ΙΩΑΝΝΗΣ ΚΑΠΕΤΑΝΓΕΩΡΓΗΣ **Α.Μ. : 1115201800061**

K22: Λειτουργικά Συστήματα (Περιττοί) – Χειμερινό '20

Project 1

Γενικές παρατηρήσεις:

- Όλα τα αρχεία κώδικα είναι πλήρως σχολιασμένα και εξηγείται η κάθε λειτουργία που γίνεται στο κάθε σημείο.
- Όση μνήμη έχει δεσμευθεί δυναμικά κατά την εκτέλεση του προγράμματος, αποδεσμεύεται πριν το τέλος αυτού με ελεγχόμενο τρόπο και χωρίς να υπάρχουν απώλειες μνήμης η errors. Έχει ελεγχθεί μέσω του valgrind.
- Ο κώδικας έχει χωριστεί σε πολλά αρχεία (.c και .h) τα οποία βρίσκονται στα αντίστοιχα directories έτσι ώστε να επιτευχθεί abstraction των δεδομένων αλλά και να είναι πιο κατανοητός και ευανάγνωστος ο συνολικός κώδικας. Επίσης επιτυγχάνεται πλήρης απόκρυψη πληροφορίας. Για κάθε αρχείο κώδικα (.c) (εκτός του αρχείου mngstd.c που περιέχει μόνο την main) υπάρχουν και τα αντίστοιχα header αρχεία. Για παράδειγμα για το αρχείο HashTableImplementation.c υπάρχει το αρχείο HashTableTypes.h (το οποίο περιέχει την δήλωση των δομών που χρησιμοποιούνται) και HashTablePrototypes.h (το οποίο περιέχει τις δηλώσεις των συναρτήσεων που υλοποιούνται HashTableImplementation.c έτσι ώστε να μπορούν να χρησιμοποιηθούν και σε κάποιο άλλο .c αρχείο). Τα αρχεία αυτά εξηγούνται και παρακάτω ξεχωριστά.
- Στο main directory υπάρχει το configure file (confingFile.txt) το οποίο μπορεί να δοθεί σαν input στο πρόγραμμα μας (βλ. παράγραφο Configure file σελίδα 5).
- Για να δοθεί ένα input file σαν είσοδος, μετά από την παράμετρο -i πρέπει να δοθεί το path του αρχείου. Αν το αρχείο βρίσκεται στο main directory, αρκεί να δώσουμε μόνο το όνομα του (π.χ. ./demo -i input1.txt).
- Τέλος, έχουν υλοποιηθεί όλα όσα ζητούνται στην εκφώνηση

Compilation:

- Έχει δημιουργηθεί αρχείο Makefile το οποίο κάνει compile τα αρχεία κώδικα και παράγει το εκτελέσιμο αρχείο με όνομα demo.
- Μέσω του Makefile γίνεται separate compilation, δηλαδή για κάθε ένα .c αρχείο δημιουργείται το αντίστοιχο object file (.o) και μέσω των object αρχείων, τελικά, δημιουργείται το εκτελέσιμο.
- Μόλις "τρέξουμε" την εντολή make, πριν γίνει το compilation διαγράφονται πρώτα όλα τα object files και το εκτελέσιμο (εάν αυτά υπάρχουν από προηγούμενο compilation).

