Δομές Δεδομένων

Οι ασκήσεις από το εργαστήριο σε γλώσσα C.

Η ονοματολογία των ασκήσεων έχει ως εξής:

a: συμβολίζει την λέξη άσκηση
f: συμβολίζει το φυλλάδιο

Τα φυλλάδια ΔΕΝ είναι δικιά μου ιδιοκτησία.

Περιεχόμενα

- a1f1.c: Εισάγουμε 2 θετικούς ακέραιους αριθμούς first < last, βρίσκει τους πρώτους αριθμούς (έχει δυο θετικούς διαιρέτες, τον εαυτό του και το 1) και τους τοποθετεί σε ένα σύνολο και τους εμφανίζει.
- a5f1.c: Εισάγουμε ένα αριθμό και δημιουργείται ένα σύνολο με Fibonacci. Στη συνέχεια να γράφει αριθμούς και εμφανίζει μήνυμα αν είναι ή όχι μέσα στο σύνολο των όρων Fibonacci.
- a5f2.c: Στοίβα 15 αριθμών όπου εισάγουμε ένα αριθμό μικρότερο του 6 και εμφανίζονται διάφοροι αριθμοί της στοίβας. Ουσιαστικά γίνεται χρήση στοιβών, βγάλε, βάλε, μετακίνησε σε 2η στοίβα, ξαναβάλε τα στοιχεία πίσω κοκ.
- a8f2.c: Εισάγουμε σε μια στοίβα χαρακτήρες (typedef **char** StackElementType), όπου μετά από την εισαγωγή του C πρέπει να ελέγχουμε αν οι χαρακτήρες που εισάγονται, είναι οι "αντίστροφοι" από αυτούς που εισήχθησαν (x C y, με έλεγχο αν x=y). Εμφάνιση ανάλογων μηνυμάτων TRUE/FALSE.
- a16f2.c: Γίνεται εισαγωγή φανελλών με μέγεθος και τιμής (ΠΡΟΣΟΧΗ στο StackElementType), σε μια στοίβα. Γίνεται ερώτηση για μέγεθος φανέλλας και ουσιαστικά γίνεται σύγκριση των στοιχείων που εξάγονται (και εισάγονται σε βοηθητική στοίβα). Αν βρεθεί το μέγεθος, εμφανίζει μήνυμα ενώ αν δεν βρεθεί, εμφανίζεται ανάλογο μήνυμα. Τελικά εμφανόζονται οι φανέλλες που βρίσκονται εκτός αρχικής στοίβας, μετά από μεταφορές κλπ.
- a17f2.c: Δημιουργούμε μια στοίβα s1 με αριθμούς. Στην συνέχεια πρέπει να υλοποιήσουμε μια συνάρτηση StackType CopyStack(StackType *s1), στην οποία θα δημιουργηθεί μια στοίβα s2 (μέσω μιας ενδιάμεσης στοίβας) και θα επιστραφεί στο κυρίως πρόγραμμα, όπου και θα εκτυπωθεί.
- a6f3.c: Δημιουργία κυκλικής ουράς στην οποία θα εισάγουμε 10 αριθμούς, να τροποποιήσουμε όμως τον ΑΔΤ με την προσθήκη ενός ακέραιου πεδίου Count στην εγγραφή τύπου QueueType, έτσι ώστε να μην έχουμε στην ουρά μια θέση κενή (που την χρησιμοποιούμε για να δείξουμε ότι η ουρά είναι γεμάτη). Στο πρόγραμμα μέσα χειριζόμαστε την προσθήκη-αφαίρεση και εμφάνιση της ουράς.
- a12f3.c: Προσομοίωση ουράς σε ταμείο τράπεζας (3 πελάτες άρα μέγεθος της ουράς=4). Τροποποίηση του QueueElementType έτσι ώστε να εισάγουμε τους χρόνους εισόδου, εξυπηρέτησης κλπ). Δημιουργούμε 2 ουρές όπου η μία περιέχει στοιχεία για τους πελάτες (αρχικοποίηση) και η άλλη με τους χρόνους που έχουν εξυπηρετηθεί τελικά οι πελάτες. Ουσιαστικά παίρνουμε τα στοιχεία από την μια ουρά και με αυτά με πράξεις, φτιάχνουμε την δεύτερη.
- a1f4.c: Κατασκευή μιας συνάρτησης boolean Search (επιστρέφει αν υπάρχει ή όχι το στοιχείο). Εισάγουμε στοιχεία, χρησιμοποιείται η συνάρτηση search και εισάγεται το στοιχείο έτσι ώστε η λίστα να παραμείνει ταξινομημένη. Στη συνέχεια γίνεται αναζήτηση για ένα στοιχείο στην λίστα και εμφανίζει μήνυμα αν βρίσκεται στην λίστα ή όχι.
- a30f4.c: Κατασκευή μιας συνάρτησης float FindMins (επιστρέφει τον μικρότερο βαθμό και μια στοίβα με τα ΑΜ με τον μικρότερο βαθμό). Χρησιμοποιούμε και στοίβες. Στον ListElementType εισάγουμε ΑΜ (int) και βαθμό (float). Στη συνέχεια εισάγουμε τα στοιχεία ενός μαθητή (ΑΜ και βαθμός). Χρησιμοποιούμε την συνάρτηση FindMins για να βρούμε το μικρότερο βαθμό που εισήχθη, και να επιστραφεί η λίστα με τα ΑΜ με τον μικρότερο βαθμό. Στην συνέχεια εμφανίζουμε την λίστα και την στοίβα.
- a2cf4.c: Συνένωση 2 απλών συνδετικών λιστών σε μία με χρήση της συνάρτησης concat_list την οποία πρέπει να κατασκευάσουμε εμείς.
- a2jf4.c: Να γραφεί συνάρτηση insert_list_m_elements στην οποία θα γίνεται η προσθήκη αριθμών που δίνει ο χρήστης, μετά από στοιχείο της λίστας που εισάγει ο χρήστης.
- a2rf4.c: Υλοποίηση συνάρτησης RemoveMins η οποία δέχεται μια λίστα, βρίσκει το ελάχιστο στοιχείο της λίστας και αφαιρεί από τη λίστα όλους τους αριθμούς ίσους με τον ελάχιστο αριθμό και

τον επιστρέφει. Η εμφάνιση του ελάχιστου γίνεται στη main().

- a9f4.c: Πρόγραμμα που διαβάζει ένα αλφαριθμητικό και στη συνέχεια να προσθέτει έναν-έναν τους χαρακτήρες του σε μια στοίβα και σε μια ουρά ταυτόχρονα. Στη συνέχεια θα ελέγχετε αν το αλφαριθμητικό είναι καρκινικό, οπότε κι θα εμφανίζετε το μήνυμα 'ACCEPTED', ή όχι, οπότε θα εμφανίζετε το μήνυμα 'REJECTED'.
- a10f4.c: Πρόγραμμα που προσομοιώνει μια ουρά με τη βοήθεια δύο στοιβών, δηλαδή οι λειτουργίες της ουράς θα προσομοιώνονται με τις λειτουργίες της στοίβας. Αντί να χρησιμοποιηθεί μια ουρά αρκεί να χρησιμοποιηθούν 2 στοίβες. Κάθε κόμβος περιέχει έναν ακέραιο αριθμό και η εισαγωγή δεδομένων θα γίνεται ως εξής: πλήθος στοιχείων, στοιχεία. Το πρόγραμμα θα εμφανίζει τα περιεχόμενα και των 2 στοιβών. Η 2η στοίβα έχει καταχωρημένα τα στοιχεία όπως θα ήταν αν είχαμε χρησιμοποιήσει μια ουρά.
- a16f4.c: Πρόγραμμα που εισάγεται το πλήθος των κόμβων της ουράς. Με τη χρήση της συνάρτησης void insert_prot(QueueType *Queue, QueueElementType Item) η εισαγωγή στοιχείων στην ουρά, εισάγεται ο κάθε κόμβος που περιέχει έναν τριψήφιο κωδικό αριθμό και τον βαθμό προτεραιότητας (1-20). Σε περίπτωση ίδιου βαθμού προτεραιότητας το στοιχείο εισέρχεται τελευταίο στην αντίστοιχη προτεραιότητα. Τη συνάρτηση void TraverseQ(QueueType Queue) που θα εμφανίζει τα περιεχόμενα της ουράς κατά αύξοντα βαθμό προτεραιότητας. Τα στοιχεία κάθε κόμβου εμφανίζονται σε ξεχωριστή σειρά με ένα κενό μεταξύ τους και πρώτο το βαθμό προτεραιότητας.
- a11f5.c: Πρόγραμμα που πραγματοποιεί ορθογραφικό έλεγχο ενός κειμένου. Οι λέξεις που απαρτίζουν το λεξικό είναι αποθηκευμένες στο αρχείο κειμένου *I112f5.TXT*. Διαβάζονται και αποθηκεύονται μία-μία σε ένα ΔΔΑ με τη συνάρτηση *CreateDictionary* και έτσι δημιουργείται το λεξικό-ΔΔΑ. Μετά τη δημιουργία του λεξικού-ΔΔΑ εμφανίζονται οι λέξεις του λεξικού. Στη συνέχεια το πρόγραμμα μέσω της συνάρτησης *SpellingCheck* διαβάζει ένα κείμενο που είναι αποθηκευμένο στο αρχείο κειμένου *I111F5.TXT*, σε κάθε γραμμή του οποίου υπάρχει μία λέξη και θα διενεργεί ορθογραφικό έλεγχο, δηλαδή θα αναζητά τη λέξη του αρχείου *I111F5.TXT* στο λεξικό-ΔΔΑ (χρήση strcmp). Κατά τον ορθογραφικό έλεγχο θα πρέπει να εκτυπώνονται λέξεις που δεν βρέθηκαν στο λεξικό και να υπολογίζεται το πλήθος τους.
- a29f5.c: Πρόγραμμα που δέχεται για κάθε άτομο τον ΑΜΚΑ, τον ΑΦΜ, την ηλικία. Καταχωρεί τα στοιχεία του κάθε ατόμου σε 2 καταλόγους ανάλογα με την ηλικία του, αυτούς με ηλικία μικρότερη ή ίση των 60 ετών και αυτούς με ηλικία μεγαλύτερη των 60. Κάθε κατάλογος θα πρέπει να οργανωθεί ως ΔΔΑ με κλειδί τον ΑΜΚΑ. Θα εμφανίσει τους 2 καταλόγους. Γίνεται αναζήτηση ατόμου με βάση τον ΑΜΚΑ και την ηλικία και εμφανίζει ανάλογα μηνύματα.
- a25f5.c: Υλοποίηση μιας αναδρομικής συνάρτησης **int CountLeaves(BinTreePointer Root)** η οποία θα υπολογίζει το πλήθος των φύλλων ενός ΔΔΑ το οποίο και θα επιστρέφει. Στο ΔΔΑ θα καταχωρούνται ακέραιοι. Θα εμφανίζονται οι κόμβοι του ΔΔΑ σε αύξουσα διάταξη και στη συνέχεια θα εμφανίζεται το πλήθος των φύλλων του ΔΔΑ στο κυρίως πρόγραμμα.
- a26f5.c: Δίνεται το αρχείο φοιτητών "foitites.dat". Πρόγραμμα που εκτελεί κάποιες από τις παρακάτω λειτουργίες και χρησιμοποιεί ως ευρετήριο ένα ΔΔΑ. Δημιουργία του index (ΔΔΑ) από το αρχείο "foitites.dat" (υλοποίηση της συνάρτησης **int BuildBST(BinTreePointer *Root)**). Εμφανίζει το πλήθος των κόμβων του ΔΔΑ όπως και τους κόμβους του ΔΔΑ με αύξουσα διάταξη ως προς ΑΜ (ενδιδοατεταγμένη διάσχιση). Εισαγωγή νέων εγγραφών φοιτητών στο αρχείο foitites.dat και ενημέρωση του ΔΔΑ. Υλοποίηση της συνάρτησης **void writeNewStudents(BinTreePointer *Root, int* size)**. Γίνεται αναζήτηση φοιτητή με το ΑΜ του φοιτητή και θα τον αναζητά στο ΔΔΑ. Στη συνέχεια εφόσον υπάρχει στο ΔΔΑ θα τον εντοπίζει στο αρχείο "foitites.dat" και θα εμφανίζει όλες τις πληροφορίες της αντίστοιχης εγγραφής. Θα εισάγεται ο βαθμός του ΜΟ και θα εκτυπώνονται τα στοιχεία όλων των φοιτητών που είναι καταχωρημένοι στο αρχείο "foitites.dat" με ΜΟ μεγαλύτερο από τον δοσμένο βαθμό (πχ 0). Αυτό θα γίνεται με υλοποίηση της συνάρτησης **void printStudentsWithGrade(float theGrade)**.
- a30f5: Το αρχείο transactions.txt περιλαμβάνει ύποπτες συναλλαγές και θέλουμε να εντοπίσουμε τις m μεγαλύτερες συναλλαγές. Η διαθέσιμη μνήμη δεν επαρκεί για να διαβάσουμε όλες τις συναλλαγές από το αρχείο και να τις αποθηκεύσουμε σε μια δομή δεδομένων στη μνήμη. Επιλέξτε την κατάλληλη δομή δεδομένων ώστε να βρίσκει τις m μεγαλύτερες συναλλαγές. Μετά την εύρεση των m μεγαλύτερων συναλλαγών θα εμφανίζει το μέγεθος και τα στοιχεία της δομής δεδομένων (προσαρμόστε κατάλληλα την PrintHeap) και στη συνέχεια θα εμφανίσει σε αύξουσα διάταξη τις m μεγαλύτερες συναλλαγές. Την τιμή της m θα τη δίνει ο χρήστης, και θεωρήστε ότι δίνεται τιμή πολύ μικρότερη του μεγέθους του αρχείου χωρίς να γίνεται σχετικός έλεγχος. Ως δομή δεδομένων θα πρέπει να χρησιμοποιηθεί σωρός (μέγιστος ή **ελάχιστος** σωρός;).

- a7f6: Πρόγραμμα για τη δημιουργία και επεξεργασία μιας ΔΔ που αποθηκεύει και επεξεργάζεται τα στοιχεία της με την τεχνική του κατακερματισμού με αλυσίδες συνωνύμων, στην οποία αποθηκεύονται τα στοιχεία των μελών ενός γυμναστηρίου. Υλοποίηση συνάρτησης void PrintListOfSynonyms(HashListType HList, int key).
- a4f6: Σε έναν εκπαιδευτικό οργανισμό εργάζονται εκπαιδευτικοί διαφόρων ειδικοτήτων. Τα βασικά τους στοιχεία υπάρχουν σε ένα αρχείου κειμένου 'i4f6.txt'. Στο κυρίως πρόγραμμα θα υλοποιούνται στη σειρά οι παρακάτω λειτουργίες:
    1. BuildHashList: Διάβασμα των στοιχείων από το αρχείο κειμένου και δημιουργία Δομής Δεδομένων (ΔΔ) που αποθηκεύει και επεξεργάζεται τα στοιχεία της με την τεχνική του κατακερματισμού με αλυσίδες συνωνύμων. Το κλειδί σχηματίζεται από το όνομα+κενό χαρακτήρα+επώνυμο. **void BuildHashList(HashListType *HList);**
    2. Insert new teacher: Εισαγωγή των στοιχείων ενός νέου εκπαιδευτικού στη ΔΔ κατακερματισμού με αλυσίδες συνωνύμων. Η εισαγωγή νέων εκπαιδευτικών στη ΔΔ γίνεται επαναληπτικά μέσω σχετικού μηνύματος 'Continue Y/N?'
    3. Delete a teacher: Διαγραφή ενός εκπαιδευτικού από τη ΔΔ κατακερματισμού με αλυσίδες συνωνύμων
    4. Search for a teacher: Αναζήτηση και εμφάνιση των στοιχείων ενός εκπαιδευτικού βάσει ονοματεπωνύμου στη ΔΔ κατακερματισμού με αλυσίδες συνωνύμων.
    5. Search by subject: Αναζήτηση και εμφάνιση των στοιχείων των εκπαιδευτικών μιας συγκεκριμένης ειδικότητας (ο κωδικός της ειδικότητας [1..20] αποτελεί παράμετρο της διαδικασίας) στη ΔΔ κατακερματισμού με αλυσίδες συνωνύμων **void Search_HashList_By_Subject(HashListType HList, int code);**

  Μετά τις λειτουργίες 1, 2, 3 θα καλείτε την PrintPinakes(HList). Ο πίνακας κατακερματισμού θα έχει 9 θέσεις και ως συνάρτηση κατακερματισμού θα χρησιμοποιηθεί η εξής:
  h(i) = average % 9
  όπου
  average = (κωδικός*πρώτου* χαρακτήρα + κωδικός*τελευταίου*χαρακτήρα) / 2 Θεωρήστε ότι χρησιμοποιούνται οι ακόλουθοι κωδικοί για τους χαρακτήρες: 'A' = 1, 'B' = 2, …, 'Z' = 26. Ο πρώτος και ο τελευταίος χαρακτήρας του ονοματεπωνύμου θα μετατρέπεται στον αντίστοιχο κεφαλαίο χαρακτήρα, εφόσον είναι πεζός. Ο μέσος όρος average θα υπολογίζεται με μια συνάρτηση findAverage, η οποία θα καλείται από τη συνάρτηση HashKey.

```
/*  Αρχείο: a1f1.c
    Φοιτητής: Ευστάθιος Ιωσηφίδης


    Άδεια χρήσης: GNU General Public License v3.0

    Ένα σημαντικό πρόβλημα της θεωρίας αριθμών (του πεδίου των μαθηματικών που έχει ως
αντικείμενο τη
    μελέτη των ιδιοτήτων των μη αρνητικών ακεραίων) είναι το πρόβλημα του προσδιορισμού
του κατά πόσο
    ένας ακέραιος είναι πρώτος. Ένας θετικός ακέραιος n είναι πρώτος (prime) αν έχει
ακριβώς δύο θετικούς
    διαιρέτες, τον εαυτό του και το 1. Οι πρώτοι αριθμοί είναι ιδιαίτερα σημαντικοί
σήμερα για την
    κρυπτογραφία, αφού πολλές από τις καλύτερες τεχνικές κρυπτογράφησης βασίζονται στους
πρώτους
    αριθμούς. Να γραφεί πρόγραμμα που θα περιλαμβάνει τα παρακάτω υποπρογράμματα:
    □ συνάρτηση isPrime που θα δέχεται ένα θετικό ακέραιο n και θα επιστρέφει την τιμή
TRUE ή FALSE
    ανάλογα με το αν ο n είναι ή όχι αντίστοιχα πρώτος αριθμός
    □ διαδικασία createPrimeSet που θα δέχεται δύο θετικούς ακεραίους, έστω first και
last, και θα
    δημιουργεί και επιστρέφει το σύνολο των πρώτων αριθμών που ανήκουν στο διάστημα
[first .. last]
    □ διαδικασία displaySet που δέχεται τον πρώτο αριθμό first ενός συνόλου θετικών
ακεραίων S και
    εμφανίζει τα στοιχεία του συνόλου στην ίδια γραμμή με ένα κενό χαρακτήρα μεταξύ τους.
    Στη συνέχεια, γράψτε κυρίως πρόγραμμα όπου θα διαβάζονται δύο ακέραιοι αριθμοί, έστω
first και last,
    που θα πρέπει να ανήκουν στο διάστημα [2..200] και επιπλέον να ισχύει first < last.
Στη συνέχεια, θα
    καλούνται οι διαδικασίες createPrimeSet και displaySet για τη δημιουργία και εμφάνιση
του συνόλου των
    πρώτων αριθμών που ανήκουν στο διάστημα [first .. last].
    Να χρησιμοποιηθεί ο ΑΤΔ σύνολο με πίνακα. Η διαδικασία displaySet θα δέχεται ένα
σύνολο θετικών
    ακεραίων S, τον πρώτο (first) και τον τελευταίο (last) αριθμό αυτού του συνόλου.
*/

#include <stdio.h>
#include <stdlib.h>

#define megisto_plithos 201
#define low 2
#define up 200

typedef enum {
    FALSE, TRUE
} boolean;

typedef boolean typos_synolou[megisto_plithos];
typedef int stoixeio_synolou;

boolean isPrime(stoixeio_synolou n);
void createPrimeSet(stoixeio_synolou first, stoixeio_synolou last,typos_synolou S);
void displaySet(stoixeio_synolou first,stoixeio_synolou last,typos_synolou S);

void Dimiourgia(typos_synolou synolo);
boolean Melos(stoixeio_synolou stoixeio, typos_synolou synolo);
void Eisagogi(stoixeio_synolou stoixeio, typos_synolou synolo);

int main()
{
    stoixeio_synolou first,last; //Ορισμός πρώτου και τελευταίου θετικού ακεραίου αριθμού
    typos_synolou S; // Ορισμός συνόλου

/* Εισαγωγή πρώτου ακεραίου αριθμού. Εκτέλεση τουλάχιστον μια φορά με το do, και έλεγχος
να είναι μεγαλύτερο ή ίσο
```

```
από το ελάχιστο που ορίσαμε (2) και να είναι first < last.
*/
    do{
        printf("\n");
        printf("DOSTE ARXIKH TIMH (APO 2 EOS 199 KAI PREPEI NA EINAI MIKROTERH THS
TELIKHS TIMHS POY THA EISAGEIS META): ");
        scanf("%d",&first);
    }while(!(first >= low && first < up));

/* Εισαγωγή δεύτερου ακεραίου αριθμού. Εκτέλεση τουλάχιστον μια φορά με το do, και
έλεγχος να είναι μεγαλύτερο ή ίσο
από το ελάχιστο που ορίσαμε (2) και να είναι μικρότερο ή ίσο από το ανώτερο όριο που
ορίσαμε (200).
*/
    do{
        printf("\n");
        printf("DOSTE TELIKH TIMH (APO 2 EOS 200 KAI NA EINAI MEGALYTERH TOY %d):
",first);
        scanf("%d",&last);
    }while(!(last >= low && last <= up) || !(first<last));

    printf("\n");
    createPrimeSet(first,last,S); // Δημιουργείται το σύνολο πρώτων αριθμών με όρια από
τον first μέχρι τον last
    displaySet(first,last,S); // Εμφάνιση του συνόλου

    return 0;
}


/* ΣΥΝΑΡΤΗΣΕΙΣ */

// Ελέγχει αν ένας αριθμός είναι πρώτος ή όχι. Ένας θετικός ακέραιος n είναι πρώτος
(prime) αν έχει ακριβώς δύο θετικούς διαιρέτες, τον εαυτό του και το 1.

boolean isPrime(stoixeio_synolou n)
{
    stoixeio_synolou i;
    boolean prime; // Boolean παίρνει τιμές TRUE,FALSE
    for (i = 2; i <= n / 2; i++) {
        prime=TRUE; // Υποθέτω ότι είναι πρώτος
        if (n % i == 0) {  // Εάν η πράξη mod του αριθμού με το i που αυξάνει κατά 1,
είναι αληθής, τότε είναι πρώτος.
            prime=FALSE;
            break;
        }
    }
    return prime; // Επέστρεψε το True ή False
}

// Δημιουργεί το σύνολο με τους πρώτους αριθμούς

void createPrimeSet(stoixeio_synolou first, stoixeio_synolou last,typos_synolou S)
{
    Dimiourgia(S); // Δημιουργία συνόλου
    stoixeio_synolou i;
    for (i=first;i<=last;i++)
    {
        if (isPrime(i)) // Κάνει τον έλεγχο αν ο αριθμός είναι πρώτος ή όχι
        {
            Eisagogi(i, S); // Τον τοποθετεί μέσα στο σύνολο
        }
    }

}

void displaySet(stoixeio_synolou first,stoixeio_synolou last,typos_synolou S)
{
        stoixeio_synolou i;
```

```
        for (i=first;i <= last;i++)
        {
                if(Melos(i,S))
                        printf("%d ",i);
        }
        printf("\n");
}

void Dimiourgia(typos_synolou synolo)
{
    stoixeio_synolou i;

    for (i = 0; i < megisto_plithos; i++)
        synolo[i] = FALSE;
}



boolean Melos(stoixeio_synolou stoixeio, typos_synolou synolo)
{
    return synolo[stoixeio];
}

void Eisagogi(stoixeio_synolou stoixeio, typos_synolou synolo)
{
    synolo[stoixeio] = TRUE;
}
```

```
/*  Αρχείο: a5f1.c
    Φοιτητής: Ευστάθιος Ιωσηφίδης


    Στα μαθηματικά, η ακολουθία Fibonacci ορίζεται ως το σύνολο των αριθμών που προκύπτουν από το
    άθροισμα των δύο προηγούμενων αριθμών του συνόλου. Εξ ορισμού, οι 2 πρώτοι αριθμοί του συνόλου είναι
    οι 0,1. Να γραφεί πρόγραμμα το οποίο θα υλοποιεί τις παρακάτω συναρτήσεις:
    a. Συνάρτηση isFibonacci, η οποία δέχεται έναν θετικό ακέραιο n και μία ακολουθία Fibonacci
    (typos_synolou) και επιστρέφει την τιμή TRUE ή FALSE ανάλογα εάν ο αριθμός ανήκει ή όχι
    αντίστοιχα στην ακολουθία Fibonacci
    b. Διαδικασία createFibonacciSet η οποία θα δέχεται έναν θετικό ακέραιο threshold και θα δημιουργεί
    και επιστρέφει το σύνολο Fibonacci, μέχρι και τον αριθμό που είναι μικρότερος ή ίσος από τον
    δοσμένο ακέραιο.
    Στη συνέχεια, γράψτε κυρίως πρόγραμμα το οποίο θα ζητάει από τον χρήστη έναν ακέραιο αριθμό max, ο
    οποίος ανήκει στο διάστημα [2...1000] και θα δημιουργεί και θα εμφανίζει την ακολουθία Fibonacci, όπου
    το μεγαλύτερο στοιχείο της θα είναι μικρότερο ή ίσο του max. Χρησιμοποιείστε την υλοποίηση ΑΔΤ
    σύνολο με πίνακα και τη διαδικασία displaySet από το TestSetADT.c για την εμφάνιση του συνόλου.
    Τέλος, μετά την εμφάνιση του συνόλου, ο χρήστης θα μπορεί να εισάγει αριθμούς επανηλητικά, τους
    οποίους το πρόγραμμα θα ελέγχει για το εάν ανήκουν στην τρέχουσα ακολουθία Fibonacci και θα εκτυπώνει
    αντίστοιχο μήνυμα Το πρόγραμμα θα τερματίζει όταν λάβει αρνητικό αριθμό.

*/

#include <stdio.h>
#include <stdlib.h>

#define megisto_plithos 1001

typedef enum {
    FALSE, TRUE
} boolean;

typedef boolean typos_synolou[megisto_plithos];
typedef int stoixeio_synolou;

boolean isFibonacci(stoixeio_synolou n, typos_synolou fibonacci);
void createFibonacciSet(stoixeio_synolou threshold, typos_synolou fibonacci);

void Dimiourgia(typos_synolou synolo);
void displayset(typos_synolou set);
void Eisagogi(stoixeio_synolou stoixeio, typos_synolou synolo);
boolean Melos(stoixeio_synolou stoixeio, typos_synolou synolo);


int main()
{
    stoixeio_synolou max, tmp;
    typos_synolou fibonacci;

/* Ερώτηση τουλάχιστον μια φορά με την do, μέχρι ποιο όρο fibonacci θέλουμε */
     do{
     printf("Dwse to megisto arithmo. ");
     scanf("%d", &max);
    }while(max < 2 || max > 1000);

/* ΕΝΑΛΛΑΚΤΙΚΟ ΣΕΝΑΡΙΟ
```

```
        while(TRUE){
        printf("Dwse to megisto arithmo. ");
        scanf("%d", &max);
        if(max>=2 && max<=1000) {break;}
        printf("O megistos arithmos prepei na anhkei sto diasthma [2...1000]\n");}

*/

    createFibonacciSet(max, fibonacci); // Δημιουργεί το σύνολο με όρους fibonacci μέχρι
τον όρο max που έχει δοθεί
    displayset(fibonacci); // Εμφανίζει όλους τους όρους του συνόλου

/* Έλεγχος εάν ο αριθμός που εισάγουμε είναι όρος fibonacci ή όχι */
    while(TRUE){
        printf("Enter number to check: ");
        scanf("%d", &tmp);
        if(tmp<0) {break;}
        if(Melos(tmp, fibonacci)) // Ελέγχει με την συνάρτηση Melos εάν ανήκει στο σύνολο
που έχει δημιουργηθεί
            {printf("Fibonacci!\n");}
        else
            {printf("Not Fibonacci...\n");}
    }

    return 0;
}

/* ΣΥΝΑΡΤΗΣΕΙΣ */

boolean isFibonacci(stoixeio_synolou f, typos_synolou fibonacci)
{
        if(Melos(f,fibonacci)) // Ελέγχει εάν ανήκει στο σύνολο των όρων fibonacci που
έχουμε δημιουργήσει με την createFibonacciSet
                {return TRUE;}
    return FALSE;
}

void createFibonacciSet(stoixeio_synolou threshold, typos_synolou fibonacci)
{
    stoixeio_synolou next=1,n1=0,n2=1;
    Dimiourgia(fibonacci); // Δημιουργία συνόλου
    Eisagogi(0,fibonacci); // Εισάγει τον πρώτο όρο
    Eisagogi(1,fibonacci); // Εισάγει τον δεύτερο όρο
    while(next<threshold)
    {
        Eisagogi(next,fibonacci); // Εισάγει τον επόμενο όρο
        n1=n2; // Αλλάζει τον πρώτο όρο στον δεύτερο
        n2=next; // Αλλάζει τον δεύτερο όρο ως το άρθοισμα των δυο προηγούμενων όρων
        next=n1+n2; // Προσθέτει τους δυο όρους που είναι προς εισαγωγή
    }

}

void Dimiourgia(typos_synolou synolo)
{
    stoixeio_synolou i;

    for (i = 0; i < megisto_plithos; i++)
        synolo[i] = FALSE;
}

void displayset(typos_synolou set)
{
        stoixeio_synolou i;

        for (i=0;i < megisto_plithos;i++)
        {
                if(Melos(i,set))
                        printf(" %d",i);
```

```
        }
        printf("\n");
}

void Eisagogi(stoixeio_synolou stoixeio, typos_synolou synolo)
{
    synolo[stoixeio] = TRUE;
}

boolean Melos(stoixeio_synolou stoixeio, typos_synolou synolo)
{
    return synolo[stoixeio];
}
```

```
/*  Αρχείο: a5f2.c
    Φοιτητής: Ευστάθιος Ιωσηφίδης


    Άδεια χρήσης: GNU General Public License v3.0

    Χρησιμοποιήστε τις λειτουργίες που έχουν οριστεί στον ΑΤΔ στοίβα υλοποίηση με πίνακα
και γράψτε στο κυρίως
    πρόγραμμα κώδικα για κάθε μία από τις παρακάτω λειτουργίες:
    (a) θέστε στη μεταβλητή x την τιμή του δεύτερου στοιχείου από την κορυφή της στοίβας,
αφήνοντας τη
    στοίβα χωρίς τα δύο πρώτα στοιχεία της κορυφής.
    (b) θέστε στη μεταβλητή x την τιμή του δεύτερου στοιχείου από την κορυφή της στοίβας,
αφήνοντας τη
    στοίβα αμετάβλητη (δεν θα διαγραφεί κανένα στοιχείο).
    (c) θέστε στη μεταβλητή x την τιμή του n-οστού στοιχείου από την κορυφή της στοίβας,
αφήνοντας τη στοίβα
    χωρίς τα n πρώτα στοιχεία της κορυφής.
    (d) θέστε στη μεταβλητή x την τιμή του n-οστού στοιχείου από την κορυφή της στοίβας,
αφήνοντας τη στοίβα
    αμετάβλητη. (Υπόδειξη: Χρησιμοποιείστε μία άλλη, βοηθητική στοίβα.)
    (e) θέστε στη μεταβλητή x την τιμή του τελευταίου στοιχείου της στοίβας, αφήνοντας τη
στοίβα αμετάβλητη.
    (f) θέστε στη μεταβλητή x την τιμή του τρίτου στοιχείου από τη βάση της στοίβας,
αφήνοντας τη στοίβα
    αμετάβλητη.
    (g) θέστε στη μεταβλητή x την τιμή του τελευταίου στοιχείου της στοίβας, αφήνοντας τη
στοίβα κενή.
    Στο κυρίως πρόγραμμα θα δημιουργείται πρώτα η στοίβα και θα προστίθενται σ’ αυτή 15
αριθμοί. Για λόγους
    απλότητας μπορεί να χρησιμοποιηθεί ένας βρόχος for, σε κάθε επανάληψη του οποίου θα
προστίθεται στη στοίβα
    το δεκαπλάσιο της τιμής της μεταβλητής ελέγχου της for. Στη συνέχεια εμφανίστε το
περιεχόμενο της στοίβας
    (καλέστε τη βοηθητική συνάρτηση TraverseStack, η εμφάνιση των στοιχείων από τη θέση 0
.. Stack.top).
    Πριν την εκτέλεση των παραπάνω λειτουργιών θα διαβάζεται ο ακέραιος αριθμός n (n ≤
(Stack.Top-1)/2) που
    χρειάζεται στις λειτουργίες (c) και (d). Δε χρειάζεται να γίνεται έλεγχος ορθής
καταχώρησης της τιμής του n εντός
    του παραπάνω ορίου θεωρούμε ότι θα δοθεί ορθά.
    Μετά από την εκτέλεση κάθε λειτουργίας (a) έως (g) θα εμφανίζεται η τιμή της
μεταβλητής x στη συνέχεια το
    πλήθος των στοιχείων της στοίβας και το περιεχόμενο της στοίβας. Η είσοδος και η
έξοδος του προγράμματος
    φαίνεται στο παρακάτω ενδεικτικό στιγμιότυπο.
*/

#include <stdio.h>
#include <stdlib.h>

#define StackLimit 15   // το όριο μεγέθους της στοίβας


typedef int StackElementType;   // ο τύπος των στοιχείων  της στοίβας
                                //ενδεικτικά τύπος int
typedef struct  {
    int Top;
    StackElementType Element[StackLimit];
} StackType;

typedef enum {
    FALSE, TRUE
} boolean;

void TraverseStack(StackType Stack);
void CreateStack(StackType *Stack);
boolean EmptyStack(StackType Stack);
```

```
boolean FullStack(StackType Stack);
void Push(StackType *Stack, StackElementType Item);
void Pop(StackType *Stack, StackElementType *Item);

int main()
{
    StackElementType temp,i,n,x;
    StackType MyStack,TempStack;

    CreateStack(&MyStack);
    for(i=0;i<StackLimit;i++)
        {Push(&MyStack, 10*i);}

    TraverseStack(MyStack);
    printf("Give nth element (n<=6) ");
    scanf("%d",&n);
    printf("\n");

    // Question a

    Pop(&MyStack,&x);
    Pop(&MyStack,&x);
    printf("Question a: x=%d",x);
    TraverseStack(MyStack);
    printf("\n");

   // Question b

    Pop(&MyStack, &temp);
    Pop(&MyStack, &x);
    Push(&MyStack, x);
    Push(&MyStack, temp);

    printf("Question b: x=%d",x);
    TraverseStack(MyStack);
    printf("\n");

   // Question c

    for(i=0;i<n;i++){
        Pop(&MyStack,&x);}
    printf("Question c: x=%d",x);
    TraverseStack(MyStack);
    printf("\n");

   // Question d
    CreateStack(&TempStack);
    for(i=0;i<n;i++)
    {
        Pop(&MyStack,&temp);
        Push(&TempStack,temp);
        x=temp;
    }
    for(i=0;i<n;i++)
    {
        Pop(&TempStack,&temp);
        Push(&MyStack,temp);
    }
    printf("Question d: x=%d",x);
    TraverseStack(MyStack);
    printf("\n");

   // Question e

    while(!EmptyStack(MyStack))
    {
        Pop(&MyStack,&x);
        Push(&TempStack,x);
    }
```

```
    while(!EmptyStack(TempStack))
    {
        Pop(&TempStack,&temp);
        Push(&MyStack,temp);
    }

    printf("Question e: x=%d",x);
    TraverseStack(MyStack);
    printf("\n");

    // Question f

    for(i=MyStack.Top;i>=2;i--)
    {
        Pop(&MyStack,&x);
        Push(&TempStack,x);
    }
    while(!EmptyStack(TempStack))
    {
        Pop(&TempStack,&temp);
        Push(&MyStack,temp);
    }

    printf("Question f: x=%d",x);
    TraverseStack(MyStack);
    printf("\n");

    // Question g

    while(!EmptyStack(MyStack))
    {
        Pop(&MyStack,&x);
    }

    printf("Question g: x=%d",x);
    TraverseStack(MyStack);

    return 0;
}



void CreateStack(StackType *Stack)
/* Λειτουργία: Δημιουργεί μια κενή στοίβα.
