ALGORITMI E CALCOLATORI A.A. 2020/2021

**GESTIONE UNIVERSITÀ E PROGRAMMAZIONE ESAMI**

Il seguente progetto si pone l'obiettivo di occuparsi della gestione delle risorse e dei dati inerenti l'università con annessa allocazione delle giornate di esame. Infatti la problematica principale nella gestione di una università è proprio quella che riguarda la gestione dell’allocazione delle giornate di esame. Tale gestione infatti richiede che, a fronte delle regole di ateneo, si organizzino le date degli esami in modo da non violare alcun regolamento interno all'università stessa.

Tale progetto si basa infatti sulla creazione e definizione di una base dati iniziale che, mediante dei comandi da terminale, può essere modificata mediante l’inserimento di nuovi dati o l’aggiornamento di quelli già esistenti.

Spiegazione del fatto che si hanno dei file di input come database, delle mappe in memoria e dei file di output.

Spiegazione TOP-DPWN delle classi: università - corsi di studio – corsi – specific year – studenti – professori

citando l’EREDITARIETA’

Spiegazione Parse

Spiegazione eccezioni

I metodi della classe “University” vengono riportati di seguito:

* **readStudents**

prototipo: void University::readStudents()

Questo metodo si occupa della lettura del database degli studenti denominato: db\_studenti.txt .

Dopo aver aperto il file in modalità lettura ed eseguiti i relativi controlli, al fine di lanciare una eccezione in caso di mancata apertura del file stesso, grazie all’uso di un while ci si assicura di leggere tutte le righe del file, su ognuna delle quali vengono eseguite le operazioni di seguito spiegate. In particolare la lettura di ogni riga viene fatta mediante il metodo getline. Tale riga, salvata in un oggetto stringa di appoggio “line”, viene passata, insieme al carattere separatore “ ; “ alla funzione “splittedLine” la quale restituisce un oggetto vector di stringhe detto “InteroStudente” in modo che ogni elemento del vector contenga l’informazione presente nella riga compresa tra due caratteri separatori “ ; ”. Successivamente mediante un oggetto stringstream (ovvero un oggetto stream che consente di effettuare delle operazioni sulle stringhe) viene salvata solo la matricola dello studente, precedentemente contenuta insieme al carattere “s” nel primo elemento di “InteroStudente”.

Infine si controlla se la matricola associata allo studente corrente è già esistente in memoria; in tal caso viene lanciata una eccezione di avviso, diversamente si procede con il salvataggio nella mappa in memoria dello studente e delle relative informazioni contenute nei diversi campi del vector “InteroStudente”:

* “InteroStudente[0]” contiene s-matricola;
* “InteroStudente[1]” contiene uno o più nomi dello studente (in quanto la funzione splittedLine è stata applicata rispetto al carattere “ ;”;
* “InteroStudente[2]” contiene uno o più cognomi dello studente (in quanto la funzione splittedLine è stata applicata rispetto al carattere “ ;”;
* “InteroStudente[3]” contiene l’indirizzo e-mail dello studente;

La chiave della mappa degli studenti è quindi la matricola dello studente (privata del carattere “s”) in quanto la matricola è univoca per ogni studente.

Il metodo si conclude con la chiusura del file database degli studenti “db\_studenti.txt”.

* **readProfessor**

prototipo: void University::readProfessor()

Questo metodo si occupa della lettura del database dei professori denominato: db\_professori.txt .

Tale metodo si sviluppa in modo analogo al precedente degli studenti.

Si riportano di seguito le informazioni contenute nei diversi campi del vector “InteroProfessore”:

* “InteroProfessore [0]” contiene d-matricola (anche in questo caso il carattere “d”;
* “InteroProfessore [1]” contiene uno o più nomi del professore (in quanto la funzione splittedLine è stata applicata rispetto al carattere “ ;”;
* “InteroProfessore [2]” contiene uno o più cognomi del professore (in quanto la funzione splittedLine è stata applicata rispetto al carattere “ ;”;
* “InteroProfessore [3]” contiene l’indirizzo e-mail del professore;

La chiave della mappa dei professori è quindi la matricola del professore (privata del carattere “d”) in quanto la matricola è univoca per ogni professore.

Il metodo si conclude con la chiusura del file database dei professori “db\_ professori.txt”.

* **readClassroom**

prototipo: void University::readClassroom()

Questo metodo si occupa della lettura del database delle aule denominato: db\_aule.txt .

Tale metodo si sviluppa in modo analogo al precedente degli studenti.

Si riportano di seguito le informazioni contenute nei diversi campi del vector “InteraClasse”:

* “InteraClasse [0]” contiene A-codice identifico dell’aula;
* “InteraClasse [1]” contiene la stringa che specifica il tipo di aula: “A” in caso di aula, “L” in caso di laboratorio;
* “InteraClasse [2]” contiene il nome dell’aula;
* “InteraClasse [3]” contiene la capienza massima dell’aula;
* “InteraClasse [4]” contiene la capienza massima dell’aula se adibita per un esame;

La chiave della mappa delle aule è quindi la matricola delle aule (privata del carattere “A”) in quanto la matricola è univoca per ogni aula.

Il metodo si conclude con la chiusura del file database dei professori “db\_ aule.txt”.

* **readStudyCourse**

prototipo: void University::readStudyCourse()

Questo metodo si occupa della lettura del database dei corsi di studio denominato: db\_corsi\_studio.txt .

Dopo aver aperto il file in modalità lettura ed eseguiti i relativi controlli, al fine di lanciare una eccezione in caso di mancata apertura del file stesso, grazie all’uso di un while ci si assicura di leggere tutte le righe del file, su ognuna delle quali vengono eseguite le operazioni di seguito spiegate. In particolare la lettura di ogni riga viene fatta mediante il metodo getline. Tale riga, salvata in un oggetto stringa di appoggio “line”, viene passata, insieme al carattere separatore “ ; “ alla funzione “splittedLine” la quale restituisce un oggetto vector di stringhe detto “InteroCorsoDiStudi” in modo che ogni elemento del vector contenga l’informazione presente nella riga compresa tra due caratteri separatori “ ; ”. Successivamente mediante un oggetto stringstream viene salvato solo il codice identificativo del corso di studio, precedentemente contenuto insieme al carattere “C” nel primo elemento di “InteroCorsoDiStudi”.