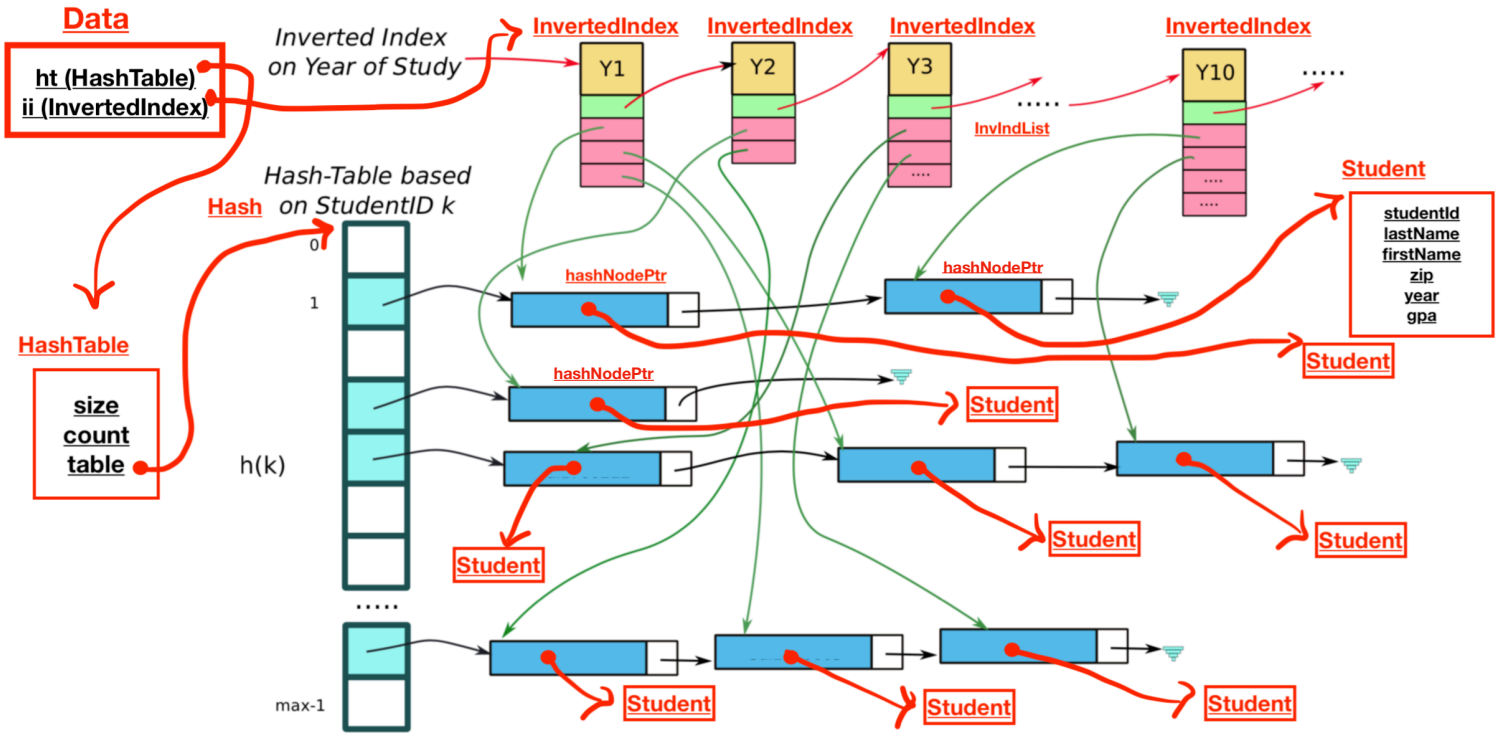
Interaction χρήστη στο prompt:

- Για την επιλογή της λειτουργίας που θέλουμε να χρησιμοποιήσουμε δίνεται μόνο το πρώτο γράμμα αυτής.
- Στις συναρτήσεις που δίνεται σαν όρισμα το έτος, χρησιμοποιείται το τρέχον έτος σπουδών και όχι η χρονία εγγραφής του φοιτητή.

Για παράδειγμα, εάν θέλουμε τον αριθμό των φοιτητών του 2ου έτους (δηλαδή με έτος εγγραφής 2019) θα "δώσουμε" την εντολή: n 2

Δομές:

Στο παρακάτω σχήμα φαίνεται η οργάνωση όλων των δομών που χρησιμοποιούνται. Τα υπογραμμισμένα ονόματα με κόκκινο χρώμα, είναι τα ονόματα τον δομών. Η κάθε δομή εξηγείται αναλυτικά στην συνέχεια.



Δομή Student:

Ο τύπος δεδομένου Student είναι ένας pointer σε έναν κόμβο τύπου studentNode. Εσωτερικά του studentNode αποθηκεύονται τα στοιχεία του κάθε φοιτητή. Επίσης έχει γίνει typedef του char * σε word για να είναι πιο ευανάγνωστη η διαχείριση συμβολοσειρών. Τα δεδομένα που αποθηκεύονται στον κάθε κόμβο είναι: studentId (το id του φοιτητή, τύπου word για να μπορεί να περιέχει και χαρακτήρες π.χ. sdi1800061), lastName (επώνυμο), firstName (όνομα), zip (ταχυδρομικός κώδικας), year (έτος εγγραφής π.χ. 2018) και gpa (μέσος όρος). Όσον αφορά το έτος εγγραφής στον Student αποθηκεύεται κανονικά και η μετατροπή σε τρέχον έτος σπουδών γίνεται εσωτερικά του InveredIndex (εξηγείται παρακάτω).
Η διαχείριση (αποθήκευση/προσπέλαση) αυτών των κόμβων γίνεται πάντα μέσω pointer στον κόμβο δηλαδή με μεταβλητές τύπου Student.
Η δήλωση της δομής αυτής βρίσκεται στο αρχείο StudentTypes.h

