Εργασία 2 – Μια πιο πολύπλοκη βάση δεδομένων στη μνήμη

Επεκτείνετε το πρόγραμμα / βάση δεδομένων που ήδη αναπτύξατε στην προηγούμενη εργασία, έτσι ώστε να υποστηρίζεται (α) η εγγραφή των φοιτητών σε μαθήματα, και (β) η γρήγορη αναζήτηση φοιτητών με βάση το όνομα τους. Η επιθυμητή λειτουργικότητα περιγράφεται παρακάτω. Το πρόγραμμα που θα υποβάλετε πρέπει να είναι αποθηκευμένο σε ένα μόνο αρχείο με όνομα project2.c

Καταγραφή εγγραφής φοιτητή σε μαθήματα

Οι εγγραφές ενός φοιτητή στα μαθήματα που επιθυμεί πρέπει να καταγράφονται χρησιμοποιώντας μια απλά συνδεδεμένη λίστα (simply linked list), με κάθε στοιχείο της λίστας να έχει τον κωδικό του μαθήματος ως unsigned short. Η λίστα μαθημάτων του φοιτητή πρέπει να είναι ταξινομημένη με αύξουσα σειρά κωδικού μαθήματος. Η κεφαλή (head) της λίστας μαθημάτων πρέπει να αποθηκεύεται μέσα στην δομή (struct) που ήδη χρησιμοποιείται για την αποθήκευση της υπόλοιπης πληροφορίας του φοιτητή (η υπάρχουσα δομή πρέπει να επεκταθεί κατάλληλα).

Επιπλέον λειτουργίες για την διαχείριση μαθημάτων

Η διαχείριση της λίστας μαθημάτων του φοιτητή πρέπει να υποστηρίζεται από αντίστοιχες (επιπλέον) λειτουργίες (η ακριβής περιγραφή των εντολών που πρέπει να μπορεί να διαβάζει από την είσοδο του το πρόγραμμα, και των απαντήσεων που πρέπει να εκτυπώνει το πρόγραμμα στην έξοδο του, δίνεται αναλυτικά σε ξεχωριστή ενότητα):

reg: Εγγραφή φοιτητή σε μάθημα, με προσθήκη ενός αντίστοιχου νέου στοιχείου στην λίστα μαθημάτων. Αν δεν υπάρχει φοιτητής με το ΑΕΜ που δίνεται ή ο φοιτητής είναι ήδη εγγεγραμμένος στο μάθημα που δίνεται, επιστρέφεται λάθος.

unreg: Διαγραφή φοιτητή από μάθημα, με απομάκρυνση του αντίστοιχου στοιχείου από την λίστα μαθημάτων. Αν δεν υπάρχει φοιτητής με το ΑΕΜ που δίνεται ή ο φοιτητής δεν είναι εγγεγραμμένος στο μάθημα που δίνεται, επιστρέφεται λάθος.

isreg: Έλεγχος της λίστας μαθημάτων του φοιτητή για το αν ο φοιτητής είναι εγγεγραμμένος στο μάθημα. Αν δεν υπάρχει φοιτητής με το ΑΕΜ που δίνεται, επιστρέφεται λάθος.

list-courses: Εκτύπωση των μαθημάτων που είναι εγγεγραμμένος ο φοιτητής, με αύξουσα σειρά κωδικού μαθήματος. Αν δεν υπάρχει φοιτητής με το ΑΕΜ που δίνεται, επιστρέφεται λάθος.

Για απλότητα, υποθέτουμε ότι ένας φοιτητής μπορεί να γράφεται σε όποιο μάθημα επιθυμεί, και να διαγράφεται από όποιο μάθημα επιθυμεί, ανά πάσα στιγμή, χωρίς περιορισμούς.

Επιπλέον λειτουργία αναζήτησης φοιτητών

Πρέπει να προστεθεί και να υλοποιηθεί ξεχωριστή λειτουργία αναζήτησης φοιτητή (δείτε παρακάτω), με βάση το όνομα (όχι το ΑΕΜ):

find-by-name: Αναζήτηση φοιτητή με βάση το όνομα του. Αν βρεθεί ο φοιτητής, εκτυπώνονται τα στοιχεία του (αν υπάρχουν πολλοί φοιτητές με το ίδιο όνομα, τυπώνονται τα στοιχεία όλων), διαφορετικά εκτυπώνεται μήνυμα ότι δεν βρέθηκε.