   Επιστρέφει: Κενή Στοίβα.*
*/
{
    Stack -> Top = -1;
   // (*Stack).Top = -1;
}

boolean EmptyStack(StackType Stack)
/* Δέχεται: Μια στοίβα Stack.
   Λειτουργία: Ελέγχει αν η στοίβα Stack είναι κενή.
   Επιστρέφει: True αν η Stack είναι κενή, False διαφορετικά
*/
{
    return (Stack.Top == -1);
}

boolean FullStack(StackType Stack)
/* Δέχεται: Μια στοίβα Stack.
   Λειτουργία: Ελέγχει αν η στοίβα Stack είναι γεμάτη.
   Επιστρέφει: True αν η Stack είναι γεμάτη, False διαφορετικά
*/
{
    return (Stack.Top == (StackLimit - 1));
}
```

```
void Push(StackType *Stack, StackElementType Item)
/* Δέχεται: Μια στοίβα Stack και ένα στοιχείο Item.
   Λειτουργία: Εισάγει το στοιχείο Item στην στοίβα Stack αν η Stack δεν είναι γεμάτη.
   Επιστρέφει: Την τροποποιημένη Stack.
   Έξοδος: Μήνυμα γεμάτης στοίβας, αν η στοίβα Stack είναι γεμάτη
*/
{
    if (!FullStack(*Stack)) {
        Stack -> Top++;
        Stack -> Element[Stack -> Top] = Item;
    } else
        printf("Full Stack...");
}

void Pop(StackType *Stack, StackElementType *Item)
/* Δέχεται: Μια στοίβα Stack.
   Λειτουργία: Διαγράφει το στοιχείο Item από την κορυφή της Στοίβας αν η Στοίβα δεν
είναι κενή.
   Επιστρέφει: Το στοιχείο Item και την τροποποιημένη Stack.
   Έξοδος:  Μήνυμα κενής στοίβας αν η Stack είναι κενή
*/
{
    if (!EmptyStack(*Stack)) {
        *Item = Stack -> Element[Stack -> Top];
        Stack -> Top--;
    } else
        printf("Empty Stack...");
}

/*
// ΑΥΤΗ Η ΕΚΔΟΣΗ ΕΜΦΑΝΙΖΕΙ ΤΑ ΣΤΟΙΧΕΙΑ ΤΗΣ ΣΤΟΙΒΑΣ ΑΠΟ ΤΗ ΘΕΣΗ TOP-0

void TraverseStack(StackType Stack)
{
    int i;
    printf("\nplithos sto stack %d\n",Stack.Top+1);
    for (i=Stack.Top;i>=0;i--) {
        printf("%d ",Stack.Element[i]);
    }
    printf("\n");
}
*/

 // ΑΥΤΗ Η ΕΚΔΟΣΗ ΕΙΝΑΙ ΣΤΙΣ ΣΗΜΕΙΩΣΕΙΣ ΚΑΙ ΕΜΦΑΝΙΖΕΙ ΤΑ ΣΤΟΙΧΕΙΑ ΤΗΣ ΣΤΟΙΒΑΣ ΑΠΟ ΤΗ ΘΕΣΗ
0-TOP
 void TraverseStack(StackType Stack)
{
    int i;
    printf("\nplithos sto stack %d\n",Stack.Top+1);
    for (i=0;i<=Stack.Top;i++) {
        printf("%d ",Stack.Element[i]);
    }
    printf("\n");
}
```

```
/*  Αρχείο: a8f2.c
    Φοιτητής: Ευστάθιος Ιωσηφίδης


    Άδεια χρήσης: GNU General Public License v3.0

    Γράψτε ένα πρόγραμμα που θα δέχεται ένα αλφαριθμητικό, διαβάζοντας το χαρακτήρα προς χαρακτήρα (μέχρι ο
    χρήστης να δώσει ως χαρακτήρα #) και θα ελέγχει αν το αλφαριθμητικό που έχει σχηματιστεί, έχει τη μορφή
                                                        x C y
    όπου x είναι ένα αλφαριθμητικό και y είναι το αντίστροφο του x. Για παράδειγμα, αν x = 'ABABBA' τότε
    y = 'ABBABA'. Ο έλεγχος θα γίνεται κατά το διάβασμα του κάθε χαρακτήρα. Το πρόγραμμα θα σταματάει να διαβάζει
    χαρακτήρες είτε όταν δοθεί ο χαρακτήρας '#', είτε μόλις διαπιστώσει ότι το αλφαριθμητικό δεν ακολουθεί την
    επιθυμητή μορφή. Πριν τερματίσει θα εμφανίζει το μήνυμα TRUE ή FALSE αντίστοιχα αν το αλφαριθμητικό έχει ή
    όχι αυτή τη μορφή.
    Υπόδειξη: οι χαρακτήρες που θα διαβαστούν μέχρι να δοθεί ο χαρακτήρας 'C' εισάγονται σε μια στοίβα και κάθε
    χαρακτήρας που διαβάζεται μετά τον 'C' συγκρίνεται με το στοιχείο που βρίσκεται στην κορυφή της στοίβας. Για το
    διάβασμα κάθε χαρακτήρα χρησιμοποιείστε : scanf("%c", &ch); getchar();

*/
#include <stdio.h>
#include <stdlib.h>

#define StackLimit 50  // το όριο μεγέθους της στοίβας

typedef char StackElementType; // ο τύπος των στοιχείων  της στοίβας είναι τύπος char

typedef struct  {
    int Top;
    StackElementType Element[StackLimit];
} StackType;

typedef enum {
    FALSE, TRUE
} boolean;

// Δήλωση συναρτήσεων
void CreateStack(StackType *Stack);
void Push(StackType *Stack, StackElementType Item);
void Pop(StackType *Stack, StackElementType *Item);
boolean FullStack(StackType Stack);
boolean EmptyStack(StackType Stack);

int main()
{
    StackElementType item, ch; // Δήλωση μεταβλητών
    StackType MyStack; // Δήλωση στοίβας
    boolean flag, valid; // Δήλωση σημαιών

    flag=FALSE; // Αρχικοποίηση σημαίας
    valid=TRUE; // Αρχικοποίηση σημαίας
    CreateStack(&MyStack); // Δημιουργία στοίβας

    // Εισαγωγή γραμμάτων
    printf("Enter string:\n");
    scanf("%c", &ch); getchar();

    if(ch=='#') return 0; // Αν εισάγει τον χαρακτήρα # τότε τερματίζει το πρόγραμμα
    while(ch!='#' && valid) // Εάν δώσει έγκυρο χαρακτήρα να ξεκινήσει την εισαγωγή και τους ελέγχους
    {
```

```
        if(!flag){
            if(ch=='C') /* Εάν ο χαρακτήρας που εισήχθη είναι ο C να σταματήσει την εισαγωγή στην στοίβα και να ξεκινήσει τους ελέγχους */
            {
                flag=TRUE; // Αλλαγή της σημαίας flag σε TRUE για να δηλώσει ότι τελείωσε η εισαγωγή στην στοίβα
                scanf("%c", &ch); getchar(); // Εισαγωγή χαρακτήρων προς έλεγχο στην στοίβα
            }
            else // Αν ο χαρακτήρας δεν είναι C
            {
                Push(&MyStack, ch); // Ώθηση χαρακτήρα στην στοίβα
                scanf("%c", &ch); getchar(); /* Εισαγωγή επόμενου χαρακτήρα για ώθηση ή το C για τερματισμό ώθησης ή # για τερματισμό προγράμματος */
            }
        }
        else{
            if(!EmptyStack(MyStack)) // Εάν η στοίβα δεν είναι κενή
            {
                Pop(&MyStack, &item); // Απώθηση του χαρακτήρα
                if(item!=ch) {valid=FALSE;} /* Έλεγχος αν ο χαρακτήρας είναι διαφορετικός από αυτόν που εισήχθη, άλλαξε την σημαία valid σε FALSE */
                else {scanf("%c", &ch); getchar();} /* Εισαγωγή χαρακτήρων προς έλεγχο στην στοίβα αφού έχει εισαχθεί ο χαρακτήρας C πιο πάνω */
            }
            else {valid=FALSE;} /* Εαν η στοίβα είναι κενή, αλλαξε την σημαία valid σε FALSE */
        }
    }

    if(valid && EmptyStack(MyStack)) {printf("TRUE\n");} // Εκτύπωση αν η σημαία valid είναι TRUE και η στοίβα είναι άδεια
    else {printf("FALSE\n");} // Εκτύπωση σε κάθε άλλη περίπτωση

    return 0;

}


// ΣΥΝΑΡΤΗΣΕΙΣ

void CreateStack(StackType *Stack)
/* Λειτουργία: Δημιουργεί μια κενή στοίβα.
   Επιστρέφει: Κενή Στοίβα.*
*/
{
    Stack -> Top = -1;
   // (*Stack).Top = -1;
}

boolean EmptyStack(StackType Stack)
/* Δέχεται: Μια στοίβα Stack.
   Λειτουργία: Ελέγχει αν η στοίβα Stack είναι κενή.
   Επιστρέφει: True αν η Stack είναι κενή, False διαφορετικά
*/
{
    return (Stack.Top == -1);
}

boolean FullStack(StackType Stack)
/* Δέχεται: Μια στοίβα Stack.
   Λειτουργία: Ελέγχει αν η στοίβα Stack είναι γεμάτη.
   Επιστρέφει: True αν η Stack είναι γεμάτη, False διαφορετικά
*/
{
    return (Stack.Top == (StackLimit - 1));
}

void Push(StackType *Stack, StackElementType Item)
```

```
/* Δέχεται: Μια στοίβα Stack και ένα στοιχείο Item.
   Λειτουργία: Εισάγει το στοιχείο Item στην στοίβα Stack αν η Stack δεν είναι γεμάτη.
   Επιστρέφει: Την τροποποιημένη Stack.
   Έξοδος: Μήνυμα γεμάτης στοίβας, αν η στοίβα Stack είναι γεμάτη
*/
{
    if (!FullStack(*Stack)) {
        Stack -> Top++;
        Stack -> Element[Stack -> Top] = Item;
    } else
        printf("Full Stack...");
}

void Pop(StackType *Stack, StackElementType *Item)
/* Δέχεται: Μια στοίβα Stack.
   Λειτουργία: Διαγράφει το στοιχείο Item από την κορυφή της Στοίβας αν η Στοίβα δεν
είναι κενή.
   Επιστρέφει: Το στοιχείο Item και την τροποποιημένη Stack.
   Έξοδος:  Μήνυμα κενής στοίβας αν η Stack είναι κενή
*/
{
    if (!EmptyStack(*Stack)) {
        *Item = Stack -> Element[Stack -> Top];
        Stack -> Top--;
    } else
        printf("Empty Stack...");
}
```

```
/*  Αρχείο: a16f2.c
    Φοιτητής: Ευστάθιος Ιωσηφίδης


    Άδεια χρήσης: GNU General Public License v3.0

    Ένα κατάστημα διατηρεί κάθε διαφορετικού τύπου προϊόν του μέσα σε κουτί όπου τα
τοποθετεί οργανωμένα σε
    μορφή στοίβας. Για το προϊόν «παιδικό φανελάκι» αποθηκεύει την τιμή και το μέγεθος.
Να γίνει πρόγραμμα
    όπου:
    1. Θα δίνεται το πλήθος από τα παιδικά φανελάκια που θα καταχωρήσει στο κουτί.
    2. Για κάθε φανελάκι θα δίνει την τιμή και το μέγεθος και θα εισάγει (καταχωρεί) το
φανελάκι στο κουτί (ως
    στοίβα). Δεν απαιτείται τα φανελάκια (στοιχεία) να είναι ταξινομημένα ως προς το
μέγεθος ή την τιμή για
    να εισαχθούν στη στοίβα-κουτί, απλά καταχωρούνται στη στοίβα-κουτί (LIFO).
    3. Όταν εισαχθούν όλα τα φανελάκια στη στοίβα-κουτί θα εμφανίζει το μήνυμα “items in
the box” και στη
    συνέχεια τα φανελάκια που καταχωρήθηκαν στη στοίβα-κουτί.
    4. Θα αναζητά ένα φανελάκι με βάση το μέγεθος. Αν βρεθεί θα εμφανίζει μήνυμα “Found
the size”
    διαφορετικά “Not Found the size”. Θα σταματά την αναζήτηση μόλις βρει το πρώτο
φανελάκι με το
    συγκεκριμένο μέγεθος. Αν βρεθεί το φανελάκι θεωρούμε ότι αμέσως πωλείται.
    5. Στη συνέχεια θα εμφανίζει το μήνυμα “Items in the box” και θα εμφανίζει αυτά που
έμειναν στη στοίβα-
    κουτί. Στη συνέχεια θα εμφανίζει το μήνυμα “Items out of the box” και θα εμφανίζει
αυτά που έχουν
    βγει από τη στοίβα-κουτί.
    6. Αν το φανελάκι δε βρεθεί τότε η στοίβα-κουτί με τα φανελάκια θα πρέπει να
αποκατασταθεί και να
    περιέχει όλα τα φανελάκια με την αρχική διάταξη. Αν το φανελάκι βρεθεί τότε η στοίβα-
κουτί με τα
    φανελάκια θα πρέπει να τα έχει διατεταγμένα με την αρχική διάταξη αλλά το φανελάκι
που αναζητήθηκε
    και βρέθηκε δε θα πρέπει να περιλαμβάνεται στη στοίβα-κουτί με τα φανελάκια, καθώς
θεωρούμε ότι
    μόλις βρεθεί πωλείται.
    7. Στη συνέχεια θα εμφανίζει το μήνυμα “Items in the box” και θα εμφανίζει αυτά που
βρίσκονται στη
    στοίβα-κουτί και το μήνυμα “Items out of the box” και θα εμφανίζει αυτά που έχουν
βγει από τη στοίβα-
    κουτί.

    Στη συνέχεια δίνονται 2 ενδεικτικά στιγμιότυπα εκτέλεσης. Οι αριθμοί στην 1 στήλη
αντιστοιχούν στην
    είσοδο και έξοδο όπως περιγράφονται παραπάνω στα σημεία 1 έως 5 και 7, ενώ το σημείο
6 δεν έχει
    είσοδο ή έξοδο.
*/

#include <stdio.h>
#include <stdlib.h>

#define StackLimit 50   // το όριο μεγέθους της στοίβας

typedef struct  {
    int price;
    char size;
} StackElementType;

typedef struct  {
    int Top;
    StackElementType Element[StackLimit];
} StackType;

typedef enum {
```

```
    FALSE, TRUE
} boolean;

void TraverseStack(StackType Stack);
void CreateStack(StackType *Stack);
boolean EmptyStack(StackType Stack);
boolean FullStack(StackType Stack);
void Push(StackType *Stack, StackElementType Item);
void Pop(StackType *Stack, StackElementType *Item);

int main()
{
    StackType ShirtStack,OutStack;
    StackElementType AnItem;
    int n,i;
    char sell;
    boolean found;

    CreateStack(&ShirtStack);
    CreateStack(&OutStack);


    // Ερώτημα 1
    printf("Give number of items ");
    scanf("%d",&n);

    //Ερώτημα 2
    printf("Give the items to store\n");
    for(i=0;i<n;i++)
    {
        printf("Give price ");
        scanf("%d",&AnItem.price);getchar();
        printf("Give size ");
        scanf("%c",&AnItem.size);getchar();
        Push(&ShirtStack,AnItem);
    }

    // Ερώτημα 3
    printf("Items in the box\n");
    TraverseStack(ShirtStack);

    // Ερώτημα 4
    printf("What size do you want? ");
    scanf("%c",&sell);getchar();

    found=FALSE;
    while(!EmptyStack(ShirtStack) && found==FALSE)
    {
        Pop(&ShirtStack,&AnItem);
        if(AnItem.size==sell){
            found=TRUE;

        }
        else{
            Push(&OutStack,AnItem);
        }
    }
    if(found==TRUE){
        printf("Found the size %c\n",sell);
        printf("\n");}
    else{
        printf("Not Found the size %c\n",sell);
        printf("\n");}

    // Ερώτημα 5

    printf("Items in the box");
    TraverseStack(ShirtStack);
```

```
    printf("Items out the box");
    TraverseStack(OutStack);

    // Ερώτημα 6

    while(!EmptyStack(OutStack))
    {
        Pop(&OutStack,&AnItem);
        Push(&ShirtStack,AnItem);
    }

   // Ερώτημα 7

    printf("Items in the box");
    TraverseStack(ShirtStack);

    printf("Items out the box");
    TraverseStack(OutStack);

    return 0;
}



void CreateStack(StackType *Stack)
/* Λειτουργία: Δημιουργεί μια κενή στοίβα.
   Επιστρέφει: Κενή Στοίβα.*
*/
{
    Stack -> Top = -1;
   // (*Stack).Top = -1;
}

boolean EmptyStack(StackType Stack)
/* Δέχεται: Μια στοίβα Stack.
   Λειτουργία: Ελέγχει αν η στοίβα Stack είναι κενή.
   Επιστρέφει: True αν η Stack είναι κενή, False διαφορετικά
*/
{
    return (Stack.Top == -1);
}

boolean FullStack(StackType Stack)
/* Δέχεται: Μια στοίβα Stack.
   Λειτουργία: Ελέγχει αν η στοίβα Stack είναι γεμάτη.
   Επιστρέφει: True αν η Stack είναι γεμάτη, False διαφορετικά
*/
{
    return (Stack.Top == (StackLimit - 1));
}

void Push(StackType *Stack, StackElementType Item)
/* Δέχεται: Μια στοίβα Stack και ένα στοιχείο Item.
   Λειτουργία: Εισάγει το στοιχείο Item στην στοίβα Stack αν η Stack δεν είναι γεμάτη.
   Επιστρέφει: Την τροποποιημένη Stack.
   Έξοδος: Μήνυμα γεμάτης στοίβας, αν η στοίβα Stack είναι γεμάτη
*/
{
    if (!FullStack(*Stack)) {
        Stack -> Top++;
        Stack -> Element[Stack -> Top] = Item;
    } else
        printf("Full Stack...");
}

void Pop(StackType *Stack, StackElementType *Item)
/* Δέχεται: Μια στοίβα Stack.
   Λειτουργία: Διαγράφει το στοιχείο Item από την κορυφή της Στοίβας αν η Στοίβα δεν
είναι κενή.
```

```
   Επιστρέφει: Το στοιχείο Item και την τροποποιημένη Stack.
   Έξοδος:  Μήνυμα κενής στοίβας αν η Stack είναι κενή
*/
{
    if (!EmptyStack(*Stack)) {
        *Item = Stack -> Element[Stack -> Top];
        Stack -> Top--;
    } else
        printf("Empty Stack...");
}

/*
// ΑΥΤΗ Η ΕΚΔΟΣΗ ΕΜΦΑΝΙΖΕΙ ΤΑ ΣΤΟΙΧΕΙΑ ΤΗΣ ΣΤΟΙΒΑΣ ΑΠΟ ΤΗ ΘΕΣΗ TOP-0

void TraverseStack(StackType Stack)
{
    int i;
    printf("\nplithos sto stack %d\n",Stack.Top+1);
    for (i=Stack.Top;i>=0;i--) {
        printf("%d ",Stack.Element[i]);
    }
    printf("\n");
}
*/


 // ΑΥΤΗ Η ΕΚΔΟΣΗ ΕΙΝΑΙ ΣΤΙΣ ΣΗΜΕΙΩΣΕΙΣ ΚΑΙ ΕΜΦΑΝΙΖΕΙ ΤΑ ΣΤΟΙΧΕΙΑ ΤΗΣ ΣΤΟΙΒΑΣ ΑΠΟ ΤΗ ΘΕΣΗ
0-TOP
 void TraverseStack(StackType Stack)
{
    int i;
    printf("\nplithos sto stack %d\n",Stack.Top+1);
    for (i=0;i<=Stack.Top;i++) {
        printf("%c, %d\n",Stack.Element[i].size,Stack.Element[i].price);
    }
    printf("\n");
}
```

```
/*  Αρχείο: a17f2.c
    Φοιτητής: Ευστάθιος Ιωσηφίδης


    Άδεια χρήσης: GNU General Public License v3.0

    Να υλοποιηθεί η συνάρτηση CopyStack η οποία θα δημιουργεί και θα επιστρέφει ένα
αντίγραφο της δοσμένης
    στοίβας s1 αφήνοντας την αρχική στοίβα s1 αναλλοίωτη. Το πρωτότυπο της συνάρτησης
είναι
                    StackType CopyStack(StackType *s1)
    Στο κυρίως πρόγραμμα δημιουργήστε τη στοίβα s1 εισάγοντας σ' αυτή 20 αριθμούς. Για
λόγους απλότητας
    μπορεί να χρησιμοποιηθεί ένας βρόχος for, σε κάθε επανάληψη του οποίου θα εισάγετε
στη στοίβα την τιμής της
    μεταβλητής ελέγχου της for. Στη συνέχεια εμφανίστε το περιεχόμενο της στοίβας s1
(καλέστε τη βοηθητική
    συνάρτηση TraverseStack, η εμφάνιση των στοιχείων από τη θέση 0 .. Stack.top).
Καλέστε την CopyStack και στη
    συνέχεια εμφανίστε τα στοιχεία της στοίβας s1 και τα στοιχεία της στοίβας s2. Δίνεται
ένα στιγμιότυπο εκτέλεσης.

*/

#include <stdio.h>
#include <stdlib.h>

#define StackLimit 20   // το όριο μεγέθους της στοίβας

typedef int StackElementType;   // ο τύπος των στοιχείων  της στοίβας ενδεικτικά τύπος
int

typedef struct  {
    int Top;
    StackElementType Element[StackLimit];
} StackType;

typedef enum {
    FALSE, TRUE
} boolean;

// Δήλωση συναρτήσεων
void TraverseStack(StackType Stack);
void CreateStack(StackType *Stack);
boolean EmptyStack(StackType Stack);
boolean FullStack(StackType Stack);
void Push(StackType *Stack, StackElementType Item);
void Pop(StackType *Stack, StackElementType *Item);
StackType CopyStack(StackType *s1);

int main()
{
    StackElementType i; // Δήλωση μεταβλητών
    StackType s1,s2; // Δήλωση στοιβών

    CreateStack(&s1); // Δημιουργία στοίβας s1

    // Γέμισμα στοίβας
    for (i=0;i<StackLimit;i++)
    {
        Push(&s1, i);
    }

    // Εκτύπωση στοίβας s1

    printf("Stack s1");
    TraverseStack(s1);
```

```
    // Αντιγραφή στοίβας s1 στην s2
    s2=CopyStack(&s1);

    // Εκτύπωση στοιβών μετά την αντιγραφή
    printf("After copying s1 to s2\n");

    // Εκτύπωση στοίβας s1
    printf("Stack s1");
    TraverseStack(s1);

    // Εκτύπωση στοίβας s2
    printf("Stack s2");
    TraverseStack(s2);


    return 0;
}

// ΣΥΝΑΡΤΗΣΕΙΣ

StackType CopyStack(StackType *s1)
{
    StackElementType temp; // Δήλωση μεταβλητών
    StackType TempStack,s2; // Δήλωση στοίβών

    CreateStack(&s2); // Δημιουργία στοίβας s2
    CreateStack(&TempStack); // Δημιουργία στοίβας TempStack

    while(!EmptyStack(*s1)) // Όσο η στοίβα s1 δεν είναι κενή, κάνε τα παρακάτω
    {
        Pop(&(*s1),&temp); // Απώθηση στοιχείου και αποθήκευση στην μεταβλητή temp
        Push(&TempStack,temp); // Ώθηση στοιχείου στην στοίβα TempStack
    }

    while(!EmptyStack(TempStack)) // Όσο η στοίβα TempStack δεν είναι κενή, κάνε τα
παρακάτω
    {
        Pop(&TempStack,&temp); // Απώθηση στοιχείου και αποθήκευση στην μεταβήτή temp
        Push(&(*s1),temp); // Ώθηση στοιχείου temp στην στοίβα s1 (pointer της στοίβας
s1)
        Push(&s2,temp); // Ώθηση στοιχείου temp στην στοίβα s2
    }
    return s2;
}

// ΣΥΝΑΡΤΗΣΕΙΣ

void CreateStack(StackType *Stack)
/* Λειτουργία: Δημιουργεί μια κενή στοίβα.
   Επιστρέφει: Κενή Στοίβα.*
*/
{
    Stack -> Top = -1;
   // (*Stack).Top = -1;
}

boolean EmptyStack(StackType Stack)
/* Δέχεται: Μια στοίβα Stack.
   Λειτουργία: Ελέγχει αν η στοίβα Stack είναι κενή.
   Επιστρέφει: True αν η Stack είναι κενή, False διαφορετικά
*/
{
    return (Stack.Top == -1);
}

boolean FullStack(StackType Stack)
/* Δέχεται: Μια στοίβα Stack.
   Λειτουργία: Ελέγχει αν η στοίβα Stack είναι γεμάτη.
   Επιστρέφει: True αν η Stack είναι γεμάτη, False διαφορετικά
```

```
*/
{
    return (Stack.Top == (StackLimit - 1));
}

void Push(StackType *Stack, StackElementType Item)
/* Δέχεται: Μια στοίβα Stack και ένα στοιχείο Item.
   Λειτουργία: Εισάγει το στοιχείο Item στην στοίβα Stack αν η Stack δεν είναι γεμάτη.
   Επιστρέφει: Την τροποποιημένη Stack.
   Έξοδος: Μήνυμα γεμάτης στοίβας, αν η στοίβα Stack είναι γεμάτη
*/
{
    if (!FullStack(*Stack)) {
        Stack -> Top++;
        Stack -> Element[Stack -> Top] = Item;
    } else
        printf("Full Stack...");
}

void Pop(StackType *Stack, StackElementType *Item)
/* Δέχεται: Μια στοίβα Stack.
   Λειτουργία: Διαγράφει το στοιχείο Item από την κορυφή της Στοίβας αν η Στοίβα δεν
είναι κενή.
   Επιστρέφει: Το στοιχείο Item και την τροποποιημένη Stack.
   Έξοδος:  Μήνυμα κενής στοίβας αν η Stack είναι κενή
*/
{
    if (!EmptyStack(*Stack)) {
        *Item = Stack -> Element[Stack -> Top];
        Stack -> Top--;
    } else
        printf("Empty Stack...");
}

/*
// ΑΥΤΗ Η ΕΚΔΟΣΗ ΕΜΦΑΝΙΖΕΙ ΤΑ ΣΤΟΙΧΕΙΑ ΤΗΣ ΣΤΟΙΒΑΣ ΑΠΟ ΤΗ ΘΕΣΗ TOP-0

void TraverseStack(StackType Stack)
{
    int i;
    printf("\nplithos sto stack %d\n",Stack.Top+1);
    for (i=Stack.Top;i>=0;i--) {
        printf("%d ",Stack.Element[i]);
    }
    printf("\n");
}
*/

 // ΑΥΤΗ Η ΕΚΔΟΣΗ ΕΙΝΑΙ ΣΤΙΣ ΣΗΜΕΙΩΣΕΙΣ ΚΑΙ ΕΜΦΑΝΙΖΕΙ ΤΑ ΣΤΟΙΧΕΙΑ ΤΗΣ ΣΤΟΙΒΑΣ ΑΠΟ ΤΗ ΘΕΣΗ
0-TOP
 void TraverseStack(StackType Stack)
{
    int i;
    printf("\nplithos sto stack %d\n",Stack.Top+1);
    for (i=0;i<=Stack.Top;i++) {
        printf("%d ",Stack.Element[i]);
    }
    printf("\n");
}
```

```
/*  Αρχείο: a6f3.c
    Φοιτητής: Ευστάθιος Ιωσηφίδης


    Άδεια χρήσης: GNU General Public License v3.0

    Μια εναλλακτική υλοποίηση μιας ουράς που χρησιμοποιεί ένα κυκλικό πίνακα και δεν
απαιτεί να διατηρούμε μία κενή
    θέση μεταξύ της εμπρός και της πίσω άκρη της για να ξεχωρίζει μια κενή ουρά από μια
γεμάτη χρειάζεται απλά την
    προσθήκη ενός ακέραιου πεδίου Count στην εγγραφή τύπου QueueType, στο οποίο
αποθηκεύεται ο τρέχων αριθμός
    στοιχείων της ουράς. Να κάνετε τις απαραίτητες αλλαγές στη δήλωση του τύπου της
εγγραφής και στις βασικές
    λειτουργίες του ΑΤΔ ουρά με πίνακες, έτσι ώστε να χρησιμοποιείται αυτό το επιπλέον
πεδίο και να μην διατηρείται
    κενή θέση στον πίνακα όπου αποθηκεύονται τα στοιχεία της ουράς. Η TraverseQ που σας
δίνεται στο TestQueue
    θα πρέπει επίσης να τροποποιηθεί. Για να ελέγξετε την ορθότητα του προγράμματος σας

    (a) Δημιουργήστε μια ουρά (QueueLimit=10) που περιλαμβάνει όλους τους ακέραιους
αριθμούς στο διάστημα [0..9].
    Εμφανίστε την ουρά (με βοηθητική διαδικασία TraverseQ) την τιμή της Front, Rear και
του μετρητή
    των στοιχείων της ουράς.
    (b) Στη συνέχεια επιχειρήστε να προσθέσετε ένα οποιοδήποτε στοιχείο.
    Εμφανίστε την ουρά (με βοηθητική διαδικασία TraverseQ)
    (c) Αφαιρέστε τη κεφαλή της ουράς και εμφανίστε την ουρά (με βοηθητική διαδικασία
TraverseQ),
    το στοιχείο που αφαιρέσατε, την τιμή της Front, Rear και του μετρητή των στοιχείων
της ουράς
    (d) Προσθέσετε ένα οποιοδήποτε στοιχείο και εμφανίστε την ουρά (με βοηθητική
διαδικασία TraverseQ),
    την τιμή της Front, Rear και του μετρητή των στοιχείων της ουράς
    (e) Στη συνέχεια επιχειρήστε να προσθέσετε ένα οποιοδήποτε στοιχείο. Εμφανίστε την
ουρά
    (με βοηθητική διαδικασία TraverseQ), την τιμή της Front, Rear και του μετρητή των
στοιχείων της ουράς.
    (f) Αδειάστε την ουρά. Μετά την αφαίρεση κάθε φορά της κεφαλής της ουράς εμφανίστε
την ουρά (με βοηθητική
    διαδικασία TraverseQ), το στοιχείο που αφαιρέσατε, την τιμή της Front, Rear και του
μετρητή των στοιχείων
    της ουράς.

*/
#include <stdio.h>

#define QueueLimit 10  //το όριο μεγέθους της ουράς

typedef int QueueElementType;   /* ο τύπος δεδομένων των στοιχείων της ουράς
                                   ενδεικτικά τύπος int */
typedef struct {
        int Front, Rear;
   int Count;
        QueueElementType Element[QueueLimit];;
} QueueType;

typedef enum {FALSE, TRUE} boolean;

void CreateQ(QueueType *Queue);
boolean EmptyQ(QueueType Queue);
boolean FullQ(QueueType Queue);
void RemoveQ(QueueType *Queue, QueueElementType *Item);
void AddQ(QueueType *Queue, QueueElementType Item);
void TraverseQ(QueueType Queue);

int main()
{
```

```
   QueueType MyQueue;
   QueueElementType i, temp;

    //Ερώτημα a
    printf("---a---\n");
    CreateQ(&MyQueue); // Δημιουργία ουράς
    for(i=0;i<QueueLimit;i++)
        AddQ(&MyQueue, i); // Προσθήκη αριθμών στην ουρά
    TraverseQ(MyQueue); // Εκτύπωση ουράς
    printf("Front=%d Rear=%d Count=%d\n", MyQueue.Front, MyQueue.Rear, MyQueue.Count);

    //Ερώτημα b
    printf("---b---\n");
    AddQ(&MyQueue, 13); // Προσθήκη στην ουρά ενός αριθμού
    printf("\n");
    TraverseQ(MyQueue); // Εκτύπωση ουράς (εδώ είναι γεμάτη)
    printf("Front=%d Rear=%d Count=%d\n", MyQueue.Front, MyQueue.Rear, MyQueue.Count);

    //Ερώτημα c
    printf("---c---\n");
    RemoveQ(&MyQueue, &temp); // Αφαίρεση ενός αριθμού
    TraverseQ(MyQueue); // Εκτύπωση ουράς (ποιος αριθμός βγήκε)
    printf("Removed item=%d Front=%d Rear=%d Count=%d\n", temp, MyQueue.Front,
MyQueue.Rear, MyQueue.Count);

    //Ερώτημα d
    printf("---d---\n");
    AddQ(&MyQueue, 13); // Προσθήκη αριθμού στην ουρά
    TraverseQ(MyQueue); // Εκτύπωση ουράς
    printf("Front=%d Rear=%d Count=%d\n", MyQueue.Front, MyQueue.Rear, MyQueue.Count);

    //Ερώτημα e
    printf("---e---\n");
    AddQ(&MyQueue, 14); // Προσθήκη επόμενου αριθμού στην ουρά
    printf("\n");
    TraverseQ(MyQueue); // Εκτύπωση ουράς (εδώ είναι γεμάτη)
    printf("Front=%d Rear=%d Count=%d\n", MyQueue.Front, MyQueue.Rear, MyQueue.Count);

    //Ερώτημα f
    printf("---f---\n");
    while(!EmptyQ(MyQueue)) // Όσο η ουρά δεν είναι άδεια
    {
        RemoveQ(&MyQueue, &temp); // Βγάζε ένα αριθμό
        TraverseQ(MyQueue); // Εκτύπωνε τον αριθμό
        printf("Removed item=%d Front=%d Rear=%d Count=%d\n", temp, MyQueue.Front,
MyQueue.Rear, MyQueue.Count);
    }

    return 0;

}

// ΣΥΝΑΡΤΗΣΕΙΣ

void CreateQ(QueueType *Queue)
/*  Λειτουργία:  Δημιουργεί μια κενή ουρά.
