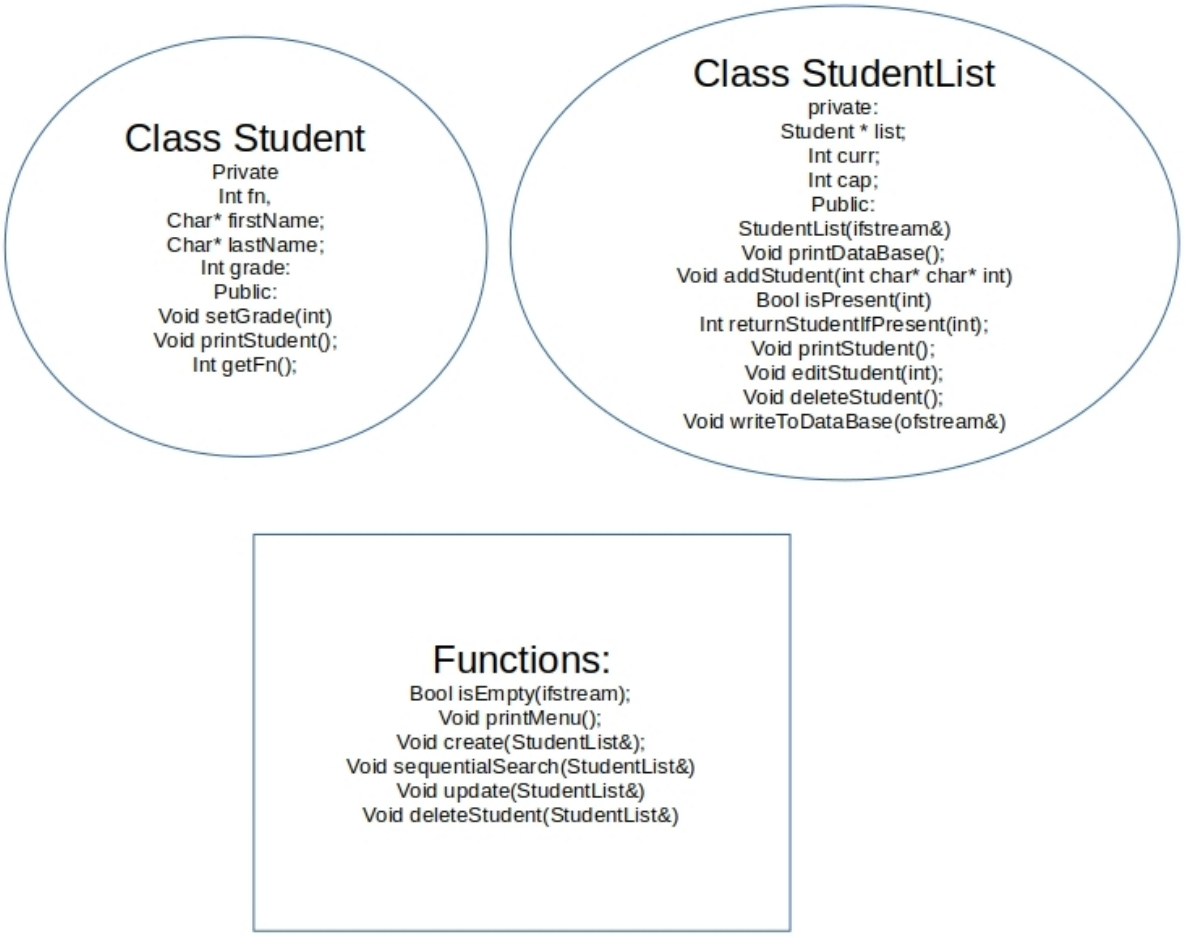
БАЗА ОТ ДАННИ ЗА ОЦЕНКИ НА СТУДЕНТИ

/Александър Бакалов/

Идеята на проекта е да се създаде локална база данни , съдържаща информация за студенти. Данните, които биват запазвани са факултетен номер, две имена, както и оценка. Базата данни , също така, може да бъде модифицирана по всяко време, като към нея могат да се добавят нови студенти, да се изтриват студенти , както и да се търси информация за студент по даден факултетен номер.

Структура на документацията :
1. Обща архитектура – чрез диаграма;
2. Подробно описание на класовете, техните методи и цялостната идея на архитектурата .



Класът Студент притежава 4 член-данни. Целочислена, която пази факултетния номер на даден студент, два динамични масива от тип char , в които се пазят двете имена на студента , както и още една целочислена променлива, която е за оценка. Също така , класът е и канонично представен. Класъс притежава сетър за оценката и гетър за факултетния номер, които биват използвани в реализирането на следващия клас от йерархията. Има и метод , който принтира данните за студента (void printStudent()) , както и предефиниран оператор „ «“ за по - лесно изписване на конзолата.

По – интересните неща се случват в класът StudentList. Той съдържа динамичен масив от тип Student, както ин две целочислени променливи , които ни помагат да оперираме с масивът от студенти. Едната променлива (cap) пази големината на нашият динамичен масив. Втората (curr) пази информация за това колко студенти са записани в масива. Чрез тези две променливи се осъществява и възможността за безкрайно разширяване на масива => неограничен брой студенти.