Απαιτήσεις υλοποίησης

- Κάθε μια από τις παραπάνω λειτουργίες πρέπει να υλοποιηθεί μέσα από μια ξεχωριστή συνάρτηση, που θα καλείται μέσα από την main του προγράμματος ανάλογα με τις εντολές που δέχεται το πρόγραμμα.
- Οι λειτουργίες reg, unreg, isreg πρέπει να χρησιμοποιούν την ήδη υπάρχουσα συνάρτηση αναζήτησης φοιτητή με βάση το ΑΕΜ που υλοποιήσατε στο πλαίσιο της προηγούμενη εργασίας.
- Απαγορεύεται η χρήση καθολικών/static μεταβλητών και η χρήση goto.

Γρήγορη αναζήτηση φοιτητών με βάση το όνομα

Για την ταχύτερη αναζήτηση των φοιτητών με βάση το όνομα τους, αντί να υλοποιήσετε (γραμμική) αναζήτηση μέσα στον δυναμικό πίνακα δεικτών, χρησιμοποιήστε έναν πίνακα κατακερματισμού (hash table). Ως συνάρτηση κατακερματισμού (hash function) χρησιμοποιήστε την :

```
unsigned long hash(char *str)
{
    unsigned long hash = 5381;
    int c;

    while ((c = *str++))
        hash = ((hash << 5) + hash) + c;

    return hash;
}
```

Οι εγγραφές που αντιστοιχίζονται στο ίδιο δοχείο θα πρέπει να αποθηκεύονται σε μια διπλά συνδεδεμένη λίστα με τερματικό στοιχείο (doubly linked list with sentinel) και να είναι ταξινομημένες σε αύξουσα αλφαβητική σειρά και, για κόμβους με ίδιο όνομα, σε αύξουσα σειρά ΑΕΜ. Οι κόμβοι της λίστας πρέπει να δείχνουν απευθείας στις αντίστοιχες εγγραφές φοιτητών, όχι σε θέσεις του δυναμικού πίνακα δεικτών. Τα άδεια δοχεία πρέπει να αρχικοποιούνται κατάλληλα, με βάση τις συμβάσεις για μια διπλά συνδεδεμένη λίστα με τερματικό (η άδεια λίστα περιέχει τον τερματικό κόμβο).

Το πρόγραμμα παίρνει ως τρίτο όρισμα από τη γραμμή εντολών το αρχικό μέγεθος του πίνακα κατακερματισμού, το οποίο είναι επίσης το ελάχιστο επιτρεπτό μέγεθος αυτού του πίνακα.

Η πολιτική ανακατακερματισμού για την δυναμική αυξομείωση του πίνακα (re-hashing) βασίζεται στο βαθμό πληρότητας (load factor) του πίνακα. Αν μετά την εισαγωγή νέας εγγραφής ο βαθμός πληρότητας είναι μεγαλύτερος ή ίσος του 4, τότε ο πίνακας διπλασιάζεται. Αν μετά τη διαγραφή μιας εγγραφής ο βαθμός πληρότητας είναι 1, τότε ο πίνακας υποδιπλασιάζεται, αλλά χωρίς να επιτραπεί ποτέ να έχει μικρότερο μέγεθος από το ελάχιστο.

Μεταβολές στις υπάρχουσες λειτουργίες

Οι υπάρχουσες λειτουργίες διαχείρισης του μητρώου των φοιτητών add, rmv, mod, find, print, clear και quit πρέπει να επεκταθούν, έτσι ώστε να γίνεται σωστή διαχείριση του πίνακα κατακερματισμού παράλληλα με τον αρχικό/βασικό πίνακα δεικτών στις εγγραφές.

Δε χρειάζεται να γίνεται κάποιος έλεγχος συμβατότητας ανάμεσα στο πλήθος χρωστούμενων μαθημάτων και στο πλήθος μαθημάτων στα οποία έχει εγγραφεί ο φοιτητής.

Η λειτουργία rmv πρέπει να διαγράφει, με αντίστοιχη απελευθέρωση της μνήμης, όλα τα δεδομένα του φοιτητή συμπεριλαμβανομένης της λίστας μαθημάτων του. Η λειτουργία clear πρέπει να διαγράφει με αντίστοιχη απελευθέρωση της μνήμης όλες τις εγγραφές φοιτητών και τον πίνακα δεικτών και να επαναφέρει τον πίνακα κατακερματισμού στο αρχικό του μέγεθος. Η λειτουργία quit πρέπει να διαγράφει όλα τα δεδομένα της βάσης, με πλήρη απελευθέρωση όλης της δυναμικά δεσμευμένης μνήμης των δομών δεδομένων του προγράμματος.

