Κ08 Δομές Δεδομένων και Τεχνικές Προγραμματισμού
Περιττοί ΑΜ

- Εαρινό Εξάμηνο 2018-2019
- Διδάσκων: Κώστας Χατζηκοκολάκης
- Εργασία 1
  - Ανακοινώθηκε στις 25 Φεβρουαρίου 2019
  - Προθεσμία παράδοσης: ~~24 Μαρτίου 2019~~ 29 Μαρτίου 2019, 12 τα μεσάνυχτα
  - 10% του συνολικού βαθμού στο μάθημα

Πιθανές αντιγραφές

Τα προγράμματα σας θα ελεγχθούν αυτόματα χρησιμοποιώντας κατάλληλο «έξυπνο» λογισμικό για ομοιότητες. Αν υπάρχουν ομοιότητες, όλες οι σχετικές ασκήσεις ή τμήματα ασκήσεων μηδενίζονται. Δεν δίνονται λεπτομέρειες για το πως και το γιατί, απλά ένα στρογγυλό μηδέν (0). Δεν θα έχετε τη δυνατότητα να μιλήσετε με τον καθηγητή ή τους βοηθούς του μαθήματος για περιπτώσεις αντιγραφής.

Κύριο πρόγραμμα

Σε όλες τις παρακάτω ασκήσεις ζητείται η υλοποίηση συνάρτησεων που έχουν κάποια λειτουργικότητα. Θα πρέπει μαζί με τη συνάρτηση αυτή να υλοποιήσετε και ένα κύριο πρόγραμμα (συνάρτηση main) το οποίο θα επιδεικνύει τη λειτουργικότητα **όλων των συναρτήσεων** για κατάλληλα επιλεγμένες εισόδους, και θα πείθει τον βαθμολογητή ότι οι συναρτήσεις λειτουργούν σωστά σε όλες τις ενδιαφέρουσες περιπτώσεις, ώστε να σας βαθμολογήσει με τον υψηλότερο δυνατό βαθμό. Ειναι προτιμότερο η main **να ΜΗΝ δέχεται είσοδο από το χρήστη**, αλλά να χρησιμοποιεί κατάλληλες προκαθορισμένες εισόδους (ώστε ο διορθωτής να μπορεί να την εκτελέσει εύκολα). Επίσης είναι χρήσιμο η main να τυπώνει μηνύματα που να εξηγούν το τί κάνει. Πχ μια ενδεικτική κλήση:

```
# make
...
# ./exercise1
Creating a new list using Create
Print:
```

```
Inseting AAA using InserLast
Print: AAA
Adding BBB at the beginning using InsertAfter
Print: BBB AAA
Adding CCC after AAA using InsertAfter
Print: BBB AAA CCC
Calling DeleteLast
Print: BBB AAA
...
```

Πως να παραδώσετε τις λύσεις σας

Οι λύσεις θα πρέπει να σταλούν στο e-mail `ddproj@di.uoa.gr` μέχρι την προθεσμία παράδοσης και να είναι οργανωμένες ως εξής. Θα στείλετε **ακριβώς ένα συμπιεσμένο αρχείο**. Η λύση κάθε προβλήματος θα είναι σε **χωριστό directory** το οποίο θα περιλαμβάνει τα source files που χρειάζονται, και ένα **Makefile** το οποίο θα μπορεί να χρησιμοποιηθεί για να μεταγλωττιστούν τα αρχεία σας και να παραχθεί το αντίστοιχο executable. Θα πρέπει να υπάρχει και ένα **αρχείο pdf** το οποίο θα περιέχει **όσο documentation χρειάζεται** ώστε οι βαθμολογητές να κατανοήσουν πλήρως τις λύσεις σας και να τις βαθμολογήσουν ανάλογα. Αυτό θα πρέπει να γίνει ανεξάρτητα με το αν είναι τα προγράμματα σας καλά σχολιασμένα, πράγμα που συνιστάται. Τέλος, το αρχείο pdf θα πρέπει να ξεκινά με το όνομα σας γραμμένο με Ελληνικούς χαρακτήρες και τον αριθμό μητρώου σας.

