ΕΛΛΗΝΙΚΗ ΔΗΜΟΚΡΑΤΙΑ
ΠΑΝΕΠΙΣΤΗΜΙΟ ΜΑΚΕΔΟΝΙΑΣ

ΟΥΡΕΣ

3.2 Υλοποίηση ΑΤΔ Ουρά με πίνακα

Εισαγωγή

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

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:

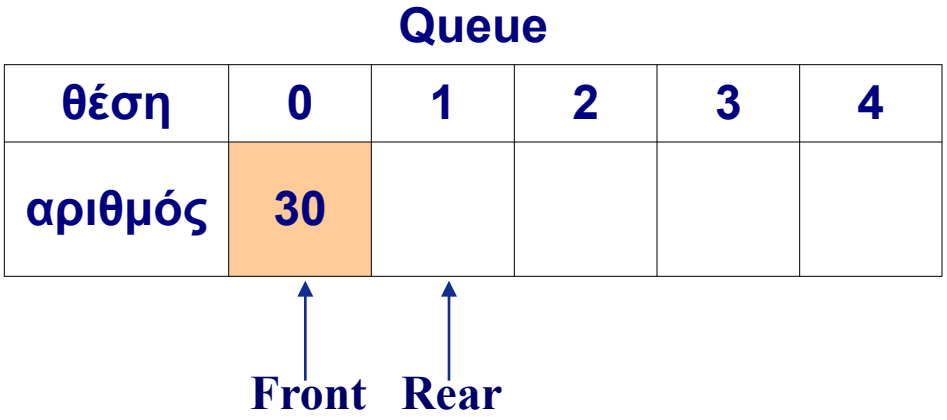
- με αποθήκευσή του στην θέση Rear του πίνακα Queue, δηλαδή Queue[Rear], με την προϋπόθεση, βέβαια ότι η μεταβλητή Rear είναι μικρότερη του ορίου του πίνακα, και
- αυξάνοντας την μεταβλητή Rear κατά 1.

