Εργασία 1η - Λίστες & Δέντρα

Τελική Ημερομηνία Υποβολής - 27.10.2019

Εισαγωγικά - ο τύπος δεδομένων type_t

Ως τύπος **type_t** μπορεί να οριστεί οποιοσδήποτε τύπος δεδομενων (π.χ. ένας ακέραιος, ένας πίνακας χαρακτήρων ή ένα **struct**). Για τις ανάγκες της παρούσας εργασίας ο τύπος **type_t** ορίζεται ισοδύναμος με το **struct student** (περιγράφεται στο αρχείο **type_t.h**) :

```
struct student {
    int aem;
    char name[32];
};

typedef struct student type_t;
```

Για το συγκεκριμένο τύπο δεδομένων ορίζονται οι συναρτήσεις **student_cmp** και **student_print** των οποίων η υλοποίηση βρίσκεται στο αρχείο **type_t.c**.

Σημείωση: Κατά τον έλεγχο της εργασίας μπορεί να δοκιμαστούν και διαφορετικοί τύποι δεδομένων για τον ορισμό του τύπου **type_t**, αλλάζοντας τα αρχεία **type_t.h** και **type_t.c**.

Μέρος 1ο - Διπλα συνδεδεμένη χωρίς τερματικούς κόμβους, μη κυκλική λίστα

Σε αυτή την ενότητα της εργασίας θα υλοποιήσετε την αποθήκευση οποιουδήποτε τύπου πληροφορίας (ορίζεται από τον τύπο δεδομένων **type_t**) σε μία διπλά συνδεδεμένη μη κυκλική λίστα, χωρίς τερματικούς κόμβους. Ο κόμβος της λίστας και η λίστα περιγράφονται από τις παρακάτω δομές:

τύπος δεδομένων	Περιγραφή

`typedef struct dlist_node {`
`  struct dlist_node* next;`
`  struct dlist_node* prev;`
`  type_t data;`
`} dnode_t;`
	Περιγράφει τον κόμβο της λίστας
	next: δείκτης προς τον επόμενο κόμβο
	prev: δείκτης προς τον προηγούμενο κόμβο
	
	data: η πληροφορία που αποθηκεύεται στο κόμβο

`typedef struct dlist {`
`  struct dlist_node *head, *tail;`
`  int size;`
 	
`  comparator_t cmp;`
`  printer_t printer;`
`} dlist_t;`
	Περιγράφει τη λίστα
	head: δείκτης προς τον 1ο κόμβο της λίστας (κεφαλή)
	tail: δείκτης προς τον τελευταίο κόμβο της λίστας (ουρά)
	size: αριθμός των στοιχείων που περιέχει η λίστα.
	
	cmp: συνάρτηση σύγκρισης
	printer: συνάρτηση εκτύπωσης

Οι τύποι δεδομένων **comparator_t**, **printer_t** είναι δείκτες σε συναρτήσεις που περιγράφονται παρακάτω:

`typedef int (*comparator_t)` `(type_t v1, type_t v2);`	Δείκτης σε συνάρτηση που συγκρίνει μεταξύ τους δύο αντικείμενα της λίστας. Ισχύουν τα εξής: 1. **v1 != v2**, επιστρέφει ακέραιο διαφορετικό του μηδέν (0). 2. **v1 == v2**, επιστρέφει μηδέν (0).

`typedef void (*printer_t)(FILE* fp, type_t data, bool only_key);`	Συνάρτηση εκτύπωσης του περιεχομένου ενός κόμβου που αφορά τον τύπο δεδομένων που αποθηκεύεται στη λίστα. Θεωρούμε ότι το **FILE* fp** είναι ανοιχτό για εγγραφή. Η παράμετρος `only_key` χρησιμεύει προκειμένου να επιλέξουμε να εκτυπώσουμε μόνο το κλειδί ή το σύνολο της πληροφορίας του κόμβου. (π.χ. για τον τύπο `type_t -> sturct student` επιλέγουμε να εκτυπώσουμε μόνο το ΑΕΜ που είναι το κλειδί ή το ΑΕΜ και το όνομα).