    Επιστρέφει:  Κενή ουρά
*/
{
        Queue->Front = 0;
        Queue->Rear = 0;
   Queue->Count = 0; // Προσθήκη ακέραιου πεδίου Count
}

boolean EmptyQ(QueueType Queue)
/* Δέχεται:    Μια ουρά.
   Λειτουργία: Ελέγχει αν η ουρά είναι κενή.
   Επιστρέφει: True αν η ουρά είναι κενή, False διαφορετικά
*/
```

```
{
        return (Queue.Count == 0); // Έλεγχος αν είναι άδεια η ουρά με το πεδίο Count
}

boolean FullQ(QueueType Queue)
/* Δέχεται:    Μια ουρά.
   Λειτουργία: Ελέγχει αν η ουρά είναι γεμάτη.
   Επιστρέφει: True αν η ουρά είναι γεμάτη, False διαφορετικά
*/
{
        return (Queue.Count == QueueLimit); // Έλεγχος αν είναι γεμάτη η ουρά με το πεδίο
Count
}

void RemoveQ(QueueType *Queue, QueueElementType *Item)
/* Δέχεται:    Μια ουρά.
   Λειτουργία: Αφαιρεί το στοιχείο Item από την εμπρός άκρη της ουράς
                αν η ουρά δεν είναι κενή.
   Επιστρέφει: Το στοιχείο Item και την τροποποιημένη ουρά.
   Έξοδος:     Μήνυμα κενής ουρά αν η ουρά είναι κενή
*/
{
        if(!EmptyQ(*Queue))
        {
                *Item = Queue ->Element[Queue -> Front];
                Queue ->Front  = (Queue ->Front + 1) % QueueLimit;
      (Queue->Count)--; // Στην αφαίρεση, τότε μείωσε και το Count
   }
        else
                printf("Empty Queue");
}

void AddQ(QueueType *Queue, QueueElementType Item)
/* Δέχεται:    Μια ουρά Queue και ένα στοιχείο Item.
   Λειτουργία: Προσθέτει το στοιχείο Item στην ουρά Queue
                αν η ουρά δεν είναι γεμάτη.
   Επιστρέφει: Την τροποποιημένη ουρά.
   Έξοδος:     Μήνυμα γεμάτης ουράς αν η ουρά είναι γεμάτη
*/
{
        if(!FullQ(*Queue))
        {
                Queue ->Element[Queue ->Rear] = Item;
                Queue ->Rear = (Queue ->Rear + 1) % QueueLimit;
      (Queue->Count)++; // Στην πρόσθεση, αύξησε και το Count
        }
        else
                printf("Full Queue");
}

void TraverseQ(QueueType Queue) {
        int current;
        current = Queue.Front;
        printf("Queue: ");
   if(EmptyQ(Queue))
   {printf("Empty Queue");}
        while (current != Queue.Rear || Queue.Count!=0) {
      printf("%d ", Queue.Element[current]);
                current = (current + 1) % QueueLimit;
      (Queue.Count)--;

        }
        printf("\n");
}

/*
// Η TraverseQ με τη χρήση for

void TraverseQ(QueueType Queue) {
```

```
        int current,i;
   printf("Queue: ");
        if(EmptyQ(Queue)){
        printf("Empty Queue");
        }
        else{
            current=Queue.Front;
        for(i=0;i<Queue.count;i++) {
        printf("%d ",Queue.Element[current]);
        current=(current+1) % QueueLimit;
        }

        }
   printf("\n");
}
*/
```

```
/*  Αρχείο: a12f3.c
    Φοιτητής: Ευστάθιος Ιωσηφίδης


    Άδεια χρήσης: GNU General Public License v3.0

    Σε μία τράπεζα κάθε πελάτης εισέρχεται σ’ αυτήν μία ορισμένη χρονική στιγμή έστω την ώρα Α και παραμένει
    κάποιο χρονικό διάστημα έστω Τ, προκειμένου να διεκπεραιώσει την εργασία του (ανάληψη, μεταφορά, δάνειο κλπ).
    Δεδομένου ότι υπάρχει μόνο ένας ταμίας, εάν κάποιος πελάτης εισέλθει ενώ δεν έχει τελειώσει ο προηγούμενος,
    η εξυπηρέτησή του αρχίζει αμέσως μόλις τελειώσει ο προηγούμενος. Γράψτε πρόγραμμα το οποίο μοντελοποιεί το
    σύστημα εξυπηρέτησης της τράπεζας, χρησιμοποιώντας τον ΑΤΔ ουρά. Κάθε πελάτης που εισέρχεται στην τράπεζα
    μπαίνει σε μια ουρά αναμονής. Για κάθε πελάτη μας ενδιαφέρει να γνωρίζουμε το χρόνο άφιξης, τη διάρκεια παραμονής,
    την ώρα έναρξης, την ώρα λήξης της εξυπηρέτησης. Ο μέγιστος αριθμός πελατών που καταχωρούνται στην ουρά αναμονής είναι 3.
    Το πρόγραμμα θα διαβάζει για κάθε πελάτη την ώρα άφιξης και τον χρόνο παραμονής με τη μορφή ώραΆφιξης, χρόνοςΠαραμονής.
    Η ώρα άφιξης και ο χρόνος παραμονής μπορούν να είναι οποιοδήποτε θετικό μέγεθος.
    Υλοποιήστε 3 συναρτήσεις:

    1. Συνάρτηση που θα διαβάζει την ώρα άφιξης και τον χρόνο παραμονής του κάθε πελάτη με τη μορφή ώραΆφιξης, χρόνοςΠαραμονής
    και θα εισάγει τον πελάτη στην ουρά αναμονής. Για την ώρα έναρξης και ώρα λήξης θα καταχωρείται η «εικονική» τιμή -1
    (η εισαγωγή των πελατών στην ουρά αναμονής θεωρήστε τι συμβαίνει τη χρονική στιγμή που εισέρχεται στην τράπεζα).
    Η συνάρτηση θα είναι void με παράμετρο την ουρά αναμονής.

    2. Συνάρτηση η οποία προσομοιώνει την εξυπηρέτηση των πελατών. Εξάγει έναν-έναν τον πελάτη από την ουρά αναμονής και
    για κάθε πελάτη που εξυπηρετείται θα υπολογίζει την ώρα έναρξης και λήξης της εξυπηρέτησης και θα καταχωρεί τον πελάτη
    στην ουρά εξυπηρετηθέντων (νέα ουρά). Η ώρα έναρξης της εξυπηρέτησης των πελατών είναι ο χρόνος άφιξης του πρώτου πελάτη
    (δες στο στιγμιότυπο εκτέλεσης). Η συνάρτηση έχει μια παράμετρο την ουρά αναμονής και επιστρέφει την ουρά των εξυπηρετηθέντων.
    Το πρωτότυπο της συνάρτησης είναι
    QueueType TimesInQueue(QueueType *Queue)
    3. Συνάρτηση με 2 παραμέτρους: την ονομασία της ουράς, και την ουρά. Αν η ουρά δεν είναι άδεια θα εμφανίζει για κάθε πελάτη
    τις καταχωρημένες πληροφορίες, ενώ αν είναι άδεια θα εμφανίζει σχετικό μήνυμα (δες στο στιγμιότυπο εκτέλεσης).

*/
#include <stdio.h>

#define QueueLimit 4  //το όριο μεγέθους της ουράς

typedef struct{
        int start,end,arrival,stay;
} QueueElementType;   /* ο τύπος δεδομένων των στοιχείων της ουράς χρόνοι έναρξης άφιξης και αναμονής, έναρξης και λήξης εξυπηρέτησης */

typedef struct {
        int Front, Rear;
        QueueElementType Element[QueueLimit];
} QueueType;

typedef enum {FALSE, TRUE} boolean;

// Δήλωση πρώτυπων συναρτήσεων
void CreateQ(QueueType *Queue);
boolean EmptyQ(QueueType Queue);
```

```
boolean FullQ(QueueType Queue);
void RemoveQ(QueueType *Queue, QueueElementType *Item);
void AddQ(QueueType *Queue, QueueElementType Item);

// Πρώτυπα συναρτήσεων που ζητάει η άσκηση
void ReadCustomer(QueueType *Queue);
QueueType TimesInQueue(QueueType *Queue);
void TraverseQ(char nameQ[],QueueType Queue);

int main()
{
   QueueType WaitingQueue; // Δήλωση ουράς αναμονής
   QueueType ServiceQueue; // Δήλωση ουράς εξυπηρέτησης

   int i;

   CreateQ(&WaitingQueue); // Δημιουργία ουράς αναμονής
   CreateQ(&ServiceQueue); // Δήλωση ουράς εξυπηρέτησης

   // Εισαγωγή δεδομένων στην ουρά αναμονής. Χρήση συνάρτησης ReadCustomer
   for(i=1;i<QueueLimit;i++){
      printf("Give: arival time,stay time for client %d: ",i);
      ReadCustomer(&WaitingQueue);
   }

   TraverseQ("Waiting Queue",WaitingQueue); // Εκτύπωση της ουράς αναμονής
   printf("\n");
   ServiceQueue=TimesInQueue(&WaitingQueue); // Ανάθεση στην ουρά εξυπηρέτησης. Χρήση της
συνάρτησης TimesInQueue

   TraverseQ("Waiting Queue",WaitingQueue); // Εκτύπωση της ουράς αναμονής (θεωρητικά δεν
υπάρχει γιατί εξυπηρετήθηκαν όλοι)
   TraverseQ("Service Queue",ServiceQueue); // Εκτύπωση της ουράς πελατών που
εξυπηρετήθηκαν
    return 0;

}

// ΣΥΝΑΡΤΗΣΕΙΣ

void ReadCustomer(QueueType *Queue)
{
   QueueElementType tmpCustomer; // Δημιουργία τύπου πελάτης
   scanf("%d,%d",&tmpCustomer.arrival,&tmpCustomer.stay); // Εισαγωγή χρόνου άφιξης και
χρόνου αναμονής
   tmpCustomer.start=-1; // Αρχικοποίηση χρόνου έναρξης εξυπηρέτησης σε -1
   tmpCustomer.end=-1; // Αρχικοποίηση χρόνου λήξης εξυπηρέτησης σε -1
   AddQ(&(*Queue),tmpCustomer); // Προσθήκη του πελάτη στην ουρά που εισήχθη ως όρισμα
}

QueueType TimesInQueue(QueueType *Queue)
{
   int earliestStart=0; // Αρχικοποίηση
   int currentStart=0; // Αρχικοποίηση
   int currentEnd=0; // Αρχικοποίηση

   QueueType QueueServiced; // Δήλωση ουράς εξυπηρετηθέντων
   QueueElementType CurrentCust; // Δήλωση τύπου τρέχοντα πελάτη

   CreateQ(&QueueServiced); // Δημιουργία ουράς εξυπηρετηθέντων

   while(!EmptyQ(*Queue)) // Όσο η ουρά δεν είναι άδεια
   {
      RemoveQ(Queue,&CurrentCust); // Αφαίρεσε από την ουρά τον πρώτο πελάτη και ανέθεσε
την τιμή στον τρέχοντα πελάτη
      if(CurrentCust.arrival>earliestStart) // Εάν ο χρόνος άφιξης είναι μετά το χρόνο
εκυπηρέτησης
      {
         currentStart=CurrentCust.arrival; // θέσε την τρέχουσα ώρα έναρξης ίση με την
```

```
ώρα άφιξης πελάτη
      }
      else
      {
         currentStart=earliestStart; // Θέσε την τρέχουσα ώρα έναρξης ίση με τον χρόνο
έναρξης εξυπηρέτησης
      }

      currentEnd=currentStart+CurrentCust.stay; // Ενημέρωσε την ώρα λήξης εξυπηρέτησης
για τον τρέχοντα πελάτη (τρέχουσα ώρα έναρξης+χρόνος παραμονής)
      earliestStart=currentEnd; // Ενημέρωσε τον χρόνο έναρξης εξυπηρέτησης για τον
επόμενο πελάτη (χρόνος λήξης τρέχοντα πελάτη)

      CurrentCust.start=currentStart; // Ενημέρωσε και τους χρόνους στον τύπο
      CurrentCust.end=currentEnd; // Ενημέρωσε και τους χρόνους στον τύπο

      AddQ(&QueueServiced,CurrentCust); // Εισαγωγή του τρέχοντα πελάτη στην ουρά
εξυπηρετηθέντων
   }
   return QueueServiced; // Επιστροφή ουράς εξυπηρετηθέντων
}

void TraverseQ(char nameQ[],QueueType Queue)
{
   int current; // Μετρητής θέσης στην ουρά
   int i=1; // Μετρητής για εμφάνιση πελάτη
   if(!EmptyQ(Queue)) // Αν η ουρά δεν είναι άδεια
   {printf("%s\n",nameQ); // Εκτύπωσε το άλφαρηθμιτικό που εισήχθη ως όρισμα)
    current=Queue.Front; // Θέσε τον μετρητή στην αρχή της ουράς
    printf("Client \t\tStart\tEnd\tArrival\tStay\n");

   while(current!=Queue.Rear)
   {
      printf("Client %-d \t%-d\t%-d\t%-d\t%-
d\n",i,Queue.Element[current].start,Queue.Element[current].end,Queue.Element[current].arr
ival,Queue.Element[current].stay);
      current=(current+1)% QueueLimit; // Αύξησε μετρητή
      i++; // Αύξησε μετρητή
   }
   }
   else{
      printf("%s is empty\n", nameQ); // Εκτύπωσε ότι η ουρά είναι άδεια
   }

}



void CreateQ(QueueType *Queue)
/*  Λειτουργία:  Δημιουργεί μια κενή ουρά.
    Επιστρέφει:  Κενή ουρά
*/
{
        Queue->Front = 0;
        Queue->Rear = 0;
}

boolean EmptyQ(QueueType Queue)
/* Δέχεται:    Μια ουρά.
   Λειτουργία: Ελέγχει αν η ουρά είναι κενή.
   Επιστρέφει: True αν η ουρά είναι κενή, False διαφορετικά
*/
{
        return (Queue.Front == Queue.Rear);
}

boolean FullQ(QueueType Queue)
/* Δέχεται:    Μια ουρά.
   Λειτουργία: Ελέγχει αν η ουρά είναι γεμάτη.
```

```
   Επιστρέφει: True αν η ουρά είναι γεμάτη, False διαφορετικά
*/
{
        return ((Queue.Front) == ((Queue.Rear +1) % QueueLimit));
}

void RemoveQ(QueueType *Queue, QueueElementType *Item)
/* Δέχεται:    Μια ουρά.
   Λειτουργία: Αφαιρεί το στοιχείο Item από την εμπρός άκρη της ουράς
               αν η ουρά δεν είναι κενή.
   Επιστρέφει: Το στοιχείο Item και την τροποποιημένη ουρά.
   Έξοδος:     Μήνυμα κενής ουρά αν η ουρά είναι κενή
*/
{
        if(!EmptyQ(*Queue))
        {
                *Item = Queue ->Element[Queue -> Front];
                Queue ->Front  = (Queue ->Front + 1) % QueueLimit;
        }
        else
                printf("Empty Queue");
}

void AddQ(QueueType *Queue, QueueElementType Item)
/* Δέχεται:    Μια ουρά Queue και ένα στοιχείο Item.
   Λειτουργία: Προσθέτει το στοιχείο Item στην ουρά Queue
               αν η ουρά δεν είναι γεμάτη.
   Επιστρέφει: Την τροποποιημένη ουρά.
   Έξοδος:     Μήνυμα γεμάτης ουράς αν η ουρά είναι γεμάτη
*/
{
        if(!FullQ(*Queue))
        {
                Queue ->Element[Queue ->Rear] = Item;
                Queue ->Rear = (Queue ->Rear + 1) % QueueLimit; ;
        }
        else
                printf("Full Queue");
}
```

```
/*  Αρχείο: a1f4.c
    Φοιτητής: Ευστάθιος Ιωσηφίδης


    Άδεια χρήσης: GNU General Public License v3.0

    Να υλοποιηθεί η συνάρτηση Search με το παρακάτω πρωτότυπο:
    boolean Search(ListPointer FreePtr, ListPointer List, NodeType Node[NumberOfNodes],
ListElementType Item, ListPointer *PredPtr);

    η οποία θα δέχεται μια Συνδεδεμένη Λίστα (ΣΛ) υλοποιημένη με πίνακα (παράμετροι
FreePtr, List, Node) και ένα στοιχείο
    Item και θα επιστρέφει boolean τιμή η οποία δείχνει αν βρέθηκε ή όχι το αναζητούμενο
στοιχείο Item στη ΣΛ και τη θέση
    (PredPtr) του στοιχείου που είναι προηγούμενο του Item, ώστε η ΣΛ που θα προκύψει
μετά την εισαγωγή ή τη διαγραφή
    του στοιχείου Item να είναι ταξινομημένη (χρήση της Search πριν την εισαγωγή ή τη
διαγραφή στοιχείου στη ΣΛ). Η
    εισαγωγή στοιχείων στην Σ.Λ. γίνεται επαναληπτικά και ελέγχεται από το μήνυμα
'Continue Y/N:' (δες στιγμιότυπο
    εκτέλεσης). Στη ΣΛ καταχωρούνται ακέραιοι. Η συνάρτηση Search θα κληθεί 2 φορές μια
για στοιχείο που υπάρχει στη ΣΛ
    και μια για στοιχείο που δεν υπάρχει στη ΣΛ. Αν το στοιχείο βρεθεί θα εμφανίζει το
μήνυμα
    «The number is in the list and its predecessor is in position» και την τιμή της
PredPtr, διαφορετικά θα εμφανίζει το μήνυμα
    «The number is not in the list». Το πρόγραμμα θα πρέπει να εκτελεί κατά σειρά τις
παρακάτω λειτουργίες και οι κλήσεις
    των συναρτήσεων θα γίνονται από τη main().
    A. Αρχικοποίηση storage pool
    B. Δημιουργία ΑΣΛ (μέγιστο μέγεθος 10)
    C. Εμφάνιση της storage pool
    D. Εμφάνιση των στοιχείων της ΣΛ
    E. Εισαγωγή στοιχείων στην ΣΛ. Μετά από κάθε εισαγωγή στοιχείου στη ΣΛ αυτή θα πρέπει
να παραμένει
    ταξινομημένη σε αύξουσα διάταξη (χρήση της Search πριν την εισαγωγή στοιχείου στη ΣΛ)
    F. Εμφάνιση της storage pool
    G. Εμφάνιση των στοιχείων της ΣΛ
    H. Έλεγχος εάν η ΣΛ είναι άδεια. Αν η ΣΛ είναι άδεια εμφανίζει μήνυμα «Empty List»
διαφορετικά «Not an Empty List»
    I. Έλεγχος εάν η ΣΛ είναι γεμάτη. Αν η ΣΛ είναι γεμάτη εμφανίζει μήνυμα «Full List»
διαφορετικά «Not a Full List»
    J. Αναζήτηση ενός στοιχείου στη ΣΛ (για στοιχείο που υπάρχει στη ΣΛ και για στοιχείο
που δεν υπάρχει).


*/

#include <stdio.h>
#include <ctype.h>

#define NumberOfNodes 10    /*όριο μεγέθους της συνδεδεμένης λίστας */
#define NilValue -1          // ειδική μεδενικη τιμη δείχνει το τέλος της Συνδ.λίστας

typedef int ListElementType; /*τύπος δεδομένων για τα στοιχεία της συνδεδεμένης λίστας,
                                ενδεικτικά επιλέχθηκε ο τύπος int */
typedef int ListPointer;

typedef struct {
    ListElementType Data;
    ListPointer  Next;
} NodeType;

typedef enum {
    FALSE, TRUE
} boolean;

// ΔΗΛΩΣΕΙΣ ΣΥΝΑΡΤΗΣΕΩΝ
```

```
boolean Search(ListPointer FreePtr, ListPointer List, NodeType Nodes[NumberOfNodes],
ListElementType Item, ListPointer *PredPtr);
void CreateList(ListPointer *List);
void InitializeStoragePool(NodeType Node[], ListPointer *FreePtr);
void Insert(ListPointer *List, NodeType Node[],ListPointer *FreePtr, ListPointer PredPtr,
ListElementType Item);
void GetNode(ListPointer *P, ListPointer *FreePtr, NodeType Node[]);
boolean FullList(ListPointer FreePtr);
boolean EmptyList(ListPointer List);
void printAll(ListPointer List, ListPointer FreePtr, NodeType Node[]);
void TraverseLinked(ListPointer List, NodeType Node[]);

int main()
{
  ListPointer AList;
  NodeType Node[NumberOfNodes];
  ListPointer FreePtr,PredPtr;
  boolean found;
  int i;

  ListElementType AnItem;

  char ch;

  // Ερώτημα A
  InitializeStoragePool(Node, &FreePtr);

  // Ερώτημα B
  CreateList(&AList);

  // Ερώτημα C
  printf("---------------Question C--------------------\n");
  printf("--------------Storage pool------------------\n");
  printAll(AList, FreePtr, Node);

  // Ερώτημα D
  printf("---------------Question D--------------------\n");
  printf("--------------Linked list------------------\n");
  TraverseLinked(AList, Node);

  // Ερώτημα E
  printf("---------------Question E--------------------\n");
  do
    {
     printf("Give a number: ");
     scanf("%d", &AnItem);

     found=Search(FreePtr, AList, Node, AnItem, &PredPtr);

     Insert(&AList, Node, &FreePtr, PredPtr, AnItem);

     printf("\nContinue Y/N: ");

     do
       {
       scanf("%c", &ch);
       } while (toupper(ch)!= 'N' && toupper(ch)!= 'Y');

       } while (toupper(ch)!='N');


  // Ερώτημα F
  printf("---------------Question F--------------------\n");
  printf("--------------Storage pool------------------\n");
  printAll(AList, FreePtr, Node);

  // Ερώτημα G
  printf("---------------Question G--------------------\n");
  printf("--------------Linked list------------------\n");
```

```
  TraverseLinked(AList, Node);

  // Ερώτημα H
  printf("---------------Question H-------------------\n");
  if(EmptyList(AList))
  {printf("Empty List\n");}
  else
  {printf("Not an Empty List\n");}

  // Ερώτημα I
  printf("---------------Question I-------------------\n");
  if(FullList(FreePtr))
  {printf("Full List\n");}
  else
  {printf("Not a Full List\n");}

  // Ερώτημα J
  printf("---------------Question J-------------------\n");
  printf("-----------------Search for a number----------------\n");
  for(i=0;i<2;i++){
     printf("Give a number ");
     scanf("%d", &AnItem);

     found=Search(FreePtr, AList, Node, AnItem, &PredPtr);
     if(found)
     {printf("The number is in the list and its predecessor is in position
%d\n",PredPtr);}
     else
     {printf("The number is not in the list\n");}
}

        return 0;
}

// ΣΥΝΑΡΤΗΣΕΙΣ

boolean Search(ListPointer FreePtr, ListPointer List, NodeType Nodes[NumberOfNodes],
ListElementType Item, ListPointer *PredPtr)
{
  boolean stop;
  boolean found;
  ListPointer current;

  stop=FALSE;
  found=FALSE;
  *PredPtr = NilValue;

  if(!EmptyList(List))
  {
    current=List;
    while (current != NilValue && !stop) {
      if(Nodes[current].Data >= Item){
      stop=TRUE;
      found=(Nodes[current].Data==Item);
      }
      else {
      *PredPtr=current;
      current=Nodes[current].Next;
     }
    }
  }

  else
    {found=FALSE;}

  return found;
}
```

```
void CreateList(ListPointer *List)
/* Λειτουργία: Δημιουργεί μια κενή συνδεδεμένη λίστα.
   Επιστρέφει: Έναν (μηδενικό) δείκτη που δείχνει σε κενή λίστα
*/
{
  *List=NilValue;
}

void InitializeStoragePool(NodeType Node[], ListPointer *FreePtr)
/* Δέχεται:   Τον πίνακα Node και τον δείκτη FreePtr που δείχνει στον
                πρώτο διαθέσιμο κόμβο.
  Λειτουργία: Αρχικοποιεί τον πίνακα Node ως συνδεδεμένη λίστα συνδέοντας μεταξύ
                τους διαδοχικές εγγραφές του πίνακα,
                και αρχικοποιεί τον δείκτη FreePtr .
  Επιστρέφει: Τον τροποποιημένο πίνακα Node και τον
                δείκτη FreePtr του πρώτου διαθέσιμου κόμβου
*/
{
   int i;

    for (i=0; i<NumberOfNodes-1;i++)
    {
        Node[i].Next=i+1;
        Node[i].Data=-1;
    }
    Node[NumberOfNodes-1].Next=NilValue;
    Node[NumberOfNodes-1].Data=NilValue;
    *FreePtr=0;
}

void Insert(ListPointer *List, NodeType Node[],ListPointer *FreePtr, ListPointer PredPtr,
ListElementType Item)
/* Δέχεται:    Μια συνδεδεμένη λίστα, τον πίνακα Node, τον δείκτη PredPtr και
                ένα στοιχείο Item.
  Λειτουργία: Εισάγει στη συνδεδεμένη λίστα, αν δεν είναι γεμάτη, το στοιχείο
                Item μετά από τον κόμβο στον οποίο δείχνει ο δείκτης PredPtr.
  Επιστρέφει: Την τροποποιημένη συνδεδεμένη λίστα, τον τροποποιημένο πίνακα Node
                και τον δείκτη FreePtr.
  Εξοδος:     Μήνυμα γεμάτης λίστας, αν η συνδεδεμένη λίστα είναι γεμάτη
*/
{
  ListPointer TempPtr;
  GetNode(&TempPtr,FreePtr,Node);
  if (!FullList(TempPtr)) {
    if (PredPtr==NilValue)
    {
        Node[TempPtr].Data =Item;
        Node[TempPtr].Next =*List;
        *List =TempPtr;
    }
    else
      {
        Node[TempPtr].Data =Item;
        Node[TempPtr].Next =Node[PredPtr].Next;
        Node[PredPtr].Next =TempPtr;
      }
}
  else
    printf("Full List ...\n");
}

void GetNode(ListPointer *P, ListPointer *FreePtr, NodeType Node[])
/* Δέχεται:   Τον πίνακα Node και τον δείκτη FreePtr.
  Λειτουργία: Αποκτά έναν "ελεύθερο" κόμβο που τον δείχνει ο δείκτης P.
  Επιστρέφει: Τον δείκτη P και τον τροποποιημένο  δείκτη FreePtr
                που δεικτοδοτεί στο πρώτο διαθέσιμο κόμβο
*/
{
```

```
  *P = *FreePtr;
  if (!FullList(*FreePtr))
    *FreePtr =Node[*FreePtr].Next;
}


boolean EmptyList(ListPointer List)
/* Δέχεται:    Έναν δείκτη List που δείχνει σε μια συνδεδεμένη λίστα.
   Λειτουργία: Ελέγχει αν η συνδεδεμένη λίστα είναι κενή.
   Επιστρέφει: True αν η συνδεδεμένη λίστα είναι κενή και false διαφορετικά
*/
{
  return (List==NilValue);
}

boolean FullList(ListPointer FreePtr)
/* Δέχεται:    Μια συνδεδεμένη λίστα.
   Λειτουργία: Ελέγχει αν η συνδεδεμένη λίστα είναι γεμάτη.
   Επιστρέφει: True αν η συνδεδεμένηλίστα είναι γεμάτη, false διαφορετικά
*/
{
  return (FreePtr == NilValue);
}

void printAll(ListPointer List, ListPointer FreePtr, NodeType Node[])
{
    int i;
    printf("1o STOIXEIO LISTAS=%d, 1H FREE POSITION=%d\n", List, FreePtr);
    printf("H STORAGE POOL EXEI TA EJHS STOIXEIA\n");
    for (i=0;i<NumberOfNodes;i++)
        printf("(%d: %d->%d) ",i,Node[i].Data, Node[i].Next);
    printf("\n");
}

void TraverseLinked(ListPointer List, NodeType Node[])
/* Δέχεται:    Μια συνδεδεμένη λίστα.
   Λειτουργία: Κάνει διάσχιση της συνδεδεμένης λίστας, αν δεν είναι κενή.
   Έξοδος:     Εξαρτάται από την επεξεργασία
*/
{
  ListPointer CurrPtr;

  if (!EmptyList(List))
  {
      CurrPtr =List;
      while (CurrPtr != NilValue)
      {
          printf("(%d: %d->%d) ",CurrPtr,Node[CurrPtr].Data, Node[CurrPtr].Next);
          CurrPtr=Node[CurrPtr].Next;
       }
       printf("\n");
   }
  else printf("Empty List ...\n");
}
```

```
/*  Αρχείο: a30f4.c
    Φοιτητής: Ευστάθιος Ιωσηφίδης


    Άδεια χρήσης: GNU General Public License v3.0

    Σε μια ΣΛ θέλουμε να καταχωρούμε τον Αριθμό Μητρώο (int) και τον βαθμό (float) ενός
μαθητή. Τροποποιήστε τον
    τύπο ListElementType ώστε να αποθηκεύει τον ΑΜ και το βαθμό του μαθητή. Κάντε τις
κατάλληλες τροποποιήσεις όπου
    είναι απαραίτητο, στις συναρτήσεις που θα αντιγράψετε από τα αρχεία L_ListADT.c &
L_ListADT.h. Θέλουμε να εντοπίζουμε
    και να εμφανίσουμε τα στοιχεία των μαθητών που έχουν τον ελάχιστο βαθμό. Να
υλοποιήσετε τη συνάρτηση FindMins που
    θα δέχεται 2 παραμέτρους: τη ΣΛ με τα στοιχεία των μαθητών και μια στοίβα όπου θα
καταχωρεί τους ΑΜ όλων των
    μαθητών με τον ίδιο ελάχιστο βαθμό, και θα επιστρέφει την τιμή του ελάχιστου βαθμού.
Το πρωτότυπο της συνάρτησης:

    float FindMins(ListPointer List, NodeType Node[NumberOfNodes], StackType *Stack)

    Αφού εντοπιστούν οι μαθητές με τους ελάχιστους βαθμούς, στη συνέχεια θα εμφανιστούν
στην οθόνη.
    Η εμφάνιση των ΑΜ των μαθητών με τον ίδιο ελάχιστο βαθμό θα γίνεται στη main() με
χρήση των επιτρεπόμενων
    λειτουργιών της στοίβας (άδειασμα στοίβας). Στη συνέχεια θα καλείται η βοηθητική
συνάρτηση TraverseStack για την
    επαλήθευση της ορθής υλοποίησης αυτού του ερωτήματος (η στοίβα θα πρέπει να είναι
κενή μετά την «εμφάνιση»-
    διαγραφή από τη στοίβα των ΑΜ). Το περιεχόμενο κάθε κόμβου της ΣΛ θα εμφανίζεται ως
εξής: <ΑΜ, Βαθμός> πχ αν ο
    κόμβος της ΣΛ είναι στη θέση 0 του πίνακα με ΑΜ 5 και βαθμό 10 και ο επόμενος κόμβος
της ΣΛ είναι αποθηκευμένος στη
    θέση 2 του πίνακα, τότε η εμφάνιση του κόμβου θα είναι: (0|<5,10.0> -> 2) που
σημαίνει ότι ο κόμβος με περιεχόμενο 5, 10
    είναι αποθηκευμένος στη θέση 0 του πίνακα και έχει επόμενο κόμβο αποθηκευμένο στη
θέση 2 του πίνακα.
    Το πρόγραμμα θα πρέπει να εκτελεί κατά σειρά τις παρακάτω λειτουργίες και οι κλήσεις
των συναρτήσεων θα γίνονται από τη main().
    A. Αρχικοποίηση storage pool
    B. Δημιουργία ΣΛ (μέγιστο μέγεθος 10)
    C. Εμφάνιση της storage pool
    D. Εμφάνιση των στοιχείων της ΣΛ
    E. Εισαγωγή 5 στοιχείων στην ΣΛ. (Η εισαγωγή στοιχείου θα γίνεται πάντα στην αρχή της
ΣΛ)
    F. Εμφάνιση της storage pool
    G. Εμφάνιση των στοιχείων της ΣΛ
    H. Έλεγχος εάν η ΣΛ είναι άδεια. Αν η ΣΛ είναι άδεια εμφανίζει μήνυμα «Empty List»
διαφορετικά «Not an Empty List»
    I. Έλεγχος εάν η ΣΛ είναι γεμάτη. Αν η ΣΛ είναι γεμάτη εμφανίζει μήνυμα «Full List»
διαφορετικά «Not a Full List»
    J. Εμφάνιση του ελάχιστου βαθμού και των ΑΜ των μαθητών με το ελάχιστο βαθμό
    K. Κλήση της TraverseStack για την επαλήθευση της ορθής υλοποίησης του ερωτήματος J
    L. Εμφάνιση της storage pool
    M. Εμφάνιση των στοιχείων της ΣΛ

*/

#include <stdio.h>

#define NumberOfNodes 10    /*όριο μεγέθους της συνδεδεμένης λίστας 10*/
#define NilValue -1          // ειδική μεδενικη τιμη δείχνει το τέλος της Συνδ.λίστας
#define StackLimit 20

// Τύποι δεδομένων λίστας
typedef struct
{
  int am;
```

```
  float grade;
} ListElementType; /*τύπος δεδομένων για τα στοιχεία της συνδεδεμένης λίστας*/

typedef int ListPointer;

typedef struct {
    ListElementType Data;
    ListPointer  Next;
} NodeType;

typedef enum {
    FALSE, TRUE
} boolean;

// Τύποι δεδομένων στοίβας
typedef int StackElementType;

typedef struct  {
    int Top;
    StackElementType Element[StackLimit];
} StackType;

// Δήλωση συναρτήσεων λίστας
float FindMins(ListPointer List, NodeType Node[NumberOfNodes], StackType *Stack);
void CreateList(ListPointer *List);
void InitializeStoragePool(NodeType Node[], ListPointer *FreePtr);
void Insert(ListPointer *List, NodeType Node[],ListPointer *FreePtr, ListPointer PredPtr,
ListElementType Item);
void GetNode(ListPointer *P, ListPointer *FreePtr, NodeType Node[]);
boolean FullList(ListPointer FreePtr);
boolean EmptyList(ListPointer List);
void printAll(ListPointer List, ListPointer FreePtr, NodeType Node[]);
void TraverseLinked(ListPointer List, NodeType Node[]);


// Δήλωση συναρτήσεων στοίβας
void CreateStack(StackType *Stack);
void Push(StackType *Stack, StackElementType Item);
void Pop(StackType *Stack, StackElementType *Item);
boolean FullStack(StackType Stack);
boolean EmptyStack(StackType Stack);
void TraverseStack(StackType Stack);


int main()
{
  int will_i=0;
  float min;
  NodeType Node[NumberOfNodes];
  ListPointer AList, FreePtr, PredPtr;
  ListElementType AnItem;
  StackElementType AM;
  StackType Stack;

  PredPtr=NilValue;

  // Ερώτημα A
  InitializeStoragePool(Node, &FreePtr);

  // Ερώτημα B
  CreateList(&AList);

  // Ερώτημα C
  printf("---------------Question C--------------------\n");
  printf("--------------Storage pool------------------\n");
  printAll(AList, FreePtr, Node);

  // Ερώτημα D
  printf("---------------Question D--------------------\n");
```

```
  printf("--------------Linked list-----------------\n");
  TraverseLinked(AList, Node);

  // Ερώτημα E
  printf("---------------Question E-------------------\n");

  for(will_i=0;will_i<5 ;will_i++)
  {
    printf("DWSE AM GIA EISAGWGH STH LISTA: ");
    scanf("%d", &AnItem.am);
    printf("DWSE VATHMO GIA EISAGWGH STH LISTA: ");
    scanf("%f", &AnItem.grade);
    Insert(&AList, Node, &FreePtr, PredPtr, AnItem);
  }

  // Ερώτημα F
  printf("---------------Question F-------------------\n");
  printf("--------------Storage pool-----------------\n");
  printAll(AList, FreePtr, Node);

  // Ερώτημα G
  printf("---------------Question G-------------------\n");
  printf("--------------Linked list-----------------\n");
  TraverseLinked(AList, Node);

  // Ερώτημα H
  printf("---------------Question H-------------------\n");
  if(EmptyList(AList))
  {printf("Empty List\n");}
  else
  {printf("Not an Empty List\n");}

  // Ερώτημα I
  printf("---------------Question I-------------------\n");
  if(FullList(FreePtr))
  {printf("Full List\n");}
  else
  {printf("Not a Full List\n");}

  // Ερώτημα J
  printf("---------------Question J-------------------\n");
  min=FindMins(AList, Node, &Stack);
  printf("Min value=%.1f\n",min);
  printf("AM with min grade are: ");
  while(!EmptyStack(Stack))
    {Pop(&Stack, &AM);
    printf("%d ",AM);}

  // Ερώτημα K
  printf("\n---------------Question K-------------------\n");

  TraverseStack(Stack);

  // Ερώτημα L
  printf("---------------Question L-------------------\n");
  printf("--------------Storage pool-----------------\n");
  printAll(AList, FreePtr, Node);

  // Ερώτημα M
  printf("---------------Question M-------------------\n");
  printf("--------------Linked list-----------------\n");
  TraverseLinked(AList, Node);

    return 0;
}

// ΣΥΝΑΡΤΗΣΕΙΣ ΛΙΣΤΑΣ

float FindMins(ListPointer List, NodeType Node[NumberOfNodes], StackType *Stack)
```

```
{
  ListPointer CurrPtr;
  float min;
  StackElementType Item, studentAM;

  CreateStack(Stack);

  if(!EmptyList(List))
  {
    CurrPtr=List;
    min=Node[CurrPtr].Data.grade;

    while(CurrPtr!=NilValue)
    {
      if(Node[CurrPtr].Data.grade<=min)
      {
        if(Node[CurrPtr].Data.grade<min){
          while(!EmptyStack(*Stack)){
            Pop(&(*Stack), &Item);
          }
          min=Node[CurrPtr].Data.grade;
        }

        studentAM = Node[CurrPtr].Data.am;
        Push(&(*Stack), studentAM);
      }
       CurrPtr = Node[CurrPtr].Next;
    }
  }
  else
  {printf("Empty List...");}

  return min;
}

void CreateList(ListPointer *List)
/* Λειτουργία: Δημιουργεί μια κενή συνδεδεμένη λίστα.