In particolare vengono riportate di seguito le informazioni contenute nel vector “InteroCorsoDiStudi”:

* “InteroCorsoDiStudi [0]” contiene C-codice identificativo del corso di studio;
* “InteroCorsoDiStudi [1]” contiene le due stringhe che specificano il tipo di corso di studio: BS in caso di laurea triennale, MS in caso di laurea magistrale;
* “InteroCorsoDiStudi [2]” contiene l’elenco dei codici identificativi dei diversi corsi raggruppati per semestre all’interno delle parentesi graffe;
* “InteroCorsoDiStudi [3]” contiene l’elenco dei codici identificativi dei corsi spenti separati da una virgola;

Visto il contenuto dell’elemento “InteroCorsoDiStudi [2]” nasce l’esigenza di separare i diversi corsi per ogni semestre. Si usa la funzione posCurlyBrackets la quale, passandole “InteroCorsoDiStudi [2]”, restituisce un vector di interi denominato “posSem” che contiene le diverse posizioni che occupano le parentesi graffe (aperte e chiuse) all’interno di “InteroCorsoDiStudi [2]”, posizione intesa come numero di caratteri che, a partire dal primo presente in “InteroCorsoDiStudi [2]”, si trova la parentesi graffa aperta o chiusa.

A questo punto, grazie ad un ciclo for, per ogni parentesi graffa aperta, che vista la formattazione del file “db\_corsi\_studio.txt“ si trovano nelle posizioni pari del vector “posSem” (motivo per cui l’indice del for viene incrementato di due ad ogni ciclo), si eseguono le seguenti operazioni :

1. si prende la posizione della parentesi graffa aperta, la si salva in una variabile di appoggio “posStart”;
2. si calcola la distanza “len” tra la posizione della “{“ e la posizione della successiva “}”; distanza intesa sempre in termini di caratteri presenti tra le due parentesi;
3. tramite il metodo delle stringhe “substr” si copiano in una stringa tutti i caratteri presenti nella stringa “InteroCorsoDiStudi [2]” che vanno dalla posizione presente in “posStart” (ovvero quella della parentesi graffa aperta) per un numero di caratteri pari al valore contenuto nella variabile “len”; il metodo “substr” restituisce così una stringa contenente i codici di tutti i corsi di un solo semestre; tale stringa viene poi copiata in coda, per ogni ciclo for, nel vector “semestri”.

Alla fine di questo for il vector “semestri” conterrà i corsi di ogni semestre disposti al suo interno in modo che ogni campo di “semestri” contiene tutti i corsi di un certo semestre nel seguente modo:

* “semestri [0]” contiene tutti i corsi del 1o anno 1o semestre;
* “semestri [1]” contiene tutti i corsi del 1o anno 2o semestre;
* “semestri [2]” contiene tutti i corsi del 2o anno 1o semestre;
* “semestri [3]” contiene tutti i corsi del 2o anno 2o semestre;
* “semestri [4]” contiene tutti i corsi del 3o anno 1o semestre;
* “semestri [5]” contiene tutti i corsi del 3o anno 2o semestre;

Si fa notare che il vector ”semestri” non è detto a priori che sia composto da sei campi in quanto questo dipende dal tipo di percorso di studio (triennale o magistrale) e anche dal numero di corsi presenti nel file “db\_corsi\_studio.txt”; quello appena riportato pertanto era solo un esempio generale con fine esplicativo.

Nel caso in cui la linea letta da file, “line”, contenga anche i corsi spenti, ovvero nel caso in cui il vector “InteroCorsoDiStudi” sia composto da un numero di campi pari a quattro (motivo per cui è presente un if con controllo sulla dimensione del vector pari a 4) allora si copiano in una stringa di appoggio mediante il metodo delle stringhe ”substr” tutti i caratteri presenti nel quarto campo di “InteroCorsoDiStudi” che vanno dal secondo carattere (in quanto il primo, vista la formattazione del file, è una parentesi quadra aperta) e fino al penultimo (in quanto l’ultimo, vista la formattazione del file, è una parentesi quadra chiusa); a questo punto l’elenco dei corsi spenti privato delle parentesi quadre (aperta e chiusa) viene passato alla funzione “splittedLine” insieme al carattere separatore, che in questo caso è la virgola “,”, la quale pertanto restituisce pertanto tutti i corsi spenti posizionati ognuno in un campo diverso del vector di stringhe “corsiSpenti”.

Dopo aver letto dalla riga tutte le informazioni inerenti un determinato corso di studio e sistemate in delle apposite variabili si passa alla creazione di un oggetto di appoggio denominato “SCourse” passando al relativo costruttore il codice identificativo del corso di studio e l’informazione inerente il tipo di corso di studio, ovvero percorso triennale o magistrale.

Con l’ausilio del metodo “addSemesterCourses”, appositamente creato per gli oggetti della classe “StudyCourse” (come dimostra il file “StudyCourse.h”), si riempie la mappa privata interna dei semestri dell’oggetto di appoggio poc’anzi creato, “SCourse”, la quale contiene i diversi corsi per ogni di semestre di ogni anno; pertanto mediante un ciclo for, per ogni stringa del vector “semestri” si procede come spiegato di seguito:

1. Si calcola l’anno associato al corso/i presente nella i-esima posizione in “semestri”; siccome il vector “semestri” contiene i corsi di ogni semestre disposti al suo interno in ordine crescente per semestre ovvero partendo dal 1o semestre del 1o anno fino al 2o semestre del 3o (o fino al 2o semestre del 2o anno in caso di laurea magistrale) è facile dedurre come l’anno sia pari al numero del semestre (incrementato di 1 perché la numerazione degli indici in un vector parte da 0) diviso 2 (in quanto ogni anno contiene due semestri);
2. Analogamente si calcola il semestre (se 1o o 2o) associato all’i-esimo corso/i che pertanto risulta essere uguale al resto dell’operazione che ha portato al calcolo dell’anno;
3. Si riempie la mappa privata dell’oggetto “SCourse” grazie al metodo “addSemesterCourses” al quale viene passato: l’anno associato all’i-esimo corso/i, il semestre (1o o 2o) associato all’i-esimo corso/i ed infine l’i-esimo corso/i;

Successivamente in caso di presenza di corsi spenti attraverso il metodo “addOffCourses”, appositamente creato per gli oggetti della classe “StudyCourse” (come dimostra il file “StudyCourse.h”), si riempie la lista di stringhe privata interna dei semestri spenti dell’oggetto di appoggio “SCourse”.

