Ε.Α.Π./ΕΙΣΑΓΩΓΗ ΣΤΗΝ ΠΛΗΡΟΦΟΡΙΚΗ

4η ΓΡΑΠΤΗ ΕΡΓΑΣΙΑ

ΑΚΑΔΗΜΑΪΚΟΥ ΕΤΟΥΣ 2014-2015

3ος Τόμος

ΔΟΜΕΣ ΔΕΔΟΜΕΝΩΝ

15/3/2015

Ημερομηνία παράδοσης εργασίας: Παρασκευή 08/05/2015
Καταληκτική ημερομηνία παραλαβής: Τετάρτη 13/05/2015[1]
Ημερομηνία ανάρτησης ενδεικτικών λύσεων: Σάββατο 16/05/2015
Καταληκτική ημερομηνία αποστολής σχολίων στον φοιτητή: Κυριακή 31/05/2015

ΥΠΟΕΡΓΑΣΙΑ 1. (βαθμοί 20)

ΥΠΟΕΡΓΑΣΙΑ 2. (βαθμοί 25)

ΥΠΟΕΡΓΑΣΙΑ 3. (βαθμοί 25)

ΥΠΟΕΡΓΑΣΙΑ 4. (βαθμοί 30)

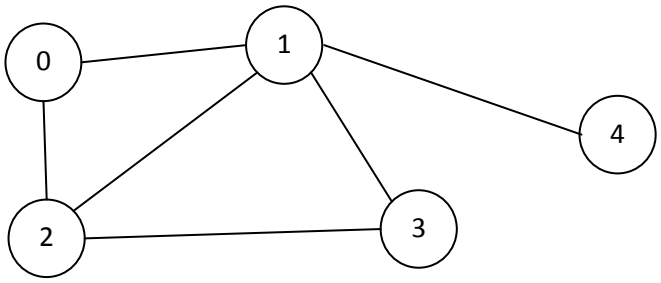
ΣΥΝΟΛΟ **(βαθμοί 100)**

[1] Σύμφωνα με τον Κανονισμό Σπουδών, η καταληκτική ημερομηνία για την παραλαβή της Γ.Ε. από το μέλος ΣΕΠ είναι η επόμενη Τετάρτη από το τέλος της εβδομάδας παράδοσης Γ.Ε.

ΥΠΟΕΡΓΑΣΙΑ 1. (βαθμοί 20)

Στην υποεργασία αυτή θα χρησιμοποιήσουμε έναν δισδιάστατο πίνακα για την αναπαράσταση της σχέσης φιλίας των χρηστών ενός κοινωνικού δικτύου. Έστω *n* ο αριθμός χρηστών. Ο κάθε χρήστης προσδιορίζεται από έναν μοναδικό κωδικό που λαμβάνει τιμές από 0 έως και *n*-1. Η σχέση φιλίας μπορεί να αναπαρασταθεί με έναν πίνακα *matrixF* διαστάσεων *n* x *n*, ο οποίος καλείται *πίνακας φιλίας*. Για παράδειγμα, εάν οι χρήστες *i* και *j* συνδέονται με σχέση φιλίας, τότε θα υπάρχει ο αριθμός 1 αποθηκευμένος στις θέσεις *matrixF*[*i*][*j*] *και matrixF*[*j*][*i*] του πίνακα *matrixF*. Στην αντίθετη περίπτωση, τα αντίστοιχα κελιά περιέχουν την τιμή 0.

Στη συνέχεια δίνουμε ένα παράδειγμα με πέντε χρήστες (*n* = 5), των οποίων η σχέση φιλίας δίνεται στο ακόλουθο σχήμα:



Ο πίνακας *matrixF* που αναπαριστά αυτές τις σχέσεις φιλίας είναι ο ακόλουθος:

	0	**1**	**2**	**3**	**4**
0	0	1	1	0	0
1	1	0	1	1	1
2	1	1	0	1	0
3	0	1	1	0	0
4	0	1	0	0	0

Ζητείται η υλοποίηση στη γλώσσα προγραμματισμού C των εξής συναρτήσεων:

1. `void loadMatrix(int **matrixF, int size)`

   Η συνάρτηση δέχεται ως ορίσματα τον πίνακα `matrixF` και έναν ακέραιο αριθμό `size` που δηλώνει το πλήθος των χρηστών. Στη συνέχεια, τυπώνει στην οθόνη όλα τα δυνατά ζεύγη χρηστών και για το καθένα ζητά από το χρήστη να δηλώσει εάν οι δύο χρήστες είναι φίλοι ή όχι και ενημερώνει κατάλληλα τον πίνακα `matrixF`.

