Δομές Δεδομένων

Ενότητα 3: Ουρές – Εισαγωγή-Υλοποίηση ΑΤΔ Ουρά με πίνακα

Καθηγήτρια Μαρία Σατρατζέμη

Τμήμα Εφαρμοσμένης Πληροφορικής

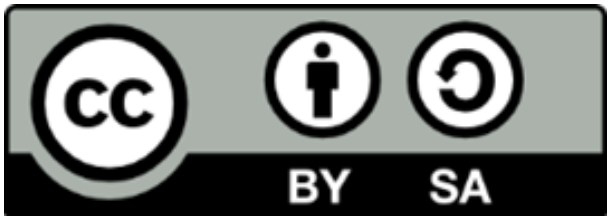
Ευρωπαϊκή Ένωση
Ευρωπαϊκό Κοινωνικό Ταμείο

ΕΠΙΧΕΙΡΗΣΙΑΚΟ ΠΡΟΓΡΑΜΜΑ
ΕΚΠΑΙΔΕΥΣΗ ΚΑΙ ΔΙΑ ΒΙΟΥ ΜΑΘΗΣΗ
επένδυση στην κοινωνία της γνώσης
ΥΠΟΥΡΓΕΙΟ ΠΑΙΔΕΙΑΣ ΚΑΙ ΘΡΗΣΚΕΥΜΑΤΩΝ
ΕΙΔΙΚΗ ΥΠΗΡΕΣΙΑ ΔΙΑΧΕΙΡΙΣΗΣ
Με τη συγχρηματοδότηση της Ελλάδας και της Ευρωπαϊκής Ένωσης

ΕΣΠΑ 2007-2013
πρόγραμμα για την ανάπτυξη
ΕΥΡΩΠΑΪΚΟ ΚΟΙΝΩΝΙΚΟ ΤΑΜΕΙΟ

Άδειες Χρήσης

- Το παρόν εκπαιδευτικό υλικό υπόκειται σε άδειες χρήσης Creative Commons.
- Για εκπαιδευτικό υλικό, όπως εικόνες, που υπόκειται σε άλλου τύπου άδειας χρήσης, η άδεια χρήσης αναφέρεται ρητώς.



Χρηματοδότηση

- Το παρόν εκπαιδευτικό υλικό έχει αναπτυχθεί στα πλαίσια του εκπαιδευτικού έργου του διδάσκοντα.
- Το έργο «Ανοικτά Ακαδημαϊκά Μαθήματα στο Πανεπιστήμιο Μακεδονίας» έχει χρηματοδοτήσει μόνο τη αναδιαμόρφωση του εκπαιδευτικού υλικού.
- Το έργο υλοποιείται στο πλαίσιο του Επιχειρησιακού Προγράμματος «Εκπαίδευση και Δια Βίου Μάθηση» και συγχρηματοδοτείται από την Ευρωπαϊκή Ένωση (Ευρωπαϊκό Κοινωνικό Ταμείο) και από εθνικούς πόρους.

Σκοποί ενότητας

Σκοπός της ενότητας είναι

- Να παρουσιάσει τον ΑΤΔ ουρά και της πράξεις που περιέχει καθώς και να δώσει ένα παράδειγμα χρήσης της ουράς.
- Να μελετηθεί ο ΑΤΔ ουρά η υλοποίηση της ουράς με πίνακα και χρήση δομής.

Περιεχόμενα ενότητας

- Εισαγωγή
- Βασικές λειτουργίες
- Ορισμός του ΑΤΔ ουρά
- Παράδειγμα Ουράς
- Χρήση ουράς για γραμμή αναμονής
- Παράδειγμα λειτουργίας ουράς
- Αποφυγή μετατοπίσεων
- Αποφυγή μετατοπίσεων: έλεγχος κενής ουράς
- Αποφυγή μετατοπίσεων: αποθηκευτική δομή
- Πακέτο για τον ΑΤΔ ουρά

Εισαγωγή

Μια ουρά είναι μια "γραμμή αναμονής", όπως για παράδειγμα:

- μια ουρά ανθρώπων, που περιμένουν στο ταμείο ενός σουπερ-μάρκετ,
- ή μια ουρά εργασιών σ' ένα σύστημα υπολογιστή, που περιμένουν για κάποια συσκευή εξόδου, π.χ. έναν εκτυπωτή.