Come ultima operazione si riempie la mappa interna privata denominata “\_studyCourse” della classe “University” (rispettando la convenzione dell’anteporre un underscore al nome dei dati privati di una classe) con il metodo “insert” al quale viene passato come chiave della mappa il codice del corso e come value corrispondente l’oggetto di appoggio “SCourse”.

Quanto spiegato si ricorda essere fatto per ogni riga del file dal momento che tali operazioni sono tutte presenti nel while inizialmente citato di tale metodo.

Tale metodo si conclude con la chiusura del file database dei corsi di studio “db\_corsi\_studio.txt”.

* **readCourse**

prototipo: void University::readCourse()

Questo metodo si occupa della lettura del database dei corsi denominato: db\_corsi.txt .

Dopo aver aperto il file in modalità lettura ed eseguiti i relativi controlli, al fine di lanciare una eccezione in caso di mancata apertura del file stesso, grazie all’uso di un while ci si assicura di leggere tutte le righe del file, su ognuna delle quali vengono eseguite le operazioni di seguito spiegate. In particolare la lettura di ogni riga viene fatta mediante il metodo getline. Tale riga, salvata in un oggetto stringa di appoggio “line”, viene passata, insieme al carattere separatore “ ; “ alla funzione “splittedLine” la quale restituisce un oggetto vector di stringhe detto “interoCorso” in modo che ogni elemento del vector contenga l’informazione presente nella riga compresa tra due caratteri separatori “ ; ”.

Il file “db\_corsi.txt” presenta una struttura secondo la quale:

* ogni riga che comincia con il carattere “c” contiene le info generali di quel corso;
* ogni riga che comincia con il carattere “a” contiene le info di quel corso specifiche per un certo anno accademico;

Pertanto si controlla il primo elemento del vector “interoCorso” (ovvero “interoCorso [0]”) e nel caso in cui fosse uguale alla stringa “c” si passa all’elaborazione delle info generali in particolare a seguire vengono riportate le info presenti nei diversi campi del vector “interoCorso”:

* “interoCorso [1]” contiene il codice identificativo del corso salvato poi in una variabile di appoggio denominata “lastReadCourse”;
* “interoCorso [2]” contiene il titolo del corso;
* “interoCorso [3]” contiene il numero di cfu del corso;
* “interoCorso [4]” contiene il numero di ore di lezione del corso;
* “interoCorso [5]” contiene il numero di ore di esercitazione del corso;
* “interoCorso [6]” contiene il numero di ore di laboratorio del corso;

Si riempie in seguito la mappa interna privata denominata “\_courses” della classe “University” (rispettando la convenzione dell’anteporre un underscore al nome dei dati privati di una classe) con il metodo “insert” al quale viene passato come chiave della mappa il codice del corso (contenuto nella variabile “lastReadCourse” e come value corrispondente l’oggetto della classe “Course”, inizializzato mediante costruttore al quale vengono passati (come da definizione presente nel file “Course.h”) i seguenti dati:

* codice identificativo del corso contenuto in “interoCorso [1]”;
* titolo del corso contenuto in “interoCorso [2]”;
* numero di cfu del corso contenuto in “interoCorso [3]” (convertito da stringa ad intero mediante il metodo delle stringhe stoi);
* numero delle ore di lezione del corso contenuto in “interoCorso [4]” (convertito da stringa ad intero mediante il metodo delle stringhe stoi);
* numero delle ore di esercitazione del corso contenuto in “interoCorso [5]” (convertito da stringa ad intero mediante il metodo delle stringhe stoi);
* numero di ore di laboratorio del corso contenuto in “interoCorso [6]” (convertito da stringa ad intero mediante il metodo delle stringhe stoi);

Nel caso in cui il primo elemento del vector “interoCorso” (ovvero “interoCorso [0]”) contenga la stringa “a” allora si passa all’elaborazione dei dati inerenti uno specifico anno accademico.

Per migliorare la leggibilità del codice si copia il vector “interoCorso” in un altro chiamato “SpecificYearCourse”. Pertanto tale vector presenta per quello specifico anno accademico le seguenti informazioni:

* “SpecificYearCourse [1]” contiene l’anno accademico del corso (anno inizio – anno fine);
* “SpecificYearCourse [2]” contiene l’info se tale corso è attivo/non attivo;
* “SpecificYearCourse [3]” contiene il numero di versioni in parallelo;
* “SpecificYearCourse [4]” contiene l’elenco degli id dei professori che insegnano quel corso quel particolare anno, compreso il professore titolare, e relative ore di lezione, laboratorio ed esercitazione proprio come riportato nel file;
* “SpecificYearCourse [5]” contiene le info inerenti l’esame di quel corso proprio come sono scritte nel file;
* “SpecificYearCourse [6]” contiene l’elenco dei codici dei corsi raggruppati ovvero dei corsi che si trovano nello stesso anno accademico insieme a quello in esame;

Si salva in una variabile di appoggio denominata “acYear” l’anno accademico contenuto ora in “SpecificYearCourse [1]”; si controlla se quel determinato corso in quell’anno fosse attivo o meno andando a confrontare il contenuto del campo “SpecificYearCourse [2]” con la stringa “attivo”; nel caso in cui lo è viene settata come true la variabile di appoggio “isActive” che tiene appunto conto dello stato del corso; si procede analogamente per il numero di versioni in parallelo contenute nella stringa “SpecificYearCourse [3]” e poi salvate in una variabile intera mediante il metodo di conversione delle stringhe stoi.

Visto il contenuto di “SpecificYearCourse [4]” il quale contiene i diversi dati proprio come scritti nel file (ovvero con caratteri delimitatori come parentesi graffe, quadre e virgole) nasce l’esigenza di togliere le parentesi quadre motivo per cui tale vector viene passato al metodo delle stringhe substr che copia nella stringa “profSenzaQuadre” tutti i caratteri presenti in “SpecificYearCourse [4]” dal secondo al penultimo andando ad escludere così il primo e l’ultimo, rispettivamente “[“ e “]”.

Successivamente si passa il vector “profSenzaQuadre” alla funzione getProfPar che restituisce il vector di stringhe “profCorsoPar” composta da un numero di campi pari al numero di versioni in parallelo dove ogni campo contiene quindi tutti i professori, compreso il titolare, che insegnano quel determinato corso parallelo.

