1. УВОД:

Проектът представлява семпла база данни за нанасяне и управление на оценки на студенти. Всеки студент, включен в базата данни, се идентифицира с уникален факултетен номер (положително число, не по-голямо от 9999). Студентът може да има само една оценка (от 2 до 6) и е характеризиран с име и фамилия.

За да е реализиран коректно проектът, базата данни трябва да използва само един постоянен файл „*StudentsGrades.db*“. Програмата трябва да помни промените след затварянето й. При ново зареждане на базата данни всички студенти трявба да са налични както са оставени преди последното затваряне. Управлението на програмата става, чрез потребителски интерфейс, който приема 5 команди – *“create”, “sequentialSearch”, “update”, “delete”* и *“exit”.* При избиране на команда “*create*”, програмата подканва да бъдат въведени данните на един студент с точна последователност – „Факултетен Номер“, „Име“, „Фамилия“ и „Оценка“. Данните биват валидирани и ако валидацията е успешна, ги добавя във файлa „*StudentsGrades.db*“. Ако въведените данни са невалидни (оценка извън зададеният стандарт, празни имена или твърде голям факултетен номер) се извежда съобщение, коя от данните е невалидна и програмата се връща в начална позиция. Командата *“sequentialSearch”* изисква от потребителя да въведе факултетен номер. След това програмата проверява за съвпадения в базата данни. При успешен резултат, програмата извежда в конзолата информацията на студента, обозначен с този факултетен номер. Ако липсва такъв студент, се извежда съответното съобщение на екрана (“*Record not found!”*) и програмата се връща в начална позиция. Команда *“update”* служи за промяна на оценката на даден студент. След избиране на командата, потребителят трябва да въведе факултетен номер, последван от новата оценка. Ако студент с такъв факултетен номер съществува, оценката му се променя с подадента, след необходимата валидация. Програмата извежда съобщението (“*Record not found!”*) ако такъв студент не съществува. *“delete”* служи за изтриване на даден студент от базата данни. За изпълнение на тази команда, програмата се нуждае от факултетен номер. Студент със съответния факултетен номер, бива заличен от файлa. Избор на *“exit”* слага край на изпълнението на програмата.

В тази документация са представени необходимите ООП принципи, начина на работа, използваните дефиниции и алгоритми.

1. Преглед на предметната област:

2.1 За проекът за използвани следните дефиниции и алгоритми:

Клас: Съставна, статична, хетерогенна структура от данни. Дефинирането на клас въвежда нов тип който може да бъде интегриран в езика. Класовете предоставят възможност за капсулиране на член данните и запазване на желаният вид на информацията.

Приятелски функции: външни функции, които имат директен достъп до капсулирани данни на клас, се наричат приятелски. Прототип на функцията се дефинира в класа с оператор *“friend”*. Тялото на функцията стои извън калса.

Предефиниране на оператори: разширява функционалонстта на даден оператор. Това става, чрез дефиниране на операторни функции. Операторните функции трябва да бъдат създадени Ако операторите изискват достъп до капсулирани член данни, трябва да бъдат дефинирани като приятелски функции.

Динамична памет: този тип се дефинира за създаване на масиви/редици с големина, зададена по време на изпълнение на програмата. За динамично заделяне на памет в *C++* се използва оператор “*new*” върху пойнтер, последван от тип данни (и при заделяне на масив - размер). Всяка динамично заделена памет трябва да бъде изтрита ръчно, за да се избегне отичането на памет. Динамично заделена променлива се изтрива с оператор “*delete*” (“*delete[]*” за масиви), последвано от пойнер, сочещ към блока с памет.

Каноничен запис: Каноничният запис е задължителен компонент на класове, в които се използват динамично заделени член данни. Целта на каноничният запис е да се избегнат неправилно заделяне и отичане на памет. Каноничният запис се състои от конструктор по подразбиране, конструктор за копиране, предефиниране на оператор „=“ и деструктор.

Текстов файл: Проектът взима и запазва информацията за всички студенти в текстов файл. За четене на файлове в този проект е използван оператор „>>“, а за писане – „<<“.