Σε καθένα από τα παραδείγματα αυτά, τα αντικείμενα εξυπηρετούνται με την σειρά με την οποία φτάνουν, δηλαδή το πρώτο στοιχείο της ουράς είναι το πρώτο που θα εξυπηρετηθεί.

Στοίβα - Ουρά

Ενώ, λοιπόν, μια στοίβα είναι μια "*τελευταίος μέσα - πρώτος έξω*" (Last In-First out, LIFO) δομή, η ουρά είναι μια ***"πρώτος μέσα - πρώτος έξω"*** (First In - First Out, **FIFO**) δομή.

Βασικές λειτουργίες ουράς

Στην **ουρά** (**queue**) οι βασικές λειτουργίες εισαγωγής και διαγραφής πραγματοποιούνται στα δύο άκρα της.

Αντίθετα με τις στοίβες, στις οποίες τα στοιχεία απωθούνται ή ωθούνται μόνο στο ένα άκρο της δομής δεδομένων,

- η διαγραφή ενός στοιχείου από την ουρά γίνεται στο ένα άκρο της, που ονομάζεται **εμπρός** (**front**) ή **κεφάλι** (**head**), και
- η εισαγωγή ενός νέου στοιχείου γίνεται στο άλλο άκρο της, που ονομάζεται **πίσω** (**rear**) ή **ουρά** (**tail**).

Οι βασικές λειτουργίες Ουράς

- Δημιουργία μιας κενής ουράς
- Έλεγχος αν μια ουρά είναι κενή
- Διαγραφή του στοιχείου που βρίσκεται στο εμπρός άκρο της ουράς
- Εισαγωγή νέου στοιχείου στο πίσω άκρο της ουράς.

Ορισμός του ΑΤΔ ουρά -1-

Συλλογή στοιχείων δεδομένων:

Μια συλλογή στοιχείων δεδομένων σε γραμμική διάταξη με την ιδιότητα ότι τα στοιχεία μπορούν να διαγράφονται μόνο από το ένα άκρο της, που ονομάζεται ***εμπρός***, και να εισάγονται μόνο στο άλλο άκρο της, που ονομάζεται ***πίσω***.

Ορισμός του ΑΤΔ ουρά -2-

Βασικές λειτουργίες:

Δημιουργία κενής ουράς (CreateQ)

Λειτουργία: Δημιουργεί μια κενή ουρά.

Επιστρέφει: Μια κενή ουρά.

Έλεγχος αν μια ουρά είναι κενή (EmptyQ)

Δέχεται: Μια ουρά.

Λειτουργία: Ελέγχει αν η ουρά είναι κενή.

Επιστρέφει: TRUE, αν η ουρά είναι κενή, FALSE διαφορετικά.

Ορισμός του ΑΤΔ ουρά -3-

Διαγραφή στοιχείου από το εμπρός άκρο της ουράς (RemoveQ)

Δέχεται: Μια ουρά.

Λειτουργία: Ανακτά και διαγράφει το στοιχείο που βρίσκεται στο εμπρός άκρο της ουράς.

Επιστρέφει: Το στοιχείο στο εμπρός άκρο της ουράς και την τροποποιημένη ουρά.

Εισαγωγή στοιχείου στο πίσω άκρο της ουράς (AddQ):

Δέχεται: Μια ουρά και ένα στοιχείο δεδομένου.

Λειτουργία: Εισάγει το στοιχείο στο πίσω άκρο της ουράς.

Επιστρέφει: Την τροποποιημένη ουρά.

Παράδειγμα -1-

Η ουρά είναι μια κατάλληλη δδ για την αποθήκευση στοιχείων τα οποία θα υποστούν επεξεργασία με τη σειρά με την οποία δημιουργήθηκαν.

Παράδειγμα: ας υποθέσουμε ότι ένα πρόγραμμα είναι φτιαγμένο να προσφέρει προβλήματα άσκησης για στοιχειώδη αριθμητική:

- Τα προβλήματα αυτά αφορούν την πρόσθεση τυχαίων ακεραίων αριθμών.
- Αν ο μαθητής απαντήσει σωστά, τότε δημιουργείται ένα άλλο πρόβλημα.
- Αν, όμως, δεν απαντήσει σωστά, τότε το πρόβλημα αποθηκεύεται προκειμένου να του ζητηθεί ξανά να το απαντήσει στο τέλος του μαθήματος.
- Όπως φαίνεται, τα προβλήματα που δεν απαντήθηκαν σωστά θα πρέπει να εμφανιστούν με την ίδια σειρά με την οποία εμφανίστηκαν αρχικά.
- Επομένως, μια ουρά είναι η κατάλληλη δομή δεδομένων για την αποθήκευση αυτών των προβλημάτων.