In seguito vengono processate le informazioni riguardanti gli esami contenute in “SpecificYearCourse [5]” copiandole, private delle parentesi graffe aperte e chiuse (mediante il metodo substr) in un vector di stringhe di appoggio detto “examData”. Passando quest’ultimo alla funzione splittedLine insieme al carattere separatore “,” si ottiene il vector “splittedExamData” composto dalle seguenti voci:

* “splittedExamData [0]” contiene la durata dell’esame in minuti;
* “splittedExamData [1]” contiene i minuti necessari per accedere all’esame;
* “splittedExamData [2]” contiene i minuti necessari per abbandonare il luogo dell’esame una volta terminato;
* “splittedExamData [3]” contiene la tipologia di esame “S”/”O” (ovvero scritto/orale);
* “splittedExamData [4]” contiene la tipologia di locazione in cui verrà sostenuto l’esame ”A”/”L” (ovvero aula/laboratorio);

In seguito si prende la stringa contente l’elenco dei codici dei corsi raggruppati contenuti in “SpecificYearCourse [6]” privandoli delle parentesi graffe aperte e chiuse sempre mediante il metodo substr, copiando della stringa tutti i caratteri esclusi il primo (ovvero “{“ ) e l’ultimo (ovvero "}”). La stringa risultante viene passato alla funzione “splittedLine” insieme al carattere separatore che in questo caso è “,”; si ottiene quindi un vector di stringhe “idGrouped”.

QUI MANCA UN PEZZO … vedi messaggi telegram

Quanto spiegato si ricorda essere fatto per ogni riga del file dal momento che tali operazioni sono tutte presenti nel while inizialmente citato di tale metodo, tenendo conto che possono esserci per uno stesso anche due o più righe che iniziano con la stringa “c” ovvero più anni accademici inerenti quello specifico corso.

Tale metodo si conclude con la chiusura del file database dei corsi di studio “db\_corsi.txt”.

* **addStuds**

prototipo: bool University::addStuds(const std::string &fileIn)

Questo metodo si occupa dell’inserimento dei nuovi studente (quindi privi di matricola) nella mappa in memoria degli studenti. Tali studenti sono riportati nel file il cui nome viene passato nel parametro “fileIn” del metodo stesso.

Dopo aver aperto il file in modalità lettura ed eseguiti i relativi controlli, al fine di lanciare una eccezione in caso di mancata apertura del file stesso (restituendo un booleano pari a false), grazie all’uso di un while ci si assicura di leggere tutte le righe del file, su ognuna delle quali vengono eseguite le operazioni di seguito spiegate. In particolare la lettura di ogni riga viene fatta mediante il metodo getline. Tale riga, salvata in un oggetto stringa di appoggio “line”, viene passata, insieme al carattere separatore “ ; “ alla funzione “splittedLine” la quale restituisce un oggetto vector di stringhe detto “tokens”. Si controlla che tale vector contenga tre campi ovvero che siano state riportate nella riga per quel determinato studente tutte e tre le info necessarie per salvarlo in memoria ovvero: nome, cognome ed email. In caso in cui il numero di campi dovesse essere diverso da tre viene lanciata una eccezione in cui si specifica il numero corrispondente alla riga in cui si è verificato l’errore (tale variabile contatore che tiene conto del numero di linee lette viene incrementata ogni volta che si rientra nel ciclo del while ed è dichiarata sotto il nome di “line\_counter”).

Nel caso in cui la riga contenga tutte le informazioni necessarie si procede con l’ottenere il codice univoco identificativo per il nuovo studente da inserire mediante l’ausilio dell’apposito metodo “getNewStudentId” creato per la classe “University” (tale metodo viene descritto nel dettaglio nelle righe avvenire).

Ottenuta la matricola dello studente salvata nella variabile “matr”, mediante il metodo insert, si procede con il salvataggio nella mappa in memoria dello studente e delle relative informazioni contenute nei diversi campi del vector “tokens”:

* “tokens [0]” contiene uno o più nomi dello studente (in quanto la funzione splittedLine è stata applicata rispetto al carattere “ ;”;
* “tokens [1]” contiene uno o più cognomi dello studente (in quanto la funzione splittedLine è stata applicata rispetto al carattere “ ;”;
* “tokens [2]” contiene l’indirizzo e-mail dello studente;

La chiave della mappa degli studenti è quindi la matricola dello studente in quanto univoca per ogni studente.

Quanto spiegato si ricorda essere fatto per ogni riga del file dal momento che tali operazioni sono tutte presenti nel while inizialmente citato di tale metodo.

Dopo la chiusura del file il metodo si conclude con la riscrittura automatizzata del file database degli studenti db\_studenti.txt, dal momento che nuovi studenti sono stati aggiunti nella base dati, il tutto attraverso il metodo dbStudsWrite (che verrà spiegato solo per puro ordine espositivo nelle righe avvenire).

Tale metodo restituisce, come si può notare dal prototipo, un valore true se giunti sino alla riscrittura del file database.

* **addProfessors**

prototipo: bool University::addProfessors(const std::string &fileIn)

Questo metodo si occupa dell’inserimento dei nuovi professori (quindi privi di matricola) nella mappa in memoria dei professori. Tali professori sono riportati nel file il cui nome viene passato nel parametro “fileIn” del metodo stesso.

Tale metodo si sviluppa in modo analogo al precedente degli studenti.

Si riportano di seguito le informazioni contenute nei diversi campi del vector “tokens”:

* “tokens [0]” contiene uno o più nomi del professore (in quanto la funzione splittedLine è stata applicata rispetto al carattere “ ;”;
* “tokens [1]” contiene uno o più cognomi del professore (in quanto la funzione splittedLine è stata applicata rispetto al carattere “ ;”;
* “tokens [2]” contiene l’indirizzo e-mail del professore;

La chiave della mappa degli studenti è quindi la matricola dello studente in quanto univoca per ogni studente.

Dopo la chiusura del file il metodo si conclude con la riscrittura automatizzata del file database dei professori db\_professori.txt, dal momento che nuovi professori sono stati aggiunti nella base dati, il tutto attraverso il metodo dbProfsWrite (che verrà spiegato solo per puro ordine espositivo nelle righe avvenire).

Tale metodo restituisce, come si può notare dal prototipo, un valore true se giunti sino alla riscrittura del file database.

* **addClassrooms**

prototipo: bool University::addClassrooms(const std::string &fileIn)

Questo metodo si occupa dell’inserimento di nuove aule (quindi prive di codice identificativo) nella mappa in memoria delle aule. Tali aule sono riportate nel file il cui nome viene passato nel parametro “fileIn” del metodo stesso.

Tale metodo si sviluppa in modo analogo al precedente degli studenti.

