Δομές Δεδομένων σε C

Μάθημα 3:

Ουρά

Δημήτρης Ψούνης

Περιεχόμενα Μαθήματος

Α. Θεωρία
1. Ουρά
   1. Ορισμός Ουράς
   2. Βασικές Πράξεις
2. Απλή Ουρά με Πίνακα
   1. Γενικά
   2. Υλοποίηση σε C: Δηλώσεις
   3. Υλοποίηση σε C: Αρχικοποίηση
   4. Υλοποίηση σε C: Έλεγχοι – Κενή Ουρά και Γεμάτη Ουρά
   5. Υλοποίηση σε C: Προσθήκη Στοιχείου
   6. Υλοποίηση σε C: Απομάκρυνση Στοιχείου
   7. Υλοποίηση σε C: Παράδειγμα
3. Κυκλική Ουρά
   1. Γενικά
   2. Υλοποίηση σε C: Δηλώσεις
   3. Υλοποίηση σε C: Αρχικοποίηση
   4. Υλοποίηση σε C: Έλεγχοι – Κενή Ουρά
   5. Υλοποίηση σε C: Έλεγχοι – Γεμάτη Ουρά
   6. Υλοποίηση σε C: Προσθήκη Στοιχείου
   7. Υλοποίηση σε C: Απομάκρυνση Στοιχείου
   8. Υλοποίηση σε C: Παράδειγμα

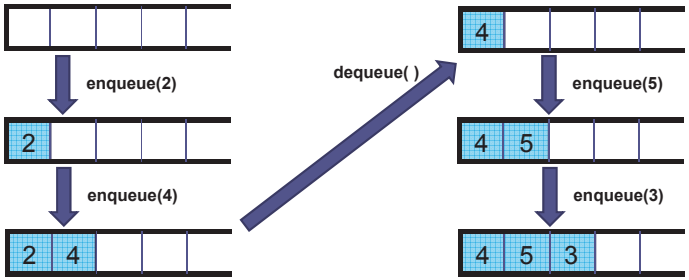
Β. Ασκήσεις

Α. Θεωρία

1. Ουρά

1. Ορισμός Ουράς

Η **«Ουρά»** είναι μια δομή δεδομένων με γραμμική διάταξη στην οποία:
- Η προσθήκη (enqueue) ενός στοιχείου, γίνεται στο τέλος της ουράς
- Η απομάκρυνση (dequeue) ενός στοιχείου, γίνεται στην αρχή της ουράς



Παράδειγμα:
- Η ουρά των πελατών στο ταμείο μιας τράπεζας!

Σημαντική Ιδιότητα:
- Το πρώτο στοιχείο που προστέθηκε στην ουρά είναι το πρώτο που θα εξαχθεί (First In – First Out: **FIFO**)

Α. Θεωρία

1. Ουρά

2. Βασικές Πράξεις

Οι **βασικές πράξεις** σε μία ουρά είναι:
- **Αρχικοποίηση** της ουράς (**init**)
- **Προσθήκη** ενός στοιχείου στην ουρά (**enqueue**)
- **Απομάκρυνση** ενός στοιχείου από τη ουρά (**dequeue**)
- **Έλεγχος** αν η ουρά είναι κενή (**empty**)
- **Έλεγχος** αν η ουρά είναι γεμάτη (**full**)

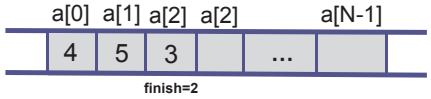
Υπάρχουν τρεις υλοποιήσεις:
- Με στατικό πίνακα (σημερινό μάθημα)
  - Απλή ουρά (με πίνακα)
  - Κυκλική ουρά
- Με απλά συνδεδεμένη λίστα (επόμενο μάθημα)

Α. Θεωρία
2. Απλή Ουρά με Πίνακα
1. Εισαγωγή

Στην πρώτη υλοποίηση (**απλή ουρά**) θα χρησιμοποιήσουμε:
- Έναν στατικό πίνακα N θέσεων για να αποθηκεύσουμε τα στοιχεία της ουράς
- Μία ακέραια μεταβλητή (finish) που θα δείχνει το τέλος της ουράς
- (Εννοείται ότι η αρχή της ουράς θα είναι το 0)