Παράδειγμα -2-

Έστω ότι έχει αναπτυχθεί ένα πακέτο ή μονάδα για τον ΑΤΔ Ουρά ώστε να ορίζει έναν τύπο δεδομένων **QueueType**, ο οποίος μπορεί να χρησιμοποιηθεί για τη δήλωση μιας ουράς WrongQueue εγγραφών προβλημάτων της μορφής

```
typedef struct {
        int Addend1, Addend2;
} QueueElementType;
```

καθώς και διαδικασίες **CreateQ**, **AddQ** και **RemoveQ** και μια boolean συνάρτηση **EmptyQ** για τις λειτουργίες της ουράς..

Παράδειγμα -3-

Τότε προτάσεις σαν τις ακόλουθες μπορούν να χρησιμοποιηθούν για τη δημιουργία ενός προβλήματος και την πρόσθεσή του στην WrongQueue, αν δεν απαντηθεί σωστά:

```
CreateQ(WrongQueue);
Problem.Addend1 = RandomInt(0,
  NumberLimit);
Problem.Addend2 = RandomInt(0,
  NumberLimit);
Ask(Problem,Correct,1);
if (!Correct) AddQ(&WrongQ,Problem);
```

Παράδειγμα -4-

- Η RandomInt είναι μια συνάρτηση που δημιουργεί τυχαίους ακεραίους από ένα καθορισμένο εύρος ακεραίων αριθμών.
- Η διαδικασία Ask εμφανίζει ένα πρόβλημα πρόσθεσης στο μαθητή, διαβάζει μια απάντηση και επιστρέφει την τιμή TRUE ή FALSE για την boolean παράμετρο Correct για να δηλώσει αν η απάντηση ήταν σωστή. Η παράμετρος 1 δηλώνει το πόσες φορές έχει ζητηθεί το πρόβλημα.

Παράδειγμα -5-

Στην συνέχεια, μπορούν να χρησιμοποιηθούν οι ακόλουθες προτάσεις για την επανάληψη των προβλημάτων που δεν απαντήθηκαν σωστά:

```
while (!EmptyQ(WrongQ)) {
        RemoveQ(&WrongQ, &Problem);
        Ask(Problem,Correct, 2);
        if (!Correct)
            Wrong ++
}
```

Εισαγωγή -1-

Επειδή οι ουρές μοιάζουν πολύ με τις στοίβες, μπορούμε να υλοποιήσουμε μια ουρά χρησιμοποιώντας πάλι πίνακες.

Για την ουρά, λοιπόν, χρειαζόμαστε έναν πίνακα, **Queue**, για την αποθήκευση των στοιχείων της ουράς, και δύο μεταβλητές:

- την μεταβλητή **Front**, όπου θα αποθηκεύεται η θέση του πίνακα στην οποία βρίσκεται το στοιχείο που μπορεί να διαγραφεί, δηλαδή το πρώτο στοιχείο της ουράς, και
- την **Rear**, όπου θα αποθηκεύεται η θέση του πίνακα στην οποία μπορεί να εισαχθεί ένα νέο στοιχείο, δηλαδή η θέση μετά το τελευταίο στοιχείο της ουράς.

Εισαγωγή -2-

Ένα στοιχείο, λοιπόν, μπορεί να διαγραφεί από την ουρά:

- ανακτώντας το στοιχείο που βρίσκεται στην θέση Front του πίνακα Queue, δηλαδή Queue[Front], και
- αυξάνοντας την μεταβλητή Front κατά 1.

Ένα στοιχείο εισάγεται στην ουρά:

- με αποθήκευσή του στην θέση Rear του πίνακα Queue, δηλαδή Queue[Rear], με την προϋπόθεση, βέβαια ότι η μεταβλητή Rear είναι μικρότερη του ορίου του πίνακα, και
- αυξάνοντας την μεταβλητή Rear κατά 1.

Η δυσκολία εδώ είναι ότι τα στοιχεία μετατοπίζονται προς τα δεξιά μέσα στον πίνακα, πράγμα το οποίο σημαίνει ότι ίσως να χρειαστεί όλα τα στοιχεία του πίνακα να μετατοπιστούν πίσω στις αρχικές θέσεις.