Si riportano di seguito le informazioni contenute nei diversi campi del vector “tokens”:

* “tokens [0]” contiene la stringa che specifica il tipo di aula: “A” in caso di aula, “L” in caso di laboratorio;
* “tokens [1]” contiene il nome dell’aula;
* “tokens [2]” contiene la capienza massima dell’aula;
* “tokens [3]” contiene la capienza massima dell’aula se adibita per un esame;

La chiave della mappa delle aule è quindi il codice identificativo in quanto univoco per ogni aula.

Dopo la chiusura del file il metodo si conclude con la riscrittura automatizzata del file database delle aule db\_aule.txt, dal momento che nuove aule sono state aggiunte nella base dati, il tutto attraverso il metodo dbClassRoomWrite (che verrà spiegato solo per puro ordine espositivo nelle righe avvenire).

Tale metodo restituisce, come si può notare dal prototipo, un valore true se giunti sino alla riscrittura del file database.

* **addStudyCourses**

prototipo: bool University::addStudyCourses(const std::string &fin)

Questo metodo si occupa dell’inserimento dei nuovi corsi di studio (quindi privi di codice identificativo) nella mappa in memoria dei corsi di studio. Tali corsi di studio sono riportati nel file il cui nome viene passato nel parametro “fin” del metodo stesso.

Dopo aver aperto il file in modalità lettura ed eseguiti i relativi controlli, al fine di lanciare una eccezione in caso di mancata apertura del file stesso (restituendo in tal caso un booleano pari a false), grazie all’uso di un while ci si assicura di leggere tutte le righe del file, su ognuna delle quali vengono eseguite le operazioni di seguito spiegate. In particolare la lettura di ogni riga viene fatta mediante il metodo getline. Tale riga, salvata in un oggetto stringa di appoggio “line”, viene passata, insieme al carattere separatore “ ; “ alla funzione “splittedLine” la quale restituisce un oggetto vector di stringhe detto “tokens”. Si controlla che tale vector contenga due campi ovvero che siano state riportate nella riga per quel determinato corso di studio tutte e due le info necessarie per salvarlo in memoria ovvero: il tipo di corso di studio (ovvero le due stringhe che specificano il tipo di corso di studio: BS in caso di laurea triennale, MS in caso di laurea magistrale) e l’elenco dei corsi presenti in tale percorso di studio. In caso in cui il numero di campi dovesse essere diverso da due viene lanciata una eccezione in cui si specifica il numero corrispondente alla riga in cui si è verificato l’errore (tale variabile contatore che tiene conto del numero di linee lette viene incrementata ogni volta che si rientra nel ciclo del while ed è dichiarata sotto il nome di “line\_counter”).

Nel caso in cui la riga contenga tutte le informazioni necessarie si procede con l’ottenere il codice univoco identificativo per il nuovo corso di studio da inserire mediante l’ausilio dell’apposito metodo “getStudyCourseId” creato per la classe “University” (tale metodo viene descritto nel dettaglio nelle righe avvenire).

Ottenuta la matricola del corso di studio salvata nella variabile “codCorso”, visto il valore della stringa “tokens [1]” (contenente l’elenco dei codici identificativi dei diversi corsi raggruppati per semestre all’interno delle parentesi graffe) nasce l’esigenza di separare i diversi corsi per ogni semestre. Si usa il metodo delle stringhe find\_first\_of il quale, relativamente alla stringa “tokens [1]”, passandogli i due caratteri “{“ e “}”, restituisce un intero (salvato nella variabile di appoggio “found”) indicante la prima posizione in corrispondenza della quale si trova nella stringa uno dei due caratteri passati. Dal momento che la find\_first\_of si blocca la primo carattere trovato allora si ripete tale operazione per tutti i rimanenti caratteri di “tokens [1]” successivi a quello appena individuato(motivo per cui nella find\_first\_of è presente come parametro “found+1”) mediante un ciclo while. Le diverse posizioni trovate vengono di volta in volta per ogni ciclo while accodate in un vector di interi detto “posSem” il quale pertanto contiene le diverse posizioni che occupano le parentesi graffe (aperte e chiuse) all’interno di “tokens [1]”, posizione intesa come numero di caratteri che, a partire dal primo presente in “tokens [1]”, si trova la parentesi graffa aperta o chiusa.

A questo punto, grazie ad un ciclo for, per ogni parentesi graffa aperta, che vista la formattazione del file “db\_corsi\_studio.txt“ si trovano nelle posizioni pari del vector “posSem” (motivo per cui l’indice del for viene incrementato di due ad ogni ciclo), si eseguono le seguenti operazioni :

1. si prende la posizione della parentesi graffa aperta, la si salva in una variabile di appoggio “posStart”;
2. si calcola la distanza “len” tra la posizione della “{“ e la posizione della successiva “}”; distanza intesa sempre in termini di caratteri presenti tra le due parentesi;
3. tramite il metodo delle stringhe “substr” si copiano in una stringa tutti i caratteri presenti nella stringa “tokens [1]” che vanno dalla posizione presente in “posStart” (ovvero quella della parentesi graffa aperta) per un numero di caratteri pari al valore contenuto nella variabile “len”; il metodo “substr” restituisce così una stringa contenente i codici di tutti i corsi di un solo semestre; tale stringa viene poi copiata in coda, per ogni ciclo for, nel vector “semestri”.

Alla fine di questo for il vector “semestri” conterrà i corsi di ogni semestre disposti al suo interno in modo che ogni campo di “semestri” contiene tutti i corsi di un certo semestre nel seguente modo:

* “semestri [0]” contiene tutti i corsi del 1o anno 1o semestre (della laurea triennale o magistrale);
* “semestri [1]” contiene tutti i corsi del 1o anno 2o semestre (della laurea triennale o magistrale);
* “semestri [2]” contiene tutti i corsi del 2o anno 1o semestre (della laurea triennale o magistrale);
* “semestri [3]” contiene tutti i corsi del 2o anno 2o semestre (della laurea triennale o magistrale);
* “semestri [4]” contiene tutti i corsi del 3o anno 1o semestre (della laurea triennale);
* “semestri [5]” contiene tutti i corsi del 3o anno 2o semestre (della laurea triennale);

Si controlla a questo punto che, a seconda del tipo laurea, il numero di semestri sia coerente ovvero si controlla che il numero di semestri per una laurea triennale siano sei o che per una laurea triennale siano quattro. Nel caso in cui così non fosse si lancia una eccezione indicando anche la linea in corrispondenza della quale si è verificato l’errore.