Ουρά

Γλώσσα C

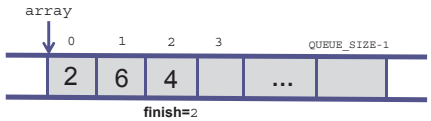


Α. Θεωρία
2. Απλή Ουρά με Πίνακα
2. Υλοποίηση σε C: Δηλώσεις

Οι **δηλώσεις** σε C είναι οι ακόλουθες:
- Η ουρά είναι μία δομή (struct) με τα εξής στοιχεία:
  - Ένας **πίνακας** με **QUEUE_SIZE** στοιχεία
  - Μία ακέραια μεταβλητή (**finish**) που δείχνει το τέλος της ουράς με τιμή:
    - Από 0…QUEUE_SIZE-1 αν η ουρά έχει τουλάχιστον ένα στοιχείο
    - -1 αν η ουρά είναι άδεια.

```
#define QUEUE_SIZE 10       /* Megethos pinaka ouras */
typedef int elem;           /* typos dedomenwn ouras */
struct queue{
   elem array[QUEUE_SIZE]; /* pinakas stoixeiwn     */
   int finish;              /* telos tis ouras       */
};
typedef struct queue QUEUE; /* Sinwnimo tis ouras    */
```

Αναπαράσταση:

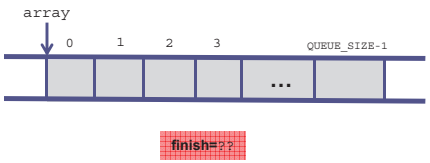


Α. Θεωρία
2. Απλή Ουρά με Πίνακα
3. Υλοποίηση σε C: Αρχικοποίηση Ουράς

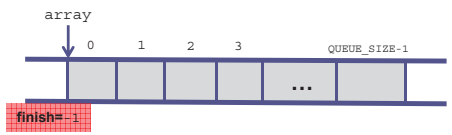
Η αρχικοποίηση γίνεται θέτοντας το τέλος της ουράς ίσο με -1

```
/* QU_init(): arxikopoiei tin oura */
void QU_init(QUEUE *q)
{
   q->finish=-1;
}
```

ΠΡΙΝ:



ΜΕΤΑ:



Προσοχή:
- Πάντα προτού ξεκινάμε την χρήση της ουράς θα πρέπει να καλούμε μία φορά αυτήν τη συνάρτηση!

Α. Θεωρία
2. Απλή Ουρά με Πίνακα
4. Υλοποίηση σε C: Έλεγχοι – Κενή Ουρά και Γεμάτη Ουρά

Ο **έλεγχος** αν η ουρά είναι **κενή** (αντίστοιχα **γεμάτη**), γίνεται βλέποντας αν η μεταβλητή `finish` είναι ίση με `-1` (αντίστοιχα `N-1`)

```
/* QU_empty(): epistrefei TRUE/FALSE
 *          analoga me to an i oura einai adeia */
int QU_empty(QUEUE q)
{
   return q.finish==-1;
}
```

```
/* QU_full(): epistrefei TRUE/FALSE
 *          analoga me to an i oura einai gemati */
int QU_full(QUEUE q)
{
   return q.finish==QUEUE_SIZE-1;
}
```

Α. Θεωρία

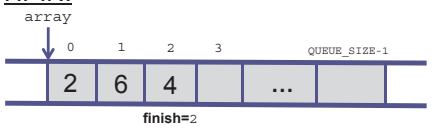
2. Απλή Ουρά με Πίνακα

5. Υλοποίηση σε C: Προσθήκη Στοιχείου

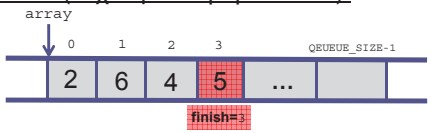
Η **προσθήκη** στην ουρά γίνεται προωθώντας το στοιχείο στη θέση finish+1 (εφόσον χωράει στην ουρά)

```
/* QU_enqueue(): Eisagei to x stin oura q
 *         epistrefei TRUE: se periptwsi epitixias
 *                    FALSE: se periptwsi apotixias */
int QU_enqueue(QUEUE *q,elem x)
{
   if (QU_full(*q))
      return FALSE;
   else
   {
      q->finish++;
      q->array[q->finish]=x;
      return TRUE;
   }
}
```