Η επιλογή να χρησιμοποιηθεί αυτή η δομή και όχι να αποθηκεύονται κατευθείαν τα στοιχεία του φοιτητή στους κόμβους του HastTable έγινε κυρίως για να είναι πιο ανεξάρτητα τα δεδομένα. Στην υλοποίηση αυτή η ύπαρξη ενός φοιτητή δεν εξαρτάται από την ύπαρξη ενός HashTable και την αποθήκευση του εκεί. Δηλαδή σε μία ενδεχόμενη επέκταση του προγράμματος (π.χ. μια νέα δομή που αποθηκεύονται όσοι αποφοιτούν), η υλοποίηση και η διαχείριση των φοιτητών θα είναι πολύ εύκολη και απλή. Κάτι που θα ήταν αρκετά πιο περίπλοκο αν δεν υπήρχε η δομή Student. Τέλος, κατά την γνώμη μου ο διαχωρισμός αυτός βοηθάει πολύ στην οργάνωση του κώδικα αλλά και στην κατανόηση του.
Επίσης, στην δομή Student έχουν υλοποιηθεί και συναρτήσεις επιστροφής των δεδομένων του κάθε φοιτητή (π.χ. getStudentId(student)) για να είναι πιο κατανοητή και σωστή η προσπέλαση των δεδομένων του (π.χ. Σε πιθανή αλλαγή ονόματος ενός χαρακτηριστικού του Student χρειάζεται να αλλαχθεί μόνο η αντίστοιχη συνάρτηση get) καθώς και συναρτήσεις για την δημιουργία/διαγραφή των κόμβων Student (createStudent/deleteStudent). Οι συναρτήσεις αυτές υλοποιούνται στο αρχείο Student.c και οι δηλώσεις αυτών στο αρχείο StudentPrototypes.h .

Σημαντική παρατήρηση:
Η ύπαρξη της δομής Student δεν αναιρεί όσα ζητούνται από την εκφώνηση της εργασίας και περιγράφονται από το σχήμα της οργάνωσης τον δομών που δίνεται. Πιο συγκεκριμένα, ο κάθε φοιτητής αποθηκεύεται μόνο στο HashTable (δηλαδή στον κόμβο της λίστας που υπάρχει σε μία θέση του Hashable) και μπορεί να προσπελαστεί μόνο από εκεί. Δηλαδή ο κάθε κόμβος της λίστας του InvertedIndex “δείχνει” σε έναν κόμβο του HashTable (και όχι κατευθείαν σε έναν Student) μέσω του οποίο μπορούμε να προσπελάσουμε τον φοιτητή. Όπως ακριβώς φαίνεται στο σχήμα της εκφώνησης αλλά και στο σχήμα που έχω παραθέσει παραπάνω.

Δομή HashTable:

Ο τύπος που χρησιμοποιείται για την αποθήκευση/διαχείριση του Hash Table είναι ο HashTable. Ο τύπος αυτός στην πραγματικότητα είναι ένας pointer σε έναν dummy node τύπου HashHeadNode. Ο dummy node έχει ως χαρακτηριστικά το μέγεθος του HashTable (size), τον τρέχον αριθμό των αντικειμένων που υπάρχουν μέσα στο HashTable (count) και έναν pointer στο actual hash table (table) το οποίο είναι τύπου Hash. Το hash table (τύπου hash) είναι ένας double pointer σε κόμβους (τύπου Node), δηλαδή ένας πίνακας από pointers σε κόμβους Node. Ένας pointer σε κόμβο node είναι τύπου hashNodePtr. Ο κάθε κόμβος έχει ως χαρακτηριστικά έναν Student (δηλαδή pointer στον φοιτητή όπως εξηγήθηκε παραπάνω) έτσι ώστε να αποθηκεύουμε τους φοιτητές αλλά και έναν δείκτη σε κόμβο ίδιου τύπου (δηλαδή hashNodePtr) έτσι ώστε να δημιουργηθεί μια συνδεδεμένη λίστα.
Άρα συνοψίζοντας, το hash table (τύπου hash) είναι ένας πίνακας όπου σε κάθε θέση του υπάρχει μία λίστα της οποίας ο κάθε κόμβος περιέχει έναν φοιτητή. Όλα τα παραπάνω υπάρχουν στο αρχείο HashTableTypes.h .

Σημαντική παρατήρηση:
Το μέγεθος του πίνακα καθορίζεται από τον αριθμό των γραμμών του inputFile ή απο το configureFile σε περίπτωση που δεν υπάρχει inputFile. Εξηγείται και παρακάτω την παράγραφο του mngstdFunctions.c .