Παράδειγμα -6-

Για να γίνει κατανοητή η λειτουργία της ουράς, ας θεωρήσουμε μια ουρά 5 θέσεων στην οποία:

- εισάγονται με τη σειρά οι αριθμοί 30, 17 και 25,
- εν συνεχεία διαγράφονται οι 30 και 17 και, τέλος,
- εισάγονται οι 7 και 53, όπως δείχνουν τα ακόλουθα σχήματα.

Εισαγωγή του 30 **Queue**

θέση	**1**	**2**	**3**	**4**	**5**
αριθμός	**30**				

Εισαγωγή του 17 **Queue**

θέση	**1**	**2**	**3**	**4**	**5**
αριθμός	30	17			

Εισαγωγή του 25 **Queue**

θέση	**1**	**2**	**3**	**4**	**5**
αριθμός	30	17	25		

Παράδειγμα -7-

Queue

θέση	1	2	3	4	5
αριθμός		17	25		

Διαγραφή του 30

Queue

θέση	1	2	3	4	5
αριθμός			25		

Διαγραφή του 17

Queue

θέση	1	2	3	4	5
αριθμός			25	7	

Εισαγωγή του 7

Queue

θέση	1	2	3	4	5
αριθμός			25	7	53

Εισαγωγή του 53

Παράδειγμα -8-

Στο σημείο αυτό, για να εισάγουμε έναν νέο αριθμό στην ουρά θα πρέπει πρώτα να μετατοπίσουμε τους 3 αριθμούς στις πρώτες θέσεις του πίνακα.

Queue

θέση	**1**	**2**	**3**	**4**	**5**
αριθμός	25	7	53		

Αποφυγή μετατοπίσεων -1-

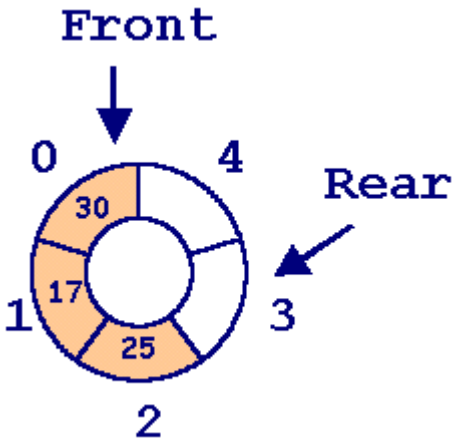
Για να αποφύγουμε τις μετατοπίσεις, μπορούμε να θεωρήσουμε έναν κυκλικό πίνακα, όπου το πρώτο στοιχείο ακολουθεί το τελευταίο.

Αυτό μπορεί να γίνει αν σχήματα: δεικτοδοτήσουμε τον πίνακα, ξεκινώντας από το 0, και αυξάνουμε τις μεταβλητές Front και Rear, ώστε να παίρνουν τιμές από 0 μέχρι το όριο της ουράς.

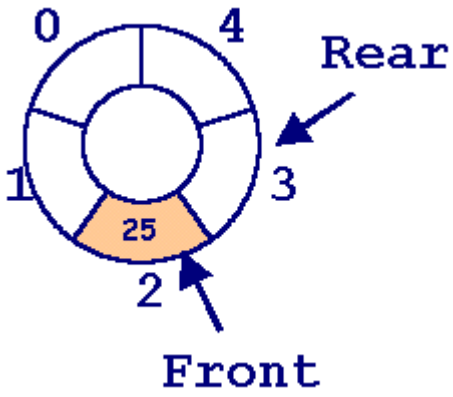
Οι παραπάνω πράξεις εισαγωγής και διαγραφής γίνονται όπως δείχνουν τα

Αποφυγή μετατοπίσεων -2-

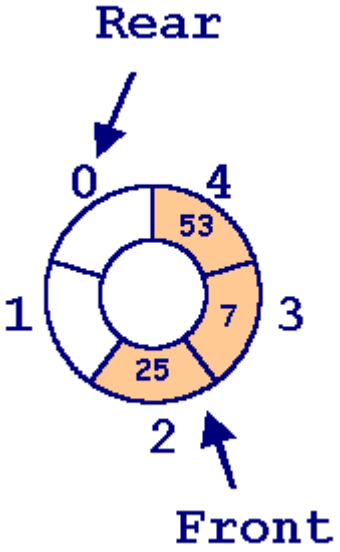
Εισαγωγή των 30, 17 και 25



Διαγραφή των 30 και 17



Εισαγωγή των 7 και 53



Τώρα μπορούμε να εισαγάγουμε νέο στοιχείο στη θέση 0 χωρίς να χρειάζεται να μετακινήσουμε τα υπόλοιπα.