Για την παραπάνω λίστα θα πρέπει να υλοποιήσετε τις εξής μεθόδους στο αρχείο **dlist.c**:

Συνάρτηση	Περιγραφή
`dlist_t* create_dlist(comparator_t cmp, printer_t printer);`	Επιστρέφει ένα δείκτη σε μία αρχικά άδεια λίστα για την οποία έχουν προσδιοριστεί 1. η συνάρτηση σύγκρισης του περιεχομένου δύο κόμβων 2. και η συνάρτηση εκτύπωσης του περιεχομένου του κόμβου.
`void clear_dlist(dlist_t* list);`	Διαγράφεται το σύνολο των περιεχομένων της λίστας, αλλά όχι η λίστα.
`void destroy_dlist(dlist_t* list);`	Διαγράφεται το σύνολο των περιεχομένων της λίστας και η λίστα.
`int size(dlist_t* list);`	Επιστρέφει τον αριθμό των κόμβων της λίστας (το μέγεθος της λίστας).
`bool insert(dlist_t* list, int index, type_t data);`	Ενθέτει την τιμή **data** στη θέση **index** της λίστας. Επιστρέφει `true` εάν η τιμή **index** είναι είναι μεταξύ **0** και **size.** Διαφορετικά αποτυγχάνει και επιστρέφει `false`. Η τιμή **size** αντιπροσωπεύει την ένθεση στο τέλος της λίστας, ενώ η τιμή 0 την ένθεση στην αρχή.
`int index_of(dlist_t* list, type_t data);`	Επιστρέφει τη θέση της πρώτης εμφάνισης της τιμής **data** στη λίστα ή **-1** εάν η τιμή **data** δεν υπάρχει.
`int instances_of(dlist_t* list, type_t data);`	Επιστρέφει τον αριθμό των εμφανίσεων της τιμής **data** στη λίστα.
`type_t get_index(dlist_t* list, int index);`	Επιστρέφει το περιεχόμενο του κόμβου στη θέση **index** ή **μηδέν** εάν η τιμή **index** δεν είναι είναι μεταξύ **0** και **size-1.**
`bool rmv(dlist_t* list, type_t data);`	Διαγράφει τον πρώτο κόμβο που περιέχει τη μεταβλητή **data**. Επιστρέφει **true** εάν η διαγραφή έγινε με επιτυχία, διαφορετικά **false**.
`type_t rmv_index(dlist_t* list, int index);`	Διαγράφει τον κόμβο που βρίσκεται στη θέση `index` της λίστας. Επιστρέφει το περιεχόμενο του κόμβου στη θέση **index** ή **μηδέν** εάν η τιμή **index** δεν είναι είναι μεταξύ **0** και **size-1.**
`void swap(dnode_t* n1, dnode_t* n2);`	Αντιμεταθέτει τα περιεχόμενα των κόμβων `n1, n2`.
`void catenate(dlist_t* list1, dlist_t* list2);`	Συνεννώνονται οι δύο λίστες. Μετά την κλίση της συνάρτησης η `list1`, περιέχει μετά το τέλος της και τα περιεχόμενα της `list2`, ενώ η `list2` είναι κενή.

Σημείωση: Η τιμή επιστροφής **0** για τον οποιονδήποτε τύπο **type_t** μπορεί να οριστεί ως εξής:

```
type_t zero_value = {0};
return zero_value;
```

Επιπλέον σας παρέχεται έτοιμη η υλοποίηση των παρακάτω συναρτήσεων.