Алгоритъм за последователно търсене: Алгоритъмът е изискване към проектът. Представлява търсене на елемент във файл, чрез последователно обхождане на всички елементи.

* 1. Възможни проблеми при реализацията на проекта са въвеждане на невалидни данни, грешно използване на динамична памет и проблеми свързани с големият обем от данни.
  2. Проблемите биват решени по следният начин:

За да се запази коректната структура на данните, всяка външна входна операция в класовете е проверена за коректност на данните. В случай на въведени невалидни данни, обектът е обявен за невалиден и не се запазва във файла.

Във всеки клас, използващ динамична памет, е създаден каноничен запис, който предотвратява възможни грешки при стандартни операции.

Проблемът с големия обем данни е разрешен, чрез използване на адрес, за прехвърляне на големи данни и обекти между функциите в програмата. По този начин се избягва копирането на обектите (което води до удвояване на използваната памет).

От потребителя се изисква точно използване на функциите определни в интерфейса на програмата (*“create”, “sequentialSearch”, “update”, “delete”* и *“exit”*). Всеки невалиден вход от конзолата извежда съобщение за съответната грешка.

1. Проектиране, реализация и тестване:

За създаване на проекта са използвани два класа. Клас “*Student*”, който се грижи за индивидуалните данни на всеки студент и клас *“Students”*, който съдържа колекция от студенти и е отговорен за основната комуникация с файла. Типът наследяване, използван в проекта е, така нареченият, *“has a”*, като класа *“Students”*, съдържа много инстанции на класа *“Student”*.

Класът *“Student”* използва 5 член данни. Факултетен номер - главен идентификатор на студента, име, фамилия, оценка и булева променлива, която указва дали обектът е създаден с валидни данни. Името и фамилията на студента се пазят в динамично заделени масиви от тип “*char”.* За управлението на динамично заделена памет се грижи каноничен запис съставен от: Конструктор по подразбиране - този конструктор задава стойности по подразбиране на обекти, създадени без подадени данни (празни елементи на масив от студенти). Конструктора по подразбиране създава обект с данни - *0 Default Student 2*. Непроменените дефолтни стойности са пренебрегнати от програмата и не указват влияние върху работата ѝ; Конструктор за копиране - този конструктор позволява създаването на студент, копие на друг студент. Конструкторът за копиране се грижи двата обекта, да не споделят едни и същи член данни (пойнтери към динамично заделени имена); Оператор “=” е дефиниран със същата цел като конструктора за копиране - да се избегне копирането на адрес към един и същ, динамично заделен блок памет. За да работи оператор “=” обаче, преди копиране се освобождава паметта, използвана от обекта, който приема данните; Деструктора *“~Student()”* цели да освободи всяка динамично заделена памет, преди терминиране на обекта. За да бъде избегнато преизползването на код, в *“Student”* са съсздадени помощни функции *“copy()”* (копира данни от друг обект) и *“clear()”* (освобождава паметта използвана от сегашният обект). Данните на класа могат да бъдат достъпени от външни функции, чрез предоставените “*get”* функции. Всяка *“get”* функция връща член данните във вид, които не може да бъде модифициран. Въведените член данни се проверяват от съответните *“set”* фунцкии. Изискването към факултетният номер е да е положително число и да е не по голям от 99999. Програмата не приема факултетен номер - 0, тъй като това се използва за дефолтна стойност. Проверката на имената се състои в това, дали подадените стрингове са празни. При подаден празен стринг в “*setFirstName*()”, програмата задава стойност по подразбиране - “*Default*”. Празният стринг подаден в “*setLastName*()” създава дефолтна фамилия - “*Student*”. Въведените оценки трябва да бъдат цели числа в интервала от 2 до 6. Невалидните оценки са приравнени на 2. За улеснено записване и прочитане от файл, за обект от тип *“Student”* са предефинирани оператоприте “<<” и “>>”. Изходен оператор “<<” извежда съответният обект във вид: “Факултетен Номер” “Име” “Фамилия” “Оценка” и добавя нов ред. Входният оператор “>>” приема данните в същата последователност. Той се грижи за валидацията на данните и за коректното създаване на съответния обект, чрез извикване на съответните *“set”* функции.

