|  |
| --- |
| CourseManager |
| set Students;  Student connected\_student; |
| Enum : manager\_result {  MANAGER\_SUCCESS  MANAGER\_INVALID\_PARAMETER  MANAGER\_STUDENT\_ALREADY\_EXISTS  MANAGER\_ALREADY\_LOGGED\_IN  MANAGER\_STUDENT\_DOES\_NOT\_EXIST  MANAGER\_NOT\_LOGGED\_IN  MANAGER\_ALREADY\_FRIEND  MANAGER\_ALREADY\_REQUESTED  MANAGER\_NOT\_REQUESTED  MANAGER\_INVALID\_PARAMETER  } |
| CourseManager createCourseManager();  static Student findStudentById (set Students, int student\_id);  manager\_result addStudent (set Students, int student\_id, char\* first\_name, char\* last\_name);  manager\_result removeStudent (set Students, int student\_id);  manager\_result studentLogIn (CourseManager system, int student\_id);  manager\_result studentlogOut(CourseManager system);  manager\_result studentAddFriendRequest (CourseManager system, int other\_id);  manager\_result studentHandleRequest (CourseManager system, int other\_id, char\* action);  manager\_result studentUnfriend (CourseManager system, int other\_id);  manager\_result studentAddGrade (Student connected\_student, Int semester, Int course\_id, Char\* points, Int grade);  manager\_result studentRemoveGrade (Student connected\_student, int semester, int course\_id);  manager\_result studentUpdateGrade (Student connected\_student, int course\_id, int new\_grade);  manager\_result studentPrintFullReport (Student connected\_student, FILE\* output\_channel);  manager\_result studentPrintCleanReport (Student connected\_student, FILE\* output\_channel);  manager\_result studentPrintBestReport (Student connected\_student, int amount, bool isBest, FILE\* output\_channel);  manager\_result studentAskReference (Student connected\_student, int course\_id, int amount);  manager\_result studentFacultyRequest (Student connected\_student, int course\_id, char\* request); |

|  |
| --- |
| student |
| int student\_id;  Char\* first\_name;  Char\* last\_name;  Set Friends;  List Grades; |
| enum : student\_result {  STUD\_SUCCESS  STUD\_INVALID\_ID  STUD\_ALREADY\_FRIEND  STUD\_ALREADY\_REQUESTED  STUD\_NOT\_REQUESTED  STUD\_NOT\_FRIEND  STUD\_INVALID\_PARAMETERS  STUD\_OUT\_OF\_MEMORY  STUD\_COURSE\_DOES\_NOT\_EXIST  } |
| static Grade findLastGrade (Student student1, Int semester, Int course\_id, Char\* points);  static int findLastSemester (Student student1, int course\_id)  static friendshipStatus checkFriendshipStatus(Set friends1, Friend other\_student1, Friend\* itarator)  static studentResult acceptFriend(Student student1, Friend other\_friend, Student other\_student, friendshipStatus status, Friend iterator)  static studentResult rejectFriend(Student student1, Friend other\_friend, Student other\_student,friendshipStatus status, Friend iterator)  static studentResult addNewFriend(Student student1,Friend other\_friend,  Student other\_student ,friendshipStatus status, Friend itarator)  static studentResult removeFriendShip(Student student1, Friend other\_friend,  Student other\_student, Friend iterator)  Static bool isIdValid (int student\_id);  Static void findAllSemesters (List Grades, int\* all\_semesters);  Static void filterIrrelevantGrades (List copied\_grades);  int summarizeSemester (student student1, int semester, int\* failed\_points, int\* effective\_points, int\* effective\_grades\_sum);  student createStudent (int id, char\* first\_name, char\* last\_name, student\_result\* result);  int compareStudents (Student student1, Student student2);  void destroyStudent (Student student1);  student\_result addFriendRequest (Student student1, Student other\_student);  student\_result handleRequest (Student student1, Student other\_student, char\* action);  student\_result unfriend (Student student1, Student other\_student);  student\_result addGrade (Studet student1, Int semester, Int course\_id, Char\* points, Int grade);  student\_result removeGrade (Student student1, int semester, int course\_id);  student\_result updateGrade (Student student1, int course\_id, int new\_grade);  void printFullReport (Student student1, FILE\* output\_channel);  student\_result printCleanReport (Student student1, FILE\* output\_channel);  student\_result printBestReport (Student student1, bool isBest, FILE\* output\_channel);  student\_result askReference (Student student1, int course\_id, int amount);  student\_result facultyRequest (Student student1, int course\_id, char\* request); |

|  |
| --- |
| **Friend** |
| Student friend\_student;  typedef enum {  WAITING\_FOR\_YOU,  FRIENDSHIP\_REQUESTED,  BOTH\_REQUESTED,  FRIENDS,  NOT\_FRIENDS  } friendshipStatus; |
| Friend createFriend (Student new\_friend , friendshipStatus status)  Element copyFriend(Element friend);  int compareFriend(Element friend1, Element friend2);  void destroyFriend(Element friend1);  friendshipStatus getCurrentFriendStatus(Friend friend1);  void changeFriendshipStatus(Friend friend1, friendshipStatus new\_status); |

|  |
| --- |
| **Grade** |
| Int semester;  Int course\_id;  Char\* points;  int certain\_grade; |
| enum : grade\_result {  GRADE\_SUCCESS  GRADE\_OUT\_OF\_MEMORY  } |
| bool isCeratinGradeValid (int certain\_grade);  bool isCoursesEqual (Grade grade1, int other\_course\_id);  bool isGradeValid (Int semester, int course\_id, char\* points, int grade);  int getGradeSemester (Grade grade1);  Grade createGrade (Int semester, int course\_id, char\* points, int grade);  GradeResult updateGrade (Grade current, int new\_grade);  Element copyGrade (Element element);  Void DestroyGrade (Grade current);  void printGrade (Grade current); |

**ניואנסים**

* **ניהול סטודנטים** : CourseManager מכיל set של סטודנטים (Students), אשר כל Node מכיל student ספציפי ומצביע לסטודנט הבא.
* **ניהול התחברות** : התחברות מנוהלת ע"י connected\_student, כאשר הוא מצביע לnull – אף סטודנט לא מחובר.
* **הגדרות** : עבור פקודות ההדפסה נגדיר את המושגים הבאים:
  + קורס ספורט – קורס שמספרו נע בין 390000 לבין 399999 ,כולל
  + ציון תקף לסמסטר – ציון עבור קורס מסוים, כך שלא קיים ציון מאוחר יותר (לפי סדר ההוספה) של אותו הקורס עבור אותו הסמסטר
  + ציון תקף לגיליון – ההגדרה שונה עבור ציון של קורס ספורט וציון של קורס שאינו קורס ספורט
    - עבור קורס ספורט, ציון נחשב לתקף לגיליון אם הוא תקף לסמסטר.
    - עבור קורס שאינו קורס ספורט, ציון נחשב לתקף לגיליון אם הוא תקף לסמסטר ולא קיים ציון תקף לסמסטר עבור סמסטר מאוחר יותר.