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

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

Παράδειγμα

Queue

θέση	0	1	2	3	4
αριθμός	30	17			

Front Rear

Εισαγωγή του 17

Queue

θέση	0	1	2	3	4
αριθμός	30	17	25		

Front Rear

Εισαγωγή του 25

Παράδειγμα

Queue

θέση	0	1	2	3	4
αριθμός		17	25		

Front Rear

Διαγραφή του 30

Queue

θέση	0	1	2	3	4
αριθμός			25		

Front Rear

Διαγραφή του 17

Παράδειγμα

Queue

θέση	0	1	2	3	4
αριθμός			25	7	

Front Rear

Εισαγωγή του 7

Queue

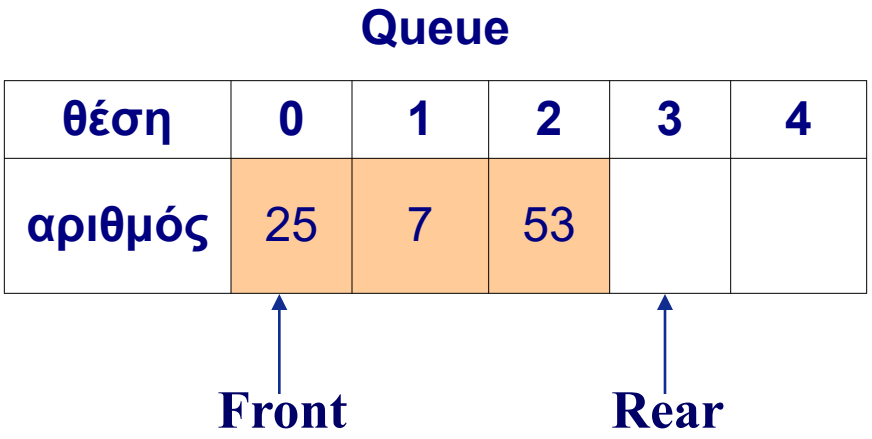
θέση	0	1	2	3	4
αριθμός			25	7	53

Front Rear

Εισαγωγή του 53

Παράδειγμα

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



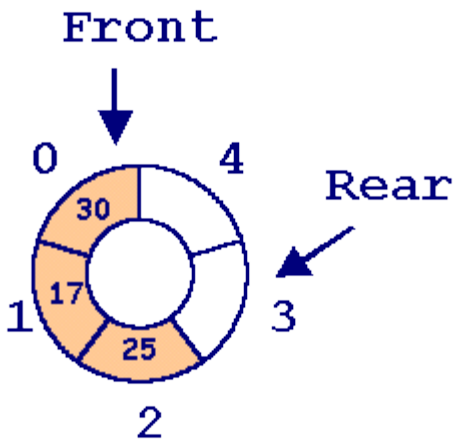
Αποφυγή μετατοπίσεων

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

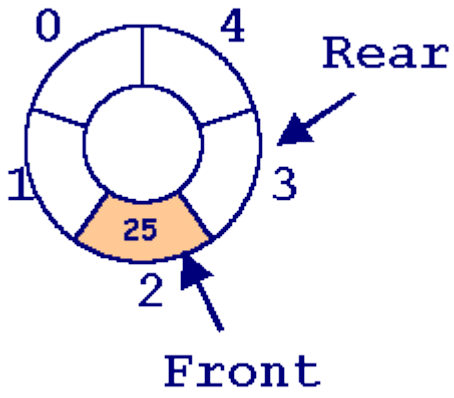
Αυτό μπορεί να γίνει αν δεικτοδοτήσουμε τον πίνακα, ξεκινώντας από το 0, και αυξάνουμε τις μεταβλητές Front και Rear, ώστε να παίρνουν τιμές από 0 μέχρι το όριο της ουράς.

Οι παραπάνω πράξεις εισαγωγής και διαγραφής γίνονται όπως δείχνουν τα σχήματα:

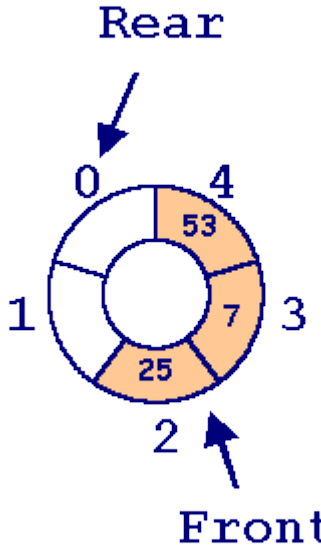
Εισαγωγή των 30, 17 και 25



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

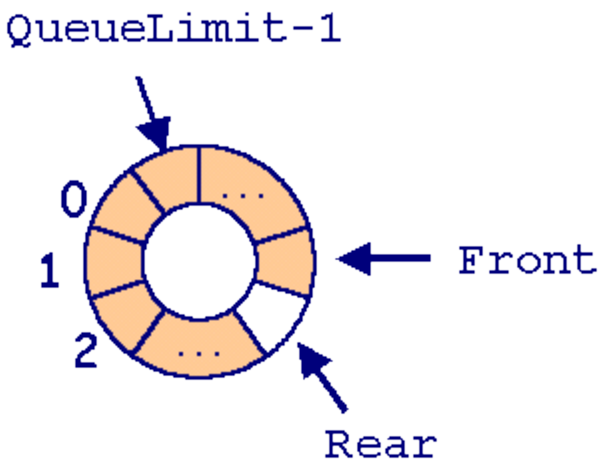
Εξέταση της βασικής διαδικασίας **EmptyQ** που καθορίζει αν η ουρά είναι κενή:

- 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

Για να δούμε πώς μπορεί να εξεταστεί η συνθήκη γεμάτης ουράς, υποθέτουμε ότι ο πίνακας είναι σχεδόν γεμάτος, με μόνο μία κενή θέση:



- 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.

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.

Αποφυγή μετατοπίσεων: αποθηκευτική δομή

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

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

QueueType Queue; /*Δήλωση μεταβλητής Queue για ουρά*/

Queue

Front	Rear	πίνακας Element[ ] κάθε στοιχείο του τύπου QueueElementType								
		0	1	2	3	4	5	6		QueueLimit-1
2	5	?	?	23	12	8	?	?	…	?