Παρατηρήσεις

- Οι λύσεις σας για όλες τις ασκήσεις πρέπει να είναι οργανωμένες σε modules της C όπως έχουμε συζητήσει στο μάθημα. Για κάθε module απαιτούνται τουλάχιστον 2 αρχεία, `module.h, module.c` και όπου χρειάζεται και ένα `moduleTypes.h`, όπως έχουμε δει στο μάθημα.
- **Δεν χρειάζεται** να ελέγχετε παντού ότι τα ορίσματα μιας συνάρτησης είναι σύμφωνα με τις προδιαγραφές. Ενας τέτοιος έλεγχος μπορεί να είναι χρονοβόρος (πχ στην `InsertAfter(list, item, node)` να ελέγξουμε ότι ο κόμβος ανήκει όντως στη λίστα), και άσκοπος (αν πχ η συνάρτηση καλείται μόνο από άλλες συναρτήσεις, όχι απ'ευθείας από το χρήστη, και πάντα με σωστό τρόπο). Σε τέτοιου είδους συναρτήσεις η σύμβαση συνήθως είναι ότι αν η είσοδος είναι λάθος το αποτέλεσμα είναι απρόβλεπτο.
- Συχνά πάλι είναι χρήσιμο να υπάρχουν τέτοιοι έλεγχοι, πχ όταν: τα δεδομένα έρχονται απ'ευθείας από το χρήστη, ο έλεγχος είναι εύκολος, γίνεται μια φορά μόνο στην αρχή, κλπ. Προσθέστε ελεύθερα ελέγχους όπου νομίζετε ότι είναι κατάλληλοι.
- Τα προγράμματα σας θα πρέπει να είναι όσο πιο καλά οργανωμένα γίνεται, σύμφωνα με όσα έχετε μάθει στο μάθημα «Εισαγωγή στον Προγραμματισμό».
- Χρησιμοποιήστε κατανοητά ονόματα μεταβλητών και προσθέστε σχόλια όπου χρειάζεται. Θυμηθείτε: όταν προγραμματίζουμε δεν μας ενδιαφέρει μόνο να τρέχει σωστά

το πρόγραμμα, αλλά και να μπορεί να γίνει κατανοητό από κάποιον που το διαβάζει (το διορθωτή, στην προκειμένη περίπτωση)

- Τα προγράμματα σας πρέπει να «τρέχουν» στους υπολογιστές του Τμήματος με το λειτουργικό linux αφού μεταφραστούν με τον μεταγλωττιστή gcc.

Καλή Επιτυχία!

Ασκηση 1 (25 μονάδες)

Οπως είδαμε στο μάθημα, η περίπτωση μια λίστα να είναι **κενή** απαιτούσε συχνά ειδική μεταχείριση, γιατί η αναπαράσταση μιας κενής λίστας ήταν ένας `NULL` δείκτης. Η υλοποίηση μπορεί να απλοποιηθεί αισθητά, αν χρησιμοποιήσουμε έναν `ξεχωριστό "εικονικό" κόμβο` (αποκαλούμενο "dummy" ή "sentinel") ο οποίος βρίσται πάντα στην **αρχή** της λίστας, **ακόμα και αν αυτή είναι κενή**. Ο dummy κόμβος θα πρέπει να έχει τον ίδιο τύπο με τους κανονικούς κόμβους (αλλά τα δεδομένα του δεν θα χρησιμοποιούνται). Υλοποιήστε μια συνδεδεμένη λίστα με dummy κόμβο για την αναπαράσταση πτήσεων, τροποποιώντας τις αντίστοιχες συναρτήσεις που είδαμε στο μάθημα. Θα πρέπει να ορίσετε τις κατάλληλες δομές της C και να υλοποιήσετε τις παρακάτω συναρτήσεις (με τα ορίσματα που δίνονται):

- `Create()`: δημιουργεί και επιστρέφει μια λίστα
- `Print(list)` : τυπώνει τη λίστα
- `Search(list, item)` : επιστρέφει τον πρώτο κόμβο με κωδικό αεροδρομίου `item`
- `InsertLast(list, item)` : προσθέτει νέο κόμβο στο τέλος, με κωδικό αεροδρομίου `item`
- `InsertAfter(list, item, node)` : προσθέτει νέο κόμβο με κωδικό αεροδρομίου `item` μετά τον `node`
- `DeleteLast(list)` : διαγράφει τον τελευταίο κόμβο
- `Delete(list, node)` : διαγράφει τον κόμβο `node`

Ενα πλεονέκτημα της τροποποίησης αυτής είναι ότι οι συναρτήσεις μπορούν (και ζητείται) να υλοποιηθούν **χωρίς τη χρήση pointer σε pointer**.