   Επιστρέφει: Έναν (μηδενικό) δείκτη που δείχνει σε κενή λίστα
*/
{
  *List=NilValue;
}

void InitializeStoragePool(NodeType Node[], ListPointer *FreePtr)
/* Δέχεται:   Τον πίνακα Node και τον δείκτη FreePtr που δείχνει στον
                πρώτο διαθέσιμο κόμβο.
  Λειτουργία: Αρχικοποιεί τον πίνακα Node ως συνδεδεμένη λίστα συνδέοντας μεταξύ
                τους διαδοχικές εγγραφές του πίνακα,
                και αρχικοποιεί τον δείκτη FreePtr .
  Επιστρέφει: Τον τροποποιημένο πίνακα Node και τον
                δείκτη FreePtr του πρώτου διαθέσιμου κόμβου
*/
{
   int i;

    for (i=0; i<NumberOfNodes-1;i++)
    {
        Node[i].Next=i+1;
        Node[i].Data.am=-1;
        Node[i].Data.grade=-1.0;
    }
    Node[NumberOfNodes-1].Next=NilValue;
    Node[NumberOfNodes-1].Data.am=NilValue;
    Node[NumberOfNodes-1].Data.grade=-1.0;
    *FreePtr=0;
}

void Insert(ListPointer *List, NodeType Node[],ListPointer *FreePtr, ListPointer PredPtr,
ListElementType Item)
```

```
/* Δέχεται:    Μια συνδεδεμένη λίστα, τον πίνακα Node, τον δείκτη PredPtr και
                ένα στοιχείο Item.
  Λειτουργία: Εισάγει στη συνδεδεμένη λίστα, αν δεν είναι γεμάτη, το στοιχείο
                Item μετά από τον κόμβο στον οποίο δείχνει ο δείκτης PredPtr.
  Επιστρέφει: Την τροποποιημένη συνδεδεμένη λίστα, τον τροποποιημένο πίνακα Node
                και τον δείκτη FreePtr.
  Εξοδος:     Μήνυμα γεμάτης λίστας, αν η συνδεδεμένη λίστα είναι γεμάτη
*/
{
  ListPointer TempPtr;
  GetNode(&TempPtr,FreePtr,Node);
  if (!FullList(TempPtr)) {
    if (PredPtr==NilValue)
    {
        Node[TempPtr].Data =Item;
        Node[TempPtr].Next =*List;
        *List =TempPtr;
    }
    else
      {
        Node[TempPtr].Data =Item;
        Node[TempPtr].Next =Node[PredPtr].Next;
        Node[PredPtr].Next =TempPtr;
      }
}
  else
    printf("Full List ...\n");
}

void GetNode(ListPointer *P, ListPointer *FreePtr, NodeType Node[])
/* Δέχεται:   Τον πίνακα Node και τον δείκτη FreePtr.
  Λειτουργία: Αποκτά έναν "ελεύθερο" κόμβο που τον δείχνει ο δείκτης P.
  Επιστρέφει: Τον δείκτη P και τον τροποποιημένο  δείκτη FreePtr
                που δεικτοδοτεί στο πρώτο διαθέσιμο κόμβο
*/
{
  *P = *FreePtr;
  if (!FullList(*FreePtr))
    *FreePtr =Node[*FreePtr].Next;
}


boolean EmptyList(ListPointer List)
/* Δέχεται:    Έναν δείκτη List που δείχνει σε μια συνδεδεμένη λίστα.
   Λειτουργία: Ελέγχει αν η συνδεδεμένη λίστα είναι κενή.
   Επιστρέφει: True αν η συνδεδεμένη λίστα είναι κενή και false διαφορετικά
*/
{
  return (List==NilValue);
}

boolean FullList(ListPointer FreePtr)
/* Δέχεται:    Μια συνδεδεμένη λίστα.
   Λειτουργία: Ελέγχει αν η συνδεδεμένη λίστα είναι γεμάτη.
   Επιστρέφει: True αν η συνδεδεμένηλίστα είναι γεμάτη, false διαφορετικά
*/
{
  return (FreePtr == NilValue);
}

void printAll(ListPointer List, ListPointer FreePtr, NodeType Node[])
{
    int i;
    printf("1o STOIXEIO LISTAS=%d, 1H FREE POSITION=%d\n", List, FreePtr);
    printf("H STORAGE POOL EXEI TA EJHS STOIXEIA\n");
    for (i=0;i<NumberOfNodes;i++)
        printf("(%d:<%d,%.1f> ->%d) ",i,Node[i].Data.am, Node[i].Data.grade,
Node[i].Next);
    printf("\n");
```

```
}

void TraverseLinked(ListPointer List, NodeType Node[])
/* Δέχεται:    Μια συνδεδεμένη λίστα.
   Λειτουργία: Κάνει διάσχιση της συνδεδεμένης λίστας, αν δεν είναι κενή.
   Έξοδος:     Εξαρτάται από την επεξεργασία
*/
{
  ListPointer CurrPtr;

  if (!EmptyList(List))
  {
      CurrPtr =List;
      while (CurrPtr != NilValue)
      {
          printf("(%d:<%d,%.1f> ->%d) ",CurrPtr,Node[CurrPtr].Data.am,
Node[CurrPtr].Data.grade, Node[CurrPtr].Next);
          CurrPtr=Node[CurrPtr].Next;
       }
       printf("\n");
   }
  else {
    printf("Empty List ...\n");
    }
}

// ΣΥΝΑΡΤΗΣΕΙΣ ΣΤΟΙΒΑΣ

void CreateStack(StackType *Stack)
/* Λειτουργία: Δημιουργεί μια κενή στοίβα.
   Επιστρέφει: Κενή Στοίβα.*
*/
{
    Stack -> Top = -1;
   // (*Stack).Top = -1;
}

boolean EmptyStack(StackType Stack)
/* Δέχεται: Μια στοίβα Stack.
   Λειτουργία: Ελέγχει αν η στοίβα Stack είναι κενή.
   Επιστρέφει: True αν η Stack είναι κενή, False διαφορετικά
*/
{
    return (Stack.Top == -1);
}

boolean FullStack(StackType Stack)
/* Δέχεται: Μια στοίβα Stack.
   Λειτουργία: Ελέγχει αν η στοίβα Stack είναι γεμάτη.
   Επιστρέφει: True αν η Stack είναι γεμάτη, False διαφορετικά
*/
{
    return (Stack.Top == (StackLimit - 1));
}

void Push(StackType *Stack, StackElementType Item)
/* Δέχεται: Μια στοίβα Stack και ένα στοιχείο Item.
   Λειτουργία: Εισάγει το στοιχείο Item στην στοίβα Stack αν η Stack δεν είναι γεμάτη.
   Επιστρέφει: Την τροποποιημένη Stack.
   Έξοδος: Μήνυμα γεμάτης στοίβας, αν η στοίβα Stack είναι γεμάτη
*/
{
    if (!FullStack(*Stack)) {
        Stack -> Top++;
        Stack -> Element[Stack -> Top] = Item;
    } else
        printf("Full Stack...");
}
```

```
void Pop(StackType *Stack, StackElementType *Item)
/* Δέχεται: Μια στοίβα Stack.
   Λειτουργία: Διαγράφει το στοιχείο Item από την κορυφή της Στοίβας αν η Στοίβα δεν
είναι κενή.
   Επιστρέφει: Το στοιχείο Item και την τροποποιημένη Stack.
   Έξοδος:  Μήνυμα κενής στοίβας αν η Stack είναι κενή
*/
{
    if (!EmptyStack(*Stack)) {
        *Item = Stack -> Element[Stack -> Top];
        Stack -> Top--;
    } else
        printf("Empty Stack...");
}

 // ΑΥΤΗ Η ΕΚΔΟΣΗ ΕΙΝΑΙ ΣΤΙΣ ΣΗΜΕΙΩΣΕΙΣ ΚΑΙ ΕΜΦΑΝΙΖΕΙ ΤΑ ΣΤΟΙΧΕΙΑ ΤΗΣ ΣΤΟΙΒΑΣ ΑΠΟ ΤΗ ΘΕΣΗ
0-TOP
 void TraverseStack(StackType Stack)
{
    int i;
    printf("\nplithos sto stack %d\n",Stack.Top+1);
    for (i=0;i<=Stack.Top;i++) {
        printf("%d ",Stack.Element[i]);
    }
    printf("\n");
}
```

```
/*  Αρχείο: a2cf4.c
    Φοιτητής: Ευστάθιος Ιωσηφίδης


    Άδεια χρήσης: GNU General Public License v3.0

    Για τις παρακάτω λειτουργίες που αφορούν σε απλή συνδετική λίστα (Α.Σ.Λ.) με δείκτες
να γραφούν διαφορετικά
    προγράμματα για καθεμιά από αυτές. Γνωρίζουμε τα εξής:
    • Κάθε κόμβος της ΑΣΛ ή των ΑΣΛ περιέχει έναν ακέραιο αριθμό.
    • Η είσοδος των δεδομένων θα γίνεται ως εξής: Πρώτα θα διαβάζεται για κάθε ΑΣΛ το
πλήθος των κόμβων της Α.Σ.Λ.
    και στη συνέχεια τα στοιχεία της.
    • Σε όλες τις παρακάτω ασκήσεις η εισαγωγή στοιχείου, για την κατασκευή της
υπάρχουσας λίστας γίνεται στην
    αρχή της λίστας.
    • Σε κάθε πρόγραμμα θα εμφανίζονται οι κόμβοι της αρχικής ή των αρχικών λιστών και
στη συνέχεια αυτών που
    δημιουργούνται κατά την εκτέλεση του προγράμματος.
    • Η εμφάνιση των δεδομένων θα γίνεται ως εξής: Τα στοιχεία των κόμβων της Α.Σ.Λ. θα
εμφανίζονται σε μια γραμμή
    με ένα κενό χαρακτήρα μεταξύ τους.
    • Αν κατά τη διάσχιση της λίστας διαπιστώσετε ότι η Α.Σ.Λ. είναι κενή, τότε να
εμφανίζετε το μήνυμα ‘EMPTY LIST’.
    Σ’ όλα τα προγράμματα θα πρέπει να κατασκευάσετε την ΑΣΛ ή τις ΑΣΛς διαβάζοντας τα
δεδομένα ως εξής: πλήθος
    στοιχείων λίστας, στοιχεία λίστας ή αν πρόκειται για 2 λίστες τότε θα τα διαβάζετε ως
εξής: πλήθος στοιχείων λίστας1,
    στοιχεία λίστας1, πλήθος στοιχείων λίστας2, στοιχεία λίστας2
    Η καθεμία από τις παρακάτω λειτουργίες να υλοποιηθεί με τη χρήση συνάρτησης.

    (c) Συνένωση 2 Α.Σ.Λ. σε μία (πχ 1 ΑΣΛ: 4, 3, 2, 1 και 2 ΑΣΛ: 8, 7, 6, τελική ΑΣΛ: 6,
7, 8, 1, 2, 3, 4). Δίνεται το πρωτότυπο
    της συνάρτησης:
    void concat_list(ListPointer List, ListPointer BList, ListPointer *FinalList)
    (τα στοιχεία των λιστών δεν ακολουθούν κάποια διάταξη)

*/


#include <stdio.h>
#include <stdlib.h>

// Δήλωση τύπων

typedef int ListElementType;           /* ο τύπος των στοιχείων της συνδεδεμένης λίστας
                                        ενδεικτικά τύπου int */
typedef struct ListNode *ListPointer;   //ο τύπος των δεικτών για τους κόμβους
typedef struct ListNode
{
        ListElementType Data;
    ListPointer Next;
} ListNode;

typedef enum {
    FALSE, TRUE
} boolean;

// Δήλωση συναρτήσεων
void concat_list(ListPointer List, ListPointer BList, ListPointer *FinalList);
void CreateList(ListPointer *List);
boolean EmptyList(ListPointer List);
void LinkedInsert(ListPointer *List, ListElementType Item, ListPointer PredPtr);
void LinkedDelete(ListPointer *List, ListPointer PredPtr);
void LinkedTraverse(ListPointer List);
void LinearSearch(ListPointer List, ListElementType Item, ListPointer *PredPtr, boolean
*Found);
void OrderedLinearSearch(ListPointer List, ListElementType Item, ListPointer *PredPtr,
```

```
boolean *Found);

int main()
{
// Δήλωση μεταβλητών
ListPointer AList, BList, FinalList;
ListElementType Item;

int i,n;

CreateList(&AList);
CreateList(&BList);
CreateList(&FinalList);

// Εισαγωγή στοιχείων στις λίστες
printf("DWSE TON PLH8OS TWN STOIXEIWN THS LISTAS 1: ");
scanf("%d",&n);
for(i=0;i<n;i++)
{
    printf("DWSE TON ARI8MO GIA EISAGWGH STH ARXH THS LISTAS 1: ");
    scanf("%d",&Item);

    LinkedInsert(&AList, Item, NULL);

}

printf("DWSE TON PLH8OS TWN STOIXEIWN THS LISTAS 2: ");
scanf("%d",&n);
for(i=0;i<n;i++)
{
    printf("DWSE TON ARI8MO GIA EISAGWGH STH ARXH THS LISTAS 2: ");
    scanf("%d",&Item);

    LinkedInsert(&BList, Item, NULL);

}

// Εμφάνιση στοιχείων των λιστών
printf("LISTA 1:\n");
LinkedTraverse(AList);
printf("\n");
printf("LISTA 2:\n");
LinkedTraverse(BList);
printf("\n");
// Κλήση συνάρτησης
concat_list(AList, BList, &FinalList);

// Εμφάνιση τελικής λίστας
printf("SYNENWMENH LISTA:\n");
LinkedTraverse(FinalList);
printf("\n");
    return 0;
}

// Υλοποίηση συναρτήσεων

void concat_list(ListPointer List, ListPointer BList, ListPointer *FinalList)
{
    ListPointer CurrPtr;

    //Διασχίζεται η 1η λίστα και τα στοιχεία της εισάγονται ένα-ένα στην τελική λίστα

    CurrPtr= List;
    while (CurrPtr != NULL)
    {
        LinkedInsert(FinalList,CurrPtr->Data,NULL);
        CurrPtr=CurrPtr->Next;
    }
```

```
    //Διασχίζεται η 1η λίστα και τα στοιχεία της εισάγονται ένα-ένα στην τελική λίστα
    CurrPtr= BList;
    while (CurrPtr != NULL)
    {
        LinkedInsert(FinalList,CurrPtr->Data,NULL);
        CurrPtr=CurrPtr->Next;
    }
}

void CreateList(ListPointer *List)
/* Λειτουργία: Δημιουργεί μια κενή συνδεδεμένη λίστα.
  Επιστρέφει:  Τον μηδενικό δείκτη List
*/
{
        *List = NULL;
}

boolean EmptyList(ListPointer List)
/* Δέχεται:   Μια συνδεδεμένη λίστα με τον List να δείχνει στον πρώτο κόμβο.
  Λειτουργία: Ελέγχει αν η συνδεδεμένη λίστα είναι κενή.
  Επιστρέφει: True αν η λίστα είναι κενή και false διαφορετικά
*/
{
        return (List==NULL);
}

void LinkedInsert(ListPointer *List, ListElementType Item, ListPointer PredPtr)
/* Δέχεται:    Μια συνδεδεμένη λίστα με τον List να δείχνει στον πρώτο κόμβο,
                ένα στοιχείο δεδομένων Item και έναν δείκτη PredPtr.
   Λειτουργία: Εισάγει έναν κόμβο, που περιέχει το Item, στην συνδεδεμένη λίστα
                μετά από τον κόμβο που δεικτοδοτείται από τον PredPtr
                ή στην αρχή  της συνδεδεμένης λίστας,
                αν ο PredPtr είναι μηδενικός(NULL).
  Επιστρέφει:  Την τροποποιημένη συνδεδεμένη λίστα με τον πρώτο κόμβο της
                να δεικτοδοτείται από τον List.
*/
{
        ListPointer TempPtr;

   TempPtr= (ListPointer)malloc(sizeof(struct ListNode));
 /*  printf("Insert &List %p, List %p,  &(*List) %p, (*List) %p, TempPtr %p\n",
   &List, List,  &(*List), (*List), TempPtr); */
   TempPtr->Data = Item;
        if (PredPtr==NULL) {
        TempPtr->Next = *List;
        *List = TempPtr;
    }
    else {
        TempPtr->Next = PredPtr->Next;
        PredPtr->Next = TempPtr;
    }
}

void LinkedDelete(ListPointer *List, ListPointer PredPtr)
/* Δέχεται:    Μια συνδεδεμένη λίστα με τον List να δείχνει στον πρώτο κόμβο της
                 και έναν δείκτη PredPtr.
   Λειτουργία: Διαγράφει από τη συνδεδεμένη λίστα τον κόμβο που έχει
                για προηγούμενό του αυτόν στον οποίο δείχνει ο PredPtr
                ή διαγράφει τον πρώτο κόμβο, αν ο PredPtr είναι μηδενικός,
                εκτός και αν η λίστα είναι κενή.
   Επιστρέφει: Την τροποποιημένη συνδεδεμένη λίστα με τον πρώτο κόμβο
                να δεικτοδοτείται από τον List.
   Έξοδος:     Ένα μήνυμα κενής λίστας αν η συνδεδεμένη λίστα ήταν κενή .
*/
{
    ListPointer TempPtr;

    if (EmptyList(*List))
            printf("EMPTY LIST\n");
```

```
   else
   {
            if (PredPtr == NULL)
        {
              TempPtr = *List;
              *List = TempPtr->Next;
        }
        else
        {
             TempPtr = PredPtr->Next;
             PredPtr->Next = TempPtr->Next;
        }
        free(TempPtr);
    }
}

void LinkedTraverse(ListPointer List)
/* Δέχεται:    Μια συνδεδεμένη λίστα με τον List να δείχνει στον πρώτο κόμβο.
   Λειτουργία: Διασχίζει τη συνδεδεμένη λίστα και
                επεξεργάζεται κάθε δεδομένο ακριβώς μια φορά.
   Επιστρέφει: Εξαρτάται από το είδος της επεξεργασίας.
*/
{
        ListPointer CurrPtr;

   if (EmptyList(List))
            printf("EMPTY LIST\n");
   else
   {
            CurrPtr = List;
         //   printf("%p\n",CurrPtr);
         //   printf("%-16s\t%-4s\t%-16s\n", "CurrPtr","Data","Next");
         while ( CurrPtr!=NULL )
        {
             //printf("%p\t%d\t%p\n",CurrPtr,(*CurrPtr).Data, (*CurrPtr).Next);
             printf("%d ",CurrPtr->Data);
             CurrPtr = CurrPtr->Next;
        }
   }
}

void LinearSearch(ListPointer List, ListElementType Item, ListPointer *PredPtr, boolean
*Found)
/* Δέχεται:   Μια συνδεδεμένη λίστα με τον List να δείχνει στον πρώτο κόμβο.
  Λειτουργία: Εκτελεί μια γραμμική αναζήτηση στην μη ταξινομημένη συνδεδεμένη
              λίστα για έναν κόμβο που να περιέχει το στοιχείο Item.
  Επιστρέφει: Αν η αναζήτηση είναι επιτυχής η Found είναι true, ο CurrPtr δείχνει
                στον κόμβο που περιέχει το Item και ο PredPtr στον προηγούμενό του
                ή είναι nil αν δεν υπάρχει προηγούμενος.
                Αν η αναζήτηση δεν είναι επιτυχής η Found είναι false.
*/
{
   ListPointer CurrPtr;
   boolean stop;

   CurrPtr = List;
    *PredPtr=NULL;
   stop= FALSE;
   while (!stop && CurrPtr!=NULL )
    {
         if (CurrPtr->Data==Item )
                stop = TRUE;
         else
         {
                *PredPtr = CurrPtr;
            CurrPtr = CurrPtr->Next;
         }
        }
        *Found=stop;
```

```
}

void OrderedLinearSearch(ListPointer List, ListElementType Item, ListPointer *PredPtr,
boolean *Found)
/* Δέχεται:    Ένα στοιχείο Item και μια ταξινομημένη συνδεδεμένη λίστα,
                που περιέχει στοιχεία δεδομένων σε αύξουσα διάταξη και στην οποία
                ο δείκτης List δείχνει στον πρώτο  κόμβο.
   Λειτουργία: Εκτελεί γραμμική αναζήτηση της συνδεδεμένης ταξινομημένης λίστας
                για τον πρώτο κόμβο που περιέχει το στοιχείο Item ή για μια θέση
                για να εισάγει ένα νέο κόμβο που να περιέχει το στοιχείο Item.
   Επιστρέφει: Αν η αναζήτηση είναι επιτυχής η Found είναι true,
                ο CurrPtr δείχνει στον κόμβο που περιέχει το Item και
                ο PredPtr στον προηγούμενό του ή είναι nil αν δεν υπάρχει προηγούμενος.
                Αν η αναζήτηση δεν είναι επιτυχής η Found είναι false.
*/
{
   ListPointer CurrPtr;
   boolean DoneSearching;

   CurrPtr = List;
   *PredPtr = NULL;
   DoneSearching = FALSE;
   *Found = FALSE;
   while (!DoneSearching && CurrPtr!=NULL )
    {
         if (CurrPtr->Data>=Item )
         {
                DoneSearching = TRUE;
                *Found = (CurrPtr->Data==Item);
         }
         else
         {
                *PredPtr = CurrPtr;
           CurrPtr = CurrPtr->Next;
         }
        }
}
```

```
/*  Αρχείο: a2jf4.c
    Φοιτητής: Ευστάθιος Ιωσηφίδης


    Άδεια χρήσης: GNU General Public License v3.0

    Για τις παρακάτω λειτουργίες που αφορούν σε απλή συνδετική λίστα (Α.Σ.Λ.) με δείκτες
να γραφούν διαφορετικά
    προγράμματα για καθεμιά από αυτές. Γνωρίζουμε τα εξής:
    • Κάθε κόμβος της ΑΣΛ ή των ΑΣΛ περιέχει έναν ακέραιο αριθμό.
    • Η είσοδος των δεδομένων θα γίνεται ως εξής: Πρώτα θα διαβάζεται για κάθε ΑΣΛ το
πλήθος των κόμβων της Α.Σ.Λ.
    και στη συνέχεια τα στοιχεία της.
    • Σε όλες τις παρακάτω ασκήσεις η εισαγωγή στοιχείου, για την κατασκευή της
υπάρχουσας λίστας γίνεται στην
    αρχή της λίστας.
    • Σε κάθε πρόγραμμα θα εμφανίζονται οι κόμβοι της αρχικής ή των αρχικών λιστών και
στη συνέχεια αυτών που
    δημιουργούνται κατά την εκτέλεση του προγράμματος.
    • Η εμφάνιση των δεδομένων θα γίνεται ως εξής: Τα στοιχεία των κόμβων της Α.Σ.Λ. θα
εμφανίζονται σε μια γραμμή
    με ένα κενό χαρακτήρα μεταξύ τους.
    • Αν κατά τη διάσχιση της λίστας διαπιστώσετε ότι η Α.Σ.Λ. είναι κενή, τότε να
εμφανίζετε το μήνυμα ‘EMPTY LIST’.
    Σ’ όλα τα προγράμματα θα πρέπει να κατασκευάσετε την ΑΣΛ ή τις ΑΣΛς διαβάζοντας τα
δεδομένα ως εξής: πλήθος
    στοιχείων λίστας, στοιχεία λίστας ή αν πρόκειται για 2 λίστες τότε θα τα διαβάζετε ως
εξής: πλήθος στοιχείων λίστας1,
    στοιχεία λίστας1, πλήθος στοιχείων λίστας2, στοιχεία λίστας2
    Η καθεμία από τις παρακάτω λειτουργίες να υλοποιηθεί με τη χρήση συνάρτησης.

    (j) Να γραφεί συνάρτηση για την εισαγωγή m στοιχείων μετά το n-οστό στοιχείο της
Α.Σ.Λ.
    Το πρωτότυπο της συνάρτησης είναι void insert_list_m_elements(ListPointer *List, int
n);
    Το διάβασμα των δεδομένων θα γίνεται ως εξής: η Σ.Λ. (πλήθος στοιχείων λίστας,
στοιχεία λίστας), στη συνέχεια ο
    αριθμός n, το πλήθος m και τέλος τα m στοιχεία. Το πλήθος m και τα m στοιχεία θα
διαβάζονται μέσα στη
    insert_list_m_elements (). Ο έλεγχος της τιμής της n θα γίνεται μέσα στη συνάρτηση
insert_list_m_elements () και
    αν n > πλήθος των κόμβων της Σ.Λ., τότε θα εμφανίζεται το μήνυμα ‘ERROR’.

*/

#include <stdio.h>
#include <stdlib.h>

// Δήλωση τύπων

typedef int ListElementType;           /* ο τύπος των στοιχείων της συνδεδεμένης λίστας
                                        ενδεικτικά τύπου int */
typedef struct ListNode *ListPointer;   //ο τύπος των δεικτών για τους κόμβους
typedef struct ListNode
{
        ListElementType Data;
    ListPointer Next;
} ListNode;

typedef enum {
    FALSE, TRUE
} boolean;

// Δήλωση συναρτήσεων
void insert_list_m_elements(ListPointer *List, int n);
void CreateList(ListPointer *List);
boolean EmptyList(ListPointer List);
void LinkedInsert(ListPointer *List, ListElementType Item, ListPointer PredPtr);
```

```
void LinkedDelete(ListPointer *List, ListPointer PredPtr);
void LinkedTraverse(ListPointer List);
void LinearSearch(ListPointer List, ListElementType Item, ListPointer *PredPtr, boolean
*Found);
void OrderedLinearSearch(ListPointer List, ListElementType Item, ListPointer *PredPtr,
boolean *Found);

int main()
{
    ListPointer AList;
    ListElementType Item;
    int i,n;

    CreateList(&AList);

    // Εισαγωγή στοιχείων στις λίστες
    printf("DWSE TON PLH8OS TWN STOIXEIWN THS LISTAS: ");
    scanf("%d",&n);
    for(i=0;i<n;i++)
    {
        printf("DWSE TON ARI8MO GIA EISAGWGH STH ARXH THS LISTAS: ");
        scanf("%d",&Item);

        LinkedInsert(&AList, Item, NULL);

    }


    // Εμφάνιση στοιχείων των λιστών
    printf("*********Arxiki lista***********\n");
    LinkedTraverse(AList);
    printf("\n");

    printf("DWSE TI THESI, META APO TIN OPOIA THA EISAXTHOUN TA STOIXEIA: ");
    scanf("%d",&n);
    insert_list_m_elements(&AList,n);


    printf("*********Teliki lista***********\n");
    LinkedTraverse(AList);
    printf("\n");
    return 0;
}

// Υλοποίηση συναρτήσεων

void insert_list_m_elements(ListPointer *List, int n)
{
    ListPointer TempPtr;

    ListElementType i, j, m;

    if (EmptyList(*List))
    {
        printf ("EMPTY LIST");
    }
    else
    {
        TempPtr=*List;
        i = 1;

        while (TempPtr->Next!=NULL && i<n)
        {
            TempPtr = TempPtr->Next;
            i++;
        }
    }

    if (i<n || n<1)
```

```
    {
        printf ("ERROR");
    }
    else
    {
        printf ("DWSE TO PLH8OS TWN STOIXEIWN THS LISTAS: ");
        scanf("%d", &m);
        for (i=0; i<m; i++)
        {
            printf ("DWSE TON ARI8MO GIA EISAGWGH STHN ARXH THS LISTAS: ");
            scanf("%d", &j);

            LinkedInsert(List,j,TempPtr);
            TempPtr = TempPtr->Next;
        }
    }


}


void CreateList(ListPointer *List)
/* Λειτουργία: Δημιουργεί μια κενή συνδεδεμένη λίστα.
  Επιστρέφει:  Τον μηδενικό δείκτη List
*/
{
        *List = NULL;
}

boolean EmptyList(ListPointer List)
/* Δέχεται:   Μια συνδεδεμένη λίστα με τον List να δείχνει στον πρώτο κόμβο.
  Λειτουργία: Ελέγχει αν η συνδεδεμένη λίστα είναι κενή.
  Επιστρέφει: True αν η λίστα είναι κενή και false διαφορετικά
*/
{
        return (List==NULL);
}

void LinkedInsert(ListPointer *List, ListElementType Item, ListPointer PredPtr)
/* Δέχεται:    Μια συνδεδεμένη λίστα με τον List να δείχνει στον πρώτο κόμβο,
               ένα στοιχείο δεδομένων Item και έναν δείκτη PredPtr.
   Λειτουργία: Εισάγει έναν κόμβο, που περιέχει το Item, στην συνδεδεμένη λίστα
               μετά από τον κόμβο που δεικτοδοτείται από τον PredPtr
               ή στην αρχή  της συνδεδεμένης λίστας,
               αν ο PredPtr είναι μηδενικός(NULL).
  Επιστρέφει:  Την τροποποιημένη συνδεδεμένη λίστα με τον πρώτο κόμβο της
               να δεικτοδοτείται από τον List.
*/
{
        ListPointer TempPtr;

   TempPtr= (ListPointer)malloc(sizeof(struct ListNode));
 /*  printf("Insert &List %p, List %p,  &(*List) %p, (*List) %p, TempPtr %p\n",
   &List, List,  &(*List), (*List), TempPtr); */
   TempPtr->Data = Item;
        if (PredPtr==NULL) {
        TempPtr->Next = *List;
        *List = TempPtr;
    }
    else {
        TempPtr->Next = PredPtr->Next;
        PredPtr->Next = TempPtr;
    }
}

void LinkedDelete(ListPointer *List, ListPointer PredPtr)
/* Δέχεται:    Μια συνδεδεμένη λίστα με τον List να δείχνει στον πρώτο κόμβο της
                 και έναν δείκτη PredPtr.
   Λειτουργία: Διαγράφει από τη συνδεδεμένη λίστα τον κόμβο που έχει
```

```
                για προηγούμενό του αυτόν στον οποίο δείχνει ο PredPtr
                ή διαγράφει τον πρώτο κόμβο, αν ο PredPtr είναι μηδενικός,
                εκτός και αν η λίστα είναι κενή.
   Επιστρέφει: Την τροποποιημένη συνδεδεμένη λίστα με τον πρώτο κόμβο
                να δεικτοδοτείται από τον List.
   Έξοδος:     Ένα μήνυμα κενής λίστας αν η συνδεδεμένη λίστα ήταν κενή .
*/
{
    ListPointer TempPtr;

    if (EmptyList(*List))
          printf("EMPTY LIST\n");
   else
   {
          if (PredPtr == NULL)
        {
              TempPtr = *List;
              *List = TempPtr->Next;
        }
        else
        {
             TempPtr = PredPtr->Next;
             PredPtr->Next = TempPtr->Next;
        }
        free(TempPtr);
    }
}

void LinkedTraverse(ListPointer List)
/* Δέχεται:    Μια συνδεδεμένη λίστα με τον List να δείχνει στον πρώτο κόμβο.
   Λειτουργία: Διασχίζει τη συνδεδεμένη λίστα και
                επεξεργάζεται κάθε δεδομένο ακριβώς μια φορά.
   Επιστρέφει: Εξαρτάται από το είδος της επεξεργασίας.
*/
{
        ListPointer CurrPtr;

   if (EmptyList(List))
          printf("EMPTY LIST\n");
   else
   {
          CurrPtr = List;
         //   printf("%p\n",CurrPtr);
         //   printf("%-16s\t%-4s\t%-16s\n", "CurrPtr","Data","Next");
         while ( CurrPtr!=NULL )
        {
             //printf("%p\t%d\t%p\n",CurrPtr,(*CurrPtr).Data, (*CurrPtr).Next);
             printf("%d ",CurrPtr->Data);
             CurrPtr = CurrPtr->Next;
        }
   }
}

void LinearSearch(ListPointer List, ListElementType Item, ListPointer *PredPtr, boolean
*Found)
/* Δέχεται:   Μια συνδεδεμένη λίστα με τον List να δείχνει στον πρώτο κόμβο.
  Λειτουργία: Εκτελεί μια γραμμική αναζήτηση στην μη ταξινομημένη συνδεδεμένη
              λίστα για έναν κόμβο που να περιέχει το στοιχείο Item.
  Επιστρέφει: Αν η αναζήτηση είναι επιτυχής η Found είναι true, ο CurrPtr δείχνει
                στον κόμβο που περιέχει το Item και ο PredPtr στον προηγούμενό του
                ή είναι nil αν δεν υπάρχει προηγούμενος.