Συνάρτηση	Περιγραφή
`void print(FILE* fp, dlist_t* list);`	Υποθέτοντας ότι ο δείκτης **fp** είναι ανοιχτός για εγγραφή, γράφει στο **fp** το περιεχόμενο της λίστας με σειρά διάτρεξης από το **head** προς το **tail**. Εκτυπώνονται μόνο τα κλειδιά.
`type_t* dlist2table(dlist_t* list);`	Δεσμεύει τον απαραίτητο χώρο στη μνήμη για ένα πίνακα τύπου **type_t** μεγέθους ίσου με τον αριθμό των στοιχείων της λίστας. Μεταφέρεται το περιεχόμενο κάθε κόμβου της λίστας στην αντίστοιχη θέση του πίνακα. Μετά την κλήση της συνάρτησης η λίστα **dlist2table** είναι άδεια.
`void table2dlist(type_t table[], int table_size, dlist_t* list);`	Ενθέτει με τη σειρά τα στοιχεία του πίνακα στη λίστα. Μετά την ένθεση ο πίνακας **table** αποδεσμεύεται.

Έλεγχος του κώδικα σας

Τα test που ελέγχουν τις παραπάνω συναρτήσεις είναι τα εξής:

test	Περιγραφή
`test1 (dlist_t1.h)`	Ελέγχει τη συνάρτηση **insert**.
`test2 (dlist_t2.h)`	Έλεγχος των συναρτήσεων **index_of** και **instances_of**
`test3 (dlist_t3.h)`	Έλεγχος των συναρτήσεων **index_of** και **get_index**
`test4 (dlist_t4.h)`	Έλεγχος των συναρτήσεων **rmv** και **rmv_index**
`test5 (dlist_t5.h)`	Έλεγχος των συναρτήσεων **catenate** και **swap**

Μέρος 2ο - Διπλο-ουρά (Double-ended queue)

Σε αυτή την ενότητα της εργασίας θα υλοποιήσετε την αποθήκευση οποιουδήποτε τύπου πληροφορίας (ορίζεται από τον τύπο δεδομένων **type_t**) σε μία διπλο-ουρά η οποία υλοποιείται εσωτερικά μέσω μιας διπλά συνδεδεμένης λίστας. Η ουρά περιγράφεται από την παρακάτω δομή (λεπτομέρειες στο αρχείο **dequeue.h**):

Τύπος δεδομένων	Περιγραφή

`typedef struct {`
`  dlist_t* list;`
`} dequeue_t;`
	`Περιγράφει την ουρά`
	`list: δείκτης προς μία διπλά συνδεδεμένη λίστα (περιγράφεται στην ενότητα 1 της παρούσας εργασίας).`

Για την παραπάνω διπλο-ουρά θα πρέπει να υλοποιήσετε στο αρχείο **dequeue.c** τις εξής μεθόδους:

Συνάρτηση	Περιγραφή
`dequeue_t* create_dequeue(printer_t printer);`	Επιστρέφει ένα δείκτη σε μία αρχικά άδεια ουρά για την οποία έχει προσδιοριστεί η συνάρτηση εκτύπωσης μιας καταχώρησης της ουράς τύπου **type_t**.
`void clear_dequeue(dequeue_t* queue);`	Διαγράφεται το σύνολο των περιεχομένων της ουράς, αλλά όχι η ουρά.
`void destroy_dequeue(dequeue_t* queue);`	Διαγράφεται το σύνολο των περιεχομένων της ουράς και η ουρά.
`int dequeue_size(dequeue_t* queue);`	Επιστρέφει τον αριθμό των στοιχείων της ουράς.
`void push_front(dequeue_t* queue, type_t data);`	Ενθέτει την τιμή **data** στην αρχή της ουράς.
`void push_back(dequeue_t* queue, type_t data);`	Ενθέτει την τιμή **data** στο τέλος της ουράς.
`type_t pop_front(dequeue_t* queue);`	Αφαιρεί την 1η εγγραφή από την αρχή της ουράς και την επιστρέφει
`type_t pop_back(dequeue_t* queue);`	Αφαιρεί την τελευταία εγγραφή από το τέλος της ουράς και την επιστρέφει
`type_t top_front(dequeue_t* queue);`	Επιστρέφει την 1η εγγραφή από την αρχή της ουράς, χωρίς να την αφαιρέσει.
`type_t top_back(dequeue_t* queue);`	Επιστρέφει την τελευταία εγγραφή από το τέλος της ουράς, χωρίς να την αφαιρέσει.
`void print_dequeue(FILE* fp, dequeue_t* queue);`	Εκτυπώνει το σύνολο της ουράς. Χρησιμοποιεί τη συνάρτηση εκτύπωσης της περιεχόμενης λίστας.