2. `int findFriends(int **matrixF, int size, int user)`

   Η συνάρτηση δέχεται ως ορίσματα τον πίνακα matrixF, την τιμή size που δηλώνει το πλήθος των χρηστών και τον κωδικό user ενός χρήστη και επιστρέφει το πλήθος των φίλων του.

3. `int commonFriends(int **matrixF, int size, int user1, int user2)`

   Η συνάρτηση δέχεται ως ορίσματα τον πίνακα matrixF, έναν ακέραιο αριθμό size που δηλώνει το πλήθος των χρηστών και δύο κωδικούς χρηστών user1 και user2 και επιστρέφει το πλήθος των κοινών τους φίλων.

4. `void sortUsers(int **matrixF, int size, int *matrixS)`

   Η συνάρτηση δέχεται ως ορίσματα τον πίνακα matrixF και ένα μονοδιάστατο πίνακα matrixS μήκους size που δηλώνει το πλήθος των χρηστών. Στη συνέχεια γεμίζει τον πίνακα matrixS με το πλήθος των φίλων του κάθε χρήστη και ταξινομεί τον πίνακα matrixS σε μη φθίνουσα διάταξη χρησιμοποιώντας τον αλγόριθμο ταξινόμησης φυσαλίδας.

Σημείωση: Η έκφραση fun(int **F, int size) είναι ισοδύναμη με την έκφραση fun(F[][], int size)

Θα πρέπει να χρησιμοποιήσετε το σχέδιο προγράμματος με όνομα αρχείου `ypoergasia1_code_template.c` που θα βρείτε στον κατάλογο `code_templates` του συμπιεσμένου αρχείου της εκφώνησης της 4ης Γραπτής Εργασίας. Στο σχέδιο προγράμματος καλείστε να συμπληρώσετε τον κώδικα των τεσσάρων συναρτήσεων που περιγράφηκαν προηγουμένως.

ΥΠΟΕΡΓΑΣΙΑ 2. (βαθμοί 25)

Για την επιλογή ενός ατόμου από μια ομάδα μπορεί να χρησιμοποιηθεί η ακόλουθη μέθοδος: Τα άτομα σχηματίζουν κύκλο, με κάποιον να θεωρείται ότι βρίσκεται στην αρχή του κύκλου και επιλέγεται ένας θετικός ακέραιος αριθμός *k*. Κατόπιν ξεκινώντας από το άτομο στην αρχή του κύκλου, μετράμε και απομακρύνουμε από τον κύκλο το k-οστό άτομο που βρίσκουμε. Επαναλαμβάνουμε τη διαδικασία για τον (κατά ένα άτομο μικρότερο) κύκλο, ξεκινώντας τη μέτρηση από το άτομο που βρίσκεται στην επόμενη θέση μετά από αυτό που απομακρύνθηκε και απομακρύνουμε το νέο άτομο στο οποίο θα καταλήξουμε μετρώντας *k* θέσεις κατά την ίδια φορά. Συνεχίζουμε την ίδια διαδικασία, έως ότου απομείνει στον κύκλο μόνο ένα άτομο, το οποίο και επιλέγεται. Για παράδειγμα, εάν πέντε άτομα με ονόματα Ann, Bob, Claudia, Dan και Ed εφαρμόσουν την παραπάνω μέθοδο για *k* = 3 και θεωρώντας ότι η Ann είναι στην αρχή του κύκλου και οι υπόλοιποι με τη σειρά που αναφέρονται, τότε θα απομακρυνθούν κατά σειρά οι Claudia, Ann, Ed και Bob και επιλέγεται ο Dan.

Σε αυτή την υποεργασία ζητείται να υλοποιήσετε αυτή τη μέθοδο χρησιμοποιώντας μια *κυκλική* απλά διασυνδεδεμένη λίστα (δηλαδή, πέραν του ότι κάθε κόμβος δείχνει στον επόμενό του στη λίστα, ο τελευταίος κόμβος δείχνει στον πρώτο κόμβο της λίστας) με κόμβους τύπου `CListNode` που ορίζεται ως εξής:

```
typedef struct node {
     char name[10];   /* 9 χαρακτήρες και το '\0' */
     struct node *next;
} CListNode;
```

και δείκτη `list_end_ptr` που είναι NULL εάν η λίστα είναι κενή, ενώ δείχνει στον κόμβο στο τέλος της λίστας εάν η λίστα δεν είναι κενή. Ζητείται η υλοποίηση στη γλώσσα προγραμματισμού C των εξής συναρτήσεων:

1. `CListNode* insert_at_end(CListNode *end_ptr, char *a)`

   Η συνάρτηση δέχεται ως ορίσματα το δείκτη `end_ptr` στον τελευταίο κόμβο μιας κυκλικής απλά διασυνδεδεμένης λίστας με κόμβους τύπου `CListNode` και πίνακα `a`, 10 χαρακτήρων, και εισάγει στο τέλος της λίστας έναν κόμβο τύπου `CListNode` αποθηκεύοντας το περιεχόμενο του πίνακα `a` στο πεδίο `name`. Η συνάρτηση επιστρέφει δείκτη στον τελευταίο κόμβο της λίστας που προκύπτει.

2. `CListNode* initialize_list(int n)`

   Η συνάρτηση δέχεται ως όρισμα το πλήθος n των ατόμων της ομάδας, κατασκευάζει κυκλική απλά διασυνδεδεμένη λίστα με κόμβους τύπου CListNode και επιστρέφει δείκτη

στον τελευταίο κόμβο της τελικής λίστας. Η κατασκευή της λίστας γίνεται εισάγοντας για κάθε άτομο έναν κόμβο στο τέλος της λίστας, χρησιμοποιώντας τις συναρτήσεις insert_at_end και get_name (δίνεται στο template).

3. `CListNode* delete_next_node(CListNode *end_ptr, CListNode *p)`

   Η συνάρτηση δέχεται ως ορίσματα το δείκτη end_ptr στον τελευταίο κόμβο μιας κυκλικής απλά διασυνδεδεμένης λίστας με κόμβους τύπου CListNode και δείκτη p σε έναν κόμβο της λίστας, διαγράφει τον κόμβο που βρίσκεται αμέσως μετά από αυτόν που δείχνεται από τον δείκτη p και αποδεσμεύει το χώρο που καταλαμβάνει χρησιμοποιώντας τη συνάρτηση free (βιβλιοθήκη stdlib.h). Η συνάρτηση επιστρέφει δείκτη στον τελευταίο κόμβο της λίστας. Στην περίπτωση που ζητείται να διαγραφεί ο τελευταίος κόμβος της λίστας, επιστρέφεται δείκτης στον κόμβο πριν από αυτόν, ο οποίος και θα είναι τελευταίος στη λίστα μετά τη διαγραφή.

4. `void print_list(CListNode *end_ptr)`

   Η συνάρτηση δέχεται ως όρισμα το δείκτη end_ptr στον τελευταίο κόμβο μιας κυκλικής απλά διασυνδεδεμένης λίστας με κόμβους τύπου CListNode και εμφανίζει τα περιεχόμενα των πεδίων name των κόμβων της λίστας (από την αρχή της έως το τέλος της), εμφανίζοντάς τα στοιχισμένα και ανά 6 σε κάθε γραμμή εκτός ίσως από την τελευταία γραμμή. Εάν η λίστα είναι κενή εμφανίζεται κατάλληλο μήνυμα.

5. `CListNode* select(CListNode *end_ptr, int k)`

   Η συνάρτηση δέχεται ως ορίσματα το δείκτη end_ptr στον τελευταίο κόμβο μιας κυκλικής απλά διασυνδεδεμένης λίστας με κόμβους τύπου CListNode και έναν θετικό ακέραιο αριθμό k και εφαρμόζει τη μέθοδο επιλογής που περιγράψαμε για τη δοθείσα λίστα και τον αριθμό k, χρησιμοποιώντας τη συνάρτηση delete_next_node. Η συνάρτηση επαναληπτικά διαγράφει κόμβους, τυπώνοντας τα περιεχόμενα της λίστας μετά από κάθε διαγραφή (χρησιμοποιώντας τη συνάρτηση print_list), έως ότου απομείνει μόνον ένας κόμβος στη λίστα, οπότε και επιστρέφει δείκτη σε αυτόν τον κόμβο.

Θα πρέπει να χρησιμοποιήσετε το σχέδιο προγράμματος με όνομα αρχείου `ypoergasia2_code_template.c` που θα βρείτε στον κατάλογο `code_templates` του συμπιεσμένου αρχείου της εκφώνησης της 4ης Γραπτής Εργασίας. Στο σχέδιο προγράμματος καλείστε να συμπληρώσετε τον κώδικα των πέντε συναρτήσεων που περιγράφηκαν προηγουμένως.