Η μορφή των παραπάνω εντολών που δέχεται το πρόγραμμα από την είσοδο του, και η μορφή των μηνυμάτων που εκτυπώνει το πρόγραμμα στην έξοδο του και στην έξοδο σφαλμάτων του, παραμένουν ως έχουν (βλέπε προηγούμενη εργασία), με εξαίρεση τις λειτουργίες print, find (βλέπε παρακάτω).

Επίσης, πρέπει να προστεθεί μια ξεχωριστή εντολή χρήστη για την αναζήτηση φοιτητή με βάση το όνομα (βλέπε παρακάτω).

Μορφοποίηση εισόδου/εξόδου

Παρακάτω ορίζεται η μορφή των επιπλέον εντολών που διαβάζει το πρόγραμμα από την είσοδο του, καθώς και η μορφή των μηνυμάτων που εκτυπώνει το πρόγραμμα στην έξοδο του.

Εντολή	Είσοδος	Έξοδος stdout (επιτυχία)	Έξοδος stdout (αποτυχία)
reg	g <aem> <course nr>	G-OK <aem> <course nr>	G-NOK <aem> ή G-NOK <course nr>
unreg	u <aem> <course nr>	U-OK <aem> <course nr>	U-NOK <aem> ή U-NOK <course nr>
isreg	i <aem> <course nr>	YES ή NO	I-NOK <aem>
list-courses	l <aem>	* βλέπε παρακάτω	L-NOK <aem>
find-by-name	n <name>	** βλέπε παρακάτω	N-NOK <name>

Αν μια εντολή αποτελείται από περισσότερα τμήματα (συνοδεύεται από μια ή περισσότερες παραμέτρους), αυτά διαχωρίζονται με έναν ή περισσότερους χαρακτήρες ‘ ‘ (space). Κάθε εντολή τερματίζει με τον χαρακτήρα ‘\n’.

Πριν και μετά από κάθε μήνυμα εξόδου βρίσκεται χαρακτήρας ‘\n’. Αν το μήνυμα αποτελείται από περισσότερα τμήματα, αυτά διαχωρίζονται από έναν μοναδικό χαρακτήρα ‘ ‘ (space).

* Στην εντολή list-courses εκτυπώνεται το μήνυμα L-OK <name> και ξεκινώντας από την επόμενη γραμμή, μια εγγραφή μαθήματος <course nr> ανά γραμμή.

** Στην εντολή find-by-name εκτυπώνεται το μήνυμα N-OK <name> και ξεκινώντας από την επόμενη γραμμή, για κάθε φοιτητή με αυτό το όνομα εκτυπώνεται το ΑΕΜ του, χαρακτήρας ' ' (space), το πλήθος μαθημάτων που χρωστάει και χαρακτήρας '\n'.

Η λειτουργία print πρέπει να αλλάξει έτσι ώστε να εκτυπώνει στην έξοδο του προγράμματος τα περιεχόμενα του πίνακα κατακερματισμού ως εξής:

Αρχικά εκτυπώνει χαρακτήρες ‘\n’ και ‘#’.

Στην επόμενη γραμμή εκτυπώνει: <size> <elements> <load factor> <largest bucket> όπου <size> είναι το μέγεθος του πίνακα, <elements> ο συνολικός αριθμός των εγγραφών, <load factor> η τιμή elements/size με δύο δεκαδικά ψηφία, και <largest bucket> το μέγεθος του μεγαλύτερου δοχείου.

Στην συνέχεια, ξεκινώντας από την επόμενη γραμμή, για κάθε θέση του πίνακα, εκτυπώνονται σε μια γραμμή τα περιεχόμενα του αντίστοιχου δοχείου, αρχίζοντας με <pos>, κενό, <len>, όπου <pos> είναι ο αριθμός της θέσης και <len> το μέγεθος του δοχείου, και μετά (αν το δοχείο δεν είναι άδειο), για κάθε εγγραφή φοιτητή εκτυπώνεται ένα κενό και ‘[‘<aem> <name> <courses>‘]’ (<courses> είναι ο αριθμός των χρωστούμενων μαθημάτων). Μετά το τέλος των περιεχομένων του δοχείου, εκτυπώνονται δύο χαρακτήρες '\n'.

Προσοχή: Το πρόγραμμα πρέπει να ακολουθεί ακριβώς τις παραπάνω προδιαγραφές, διαφορετικά θα αποτυγχάνει ο αυτοματοποιημένος έλεγχος της λειτουργίας του προγράμματος.

Μελέτη απόδοσης

Μελετήστε (ξανά) την απόδοση της υλοποίησης σας, καταγράφοντας αυτη τη φορά τον αριθμό των βημάτων και τον χρόνο εκτέλεσης της εντολής find-by-name, για ονόματα φοιτητών που υπάρχουν και ονόματα φοιτητών που δεν υπάρχουν, όταν η βάση έχει 1.000, 10.000 και 100.000 εγγραφές.