Έλεγχος του κώδικα σας

Τα test που ελέγχουν τις παραπάνω συναρτήσεις είναι τα εξής:

test	Περιγραφή
`test1 (dequeue_t1.h)`	Ελέγχει τις συναρτήσεις **push_front, push_back** και τη συναρτηση εκτύπωσης **print_dequeue**.
`test2 (dequeue_t2.h)`	Ελέγχει τις συναρτήσεις **top_front, top_back, pop_fron** και **pop_back**.

Μέρος 3ο - Αζύγιστο δυαδικό δέντρο

Σε αυτή την ενότητα της εργασίας θα υλοποιήσετε την αποθήκευση οποιουδήποτε τύπου πληροφορίας (ορίζεται από τον τύπο δεδομένων **type_t**) σε ένα αζύγιστο δυαδικό δέντρο αναζήτησης. Ο κόμβος του δέντρου και το δέντρο περιέχονται στο αρχείο **tree.h** και επεξηγούνται παρακάτω:

τύπος δεδομένων	Περιγραφή

`typedef struct bt_node {`
`  struct bt_node* parent;`
`  struct bt_node* left;`
`  struct bt_node* right;`
`  type_t data;`
`} node_t;`
	`Περιγράφει τον κόμβο του δέντρου`
	`parent: δείκτης προς τον πατέρα του κόμβου`
	`left: δείκτης προς το αριστερό παιδί`
	`right: δείκτης προς το δεξί παιδί`
	
	`data: η πληροφορία που αποθηκεύεται στο κόμβο`

`typedef struct btree {`
`  node_t* root;`
`  int size;`
 	
`  comparator_t cmp;`
`  printer_t printer;`
`} tree_t;`
	`Περιγράφει το δέντρο`
	`root: δείκτης προς τον κόμβο "ρίζα" του δέντρου`
	`size: αριθμός των στοιχείων που περιέχει η λίστα.`
	
	`cmp: συνάρτηση σύγκρισης`
	`printer: συνάρτηση εκτύπωσης`

Οι τύποι δεδομένων **comparator_t**, **printer_t** είναι δείκτες σε συναρτήσεις που περιγράφονται παρακάτω:

`typedef int (*comparator_t)` `(type_t v1, type_t v2);`	Δείκτης σε συνάρτηση που συγκρίνει μεταξύ τους δύο αντικείμενα του δέντρου. Ισχύουν τα εξής: 1. **v1 > v2**, επιστρέφει θετικό ακέραιο. 2. **v1 < v2**, επιστρέφει αρνητικό ακέραιο. 3. **v1 == v2**, επιστρέφει μηδέν (0).
`typedef void (*printer_t)(FILE* fp, type_t data, bool only_key);`	Συνάρτηση εκτύπωσης του περιεχομένου ενός κόμβου που αφορά τον τύπο δεδομένων που αποθηκεύεται στο δέντρο. Θεωρούμε ότι το **FILE* fp** είναι ανοιχτό για εγγραφή. Η παράμετρος **only_key** χρησιμεύει προκειμένου να επιλέξουμε να εκτυπώσουμε μόνο το κλειδί ή το σύνολο της πληροφορίας του κόμβου.