                Αν η αναζήτηση δεν είναι επιτυχής η Found είναι false.
*/
{
   ListPointer CurrPtr;
   boolean stop;

   CurrPtr = List;
    *PredPtr=NULL;
```

```
   stop= FALSE;
   while (!stop && CurrPtr!=NULL )
    {
         if (CurrPtr->Data==Item )
                stop = TRUE;
         else
         {
                *PredPtr = CurrPtr;
            CurrPtr = CurrPtr->Next;
         }
        }
        *Found=stop;
}

void OrderedLinearSearch(ListPointer List, ListElementType Item, ListPointer *PredPtr,
boolean *Found)
/* Δέχεται:    Ένα στοιχείο Item και μια ταξινομημένη συνδεδεμένη λίστα,
                που περιέχει στοιχεία δεδομένων σε αύξουσα διάταξη και στην οποία
                ο δείκτης List δείχνει στον πρώτο  κόμβο.
   Λειτουργία: Εκτελεί γραμμική αναζήτηση της συνδεδεμένης ταξινομημένης λίστας
                για τον πρώτο κόμβο που περιέχει το στοιχείο Item ή για μια θέση
                για να εισάγει ένα νέο κόμβο που να περιέχει το στοιχείο Item.
   Επιστρέφει: Αν η αναζήτηση είναι επιτυχής η Found είναι true,
                ο CurrPtr δείχνει στον κόμβο που περιέχει το Item και
                ο PredPtr στον προηγούμενό του ή είναι nil αν δεν υπάρχει προηγούμενος.
                Αν η αναζήτηση δεν είναι επιτυχής η Found είναι false.
*/
{
   ListPointer CurrPtr;
   boolean DoneSearching;

   CurrPtr = List;
   *PredPtr = NULL;
   DoneSearching = FALSE;
   *Found = FALSE;
   while (!DoneSearching && CurrPtr!=NULL )
    {
         if (CurrPtr->Data>=Item )
         {
                DoneSearching = TRUE;
                *Found = (CurrPtr->Data==Item);
         }
         else
         {
                *PredPtr = CurrPtr;
            CurrPtr = CurrPtr->Next;
         }
        }
}
```

```
/*  Αρχείο: a2rf4.c
    Φοιτητής: Ευστάθιος Ιωσηφίδης


    Άδεια χρήσης: GNU General Public License v3.0

    Για τις παρακάτω λειτουργίες που αφορούν σε απλή συνδετική λίστα (Α.Σ.Λ.) με δείκτες
να γραφούν διαφορετικά
    προγράμματα για καθεμιά από αυτές. Γνωρίζουμε τα εξής:
    • Κάθε κόμβος της ΑΣΛ ή των ΑΣΛ περιέχει έναν ακέραιο αριθμό.
    • Η είσοδος των δεδομένων θα γίνεται ως εξής: Πρώτα θα διαβάζεται για κάθε ΑΣΛ το
πλήθος των κόμβων της Α.Σ.Λ.
    και στη συνέχεια τα στοιχεία της.
    • Σε όλες τις παρακάτω ασκήσεις η εισαγωγή στοιχείου, για την κατασκευή της
υπάρχουσας λίστας γίνεται στην
    αρχή της λίστας.
    • Σε κάθε πρόγραμμα θα εμφανίζονται οι κόμβοι της αρχικής ή των αρχικών λιστών και
στη συνέχεια αυτών που
    δημιουργούνται κατά την εκτέλεση του προγράμματος.
    • Η εμφάνιση των δεδομένων θα γίνεται ως εξής: Τα στοιχεία των κόμβων της Α.Σ.Λ. θα
εμφανίζονται σε μια γραμμή
    με ένα κενό χαρακτήρα μεταξύ τους.
    • Αν κατά τη διάσχιση της λίστας διαπιστώσετε ότι η Α.Σ.Λ. είναι κενή, τότε να
εμφανίζετε το μήνυμα 'EMPTY LIST'.
    Σ' όλα τα προγράμματα θα πρέπει να κατασκευάσετε την ΑΣΛ ή τις ΑΣΛς διαβάζοντας τα
δεδομένα ως εξής: πλήθος
    στοιχείων λίστας, στοιχεία λίστας ή αν πρόκειται για 2 λίστες τότε θα τα διαβάζετε ως
εξής: πλήθος στοιχείων λίστας1,
    στοιχεία λίστας1, πλήθος στοιχείων λίστας2, στοιχεία λίστας2
    Η καθεμία από τις παρακάτω λειτουργίες να υλοποιηθεί με τη χρήση συνάρτησης.

    (r) Αφαίρεση ελάχιστων. Υλοποιήστε την παρακάτω συνάρτηση: ListElementType
RemoveMins(ListPointer *InList);
    η οποία δέχεται ως είσοδο μια λίστα ( InList ) και θα:
    • Βρίσκει τον ελάχιστο στοιχείο της λίστας και
    • αφαιρεί από τη λίστα όλους τους αριθμούς ίσους με τον ελάχιστο αριθμό και τον
επιστρέφει. Η εμφάνιση του
    ελάχιστου θα γίνεται στη main()

*/

#include <stdio.h>
#include <stdlib.h>

// Δήλωση τύπων

typedef int ListElementType;           /* ο τύπος των στοιχείων της συνδεδεμένης λίστας
                                        ενδεικτικά τύπου int */
typedef struct ListNode *ListPointer;   //ο τύπος των δεικτών για τους κόμβους
typedef struct ListNode
{
        ListElementType Data;
    ListPointer Next;
} ListNode;

typedef enum {
    FALSE, TRUE
} boolean;

// Δήλωση συναρτήσεων
ListElementType RemoveMins(ListPointer *InList);
void CreateList(ListPointer *List);
boolean EmptyList(ListPointer List);
void LinkedInsert(ListPointer *List, ListElementType Item, ListPointer PredPtr);
void LinkedDelete(ListPointer *List, ListPointer PredPtr);
void LinkedTraverse(ListPointer List);
void LinearSearch(ListPointer List, ListElementType Item, ListPointer *PredPtr, boolean
*Found);
```

```
void OrderedLinearSearch(ListPointer List, ListElementType Item, ListPointer *PredPtr,
boolean *Found);

int main()
{
    ListPointer AList;
    ListElementType Item,min;
    int i,n;

    CreateList(&AList);

    // Εισαγωγή στοιχείων στις λίστες
    printf("DWSE TON PLH8OS TWN STOIXEIWN THS LISTAS: ");
    scanf("%d",&n);
    for(i=0;i<n;i++)
    {
        printf("DWSE TON ARI8MO GIA EISAGWGH STH ARXH THS LISTAS 1: ");
        scanf("%d",&Item);

        LinkedInsert(&AList, Item, NULL);

    }

    // Εμφάνιση στοιχείων των λιστών
    printf("Traversing list\n");
    LinkedTraverse(AList);
    printf("\n");

    printf("Removing min from list\n");
    min=RemoveMins(&AList);
    printf("Min was %d\n",min);

    printf("Traversing list with min removed\n");
    LinkedTraverse(AList);
    printf("\n");


    return 0;
}

// Υλοποίηση συναρτήσεων

ListElementType RemoveMins(ListPointer *InList){

    ListPointer CurrPtr, PredPtr;
    ListElementType min;

    /*Εύρεση του ελάχιστου στοιχείου*/
    CurrPtr=*InList;
    min=CurrPtr->Data;

    while (CurrPtr != NULL)
    {
        if(CurrPtr->Data < min)
        {
            min=CurrPtr->Data;

        }
        CurrPtr=CurrPtr->Next;
    }

    /*Διάσχιση της ΣΛ και διαγραφή όλων των στοιχείων με τιμή ίση με το ελάχιστο*/

    CurrPtr=*InList;
    PredPtr=NULL;
    while (CurrPtr != NULL)
    {
        if(CurrPtr->Data == min)
        {
```

```
            CurrPtr=CurrPtr->Next;
            LinkedDelete(InList,PredPtr);
        }
        else{
            PredPtr=CurrPtr;
            CurrPtr=CurrPtr->Next;

        }
    }

    return min;
}

void CreateList(ListPointer *List)
/* Λειτουργία: Δημιουργεί μια κενή συνδεδεμένη λίστα.
  Επιστρέφει:  Τον μηδενικό δείκτη List
*/
{
        *List = NULL;
}

boolean EmptyList(ListPointer List)
/* Δέχεται:   Μια συνδεδεμένη λίστα με τον List να δείχνει στον πρώτο κόμβο.
  Λειτουργία: Ελέγχει αν η συνδεδεμένη λίστα είναι κενή.
  Επιστρέφει: True αν η λίστα είναι κενή και false διαφορετικά
*/
{
        return (List==NULL);
}

void LinkedInsert(ListPointer *List, ListElementType Item, ListPointer PredPtr)
/* Δέχεται:    Μια συνδεδεμένη λίστα με τον List να δείχνει στον πρώτο κόμβο,
                ένα στοιχείο δεδομένων Item και έναν δείκτη PredPtr.
   Λειτουργία: Εισάγει έναν κόμβο, που περιέχει το Item, στην συνδεδεμένη λίστα
                μετά από τον κόμβο που δεικτοδοτείται από τον PredPtr
                ή στην αρχή  της συνδεδεμένης λίστας,
                αν ο PredPtr είναι μηδενικός(NULL).
  Επιστρέφει:  Την τροποποιημένη συνδεδεμένη λίστα με τον πρώτο κόμβο της
                να δεικτοδοτείται από τον List.
*/
{
        ListPointer TempPtr;

   TempPtr= (ListPointer)malloc(sizeof(struct ListNode));
 /*  printf("Insert &List %p, List %p,  &(*List) %p, (*List) %p, TempPtr %p\n",
   &List, List,  &(*List), (*List), TempPtr); */
   TempPtr->Data = Item;
        if (PredPtr==NULL) {
        TempPtr->Next = *List;
        *List = TempPtr;
    }
    else {
        TempPtr->Next = PredPtr->Next;
        PredPtr->Next = TempPtr;
    }
}

void LinkedDelete(ListPointer *List, ListPointer PredPtr)
/* Δέχεται:    Μια συνδεδεμένη λίστα με τον List να δείχνει στον πρώτο κόμβο της
                 και έναν δείκτη PredPtr.
   Λειτουργία: Διαγράφει από τη συνδεδεμένη λίστα τον κόμβο που έχει
                για προηγούμενό του αυτόν στον οποίο δείχνει ο PredPtr
                ή διαγράφει τον πρώτο κόμβο, αν ο PredPtr είναι μηδενικός,
                εκτός και αν η λίστα είναι κενή.
   Επιστρέφει: Την τροποποιημένη συνδεδεμένη λίστα με τον πρώτο κόμβο
                να δεικτοδοτείται από τον List.
   Έξοδος:     Ένα μήνυμα κενής λίστας αν η συνδεδεμένη λίστα ήταν κενή .
*/
{
```

```
    ListPointer TempPtr;

    if (EmptyList(*List))
            printf("EMPTY LIST\n");
   else
   {
            if (PredPtr == NULL)
        {
              TempPtr = *List;
              *List = TempPtr->Next;
        }
        else
        {
             TempPtr = PredPtr->Next;
             PredPtr->Next = TempPtr->Next;
        }
        free(TempPtr);
    }
}

void LinkedTraverse(ListPointer List)
/* Δέχεται:    Μια συνδεδεμένη λίστα με τον List να δείχνει στον πρώτο κόμβο.
   Λειτουργία: Διασχίζει τη συνδεδεμένη λίστα και
                επεξεργάζεται κάθε δεδομένο ακριβώς μια φορά.
   Επιστρέφει: Εξαρτάται από το είδος της επεξεργασίας.
*/
{
        ListPointer CurrPtr;

   if (EmptyList(List))
            printf("EMPTY LIST\n");
   else
   {
            CurrPtr = List;
         //   printf("%p\n",CurrPtr);
         //   printf("%-16s\t%-4s\t%-16s\n", "CurrPtr","Data","Next");
         while ( CurrPtr!=NULL )
        {
             //printf("%p\t%d\t%p\n",CurrPtr,(*CurrPtr).Data, (*CurrPtr).Next);
             printf("%d ",CurrPtr->Data);
             CurrPtr = CurrPtr->Next;
        }
   }
}

void LinearSearch(ListPointer List, ListElementType Item, ListPointer *PredPtr, boolean
*Found)
/* Δέχεται:   Μια συνδεδεμένη λίστα με τον List να δείχνει στον πρώτο κόμβο.
  Λειτουργία: Εκτελεί μια γραμμική αναζήτηση στην μη ταξινομημένη συνδεδεμένη
              λίστα για έναν κόμβο που να περιέχει το στοιχείο Item.
  Επιστρέφει: Αν η αναζήτηση είναι επιτυχής η Found είναι true, ο CurrPtr δείχνει
                στον κόμβο που περιέχει το Item και ο PredPtr στον προηγούμενό του
                ή είναι nil αν δεν υπάρχει προηγούμενος.
                Αν η αναζήτηση δεν είναι επιτυχής η Found είναι false.
*/
{
   ListPointer CurrPtr;
   boolean stop;

   CurrPtr = List;
    *PredPtr=NULL;
   stop= FALSE;
   while (!stop && CurrPtr!=NULL )
    {
         if (CurrPtr->Data==Item )
                stop = TRUE;
         else
         {
                *PredPtr = CurrPtr;
```

```
            CurrPtr = CurrPtr->Next;
          }
         }
         *Found=stop;
}

void OrderedLinearSearch(ListPointer List, ListElementType Item, ListPointer *PredPtr,
boolean *Found)
/* Δέχεται:    Ένα στοιχείο Item και μια ταξινομημένη συνδεδεμένη λίστα,
                που περιέχει στοιχεία δεδομένων σε αύξουσα διάταξη και στην οποία
                ο δείκτης List δείχνει στον πρώτο  κόμβο.
   Λειτουργία: Εκτελεί γραμμική αναζήτηση της συνδεδεμένης ταξινομημένης λίστας
                για τον πρώτο κόμβο που περιέχει το στοιχείο Item ή για μια θέση
                για να εισάγει ένα νέο κόμβο που να περιέχει το στοιχείο Item.
   Επιστρέφει: Αν η αναζήτηση είναι επιτυχής η Found είναι true,
                ο CurrPtr δείχνει στον κόμβο που περιέχει το Item και
                ο PredPtr στον προηγούμενό του ή είναι nil αν δεν υπάρχει προηγούμενος.
                Αν η αναζήτηση δεν είναι επιτυχής η Found είναι false.
*/
{
   ListPointer CurrPtr;
   boolean DoneSearching;

   CurrPtr = List;
   *PredPtr = NULL;
   DoneSearching = FALSE;
   *Found = FALSE;
   while (!DoneSearching && CurrPtr!=NULL )
    {
          if (CurrPtr->Data>=Item )
          {
                 DoneSearching = TRUE;
                 *Found = (CurrPtr->Data==Item);
          }
          else
          {
                 *PredPtr = CurrPtr;
             CurrPtr = CurrPtr->Next;
          }
         }
}
```

```
/*  Αρχείο: a9f4.c
    Φοιτητής: Ευστάθιος Ιωσηφίδης


    Άδεια χρήσης: GNU General Public License v3.0

    Να γραφεί πρόγραμμα που να διαβάζει ένα αλφαριθμητικό και στη συνέχεια να προσθέτει
έναν-ένα τους χαρακτήρες του
    σε μια στοίβα και σε μια ουρά ταυτόχρονα. Μετά το τέλος της εισαγωγής των χαρακτήρων
του αλφαριθμητικού στη στοίβα
    και στην ουρά θα εμφανίζετε τα στοιχεία της στοίβας και της ουράς. Η εμφάνιση των
στοιχείων της στοίβας και ουράς θα
    γίνεται όπως περιγράφεται στις Παρατηρήσεις στο σημείο 2i. Στη συνέχεια θα ελέγχετε
αν το αλφαριθμητικό είναι
    καρκινικό, οπότε κι θα εμφανίζετε το μήνυμα 'ACCEPTED', ή όχι, οπότε θα εμφανίζετε το
μήνυμα 'REJECTED'
    (χρησιμοποιήστε τις διαδικασίες: εισαγωγή στοιχείου σε στοίβα, εισαγωγή στοιχείου σε
ουρά, εξαγωγή στοιχείου από
    στοίβα, εξαγωγή στοιχείου από ουρά με δυναμική υλοποίηση).


*/


#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include <string.h>

#define len 60

// Δήλωση τύπων
// ΟΥΡΑ
typedef char QueueElementType;           /*ο τύπος των στοιχείων της συνδεδεμένης ουράς
                                        ενδεικτικά τύπου char*/
typedef struct QueueNode *QueuePointer;

typedef struct QueueNode
{
        QueueElementType Data;
    QueuePointer Next;
} QueueNode;

typedef struct
{
    QueuePointer Front;
    QueuePointer Rear;
} QueueType;

// ΣΤΟΙΒΑ
typedef char StackElementType;           /*ο τύπος των στοιχείων της στοίβας
                                        ενδεικτικά τύπου char */
typedef struct StackNode *StackPointer;
typedef struct StackNode
{
        StackElementType Data;
    StackPointer Next;
} StackNode;

typedef enum {
    FALSE, TRUE
} boolean;

// Δήλωση συναρτήσεων
//ΟΥΡΑ
void CreateQ(QueueType *Queue);
boolean EmptyQ(QueueType Queue);
void AddQ(QueueType *Queue, QueueElementType Item);
```

```
void RemoveQ(QueueType *Queue, QueueElementType *Item);
void TraverseQ(QueueType Queue);

// ΣΤΟΙΒΑ
void CreateStack(StackPointer *Stack);
boolean EmptyStack(StackPointer Stack);
void Push(StackPointer *Stack, StackElementType Item);
void Pop(StackPointer *Stack, StackElementType *Item);
void TraverseStack(StackPointer Stack);

int main()
{
    char str[len];
    char ch1, ch2;
    int i=0,counter=0;
    QueueType iQueue;
    StackPointer iStack;
    boolean flag;


    CreateStack(&iStack);
    CreateQ(&iQueue);

    printf("DWSE TO ALFARITHMITIKO: ");
    gets(str); // Διαβάζω το αλφαριθμητικό

    for(i=0;i<strlen(str);i++) // Η strlen(str) επιστρέφει το μήκος του αλφαριθμητικού
    {
        Push(&iStack, str[i]); // Έναν-έναν χαρακτήρα του αλφαριθμητικού τον εισάγω στη
στοίβα
        AddQ(&iQueue, str[i]); // Έναν-έναν χαρακτήρα του αλφαριθμητικού τον εισάγω στην
ουρά

    }

    printf("-------Stack of characters-------\n");
    TraverseStack(iStack); // Εμφάνιση της στοίβας
    printf("-------Queue of characters-------\n");
    TraverseQ(iQueue); // Εμφάνιση της ουράς

    flag=TRUE; //το αλφαριθμητικό θεωρείται αρχικά ως καρκινικό
    while(flag && counter<strlen(str)) // Οσο flag & δεν ξεπέρασες το μήκος του
αλφαριθμητικού
    {
        RemoveQ(&iQueue, &ch2); //Αφαιρούμε ένα-ένα στοιχεία από στοίβα και ουρά
        Pop(&iStack, &ch1);

        if(ch1!=ch2) // και συγκρίνουμε αν είναι διαφορετικά, το αλφαριθμητικό δεν είναι
καρκινικό και σταματά η διαδικασία
        {
            flag=FALSE;
        }
        counter++; // Αύξησε τον μετρητή που παρακολουθεί το μήκος του αλφαριθμητικού
    }

    for(i=0;i<strlen(str);i++)
        printf("%c", str[i]);
    printf(" ");

    if(!flag) printf("REJECTED\n");  //να εμφανίζεται και το αλφαριθμητικό
    else printf("ACCEPTED\n");  //να εμφανίζεται και το αλφαριθμητικό

    return 0;
}

// Υλοποίηση συναρτήσεων
//ΟΥΡΑ
void CreateQ(QueueType *Queue)
/* Λειτουργία: Δημιουργεί μια κενή συνδεδεμένη ουρά.
```

```
   Επιστρέφει: Μια κενή συνδεδεμένη ουρά
*/
{
        Queue->Front = NULL;
        Queue->Rear = NULL;
}

boolean EmptyQ(QueueType Queue)
/* Δέχεται:    Μια συνδεδεμένη ουρά.
   Λειτουργία: Ελέγχει αν η συνδεδεμένη ουρά είναι κενή.
   Επιστρέφει: True αν η ουρά είναι κενή, false  διαφορετικά
*/
{
        return (Queue.Front==NULL);
}

void AddQ(QueueType *Queue, QueueElementType Item)
/* Δέχεται:    Μια συνδεδεμένη ουρά Queue και ένα  στοιχείο Item.
   Λειτουργία: Προσθέτει το στοιχείο Item στο τέλος της συνδεδεμένης ουράς Queue.
   Επιστρέφει: Την τροποποιημένη ουρά
*/
{
        QueuePointer TempPtr;

    TempPtr= (QueuePointer)malloc(sizeof(struct QueueNode));
    TempPtr->Data = Item;
    TempPtr->Next = NULL;
    if (Queue->Front==NULL)
        Queue->Front=TempPtr;
    else
        Queue->Rear->Next = TempPtr;
    Queue->Rear=TempPtr;
}

void RemoveQ(QueueType *Queue, QueueElementType *Item)
/* Δέχεται:    Μια συνδεδεμένη ουρά.
   Λειτουργία: Αφαιρεί το στοιχείο Item από την  κορυφή της συνδεδεμένης ουράς,
                αν δεν είναι  κενή.
   Επιστρέφει: Το στοιχείο Item και την τροποποιημένη συνδεδεμένη ουρά.
   Έξοδος:     Μήνυμα κενής ουράς, αν η ουρά είναι  κενή
*/
{
    QueuePointer TempPtr;

    if (EmptyQ(*Queue)) {
            printf("EMPTY Queue\n");
    }
   else
   {
        TempPtr = Queue->Front;
        *Item=TempPtr->Data;
        Queue->Front = Queue->Front->Next;
        free(TempPtr);
        if (Queue->Front==NULL) Queue->Rear=NULL;
    }
}

void TraverseStack(StackPointer Stack)
{
    StackPointer CurrPtr;

   if (EmptyStack(Stack))
   {
           printf("EMPTY Stack\n");
    }
   else
   {
           CurrPtr = Stack;
         while ( CurrPtr!=NULL )
```

```
        {
               printf("%c ", CurrPtr->Data);
             CurrPtr = CurrPtr->Next;
        }
   }
    printf("\n");
}

// ΣΤΟΙΒΑ
void CreateStack(StackPointer *Stack)
/* Λειτουργία: Δημιουργεί μια κενή συνδεδεμένη στοίβα.
   Επιστρέφει: Μια κενή συνδεδεμένη στοίβα, Stack
*/
{
        *Stack = NULL;
}

boolean EmptyStack(StackPointer Stack)
/* Δέχεται:     Μια συνδεδεμένη στοίβα, Stack.
   Λειτουργία:  Ελέγχει αν η Stack είναι κενή.
   Επιστρέφει:  TRUE αν η στοίβα είναι κενή, FALSE διαφορετικά
*/
{
        return (Stack==NULL);
}

void Push(StackPointer *Stack, StackElementType Item)
/* Δέχεται:    Μια συνδεδεμένη στοίβα που η κορυφή της δεικτοδοτείται από τον
                δείκτη Stack και ένα στοιχείο Item.
   Λειτουργία: Εισάγει στην κορυφή της συνδεδεμένης στοίβας, το στοιχείο Item.
   Επιστρέφει: Την τροποποιημένη συνδεδεμένη στοίβα
*/
{
        StackPointer TempPtr;

    TempPtr= (StackPointer)malloc(sizeof(struct StackNode));
    TempPtr->Data = Item;
    TempPtr->Next = *Stack;
    *Stack = TempPtr;
}

void Pop(StackPointer *Stack, StackElementType *Item)
/* Δέχεται:    Μια συνδεδεμένη στοίβα που η κορυφή της δεικτοδοτείται από τον δείκτη
Stack.
   Λειτουργία: Αφαιρεί από την κορυφή της συνδεδεμένης στοίβας,
                αν η στοίβα δεν είναι κενή, το στοιχείο Item.
   Επιστρέφει: Την τροποποιημένη συνδεδεμένη στοίβα και το στοιχείο Item.
   Έξοδος:     Μήνυμα κενής στοίβας, αν η συνδεδεμένη στοίβα είναι κενή
*/
{
    StackPointer TempPtr;

    if (EmptyStack(*Stack)) {
            printf("EMPTY Stack\n");
    }
   else
   {
        TempPtr = *Stack;
        *Item=TempPtr->Data;
        *Stack = TempPtr->Next;
        free(TempPtr);
    }
}

void TraverseQ(QueueType Queue)
{
    QueuePointer CurrPtr;

   if (EmptyQ(Queue))
```

```
    {
           printf("EMPTY Queue\n");
    }
   else
   {
           CurrPtr = Queue.Front;
         while ( CurrPtr!=NULL )
        {
               printf("%c ", CurrPtr->Data);
             CurrPtr = CurrPtr->Next;
        }
   }
    printf("\n");
}
```

```
/*  Αρχείο: a10f4.c
    Φοιτητής: Ευστάθιος Ιωσηφίδης


    Άδεια χρήσης: GNU General Public License v3.0

    Να γίνει πρόγραμμα που θα προσομοιώνει μια ουρά με τη βοήθεια δύο στοιβών, δηλαδή οι λειτουργίες της ουράς θα
    προσομοιώνονται με τις λειτουργίες της στοίβας. Αντί να χρησιμοποιήσετε μια ουρά αρκεί να χρησιμοποιήσετε 2 στοίβες.
    Κάθε κόμβος περιέχει έναν ακέραιο αριθμό και η εισαγωγή δεδομένων θα γίνεται ως εξής: πλήθος στοιχείων, στοιχεία. Το
    πρόγραμμα θα εμφανίζει τα περιεχόμενα και των 2 στοιβών. Η 2η στοίβα έχει καταχωρημένα τα στοιχεία όπως θα ήταν αν
    είχαμε χρησιμοποιήσει μια ουρά.


*/

#include <stdio.h>
#include <stdlib.h>

// Δήλωση τύπων
typedef int StackElementType;
typedef struct StackNode *StackPointer;
typedef struct StackNode
{
        StackElementType Data;
    StackPointer Next;
} StackNode;

typedef enum {
    FALSE, TRUE
} boolean;

// Δήλωση συναρτήσεων
void CreateStack(StackPointer *Stack);
boolean EmptyStack(StackPointer Stack);
void Push(StackPointer *Stack, StackElementType Item);
void Pop(StackPointer *Stack, StackElementType *Item);
void TraverseStack(StackPointer Stack);

int main()
{

    StackPointer AStack1, AStack2;
    StackElementType Item;
    int plithos,i;

    CreateStack(&AStack1);  //Δημιουργώ τη στοίβα AStack1
    CreateStack(&AStack2);  //Δημιουργώ τη στοίβα AStack2

    printf("PLITHOS STOIXEIWN: ");
    scanf("%d", &plithos);  //Διαβάζω το πλήθος των στοιχείων που θα εισαχθούν στη στοίβα
    for(i=0;i<plithos;i++)
    {
        printf("DWSE TO 1o STOIXEIO: ");
        scanf("%d", &Item); // Διαβάζω 1-1 στοιχείο και το εισάγω στην στοίβα AStack1
        Push(&AStack1, Item);
    }

    printf("*********1i stoiva**********\n");
    TraverseStack(AStack1);  //Εμφανίζω τα στοιχεία της στοίβας AStack1

    while(!EmptyStack(AStack1))
    {
        Pop(&AStack1, &Item);  //Απωθώ το κορυφαίο στοιχείο Item από τη στοίβα AStack1
        Push(&AStack2, Item);  //Εισάγω στο στοιχείο Item στη στοίβα AStack2
```

```
    }

    printf("*********2i stoiva**********\n");
    TraverseStack(AStack2);  //Εμφανίζω τα στοιχεία της στοίβας AStack2

    return 0;
}

// Υλοποίηση συναρτήσεων

void CreateStack(StackPointer *Stack)
/* Λειτουργία: Δημιουργεί μια κενή συνδεδεμένη στοίβα.
   Επιστρέφει: Μια κενή συνδεδεμένη στοίβα, Stack
*/
{
        *Stack = NULL;
}

boolean EmptyStack(StackPointer Stack)
/* Δέχεται:     Μια συνδεδεμένη στοίβα, Stack.
   Λειτουργία:  Ελέγχει αν η Stack είναι κενή.
   Επιστρέφει:  TRUE αν η στοίβα είναι κενή, FALSE διαφορετικά
*/
{
        return (Stack==NULL);
}

void Push(StackPointer *Stack, StackElementType Item)
/* Δέχεται:    Μια συνδεδεμένη στοίβα που η κορυφή της δεικτοδοτείται από τον
               δείκτη Stack και ένα στοιχείο Item.
   Λειτουργία: Εισάγει στην κορυφή της συνδεδεμένης στοίβας, το στοιχείο Item.
   Επιστρέφει: Την τροποποιημένη συνδεδεμένη στοίβα
*/
{
        StackPointer TempPtr;

    TempPtr= (StackPointer)malloc(sizeof(struct StackNode));
    TempPtr->Data = Item;
    TempPtr->Next = *Stack;
    *Stack = TempPtr;
}

void Pop(StackPointer *Stack, StackElementType *Item)
/* Δέχεται:    Μια συνδεδεμένη στοίβα που η κορυφή της δεικτοδοτείται από τον δείκτη
Stack.
   Λειτουργία: Αφαιρεί από την κορυφή της συνδεδεμένης στοίβας,
               αν η στοίβα δεν είναι κενή, το στοιχείο Item.
   Επιστρέφει: Την τροποποιημένη συνδεδεμένη στοίβα και το στοιχείο Item.
   Έξοδος:     Μήνυμα κενής στοίβας, αν η συνδεδεμένη στοίβα είναι κενή
*/
{
    StackPointer TempPtr;

    if (EmptyStack(*Stack)) {
            printf("EMPTY Stack\n");
    }
   else
   {
        TempPtr = *Stack;
        *Item=TempPtr->Data;
        *Stack = TempPtr->Next;
        free(TempPtr);
    }
}

void TraverseStack(StackPointer Stack)
{
    StackPointer CurrPtr;
```

```
   if (EmptyStack(Stack))
   {
           printf("EMPTY Stack\n");
    }
   else
   {
           CurrPtr = Stack;
         while ( CurrPtr!=NULL )
        {
               printf("%d ", CurrPtr->Data);
             CurrPtr = CurrPtr->Next;
        }
   }
    printf("\n");
}
```

```
/*  Αρχείο: a16f4.c
    Φοιτητής: Ευστάθιος Ιωσηφίδης


    Άδεια χρήσης: GNU General Public License v3.0

    Σε μία ουρά κάθε στοιχείο εισέρχεται σε κάποια συγκεκριμένη θέση σύμφωνα με το βαθμό προτεραιότητας (σε
    αύξουσα σειρά). Να γραφεί πρόγραμμα που θα περιλαμβάνει τα παρακάτω:
    - Εισαγωγή του πλήθους των κόμβων της ουράς.
    - Τη συνάρτηση void insert_prot(QueueType *Queue, QueueElementType Item) την εισαγωγή στοιχείων
    στην ουρά, κάθε κόμβος της οποίας θα περιέχει έναν τριψήφιο κωδικό αριθμό και τον βαθμό προτεραιότητας (1-20).
    Σε περίπτωση ίδιου βαθμού προτεραιότητας το στοιχείο εισέρχεται τελευταίο στην αντίστοιχη προτεραιότητα.
    - Τη συνάρτηση void TraverseQ(QueueType Queue) που θα εμφανίζει τα περιεχόμενα της ουράς κατά αύξοντα
    βαθμό προτεραιότητας. Τα στοιχεία κάθε κόμβου εμφανίζονται σε ξεχωριστή σειρά με ένα κενό μεταξύ τους και
    πρώτο το βαθμό προτεραιότητας.

*/

#include <stdio.h>
#include <stdlib.h>

// Δήλωση τύπων
typedef struct QueueNode *QueuePointer;

typedef struct
{
    int code;
    int prot;
} QueueElementType;

typedef struct QueueNode
{
        QueueElementType Data;
    QueuePointer Next;
} QueueNode;

typedef struct
{
    QueuePointer Front;
    QueuePointer Rear;
} QueueType;

typedef enum {
    FALSE, TRUE
} boolean;


// Δήλωση συναρτήσεων
void CreateQ(QueueType *Queue);
boolean EmptyQ(QueueType Queue);
void AddQ(QueueType *Queue, QueueElementType Item);
void RemoveQ(QueueType *Queue, QueueElementType *Item);
void TraverseQ(QueueType Queue);

void insert_prot(QueueType *Queue, QueueElementType Item);

int main()
{
    QueueType AQueue;
    QueueElementType AnItem;
    int plithos,i;
```

```
    CreateQ(&AQueue); // Δημιουργώ την ουρά AQueue

    printf("DWSE TO PLITHOS: ");
    scanf("%d", &plithos); // Διαβάζω το πλήθος των στοιχείων που θα εισαχθούν στην ουρά
    for(i=0;i<plithos;i++)
    {
        printf("DWSE TON KODIKO TOY %dou KOMVOU: ", i+1);
        scanf("%d", &AnItem.code); // Διαβάζω τον κωδικό του κόμβου
        printf("DWSE TO VATHMO PROTERAIOTITAS TOU %dou KOMVOU: ", i+1);
        scanf("%d", &AnItem.prot); // Διαβάζω τον βαθμό προτεραιότητας του κόμβου
        insert_prot(&AQueue, AnItem); // Εισάγω τον κόμβο στην ουρά
    }

    printf("----------Priority Queue-------------\n");
    TraverseQ(AQueue); // Εμφάνιση της ουράς

    return 0;
}

// Υλοποίηση συναρτήσεων

void insert_prot(QueueType *Queue, QueueElementType Item)
{
    QueuePointer TempPtr;
    QueuePointer CurrPtr;
    QueuePointer PredPtr;
    boolean flag;

    TempPtr=(QueuePointer)malloc(sizeof(struct QueueNode)); // Δέσμευση μνήμης για τον
νέο κόμβο
    TempPtr->Data=Item; // Ανάθεση τιμής στα δεδομένα (Data) του νέου κόμβου
    TempPtr->Next=NULL; // Αρχικοποίηση του δείκτη (Next) του νέου κόμβου

    if(EmptyQ(*Queue)) // Αν η ουρά είναι κενή
    {
        Queue->Front=TempPtr; // Ενημέρωση του δείκτη Front της ουράς στον νέο (και
μοναδικό) κόμβο
        Queue->Rear=TempPtr; // Ενημέρωση του δείκτη Rear της ουράς στον νέο (και
μοναδικό) κόμβο
    }
    else
    {
        PredPtr=Queue->Front; // Αρχικοποίηση του PredPtr στην αρχή της ουράς
        CurrPtr=Queue->Front; // Αρχικοποίηση του CurrPtr στην αρχή της ουράς
        flag=FALSE; // Αρχικοποίηση flag

        if(CurrPtr->Data.prot>Item.prot) /* Αν ο βαθμός προτεραιότητας του τρέχοντος
κόμβου (1ου κόμβου) είναι
                                            μεγαλύτερος από τον βαθμό του νέου αντικειμένου
*/
        //Ο κόμβος εισάγεται στην αρχή της ουράς
        {
            Queue->Front=TempPtr; // Ενημέρωση του δείκτη Front της ουράς στον νέο κόμβο
            TempPtr->Next=CurrPtr; // Ενημέρωση του δείκτη Next του νέου κόμβου στον
τρέχοντα κόμβο
        }
        else
        {
            while(CurrPtr!=NULL && flag==FALSE) // Όσο δεν έχουμε φτάσει στο τέλος της
ουράς και δεν έγινε η εισαγωγή
            {
                if(CurrPtr->Data.prot>Item.prot) /* Αν ο βαθμός προτεραιότητας του
τρέχοντος κόμβου είναι
                                                    μεγαλύτερος από τον βαθμό του νέου
αντικειμένου */
                {
                    PredPtr->Next=TempPtr; // Ενημέρωση του δείκτη Next του προηγούμενου
κόμβου στον νέο κόμβο
                    TempPtr->Next=CurrPtr; // Ενημέρωση του δείκτη Next του νέου κόμβου
```

```
στον τρέχοντα κόμβο
                    flag=TRUE; // Ενημέρωση του flag
                }
                PredPtr=CurrPtr; // Ενημέρωση του PredPtr στον τρέχοντα κόμβο της ουράς
                CurrPtr=CurrPtr->Next; // Ενημέρωση του CurrPtr στον επόμενο κόμβο της
ουράς
            }
            if(flag==FALSE) // Αν δεν έχει γίνει η εισαγωγή
            {
               PredPtr->Next=TempPtr; // Ενημέρωση του δείκτη Next του προηγούμενου
κόμβου στον νέο κόμβο
               Queue->Rear=TempPtr; // Ενημέρωση του δείκτη Rear στον νέο κόμβο
            }
        }
    }
}

void CreateQ(QueueType *Queue)
/* Λειτουργία: Δημιουργεί μια κενή συνδεδεμένη ουρά.