Δημιουργία και Έλεγχος Κενής Ουράς

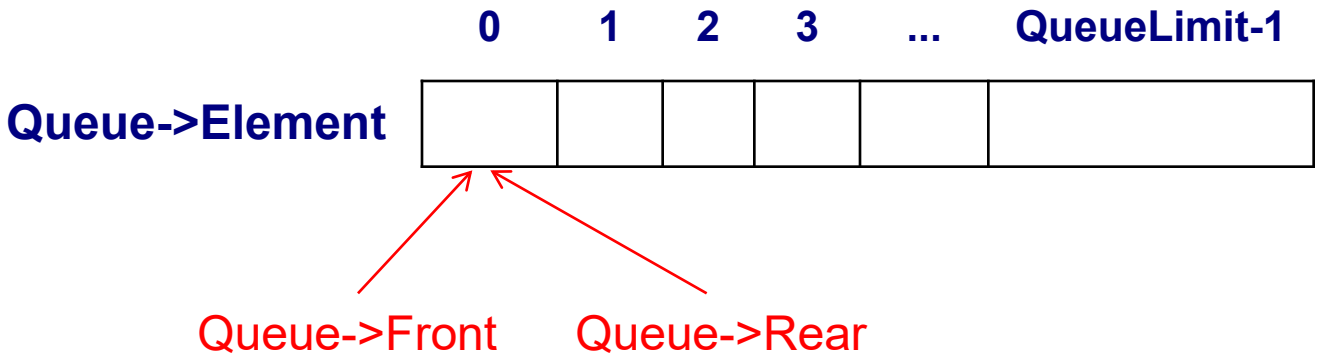
Η λειτουργία δημιουργίας μιας κενής ουράς συνίσταται απλά στο να τεθούν οι μεταβλητές Front και Rear της ουράς ίσες με 0,

και μια ουρά είναι κενή όταν η boolean έκφραση Queue.Front == Queue.Rear είναι αληθής.

Πακέτο για τον ΑΤΔ ουρά

```
/*Filename: QueueADT.h */
#define QueueLimit 20                /*ενδεικτικό 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;

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

Δημιουργία ουράς

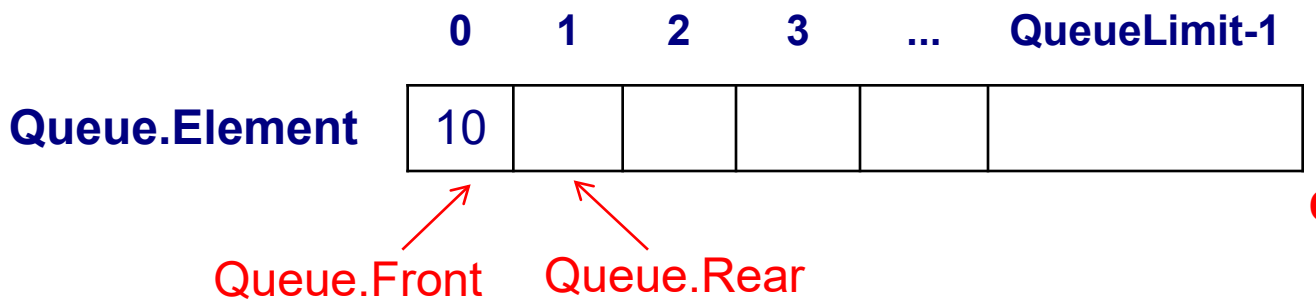



```
void CreateQ(QueueType *Queue){
    Queue->Front = 0;
    Queue->Rear = 0;
 }
```

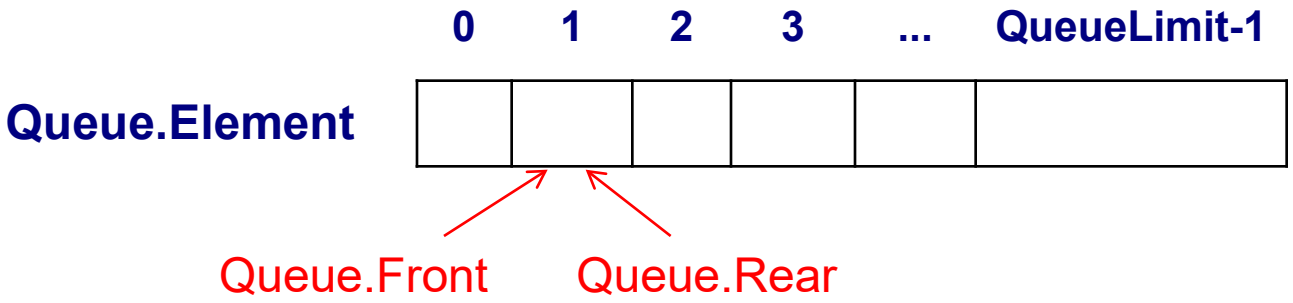
Πακέτο για τον ΑΤΔ ουρά