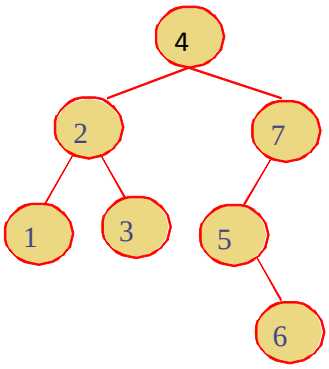
ΥΠΟΕΡΓΑΣΙΑ 3. (βαθμοί 25)

Στην εργασία αυτή θα ασχοληθούμε με ένα παιχνίδι δύο παικτών PRE και POST, οι οποίοι επιχειρούν να κτίσουν ένα δυαδικό δέντρο αναζήτησης προσπαθώντας, όμως, ο καθένας να μεγιστοποιήσει το δικό του κέρδος. Πιο συγκεκριμένα, στους δύο παίκτες είναι διαθέσιμοι οι αριθμοί από το 1 έως και το *n*. Ξεκινά ο παίκτης PRE επιλέγοντας έναν αριθμό και εισάγοντάς τον στο, αρχικά κενό, δυαδικό δέντρο αναζήτησης. Ακολουθεί ο παίκτης POST ο οποίος επιλέγει έναν από τους εναπομείναντες αριθμούς και τον εισάγει με τη σειρά του στο δυαδικό δέντρο αναζήτησης. Οι παίκτες συνεχίζουν, ο καθένας με τη σειρά του, επιλέγοντας και εισάγοντας στο δέντρο έναν από τους εναπομείναντες αριθμούς.

Μόλις εξαντληθούν οι αριθμοί, ο παίκτης PRE εκτελεί μία προδιατεταγμένη (preorder) διαπέραση στο δέντρο που σχηματίστηκε και μετρά πόσοι από τους αριθμούς βρίσκονται στη σωστή τους θέση. Ένας αριθμός *k* βρίσκεται στη σωστή του θέση αν η σειρά του στη διαπέραση είναι *k*. Το αποτέλεσμα της μέτρησης αυτής είναι η βαθμολογία του παίκτη PRE. Ομοίως, ο παίκτης POST εκτελεί τη μεταδιατεταγμένη (postorder) διαπέραση και μετρά πόσοι από τους αριθμούς βρίσκονται στη σωστή τους θέση. Το αποτέλεσμα της μέτρησης αυτής είναι η βαθμολογία για τον παίκτη POST.

Για παράδειγμα, για *n* = *7*, οι διαθέσιμοι αριθμοί στους δύο παίκτες είναι 1, 2, 3, 4, 5, 6, και 7. Ακολουθεί ένα πιθανό παιχνίδι μεταξύ του PRE και του POST (όπου πρώτος ξεκινά ο PRE), μαζί με το δένδρο που σχηματίζεται και τις βαθμολογίες των δύο παικτών:

PRE	POST
4	7
2	5
3	1
6	



Προδιατεταγμένη διαπέραση (παίκτης PRE): 4, 2, 1, 3, 7, 5, 6

Βαθμολογία παίκτη PRE: 1 (επειδή μόνο ο αριθμός 2 είναι στη σωστή του θέση)

Μεταδιατεταγμένη διαπέραση (παίκτης POST): 1, 3, 2, 6, 5, 7, 4

Βαθμολογία παίκτη POST: 2 (οι αριθμοί 1 και 5 είναι στις σωστές θέσεις)

Θεωρώντας ότι $n = 9$, απαντήστε στα ακόλουθα ερωτήματα:

1. Δοθέντος του παιχνιδιού

PRE	**POST**
5	2
7	3
6	4
8	9
1	

   κατασκευάστε το δυαδικό δέντρο αναζήτησης, το οποίο προκύπτει από τις επιλογές των δύο παικτών και υπολογίστε τις βαθμολογίες τους.

2. Μετά το τέλος ενός παιχνιδιού, η προδιατεταγμένη διαπέραση είναι η εξής: 4, 2, 1, 3, 7, 6, 5, 9, 8. Να γράψετε τις κινήσεις που έχουν γίνει από τον παίκτη PRE (μπορεί να υπάρχουν περισσότερες της μιας λύσεις) στον επόμενο πίνακα με βάση τη διαπέραση αυτή, δοθέντων των κινήσεων που έπαιξε ο POST, και να υπολογίσετε τη βαθμολογία του:

PRE	**POST**
	7
	9
	3
8	5

3. Ποια είναι η μέγιστη βαθμολογία του παίκτη PRE και ποια του POST; Καταγράψτε ένα παιχνίδι στο οποίο ο παίκτης PRE επιτυγχάνει τη μέγιστη βαθμολογία του και ένα άλλο στο οποίο ο παίκτης POST επιτυγχάνει τη μέγιστη βαθμολογία του.