   Επιστρέφει: Μια κενή συνδεδεμένη ουρά
*/
{
        Queue->Front = NULL;
        Queue->Rear = NULL;
}

boolean EmptyQ(QueueType Queue)
/* Δέχεται:    Μια συνδεδεμένη ουρά.
   Λειτουργία: Ελέγχει αν η συνδεδεμένη ουρά είναι κενή.
   Επιστρέφει: True αν η ουρά είναι κενή, false  διαφορετικά
*/
{
        return (Queue.Front==NULL);
}

void AddQ(QueueType *Queue, QueueElementType Item)
/* Δέχεται:    Μια συνδεδεμένη ουρά Queue και ένα  στοιχείο Item.
   Λειτουργία: Προσθέτει το στοιχείο Item στο τέλος της συνδεδεμένης ουράς Queue.
   Επιστρέφει: Την τροποποιημένη ουρά
*/
{
        QueuePointer TempPtr;

    TempPtr= (QueuePointer)malloc(sizeof(struct QueueNode));
    TempPtr->Data = Item;
    TempPtr->Next = NULL;
    if (Queue->Front==NULL)
        Queue->Front=TempPtr;
    else
        Queue->Rear->Next = TempPtr;
    Queue->Rear=TempPtr;
}

void RemoveQ(QueueType *Queue, QueueElementType *Item)
/* Δέχεται:    Μια συνδεδεμένη ουρά.
   Λειτουργία: Αφαιρεί το στοιχείο Item από την  κορυφή της συνδεδεμένης ουράς,
                αν δεν είναι  κενή.
   Επιστρέφει: Το στοιχείο Item και την τροποποιημένη συνδεδεμένη ουρά.
   Έξοδος:     Μήνυμα κενής ουράς, αν η ουρά είναι  κενή
*/
{
    QueuePointer TempPtr;

    if (EmptyQ(*Queue)) {
            printf("EMPTY Queue\n");
    }
   else
   {
        TempPtr = Queue->Front;
```

```
        *Item=TempPtr->Data;
        Queue->Front = Queue->Front->Next;
        free(TempPtr);
        if (Queue->Front==NULL) Queue->Rear=NULL;
    }
}

void TraverseQ(QueueType Queue)
{
    QueuePointer CurrPtr;

   if (EmptyQ(Queue))
    {
           printf("EMPTY Queue\n");
    }
   else
   {
           CurrPtr = Queue.Front;
         while ( CurrPtr!=NULL )
        {
               printf("%d %d\n", CurrPtr->Data.prot, CurrPtr->Data.code); // Αλλαγή στον
τρόπο εμφάνισης
             CurrPtr = CurrPtr->Next;
        }
   }
    printf("\n");
}
```

```
/*  Αρχείο: a11f5.c
    Φοιτητής: Ευστάθιος Ιωσηφίδης


    Άδεια χρήσης: GNU General Public License v3.0

    Γράψτε ένα πρόγραμμα που θα πραγματοποιεί ορθογραφικό έλεγχο ενός κειμένου. Οι λέξεις
που απαρτίζουν το λεξικό
    είναι αποθηκευμένες στο αρχείο κειμένου ‘I112f5.TXT’ (σε κάθε γραμμή του υπάρχει μία
λέξη). Διαβάζονται και
    αποθηκεύονται μία-μία σε ένα ΔΔΑ με τη συνάρτηση CreateDictionary και έτσι
δημιουργείται το λεξικό-ΔΔΑ. Μετά τη
    δημιουργία του λεξικού-ΔΔΑ θα εμφανίζετε τις λέξεις του λεξικού. Στη συνέχεια το
πρόγραμμα μέσω της συνάρτησης
    SpellingCheck θα διαβάζει ένα κείμενο που είναι αποθηκευμένο στο αρχείο κειμένου
‘I111F5.TXT’, σε κάθε γραμμή του
    οποίου υπάρχει μία λέξη και θα διενεργεί ορθογραφικό έλεγχο, δηλαδή θα αναζητά τη
λέξη του αρχείου ‘I111F5.TXT’ στο
    λεξικό-ΔΔΑ. Κατά τον ορθογραφικό έλεγχο θα πρέπει να εκτυπώνετε τις λέξεις που δεν
βρέθηκαν στο λεξικό και να
    υπολογίζετε το πλήθος τους. Το πλήθος των λέξεων που δεν βρέθηκαν στο λεξικό θα
εμφανίζεται στη main.
    Δίνονται τα πρωτότυπα των συναρτήσεων
    void CreateDictionary(BinTreePointer *Root, FILE *fp);
    int SpellingCheck(BinTreePointer Root, FILE *fp);

    Η 1 παράμετρος της fopen θα πρέπει να περιλαμβάνει το όνομα του αρχείου χωρίς
διαδρομή δηλαδή:
    fp1 = fopen("i112f5.txt", "r");
    fp2 = fopen("i111f5.txt", "r");


*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define size 20

// Δήλωση τύπων
typedef char BinTreeElementType[size];

typedef struct BinTreeNode *BinTreePointer;
struct BinTreeNode {
    BinTreeElementType Data;
    BinTreePointer LChild, RChild;
} ;

typedef enum {
    FALSE, TRUE
} boolean;

// Δήλωση συναρτήσεων
void CreateBST(BinTreePointer *Root);
boolean EmptyBST(BinTreePointer Root);
void BSTInsert(BinTreePointer *Root, BinTreeElementType Item);
void BSTSearch(BinTreePointer Root, BinTreeElementType KeyValue, boolean *Found,
BinTreePointer *LocPtr);

void InorderTraversal(BinTreePointer Root);

void CreateDictionary(BinTreePointer *Root, FILE *fp);
int SpellingCheck(BinTreePointer Root, FILE *fp);

int main ()
{
    // Δήλωση μεταβλητών
```

```
    BinTreePointer ARoot;

    FILE *fp1;
    FILE *fp2;

    // Άνοιγμα αρχείων
    fp1 = fopen("i112f5.txt","r");
    fp2 = fopen("i111f5.txt","r");

    CreateBST(&ARoot); // Δημιουργία κενού ΔΔΑ
    CreateDictionary(&ARoot,fp1); // Κατασκευή λεξικού

    // Εμφάνιση του λεξικού
    printf("**********Dictionary**********\n");
    InorderTraversal(ARoot);
    printf("\n");

    // Εμφάνιση των λέξεων που δεν είναι στο λεξικό (δηλαδή στο ΔΔΑ)
    printf("**********Not in Dictionary**********\n");
    printf("Number of words not in Dictionary: %d\n",SpellingCheck(ARoot, fp2));

    // Κλείσιμο αρχείων
    fclose(fp1);
    fclose(fp2);

    return 0;
}


// Υλοποίηση συναρτήσεων
void CreateBST(BinTreePointer *Root)
/* Λειτουργία: Δημιουργεί ένα κενό ΔΔΑ.
   Επιστρέφει:  Τον μηδενικό δείκτη(NULL) Root
*/
{
    *Root = NULL;
}

boolean EmptyBST(BinTreePointer Root)
/* Δέχεται:   Ενα ΔΔα με το Root να δείχνει στη ρίζα του.
  Λειτουργία: Ελέγχει αν το ΔΔΑ είναι κενό.
  Επιστρέφει: TRUE αν το ΔΔΑ είναι κενό και FALSE διαφορετικά
*/
{
    return (Root==NULL);
}

void BSTInsert(BinTreePointer *Root, BinTreeElementType Item)
/* Δέχεται:    Ένα ΔΔΑ με το δείκτη Root να δείχνει στη ρίζα του και ένα στοιχείο Item.
   Λειτουργία:  Εισάγει το στοιχείο Item στο ΔΔΑ.
   Επιστρέφει: Το τροποποιημένο ΔΔΑ με τον δείκτη Root να δείχνει στη ρίζα του
*/
{
    BinTreePointer LocPtr, Parent;
    boolean Found;

    LocPtr = *Root;
    Parent = NULL;
    Found = FALSE;
    while (!Found && LocPtr != NULL) {
        Parent = LocPtr;
        if (strcmp(Item, LocPtr->Data) < 0)
            LocPtr = LocPtr ->LChild;
        else if (strcmp(Item, LocPtr->Data) > 0)
            LocPtr = LocPtr ->RChild;
        else
            Found = TRUE;
    }
    if (Found)
```

```
        printf("To %s EINAI HDH STO DDA\n", Item);
    else {
        LocPtr = (BinTreePointer)malloc(sizeof (struct BinTreeNode));
        //LocPtr ->Data = Item;//kati edo
        strcpy(LocPtr ->Data,Item);
        LocPtr ->LChild = NULL;
        LocPtr ->RChild = NULL;
        if (Parent == NULL)
            *Root = LocPtr;
        else if (strcmp(Item,Parent ->Data) < 0)
            Parent ->LChild = LocPtr;
        else
            Parent ->RChild = LocPtr;
    }
}

void BSTSearch(BinTreePointer Root, BinTreeElementType KeyValue, boolean *Found,
BinTreePointer *LocPtr)
/* Δέχεται:    Ένα ΔΔΑ με το δείκτη Root να δείχνει στη ρίζα του και μια τιμή KeyValue.
   Λειτουργία: Αναζητά στο ΔΔΑ έναν κόμβο με τιμή KeyValue στο πεδίο κλειδί του.
   Επιστρέφει: Η Found έχει τιμή TRUE και ο δείκτης LocPtr δείχνει στον κόμβο που
               περιέχει την τιμή KeyValue, αν η αναζήτηση είναι επιτυχής.
               Διαφορετικά η Found έχει τιμή FALSE
*/
{
    (*LocPtr) = Root;
    (*Found) = FALSE;
    while (!(*Found) && (*LocPtr) != NULL)
    {
        if (strcmp(KeyValue,(*LocPtr)->Data) < 0)
            (*LocPtr) = (*LocPtr)->LChild;
        else
            if (strcmp(KeyValue,(*LocPtr)->Data) > 0)
                (*LocPtr) = (*LocPtr)->RChild;
            else (*Found) = TRUE;
    }
}

void InorderTraversal(BinTreePointer Root)
/* Δέχεται:    Ένα δυαδικό δέντρο με το δείκτη Root να δείχνει στην ρίζα του.
   Λειτουργία: Εκτελεί ενδοδιατεταγμένη διάσχιση του δυαδικού δέντρου και
               επεξεργάζεται κάθε κόμβο ακριβώς μια φορά.
   Εμφανίζει: Το περιεχόμενο του κόμβου, και εξαρτάται από το είδος της επεξεργασίας
*/
{
    if (Root!=NULL) {
        InorderTraversal(Root->LChild);
        printf("%s\n",Root->Data);
        InorderTraversal(Root->RChild);
    }
}

void CreateDictionary(BinTreePointer *Root, FILE *fp)
{
    BinTreeElementType Item; // Δήλωση μεταβλητών

    while(!feof(fp)){ // Όσο το αρχείο δεν τελείωσε

        fscanf(fp,"%s", Item); // Διάβασε την λέξη από το αρχείο
        BSTInsert(&(*Root), Item); // Εισήγαγε την λέξη στο ΔΔΑ-λεξικό
    }
}

int SpellingCheck(BinTreePointer Root, FILE *fp)
{
    // Δήλωση μεταβλητών
    boolean Found;
    BinTreePointer LocPtr;
    BinTreeElementType Item;
```

```
    int count = 0; // Αρχικοποίηση μετρητή λέξεων

    // διαβάζω 1-1 λέξη του αρχείου και αναζητώ στο ΔΔΑ-λεξικό
    // αν δεν υπάρχει στο ΔΔΑ-λεξικό τότε την εμφανίζω

    while(!feof(fp)){ // Όσο το αρχείο δεν τελείωσε

        fscanf(fp,"%s", Item); // Διάβασε την λέξη από το αρχείο
        BSTSearch(Root, Item, &Found, &LocPtr); // Αναζήτησε τη λέξη στο ΔΔΑ-λεξικό

        if(!Found){ // Αν η λέξη δεν βρέθηκε στο ΔΔΑ-λεξικό
            printf("%s \n",Item); // Εμφάνισε την λέξη
            count++; // Αύξησε τον μετρητή
        }
    }

    return count; // Επέστρεψε τον μετρητή
}
```

```
/*  Αρχείο: a29f5.c
    Φοιτητής: Ευστάθιος Ιωσηφίδης


    Άδεια χρήσης: GNU General Public License v3.0

    Γράψτε πρόγραμμα που θα δέχεται για κάθε άτομο τον ΑΜΚΑ (ακέραιος), τον ΑΦΜ
(ακέραιος), την ηλικία (ακέραιος). Θα
    καταχωρεί τα στοιχεία του κάθε ατόμου σε 2 καταλόγους ανάλογα με την ηλικία του,
αυτούς με ηλικία μικρότερη ή ίση
    των 60 ετών και αυτούς με ηλικία μεγαλύτερη των 60.
    Κάθε κατάλογος θα πρέπει να οργανωθεί ως ΔΔΑ με κλειδί τον ΑΜΚΑ. Το πρόγραμμα θα
περιλαμβάνει τις εξής
    λειτουργίες
    - Εισαγωγή των στοιχείων του κάθε ατόμου στο αντίστοιχο ΔΔΑ ανάλογα με την ηλικία του
(ΔΔΑ για άτομα με ηλικίες
    <=60 και ΔΔΑ για άτομα με ηλικίες > 60). Και στα 2 ΔΔΑ το κλειδί θα είναι ο ΑΜΚΑ.
    - Εμφάνιση των 2 καταλόγων
    - Αναζήτηση ατόμου με βάση τον ΑΜΚΑ και την ηλικία.
    Δίνεται ένα στιγμιότυπο εκτέλεσης όπου φαίνεται πως θα γίνεται το διάβασμα των
δεδομένων και η εμφάνιση του
    κάθε καταλόγου. Η αναζήτηση θα γίνει για 3 άτομα: α) ένα άτομο του οποίου τα στοιχεία
έχουν καταχωρηθεί σε ένα εκ
    των 2 καταλόγων (ΑΜΚΑ, ηλικία), β) ένα άτομο του οποίου το ΑΜΚΑ του έχει καταχωρηθεί
στον αντίστοιχο κατάλογο
    αλλά η ηλικία του δεν ταυτίζεται με τη δοθείσα και γ) ένα άτομο του οποίου το ΑΜΚΑ
δεν έχει καταχωρηθεί σε κανένα
    κατάλογο. Τα μηνύματα της κάθε περίπτωσης φαίνονται στο στιγμιότυπο εκτέλεσης.
    (για την απλοποίηση της εισαγωγής δεδομένων δόθηκαν ίδιες τιμές για ΑΜΚΑ, ΑΦΜ,
ηλικία)


*/


#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>

// Δήλωση τύπων
typedef struct{
    int AMKA;
    int afm;
    int age;
} BinTreeElementType;

typedef struct BinTreeNode *BinTreePointer;
 struct BinTreeNode {
    BinTreeElementType Data;
    BinTreePointer LChild, RChild;
} ;

typedef enum {
    FALSE, TRUE
} boolean;

// Δήλωση συναρτήσεων
void CreateBST(BinTreePointer *Root);
boolean EmptyBST(BinTreePointer Root);
void BSTInsert(BinTreePointer *Root, BinTreeElementType Item);
void BSTSearch(BinTreePointer Root, BinTreeElementType KeyValue, boolean *Found,
BinTreePointer *LocPtr);
void InorderTraversal(BinTreePointer Root);

int main()
{
    // Δήλωση μεταβλητών
    BinTreePointer Root1, Root2, LocPtr;
```

```
    BinTreeElementType person;
    boolean found;
    char ch;
    int i;

    CreateBST(&Root1); // Δημιουργία ΔΔΑ για τον κατάλογο <=60
    CreateBST(&Root2); // Δημιουργία ΔΔΑ για τον κατάλογο >60

    do // Επαναληπτικά
    {
     printf("Give AMKA? ");
     scanf("%d", &person.AMKA); // Διάβασε το ΑΜΚΑ του ατόμου
     printf("Give AFM? ");
     scanf("%d", &person.afm); // Διάβασε το ΑΦΜ του ατόμου
     printf("Give age? ");
     scanf("%d", &person.age); // Διάβασε την ηλικία του ατόμου

     if(person.AMKA<=60){ // Αν η ηλικία του ατόμου είναι μικρότερη ή ίση του 60
        BSTInsert(&Root1, person); // Εισαγωγή του ατόμου στο ΔΔΑ με τους <=60
     }
     else{ // Αλλιώς
        BSTInsert(&Root2, person); // Εισαγωγή του ατόμου στο ΔΔΑ με τους >60
     }
     printf("\nContinue Y/N: ");

     do // Επαναληπτικά
       {
       scanf("%c", &ch); // Διάβασε τον χαρακτήρα ch
       } while (toupper(ch)!= 'N' && toupper(ch)!= 'Y'); // όσο ο χαρακτήρας δεν είναι N
ή Y

       } while (toupper(ch)!='N'); // όσο ο χαρακτήρας δεν είναι N

    printf("\nPeople with age less-equal 60\n");
    InorderTraversal(Root1); // Εμφάνισε το ΔΔΑ με τα άτομα <=60
    printf("\n");
    printf("\nPeople with age greater than 60\n");
    InorderTraversal(Root2); // Εμφάνισε το ΔΔΑ με τα άτομα > 60
    printf("\n");
    for(i=0;i<3;i++)
    {
     printf("Give AMKA: ");
     scanf("%d", &person.AMKA); // Διάβασε το ΑΜΚΑ του ατόμου
     printf("Give age: ");
     scanf("%d", &person.age); // Διάβασε την ηλικία του ατόμου

    if(person.age<=60) // Αν η ηλικία του ατόμου είναι μικρότερη ή ίση του 60
    {
        BSTSearch(Root1, person, &found, &LocPtr); // Αναζήτηση του ατόμου στο ΔΔΑ των
<=60
    }
    else // Αλλιώς
    {
        BSTSearch(Root2, person, &found, &LocPtr); // Αναζήτηση του ατόμου στο ΔΔΑ των >
60
    }
    if(found) // Αν το άτομο βρέθηκε
    {
        if(person.age==LocPtr->Data.age) // Αν η ηλικία του ατόμου είναι ίση με τη
δοσμένη από το χρήστη ηλικία
        {   // Εμφάνισε ότι το άτομο βρέθηκε
            printf("The person with AMKA %d, AFM %d and age %d is in the
catalogue\n",LocPtr->Data.AMKA,LocPtr->Data.afm,LocPtr->Data.age);
        }
        else{ // Αλλιώς Εμφάνισε ότι το άτομο βρέθηκε αλλά με άλλη ηλικία
            printf("The person with AMKA %d has age %d different from the person you are
looking for\n",LocPtr->Data.AMKA,LocPtr->Data.age);
        }
    }
```

```
    else{ // Αλλιώς Εμφάνισε ότι το άτομο δεν βρέθηκε
        printf("The person with AMKA %d and age %d is Unknown\n",person.AMKA,person.age);
    }
    }


    return 0;
}

// Υλοποίηση συναρτήσεων
void CreateBST(BinTreePointer *Root)
/* Λειτουργία: Δημιουργεί ένα κενό ΔΔΑ.
   Επιστρέφει:  Τον μηδενικό δείκτη(NULL) Root
*/
{
    *Root = NULL;
}

boolean EmptyBST(BinTreePointer Root)
/* Δέχεται:   Ενα ΔΔα με το Root να δείχνει στη ρίζα του.
  Λειτουργία: Ελέγχει αν το ΔΔΑ είναι κενό.
  Επιστρέφει: TRUE αν το ΔΔΑ είναι κενό και FALSE διαφορετικά
*/
{   return (Root==NULL);
}

void BSTInsert(BinTreePointer *Root, BinTreeElementType Item)
/* Δέχεται:     Ένα ΔΔΑ με το δείκτη Root να δείχνει στη ρίζα του και ένα στοιχείο Item.
   Λειτουργία:  Εισάγει το στοιχείο Item στο ΔΔΑ.
   Επιστρέφει: Το τροποποιημένο ΔΔΑ με τον δείκτη Root να δείχνει στη ρίζα του
*/
{
    BinTreePointer LocPtr, Parent;
    boolean Found;

    LocPtr = *Root;
    Parent = NULL;
    Found = FALSE;
    while (!Found && LocPtr != NULL) {
        Parent = LocPtr;
        if (Item.AMKA < LocPtr->Data.AMKA)
            LocPtr = LocPtr ->LChild;
        else if (Item.AMKA > LocPtr ->Data.AMKA)
            LocPtr = LocPtr ->RChild;
        else
            Found = TRUE;
    }
    if (Found)
        printf("To %d EINAI HDH STO DDA\n", Item.AMKA);
    else {
        LocPtr = (BinTreePointer)malloc(sizeof (struct BinTreeNode));
        LocPtr ->Data = Item;
        LocPtr ->LChild = NULL;
        LocPtr ->RChild = NULL;
        if (Parent == NULL)
            *Root = LocPtr;
        else if (Item.AMKA < Parent ->Data.AMKA)
            Parent ->LChild = LocPtr;
        else
            Parent ->RChild = LocPtr;
    }
}


void BSTSearch(BinTreePointer Root, BinTreeElementType KeyValue, boolean *Found,
BinTreePointer *LocPtr)
/* Δέχεται:    Ένα ΔΔΑ με το δείκτη Root να δείχνει στη ρίζα του και μια τιμή KeyValue.
   Λειτουργία: Αναζητά στο ΔΔΑ έναν κόμβο με τιμή KeyValue στο πεδίο κλειδί του.
```

```
   Επιστρέφει: Η Found έχει τιμή TRUE και ο δείκτης LocPtr δείχνει στον κόμβο που
               περιέχει την τιμή KeyValue, αν η αναζήτηση είναι επιτυχής.
               Διαφορετικά η Found έχει τιμή FALSE
*/
{
    boolean stop;

    *LocPtr = Root;
    stop = FALSE;
    while (!stop && *LocPtr != NULL)
    {
        if (KeyValue.AMKA < (*LocPtr)->Data.AMKA)
            *LocPtr = (*LocPtr)->LChild;
        else
            if (KeyValue.AMKA > (*LocPtr)->Data.AMKA)
                *LocPtr = (*LocPtr)->RChild;
            else stop = TRUE;
    }
   *Found=stop;
}

void InorderTraversal(BinTreePointer Root)
/* Δέχεται:    Ένα δυαδικό δέντρο με το δείκτη Root να δείχνει στην ρίζα του.
   Λειτουργία: Εκτελεί ενδοδιατεταγμένη διάσχιση του δυαδικού δέντρου και
               επεξεργάζεται κάθε κόμβο ακριβώς μια φορά.
   Εμφανίζει: Το περιεχόμενο του κόμβου, και εξαρτάται από το είδος της επεξεργασίας
*/
{
    if (Root!=NULL) {
        InorderTraversal(Root->LChild);
        printf("(%d, %d, %d)",Root->Data.AMKA,Root->Data.afm,Root->Data.age);
        InorderTraversal(Root->RChild);
    }
}
```

```
/*  Αρχείο: a25f5.c
    Φοιτητής: Ευστάθιος Ιωσηφίδης


    Άδεια χρήσης: GNU General Public License v3.0

    Υλοποιείστε την αναδρομική συνάρτηση int CountLeaves(BinTreePointer Root) η οποία θα
υπολογίζει το πλήθος των
    φύλλων ενός ΔΔΑ το οποίο και θα επιστρέφει. Στο ΔΔΑ θα καταχωρούνται ακέραιοι. Θα
εμφανίζονται οι κόμβοι του ΔΔΑ
    σε αύξουσα διάταξη και στη συνέχεια θα εμφανίζεται το πλήθος των φύλλων του ΔΔΑ στο
κυρίως πρόγραμμα

*/

#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>

// Δήλωση τύπων
typedef int BinTreeElementType;             /*ο τύπος των στοιχείων του ΔΔΑ
                                              ενδεικτικά τύπου int */
typedef struct BinTreeNode *BinTreePointer;
typedef struct BinTreeNode {
    BinTreeElementType Data;
    BinTreePointer LChild, RChild;
} BinTreeNode;

typedef enum {
    FALSE, TRUE
} boolean;

// Δήλωση συναρτήσεων
void CreateBST(BinTreePointer *Root);
boolean BSTEmpty(BinTreePointer Root);
void RecBSTInsert(BinTreePointer *Root, BinTreeElementType Item);
void RecBSTSearch(BinTreePointer Root, BinTreeElementType KeyValue, boolean *Found,
BinTreePointer *LocPtr);
void RecBSTDelete(BinTreePointer *Root, BinTreeElementType KeyValue);
void RecBSTInorder(BinTreePointer Root);
void RecBSTPreorder(BinTreePointer Root);
void RecBSTPostorder(BinTreePointer Root);
int CountLeaves(BinTreePointer Root);


int main()
{
    // Δήλωση μεταβλητών
    BinTreePointer ARoot;
    BinTreeElementType Item;
    char ch;
    int fylla;

    CreateBST(&ARoot); // Δημιουργία ΔΔΑ

    do // Επαναληπτικά
    {
     printf("Enter a number for insertion in the Tree: ");
     scanf("%d", &Item); // Διάβασε τον αριθμό
     RecBSTInsert(&ARoot,Item); // Εισαγωγή του αριθμού στο ΔΔΑ


     printf("Continue Y/N: "); // Ερώτηση για εισαγωγή και άλλων αριθμών

     do // Επαναληπτικά
       {
       scanf("%c", &ch); // Διάβασε τον χαρακτήρα ch
       } while (toupper(ch)!= 'N' && toupper(ch)!= 'Y'); // όσο ο χαρακτήρας δεν είναι N
```

```
ή Y

       } while (toupper(ch)!='N'); // όσο ο χαρακτήρας δεν είναι N

    printf("Elements of BST\n");
    RecBSTInorder(ARoot); // Εμφάνιση των στοιχείων του ΔΔΑ (σε αύξουσα διάταξη)

    fylla=CountLeaves(ARoot); // Καλούμε την συνάρτηση CountLeaves
    printf("\nNumber of leaves %d\n",fylla); // Εμφάνιση του πλήθους των φύλλων
    return 0;
}

// Συναρτήσεις

int CountLeaves(BinTreePointer Root)
{


    if(Root==NULL) // Αν το ΔΔΑ είναι κενό
        {return 0;} // Επέστρεψε 0
    if (Root->LChild==NULL && Root->RChild==NULL) // Αν ο κόμβος είναι φύλλο
        {return 1;} // Επέστρεψε 1

     // Επέστρεψε το άθροισμα της Countleaves για το αριστερό και το δεξί υποδέντρο
     return CountLeaves(Root->LChild)+CountLeaves(Root->RChild);
}



void CreateBST(BinTreePointer *Root)
/* Λειτουργία: Δημιουργεί ένα κενό ΔΔΑ.
   Επιστρέφει:  Τον μηδενικό δείκτη(NULL) Root
*/
{
    *Root = NULL;
}

boolean BSTEmpty(BinTreePointer Root)
/* Δέχεται:   Ενα ΔΔα με το Root να δείχνει στη ρίζα του.
  Λειτουργία: Ελέγχει αν το ΔΔΑ είναι κενό.
  Επιστρέφει: TRUE αν το ΔΔΑ είναι κενό και FALSE διαφορετικά
*/
{
    return (Root==NULL);
}

void RecBSTInsert(BinTreePointer *Root, BinTreeElementType Item)
/* Δέχεται:     Ένα ΔΔΑ με το δείκτη Root να δείχνει στη ρίζα του και ένα στοιχείο Item.
   Λειτουργία:  Εισάγει το στοιχείο Item στο ΔΔΑ.
   Επιστρέφει: Το τροποποιημένο ΔΔΑ με τον δείκτη Root να δείχνει στη ρίζα του
*/
{
    if (BSTEmpty(*Root)) {
        (*Root) = (BinTreePointer)malloc(sizeof (struct BinTreeNode));
        (*Root) ->Data = Item;
        (*Root) ->LChild = NULL;
        (*Root) ->RChild = NULL;
    }
    else
        if (Item < (*Root) ->Data)
            RecBSTInsert(&(*Root) ->LChild,Item);
        else if (Item > (*Root) ->Data)
            RecBSTInsert(&(*Root) ->RChild,Item);
        else
            printf("TO STOIXEIO EINAI HDH STO DDA\n");
}

void RecBSTSearch(BinTreePointer Root, BinTreeElementType KeyValue,
                  boolean *Found, BinTreePointer *LocPtr)
```

```
/* Δέχεται:    Ένα ΔΔΑ με το δείκτη Root να δείχνει στη ρίζα του και μια τιμή KeyValue.
   Λειτουργία: Αναζητά στο ΔΔΑ έναν κόμβο με τιμή KeyValue στο πεδίο κλειδί του.
   Επιστρέφει: Η Found έχει τιμή TRUE και ο δείκτης LocPtr δείχνει στον κόμβο που
               περιέχει την τιμή KeyValue, αν η αναζήτηση είναι επιτυχής.
               Διαφορετικά η Found έχει τιμή FALSE
*/
{
    if (BSTEmpty(Root))
        *Found=FALSE;
    else
        if (KeyValue < Root->Data)
            RecBSTSearch(Root->LChild, KeyValue, &(*Found), &(*LocPtr));
        else
            if (KeyValue > Root->Data)
                RecBSTSearch(Root->RChild, KeyValue, &(*Found), &(*LocPtr));
            else
                {
                    *Found = TRUE;
                    *LocPtr=Root;
                }
}

void RecBSTDelete(BinTreePointer *Root, BinTreeElementType KeyValue)
/* Δέχεται:  Ένα ΔΔΑ με το δείκτη Root να δείχνει στη ρίζα του και μια τιμή KeyValue.
   Λειτουργία: Προσπαθεί να βρει έναν κόμβο στο ΔΔΑ που να περιέχει την τιμή
               KeyValue στο πεδίο κλειδί του τμήματος δεδομένων του και,
               αν τον βρει, τον διαγράφει από το ΔΔΑ.
  Επιστρέφει: Το τροποποιημένο ΔΔΑ με τον δείκτη Root να δείχνει στη ρίζα του.
*/
{

   BinTreePointer TempPtr;          //* true AN TO STOIXEIO KeyValue EINAI STOIXEIO TOY
DDA *)

    if (BSTEmpty(*Root))          //* ΑΔΕΙΟ ΔΕΝΔΡΟ ΤΟ KeyValue ΔΕ ΘΑ ΒΡΕΘΕΙ *)
        printf("TO STOIXEIO DEN BRE8HKE STO DDA\n");
     else
         //* αναζήτησε αναδρομικά τον κόμβο που περιέχει την τιμή KeyValue και διάγραψέ
τον
          if (KeyValue < (*Root)->Data)
             RecBSTDelete(&((*Root)->LChild), KeyValue);       //* ΑΡΙΣΤΕΡΟ ΥΠΟΔΕΝΔΡΟ *
          else
            if (KeyValue > (*Root)->Data)
                  RecBSTDelete(&((*Root)->RChild), KeyValue);   //* ΔΕΞΙ  ΥΠΟΔΕΝΔΡΟ *
            else                                        //* ΤΟ KeyValue ΒΡΕΘΗΚΕ ΔΙΑΓΡΑΦΗ
ΤΟΥ ΚΟΜΒΟΥ *)
                if ((*Root)->LChild ==NULL)
                 {
                      TempPtr = *Root;
                      *Root = (*Root)->RChild;      //* ΔΕΝ ΕΧΕΙ ΑΡΙΣΤΕΡΟ ΠΑΙΔΙ *)
                      free(TempPtr);
                 }
                else if ((*Root)->RChild == NULL)
                   {
                        TempPtr = *Root;
                        *Root = (*Root)->LChild;      //* ΕΧΕΙ ΑΡΙΣΤΕΡΟ ΠΑΙΔΙ, ΑΛΛΑ ΟΧΙ
ΔΕΞΙ *)
                        free(TempPtr);
                   }
                   else                                 //* ΕΧΕΙ 2 ΠΑΙΔΙΑ *)
                   {
                        //* ΕΥΡΕΣΗ ΤΟΥ INORDER ΑΠΟΓΟΝΟΥ ΤΟΥ *)
                        TempPtr = (*Root)->RChild;
                        while (TempPtr->LChild != NULL)
                              TempPtr = TempPtr->LChild;
                        /* ΜΕΤΑΚΙΝΗΣΗ ΤΟΥ ΑΠΟΓΟΝΟΥ ΤΗς ΡΙΖΑΣ ΤΟΥ ΥΠΟΔΕΝΔΡΟΥ
                        ΠΟΥ ΕΞΕΤΑΖΕΤΑΙ ΚΑΙ ΔΙΑΓΡΑΦΗ ΤΟΥ ΑΠΟΓΟΝΟΥ ΚΟΜΒΟΥ */
                        (*Root)->Data = TempPtr->Data;
                        RecBSTDelete(&((*Root)->RChild), (*Root)->Data);
```

```
                    }
}

void RecBSTInorder(BinTreePointer Root)
/* Δέχεται:    Ένα δυαδικό δέντρο με το δείκτη Root να δείχνει στην ρίζα του.
   Λειτουργία: Εκτελεί ενδοδιατεταγμένη διάσχιση του δυαδικού δέντρου και
                επεξεργάζεται κάθε κόμβο ακριβώς μια φορά.
   Εμφανίζει: Το περιεχόμενο του κόμβου, και εξαρτάται από το είδος της επεξεργασίας
*/
{
    if (Root!=NULL) {
    //    printf("L");
        RecBSTInorder(Root->LChild);
        printf("%d ",Root->Data);
    //    printf("R");
        RecBSTInorder(Root->RChild);
    }
   // printf("U");
}

void RecBSTPreorder(BinTreePointer Root)
/* Δέχεται:    Ένα δυαδικό δέντρο με το δείκτη Root να δείχνει στην ρίζα του.