```
/*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;
 }
```

Έλεγχος κενής ουράς



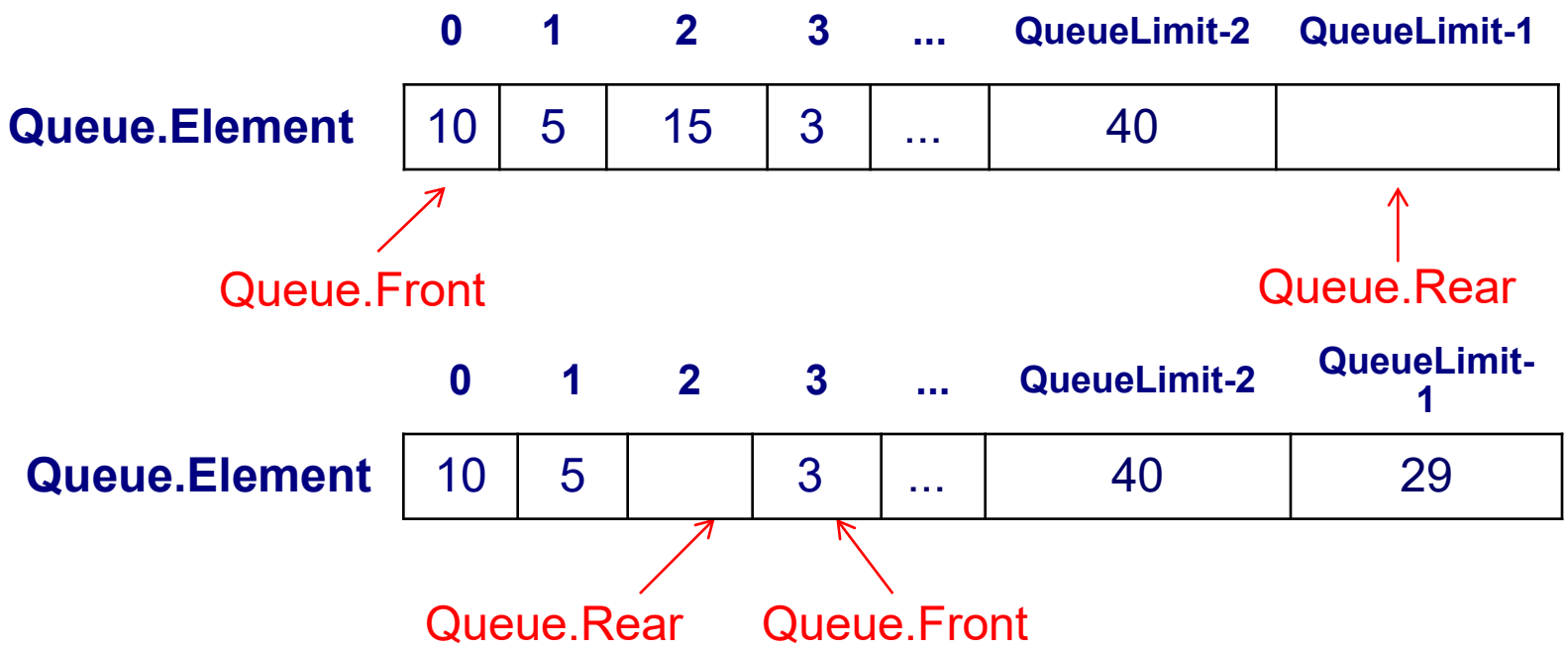
Queue.Front !=Queue.Rear
Not Empty



Queue.Front ==Queue.Rear
Empty