ΥΠΟΕΡΓΑΣΙΑ 4. (βαθμοί 30)

Μία γεννήτρια ψευδοτυχαίων φυσικών αριθμών είναι μία συνάρτηση που παράγει μία ακολουθία R_0, R_1, R_2, … φυσικών αριθμών με χαρακτηριστικά *τυχαιότητας* σύμφωνα με κάποια *κριτήρια*. Στη γενική της μορφή, μία τέτοια γεννήτρια έχει τη μορφή $R_i = f(R_{i-1})$ για $i \geq 0$, με R_{-1} μία αρχική τιμή (που δεν θεωρείται μέλος της ακολουθίας) και f μια ακέραια συνάρτηση. Στην πράξη επιθυμούμε οι αριθμοί να είναι μικρότεροι από ένα άνω όριο, δηλαδή $0 \leq R_i \leq m$, οπότε ξεκινούμε με την επιλογή ενός κατάλληλου m και μιας συνάρτησης $f(x)$ τέτοιας ώστε $0 \leq f(x) < m$, όπου $0 \leq x < m$.

Μια γεννήτρια βασισμένη σε μία τέτοια συνάρτηση γίνεται, τελικά, περιοδική, δηλαδή από ένα σημείο και μετά επαναλαμβάνει την ίδια σειρά αριθμών. Αυτό, σε μαθηματική διατύπωση, σημαίνει ότι υπάρχουν δύο αριθμοί μ και λ, τέτοιοι ώστε οι τιμές της ακολουθίας R_0, R_1, …, R_μ, …, $R_{\mu+\lambda-1}$ να είναι διαφορετικές, ενώ η αμέσως επόμενη $R_{\mu+\lambda}$ να είναι ίση με την τιμή R_μ. Είναι φανερό ότι από το σημείο αυτό και μετά από τη γεννήτρια παράγονται συνεχώς οι ίδιες τιμές R_μ, …, $R_{\mu+\lambda-1}$. Καλούμε το λ *περίοδο* της γεννήτριας αυτής. Μία ευρέως χρησιμοποιούμενη κλάση συναρτήσεων για την παραγωγή ψευδοτυχαίων αριθμών είναι οι *γραμμικές γεννήτριες υπολοίπων* με γενικό τύπο $f(x)=(ax+c) \bmod m$, όπου $0 \leq a,c,x < m$ και $m > 0$. Για παράδειγμα, για $a = 3$, $c = 4$, $m = 10$, $R_{-1} = 5$ με τη συνάρτηση αυτή παίρνουμε την ακολουθία 9, 1, 7, 5, 9, 1, 7, 5, 9, … με περίοδο 4 (επαληθεύστε το). Φυσικά, για διαφορετικές τιμές των παραμέτρων λαμβάνουμε και διαφορετικές ακολουθίες αριθμών με διαφορετικές περιόδους.

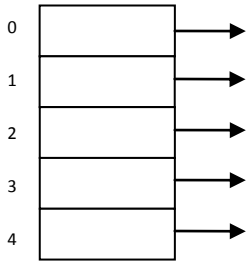
Αντικείμενο της υποεργασίας αυτής είναι η εύρεση της περιόδου μιας γεννήτριας ψευδοτυχαίων αριθμών. Πιο συγκεκριμένα, ζητείται να υλοποιηθεί αλγόριθμος που, δοθείσας μιας γραμμικής γεννήτριας υπολοίπων, παράγει διαδοχικούς όρους της ακολουθίας και μόλις βρει την πρώτη τιμή που είναι ίση με κάποια προηγούμενη τιμή σταματά και επιστρέφει τον αριθμό των βημάτων που μεσολάβησαν μεταξύ των δύο αυτών ίδιων τιμών. Μια βασική λειτουργία στην τεχνική αυτή είναι η διαπίστωση αν η τρέχουσα τιμή της ακολουθίας είναι ίση με μία από τις τιμές που έχουν εμφανιστεί προηγουμένως.