   Λειτουργία: Εκτελεί προδιατεταγμένη διάσχιση του δυαδικού δέντρου και
                επεξεργάζεται κάθε κόμβο ακριβώς μια φορά.
   Εμφανίζει: Το περιεχόμενο του κόμβου, και εξαρτάται από το είδος της επεξεργασίας
*/
{
    if (Root!=NULL) {
        printf("%d ",Root->Data);
       // printf("L");
        RecBSTPreorder(Root->LChild);
      //  printf("R");
        RecBSTPreorder(Root->RChild);
    }
   // printf("U");
}

void RecBSTPostorder(BinTreePointer Root)
/* Δέχεται:    Ένα δυαδικό δέντρο με το δείκτη Root να δείχνει στην ρίζα του.
   Λειτουργία: Εκτελεί μεταδιατεταγμένη διάσχιση του δυαδικού δέντρου και
                επεξεργάζεται κάθε κόμβο ακριβώς μια φορά.
   Εμφανίζει: Το περιεχόμενο του κόμβου, και εξαρτάται από το είδος της επεξεργασίας
*/
{
    if (Root!=NULL) {
      //  printf("L");
        RecBSTPostorder(Root->LChild);
     //   printf("R");
        RecBSTPostorder(Root->RChild);
        printf("%d ",Root->Data);
    }
   // printf("U");
}
```

```
/*  Αρχείο: a26f5.c
    Φοιτητής: Ευστάθιος Ιωσηφίδης


    Άδεια χρήσης: GNU General Public License v3.0

    Δίνεται το αρχείο φοιτητών “foitites.dat”. Κάθε στοιχείο του αρχείου αυτού είναι μια εγγραφή με πεδία:
    Αριθμός μητρώου (ΑΜ: int), Επώνυμο (αλφαριθμητικό 20 χαρακτήρες), Ονομα (αλφαριθμητικό 20 χαρακτήρες), Φύλο
    (χαρακτήρας, τιμή F/M), Ετος (int), Βαθμός (μέσος όρος: float). Για την πιο αποτελεσματική επεξεργασία του αρχείου
    αυτού δημιουργείται ένα ευρετήριο (index) ως ΔΔΑ. Κάθε στοιχείο του ευρετηρίου αυτού αποτελείται από τον ΑΜ και τον
    αντίστοιχο αριθμό εγγραφής (γραμμής) στο αρχείο “foitites.dat”. Η αρίθμηση των γραμμών του αρχείου ξεκινούν από την
    τιμή 0.
    Γράψτε ένα πρόγραμμα που εκτελεί τις παρακάτω λειτουργίες και χρησιμοποιεί ως ευρετήριο ένα ΔΔΑ:
    1. Δημιουργία του index (ΔΔΑ) από το αρχείο “foitites.dat”.
    θα διαβάζει 1-1 τις εγγραφές του αρχείου “foitites.dat”και θα καταχωρεί στο ΔΔΑ τον ΑΜ του φοιτητή και τον
    αντίστοιχο αύξοντα αριθμό εγγραφής (γραμμής) στο αρχείο. Θα επιστρέφει το πλήθος των κόμβων του ΔΔΑ.
    Συνάρτηση int BuildBST(BinTreePointer *Root);
    2. Θα εμφανίζει το πλήθος των κόμβων του ΔΔΑ όπως και τους κόμβους του ΔΔΑ με αύξουσα διάταξη ως προς ΑΜ.
    3. Εισαγωγή νέων εγγραφών φοιτητών στο αρχείο foitites.dat και ενημέρωση του ΔΔΑ. Κάθε αλφαριθμητικό πεδίο όπως
    και το πεδίο φύλο (τύπου χαρακτήρας) να διαβάζονται με scanf() και στη συνέχεια getchar(). Για το φύλο δε χρειάζεται
    η να γίνεται έλεγχος εγκυρότητας θεωρούμε ότι θα δοθεί F ή M. Η fopen θα κληθεί με 2 παράμετρο “a” καθώς οι νέες
    εγγραφές θα προστεθούν μετά την τελευταία εγγραφή του αρχείου. Μετά από κάθε προσθήκη εγγραφής στο αρχείο θα εμφανίζετε
    το μέγεθος του αρχείου. (δείτε στο στιγμιότυπο εκτέλεσης).
    Συνάρτηση void writeNewStudents(BinTreePointer *Root, int *size);
    4. Θα εμφανίζει το πλήθος των κόμβων του ΔΔΑ όπως και τους κόμβους του ΔΔΑ με αύξουσα διάταξη ως προς ΑΜ.
    5. Αναζήτηση φοιτητή. Θα δίνεται ο ΑΜ του φοιτητή και θα τον αναζητά στο ΔΔΑ. Στη συνέχεια εφόσον υπάρχει στο ΔΔΑ
    θα τον εντοπίζει στο αρχείο “foitites.dat” και θα εμφανίζει όλες τις πληροφορίες της αντίστοιχης εγγραφής. Αν δεν
    υπάρχει στο ΔΔΑ θα εμφανίζει σχετικό μήνυμα.
    6. Θα εμφανίζει το πλήθος των κόμβων του ΔΔΑ όπως και τους κόμβους του ΔΔΑ με αύξουσα διάταξη ως προς ΑΜ.
    7. Εκτύπωση των στοιχείων όλων των φοιτητών που είναι καταχωρημένοι στο αρχείο “foitites.dat” με ΜΟ μεγαλύτερο
    από ένα δοσμένο βαθμό (πχ 0).
    Συνάρτηση void printStudentsWithGrade(float theGrade);
    Το αρχείο “foitites.dat” θα «ανοίγει» και θα «κλείνει» σε κάθε συνάρτηση που χρησιμοποιείται και με την κατάλληλη
    παράμετρο (“a” για εγγραφή στο τέλος του αρχείου ή “r” για διάβασμα των εγγραφών του αρχείου).
    Για κάθε μια από τις παραπάνω λειτουργίες εμφανίζεται στη main() σχετικό μήνυμα (Qx...)
*/

#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include <string.h>

// Δήλωση τύπων
typedef struct
{
    int code;
    int recNo;
```

```
} BinTreeElementType;

typedef struct BinTreeNode *BinTreePointer;

typedef struct BinTreeNode {
    BinTreeElementType Data;
    BinTreePointer LChild, RChild;
} BinTreeNode;

typedef struct
{
    int code;
    char surname[20];
    char name[20];
    char sex;
    int year;
    float grade;
} StudentT;

typedef enum {
    FALSE, TRUE
} boolean;

// Δήλωση συναρτήσεων
void CreateBST(BinTreePointer *Root);
boolean BSTEmpty(BinTreePointer Root);
void RecBSTInsert(BinTreePointer *Root, BinTreeElementType Item);
void RecBSTInorder(BinTreePointer Root);
void RecBSTSearch(BinTreePointer Root, BinTreeElementType KeyValue, boolean *Found,
BinTreePointer *LocPtr);

int BuildBST(BinTreePointer *Root);
void writeNewStudents(BinTreePointer *Root, int *size);
void printStudentsWithGrade(float theGrade);
void PrintStudent(int recNum);

int main()
{
   // Δηλώνω μεταβλητές
   BinTreePointer ARoot, LocPtr;
   float theGrade;
   int size; // πλήθος κόμβων του ΔΔΑ
   StudentT student;
   boolean Found;
   BinTreeElementType key;


   // Ερώτημα 1
   printf("Q1. Build BST from a file\n");
   size=BuildBST(&ARoot); // Δημιουργία ευρετηρίου και επιστροφή πλήθους κόμβων

   // Ερώτημα 2
   printf("Q2. Print size and BST\n");
   printf("Size=%d\n",size);
   printf("Nodes of BST\n");
   RecBSTInorder(ARoot); // Εμφάνιση των κόμβων δέντρου (σε αύξουσα διάταξη)

   // Ερώτημα 3
   printf("\nQ3. Write new students, update file and BST\n");
   writeNewStudents(&ARoot, &size); // Εισαγωγή νέων φοιτητών

   // Ερώτημα 4
   printf("Q4. Print size and BST\n");
   printf("Size=%d\n",size);
   printf("Nodes of BST\n");
   RecBSTInorder(ARoot); // Εμφάνιση των κόμβων δέντρου (σε αύξουσα διάταξη)

   // Ερώτημα 5
   printf("\nQ5. Search for a student\n");
```

```
   printf("Give student's code? "); // Διάβασε το ΑΜ
   scanf("%d",&key.code);
   RecBSTSearch(ARoot,key,&Found,&LocPtr); // Αναζήτησε τον φοιτητή
   if (Found)
   { // key.recNo = LocPtr->Data.recNo
     PrintStudent(LocPtr->Data.recNo);}
   else
   {printf("Student with code %d not found!\n",LocPtr->Data.code);}

   // Ερώτημα 6
   printf("Q6. Print size and BST\n");
   printf("Size=%d\n",size);
   printf("Nodes of BST\n");
   RecBSTInorder(ARoot);// Εμφάνιση των κόμβων δέντρου (σε αύξουσα διάταξη)

   // Ερώτημα 7
   printf("\nQ7. Print students with grades >= given grade\n");
    printf("Give the grade: ");
    scanf("%f", &theGrade);
    printStudentsWithGrade(theGrade);


   return 0;
}

// Συναρτήσεις

int BuildBST(BinTreePointer *Root)
{
  FILE *fp;
  int size; // Πλήθος εγγραφών φοιτητή
  int nscan;
  char termch;
  BinTreeElementType indexRec; // Εγγραφή φοιτητή για το ΔΔΑ
  StudentT student; // Εγγρφή για την ανάγνωση των στοιχείων φοιτητή από το αρχείο

  CreateBST(Root);

  fp = fopen("foitites.dat" ,"r"); // Ανοίγω αρχείο για ανάγνωση
  size=0; // Αρχικοποίηση του size

  if(fp == NULL) // Αν το αρχείο δεν άνοιξε, βγάλε μήνυμα σφάλματος
      {printf("Problem opening file\n");
      exit(1);}
  else // Αν το αρχείο άνοιξε επιτυχώς
    {
     while(TRUE)
     {
         nscan =
fscanf(fp,"%d,%20[^,],%20[^,],%c,%d,%f%c",&student.code,student.name,student.surname,&stu
dent.sex,&student.year,&student.grade,&termch); // Διάβασε το από το αρχείο την εγγραφή
φοιτητή
         if ( nscan == EOF ) // Αν έφτασε στο τέλος του αρχείου
            {printf("Tree built successfully!\n"); // Διαβάστηκε
            break;} // Σπάσε ρε μόρτη
         if ( nscan != 7||termch!='\n' ) // Αν η ανάγνωση της εγγραφής δεν ήταν επιτυχής
            {printf("Error\n");break;} // Εμφάνισε σφάλμα. Ουδείς άσφαλτος
         else // Αλλιώς
         {
             indexRec.code=student.code; // Καταχώρησε στο indexRec τον ΑΜ που διάβασες
             indexRec.recNo=size; // Καταχώρησε στον indexRec τον τρέχοντα αριθμό
εγγραφής
             RecBSTInsert(&(*Root),indexRec); // Εισαγωγή της εγγραφής του φοιτητή στο
ΔΔΑ
             size++; // Αύξησε κατα 1 το πλήθος των εγγραφών
         }
     }

    }
```

```
  fclose(fp); // Κλείσε το αρχείο
  return size; // Επέστρεψε το size
}

void PrintStudent(int recNum)
{
    FILE *fp;
    int nscan;
    int lines; // γραμμές του αρχείου που διαβάστηκαν
    char termch;
    StudentT student; // εγγραφή για την ανάγνωση των στοιχείων του φοιτητή από το αρχείο
    BinTreeElementType indexRec;

    lines=0; // Αρχικοποίηση του lines

    fp=fopen("foitites.dat","r"); // Άνοιξε το αρχείο

    if(fp == NULL) // Αν δεν άνοιξε το αρχείο
      printf("Problem opening file"); // Εμφάνισε σφάλμα
    else // Αν άνοιξε σωστά
      {
         while(lines<=recNum) // Όσο δεν έφτασε στη γραμμή του ζητούμενου φοιτητή
         {
          nscan = fscanf(fp, "%d,%20[^,],%20[^,],%c,%d,%f%c",&student.code,student.name,
student.surname,&student.sex,&student.year,&student.grade,&termch); // Διάβασε από το
αρχείο την εγγραφή του φοιτητή
          if ( nscan == EOF ) // Αν η ανάγνωση έφτασε στο τέλος του αρχείου
            {break;} // Σπάσε ρε μόρτη
          if ( nscan != 7||termch!='\n' ) // Αν η ανάγνωση της εγγραφή δεν ήταν επιτυχής
            { printf("Error\n");} // Εμφάνισε λάθος

          lines++; // Αύξησε το πλήθος των γραμμών

          }
          if(lines==recNum+1)
        printf("%d,%s,%s,%c,%d,%.1f\n",student.code,student.name,
student.surname,student.sex,student.year,student.grade); // Εμφάνισε όλα τα στοιχεία του
φοιτητή
        fclose(fp); // Κλείσε το αρχείο
    }

}


void printStudentsWithGrade(float theGrade)
{
    // Δήλωση μεταβλητών
    FILE *fp;
    int nscan;
    char termch;
    StudentT student;

    fp = fopen("foitites.dat" ,"r"); // Ανοίγω αρχείο για ανάγνωση

    if(fp == NULL) // Αν απέτυχε να ανοίξει το αρχείο
      printf("Problem opening file"); // Μήνυμα λάθους
    else // Αν άνοιξε με επιτυχία
      {
         while(TRUE)
         {
          nscan = fscanf(fp, "%d,%20[^,],%20[^,],%c,%d,%f%c",&student.code,student.name,
student.surname,&student.sex,&student.year,&student.grade,&termch); // Διάβασε το αρχείο
από την εγγραφή του φοιτητή
          if ( nscan == EOF ) // Αν έχει φτάσει στο τέλος του αρχείου
            {break;} // Σπάσε ρε μόρτη
          if ( nscan != 7||termch!='\n' ) // Αν η ανάγνωση της εγγραφής δεν ήταν επιτυχής
            { printf("Error\n");} // Κάπου έγινε λάθος. Ξαναψάξε
          else // Αλλιώς
           {
```

```
            if (student.grade>=theGrade) // Αν ο βαθμός του φοιτητή είναι >= theGrade
            {
                printf("%d,%s,%s,%c,%d,%.1f\n",student.code,student.name,
student.surname,student.sex,student.year,student.grade); // Εμφάνισε τα στοιχεία του
φοιτητή
            }
           }
          }

        fclose(fp); // Κλείνω το αρχείο
    }


}

void writeNewStudents(BinTreePointer *Root, int *size)

{
   BinTreePointer LocPtr;
   StudentT student; // εγγραφή για ανάγνωση των στοιχείων φοιτητή από το αρχείο
   boolean Found;
   int code;
   char ch;
   FILE *fp;
   BinTreeElementType indexRec; // Εγγραφή φοιτητή για το ΔΔΑ


    fp=fopen("foitites.dat","a");
    if(fp == NULL) // Αν απέτυχε να ανοίξει το αρχείο
      printf("Problem opening file"); // Μήνυμα λάθους
    else // Αν άνοιξε με επιτυχία
      {
          do // Επαναληπτικά
            {
                 printf("Give student's AM? "); // Διάβασε το ΑΜ
                 scanf("%d",&indexRec.code);
                 RecBSTSearch(*Root,indexRec,&Found,&LocPtr); // Αναζήτησε τον φοιτητή με
το παραπάνω ΑΜ
                 if(!Found) // Αν δε βρέθηκε
                 {
                 student.code=indexRec.code; // Καταχώρησε το ΑΜ στην εγγραφή για το
αρχείο
                 printf("Give student surname? "); // Διάβασε επώνυμο
                 scanf("%s",student.surname); getchar(); // Προσοχή, όχι &
                 printf("Give student name? "); // Διάβασε όνομα
                 scanf("%s",student.name); getchar() ;
                 printf("Give student sex F/M? "); // Διάβασε φύλο
                 scanf("%c",&student.sex); getchar();
                 printf("Give student's registration year? "); // Διάβασε έτος
                 scanf("%d",&student.year);
                 printf("Give student's grade? "); // Διάβασε βαθμό
                 scanf("%f",&student.grade);

                 indexRec.code = student.code;
                 indexRec.recNo = *size;
                 RecBSTInsert(Root,indexRec); // Εισαγωγή του φοιτητή στο ΔΔΑ
                 fprintf(fp,"%d,%s,%s,%c,%d,%.1f\n",student.code,student.name,
student.surname,student.sex,student.year,student.grade); // Εγγραφή στο αρχείο
                 (*size)++; // Αύξηση πλήθους φοιτητών κατά 1
                 printf("Size=%d\n",*size);
                 }
                 else
                  {printf("Student with code AM %d already exists!\n",indexRec.code);}


            printf("\nContinue Y/N: ");
            do // Επαναληπτικά
            {
```

```
            scanf("%c", &ch); // Διάβασε τον χαρακτήρα ch
            } while (toupper(ch)!= 'N' && toupper(ch)!= 'Y'); // όσο ο χαρακτήρας δεν
είναι N ή Y

       } while (toupper(ch)!='N'); // όσο ο χαρακτήρας δεν είναι N
       }

    fclose(fp); // Κλείνω το αρχείο
}

void RecBSTInorder(BinTreePointer Root)
/* Δέχεται:    Ένα δυαδικό δέντρο με το δείκτη Root να δείχνει στην ρίζα του.
   Λειτουργία: Εκτελεί ενδοδιατεταγμένη διάσχιση του δυαδικού δέντρου και
                επεξεργάζεται κάθε κόμβο ακριβώς μια φορά.
   Εμφανίζει: Το περιεχόμενο του κόμβου, και εξαρτάται από το είδος της επεξεργασίας
*/
{
    if (Root!=NULL) {
    //    printf("L");
        RecBSTInorder(Root->LChild);
        printf("(%d, %d), ",Root->Data.code,Root->Data.recNo); // 'Αλλαξε από printf("%d
",Root->Data);
    //    printf("R");
        RecBSTInorder(Root->RChild);
    }
   // printf("U");
}

void CreateBST(BinTreePointer *Root)
/* Λειτουργία: Δημιουργεί ένα κενό ΔΔΑ.
   Επιστρέφει:  Τον μηδενικό δείκτη(NULL) Root
*/
{
    *Root = NULL;
}

boolean BSTEmpty(BinTreePointer Root)
/* Δέχεται:   Ενα ΔΔα με το Root να δείχνει στη ρίζα του.
  Λειτουργία: Ελέγχει αν το ΔΔΑ είναι κενό.
  Επιστρέφει: TRUE αν το ΔΔΑ είναι κενό και FALSE διαφορετικά
*/
{
    return (Root==NULL);
}


void RecBSTSearch(BinTreePointer Root, BinTreeElementType KeyValue,
                  boolean *Found, BinTreePointer *LocPtr)
/* Δέχεται:    Ένα ΔΔΑ με το δείκτη Root να δείχνει στη ρίζα του και μια τιμή KeyValue.
   Λειτουργία: Αναζητά στο ΔΔΑ έναν κόμβο με τιμή KeyValue στο πεδίο κλειδί του.
   Επιστρέφει: Η Found έχει τιμή TRUE και ο δείκτης LocPtr δείχνει στον κόμβο που
                περιέχει την τιμή KeyValue, αν η αναζήτηση είναι επιτυχής.
                Διαφορετικά η Found έχει τιμή FALSE
*/
{
    if (BSTEmpty(Root))
        *Found=FALSE;
    else
        if (KeyValue.code < Root->Data.code)
            RecBSTSearch(Root->LChild, KeyValue, &(*Found), &(*LocPtr));
        else
            if (KeyValue.code > Root->Data.code)
                RecBSTSearch(Root->RChild, KeyValue, &(*Found), &(*LocPtr));
            else
                {
                    *Found = TRUE;
                    *LocPtr=Root;
                }
}
```

```
void RecBSTInsert(BinTreePointer *Root, BinTreeElementType Item)
/* Δέχεται:     Ένα ΔΔΑ με το δείκτη Root να δείχνει στη ρίζα του και ένα στοιχείο Item.
   Λειτουργία:  Εισάγει το στοιχείο Item στο ΔΔΑ.
   Επιστρέφει: Το τροποποιημένο ΔΔΑ με τον δείκτη Root να δείχνει στη ρίζα του
*/
{
    if (BSTEmpty(*Root)) {
        (*Root) = (BinTreePointer)malloc(sizeof (struct BinTreeNode));
        (*Root) ->Data = Item;
        (*Root) ->LChild = NULL;
        (*Root) ->RChild = NULL;
    }
    else
        if (Item.code < (*Root) ->Data.code)
            RecBSTInsert(&(*Root) ->LChild,Item);
        else if (Item.code > (*Root) ->Data.code)
            RecBSTInsert(&(*Root) ->RChild,Item);
        else
            printf("TO STOIXEIO EINAI HDH STO DDA\n");
}
```

```
/*  Αρχείο: a30f5.c
    Φοιτητής: Ευστάθιος Ιωσηφίδης


    Άδεια χρήσης: GNU General Public License v3.0

    Το αρχείο transactions.txt περιλαμβάνει ύποπτες συναλλαγές και θέλουμε να εντοπίσουμε
τις m μεγαλύτερες συναλλαγές.
    Η διαθέσιμη μνήμη δεν επαρκεί για να διαβάσουμε όλες τις συναλλαγές από το αρχείο και
να τις αποθηκεύσουμε σε μια
    δομή δεδομένων στη μνήμη. Επιλέξτε την κατάλληλη δομή δεδομένων ώστε να βρίσκει τις m
μεγαλύτερες συναλλαγές.
    Μετά την εύρεση των m μεγαλύτερων συναλλαγών θα εμφανίζει το μέγεθος και τα στοιχεία
της δομής δεδομένων
    (προσαρμόστε κατάλληλα την PrintHeap) και στη συνέχεια θα εμφανίσει σε αύξουσα
διάταξη τις m μεγαλύτερες
    συναλλαγές. Την τιμή της m θα τη δίνει ο χρήστης, και θεωρήστε ότι δίνεται τιμή πολύ
μικρότερη του μεγέθους του
    αρχείου χωρίς να γίνεται σχετικός έλεγχος. Ως δομή δεδομένων θα πρέπει να
χρησιμοποιηθεί σωρός (μέγιστος ή ελάχιστος
    σωρός;).

*/

#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>

#define  MaxElements 10          //το μέγιστο πλήθος των στοιχείων του σωρού

// Δήλωση τύπων
typedef float HeapElementType;    //ο τύπος δεδομένων των στοιχείων του σωρού
typedef  struct {
     HeapElementType key;
    // int Data;                // οποισδήποτε τύπος για τα παρελκόμενα δεδομένα κάθε
κόμβου
} HeapNode;


typedef struct {
        int Size;
        HeapNode Element[MaxElements+1];
} HeapType;

typedef enum {
    FALSE, TRUE
} boolean;

// Δήλωση συναρτήσεων
void CreateMinHeap(HeapType *Heap);
boolean FullHeap(HeapType Heap);
void InsertMinHeap(HeapType *Heap, HeapNode Item);
boolean EmptyHeap(HeapType Heap);
void DeleteMinHeap(HeapType *Heap, HeapNode *Item);
void PrintHeap(HeapType Heap);


int main()
{
    // Δήλωση μεταβλητών
    HeapType AHeap;
    HeapNode AnItem;
    int m;
    FILE *fp;

    // Διάβασε το m
    printf("Give m: ");
    scanf("%d", &m);
```

```
    CreateMinHeap(&AHeap); // Δημιούργησε τον σωρό ελαχίστων

    fp=fopen("transactions.txt","r"); // Άνοιξε το αρχείο για ανάγνωση

    while(!feof(fp)) // Όσο δεν έφτασες στο τέλος του αρχείου
    {
        fscanf(fp, "%f [^\n]", &AnItem.key); // Διάβασε την συναλλαγή
        InsertMinHeap(&AHeap, AnItem); // Εισαγωγή της συναλλαγής στον σωρό ελαχίστων
        if(AHeap.Size>m) // Αν το μέγεθος του σωρού ξεπέρασε το m
            DeleteMinHeap(&AHeap, &AnItem); // Διαγραφή του ελάχιστου στοιχείου από τον
σωρό
    }
    PrintHeap(AHeap); // Εμφάνισε το μέγεθος και το περιεχόμενο του σωρού
    printf("Transactions\n");

    while(!EmptyHeap(AHeap)) // Όσο ο σωρός δεν είναι κενός
    {
        DeleteMinHeap(&AHeap, &AnItem); // Διαγραφή του ελάχιστου στοιχείου από τον σωρό
        printf("%.2f ", AnItem.key); // Εμφάνιση του στοιχείου
    }

    printf("\n");
    return 0;
}

// Συναρτήσεις

//Αλλαγή βασικών λειτουργιών για σωρό ελαχίστων: InsertMinHeap, DeleteMinHeap
//Προσαρμογή PrintHeap: αφαιρέστε την NLR εμφάνιση

void CreateMinHeap(HeapType *Heap)
/* Λειτουργία: Δημιουργεί ένα κενό σωρό.
   Επιστρέφει: Ένα κενό σωρό
*/
{
  (*Heap).Size=0;
}

boolean EmptyHeap(HeapType Heap)
/* Δέχεται:    Ένα σωρό Heap.
   Λειτουργία: Ελέγχει αν ο σωρός είναι κενός.
   Επιστρέφει: TRUE αν ο σωρός είναι κενός, FALSE διαφορετικά
*/
{
  return (Heap.Size==0);
}

boolean FullHeap(HeapType Heap)
/* Δέχεται:    Ένα σωρό.
   Λειτουργία: Ελέγχει αν ο σωρός είναι γεμάτος.
   Επιστρέφει: TRUE αν ο σωρός είναι γεμάτος, FALSE διαφορετικά
*/
{
  return (Heap.Size==MaxElements);
}

void InsertMinHeap(HeapType *Heap, HeapNode Item)
/*  Δέχεται:    Ένα σωρό Heap και ένα στοιχείο δεδομένου Item .
    Λειτουργία: Εισάγει το στοιχείο Item στο σωρό, αν ο σωρός δεν είναι γεμάτος.
    Επιστρέφει: Τον τροποποιημένο σωρό.
    Έξοδος:     Μήνυμα γεμάτου σωρού αν ο σωρός είναι γεμάτος
*/
{
   int hole;

   if (!FullHeap(*Heap))
   {
          (*Heap).Size++;
```

```
          hole=(*Heap).Size;
          while (hole>1 && Item.key < Heap->Element[hole/2].key)
          {
            (*Heap).Element[hole]=(*Heap).Element[hole/2];
                     hole=hole/2;
          }
          (*Heap).Element[hole]=Item;
   }
   else
     printf("Full Heap...\n");
}

void DeleteMinHeap(HeapType *Heap, HeapNode *Item)
/* Δέχεται:    Ένα σωρό Heap.
   Λειτουργία: Ανακτά και διαγράφει το μεγαλύτερο στοιχείο του σωρού.
   Επιστρέφει: Το μεγαλύτερο στοιχείο Item του σωρού και τον τροποποιημένο σωρό
*/
{
   int parent, child;
   HeapNode last;
   boolean done;

   if (!EmptyHeap(*Heap))
   {
        done=FALSE;
        *Item=(*Heap).Element[1];
        last=(*Heap).Element[(*Heap).Size];
        (*Heap).Size--;

        parent=1; child=2;

        while (child<=(*Heap).Size && !done)
        {
            if (child<(*Heap).Size)
                if ((*Heap).Element[child].key > (*Heap).Element[child+1].key)
                    child++;
            if (last.key <= (*Heap).Element[child].key)
                done=TRUE;
            else
            {
               (*Heap).Element[parent]=(*Heap).Element[child];
               parent=child;
               child=2*child;
            }
        }
        (*Heap).Element[parent]=last;
    }
   else
       printf("Empty heap...\n");
}

void PrintHeap(HeapType Heap)
{
    int i;
    printf("Data structure size =%d\n", Heap.Size); // Εμφάνισε “Data structure size=”,
μέγεθος του σωρού
    for (i=1; i<=Heap.Size;i++)
        printf("%.2f ",Heap.Element[i].key); // Εμφάνισε το στοιχείο του σωρού που
βρίσκεται στη θέση της τιμής της μεταβλητής ελέγχου
    printf(" \n");
}
```

```
/*  Αρχείο: a7f6.c
    Φοιτητής: Ευστάθιος Ιωσηφίδης


    Άδεια χρήσης: GNU General Public License v3.0

    Γράψτε ένα πρόγραμμα για τη δημιουργία και επεξεργασία μιας ΔΔ που αποθηκεύει και επεξεργάζεται τα στοιχεία της με
    την τεχνική του κατακερματισμού με αλυσίδες συνωνύμων, στην οποία αποθηκεύονται τα στοιχεία των μελών ενός
    γυμναστηρίου. Κάθε εγγραφή περιλαμβάνει τα εξής στοιχεία της κάρτας μέλους που δίνεται σε κάθε μέλος όταν
    εγγράφεται στο γυμναστήριο:
    - τον κωδικό (ακέραιος αριθμός – κλειδί κατακερματισμού)
    - το όνομα μέλους (username) (αλφαριθμητικό 20 θέσεων)
    - το ποσό οφειλής του μέλους στο γυμναστήριο (ακέραιος)
    Η συνάρτηση κατακερματισμού να είναι: h(i) = i % 5.
    Στο κυρίως πρόγραμμα θα υλοποιούνται στη σειρά οι παρακάτω λειτουργίες:
    1. Create HashList
    Δημιουργία της δομής δεδομένων

    2. Insert new member
    Εισαγωγή νέου μέλους

    3. Search for a member
    Αναζήτηση μέλους – αν υπάρχει μέλος με το συγκεκριμένο κωδικό θα εμφανίζονται τα στοιχεία του,
    αλλιώς θα εμφανίζεται το μήνυμα 'DEN YPARXEI EGGRAFH ME KLEIDI x', όπου x ο κωδικός που δόθηκε προς αναζήτηση

    4. Update member amount
    Ενημέρωση της οφειλής του μέλους – ο χρήστης δίνει τον κωδικό του μέλους και το ποσό και ενημερώνεται κατάλληλα
    το ποσό της οφειλής. Κατ' αρχήν θα γίνεται έλεγχος αν υπάρχει μέλος με το συγκεκριμένο κωδικό, αν δεν
    υπάρχει θα εμφανίζεται το μήνυμα 'DEN YPARXEI EGGRAFH ME KLEIDI x', όπου x ο κωδικός που δόθηκε.
    Αν ο κωδικός υπάρχει θα διαβάζεται το ποσό που θα πληρώσει. Θα γίνεται έλεγχος ώστε το ποσό που δίνεται να είναι
    μικρότερο ή ίσο της καταχωρημένης οφειλής.

    5. Delete a member
    Διαγραφή μέλους – η διαγραφή δεν μπορεί να πραγματοποιηθεί αν το ποσό οφειλής του μέλους δεν έχει εξοφληθεί.
    Σε αυτή την περίπτωση η διαγραφή δεν πραγματοποιείται και εμφανίζεται το μήνυμα 'Not deleted arrange amount'.
    Κατ' αρχήν θα γίνεται έλεγχος αν υπάρχει μέλος με το συγκεκριμένο κωδικό, αν δεν υπάρχει θα εμφανίζεται το μήνυμα
    'DEN YPARXEI EGGRAFH ME KLEIDI x', όπου x ο κωδικός που δόθηκε. Ο χρήστης δίνει τον κωδικό ενός μέλους
    του γυμναστηρίου και εμφανίζονται τα περιεχόμενα της υπολίστας των συνωνύμων στην οποία ανήκει. Θα γίνεται έλεγχος
    αν υπάρχει μέλος με το συγκεκριμένο κωδικό, αν δεν υπάρχει θα εμφανίζεται το μήνυμα 'DEN YPARXEI EGGRAFH ME KLEIDI x',
    όπου x ο κωδικός που δόθηκε.

    6. Print list of synonyms
    Συνάρτηση void PrintListOfSynonyms(HashListType HList, int key);
    Για τις λειτουργίες 2 έως και 6 ο χρήστης του προγράμματος θα έχει τη δυνατότητα εισαγωγής, αναζήτησης, ενημέρωσης,
    διαγραφής και εμφάνισης της λίστας συνωνύμων για όσα μέλη του γυμναστηρίου επιθυμεί μέσω σχετικού μηνύματος
    'Continue Y/N?' Μετά τις λειτουργίες 1, 2, 4, 5 θα καλείτε τις Print_HashList(HList) και PrintPinakes(HList). Στην
    Print_HashList(HList) δε θα εμφανίζετε τη λίστα με τις ελεύθερες θέσεις της δομής.

