cout << endl << "Броят, който искате да въведете, надвишава максималната стойност на групата. ("

<< countAddedStudents << "+" << N

<< ")/30" << endl

<< endl << "Моля въведете по-малко число, така че групата да не надвишава 30 студенти: ";

}

}

cin.ignore();

addStudents(student, N, countAddedStudents);

cout << "Добавихте успешно нов студент в групата. Натиснете произволен бутон от клавиатурата... \n";

cin.ignore();

getline(cin, next);

system("CLS");

break;

case 2:

if (countAddedStudents == 0)

{

cout << "Не сте добавили никакви студенти в групата.";

nextCommand();

break;

}

print(student, countAddedStudents);

nextCommand();

break;

case 3:

if (countAddedStudents == 0)

{

cout << "Не сте добавили никакви студенти в групата.";

nextCommand();

break;

}

studentsInInterval(student, countAddedStudents);

studentsWithFailDiscipline(student, countAddedStudents);

nextCommand();

break;

case 4:

if (countAddedStudents == 0)

{

cout << "Не сте добавили никакви студенти в групата.";

nextCommand();

break;

}

sorting(student, countAddedStudents);

nextCommand();

break;

case 5:

if (countAddedStudents == 0)

{

cout << "Не сте добавили никакви студенти в групата.";

nextCommand();

break;

}

SaveStudentsToFile(student, countAddedStudents);

ReadStudentsFromFile(student, countAddedStudents);

nextCommand();

break;

case 6:

system("CLS");

SubMenu();

//Подменю

do

{

cout << "Въведи твоя избор: ";

cin.clear();

cin >> secondChoice;

cin.clear();

cin.ignore(1000, '\n');

while (secondChoice < 1 || secondChoice > 5)

{

cout << endl << "Не сте избрали нито едно от дадените числа. Изберете от показаните в подменюто." << endl;

cout << "\nВъведи твоя избор: ";

cin >> secondChoice;

cin.clear();

cin.ignore(1000, '\n');

}

switch (secondChoice)

{

case 1:

if (countAddedStudents == 0)

{

cout << "Не сте добавили никакви студенти в групата.";

nextCommand();

SubMenu();

break;

}

separateStudents(student, size, countAddedStudents);

separateStudentsByScore(student, size, countAddedStudents);

nextCommand();

SubMenu();

break;

case 2:

if (countAddedStudents == 0)

{

cout << "Не сте добавили никакви студенти в групата.";

nextCommand();

SubMenu();

break;

}

newGrade(student, countAddedStudents);

nextCommand();

SubMenu();

break;

case 3:

if (countAddedStudents == 0)

{

cout << "Не сте добавили никакви студенти в групата.";

nextCommand();

SubMenu();

break;

}

avarageGrades(student, countAddedStudents);

nextCommand();

SubMenu();

break;

case 4:

if (countAddedStudents == 0)

{

cout << "Не сте добавили никакви студенти в групата.";

nextCommand();

SubMenu();

break;

}

newStatut(student, countAddedStudents);

nextCommand();

SubMenu();

break;

case 5:

break;

}

} while (secondChoice != 5);

system("CLS");

choice = 0;

break;

case 7:

break;

}

} while (choice != 7);

return 0;

}