Για το παραπάνω δέντρο θα πρέπει να υλοποιήσετε τις εξής μεθόδους στο αρχείο **tree.c**:

Συνάρτηση	Περιγραφή
`tree_t* create_tree(comparator_t cmp, printer_t printer);`	Επιστρέφει ένα δείκτη σε ένα αρχικά άδειο δέντρο για το οποίο προσδιορίζεται η συνάρτηση σύγκρισης και η συνάρτηση εκτύπωσης
`void clear_tree(tree_t* list);`	Διαγράφεται το σύνολο των περιεχομένων του δέντρου, αλλά όχι το δέντρο.
`void destroy_tree(tree_t* list);`	Διαγράφεται το σύνολο των περιεχομένων του δέντρου και το δέντρο.
`int size(tree_t* tree);`	Επιστρέφει τον αριθμό των κόμβων του δέντρου.
`node_t* get_sibling(node_t* node);`	Επιστρέφει τον αδελφό του κόμβου, εφόσον αυτός υπάρχει διαφορετικά **NULL**.
`bool is_left_child(node_t* node);`	Επιστρέφει **true** εάν ο κόμβος είναι αριστερό παιδί, διαφορετικά **false**. Εάν ο κόμβος είναι ρίζα του δέντρου

	επιστρέφει **false**.
`bool is_right_child(node_t* node);`	Επιστρέφει **true** εάν ο κόμβος είναι δεξί παιδί, διαφορετικά **false**. Εάν ο κόμβος είναι ρίζα του δέντρου επιστρέφει **false**.
`void swap(node_t* n1, node_t* n2);`	Αντιμεταθέτει τα περιεχόμενα (δείκτης data), των κόμβων στους οποίους δείχνουν οι δείκτες **n1**, **n2**.
`bool contains(tree_t* tree, type_t data);`	Επιστρέφει **true** εάν το δέντρο περιέχει τη συγκεκριμένη τιμή **data**. Διαφορετικά **false**.
`type_t find(tree_t* tree, type_t data);`	Αναζητεί τον κόμβο που περιέχει το κλειδί που δίνεται στην παράμετρο **data**. Εφόσον υπάρχει στο δέντρο τέτοιος κόμβος, επιστρέφει το περιεχόμενο του. Εάν δεν υπάρχει επιστρέφει **0**.
`bool add(tree_t* tree, type_t data);`	Εφόσον δεν υπάρχει κόμβος με τη συγκεκριμένη τιμή, εισάγει ένα νέο κόμβο στο δέντρο, με την τιμή **data**. Επιστρέφει **true** εάν η εισαγωγή έγινε με επιτυχία, διαφορετικά **false**.
`bool update(tree_t* tree, type_t data, type_t new_data);`	Εφόσον υπάρχει κόμβος με την τιμή κλειδιού **data**, τότε ανανεώνει τα περιεχόμενα του κόμβου με την τιμή **new_data** και επιστρέφει **true**. Εάν κόμβος δεν υπάρχει επιστρέφει **false**.
`bool rmv(tree_t* tree, type_t data);`	Αναζητεί τον κόμβο με τιμή **data**. Εφόσον τον βρει τον διαγράφει, απελευθερώνοντας παράλληλα τη μνήμη που έχει δεσμευτεί για το περιεχόμενο του κόμβου. Επιστρέφει **true** σε περίπτωση επιτυχίας, διαφορετικά **false**. **Σημείωση:** Η διαγραφή γίνεται ΥΠΟΧΡΕΩΤΙΚΑ αντιμεταθέτοντας το περιεχόμενου του κόμβου προς διαγραφή με το περιεχόμενο του αριστερότερου κόμβου του δεξιού υποδέντρου.

Σημείωση: Η τιμή επιστροφής **0** για τον οποιονδήποτε τύπο **type_t** μπορεί να οριστεί ως εξής:

```
type_t zero_value = {0};
return zero_value;
```

Επιπλέον σας παρέχεται έτοιμη η υλοποίηση των παρακάτω συναρτήσεων.

