Θα υλοποιήσετε ένα πρόγραμμα με όνομα mngstd, που με την βοήθεια μιας σύνθεσης δομών θα επιτρέπει την εξαγωγή πληροφοριών και στατιστικών στοιχείων για την γραμματεία του τμήματος. Θα πρέπει να υλοποιήσετε μια δομή πάνω στην οποία θα κάνετε εισαγωγές/διαγραφές αλλά και αναζητήσεις σε εγγραφές φοιτητών. Οι λειτουργίες αυτές θα πρέπει εν γένει να έχουν κόστος προσπέλασης O(1). Κάτι τέτοιο μπορείτε να το επιτύχετε όταν χρησιμοποιείτε δομή κατακερματισμού (hashing). Επίσης, θα πρέπει να μπορείτε δυναμικά να απαντήσετε επερωτήσεις που έχουν να κάνουν με συγκεκριμένες ομάδες δεδομένων χρησιμοποιώντας μια δομή τύπου ανεστραμμένου καταλόγου (inverted index).
Μερικές βασικές προϋποθέσεις για την παραπάνω εφαρμογή είναι οι εξής:
1. Η βασική δομή δεδομένων είναι το hashing και οργανώνει τις εγγραφές των φοιτητών σύμφωνα με το StudentID που λειτουργεί σαν κλειδί.
2. Η δομή δέχεται εισαγωγές, διαγραφές και επερωτήσεις.
3. Η εγγραφή κάθε φοιτήτριας/φοιτητή αποτελείται από το StudentID της/του (κλειδί), όνομα, επίθετο, Τ.Κ.πόλης μόνιμης κατοικίας, έτος 1ης εγγραφής και μέσο όρο μαθημάτων μέχρι στιγμής που έχει πάρει μέχρι στιγμής.
4. Ταυτόχρονα η εφαρμογή χρησιμοποιεί έναν inverted index ώστε να ομαδοποιεί τις εγγραφές που υπάρχουν ανά έτος φοίτησης ώστε να μπορούμε να ρωτάμε την δομή για χαρακτηριστικά και μεγέθη που αφορούν στην συγκεκριμένη ομάδα.
5. Η εφαρμογή θα πρέπει να μπορεί να διαβάσει δεδομένα και στην εκκίνηση της από ένα αρχείο εισόδου.
6. Το πρόγραμμα σας θα πρέπει –όποτε αυτό απαιτείται– να ελευθερώνει όλη την μνήμη που έχει δεσμεύσει. Το ίδιο ισχύει στον τερματισμό της εφαρμογής. Μέθοδοι προσπέλασης σαν και αυτές που αναφέρονται παραπάνω είναι πολύ κοινές σε υλοποιήσεις συστημάτων ανάκτησης πληροφορίας και μηχανών αναζήτησης.