Λειτουργίες/Συναρτήσεις HashTable (HashTableImplementation.c):

- *htCreate*: Η συνάρτηση αυτή είναι υπεύθυνη για την δημιουργία και αρχικοποίηση του HashTable. Αρχικά δημιουργείται ο dummy node (δηλαδή δεσμεύεται δυναμικά μνήμη) και αρχικοποιούνται οι τιμές size και count. Επίσης δημιουργείται και ο πίνακας του HashTable (και αυτός δυναμικά) και κάθε θέση του αρχικοποείται με NULL (αφού δεν υπάρχει ακόμα κανένας φοιτητής). Η συνάρτηση αυτή δέχεται σαν όρισμα το μέγεθος του HashTable (το οποίο παραμένει σταθερό καθ’ όλη την εκτέλεση) έτσι ώστε να γνωρίζει η συνάρτηση πόσων θέσεων θα είναι ο πίνακας που δεσμεύεται.
- *htInsert*: Η συνάρτηση αυτή εισάγει έναν νέο φοιτητή στο HashTable. Δέχεται σαν όρισμα τα στοιχεία του φοιτητή (studentId,lastName…) καθώς και το HashTable στο οποίο τον εισάγουμε. Αρχικά, βρίσκουμε σε ποια θέση του πίνακα πρέπει να εισαχθεί ο φοιτητής μέσω της hash function (hash). Η hash δέχεται σαν όρισμα το id του φοιτητή και επιστρέφει το index του πίνακα στο οποίο πρέπει να εισάγουμε τον φοιτητή. Είναι παραλλαγή της djb2 την οποία διδαχθήκαμε στο μάθημα των δομών δεδομένων. Η συνάρτηση αυτή λειτουργεί εξίσου σωστά είτε το id περιέχει μόνο χαρακτήρες, είτε μόνο αριθμούς, είτε συνδυασμό και των δυο και κατανέμει αρκετά (έως πολύ) ομοιόμορφα τους φοιτητές (τουλάχιστον σε όσες δοκιμές έχω κάνει). Αφού βρούμε το index, ελέγχω αν ο φοιτητής υπάρχει ήδη. Αν δεν υπάρχει ο φοιτητής, δημιουργούμε έναν κόμβο Student, έναν κόμβο λίστας που περιέχει τον φοιτητή και τον εισάγουμε στην λίστα των φοιτητών στην συγκεκριμένη θέση του HashTable. Ο κόμβος της λίστας του HashTable δημιουργείται με την συνάρτηση createNode. Τέλος, η συνάρτηση επιστρέφει τον κόμβο λίστας (hashNodePtr) που εισήχθη ο φοιτητής έτσι ώστε να μπορούμε να τον εισάγουμε και στο InvertedIndex (εξηγείται και παρακάτω) ή NULL αν ο φοιτητής δεν εισήχθη γιατί υπήρχε ήδη.

- *htLookup*: Η συνάρτηση αυτή εκτελεί την λειτουργία look-up που ζητείται στην εκφώνηση. Δέχεται σαν όρισμα το studentId και ο HashTable. Μέσω της hash function και του id που δόθηκε βρίσκουμε το index στο οποίο έχει εισαχθεί ο φοιτητής (αν αυτός υπάρχει). Ελέγχουμε τον φοιτητή που υπάρχει σε αυτό το index (ή τους φοιτητές σε περίπτωση που έχουμε collision) συγκρίνοντας τα id των φοιτητών. Αν βρούμε τον φοιτητή που ψάχνουμε εκτυπώνουμε τα στοιχεία του, ενώ αν δεν τον βρούμε εκτυπώνουμε το αντίστοιχο μήνυμα.
- *htReturnStudent*: Η συνάρτηση αυτή είναι βοηθητική και χρησιμοποιείται στην λειτουργία delete του αρχείου mngstdFunctions.c (εξηγείται παρακάτω). Είναι παρόμοια με την htLookup με μόνη διαφορά ότι δεν εκτυπώνει τίποτα αλλά επιστρέφει τον Student node του φοιτητή που ψάχνουμε.
- *htDelete*: Η συνάρτηση αυτή διαγράφει έναν φοιτητή από το HashTable. Δέχεται σαν όρισμα το studentId του φοιτητή που θέλουμε να διαγράψουμε και το HashTable. Ακολουθεί ίδια διαδικασία με την htLookup για να εντοπίσει τον φοιτητή. Μόλις τον εντοπίσει, διαγράφει τον κόμβο της λίστας μέσω της συνάρτησης deleteNode και συνδέει τους υπόλοιπους κόμβους της λίστας (αν υπάρχουν). Η συνάρτηση deleteNode δέχεται σαν όρισμα έναν κόμβο λίστας του HashTable (hashNodePtr), διαγραφεί τον κόμβο Student που περιέχει καλώντας την deleteStudent και τέλος κάνει free τον ίδιο τον κόμβο που δόθηκε σαν όρισμα.
- *htDestroy*: Διαγράφει/Αποδεσμεύει το HashTable. Αρχικά για κάθε λίστα που υπάρχει σε κάθε θέση του πίνακα, διαγράφει κάθε έναν κόμβο της μέσω της deleteNode. Αφού διαγραφούν όλες οι λίστες, αποδεσμεύεται το HashTable (δηλαδή ο πίνακας που περιείχε τις λίστες αυτές). Τέλος αποδεσμεύεται ο dummyNode.