Συνάρτηση	Περιγραφή
`void print(FILE* fp, tree_t* tree);`	Υποθέτοντας ότι ο δείκτης **fp** είναι ανοιχτός για εγγραφή, γράφει το περιεχόμενο του δέντρου με σειρά διάτρεξης **pre-order**. Εκτυπώνονται μόνο τα κλειδιά.
`void dot_print(FILE* fp, tree_t* tree);`	Υποθέτοντας ότι ο δείκτης **fp** είναι ανοιχτός για εγγραφή, γράφει το περιεχόμενο του δέντρου με σειρά διάτρεξης **pre-order**, ώστε το αρχείο που θα προκύψει να μπορεί να εκτυπωθεί από το πρόγραμμα dot. Εσωτερικά χρησιμοποιεί το δείκτη σε συνάρτηση **printer** της μεταβλητής **tree**. Εκτυπώνονται μόνο τα κλειδιά.

Έλεγχος του κώδικα σας

Τα test που ελέγχουν τις παραπάνω συναρτήσεις είναι τα εξής:

test	Περιγραφή
`test1 (tree_t1.h)`	Έλεγχος της συνάρτησης **add**.
`test2 (tree_t2.h)`	Έλεγχος της συνάρτησης **contains**.
`test3 (tree_t3.h)`	Έλεγχος της συνάρτησης **update**.
`test4 (tree_t4.h)`	Έλεγχος της συνάρτησης **rmv**.

Για να ζωγραφίσετε το δέντρο (για debugging)

1. Στο **Makefile** που σας έχει δοθεί αντικαθιστάτε στις εντολές εκτέλεσης του κώδικα για το δέντρο **rt1** και **rt4** την παράμετρο autolab με την παράμετρο dot. Δεν αφορά τις περιπτώσεις **rt2** και **rt3** διότι γίνεται αναζήτηση και αντικατάσταση και δεν μεταβάλλεται η δομή του δέντρου.
2. Εγκαθιστάτε στον υπολογιστή σας το πακέτο graphviz μέσω της εντολής: **$> sudo apt-get install graphviz**
3. Κατά την εκτέλεση του κώδικα σας δεν θα δουλεύουν σωστά οι εντολές diff, αλλά θα ζωγραφίζονται τα στιγμιότυπα του δέντρου μέσα στον τοπικό φάκελο **dot/**.

Τρόπος Αποστολής

Η αποστολή της εργασία θα γίνει μέσω της πλατφόρμας autolab (απαιτείται συνδεση VPN). Ακολουθήστε τα εξής βήματα:

1. Φτιάξτε ένα φάκελο με όνομα **`hw1submit`** και αντιγράψτε μέσα εκεί τα αρχεία **dlist.c, dequeue.c, tree.c**.
2. Συμπιέστε ως tar.gz. Σε Linux/KDE πάτε πάνω στο φάκελο, κάνετε δεξί κλικ και επιλέγετε **Compress -> Here (as tar.gz).** Δημιουργείται το αρχείο **hw1submit.tar.gz**.
3. Συνδέεστε στη διεύθυνση **https://autolab.e-ce.uth.gr** και επιλέγετε το μάθημα **CE210_2019-2020 (F19)** και από αυτό την εργασία **HW1**.
4. **Μόνο για όσους θα δουλέψουν σε ομάδες των δύο ατόμων:** Επιλέγετε Group options για να δημιουργήσετε το group. Συμπληρώνετε το όνομα το group και το e-mail του/της συνεργάτη σας. Πατάτε create group. Ο συνεργάτης σας θα πρέπει με τη σειρά του να κάνει login, να επιλέξει με τη σειρά του την επιλογή Group options και να αποδεχτεί τη δημιουργία του group.
5. Για να υποβάλετε την εργασία σας κάνετε click στην επιλογή **"I affirm that I have compiled with this course academic integrity policy…"** και πατάτε **submit**. Στη συνέχεια επιλέγετε το αρχείο **hw1submit.tar.g**z που δημιουργήσατε στο βήμα 2.