Μία αποδοτική λύση είναι η διατήρηση των όρων της ακολουθίας σε έναν *πίνακα κατακερματισμού* (hash table). Ένας πίνακας κατακερματισμού είναι μία δομή δεδομένων, η οποία χρησιμοποιείται για τη διαχείριση ενός συνόλου στοιχείων (π.χ., ακέραιων αριθμών, σειρών χαρακτήρων κλπ.), υποστηρίζοντας αποδοτικά τις λειτουργίες της αναζήτησης, εισαγωγής και διαγραφής. Στη συνέχεια περιγράφεται μία από τις γνωστότερες μεθόδους υλοποίησης του πίνακα κατακερματισμού, η οποία αναφέρεται ως *κατακερματισμός με αλυσίδες*.

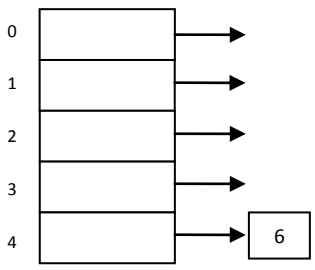
Υποθέτουμε ότι το σύνολο που θέλουμε να οργανώσουμε αποτελείται από ακέραιους αριθμούς. Έστω *m* το μέγιστο πλήθος των αριθμών του συνόλου. Ο πίνακας κατακερματισμού είναι ένας μονοδιάστατος πίνακας μήκους *htsize*, όπου *htsize* < *m*. Εκτός του πίνακα, απαιτείται και μία συνάρτηση κατακερματισμού *h*(*x*), η οποία για έναν ακέραιο αριθμό *x* προσδιορίζει τη θέση του πίνακα, όπου ο αριθμός *x* θα αποθηκευθεί σε μία απλά διασυνδεδεμένη λίστα, στην οποία θα αποθηκευθούν επίσης όλοι οι αριθμοί για τους οποίους η συνάρτηση *h*(*x*) δίνει το ίδιο αποτέλεσμα (την ίδια θέση του πίνακα). Κάθε νέο στοιχείο αποθηκεύεται στο τέλος της λίστας.

Η αναζήτηση ενός στοιχείου *x* στον πίνακα κατακερματισμού πραγματοποιείται με παρόμοιο τρόπο: Αρχικά εφαρμόζεται η συνάρτηση *h*(*x*), η οποία επιστρέφει μία θέση μέσα στον πίνακα. Εάν η αλυσίδα (λίστα) που αντιστοιχεί στη θέση *h*(*x*) είναι άδεια, τότε αυτό σημαίνει ότι το στοιχείο που αναζητούμε δεν υπάρχει. Διαφορετικά, το στοιχείο πρέπει να αναζητηθεί στη διασυνδεδεμένη λίστα που αντιστοιχεί στη θέση *h*(*x*).

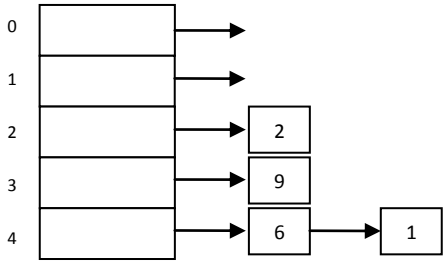
Έστω *S*={6,1,2,9,12,20,10,3} τα στοιχεία που θέλουμε να αποθηκεύσουμε σε έναν πίνακα κατακερματισμού με μέγεθος 5. Υποθέτουμε ότι η αρίθμηση των θέσεων του πίνακα αρχίζει από το 0. Επίσης, η συνάρτηση κατακερματισμού που θα χρησιμοποιήσουμε είναι *h*(*x*) = (3**x*+1) mod 5. Θα εισάγουμε τα στοιχεία σε έναν αρχικά κενό πίνακα κατακερματισμού με την ακόλουθη μορφή:



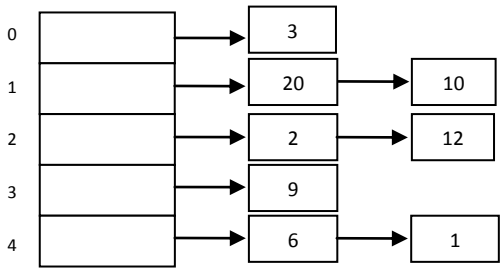
Στη συνέχεια θα εισάγουμε τα στοιχεία του *S* με τη σειρά που αναγράφονται, οπότε θα ξεκινήσουμε από το 6. Αρχικά, εφαρμόζουμε τη συνάρτηση *h*(*x*) για το στοιχείο *x*=6 και έχουμε *h*(6) = (3*6+1) mod 5 = 19 mod 5 = 4. Άρα, θα τοποθετήσουμε το στοιχείο 6 στην αλυσίδα που αντιστοιχεί στη θέση 4 του πίνακα.