Δομή InvertedIndex:

Ο τύπος που χρησιμοποιείται για την αποθήκευση/διαχείριση του Inverted Index είναι ο InvertedIndex. Ο InvertedIndex είναι δείκτης σε έναν κόμβου λίστας (InvIndNode). Κάθε ένας από αυτούς τους κόμβους αντιστοιχεί σε ένα έτος φοίτησης. Δηλαδή InvertedIndex είναι μια λίστα ετών φοίτησης. Κάθε ένας από αυτούς τους κόμβους έχει ως χαρακτηριστικά το έτος φοίτησης (year), δείκτη στον επόμενο κόμβο της λίστας και περιέχει επίσης μία λίστα (τύπου InvIndList) με τους φοιτητές του έτους αυτού καθώς και τον τρέχον αριθμό φοιτητών που περιέχει η λίστα αυτή (count). Ο τύπος InvIndList είναι ένας pointer στον κόμβο της λίστας των φοιτητών (τύπου iNode). Οι κόμβοι αυτοί περιέχουν έναν pointer σε κόμβο του HashTable (τύπου hashNodePtr) έτσι ώστε να προσπελάσουμε τον φοιτητή (αφού είναι αποθηκευμένος μόνο εκεί) καθώς και έναν pointer στον επόμενο κόμβο της λίστας. Επομένως ο InvertedIndex είναι μια λίστα από λίστες που δείχνουν σε κόμβους του HashTable (ακριβώς από φαίνεται και στο σχήμα της πρώτης σελίδας). Οι δηλώσεις τον δομών αυτών βρίσκονται στο αρχείο InvertIndexTypes.h .

Λειτουργίες/Συναρτήσεις InvertedIndex (InvertedIndexImplementation.c):

- iiCreate: Η συνάρτηση αυτή δημιουργεί και αρχικοποιεί τον InvertedIndex. Στην ουσία, επειδή δεν υπάρχει dummy node, δημιουργεί τον πρώτο κόμβο της λίστας, “βάζει” σε όλα τα χαρακτηριστικά μηδενικές τιμές και στο καταχωρεί το έτος ως -1 έτσι ώστε να γνωρίζουμε κατά την πρώτη εισαγωγή ότι πρόκειται για τον αρχικό κόμβο.
- iiInsert: Η συνάρτηση αυτή εισάγει έναν φοιτητή στον InvertedIndex. Ουσιαστικά δεν εισάγει τον φοιτητή αλλά τον κόμβο του HashTable ο οποίος δίνεται σαν όρισμα και στον οποίο είναι αποθηκευμένος ο φοιτητής. Βρίσκουμε τον κόμβο που αντιστοιχεί στο έτος φοίτησης του φοιτητή (αν δεν υπάρχει τον δημιουργούμε) και εισάγουμε τον φοιτητή στην λίστα φοιτητών του αντίστοιχου έτους. Η εισαγωγή του φοιτητή γίνεται μέσω της συνάρτησης insertToList η οποία εισάγει τους φοιτητές στην λίστα ταξινομημένους βάση gpa. Επίσης οι αρχικοί κόμβοι του κάθε έτους είναι και αυτοί ταξινομημένοι σε αύξουσα σειρά βάση έτους σπουδών. Επίσης χρησιμοποιείται η συνάρτηση currentYear η οποία επιστρέφει το τρέχον έτος έτσι ώστε να γίνει ο υπολογισμός του τρέχοντος έτους σπουδών το οποίο χρησιμοποιείται αντί του έτους εγγραφής σε ολόκληρη την δομή InvertedIndex.
- iiNumber: Η συνάρτηση αυτή υλοποιεί την λειτουργία number της εκφώνησης. Παίρνει το έτος σπουδών σαν όρισμα, βρίσκει τον αντίστοιχο κόμβο της λίστας και επιστρέφει το count το οποίο κρατάμε σε κάθε κόμβο έτους και ανανεώνεται με κάθε εισαγωγή/διαγραφή.
- iiTop: Η συνάρτηση αυτή υλοποιεί την λειτουργία number της εκφώνησης. Δέχεται σαν όρισμα τον αριθμό των φοιτητών που θέλουμε (num) και την χρονία. Βρίσκουμε στην λίστα τον κόμβο του αντίστοιχου έτους και απλά τυπώνουμε τους num πρώτους φοιτητές της λίστας καθώς είναι ήδη ταξινομημένοι κατά φθίνουσα σειρά. Αν υπάρχουν λιγότεροι από num φοιτητές σε αυτό το έτος, τυπώνονται όσοι υπάρχουν καθώς και ένα ενημερωτικό μήνυμα.
- iiMinimum: Η συνάρτηση αυτή υλοποιεί την λειτουργία minimum της εκφώνησης. Δέχεται σαν όρισμα το έτος σπουδών που μας ενδιαφέρει. Αφού βρούμε τον κατάλληλο κόμβο (όπως και στις παραπάνω συναρτήσεις) μας ενδιαφέρει ο τελευταίος κόμβος (αφού οι φοιτητές είναι ταξινομημένοι) τον οποίο και διατρέχουμε. Ελέγχουμε και τους προηγούμενους κόμβους από τον τελευταίο (έναν προς έναν από τον προτελευταίο προς την αρχή) σε περίπτωση που έχουν ίδιο gpa με τον τελευταίο και πρέπει να εκτυπωθούν και αυτοί.
- iiAverage: Η συνάρτηση αυτή υλοποιεί την λειτουργία average της εκφώνησης. Δέχεται σαν όρισμα το έτος σπουδών που μας ενδιάφερει. Εντοπίζουμε τον κόμβο του έτους αυτού και προσπελαύνουμε έναν έναν τους φοιτητές αυτού του έτους αθροίζοντας τα gpa τους. Τέλος, υπολογίζουμε τον μέσο όρο χρησιμοποιώντας το count και τον επιστρέφουμε.
- iiCount: Η συνάρτηση αυτή υλοποιεί την λειτουργία count της εκφώνησης. Ανατρέχει έναν προς έναν τους κόμβους του InvertedIndex (δηλαδή τους κόμβους κάθε έτους) και για κάθε έναν από αυτούς τυπώνει το έτος και τον αριθμό φοιτητών που υπάρχουν στην λίστα των φοιτητών (δηλαδή το count).