Dopo aver letto dalla riga tutte le informazioni inerenti un determinato corso di studio e sistemate in delle apposite variabili si passa alla creazione di un oggetto di appoggio denominato “SCourse” passando al relativo costruttore il codice identificativo del corso di studio e l’informazione inerente il tipo di corso di studio, ovvero percorso triennale o magistrale.

Con l’ausilio del metodo “addSemesterCourses”, appositamente creato per gli oggetti della classe “StudyCourse” (come dimostra il file “StudyCourse.h”), si riempie la mappa privata interna dei semestri dell’oggetto di appoggio poc’anzi creato, “SCourse”, la quale contiene i diversi corsi per ogni di semestre di ogni anno; pertanto mediante un ciclo for, per ogni stringa del vector “semestri” si procede come spiegato di seguito:

1. Si calcola l’anno associato al corso/i presente nella i-esima posizione in “semestri”; siccome il vector “semestri” contiene i corsi di ogni semestre disposti al suo interno in ordine crescente per semestre ovvero partendo dal 1o semestre del 1o anno fino al 2o semestre del 3o (o fino al 2o semestre del 2o anno in caso di laurea magistrale) è facile dedurre come l’anno sia pari al numero del semestre (incrementato di 1 perché la numerazione degli indici in un vector parte da 0) diviso 2 (in quanto ogni anno contiene due semestri);
2. Analogamente si calcola il semestre (se 1o o 2o) associato all’i-esimo corso/i che pertanto risulta essere uguale al resto dell’operazione che ha portato al calcolo dell’anno;
3. Si riempie la mappa privata dell’oggetto “SCourse” grazie al metodo “addSemesterCourses” al quale viene passato: l’anno associato all’i-esimo corso/i, il semestre (1o o 2o) associato all’i-esimo corso/i ed infine l’i-esimo corso/i;

Come ultima operazione si riempie la mappa interna privata denominata “\_studyCourse” della classe “University” (rispettando la convenzione dell’anteporre un underscore al nome dei dati privati di una classe) con il metodo “insert” al quale viene passato come chiave della mappa il codice del corso e come value corrispondente l’oggetto di appoggio “SCourse”.

Quanto spiegato si ricorda essere fatto per ogni riga del file dal momento che tali operazioni sono tutte presenti nel while inizialmente citato di tale metodo.

La chiave della mappa dei corsi di studio è quindi il codice identificativo in quanto univoco per ogni corso di studio.

Dopo la chiusura del file il metodo si conclude con la riscrittura automatizzata del file database dei corsi di studio db\_corsi\_studio.txt, dal momento che nuovi corsi di studio sono stati aggiunti nella base dati, il tutto attraverso il metodo dbStudyCourseWrite (che verrà spiegato solo per puro ordine espositivo nelle righe avvenire).

Tale metodo restituisce, come si può notare dal prototipo, un valore true se giunti sino alla riscrittura del file database.

* **addCourses**

prototipo: bool University::addCourses(const std::string &fin)

Questo metodo si occupa dell’inserimento dei nuovi corsi (quindi privi di codice identificativo) nella mappa in memoria dei corsi. Tali corsi sono riportati nel file il cui nome viene passato nel parametro “fin” del metodo stesso.

Dopo aver aperto il file in modalità lettura ed eseguiti i relativi controlli, al fine di lanciare una eccezione in caso di mancata apertura del file stesso (restituendo in tal caso un booleano pari a false), grazie all’uso di un while ci si assicura di leggere tutte le righe del file, su ognuna delle quali vengono eseguite le operazioni di seguito spiegate. In particolare la lettura di ogni riga viene fatta mediante il metodo getline. Tale riga, salvata in un oggetto stringa di appoggio “line”, viene passata, insieme al carattere separatore “; “ alla funzione “splittedLine” la quale restituisce un oggetto vector di stringhe detto “specificYearCourse”. Si controlla che tale vector contenga dieci campi ovvero che siano state riportate nella riga per quel determinato corso di studio tutte e dieci le info necessarie per salvarlo in memoria ovvero:

1. anno accademico;
2. titolo;
3. numero di cfu;
4. numero di ore destinate alle lezioni;
5. numero di ore destinate alle esercitazioni;
6. numero di ore rivolte ai laboratori;
7. numero di versioni in parallelo esistenti per questo corso;
8. elenco degli id dei professori che insegnano quel corso in quel particolare anno, compreso il professore titolare, e relative ore di lezione, laboratorio ed esercitazione;
9. elenco delle info riguardanti l’esame comprendenti la durata dell’esame, il tempo necessario per accedere all’esame, il tempo necessario per abbandonare l’esame, modalità di svolgimento, luogo in cui si terrà;
10. elenco degli id dei corsi raggruppati a quello attualmente in esame;

In caso in cui il numero di campi dovesse essere diverso da dieci viene lanciata una eccezione in cui si specifica il numero corrispondente alla riga in cui si è verificato l’errore (tale variabile contatore che tiene conto del numero di linee lette viene incrementata ogni volta che si rientra nel ciclo del while ed è dichiarata sotto il nome di “line\_counter”).

Nel caso in cui la riga contenga tutte le informazioni necessarie ci si assicura che il corso che viene aggiunto non esista già nel database motivo per cui si procede con un ciclo for nel quale si scandaglia l’intera mappa dei corsi in memoria mediante l’uso di un iteratore che quindi parte dal primo corso in mappa sino all’ultimo (questo possibile grazie all’uso rispettivamente dei metodi begin ed end rivolti agli iteratori). Nel caso in cui dovesse essere presente in memoria un corso uguale a quello che si vuole caricare (appena letto da file), avente pertanto stesso titolo e stesso numero di cfu (caratteristiche che assieme, in assenza di un id, risultano univoche nell’identificazione di un corso) si procede con il lancio di una eccezione che avvisa l’utente della presenza di un corso già esistente informandolo anche del numero della riga corrispondente a quanto accaduto (ancora una volta mediante l’uso di una variabile contatore, “line\_counter”).

Nel caso in cui tale corso non sia già esistente si procede quindi con l’ottenere il codice univoco identificativo per il nuovo corso da inserire mediante l’ausilio dell’apposito metodo “getNewCourseId” creato per la classe “University” (tale metodo viene descritto nel dettaglio nelle righe avvenire).