Με εφαρμογή της ίδιας διαδικασίας εισάγουμε τα στοιχεία 1, 2 και 9 τα οποία αποθηκεύονται στις θέσεις $h(1) = 4$, $h(2) = 2$ και $h(9) = 3$, αντίστοιχα.



Ακολουθώντας την ίδια διαδικασία και για τα υπόλοιπα στοιχεία, λαμβάνουμε την τελική μορφή της δομής που δίνεται παρακάτω:



Η αναζήτηση του στοιχείου 12 θα πραγματοποιηθεί στην αλυσίδα $h(12) = 2$ και η αναζήτηση ολοκληρώνεται με επιτυχία, καθώς το στοιχείο υπάρχει στη λίστα. Αντιθέτως, η αναζήτηση του στοιχείου 7 είναι ανεπιτυχής, καθώς το στοιχείο δεν υπάρχει στην αλυσίδα που βρίσκεται στη θέση $h(7) = 2$. Σημειώνεται ότι σε κάθε αλυσίδα η αναζήτηση ενός στοιχείου x ξεκινά από το πρώτο στοιχείο της αλυσίδας και τα στοιχεία εξετάζονται το ένα μετά το άλλο ακολουθώντας τους αντίστοιχους δείκτες.

Ζητείται να γραφεί πρόγραμμα στη γλώσσα προγραμματισμού C, το οποίο να υπολογίζει την περίοδο μιας γραμμικής γεννήτριας υπολοίπων χρησιμοποιώντας την τεχνική του κατακερματισμού. Ως συνάρτηση κατακερματισμού να χρησιμοποιήσετε την $h(x) = x \bmod 500$.

Ο τύπος δεδομένων του κόμβου της κάθε λίστας είναι:

```
typedef struct node {
     int value;          // Η τιμή
     int counter;        // Η σειρά εισαγωγής
     struct node *next;  // Δείκτης στον επόμενο κόμβο
} t_listnode;
```

Ο πίνακας κατακερματισμού αποτελείται από κελιά που έχουν τον ακόλουθο τύπο δεδομένων:

```
typedef struct {
     t_listnode *head;
     t_listnode *tail;
} t_htentry;
```

όπου head είναι δείκτης στον πρώτο κόμβο της αντίστοιχης λίστας και tail είναι δείκτης στον τελευταίο κόμβο της αντίστοιχης λίστας.

Αναλυτικότερα, πρέπει να υλοποιηθούν οι εξής συναρτήσεις:

1. `int insertElement(int x, int counter, t_htentry *ht, int size)`

   Η συνάρτηση δέχεται ως ορίσματα έναν ακέραιο `x` που είναι το στοιχείο που εισάγεται, έναν ακέραιο counter που δηλώνει τη σειρά εισαγωγής του στοιχείου, τον πίνακα κατακερματισμού ht και το μέγεθος του πίνακα κατακερματισμού size. Εάν η εισαγωγή είναι επιτυχής, τότε η συνάρτηση επιστρέφει `1`, αλλιώς επιστρέφει `0`.

2. `int searchElement(int x, t_htentry *ht, int size)`

   Η συνάρτηση δέχεται ως ορίσματα έναν ακέραιο x που είναι το στοιχείο που αναζητείται, τον πίνακα κατακερματισμού ht και το μέγεθος του πίνακα κατακερματισμού size. Εάν το στοιχείο x υπάρχει στον πίνακα κατακερματισμού, τότε η συνάρτηση επιστρέφει την τιμή του μετρητή (counter) του x, διαφορετικά επιστρέφει 0.

3. `int generate(int a, int c, int m, int x)`

   Η συνάρτηση δέχεται ως ορίσματα τις τρεις παραμέτρους a, c και m, καθώς και μία τιμή x, και επιστρέφει την επόμενη τιμή σύμφωνα με τον τύπο της γεννήτριας. Για τους στόχους της εργασίας, χρησιμοποιήστε τις τιμές a = 3, c = 19, m = 15001 με αρχική τιμή την 27, οι οποίες θα πρέπει να δηλώνονται ως σταθερές.