Αποφυγή μετατοπίσεων: έλεγχος κενής ουράς -1-

Εξέταση της βασικής διαδικασίας **EmptyQ** που καθορίζει αν η ουρά είναι κενή:

- Αν η ουρά περιέχει ένα μόνο στοιχείο, τότε αυτό βρίσκεται στη θέση Front του πίνακα και η Rear είναι η κενή θέση που ακολουθεί.
- Αν αυτό το στοιχείο διαγραφεί, η μεταβλητή Front αυξάνεται κατά 1 και η Rear έχει την ίδια τιμή.
- Επομένως, για να καθορίσουμε αν η ουρά είναι κενή, το μόνο που χρειάζεται να κάνουμε είναι να ελέγξουμε τη συνθήκη **Front == Rear**.
- Αρχικά, η CreateQ θέτει τις Front και Rear ίσες με 0.

Όπως στην υλοποίηση της στοίβας με πίνακα υπήρχε η πιθανότητα γεμάτης στοίβας, έτσι και στην υλοποίηση της ουράς προκύπτει η πιθανότητα γεμάτης ουράς..

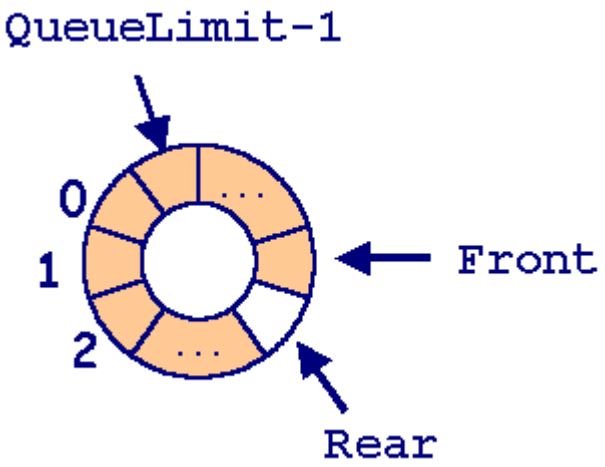
Αποφυγή μετατοπίσεων: έλεγχος κενής ουράς -2-

Εξέταση της βασικής διαδικασίας **EmptyQ** που καθορίζει αν η ουρά είναι κενή:

- Αν η ουρά περιέχει ένα μόνο στοιχείο, τότε αυτό βρίσκεται στη θέση Front του πίνακα και η Rear είναι η κενή θέση που ακολουθεί.
- Αν αυτό το στοιχείο διαγραφεί, η μεταβλητή Front αυξάνεται κατά 1 και η Rear έχει την ίδια τιμή.
- Επομένως, για να καθορίσουμε αν η ουρά είναι κενή, το μόνο που χρειάζεται να κάνουμε είναι να ελέγξουμε τη συνθήκη **Front == Rear**.
- Αρχικά, η CreateQ θέτει τις Front και Rear ίσες με 0.

Όπως στην υλοποίηση της στοίβας με πίνακα υπήρχε η πιθανότητα γεμάτης στοίβας, έτσι και στην υλοποίηση της ουράς προκύπτει η πιθανότητα γεμάτης ουράς..

Αποφυγή μετατοπίσεων: έλεγχος γεμάτης ουράς



Επομένως, η συνθήκη που δείχνει αν η ουρά είναι γεμάτη είναι τώρα η

(Rear+1) **%** QueueLimit == Front,

όπου QueueLimit είναι το μέγιστο μέγεθος της ουράς.

- Αν ένα στοιχείο αποθηκευόταν σ' αυτήν τη θέση, τότε η Rear θα αυξανόταν κατά 1 και θα είχε την ίδια τιμή με την Front.
- Όμως, η συνθήκη Front==Rear δηλώνει ότι η ουρά είναι κενή.
- Έτσι, λοιπόν, δεν θα μπορούσαμε να ξεχωρίσουμε μια κενή ουρά από μια γεμάτη αν αυτή η θέση χρησιμοποιούνταν για την αποθήκευση ενός στοιχείου.
- Κάτι τέτοιο μπορεί να αποφευχθεί αν διατηρούμε μια κενή θέση στον πίνακα..