Si riempie in seguito la mappa interna privata denominata “\_courses” della classe “University” (rispettando la convenzione dell’anteporre un underscore al nome dei dati privati di una classe) con il metodo “insert” al quale viene passato come chiave della mappa il codice del corso (contenuto nella variabile “newIdCourse” e come value corrispondente l’oggetto della classe “Course”, inizializzato mediante costruttore al quale vengono passati (come da definizione presente nel file “Course.h”) i seguenti dati:

1. titolo del corso contenuto in “specificYearCourse [1]”;
2. numero di cfu del corso contenuto in “interoCorso [2]” (convertito da stringa ad intero mediante il metodo delle stringhe stoi);
3. numero delle ore di lezione del corso contenuto in “interoCorso [3]” (convertito da stringa ad intero mediante il metodo delle stringhe stoi);
4. numero delle ore di esercitazione del corso contenuto in “interoCorso [4]” (convertito da stringa ad intero mediante il metodo delle stringhe stoi);
5. numero di ore di laboratorio del corso contenuto in “interoCorso [5]” (convertito da stringa ad intero mediante il metodo delle stringhe stoi);

Si salva in una variabile di appoggio denominata “acYear” l’anno accademico contenuto ora in “SpecificYearCourse [0]”; il numero di versioni in parallelo contenute nella stringa “SpecificYearCourse [6]” vengono salvate in una variabile intera mediante il metodo di conversione delle stringhe stoi.

Visto il contenuto di “SpecificYearCourse [7]” il quale contiene i diversi dati proprio come scritti nel file (ovvero con caratteri delimitatori come parentesi graffe, quadre e virgole) nasce l’esigenza di togliere le parentesi quadre motivo per cui tale vector viene passato al metodo delle stringhe substr che copia nella stringa “profSenzaQuadre” tutti i caratteri presenti in “SpecificYearCourse [7]” dal secondo al penultimo andando ad escludere così il primo e l’ultimo, rispettivamente “[“ e “]”.

Successivamente si passa il vector “profSenzaQuadre” alla funzione getProfPar che restituisce il vector di stringhe “profCorsoPar” composta da un numero di campi pari al numero di versioni in parallelo dove ogni campo contiene quindi tutti i professori, compreso il titolare, che insegnano quel determinato corso parallelo.

In seguito vengono processate le informazioni riguardanti gli esami contenute in “SpecificYearCourse [8]” copiandole, private delle parentesi graffe aperte e chiuse (mediante il metodo substr) in un vector di stringhe di appoggio detto “examData”. Passando quest’ultimo alla funzione splittedLine insieme al carattere separatore “,” si ottiene il vector “splittedExamData” composto dalle seguenti voci:

* “splittedExamData [0]” contiene la durata dell’esame in minuti;
* “splittedExamData [1]” contiene i minuti necessari per accedere all’esame;
* “splittedExamData [2]” contiene i minuti necessari per abbandonare il luogo dell’esame una volta terminato;
* “splittedExamData [3]” contiene la tipologia di esame “S”/”O” (ovvero scritto/orale);
* “splittedExamData [4]” contiene la tipologia di locazione in cui verrà sostenuto l’esame ”A”/”L” (ovvero aula/laboratorio);

In seguito si prende la stringa contente l’elenco dei codici dei corsi raggruppati contenuti in “SpecificYearCourse [9]” privandoli delle parentesi graffe aperte e chiuse sempre mediante il metodo substr, copiando della stringa tutti i caratteri esclusi il primo (ovvero “{“ ) e l’ultimo (ovvero "}”). La stringa risultante viene passato alla funzione “splittedLine” insieme al carattere separatore che in questo caso è “,”; si ottiene quindi un vector di stringhe “idGrouped”.

QUI MANCA STESSO PEZZO DELLA READCOURSE… vedi messaggi telegram

Come ultima operazione si riempie la mappa interna privata denominata “\_courses” della classe “University” (rispettando la convenzione dell’anteporre un underscore al nome dei dati privati di una classe) in particolare con il metodo “at” si recupera il nuovo corso da aggiungere, mediante il suo id ricavato, e si procede con l’inserimento delle seguenti informazioni inerenti quel corso:

* anno accademico;
* l’informazione booleana che mi dice che il corso è attivo (in quanto appena aggiunto);
* numero di corsi paralleli;
* vector con un numero di campi pari al numero di verisoni parallele del seguente corso; ogni campo comprende quindi le diverse informazioni riguardanti il prof titolare e tutti gli altri prof che insegnano quel determinato corso parallelo con relative ore di insegnamento;
* vector con le Informazioni dell’esame;
* vector con i diversi corsi raggruppati;
* ………YY\_SEMESTER………………. INERENTE IL PEZZO CHE MANCA
* ………STUDYCOURSE………………. INERENTE IL PEZZO CHE MANCA

Quanto spiegato si ricorda essere fatto per ogni riga del file dal momento che tali operazioni sono tutte presenti nel while inizialmente citato di tale metodo.

La chiave della mappa dei corsi di studio è quindi il codice identificativo in quanto univoco per ogni corso di studio.

Dopo la chiusura del file il metodo si conclude con la riscrittura automatizzata del file database dei corsi di studio db\_corsi\_studio.txt, dal momento che nuovi corsi di studio sono stati aggiunti nella base dati, il tutto attraverso il metodo dbCourseWrite (che verrà spiegato solo per puro ordine espositivo nelle righe avvenire).

Tale metodo restituisce, come si può notare dal prototipo, un valore true se giunti sino alla riscrittura del file database stesso.

* **getNewStudentId**

prototipo: const int University::getNewStudentId() const

Tale metodo, come anticipato nei precedenti, si occupa della generazione di un nuovo id (privato del carattere “s”) rivolto agli studenti da aggiungere nel database.

Inizialmente si controlla se il database degli studenti risulti vuoto o meno; qualora la mappa degli studenti in memoria sia vuota tale metodo restituisce il valore 1 al fine di impedire che venga generata una matricola priva di significato pratico come: “s00000”.

Si dichiara un iteratore “last” con deduzione di tipo (ovvero “auto”) che punta all’ultimo studente della mappa.

Si dichiara una variabile denominata “toReturn” la quale recupera il value associato all’iteratore “last” (ovvero la matricola associata all’ultimo studente) e la si incrementa di 1 in quanto si vuole creare una nuova matricola consecutiva all’ultima già presente in memoria.

Infine si ritorna, come mostra il prototipo di tale metodo, un numero intero, nonché la nuova matricola generata.

Si noti come questo metodo è stato marcato come const al fine di impedire che il risultato restituito da tale metodo, ovvero la nuova matricola generata, venga modificato.