В private секцията , класът има и метод , който се казва expand. Този метод се вика, ако curr == cap. Ако условието е вярно , това значи , че в масивът вече няма свободни места и затова се вика методът expand , който „разширява“ масива. Това се постига, чрез създаване на нов динамичен масив от тип Студент с cap места. След което , данните , запазени в стария масив се копират в новия. Старият се изтрива , а на негово място се създава нов , с размер cap + 5. Информацията за студентите още веднъж се прехвърля в повторно създаденият масив , с по- голяма размерност, след което буферният масив се трие.

Класът е канонично представен , но има и допълнителен конструктор, който приема файл. Този констуктор играе ключова роля , защото чрез него се зарежда цялата база с данни. За начало, слага стойност на curr – 0 , а на cap – 5. След това проверява дали файлът е отворен, и ако е , проверява дали е празен, чрез отделно дефинирана функция bool isEmpty(ifStream&) , която ще опишем малко по – късно. Ако файлът е отворен и не е празен , тогава конструктурът започва да запълва масивът от студенти с информация от файла. Върху самата база данни , може да пише само програмата, следователно форматът ще е винаги един и същ, а именно - <факултетен номер> <Име> <Фамилия> <Оценка>. След като знаем , как са представени данните във нашия файл , съвсем лесно можем да ги запишем в 4 променливи, от които да създадем обект от тип студент, който да добавим в нашия масив от студенти. Тази процедура се повтаря докато не се достигне до края на файла. При всяко добавяне на нов студент , curr += 1, както и всеки път се прави проверка, дали има достатъчно мяст в масива. Ако няма се вика методът expand.

Следващият метод на класа е addStudent(int char* char* int): Този метод създава Студент от подадените му параметри , след което новосъздаденият студент се добавя в масива . Отново се прави проверка , дали има място в масива . Ако няма се вика expand(); методът bool isPresent(int) приема цяло число с идеята че това е факултетен номер. След което минава през целия масив от студенти и проверява дали съществува студент с факултетен номер , равен на подадения в метода. Ако има , метода връща `true . Ако не успее да намери такъв студент, методът връща false.

Следващият метод int returnStudentIfPresent(int) върши подобна работа, като isPresent, но този път , ако намери съвпадение във факултетните номера, връща позицията на студента в масива. Ако не намери съвпадение, връща -1

void printStudent(int n) принтира студентът, който се намира на n-тото място в масива от студенти.

В класът има и метод editStudent който приема две цели числа. Първото число е позицията на студента , а второто – новата му оценка. Методът променя оценката на даден студент от масива.

DeleteStudent , приема едно цяло число , което е позицията на Студента, който искаме да изтрием, в масива. След което „изтрива“ избраният студент. Това става чрез създаване на нов масив, в който се копират всички студенти , но се пропуска избрания. След което

оригиналният масив от студенти бива изтрит , а на негово място се създава нов, в който се поставя цялата информация от буферния масив, в който вече избраният студент не присъства. След което буферният масив също се изтрива.

Последният метод на този клас е void writeToDataBase(ofstream&). Този метод приема като аргумент, отворена , базата от данни , след което изтрива цялото и съдържание и записва новата , вече променена база. Фактически целият клас StudentList играе ролята на нашата база данни , само че тази , върху която правим промени. След като всички промени са завършени, те се записват на файла.

Има и няколко отделни, автономни функции , които се извикват, след като потребителят избере една от опциите. Те са следните :

- void printMenu(): Тази функция просто отпечатва на конзолата наборът от възможности, измежду които потребителят трябва да избере.
- void create(StudentList&) - Тази функция се извиква , след като потребителят е решил да добави нов студент в базата данни. След избора си , от потребителя ще бъде поискано да въведе уникален ФН на новия студент, последвано от име, фамилия и оценка. При грешно подадени данни , програмата изисква повторно въвеждане. След като всички изискани данни са въведени успешно се създава нов студент с тези данни , който се добавя в масива.
- void sequentialSearch(StudentList&) - Тази функция се извиква, когато потребителят избере опцията да търси студент. След извикването се изисква въвеждането на факултетния номер на студент, и ако той съвпадне с някой от запазените номера на студентите, студентът притежател бива принтиран на конзолата. Ако не съществува студент с такъв факултетен номер, на конзолата се изписва подобаващо съобщение. Търсенето се осъществява чрез метода на StudentList - returnStudentIfPresent.
- void update(StudentList&) - Ако потребителят избере да променя оценката на някой от студентите , или да го „ъпдейтне“ , тази функция се извиква. Тя изисква от потребителя да въведе факултетен номер , последван от новата оценка. След което търси , в масива, дали въведеният ФН ще съвпадне с този на някой студент. Ако да – оценката на студента се променя. Ако ли не , на конзолата се изписва подобаващо съобщение.
- void deleteStudent(StudentList&) - Тази функция , както подсказва името, се извиква , когато потребителят иска да изтрие някой от студентите в базата данни. Изисква от потребителя да се въведе ФН на студента , който иска да изтрие. След което претърсва масива от студенти за съвпадение на факултетните номера, използвайки метода returnStudentIfPresent. Ако има такова се извиква методът void deleteStudent(int) , с аргумент, номера на студента в масива. Този индекс е върнат от извиканията преди това метод. Ако няма съвпадение се изписва подобаващо съобщение.

Последната възможност, която се предоставя на потребителя е Save and exit. При избиране на тази опция, файлът се отваря за output, след което се извиква методът writeToDataBase(с аргумент, отворената база данни). Модифицираните данни се записват на файла, след което той се затваря и програмата се прекратява.

При всяко едно подаване на информация от потребителя, програмата следи дали данните, които са подадени , са валидни. Ако не са, се поисква повторно въвеждане.