Ας σημειωθεί ότι μια υλοποίηση λίστας μπορούμε να προσθέσουμε είτε dummy κόμβο, είτε "header" κόμβο ("κεφαλή") με επιπλέον δεδομένα (πχ τον συνολικό αριθμό κόμβων), είτε και τα δύο. Στην άσκηση αυτή ζητείται μόνο dummy.

Ασκηση 2 (25 μονάδες)

Τροποποιήστε την προηγούμενη άσκηση ώστε να υλοποιήσετε τον αφηρημένο τύπο δεδομένων **Διπλά Συνδεδεμένη Λίστα**. Σε μια τέτοια λίστα κάθε κόμβος έχει δείκτη στον επόμενο αλλά και στον **προηγούμενο**. Θα υλοποιήσετε τις ίδιες συναρτήσεις με την προηγούμενη άσκηση, με τις παρακάτω τροποιήσεις:

- Θα πρέπει να υπάρχουν dummy κόμβοι **στην αρχη και στο τέλος** (που να έχουν τον ίδιο τύπο με τους κανονικούς κόμβους).
- Ο τύπος δεδομένων θα πρέπει να είναι `ItemType` (`typedef` ορισμένο από το χρήστη της λίστας).
- Η συνάρτηση `Search`, για να μπορεί να λειτουργήσει για έναν οποιοδήποτε τύπο δεδομένων `ItemType`, θα πρέπει να παίρνει ως όρισμα (έναν pointer σε) μια συνάρτηση `compare` που συγκρίνει 2 τιμές και επιστρέφει 0 αν είναι ίδιες και μη μηδενική τιμή αν διαφέρουν. Για να γίνει αυτό:
  - Θα ορίσετε έναν τύπο `CompareType` ως έναν pointer σε συνάρτηση που παίρνει 2 ορίσματα `a,b` τύπου `ItemType` και επιστρέφει `int`. Αυτό γίνεται ως εξης:
    ```
    typedef int (*CompareType)(ItemType a, ItemType b);
    ```
  - Η συνάρτηση εύρεσης θα πρέπει να οριστεί ως εξής:
    ```
    Search(ListType list, ItemType item, CompareType compare)
    ```
    Και θα πρέπει να επιστρέψει τον πρώτο κόμβο `node` για το οποίο:
    ```
    compare(item, node->item) == 0
    ```
- Η συνάρτηση εκτύπωσης θα πρέπει να οριστεί ως `Print(list, format)`, όπου το format που θα περνάει ο χρήστης θα αντιστοιχεί στο `ItemType` που χρησιμοποιεί, πχ `"%d"`.
- Θα πρέπει να προσθέσετε και μια συνάρτηση

  `InsertBefore(list, item, node)`

  που προσθέτει νέο κόμβο με δεδομένα `item` **πριν** τον `node`.

Τέλος, στο πρόγραμμα main, θα πρέπει να χρησιμοποιήσετε την υλοποίηση που φτιάξατε για να διαχειριστείτε λίστες πτήσεων. Για το σκοπό αυτό θα πρέπει να ορίσετε το `ItemType` ως `AirportCode` (ώστε η λίστα να αποθηκεύει κωδικούς αεροδρομίων) και να υλοποιήστε μια κατάλληλη συνάρτηση `compare` για χρήση με τη `Search`. Θα πρέπει να υπάρχουν παραδείγματα χρήσης όλων των συναρτήσεων διαχείρισης της λίστας.

(Ας σημειωθεί ότι μια πλήρης υλοποίηση ενός ADT διπλής συνδεδεμένης λίστας μπορεί να περιέχει πολλές ακόμα ενέργειες, πχ `Size, IsEmpty, GetFirst, GetNext, GetPrev, GetItem` κλπ. Οποιος ενδιαφέρεται μπορεί να υλοποιήσει και τέτοιου είδους ενέργειες, αν και δεν ζητείται για την άσκηση.)

Ασκηση 3 (25 μονάδες)

Υλοποιήστε μια συνάρτηση η οποία παίρνει σαν όρισμα **δύο ταξινομημένες** (σε αύξουσα σειρά) απλά συδεδεμένες **λίστες ακεραίων**, τις ενώνει σε μια ταξινομημένη λίστα (τροποποιώντας τους υπάρχοντες κόμβους, χωρίς νέα εκχώρηση μνήμης), και επιστρέφει την ενωμένη λίστα. Μπορείτε να χρησιμοποιήσετε την υλοποίηση απλά συνδεδεμένων λιστών του μαθήματος (προσθέτωντας και κόμβο κεφαλής, αν επιθυμείτε). **Δεν χρειάζεται να ελέγξετε** ότι οι λίστες είναι ήδη ταξινομημένες (η σύμβαση με το χρήστη της συνάρτησης είναι ότι αν οι λίστες δεν είναι ταξινομημένες δεν υποσχόμαστε τίποτα για το αποτέλεσμα).