- iiPostalCode: Η συνάρτηση αυτή υλοποιεί την λειτουργία postal code της εκφώνησης. Δέχεται σαν όρισμα το rank που μας ενδιαφέρει. Χρησιμοποιείται μια λίστα (intList) της οποίας ο κάθε κόμβος περιέχει ένα zip και τον αριθμό των φορών που έχει εισαχθεί στην λίστα (count). Δηλαδή κάθε φορά που κάνουμε insert ένα zip στην λίστα: εάν υπάρχει ήδη απλά αυξάνουμε του count κατά 1 ενώ αν δεν υπάρχει δημιουργούμε έναν νέο κόμβο στην λίστα για αυτό το zip με count=1. Παρέχονται η συνάρτησή insertZipCode, μια συνάρτηση BubbleSort η οποία ταξινομεί την λίστα σε φθίνουσα σειρά με βάση το count του κάθε κόμβου και η deleteIntList για την διαγραφή/αποδέσμευση της λίστας. Η υλοποίηση βρίσκεται στο directory intList εσωτερικά του directory InvertedIndex. Χρησιμοποιώντας την λίστα αυτή ανατρέχουμε έναν προς έναν τους φοιτητές κάθε έτους και εισάγουμε τα zip τους στην λίστα αυτή. Θα μπορούσαμε να γλιτώσουμε αυτήν την διαδικασία εάν είσαγαμε τα zip στην λίστα όταν γίνεται η εισαγωγή του κάθε φοιτητή στον InvertedIndex. Σε αυτήν την περίπτωση, όμως, θα χρειαζόταν να έχουμε στην μνήμη καθ' όλη την διάρκεια εκτέλεσης του προγράμματος την λίστα αυτή χωρίς κανέναν λόγο, καθώς υπάρχει και περίπτωση να μην χρησιμοποιηθεί ποτέ. Κάτι τέτοιο για μεγάλο όγκο δεδομένων μπορεί να αποβεί μοιραίο. Έτσι, έκανα αυτή την επιλογή η οποία ναι μεν θέλει λίγο παραπάνω χρόνο για να εκτελεστεί, αλλά είναι πιο safe για το πρόγραμμα και δεν σπαταλάει τόση πολλή μνήμη. Μόλις ολοκληρωθεί η διαδικασία για όλους τους φοιτητές, ταξινομούμε την λίστα με την χρήση της bubbleSort από το αρχείο intListImplementation.c . Αφού ταξινομηθεί η λίστα, και ελέγχοντας για ισοβαθμίες επιστρέφουμε το rank που δόθηκε σαν όρισμα.
- iiDelete: Η συνάρτηση αυτή διαγράφει τον δοσμένο φοιτητή από τον InvertedIndex. Ο φοιτητής δεν διαγράφεται στην πραγματικότητα καθώς δεν είναι αποθηκευμένος στον InvertedIndex αλλά διαγράφεται ο κόμβος ο οποίος περιέχει τον pointer που δείχνει στον κόμβο στον οποίο είναι αποθηκευμένος. Ένας φοιτητής (ένας κόμβος Student) διαγράφεται μόνο από το HashTable. Η συνάρτηση δέχεται όρισμα τύπου Student καθώς χρειάζεται και το id και το έτος εγγραφής του φοιτητή και είναι πιο εύκολα διαχειρίσιμο και πιο ευανάγνωστο από το να περνάγαμε 2 ορίσματα.
- iiDestroy: Η συνάρτηση αυτή διαγράφει/αποδεσμεύει τον InvertedIndex. Στους κόμβους τις λίστας των φοιτητών, προφανώς δεν αποδεσμεύεται το περιεχόμενο τους (δηλαδή οι κόμβοι του HashTable).