4. `int findPeriod(int a, int c, int m, int x)`

   Η συνάρτηση δέχεται ως ορίσματα τις παραμέτρους a, c και m, καθώς και μια αρχική τιμή x (δείτε το προηγούμενο ερώτημα 3) και καλεί επαναληπτικά τη συνάρτηση generate() που παράγει την επιθυμητή ακολουθία αριθμών χρησιμοποιώντας τις συναρτήσεις insertElement(), searchElement() και τον πίνακα που αυτές χειρίζονται για να εντοπίσει την πρώτη επανάληψη τιμής. Η συνάρτηση επιστρέφει την περίοδο της ακολουθίας.

   Σημείωση: μέσα στη συνάρτηση αυτή θα πρέπει να δεσμευθεί χώρος για τη δημιουργία του πίνακα κατακερματισμού.

Θα πρέπει να χρησιμοποιήσετε το σχέδιο προγράμματος με όνομα αρχείου `ypoergasia4_code_template.c` που θα βρείτε στον κατάλογο `code_templates` του συμπιεσμένου αρχείου της εκφώνησης της 4ης Εργασίας. Στο σχέδιο προγράμματος καλείστε να συμπληρώσετε τον κώδικα των τεσσάρων συναρτήσεων που περιγράφηκαν προηγουμένως.

Γενικές Υποδείξεις:

I) Αμυντικός προγραμματισμός και σχολιασμός κώδικα

Ο κώδικας πρέπει να είναι καλά σχολιασμένος και να χρησιμοποιεί στοιχεία αμυντικού προγραμματισμού, όπου αυτό ζητείται. Ειδικά για τον αμυντικό προγραμματισμό, θα πρέπει ο κώδικας να εντοπίζει και να απορρίπτει μη αποδεκτές τιμές εισόδου από τον χρήστη (εμφανίζοντας σε αυτόν το ανάλογο μήνυμα) και να τον προτρέπει να εισάγει αποδεκτές τιμές. Για την εφαρμογή του αμυντικού προγραμματισμού αρκεί να γίνεται έλεγχος ως προς το εάν μία τιμή που εισάγεται από τον χρήστη του προγράμματος βρίσκεται εντός αποδεκτών ορίων (π.χ. να είναι θετική, διαφορετική από το 0, μεγαλύτερη από 20 κλπ.). Δεν απαιτείται έλεγχος ως προς το εάν η τιμή ανήκει στο σωστό τύπο δεδομένων (π.χ. ακέραιος, πραγματικός αριθμός, χαρακτήρας κλπ.) σύμφωνα με τον τύπο της μεταβλητής στην οποία θα αποθηκευτεί η τιμή αυτή.

II) Τρόπος παράδοσης εργασίας

α) Στα ερωτήματα που ζητείται υλοποίηση κώδικα στη γλώσσα προγραμματισμού C, για να θεωρηθούν οι απαντήσεις σας ολοκληρωμένες θα πρέπει:

- Ο κώδικας (όπου ζητείται) να είναι επαρκώς σχολιασμένος και ενσωματωμένος μέσα στο .doc αρχείο του Word με τις απαντήσεις σας καθώς και σε ξεχωριστό .c αρχείο (ANSI C).
- Το όνομα κάθε .c αρχείου να περιλαμβάνει το επώνυμό σας με λατινικούς χαρακτήρες, το χαρακτήρα της υπογράμμισης και τον αριθμό του συγκεκριμένου υποερωτήματος (π.χ. αν το επώνυμό σας είναι Γεωργίου, τότε ο κώδικας για την υποεργασία 1Β θα έχει το όνομα Georgiou_1b.c).
- Κάθε αρχείο C (ANSI C) που θα παραδοθεί θα πρέπει τουλάχιστον να περνάει τη φάση της μεταγλώττισης χωρίς λάθη.

β) Οι απαντήσεις πρέπει να είναι γραμμένες με χρήση **επεξεργαστή κειμένου** (π.χ. **Word**) σε σελίδες **διαστάσεων Α4 χωρίς χρώματα.** Το αρχείο να περιέχει ως **πρώτη σελίδα** το κείμενο του **Εντύπου Υποβολής - Αξιολόγησης** και τις απαντήσεις σας στη συνέχεια, **χωρίς να επαναλαμβάνονται οι εκφωνήσεις των υποεργασιών.**

γ) Τα .c αρχεία με τον πηγαίο κώδικα και το .doc αρχείο κειμένου να υποβληθούν στη διεύθυνση **http://study.eap.gr** όλα μαζί **σε συμπιεσμένη μορφή σε ένα αρχείο τύπου .zip**, με **όνομα αρχείου το επώνυμό σας με λατινικούς χαρακτήρες και τον Αριθμό Μητρώου σας**, π.χ. Ioannou_82345.zip.

****