Класът *“Students”* се грижи за цялостната работа на нужната база данни. Базата данни в паметта, представлява обект, инстанция на този клас. В класът са заделени 3 член данни - пойнтер от тип *“Student”*, брой **създадени** студенти - *“count”* и размер на масива от тип *“Student”*. Понеже масива от студенти е заделен динамично, в този клас е дефиниран каноничен запис, подобен на записа в класa “*Student”.* Целта на този каноничен запис, отново, е да гарантира коректното използване на динамична памет от масива студенти. Класът съдържа и два сет метода, които проверяват валидността, на копираните размер на масива и брой на студентите. *“Students”* съдържа всички функции, необходими за управлението на данните и няколко помощни функции.

Помощните фунцкии са “*resize*()”, “*findFN*()”, “*getFileSize*()”, “*saveToFile*()” и “*replaceFile*()”. Функцията “*resize*()” преоразмерява масива при достигане на лимита студенти. Всяко извикване на функцията удвоява размера на сегашният масив от студенти, като заделя памет за нов, двойно по-голям масив и освобождава паметта на предходния. “*findFN*()” приема факултетен номер като параметър и обхожда масива студенти последователно и връща като резултат позицията на която се намира съответният факултетен номер, ако такъв съществува. При липса на резултат, функцията връща -1. Функцията “*getFileSize*()” връща броят студенти, които са налични във файла „*StudentsGrades.db*“. *“saveToFile()”* отваря файла в режим *“std::ios::app”*, който поставя указателя “put” в краят на файла. При успешно отваряне на файла, функцията добавя последният студент във файла на нов ред. “*replaceFile*()” служи за промени във вътрешността на файла. Функцията се вика при необходимост от промяна или изтриване на даден ред. Тя отваря файла в режим *“std::ios::trunc”,* който изчиства съдържанието на файла и след това записва цялата база данни, налична в *“Students”.*

Командите избрани от потребителя се управляват от останалите методи на класът. При избор на команда “*sequentialSearch*”, програмата извиква функцията *findStudent*(). Този метод проверява дали ли съответният факултетен номер съществува в базата данни и ако да, извежда цялата информацията за него в конзолата. *“addStudent()”* с параметри - информацията за един студент проверява дали съответният факултетен номер е наличен. Ако не съществува, функцията подава на класът *“Student”* съответните параметри и ако са валидни го записва във файла. Студентите биват изтрити при извикване на функцията *“deleteStudent()”.* Функцията проверява дали въведеният факултетен номер съществува. Ако е наличен, програмата го премахва от масива студенти. След успешно завършване на операцията, файлът бива обновен. *“updateStudent()”* служи за подмяна на оценката на даден студент. При успешна промяна, файлът бива заместен с функцията “*replaceFile*()”. За да бъде изпълнено изискването, програмата да зарежда, както е била затворена след последна модификация, при всяко отваряне на програмата зарежда функция *“loadFile()”*. Функцията отваря файла за четене и ако в него има записани студенти ги добавя в масива “*studentCollection*”. За успешната работа на този метод са използвани, размер на файла, оператор “>>” за четене на студент от файла и функция, която приема студент като параметър и го добавя в масива. Ако програмата попадне на невалидни стойности, тя пропуска създаването на съответният студент и извежда съобщение на екрана за загубените данни.

За тестването на програмата е създаден файл „*StudentsGrades.db*“ с 1000 студента. След това са тествани потребителският интерфейс и коректната операция на методите. За да бъде проверено състоянието на програмата ако файлът е бил променен от външен източник, на един от редовете във файла са въведени невалидни стойности.

1. Обобщение:

Програмата изпълнява поставените в условието на задачата изисквания. Спазени са основните ООП принципи. Потребителският интерфейс работи и дава обратна връзка, при грешни действия.

Относно насоки за бъдещо развитие и усъвършенстване, функционалността на приложението може да бъде разширена, като позволи на потребителя да подава повече от една оценка на студент. Оценките могат да бъдат приемани, като дробни числа, за да повиши реализма на програмата. Базата данни може да бъде променена, така че да е достъпна за повече видове потребители (преподаватели, студенти, факултетно управление...), но... това остава за следващ път.