```
boolean EmptyQ(QueueType Queue){
        return (Queue.Front == Queue.Rear);
}
```

Έλεγχος γεμάτης ουράς

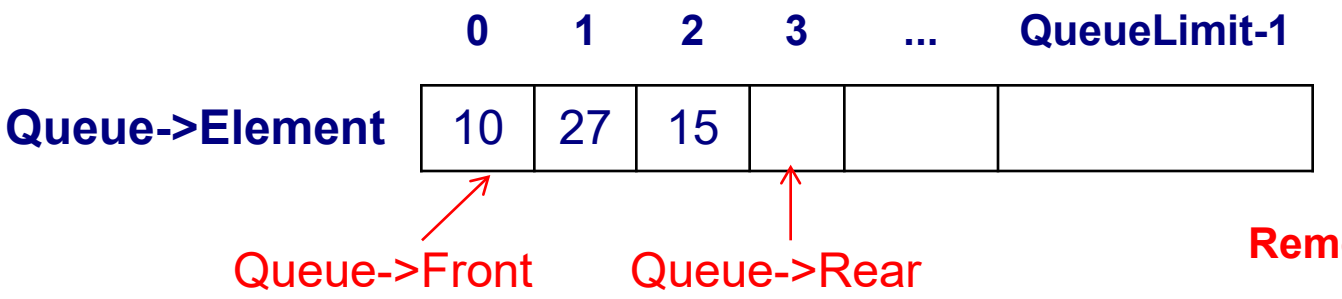



```
boolean FullQ(QueueType Queue)

        return ((Queue.Front) == ((Queue.Rear +1) % QueueLimit));
}
```

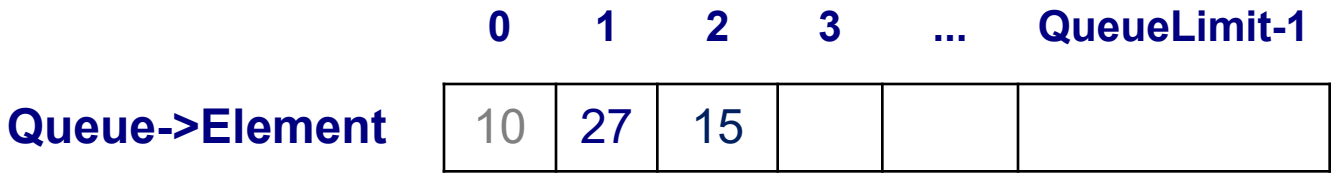
Πακέτο για τον ΑΤΔ ουρά