*/
```

```
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>

#define HMax 5              /*     το μέγεθος του πίνακα HashTable
                                ενδεικτικά ίσο με 5 */
#define VMax 30             /*το μέγεθος της λίστας,
                                ενδεικτικά ίσο με 30 */
#define EndOfList -1        /* σημαία που σηματοδοτεί το τέλος της λίστας
                                και της κάθε υπολίστας συνωνύμων */

// Δήλωση τύπων
typedef struct{
        char name[20];
        int amount;
} ListElementType;  /*τύπος δεδομένων για τα στοιχεία της λίστας */
typedef int KeyType;

typedef struct {
        KeyType RecKey;
        ListElementType Data;
        int Link;
} ListElm;

typedef struct  {
        int HashTable[HMax];   // πίνακας δεικτών προς τις υπολίστες συνωνύμων
    int Size;                 // πλήθος εγγραφών της λίστας List
        int SubListPtr;          // Δδείκτης σε μια υπολίστα συνωνύμων
    int StackPtr;           // δείκτης προς την πρώτη ελεύθερη θέση της λίστας List
        ListElm List[VMax];
} HashListType;

typedef enum {
    FALSE, TRUE
} boolean;

// Δήλωση συναρτήσεων
void CreateHashList(HashListType *HList);
int HashKey(KeyType Key);
boolean FullHashList(HashListType HList);
void SearchSynonymList(HashListType HList,KeyType KeyArg,int *Loc,int *Pred);
void SearchHashList(HashListType HList,KeyType KeyArg,int *Loc,int *Pred);
void AddRec(HashListType *HList,ListElm InRec);
void DeleteRec(HashListType *HList,KeyType DelKey);

void Print_HashList(HashListType HList);
void PrintPinakes(HashListType HList);
void PrintListOfSynonyms(HashListType HList, KeyType key);

int main()
{
        // Δήλωση μεταβητών
    char ch;
    HashListType HList;
    ListElm AnItem;
    KeyType AKey;
    int Loc, Pred;

        // Ερώτημα 1
    printf("1. Create HashList\n");
    CreateHashList(&HList); // Δημιούργησε τη δομή κατακερματισμού
    Print_HashList(HList);
    PrintPinakes(HList);

        // Ερώτημα 2
    printf("2. Insert new member\n");
    do // Επαναληπτικά
    {
        printf("Give code: ");
```

```
        scanf("%d", &AnItem.RecKey); // Διάβασε κωδικό πελάτη
        printf("Give name: ");
        scanf("%s", AnItem.Data.name); getchar(); // Διάβασε όνομα πελάτη
        printf("Give amount: ");
        scanf("%d", &AnItem.Data.amount); // Διάβασε το ποσό που χρωστάει ο πελάτης
        AddRec(&HList, AnItem);

        printf("\nContinue Y/N: ");
        do{ // Επαναληπτικά
            scanf("%c", &ch); // Διάβασε τον χαρακτήρα ch
        } while (toupper(ch)!= 'N' && toupper(ch)!= 'Y');
    } while (toupper(ch)!='N');
    Print_HashList(HList);
    PrintPinakes(HList);

    // Ερώτημα 3
    printf("3. Search for a member\n");
    do // Επαναληπτικά
    {
        printf("Give code: ");
        scanf("%d", &AnItem.RecKey); // Διάβασε κωδικό πελάτη
        SearchHashList(HList, AnItem.RecKey, &Loc, &Pred); // Αναζήτησε τον πελάτη με τον
παραπάνω κωδικό
        if(Loc!=EndOfList) // Αν βρεθεί ο πελάτης Εμφάνισε τα στοιχεία του
            printf("[%d, %s, %d, %d]\n", HList.List[Loc].RecKey,
HList.List[Loc].Data.name, HList.List[Loc].Data.amount, HList.List[Loc].Link);
        else // Αλλιώς εμφάνισε μήνυμα
            printf("DEN YPARXEI EGGRAFH ME KLEIDI %d\n", AnItem.RecKey);
        printf("\nContinue Y/N: ");
        do{ // Επαναληπτικά
            scanf("%c", &ch); // Διάβασε τον χαρακτήρα ch
        } while (toupper(ch)!= 'N' && toupper(ch)!= 'Y');
    } while (toupper(ch)!='N');

    // Ερώτημα 4
    printf("4. Update member amount\n");
    do // Επαναληπτικά
    {
        printf("Give code: ");
        scanf("%d", &AnItem.RecKey); // Διάβασε κωδικό πελάτη
        SearchHashList(HList, AnItem.RecKey, &Loc, &Pred); // Αναζήτησε τον πελάτη με τον
παραπάνω κωδικό
        if(Loc!=EndOfList) // Αν βρεθεί ο πελάτης
        { // Εμφάνισε τα στοιχεία του
            printf("[%d, %s, %d, %d]\n", HList.List[Loc].RecKey,
HList.List[Loc].Data.name, HList.List[Loc].Data.amount, HList.List[Loc].Link);
            printf("Give amount: ");
            scanf("%d", &AKey); // Διάβασε το νέο ποσό
            if(HList.List[Loc].Data.amount>=AKey) // Αν το ποσό του πελάτη είναι
μεγαλύτερο ή ίσο του νέου ποσού
                HList.List[Loc].Data.amount-=AKey; // Ενημέρωσε το ποσό του πελάτη
        }
        else // Αλλιώς εμφάνισε μήνυμα
            printf("DEN YPARXEI EGGRAFH ME KLEIDI %d\n", AnItem.RecKey);
        printf("\nContinue Y/N: ");
        do{ // Επαναληπτικά
            scanf("%c", &ch); // Διάβασε τον χαρακτήρα ch
        } while (toupper(ch)!= 'N' && toupper(ch)!= 'Y');
    } while (toupper(ch)!='N');

    Print_HashList(HList);
    PrintPinakes(HList);

    // Ερώτημα 5
    printf("5. Delete a member\n");
    do // Επαναληπτικά
    {
        printf("Give code: ");
        scanf("%d", &AnItem.RecKey); // Διάβασε κωδικό πελάτη
```

```
        SearchHashList(HList, AnItem.RecKey, &Loc, &Pred); // Αναζήτησε τον πελάτη με τον
παραπάνω κωδικό
        if(Loc!=EndOfList) // Αν βρεθεί ο πελάτης
        {
            if(HList.List[Loc].Data.amount==0) // Αν ο πελάτης δεν χρωστάει ποσό
                DeleteRec(&HList, AnItem.RecKey); // Διέγραψε τον πελάτη από τη δομή
            else // Αν χρωστάει, εμφάνισε μήνυμα
                printf("Not deleted arrange amount\n");
        }
        else // Αν δεν βρεθεί ο πελάτης εμφάνισε μήνυμα
            printf("DEN YPARXEI EGGRAFH ME KLEIDI %d\n", AnItem.RecKey);

        printf("\nContinue Y/N: ");
        do{ // Επαναληπτικά
            scanf("%c", &ch);
        } while (toupper(ch)!= 'N' && toupper(ch)!= 'Y');
    } while (toupper(ch)!='N');
    Print_HashList(HList);
    PrintPinakes(HList);

    // Ερώτημα 6
    printf("6. Print list of synonyms\n");
    do // Επαναληπτικά
    {
        printf("Give code: ");
        scanf("%d", &AnItem.RecKey); // Διάβασε κωδικό πελάτη
        SearchHashList(HList, AnItem.RecKey, &Loc, &Pred); // Αναζήτησε τον πελάτη με τον
παραπάνω κωδικό
        if(Loc!=EndOfList) // Αν βρεθεί ο πελάτης
        { // Εμφάνισε τα στοιχεία του
            printf("[%d, %s, %d, %d]\n", HList.List[Loc].RecKey,
HList.List[Loc].Data.name, HList.List[Loc].Data.amount, HList.List[Loc].Link);
            PrintListOfSynonyms(HList, AnItem.RecKey); // Εμφάνισε τη λίστα συνωνύμων για
τον κωδικό του πελάτη
        }
        else // Αλλιώς εμφάνισε μήνυμα
            printf("DEN YPARXEI EGGRAFH ME KLEIDI %d\n", AnItem.RecKey);
        printf("\nContinue Y/N: ");
        do{ // Επαναληπτικά
            scanf("%c", &ch);// Διάβασε τον χαρακτήρα ch
        } while (toupper(ch)!= 'N' && toupper(ch)!= 'Y');
    } while (toupper(ch)!='N');

    return 0;
}

// Συναρτήσεις

int HashKey(KeyType Key)
/* Δέχεται:    Την τιμή Key ενός κλειδιού.
   Λειτουργία: Βρίσκει την τιμή κατακερματισμού HValue για το κλειδί Key.
   Επιστρέφει: Την τιμή κατακερματισμού HValue
*/
{
        /*σε περίπτωση που το KeyType δεν είναι ακέραιος
        θα πρέπει να μετατρέπεται κατάλληλα το κλειδί σε αριθμό*/
        return Key%HMax;
}

void CreateHashList(HashListType *HList)
/* Λειτουργία: Δημιουργεί μια δομή HList.
   Επιστρέφει: Την δομή HList
*/
{
        int index;

        HList->Size=0;          //ΔΗΜΙΟΥΡΓΕΙ ΜΙΑ ΚΕΝΗ ΛΙΣΤΑ
        HList->StackPtr=0;      //ΔΕΙΚΤΗς ΣΤΗ ΚΟΡΥΦΗ ΤΗΣ ΣΤΟΙΒΑΣ ΤΩΝ ΕΛΕΥΘΕΡΩΝ ΘΕΣΕΩΝ
```

```
    /*ΑΡΧΙΚΟΠΟΙΕΙ ΤΟΝ ΠΙΝΑΚΑ HashTable ΤΗΣ ΔΟΜΗΣ HList ΩΣΤΕ ΚΑΘΕ ΣΤΟΙΧΕΙΟΥ ΤΟΥ
        ΝΑ ΕΧΕΙ ΤΗ ΤΙΜΗ EndOfList (-1)*/
    index=0;
        while (index<HMax)
        {
                HList->HashTable[index]=EndOfList;
                index=index+1;
    }

     //Δημιουργία της στοίβας των ελεύθερων θέσεων στη λίστα HList
    index=0;
        while(index < VMax-1)
        {
                HList->List[index].Link=index+1;
                index=index+1;
        }
        HList->List[index].Link=EndOfList;
}

boolean FullHashList(HashListType HList)
/* Δέχεται:    Μια δομή HList.
   Λειτουργία: Ελέγχει αν η λίστα List της δομής HList είναι γεμάτη.
   Επιστρέφει: TRUE αν η λίστα List είναι γεμάτη, FALSE διαφορετικά.
*/
{
        return(HList.Size==VMax);
}

void SearchSynonymList(HashListType HList,KeyType KeyArg,int *Loc,int *Pred)
/* Δέχεται:     Μια δομή HList και μια τιμή κλειδιού KeyArg.
    Λειτουργία: Αναζητά μια εγγραφή με κλειδί KeyArg στην υπολίστα συνωνύμων.
    Επιστρέφει: Τη θέση Loc της εγγραφής και τη θέση Pred της προηγούμενης
                εγγραφής στην υπολίστα
*/
{
        int Next;
        Next=HList.SubListPtr;
        *Loc=-1;
        *Pred=-1;
        while(Next!=EndOfList)
        {
                if (HList.List[Next].RecKey==KeyArg)
                {
                        *Loc=Next;
                        Next=EndOfList;
                }
                else
                {
                        *Pred=Next;
                        Next=HList.List[Next].Link;
                }
        }
}
void SearchHashList(HashListType HList,KeyType KeyArg,int *Loc,int *Pred)
/* Δέχεται:     Μια δομή HList και μια τιμή κλειδιού KeyArg.
    Λειτουργία: Αναζητά μια εγγραφή με κλειδί KeyArg στη δομή HList.
    Επιστρέφει: Τη θέση Loc της εγγραφής και τη θέση Pred της
                προηγούμενης εγγραφής της υπολίστας στην οποία ανήκει.
                Αν δεν υπάρχει εγγραφή με κλειδί KeyArg τότε Loc=Pred=-1
*/
{
        int HVal;
        HVal=HashKey(KeyArg);
    if (HList.HashTable[HVal]==EndOfList)
        {
                *Pred=-1;
                *Loc=-1;
        }
        else
```

```
        {
                HList.SubListPtr=HList.HashTable[HVal];
                SearchSynonymList(HList,KeyArg,Loc,Pred);
        }
}

void AddRec(HashListType *HList,ListElm InRec)
/* Δέχεται:    Μια δομή HList και μια εγγραφή InRec.
   Λειτουργία: Εισάγει την εγγραφή InRec στη λίστα List, αν δεν είναι γεμάτη,
               και ενημερώνει τη δομή HList.
   Επιστρέφει: Την τροποποιημένη δομή HList.
   Έξοδος:     Μήνυμα γεμάτης λίστας, αν η List είναι γεμάτη, διαφορετικά,
               αν υπάρχει ήδη εγγραφή με το ίδιο κλειδί,
               εμφάνιση αντίστοιχου μηνύματος
*/
{
        int Loc, Pred, New, HVal;

   // New=0;
        if(!(FullHashList(*HList)))
        {
                Loc=-1;
                Pred=-1;
                SearchHashList(*HList,InRec.RecKey,&Loc,&Pred);
                if(Loc==-1)
                {
                        HList->Size=HList->Size +1;
                        New=HList->StackPtr;
                        HList->StackPtr=HList->List[New].Link;
                        HList->List[New]=InRec;
                        if (Pred==-1)
                        {
                            HVal=HashKey(InRec.RecKey);
                HList->HashTable[HVal]=New;
                                HList->List[New].Link=EndOfList;
                        }
                        else
                        {
                HList->List[New].Link=HList->List[Pred].Link;
                                HList->List[Pred].Link=New;
                        }
                }

                else
                {
                        printf("YPARXEI HDH EGGRAFH ME TO IDIO KLEIDI \n");
                }
        }
        else
        {
                printf("Full list...");
        }
}
void DeleteRec(HashListType *HList,KeyType DelKey)
/* DEXETAI:    TH DOMH (HList) KAI To KLEIDI (DelKey) THS EGGRAFHS
               POY PROKEITAI NA DIAGRAFEI
   LEITOYRGIA: DIAGRAFEI, THN EGGRAFH ME KLEIDI (DelKey) APO TH
               LISTA (List), AN YPARXEI ENHMERWNEI THN DOMH HList
   EPISTREFEI: THN TROPOPOIHMENH DOMH (HList)
   EJODOS:     "DEN YPARXEI EGGRAFH ME KLEIDI" MHNYMA
*/
{
        int Loc, Pred, HVal;

        SearchHashList(*HList,DelKey,&Loc,&Pred);
        if(Loc!=-1)
        {
                if(Pred!=-1)
                {
```

```
                        HList->List[Pred].Link=HList->List[Loc].Link;
                }
                else
                {
                        HVal=HashKey(DelKey);
                    HList->HashTable[HVal]=HList->List[Loc].Link;
                }
                HList->List[Loc].Link=HList->StackPtr;
                HList->StackPtr=Loc;
                HList->Size=HList->Size -1;
        }
        else
        {
                printf("DEN YPARXEI  EGGRAFH ME KLEIDI %d \n",DelKey);
        }
}

void Print_HashList(HashListType HList)
{
   int i, SubIndex;

     printf("HASHLIST STRUCTURE with SYNONYM CHAINING\n");
     printf("========================================\n");

          printf("PINAKAS DEIKTWN STIS YPO-LISTES SYNWNYMWN EGGRAFWN:\n");
            for (i=0; i<HMax;i++)
              printf("%d| %d\n",i,HList.HashTable[i]);

          printf("OI YPO-LISTES TWN SYNWNYMWN EGGRAFWN:\n");
          for (i=0; i<HMax;i++)
          {
              SubIndex = HList.HashTable[i];
              while ( SubIndex != EndOfList )
              {
                   printf("[%d, %s, %d,
%d]",HList.List[SubIndex].RecKey,HList.List[SubIndex].Data.name,HList.List[SubIndex].Data
.amount,HList.List[SubIndex].Link);
                   printf(" -> ");
                   SubIndex = HList.List[SubIndex].Link;
               } //* while *)
              printf("TELOS % dHS YPO-LISTAS\n", i);
          }

          printf("MEGE8OS THS LISTAS = %d\n", HList.Size);
          printf("========================================\n");
}

void PrintPinakes(HashListType HList)
{
    int i;
        printf("Hash table\n");
        for (i=0; i<HMax;i++)
              printf("%d| %d\n",i,HList.HashTable[i]);

        printf("Hash List\n");
    for (i=0;i<HList.Size;i++)
           printf("%d) %d, %s, %d,
%d\n",i,HList.List[i].RecKey,HList.List[i].Data.name,HList.List[i].Data.amount,HList.List
[i].Link);
        printf("\n");
}

void PrintListOfSynonyms(HashListType HList, KeyType key)
{
    KeyType Hval, SubIndex; // Δήλωση μεταβλητών

    Hval=HashKey(key); // Υπολογίζω την τιμή κατακερματισμού του key

    SubIndex=HList.HashTable[Hval]; // Αρχικοποίησε κατάλληλα το SubIndex στην αρχή της
```

```
λίστας συνωνύμων
    while(SubIndex!=EndOfList) // Όσο δεν φτάνουμε στο τέλος της λίστας
    {  // Εμφάνισε όλα τα στοιχεία του τρέχοντος πελάτη της λίστας
        printf("%d: [%d, %s,
%d]\n",SubIndex,HList.List[SubIndex].RecKey,HList.List[SubIndex].Data.name,HList.List[Sub
Index].Data.amount);
        SubIndex=HList.List[SubIndex].Link; // Ενημέρωσε το SubIndex στο επόμενο στοιχείο
της λίστας συνωνύμων
    }
}
```

```
/*  Αρχείο: a4f6.c
    Φοιτητής: Ευστάθιος Ιωσηφίδης

    Άδεια χρήσης: GNU General Public License v3.0

    Σε έναν εκπαιδευτικό οργανισμό εργάζονται εκπαιδευτικοί διαφόρων ειδικοτήτων. Τα βασικά τους στοιχεία υπάρχουν σε
    ένα αρχείου κειμένου 'i4f6.txt' (σε διαφορετικές γραμμές για κάθε εκπαιδευτικό):
    Ονομα: αλφαριθμητικό 10 χαρακτήρες
    Επώνυμο: αλφαριθμητικό 20 χαρακτήρες
    Τηλέφωνο: αλφαριθμητικό 10 θέσεων
    Κωδικός ειδικότητας: byte (1=θεολόγοι, 2=Φιλόλογοι, …20=Πληροφορικοί)

    Στο κυρίως πρόγραμμα θα υλοποιούνται στη σειρά οι παρακάτω λειτουργίες:

    1. BuildHashList
    Διάβασμα των στοιχείων από το αρχείο κειμένου και δημιουργία Δομής Δεδομένων (ΔΔ) που αποθηκεύει και επεξεργάζεται
    τα στοιχεία της με την τεχνική του κατακερματισμού με αλυσίδες συνωνύμων. Το κλειδί σχηματίζεται από το όνομα+κενό χαρακτήρα+επώνυμο.
    Πχ αν το όνομα είναι «nikos» και το επώνυμο «dimitriou», τότε το κλειδί κατακερματισμού που θα σχηματιστεί είναι:
    «nikos dimitriou»
    void BuildHashList(HashListType *HList);

    2. Insert new teacher
    Εισαγωγή των στοιχείων ενός νέου εκπαιδευτικού στη ΔΔ κατακερματισμού με αλυσίδες συνωνύμων. Η εισαγωγή νέων εκπαιδευτικών στη ΔΔ
    γίνεται επαναληπτικά μέσω σχετικού μηνύματος 'Continue Y/N?'

    3. Delete a teacher
    Διαγραφή ενός εκπαιδευτικού από τη ΔΔ κατακερματισμού με αλυσίδες συνωνύμων

    4. Search for a teacher
    Αναζήτηση και εμφάνιση των στοιχείων ενός εκπαιδευτικού βάσει ονοματεπωνύμου στη ΔΔ κατακερματισμού με αλυσίδες συνωνύμων

    5. Search by subject
    Αναζήτηση και εμφάνιση των στοιχείων των εκπαιδευτικών μιας συγκεκριμένης ειδικότητας (ο κωδικός της ειδικότητας [1..20]
    αποτελεί παράμετρο της διαδικασίας) στη ΔΔ κατακερματισμού με αλυσίδες συνωνύμων
    void Search_HashList_By_Subject(HashListType HList, int code);

    Μετά τις λειτουργίες 1, 2, 3 θα καλείτε την PrintPinakes(HList).
    Ο πίνακας κατακερματισμού θα έχει 9 θέσεις και ως συνάρτηση κατακερματισμού θα χρησιμοποιηθεί η εξής:
    h(i) = average % 9
    όπου
    average = (κωδικός_πρώτου_ χαρακτήρα + κωδικός_τελευταίου_χαρακτήρα) / 2
    Θεωρήστε ότι χρησιμοποιούνται οι ακόλουθοι κωδικοί για τους χαρακτήρες: 'A' = 1, 'B' = 2, …, 'Z' = 26. Ο πρώτος και ο
    τελευταίος χαρακτήρας του ονοματεπωνύμου θα μετατρέπεται στον αντίστοιχο κεφαλαίο χαρακτήρα, εφόσον είναι πεζός.
    Ο μέσος όρος average θα υπολογίζεται με μια συνάρτηση findAverage, η οποία θα καλείται από τη συνάρτηση HashKey.


*/

#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include <string.h>

#define HMax 9              /*     το μέγεθος του πίνακα HashTable
                                ενδεικτικά στην άσκηση ίσο με 9 */
#define VMax 30             /*το μέγεθος της λίστας,
                                ενδεικτικά ίσο με 30 */
```

```
#define EndOfList -1        /* σημαία που σηματοδοτεί το τέλος της λίστας
                               και της κάθε υπολίστας συνωνύμων */

// Δήλωση τύπων
typedef char KeyType[32]; // Τύπος για κλειδί κατακερματισμού

typedef struct{
    char phone[11];
    int type;
} ListElementType;  /*τύπος δεδομένων για τα στοιχεία της λίστας */

typedef struct {
        KeyType RecKey;
        ListElementType Data;
        int Link;
} ListElm;

typedef struct  {
        int HashTable[HMax];   // πίνακας δεικτών προς τις υπολίστες συνωνύμων
    int Size;                 // πλήθος εγγραφών της λίστας List
        int SubListPtr;           // Δδείκτης σε μια υπολίστα συνωνύμων
    int StackPtr;            // δείκτης προς την πρώτη ελεύθερη θέση της λίστας List
        ListElm List[VMax];
} HashListType;

typedef enum {
    FALSE, TRUE
} boolean;

// Δήλωση συναρτήσεων
void CreateHashList(HashListType *HList);
int HashKey(KeyType Key);
boolean FullHashList(HashListType HList);
void SearchSynonymList(HashListType HList,KeyType KeyArg,int *Loc,int *Pred);
void SearchHashList(HashListType HList,KeyType KeyArg,int *Loc,int *Pred);
void AddRec(HashListType *HList,ListElm InRec);
void DeleteRec(HashListType *HList,KeyType DelKey);
void PrintPinakes(HashListType HList);

int findAverage(char s[]);
void BuildHashList(HashListType *HList);
void Search_HashList_By_Subject(HashListType HList, int code);


int main()
{
        // Δήλωση μεταβλητών
    HashListType HList;
    ListElm AnItem;
    KeyType AKey;
    int Loc, Pred;
    char surname[20];
    char ch;
    int code;

        // Ερώτημα 1
    printf("1. Create HashList\n");
    BuildHashList(&HList); // Δημιουργία δομής κατακερματισμού από το αρχείο
    PrintPinakes(HList);

        // Ερώτημα 2
    printf("2. Insert new teacher\n");
    do // Επαναληπτικά
    {
        printf("Enter teacher's name: ");
        scanf("%s", AnItem.RecKey); getchar(); // Διάβασε όνομα εκπαιδευτικού
        printf("Enter teacher's surname: ");
        scanf("%s", surname); getchar(); // Διάβασε επώνυμο εκπαιδευτικού
        strcat(AnItem.RecKey, " ");
```

```
        strcat(AnItem.RecKey, surname); // Δημιουργία κλειδιού κατακερματισμού
(ονοματεπώνυμο)
        printf("Enter teacher's phone: ");
        scanf("%s", AnItem.Data.phone); getchar(); // Διάβασε το τηλέφωνο του
εκπαιδευτικού
        printf("Enter teacher code: ");
        scanf("%d", &AnItem.Data.type); // Διάβασε τον κωδικό ειδικότητας του
εκπαιδευτικού
        AnItem.Link = EndOfList;
        AddRec(&HList, AnItem); // Εισήγαγε τον εκπαιδευτικό στη δομή

        printf("\nContinue Y/N: ");
        do{
            scanf("%c", &ch); // Διάβασε τον χαρακτήρα ch
        } while (toupper(ch)!= 'N' && toupper(ch)!= 'Y');
    } while (toupper(ch)!='N');
    PrintPinakes(HList);

    // Ερώτημα 3
    printf("3. Delete a teacher\n");
    printf("Enter teacher's name: ");
    scanf("%s", AKey); getchar(); // Διάβασε όνομα εκπαιδευτικού
    printf("Enter teacher's surname: ");
    scanf("%s", surname); getchar(); // Διάβασε επώνυμο εκπαιδευτικού
    strcat(AKey, " ");
    strcat(AKey, surname); // Δημιουργία κλειδιού κατακερματισμού
    DeleteRec(&HList, AKey); // Διέγραψε τον εκπαιδευτικό από τη δομή
    PrintPinakes(HList);

    // Ερώτημα 4
    printf("4. Search for a teacher\n");
    printf("Enter teacher's name: ");
    scanf("%s", AnItem.RecKey); getchar(); // Διάβασε όνομα εκπαιδευτικού
    printf("Enter teacher's surname: ");
    scanf("%s", surname); getchar(); // Διάβασε επώνυμο εκπαιδευτικού
    strcat(AnItem.RecKey, " ");
    strcat(AnItem.RecKey, surname); // Δημιουργία κλειδιού κατακερματισμού
    SearchHashList(HList, AnItem.RecKey, &Loc, &Pred); // Αναζήτησε τον εκπαιδευτικό στη
δομή
    if(Loc!=-1) // Αν βρεθεί ο εκπαιδευτικός
        printf("[%s, %s, %d]\n\n", HList.List[Loc].RecKey, HList.List[Loc].Data.phone,
HList.List[Loc].Data.type); // Εμφάνισε τα στοιχεία του
    else // Διαφορετικά
        printf("DEN YPARXEI EGGRAFH ME KLEIDI %s\n\n", AnItem.RecKey); // Εμφάνισε μήνυμα

    // Ερώτημα 5
    printf("5. Search by subject\n");
    printf("Enter code: ");
    scanf("%d", &code); // Διάβασε κωδικό ειδικότητας
    Search_HashList_By_Subject(HList, code); // Αναζήτησε τους εκπαιδευτικούς της
ειδικότητας

    return 0;
}

// Συναρτήσεις

int findAverage(KeyType s)
{
    // Δηλώνω τις μεταβλητές
    int first, last;

    first=toupper(s[0]); // Αποθήκευσε στην first τον πρώτο χαρακτήρα του s (αφού έχει
μετατραπεί σε κεφαλαία)
    last=toupper(s[strlen(s)-1]); // Αποθήκευσε στην last τον κωδικό του τελευταίου
χαρακτήρα του (αφού έχει μετατραπεί σε κεφαλαία)

    return ((first-64)+(last-64))/2; // Επέστρεψε (κωδικός_πρώτου_ χαρακτήρα +
κωδικός_τελευταίου_χαρακτήρα) / 2
```

```
}


int HashKey(KeyType Key)
/* Δέχεται:    Την τιμή Key ενός κλειδιού.
   Λειτουργία: Βρίσκει την τιμή κατακερματισμού HValue για το κλειδί Key.
   Επιστρέφει: Την τιμή κατακερματισμού HValue
*/
{
    int avg;

        avg=findAverage(Key); // Υπολόγισε τον ακέραιο avg από το κλειδί κατακερματισμού

        /*σε περίπτωση που το KeyType δεν είναι ακέραιος
        θα πρέπει να μετατρέπεται κατάλληλα το κλειδί σε αριθμό*/
        return avg%HMax; // Επέστρεψε την τιμή κατακερματισμού
}

void BuildHashList(HashListType *HList)
{
    // Δηλώνω τις μεταβλητές
    ListElm AnItem;
    FILE *fp;
    char surname[20], termch;
    int nscan;

    CreateHashList(HList); // Δημιουργία δομή κατακερματισμού

    fp=fopen("i4f6.txt", "r"); // Άνοιξε το αρχείο "i4f6.txt" για ανάγνωση

    if(fp!=NULL){ // Αν το αρχείο άνοιξε επιτυχώς
        while(TRUE){
            // Διάβασε τα στοιχεία του εκπαιδευτικού
            nscan=fscanf(fp, "%10[^,],%20[^,],%10[^,],%d%c",
AnItem.RecKey,surname,AnItem.Data.phone,&AnItem.Data.type,&termch);
            if(nscan==EOF) // Αν η ανάγνωση έφτασε στο τέλος του αρχείου
                break; // σπάσε ρε μόρτη
            if(nscan!=5 || termch != '\n') // Αν η ανάγνωση της εγγραφής δεν ήταν
επιτυχής
                printf("Error\n"); // Κάτι δεν έκανες σωστά
            else{ // Αλλιώς
                strcat(AnItem.RecKey, " ");
                strcat(AnItem.RecKey, surname); // Δημιούργησε το κλειδί κατακερματισμού
                AddRec(HList, AnItem); // Εισήγαγε τον εκπαιδευτικό στη δομή
            }
        }
    }
    fclose(fp); // Κλείσε το αρχείο
}

void Search_HashList_By_Subject(HashListType HList, int code)
{
    // Δηλώνω τις μεταβλητές
    int i, SubIndex;

    printf("List of teachers with subject code %d: \n", code);
    for (i=0; i<HMax;i++) // Για κάθε θέση i του πίνακα κατακερματισμού
    {
        //Διατρέχουμε τη λίστα συνωνύμων για τη θέση i
        SubIndex = HList.HashTable[i]; // Ανάθεση στο SubIndex του πρώτου στοιχείου της
λίστας συνωνύμων για τη θέση i
        while ( SubIndex != EndOfList ) // Όσο δεν έχουμε φτάσει στο τέλος της λίστας
συνωνύμων
            {
            if(HList.List[SubIndex].Data.type==code) // Αν η ειδικότητα του εκπαιδευτικού
για το τρέχον στοιχείο είναι ίσο με το code
            {
                //Εμφάνισε τα στοιχεία του εκπαιδευτικού
                printf("%d: [%s, %s, %d]\n", SubIndex,
```

```
HList.List[SubIndex].RecKey,HList.List[SubIndex].Data.phone, code);
            }
            SubIndex = HList.List[SubIndex].Link; // Ανάθεση στο SubIndex του επόμενου
στοιχείου της λίστας συνωνυμων
    }
}
}

void CreateHashList(HashListType *HList)
/* Λειτουργία: Δημιουργεί μια δομή HList.
   Επιστρέφει: Την δομή HList
*/
{
        int index;

        HList->Size=0;          //ΔΗΜΙΟΥΡΓΕΙ ΜΙΑ ΚΕΝΗ ΛΙΣΤΑ
        HList->StackPtr=0;      //ΔΕΙΚΤΗς ΣΤΗ ΚΟΡΥΦΗ ΤΗΣ ΣΤΟΙΒΑΣ ΤΩΝ ΕΛΕΥΘΕΡΩΝ ΘΕΣΕΩΝ

    /*ΑΡΧΙΚΟΠΟΙΕΙ ΤΟΝ ΠΙΝΑΚΑ HashTable ΤΗΣ ΔΟΜΗΣ HList ΩΣΤΕ ΚΑΘΕ ΣΤΟΙΧΕΙΟΥ ΤΟΥ
        ΝΑ ΕΧΕΙ ΤΗ ΤΙΜΗ EndOfList (-1)*/
    index=0;
        while (index<HMax)
        {
                HList->HashTable[index]=EndOfList;
                index=index+1;
    }

     //Δημιουργία της στοίβας των ελεύθερων θέσεων στη λίστα HList
    index=0;
        while(index < VMax-1)
        {
                HList->List[index].Link=index+1;
                index=index+1;
        }
        HList->List[index].Link=EndOfList;
}

boolean FullHashList(HashListType HList)
/* Δέχεται:    Μια δομή HList.
   Λειτουργία: Ελέγχει αν η λίστα List της δομής HList είναι γεμάτη.
   Επιστρέφει: TRUE αν η λίστα List είναι γεμάτη, FALSE διαφορετικά.
*/
{
        return(HList.Size==VMax);
}

void SearchSynonymList(HashListType HList,KeyType KeyArg,int *Loc,int *Pred)
/* Δέχεται:     Μια δομή HList και μια τιμή κλειδιού KeyArg.
    Λειτουργία: Αναζητά μια εγγραφή με κλειδί KeyArg στην υπολίστα συνωνύμων.
    Επιστρέφει: Τη θέση Loc της εγγραφής και τη θέση Pred της προηγούμενης
                εγγραφής στην υπολίστα
*/
{
        int Next;
        Next=HList.SubListPtr;
        *Loc=-1;
        *Pred=-1;
        while(Next!=EndOfList)
        {
                if (strcmp(HList.List[Next].RecKey, KeyArg) == 0) //Αλλαγές στην
SearchSynonymList για τη διαχείριση συμβολοσειρών (χρήση strcmp)
                {
                        *Loc=Next;
                        Next=EndOfList;
                }
                else
                {
                        *Pred=Next;
                        Next=HList.List[Next].Link;
```

```
                }
        }
}
void SearchHashList(HashListType HList,KeyType KeyArg,int *Loc,int *Pred)
/* Δέχεται:     Μια δομή HList και μια τιμή κλειδιού KeyArg.
    Λειτουργία: Αναζητά μια εγγραφή με κλειδί KeyArg στη δομή HList.
    Επιστρέφει: Τη θέση Loc της εγγραφής και τη θέση Pred της
                προηγούμενης εγγραφής της υπολίστας στην οποία ανήκει.
                Αν δεν υπάρχει εγγραφή με κλειδί KeyArg τότε Loc=Pred=-1
*/
{
        int HVal;
        HVal=HashKey(KeyArg);
    if (HList.HashTable[HVal]==EndOfList)
        {
                *Pred=-1;
                *Loc=-1;
        }
        else
        {
                HList.SubListPtr=HList.HashTable[HVal];
                SearchSynonymList(HList,KeyArg,Loc,Pred);
        }
}

void AddRec(HashListType *HList,ListElm InRec)
/* Δέχεται:    Μια δομή HList και μια εγγραφή InRec.
   Λειτουργία: Εισάγει την εγγραφή InRec στη λίστα List, αν δεν είναι γεμάτη,
                και ενημερώνει τη δομή HList.
   Επιστρέφει: Την τροποποιημένη δομή HList.
   Έξοδος:     Μήνυμα γεμάτης λίστας, αν η List είναι γεμάτη, διαφορετικά,
                αν υπάρχει ήδη εγγραφή με το ίδιο κλειδί,
                εμφάνιση αντίστοιχου μηνύματος
*/
{
        int Loc, Pred, New, HVal;


        if(!(FullHashList(*HList)))
        {
                Loc=-1;
                Pred=-1;
                SearchHashList(*HList,InRec.RecKey,&Loc,&Pred);
                if(Loc==-1)
                {
                        HList->Size=HList->Size +1;
                        New=HList->StackPtr;
                        HList->StackPtr=HList->List[New].Link;
                        HList->List[New]=InRec;
                        if (Pred==-1)
                        {
                            HVal=HashKey(InRec.RecKey);
                HList->HashTable[HVal]=New;
                                HList->List[New].Link=EndOfList;
                        }
                        else
                        {
                HList->List[New].Link=HList->List[Pred].Link;
                                HList->List[Pred].Link=New;
                        }
                }

                else
                {
                        printf("YPARXEI HDH EGGRAFH ME TO IDIO KLEIDI \n");
                }
        }
        else
        {
```

```
                printf("Full list...");
        }
}
void DeleteRec(HashListType *HList,KeyType DelKey)
/* DEXETAI:    TH DOMH (HList) KAI To KLEIDI (DelKey) THS EGGRAFHS
               POY PROKEITAI NA DIAGRAFEI
   LEITOYRGIA: DIAGRAFEI, THN EGGRAFH ME KLEIDI (DelKey) APO TH
               LISTA (List), AN YPARXEI ENHMERWNEI THN DOMH HList
   EPISTREFEI: THN TROPOPOIHMENH DOMH (HList)
   EJODOS:     "DEN YPARXEI EGGRAFH ME KLEIDI" MHNYMA
*/
{
        int Loc, Pred, HVal;

        SearchHashList(*HList,DelKey,&Loc,&Pred);
        if(Loc!=-1)
        {
                if(Pred!=-1)
                {
                        HList->List[Pred].Link=HList->List[Loc].Link;
                }
                else
                {
                        HVal=HashKey(DelKey);
                    HList->HashTable[HVal]=HList->List[Loc].Link;
                }
                HList->List[Loc].Link=HList->StackPtr;
                HList->StackPtr=Loc;
                HList->Size=HList->Size -1;
        }
        else
        {
                printf("DEN YPARXEI  EGGRAFH ME KLEIDI %s \n",DelKey); // Αλλαγή από %d
σε %s επειδή είναι string
        }
}

void PrintPinakes(HashListType HList)
{
    int i;
    printf("Hash table\n");
    for(i=0;i<HMax;i++)
        printf("%d, ",HList.HashTable[i]);

    printf("\nHash List\n");
    for(i=0;i<HList.Size;i++)
       printf("%d) %s, %d\n",i,HList.List[i].RecKey,HList.List[i].Link);
    printf("\n");
}
```