Περιγραφή της Διεπαφής του Προγράμματος:
Αφού το πρόγραμμα κληθεί, και οι σχετικές δομές για όλους τους φοιτητές εισαχθούν, η/ο χρήστης μπορεί να
αλληλεπιδράσει με τα δεδομένα του mngstd μέσω ενός prompt. Μέσω του τελευταίου οι παρακάτω εντολές μπορούν να κληθούν:
1. i(nsert) studentid lastname firstname zip year gpa: εισήγαγε στην δομή ένα φοιτητή με κλειδί studentid που έχει εισαχθεί το έτος year, έχει μόνιμη κατοικία στον Τ.Κ. zip και έχει μέσο όρο μέχρι στιγμής gpa. Επανάληψη εισαγωγής εγγραφής με υπάρχον κλειδί studentid στην δομή δεν είναι εφικτή και σχετική ένδειξη λάθους εμφανίζεται στην έξοδο.
2. l(ook-up) studentid: ανάκτησε και τύπωσε στην εγγραφή του εν λόγω φοιτητή στο tty. Αν δεν υπάρχει, τύπωσε σχετικό μήνυμα.
3. d(elete) studentid: διέγραψε από την δομή την/τον φοιτήτρια/φοιτητή με αριθμό studentid. Αν δεν υπάρχει τέτοια εγγραφή, τύπωσε σχετικό μήνυμα.
4. n(umber) year: για την ακαδημαϊκή χρονιά year βρες πόσοι παραμένουν εγγεγραμμένοι. Στο ΕΚΠΑ εγγεγραμμένος μπορεί να παραμείνει κάποιος χωρίς περιορισμό. Η εντολή αυτή βρίσκει πόσοι φοιτητές συνεχίζουν να δραστηριοποιούνται πχ. στο 6ο χρόνο, κλπ.
5. t(op) num year: για την ακαδημαϊκή χρονιά year βρες τους num φοιτητές με την καλύτερη απόδοση.
6. a(verage) year: για την ακαδημαϊκή χρονιά year βρες τους τον μέσο όρο.
7. m(inimum) year: για την ακαδημαϊκή χρονιά year βρες την/τον φοιτήτρια/φοιτητή με το μικρότερο μέσο όρο.
8. c(ount): για κάθε έτος φοίτησης βρες τον αριθμό των φοιτητών που υπάρχουν στην δομή.
9. p(ostal code) rank: βρες την πόλη (δηλ. Τ.Κ.) που είναι ή rank th πιο ‘δημοφιλής’ όσον αφορά το μέρος μόνιμης κατοικίας των φοιτητών.
10. exit: το πρόγραμμα τερματίζει αφού απελευθερώσει με συγκροτημένο τρόπο όλο το χώρο που έχει καταλάβει στην μνήμη.

Complile with: 1) make
2)../main -p input1.csv

//fuction in order to count the line of file and use for hash_table
int count_linee(char const *argv[])

//read the line of file and insert in struct hash_table_student, listehead and zip(use for count of cities)
int read_file(char const *argv[],hash_table_student hash_student ,record **head1 ,time_in **Listhead,city **zip_city)

//find the position from id so find or insert the record
int hash_function(char* id , int lenght , int size)

//create space memory for hash_student
void my_constructor_hash_table_student(char const *argv[] , hash_table_student *hash_student)

//free memory space for hash_student
void distractor_hash_table(char const *argv[] , hash_table_student *hash_student)

//special insert for struct hash_student
void insert_hash_student(record* my_rec ,hash_table_student * hash_student , char const *argv[])

//special insert inside insert_hash_student in order to create unique list for every hash_student node
void insert_record(record* head,record* my_rec)

//insert for new record in hash_student
void insert(record* my_rec , hash_table_student * hash_student , char const *argv[])

//print in terminal map for help
void printMenu(void)

//print record inside hash_student
void print_hash_table(hash_table_student * hash_student , char const *argv[])

//print a record
void print_record(record* temp)

//copy from a record to another record
void nodecpy(record* temp_node,record *save)

//initialize a strruct record
void constructor_record(record *node)

//initialize the struct time_in
void constructor_time_in(time_in* node)

//ind and print the record with id my_rec
void lookup(char* save, hash_table_student* hash_student,char const *argv[])

```
//find and delete the record with id my_rec
void delet(char* my_rec ,hash_table_student* hash_student  , char const *argv[])

//find the count of student every year
void count(time_in* Listhead)

//find the number  of students who are activate every moment until this year
void number(char* save, time_in* Listhead)

//find the average of student who inside the the university same year
void avarage(char* save, time_in* Listhead)

//find the id which have lower gpa in the same year
void min_gpa(char* save, time_in* Listhead)

//insert record* save  inside to time_in ** Listhead and create one list of lists base on the different year which inside to university

void insert_time(record **head , time_in **Listhead ,record* save )



//insert zip in list of zip and count of zip in order to find the more popular cities of file
void insert_zip( city **head ,int zip )

//find the top-k cities form list city
void postal_code(city *zip,int k)


void print_node(record* node)

//free struct and exit
void my_exit(hash_table_student *hash_student,time_in *Listhead,record* head1,city
*zip_city,char const *argv[])
```