```
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));
}
```

Διαγραφή στοιχείου από το εμπρός άκρο της ουράς



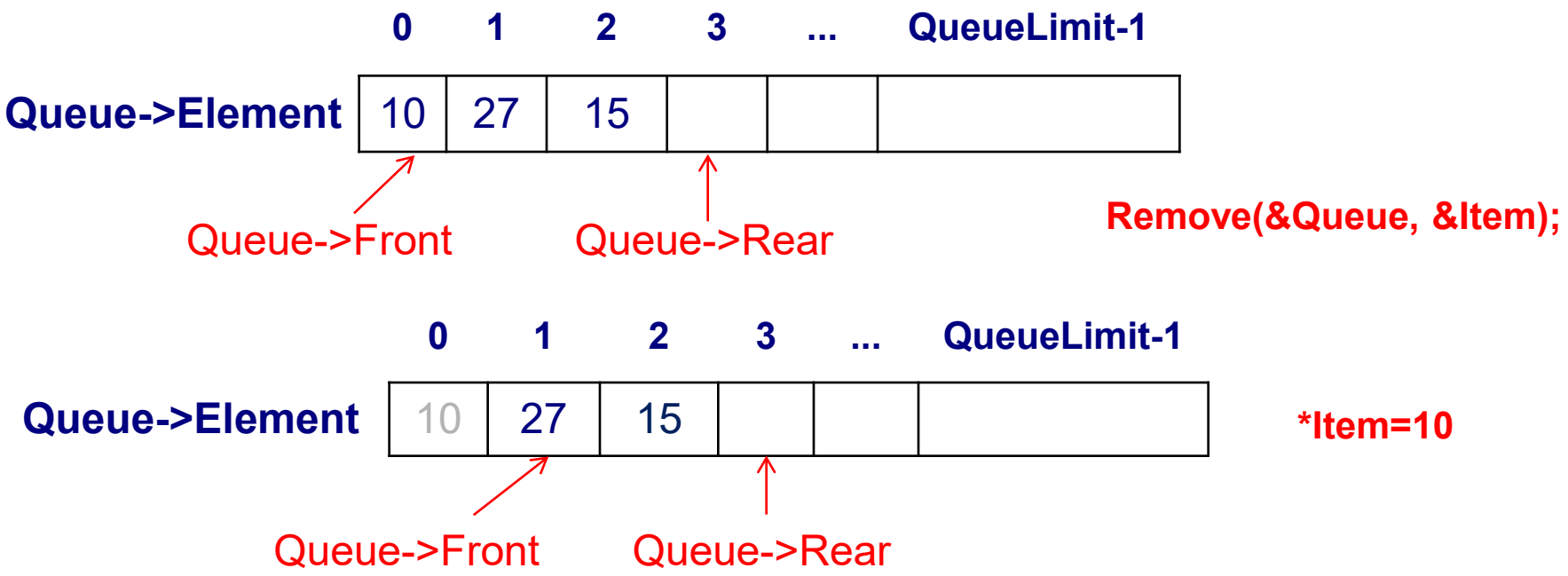
Remove(&Queue, &Item);



***Item=10**

Queue->Front ??? Queue->Rear ???

Διαγραφή στοιχείου από το εμπρός άκρο της ουράς

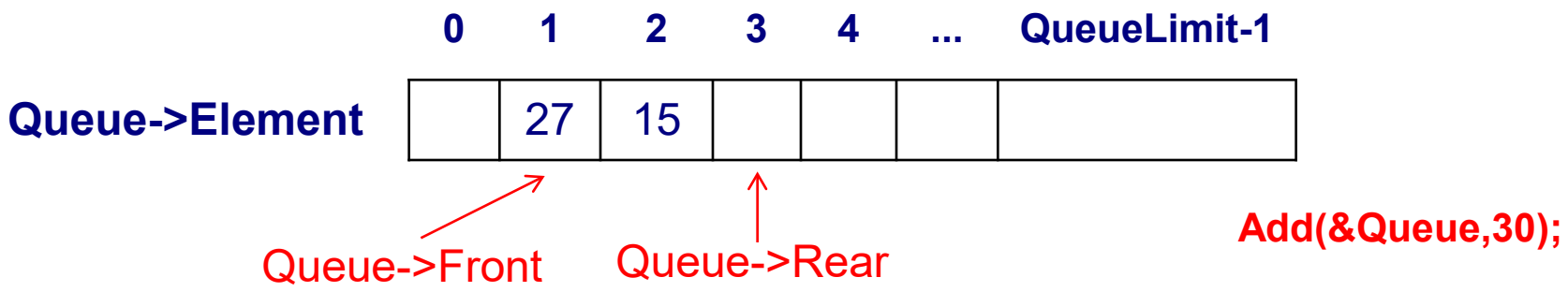


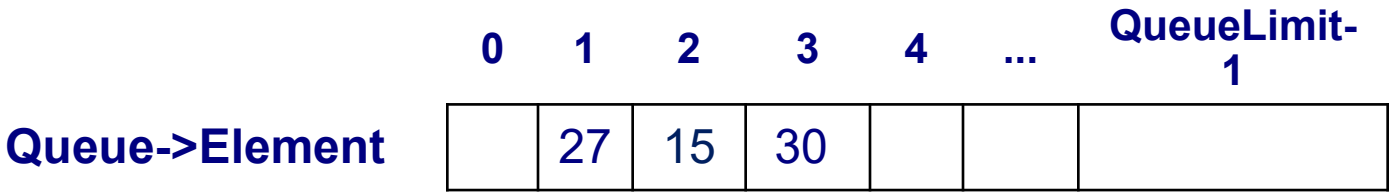

```
void RemoveQ(QueueType *Queue, QueueElementType *Item) {
if (! EmptyQ(*Queue))
{
        *Item = Queue ->Element[Queue -> Front];
        Queue ->Front = (Queue ->Front + 1) % QueueLimit;
}
else
        printf("Empty Queue\n");
}
```

Πακέτο για τον ΑΤΔ ουρά