Ασκηση 4 (25 μονάδες)

Ο αλγόριθος ταξινόμησης **MergeSort** μπορεί να εφαρμοστεί σε μια απλά συνδεδεμένη λίστα ως εξής: πρώτα βρίσκουμε τη **μέση** της λίστας (αυτό μπορεί να γίνει σε ένα πέρασμα διατηρώντας έναν "αργό" δείκτη που προχωράει ένα κόμβο τη φορά, και ένα "γρήγορο" που προχωράει ανά δύο κόμβους). Στη συνέχεια **σπάμε τη λίστα** στα δύο, ταξινομούμε **αναδρομικά** το κάθε μισό, και μετά **ενώνουμε** τις δύο ταξινομημένες λίστες. Υλοποιήστε τον αλγόριθμο αυτό για λίστες ακεραίων, τροποποιώντας την υλοποίηση απλά συνδεδεμένων λιστών του μαθήματος (προσθέτωντας και κόμβο κεφαλής, αν επιθυμείτε).

Ασκηση 5 (25 μονάδες)

Υλοποιήστε μια συνάρτηση `Similar(word)` η οποία παίρνει σαν όρισμα μία λέξη `word` και επιστρέφει μια **λίστα** με όλες τις λέξεις ενός λεξικού που διαφέρουν από την `word` κατά **ακριβώς ένα γράμμα** (και έχουν το ίδιο μήκος).

Το πρόγραμμα θα πρέπει να δουλεύει ως εξής: αρχικά θα διαβάζει το λεξικό από το αρχείο `words` που δίνεται μαζί με την εκφώνηση, θα το αποθηκεύει σε έναν πίνακα και θα **ταξινομεί** τον πίνακα. Στη συνέχεια, θα δοκιμάζει να αλλάζει έναν-έναν τους χαρακτήρες του `word` δοκιμάζοντας όλους τους συνδυασμούς από 'a' έως 'z'. Για κάθε "υποψήφια" λέξη που δημιουργείται, η συνάρτηση θα χρησιμοποιεί **δυαδική αναζήτηση** στον πίνακα όλων των λέξεων για να βρει **αν η λέξη είναι έγκυρη** ή όχι. Αν ναι, τότε θα την προσθέτει στη λίστα.

Παράδειγμα, η `Similar("art")` θα πρέπει να επιστρέφει τη λίστα:

act, aft, ant, apt, ara, arc, are, ark, arm

Μπορείτε να χρησιμοποιήσετε τις υλοποιήσεις ταξινόμησης και δυαδικής αναζήτησης που είδατε στην εισαγωγή στον προγραμματισμό (ή, αν προτιμάτε, τις αντίστοιχες συναρτήσεις `qsort, bsearch` της C). Ο πίνακας λέξεων μπορεί είτε να αποθηκευτεί σε global μεταβλητή (αν το επιθυμείται), είτε να περνιέται ως παράμετρος στην Similar.

Ασκηση 6 (30 μονάδες)

Θεωρήστε το ακόλουθο πρόβλημα: μας δίνονται **δύο λέξεις Α και Β ίδιου μήκους** από ένα λεξικό. Σκοπός μας είναι να **μετατρέψουμε την Α στη Β** με τον ακόλουθο τρόπο:

- Σε κάθε βήμα μπορούμε να τροποποιούμε **μόνο ένα γράμμα της τελευταίας λέξης**.
- Κάθε ενδιάμεση λέξη πρέπει να είναι **έγκυρη λέξη του λεξικού**.

Πχ μπορούμε να μετατρέψουμε το Α="dog" σε Β="cat" ως εξής:

dog -> cog -> cot -> cat

Φτιάξτε ένα πρόγραμμα το οποίο θα βρίσκει και θα τυπώνει τη **βέλτιστη λύση** σε αυτό το πρόβλημα, δηλαδή τη μετατροπή με τα λιγότερα δυνατά βήματα. **Αν δεν υπάρχει καμία**

έγκυρη μετατροπή το πρόγραμμα θα πρέπει να το ανακαλύπτει (σκεφτείτε το πώς, είναι απλό) και να το τυπώνει!