Δομή Data και αρχείο mngstdFunctions.c :

Ο τύπος δεδομένου Data είναι ένας pointer σε έναν κόμβο τύπου dataNode. Ο κόμβος αυτός έχει ως χαρακτηριστικά ένα δεδομένο τύπου Hashable και ένα τύπου ii. Ουσιαστικά χρησιμοποιείται για να αποθηκεύσουμε τις δομές τις οποίες χρησιμοποιούμε για την αποθήκευση και διαχείριση των φοιτητών. Η δομή αυτή χρησιμοποιείται από την συνάρτηση main στο αρχείο mngstd.c έτσι ώστε να "κρατάμε" όλες τις δομές δεδομένων σε μία μόνο μεταβλητή. Η δήλωση της δομής Data βρίσκεται στο αρχείο mngstdTypes.h. Επίσης έχουν υλοποιηθεί συναρτήσεις αρχικοποίησης των αντικειμένων Data (initializeData) και διαγραφής/αποδέσμευσης (deleteData) και βρίσκονται στο αρχείο mngstdFunctions.c. Επομένως η δομές που χρησιμοποιούνται για τους φοιτητές (δηλαδή το HashTable και ο InvertedIndex) αρχικοποιούνται και διαγράφονται από αυτές τις 2 συναρτήσεις οι οποίες εσωτερικά καλούν τις αντίστοιχες συναρτήσεις αρχικοποίησης/διαγραφής της κάθε δομής.

Επίσης στο αρχείο mngstdFunctions.c υπάρχει μία συνάρτηση για κάθε μία λειτουργία που παρέχεται από το promt (και ζητούνται από την εκφώνηση) π.χ. insert, lookup, top… και έχουν τα αντίστοιχα ονόματα. Η κάθε μία δέχεται τα κατάλληλα ορίσματα τα οποία απαιτεί η εκάστοτε λειτουργία και μία μεταβλητή τύπου Data μέσω της οποίας θα χρησιμοποιήσουμε την κατάλληλη δομή για να εκτελέσουμε την λειτουργία. Οι υλοποιήσεις είναι πολύ απλές καθώς απλά καλείται η αντίστοιχη συνάρτηση της κατάλληλης δομής (ανάλογα την λειτουργία) η οποία και εκτελεί την λειτουργία αυτή όπως έχει περιγράφει παραπάνω (π.χ. για την lookup καλείται η αντίστοιχη συνάρτηση του HashTable ενώ για την average η αντίστοιχη συνάρτηση του InvertedIndex). Επίσης γίνονται και οι κατάλληλες εκτυπώσεις σε όσες λειτουργίες δεν γίνονται εσωτερικά.
Τέλος υπάρχει η συνάρτηση readInput η οποία καλείται στην αρχή της main και είναι υπεύθυνη για την αρχικοποίηση των δεδομένων (δηλαδή των δομών). Δέχεται σαν ορίσματα τα argc και argv δηλαδή τις παραμέτρους που έδωσε ο χρήστης στην γραμμή εντολών κατά της εκτέλεση του προγράμματος. Αρχικά ελέγχει αν έχουν δοθεί inputFile και configureFile. Αν έχει δοθεί inputFile, ανοίγει ο αρχείο αυτό και βρίσκει πόσες γραμμές περιέχει (δηλαδή πόσες εγγραφές φοιτητών) έτσι ώστε να ορίσουμε το κατάλληλο μέγεθος στο HashTable. Έπειτα αρχικοποιούμε τις δομές (initializeData) δίνοντας ως μέγεθος του HashTable τον αριθμό των γραμμών. Τέλος διαβάζουμε μία προς μία τις γραμμές και εισάγουμε τους φοιτητές στις δομές μας μέσω της insert.
Το configureFile χρησιμοποιείται μόνο σε περίπτωση που δεν δοθεί inputFile καθώς υπάρχει μέσα μόνο ένα standard μέγεθος για τον Hashable με το οποίο και και τον αρχικοποιούμε σε τέτοια περίπτωση. Άρα αποτελεί **απαραίτητη προϋπόθεση το εκτελέσιμο να κληθεί με τουλάχιστον ένα από τα δύο σαν παραμέτρους (inputFile ή configureFile).**
Οι δηλώσεις των συναρτήσεων αυτών γίνονται γνωστές στην main (mngstd.c) μέσω του αρχείου mngstdPrototypes.h