Αποφυγή μετατοπίσεων : αποθηκευτική δομή -1-

Για να υλοποιήσουμε μια ουρά, μπορούμε να χρησιμοποιήσουμε ως αποθηκευτική δομή μια **εγγραφή** :

- αποτελούμενη από έναν κυκλικό πίνακα, τον Element, για την αποθήκευση των στοιχείων της ουράς, και
- από τα πεδία Front και Rear, όπου αποθηκεύουμε τη θέση της εμπρός άκρης της ουράς και τη θέση που ακολουθεί αμέσως μετά την πίσω άκρη της ουράς.

Αποφυγή μετατοπίσεων : αποθηκευτική δομή -2-

```
#define QueueLimit 20
typedef int QueueElementType;
        /* ο τύπος των στοιχείων της ουράς ενδεικτικά τύπου int */
typedef struct {
        int Front, Rear;
        QueueElementType Element[QueueLimit];
} QueueType;
```

Η λειτουργία δημιουργίας μιας κενής ουράς συνίσταται απλά στο να τεθούν οι μεταβλητές Front και Rear της ουράς ίσες με 0, και **μια ουρά είναι κενή όταν η boolean έκφραση Queue.Front == Queue.Rear είναι αληθής**.

Πακέτο για τον ΑΤΔ ουρά -1-

```
/*Filename: QueueADT.h */
#define QueueLimit 20                    /*μέγιστο μέγεθος της ουράς*/

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

typedef enum {FALSE, TRUE} boolean;
```

Πακέτο για τον ΑΤΔ ουρά -2-

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

Πακέτο για τον ΑΤΔ ουρά -3-

```
/*Filename: QueueADT.c */
#include <stdio.h>
#include “QueueADT.h”
void CreateQ(QueueType *Queue)
/*  Λειτουργία:  Δημιουργεί μια κενή ουρά.
    Επιστρέφει:  Κενή ουρά.*/
{
     Queue->Front = 0;
     Queue->Rear = 0;
 }
```

Πακέτο για τον ΑΤΔ ουρά -4-

```
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);
}
```

Πακέτο για τον ΑΤΔ ουρά -5-

```
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));
}
```

Πακέτο για τον ΑΤΔ ουρά -6-

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.*/

Πακέτο για τον ΑΤΔ ουρά -7-

```
{
if (! EmptyQ(*Queue))
{
        *Item = Queue ->Element[Queue -> Front];
                Queue ->Front = (Queue ->Front + 1)
                % QueueLimit;
        }
        else
                printf("Empty Queue\n");
}
```

Πακέτο για τον ΑΤΔ ουρά -8-

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.*/

{

Πακέτο για τον ΑΤΔ ουρά -9-

```
{
        int NewRear;
        if  (!FullQ(*Queue)) {
                NewRear = (Queue ->Rear + 1) %
                QueueLimit;
                Queue ->Element[Queue ->Rear] =
                Item;
                Queue ->Rear = NewRear;
        }
        else  printf("Full Queue");
}
```

Τέλος Ενότητας

Ευρωπαϊκή Ένωση
Ευρωπαϊκό Κοινωνικό Ταμείο
ΕΠΙΧΕΙΡΗΣΙΑΚΟ ΠΡΟΓΡΑΜΜΑ
ΕΚΠΑΙΔΕΥΣΗ ΚΑΙ ΔΙΑ ΒΙΟΥ ΜΑΘΗΣΗ
επένδυση στην κοινωνία της γνώσης
ΥΠΟΥΡΓΕΙΟ ΠΑΙΔΕΙΑΣ ΚΑΙ ΘΡΗΣΚΕΥΜΑΤΩΝ
ΕΙΔΙΚΗ ΥΠΗΡΕΣΙΑ ΔΙΑΧΕΙΡΙΣΗΣ
ΕΣΠΑ 2007-2013
πρόγραμμα για την ανάπτυξη
ΕΥΡΩΠΑΪΚΟ ΚΟΙΝΩΝΙΚΟ ΤΑΜΕΙΟ
Με τη συγχρηματοδότηση της Ελλάδας και της Ευρωπαϊκής Ένωσης