Για κάθε μια μέτρηση, θα βρείτε στο site του μαθήματος ένα text file που σε πρώτη φάση περιέχει τις εντολές add (για να γεμίσει η βάση), και σε δεύτερη φάση τις εντολές find-by-name (για να γίνει αναζήτηση). Εσείς απλά ανακατευθύνετε την είσοδο του προγράμματος στο αρχείο, και την έξοδο σφαλμάτων (όπου εκτυπώνονται τα βήματα και ο χρόνος αναζήτησης) σε ένα αρχείο που θα αποθηκευτούν όλες οι μετρήσεις. Όταν τερματίσει η κάθε εκτέλεση, εξάγετε από το αρχείο τις τιμές που εκτύπωσε το πρόγραμμα, και τις επεξεργάζεστε, π.χ., με την βοήθεια του Excel, για να βγάλετε τα βασικά στατιστικά μεγέθη (min, max, average, median).

Τέλος, πρέπει να φτιάξετε γραφικά διαγράμματα (graph plots) με τα παραπάνω αποτελέσματα, τα οποία θα παρουσιάσετε/σχολιάσετε στην εξέταση της εργασίας. Επίσης, συγκρίνετε τα αποτελέσματα με αυτά που είχατε καταγράψει για την προηγούμενη υλοποίηση για την περίπτωση που η αναζήτηση με ΑΕΜ γίνεται σε πλήρως ταξινομημένο πίνακα και παρουσιάστε τα συμπεράσματά τας.

Στάδια ανάπτυξης του κώδικα

Στάδιο 0:

Ορίστε το struct για την απλή λίστα μαθημάτων και υλοποιήστε ξεχωριστές συναρτήσεις για τις βασικές πράξεις προσθήκης, αφαίρεσης, εύρεσης μαθήματος στη λίστα, εκτύπωσης των περιεχομένων της λίστας σύμφωνα με τις οδηγίες της εκφώνησης, και απελευθέρωσης όλης της δυναμικά δεσμευμένης μνήμης της λίστας (για την περίπτωση που ο φοιτητής αφαιρείται από το μητρώο).

Ελέγξτε ενδελεχώς την ορθότητα του κώδικά σας πριν προχωρήσετε.

Προσθέστε στο struct φοιτητή της πρώτης εργασίας ένα πεδίο για την κεφαλή της λίστας μαθημάτων του φοιτητή.

Υλοποιήστε τις reg, unreg, isreg, list-courses και κάντε ότι αλλαγές χρειάζεται στις αρχικές λειτουργίες add, rmv, clear ώστε να γίνεται σωστή διαχείριση της λίστας μαθημάτων

Στάδιο 1:

Αλλάξτε το struct φοιτητή της πρώτης εργασίας ώστε να μπορεί να χρησιμοποιηθεί ως τύπος κόμβου σε μια διπλά συνδεδεμένη λίστα. Υλοποιήστε ξεχωριστές συναρτήσεις για τις βασικές πράξεις προσθήκης, αφαίρεσης, εύρεσης φοιτητή στη λίστα, εκτύπωσης των περιεχομένων της λίστας σύμφωνα με τις οδηγίες της εκφώνησης, και απελευθέρωσης όλης της δυναμικά δεσμευμένης μνήμης της λίστας. Συμβουλή: Σκεφτείτε πρώτα πώς θα χρησιμοποιήσετε τον πίνακα κατακερματισμού, ώστε να σχεδιάσετε τις συναρτήσεις για τις λίστες με τρόπο που θα σας διευκολύνει.

Στάδιο 2:

Κατασκευάστε τον πίνακα κατακερματισμού και υλοποιήστε τις βασικές λειτουργίες προσθήκης, εύρεσης και αφαίρεσης εγγραφών σε αυτόν. Κάντε ότι αλλαγές χρειάζεται στις αρχικές λειτουργίες add, rmv, clear ώστε να γίνεται σωστή διαχείριση του πίνακα κατακερματισμού.

Στάδιο 3:

Υλοποιήστε τη λειτουργία ανακερματισμού και τη λειτουργία find-by-name που θα καλείται μέσα από την main όταν ο χρήστης δώσει την αντίστοιχη εντολή. Τροποποιήστε κατάλληλα τη λειτουργία print.

Στάδιο 4:

Αν έχετε χρόνο και κέφι για περισσότερα, επικοινωνήστε με τους διδάσκοντες.