Αιτιολόγηση της επιλογής αυτής:
Η επιλογή να δημιουργηθεί η δομή Data καθώς και οι συναρτήσεις του αρχείου mngstdFunctions.c (αντί απλά να καλούνται από την main κατευθείαν συναρτήσεις όπως η htLookup ή η iiAverage) έγινε για λόγους απόκρυψης πληροφορίας. Πιο συγκεκριμένα, μέσω των παραπάνω, έχει επιτευχθεί απόλυτη απόκρυψη πληροφορίας καθώς η main (ή κάποιος που την χειρίζεται) δεν γνωρίζει ούτε τι δομές χρησιμοποιούνται για την αποθήκευση/διαχείριση των δεδομένων αλλά ούτε και το πώς υλοποιείται η κάθε λειτουργία. Το μόνο που γνωρίζει είναι ότι τα δεδομένα αποθηκεύονται σε μια δομή τύπου Data αλλά και τα πρωτότυπα των συναρτήσεων (insert,lookup,…).

Τα πλεονεκτήματα της απόκρυψης πληροφορίας είναι ευρέως γνωστά, για αυτό και έγινε αυτή η επιλογή.

Συνάρτηση main (mngstd.c):

Η συνάρτηση main αρχικά κάνει initialize τα δεδομένα μέσω των παραμέτρων που δόθηκαν στο command line μέσω της συνάρτησης readInput (όπως εξηγήθηκε παραπάνω) και τα αποθηκεύει σε μια μεταβλητή τύπου Data. Έπειτα ακολουθεί η υλοποίηση του prompt η οποία εξηγείται και στα σχόλια του κώδικα. Συνοπτικά:

- Αρχικά διαβάζεται το input του χρήστη.
- Μέσω της πρώτης λέξης του input (π.χ. i, n, c) εντοπίζουμε την λειτουργία που θέλει ο χρήστης να εκτελέσει και ανάλογα με αυτή το πρόγραμμα καταλήγει στο κατάλληλο if statement.
- Από εκεί διαβάζεται ο κατάλληλος αριθμός παραμέτρων ανάλογα με την λειτουργία (ο οποίος και ελέγχεται σε περίπτωση λάθος input). Αμέσως μετά καλείται η αντίστοιχη συνάρτηση του αρχείου mngstdFunction.c με ορίσματα τις παραμέτρους που μόλις διαβάστηκαν από την είσοδο του χρήστη.
- Αφού εκτελεστεί η λειτουργία αυτή, το πρόγραμμα περιμένει το επόμενο input του χρήστη το οποίο και διαβάζει.
- Η παραπάνω λειτουργία του prompt εκτελείται επαναληπτικά έως ότου δοθεί σαν input από τον χρήστη η λέξη exit.

Μόλις δοθεί η εντολή exit από τον χρήστη διαγράφουμε τα δεδομένα του προγράμματος μέσω της συνάρτησης deleteData (δηλαδή αποδεσμεύουμε όλες τις δομές που χρησιμοποιήθηκαν) και τερματίζεται το πρόγραμμα.

Παρατήρηση:
Έχει γίνει η παραδοχή ότι οι συμβολοσειρές studentId, lastName, firstName (στα δεδομένα των φοιτητών) έχουν το πολύ 20 χαρακτήρες.
(Μπορεί να αλλαχθεί στην γραμμή 19 του αρχείου mngstd.c)

Configure File:

Το configure file (configureFile.txt) χρησιμοποιείται για τον καθορισμό του μεγέθους του HashTable στην περίπτωση που δεν δοθεί κάποιο input file. Βέβαια μπορεί να δοθεί και σαν παράμετρο μαζί με κάποιο input file (π.χ. ./demo -i inputFile.txt -c configureFile.txt). Στην περίπτωση αυτή, βέβαια, δεν θα ληφθεί υπόψιν καθώς το μέγεθος θα καθοριστεί από το input file.
Το configure file είναι αρχείο τύπου .txt και περιέχει την λέξη HashTableSize ακολουθούμενη από έναν ακέραιο αριθμό. Ο αριθμός αυτός είναι το μέγεθος που θα πάρει το HashTable και μπορούμε να τον αλλάξουμε σε περίπτωση που θέλουμε το default μέγεθος να είναι κάποιο άλλο.

Πηγές:
Κάθε σημείο κώδικα γράφτηκε από έμενα συγκεκριμένα για αυτήν την εργασία. Δηλαδή δεν χρησιμοποιήθηκαν έτοιμα δικά μου κομμάτια κώδικα από παλαιότερα project και προφανώς δεν χρησιμοποιήθηκε κάποιο έτοιμο κομμάτι κώδικα από κάποια τρίτη πηγή (π.χ. Internet ή κάποιο άλλο άτομο).