```
void RemoveQ(QueueType *Queue, QueueElementType *Item)
/*Δέχεται:     Μια ουρά.
Λειτουργία:   Ανακτά και διαγράφει το στοιχείο που βρίσκεται στο εμπρός άκρο της
              ουράς Queue αν η ουρά δεν είναι άδεια.
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\n");
}
```

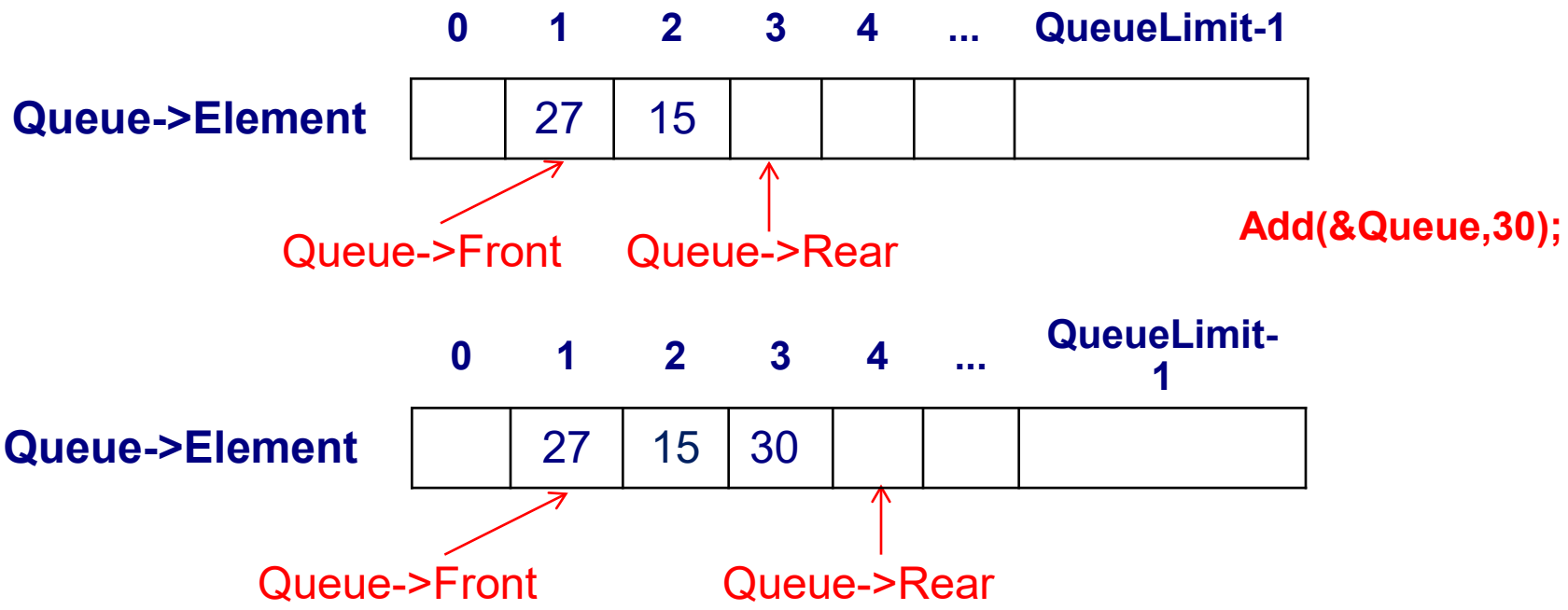
Εισαγωγή στοιχείου στο πίσω άκρο της ουράς





Queue->Front ??? Queue->Rear ???

Εισαγωγή στοιχείου στο πίσω άκρο της ουράς




```
void AddQ(QueueType *Queue, QueueElementType Item){
if  (!FullQ(*Queue))
{
        Queue ->Element[Queue ->Rear] = Item;
        Queue ->Rear = (Queue ->Rear + 1) % QueueLimit;
}
else
        printf("Full Queue");
}
```

Πακέτο για τον ΑΤΔ ουρά

```
void AddQ(QueueType *Queue, QueueElementType Item)
/*Δέχεται:      Μια ουρά Queue και ένα στοιχείο Item.
Λειτουργία:    Εισάγει το στοιχείο στο πίσω άκρο της ουράς 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");
}
```