

Δομές Δεδομένων

Υλοποίηση Ουράς σε γλώσσα C

Δημήτρης Μιχαήλ



Τμήμα Πληροφορικής και Τηλεματικής
Χαροκόπειο Πανεπιστήμιο

Μία FIFO ουρά (queue) είναι ένας ΑΤΔ που περιλαμβάνει τις εξής λειτουργίες:

- Εισαγωγή ενός νέου στοιχείου, $PUSH(S,x)$.
- Διαγραφή του πιο παλιού στοιχείου που προστέθηκε στην ουρά, $POP(S)$.
- Έλεγχος κενής ουράς, $EMPTY(S)$.
- Επιστροφή μεγέθους ουράς, $SIZE(S)$.
- κ.τ.λ

Ουρά

Η ουρά ακολουθεί την αρχή **πρώτο μέσα, πρώτο έξω** ή αλλιώς FIFO (first-in, first-out)



Αφού ορίζουμε μία ουρά ως ΑΤΔ δεν μας ενδιαφέρει η ακριβής υλοποίηση της ουράς. Μπορούμε για παράδειγμα να υλοποιήσουμε μια ουρά με λίστα αλλά και με πίνακα.

Μερικές προϋποθέσεις:

- Ξεχωριστή υλοποίηση με διασύνδεση
- Οι χρήστες δεν επιτρέπεται να πειράζουν την εσωτερική αναπάσταση
- Οι χρήστες πρέπει να μπορούν να φτιάξουν πολλές ουρές ταυτόχρονα

Ουρά

διασύνδεση σε C (queue.h)

```
#ifndef _QUEUE_H
#define _QUEUE_H

typedef int item_type;
typedef struct _queue* queue;

queue queue_create();
void queue_destroy(queue q);

void queue_push(queue q, item_type elem);
item_type queue_pop(queue q);
item_type queue_first(queue q);

int queue_is_empty(queue q);
int queue_size(queue q);
void queue_clear(queue q);

#endif
```

Υλοποίηση Ουράς

Αναπαράσταση Ουράς (queue.c)

```
#include <stdio.h>
#include <stdlib.h>
#include "queue.h"

struct node {
    item_type data;
    struct node* next;
};

struct _queue {
    struct node* head;
    struct node* tail;
    int size;
};
```

Υλοποίηση Ουράς

Δημιουργία Ουράς (queue.c)

```
queue queue_create() {
    queue q = (queue) malloc(sizeof(struct _queue));
    if (q == NULL) {
        fprintf(stderr, "Insufficient memory to \
initialize queue.\n");
        abort();
    }

    q->head = NULL;
    q->tail = NULL;
    q->size = 0;

    return q;
}
```

Υλοποίηση Ουράς

Καταστροφή Ουράς (queue.c)

```
void queue_destroy(queue q) {  
    if (q == NULL) {  
        fprintf(stderr, "Cannot destroy queue\n");  
        abort();  
    }  
  
    queue_clear(q);  
    free(q);  
}
```


Υλοποίηση Ουράς

Push (queue.c)

```
void queue_push(queue q, item_type elem) {
    struct node* n;
    n = (struct node*) malloc(sizeof(struct node));
    if (n == NULL) {
        fprintf(stderr, "Insufficient memory to \
create node.\n");
        abort();
    }

    n->data = elem;
    n->next = NULL;

    if (q->head == NULL) {
        q->head = q->tail = n;
    } else {
        q->tail->next = n;
        q->tail = n;
    }
    q->size += 1;
}
```

Υλοποίηση Ουράς

Pop (queue.c)

```
item_type queue_pop(queue q) {
    if (queue_is_empty(q)) {
        fprintf(stderr, "Can't pop element from queue: \
            queue is empty.\n");
        abort();
    }

    struct node* head = q->head;
    if (q->head == q->tail) {
        q->head = NULL;
        q->tail = NULL;
    } else {
        q->head = q->head->next;
    }
    q->size -= 1;

    item_type data = head->data;
    free(head);
    return data;
}
```

Υλοποίηση Ουράς

First (queue.c)

```
item_type queue_first(queue q) {  
    if (queue_is_empty(q)) {  
        fprintf(stderr, "Can't pop element from queue: \  
            queue is empty.\n");  
        abort();  
    }  
  
    return q->head->data;  
}
```

Υλοποίηση Ουράς

Empty και Size (queue.c)

```
int queue_is_empty(queue q) {
    if (q==NULL) {
        fprintf(stderr, "Cannot work with NULL queue.\n");
        abort();
    }

    return q->head == NULL;
}

int queue_size(queue q) {
    if (q==NULL) {
        fprintf(stderr, "Cannot work with NULL queue.\n");
        abort();
    }

    return q->size;
}
```

Υλοποίηση Ουράς

Clear (queue.c)

```
void queue_clear(queue q) {
    if (q==NULL) {
        fprintf(stderr, "Cannot work with NULL queue.\n");
        abort();
    }

    while(q->head != NULL) {
        struct node* tmp = q->head;
        q->head = q->head->next;
        free(tmp);
    }
    q->tail = NULL;
    q->size = 0;
}
```

Δοκιμάζοντας την Ουρά

```
#include <stdio.h>
#include <stdlib.h>
#include <assert.h>
#include "queue.h"

int main() {
    queue q;
    int i;

    q = queue_create();

    assert(queue_is_empty(q));

    for(i = 0; i < 10000; i++) {
        queue_push(q, i);
        assert(queue_first(q) == 0);
    }
    assert(queue_size(q) == 10000);
    assert(!queue_is_empty(q));

    for(i = 0; i < 10000; i++) {
        assert(queue_pop(q) == i);
    }
    assert(queue_is_empty(q));
    assert(queue_size(q) == 0);

    queue_destroy(q);
}
```