Θα αναπαριστούμε μία **μετατροπή** λέξεων με μια **λίστα** που περιέχει τις αντίστοιχες λέξεις σε **αντίστροφη σειρά** (πχ: cat -> cot -> dot -> dog για το παραπάνω παράδειγμα). Το πρόγραμμα σας θα χρησιμοποιεί μια **ουρά** (μπορείτε να χρησιμοποιήσετε την υλοποίηση που δόθηκε στην ενότητα "Ουρές" του μαθήματος, κατάλληλα τροποποιημένη, η οποία βρίσκεται στην ιστοσελίδα του μαθήματος). Η ουρά χρησιμοποιείται για να αποθηκεύσουμε τις διάφορες μετατροπές που εξετάζουμε, οπότε **τα δεδομένα της ουράς θα είναι μετατροπές** (δηλαδή λίστες).

Ξεκινάμε τοποθετώντας στην ουρά την κενή μετατροπή που περιέχει **μόνο το A**. Στη συνέχεια εκτελείται μια επανάληψη (loop): σε κάθε βήμα, αφαιρούμε το πρώτο στοιχείο της ουράς. Αν η μετατροπή αυτή καταλήγει στο B τελειώσαμε. Αν όχι, **επεκτείνουμε τη μετατροπή** με όλους τους δυνατούς τρόπους, βρίσκοντας δηλαδή όλες τις λέξεις που διαφέρουν κατά ένα μόνο γράμμα από την τελευταία λέξη (χρησιμοποιώντας την υλοποίηση της προηγούμενης άσκησης), και που **δεν περιέχονται ήδη στην μετατροπή**. Οι νέες μετατροπές που δημιουργούμε προστίθενται στην ουρά, και η διαδικασία συνεχίζεται.

Το πρόγραμμα θα πρέπει να τυπώνει τη λύση, καθώς και τον αριθμό βημάτων που χρειάστηκαν για να βρεθεί (πόσες φορές δηλαδή εκτελέστηκε η παραπάνω επανάληψη).

Μερικά ακόμα παραδείγματα βέλτιστων μετατροπών:

cans, cons, coos, coon, coin

dart, tart, tort, tors, togs, dogs

Προσοχή: ο αλγόριθμος αυτός μπορεί να χρειαστεί πολύ χρόνο για να τερματίσει για μεγάλες μετατροπές. Δοκιμάστε πρώτα το παράδειγμα dog -> cat.

Ασκηση 7 (25 μονάδες)

Θα βελτιώσουμε την απόδοση του προγράμματος της προηγούμενης άσκησης, αντικαθιστώντας την ουρά που αποθηκεύουμε τις μετατροπές με μια **ουρά προτεραιότητας**. Κάθε μετατροπή m που προστίθεται στην ουρά συνοδεύεται από μία **αξιολόγηση** $f(m)$, που δίνεται από την παρακάτω συνάρτηση:

$$f(m) = g(m) + h(m)$$

Η συνάρτηση $g(m)$ είναι ο αριθμός βημάτων της μετατροπής (πόσες λέξεις περιέχει πέραν της αρχικής). Η συνάρτηση $h(m)$ ορίζεται ως ο αριθμός των γραμμάτων που διαφέρει η τελευταία λέξη της m, από το στόχο B (η $h(m)$ ονομάζεται "ευρετική" συνάρτηση).

Η διαδικασία είναι ακριβώς η ίδια, με μόνη εξαίρεση το γεγονός ότι σε κάθε επανάληψη αφαιρούμε τη μετατροπή με το **μικρότερο** $f(m)$ (μικρότερες τιμές της f θεωρούνται "μεγαλύτερη προτεραιότητα". Υλοποιήστε τη μέθοδο αυτή, και συγκρίνετε (πειραματικά) την αποδοσή της, δηλαδή τον αριθμό επαναλήψεων μέχρι να βρεθεί η λύση, με τη μέθοδο της προηγούμενης άσκησης.

Μερικά μεγαλύτερα παραδείγματα που η μέθοδος αυτή μπορεί να λύσει:

cans, cons, coos, coop, clop, clap, claw

cans, pans, pwns, owns, owes, awes, awed, abed, abel

table, gable, gayle, gayly, gaily, daily, drily, drill, trill, trial, triad, tread, bread

table, gable, gayle, gayly, gaily, daily, dally, daley, dales, dates, oates, oaten, eaten, eaton, elton, alton, acton