* **getNewProfessorId**

prototipo: const int University::getNewProfessorId() const

Tale metodo, come anticipato nei precedenti, si occupa della generazione di un nuovo id (privato del carattere “d”) rivolto ai professori da aggiungere nel database. Tale metodo si sviluppa in modo analogo a quello degli studenti.

Inizialmente si controlla se il database dei professori risulti vuoto o meno; qualora la mappa dei professori in memoria sia vuota tale metodo restituisce il valore 1 al fine di impedire che venga generata una matricola priva di significato pratico come: “d00000”.

Si dichiara un iteratore “last” con deduzione di tipo (ovvero “auto”) che punta all’ultimo professore della mappa.

Si dichiara una variabile denominata “toReturn” la quale recupera il value associato all’iteratore “last” (ovvero la matricola associata all’ultimo professore) e la si incrementa di 1 in quanto si vuole creare una nuova matricola consecutiva all’ultima già presente in memoria.

Infine si ritorna, come mostra il prototipo di tale metodo, un numero intero, nonché la nuova matricola generata.

Si noti come questo metodo è stato marcato come const al fine di impedire che il risultato restituito da tale metodo, ovvero la nuova matricola generata, venga modificato.

* **getNewProfessorId**

prototipo: const int University::getNewClassroomId() const

Tale metodo, come anticipato nei precedenti, si occupa della generazione di un nuovo id (privato del carattere “A”) rivolto alle aule da aggiungere nel database. Tale metodo si sviluppa in modo analogo a quello degli studenti.

Inizialmente si controlla se il database delle aule risulti vuoto o meno; qualora la mappa delle aule in memoria sia vuota tale metodo restituisce il valore 1 al fine di impedire che venga generata un codice identificativo privo di significato pratico come: “A000”.

Si dichiara un iteratore “last” con deduzione di tipo (ovvero “auto”) che punta all’ultima aula della mappa.

Si dichiara una variabile denominata “toReturn” la quale recupera il value associato all’iteratore “last” (ovvero l’id associato all’ultima aula) e la si incrementa di 1 in quanto si vuole creare una nuova matricola consecutiva all’ultima già presente in memoria.

Infine si ritorna, come mostra il prototipo di tale metodo, un numero intero, nonché la nuova matricola generata.

Si noti come questo metodo è stato marcato come const al fine di impedire che il risultato restituito da tale metodo, ovvero la nuova matricola generata, venga modificato.

* **getNewStudyCourseId**

prototipo: const int University::getNewStudyCourseId() const

Tale metodo, come anticipato nei precedenti, si occupa della generazione di un nuovo id (privato del carattere “C”) rivolto ai corsi di studio da aggiungere nel database. Tale metodo si sviluppa in modo analogo a quello degli studenti.

Inizialmente si controlla se il database dei corsi di studio risulti vuoto o meno; qualora la mappa dei corsi di studio in memoria sia vuota tale metodo restituisce il valore 1 al fine di impedire che venga generata un codice identificativo privo di significato pratico come: “C000”.

Si dichiara un iteratore “last” con deduzione di tipo (ovvero “auto”) che punta all’ultimo corso di studio della mappa.

Si dichiara una variabile denominata “toReturn” la quale recupera la chiave associata all’iteratore “last” (ovvero l’id associato all’ultimo corso di studi) e la si incrementa di 1 in quanto si vuole creare una nuova matricola consecutiva all’ultima già presente in memoria.

Infine si ritorna, come mostra il prototipo di tale metodo, un numero intero, nonché la nuova matricola generata.

Si noti come questo metodo è stato marcato come const al fine di impedire che il risultato restituito da tale metodo, ovvero la nuova matricola generata, venga modificato.

* **getNewCourseId**

prototipo: const int University::getNewCourseId() const

Tale metodo, come anticipato nei precedenti, si occupa della generazione di un nuovo codice identificativo rivolto ai corsi da aggiungere nel database. Tale metodo si sviluppa in modo analogo a quello degli studenti.

Si controlla inizialmente se la mappa dei corsi in memoria è vuota o meno; in tal caso l’id da restituire coincide con il primo che si può avere ovvero “01AAAAA”.

Si prende l’ultimo corso presente in memoria (avente quindi l’ultimo id) e lo si salva in un oggetto stringstream detto “Id”. Successivamente si estraggono da quest’ultimo i due campi di cui è composto ovvero una parte numerica detta “num”, costituita da un numero a due cifre, e una parte alfabetica detta “cod” composta da quattro cifre.

Mediante un ciclo for si analizza l’ultima parte dell’id ovvero quella alfabetica iniziando dall’ultimo carattere spostandosi così in ogni ciclo ai caratteri precedenti (in quanto come ogni somma anche quella che opera con dei caratteri prevede di partire dalle posizioni meno significative).

Partendo dall’ultimo carattere si controlla se questo risulta diverso dal carattere “Z”:

* se risulta diverso da “Z” allora si incrementa di uno il carattere in questione, ovvero si passa al carattere successivo in ordine alfabetico e si setta a true un flag booleano detto “noZ” che appena diventa vero provoca al prossimo ciclo l’uscita dal for (in quanto nella condizione del for tale flag negato è in and con l’indice della stringa che deve essere maggiore o uguale a zero, dal momento che l’analisi della stringa è fatta dal fondo verso la testa);
* se risulta uguale a “Z” allora si setta il carattere in questione con quello successivo nell’alfabeto ovvero “A” e si passa con il prossimo ciclo for all’analisi del carattere precedente; pertanto si riparte dal confronto con il carattere “Z”.

Solo nel momento in cui tutti i caratteri della stringa sono pari ad “A” si prosegue con l’incremento di uno della parte numerica, questo viene fatto con un if esterno al for, diversamente si prosegue col conservare il valore vecchio.

Successivamente mediante il metodo setw si decide di scrivere la parte numerica sempre su due cifre anteponendo uno 0 mediante il metodo setfill nel caso in cui la variabile “cod” sia costituita da un numero ad una cifra.

Si copia l’oggetto stringstream in una stringa la quale conterrà pertanto il nuovo codice identifcativo che verrà restituito come mostrato dal prototipo del metodo.

* **updateStuds**

prototipo: bool University::updateStuds(const std::string &fin)

|  |  |  |  |
| --- | --- | --- | --- |
| STUDENTI | SISTEMA OPERATIVO | COMPILATORE | VERSIONE COMPILATORE |
| Andrea Chiapparo |  |  |  |
| Luca Montangero |  |  |  |
| Ricardo Maria Galiano |  |  |  |