ΠΡΙΝ:



ΜΕΤΑ (π.χ. προσθήκη του «5»):



Α. Θεωρία

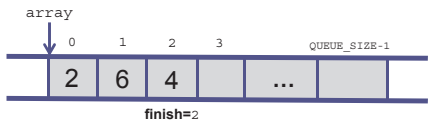
2. Απλή Ουρά με Πίνακα

6. Υλοποίηση σε C: Απομάκρυνση Στοιχείου

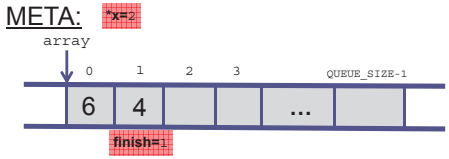
Η **απομάκρυνση** (του πρώτου) στοιχείου της ουράς γίνεται ως εξής:

1. Αποθηκεύεται το στοιχείο που είναι στην αρχή της ουράς (για να επιστραφεί)
2. Μετακινούνται τα υπόλοιπα στοιχεία κατά μία θέση αριστερά
3. Διορθώνεται το finish (μειώνεται κατά 1)

ΠΡΙΝ:



ΜΕΤΑ:



Α. Θεωρία

2. Απλή Ουρά με Πίνακα

6. Υλοποίηση σε C: Απομάκρυνση Στοιχείου

```
/* QU_dequeue(): Kanei apomakrinsi tou prwtou stoixeiou tis ouras
 *         epistrefei TRUE: se periptwsi epitixias
 *                          FALSE: se periptwsi apotixias */
int QU_dequeue(QUEUE *q,elem *x)
{
   int i;

   if (QU_empty(*q))
      return FALSE;
   else
   {
       /* 1. Apothikeysi tou stoixeiou pou eksagetai*/
       *x=q->array[0];

      /* 2. Aristeri metakinisi twn stoixeiwn kata mia thesi */
      for (i=0; i<q->finish; i++)
         q->array[i]=q->array[i+1];

      /* 3. To finish meiwnetai kata 1 */
      q->finish--;

      return TRUE;
   }
}
```


Α. Θεωρία

2. Απλή Ουρά με Πίνακα

7. Υλοποίηση σε C: Παράδειγμα

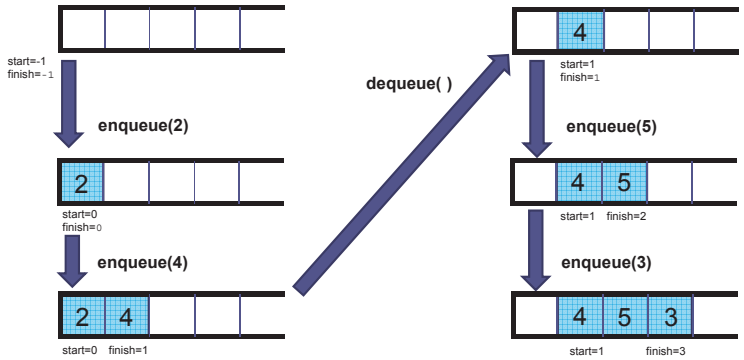
- Μεταγλωττίστε και εκτελέστε το project queue.dev στο οποίο:
  - Το queue.h έχει τα πρωτότυπα των συναρτήσεων και την δήλωση της δομής
  - Το queue.c έχει τα σώματα των συναρτήσεων
  - Το main.c έχει ένα πρόγραμμα που επιδεικνύει την χρήση μίας ουράς ακεραίων.
- «Παίξτε» με το πρόγραμμα ώστε να γίνει πλήρως κατανοητή η λειτουργία της ουράς.

Υπενθύμιση:

- Το σπάσιμο ενός προγράμματος σε επιμέρους αρχεία μελετήθηκε στο «Γλώσσα C – Μάθημα 14: Εμβέλεια Μεταβλητών»

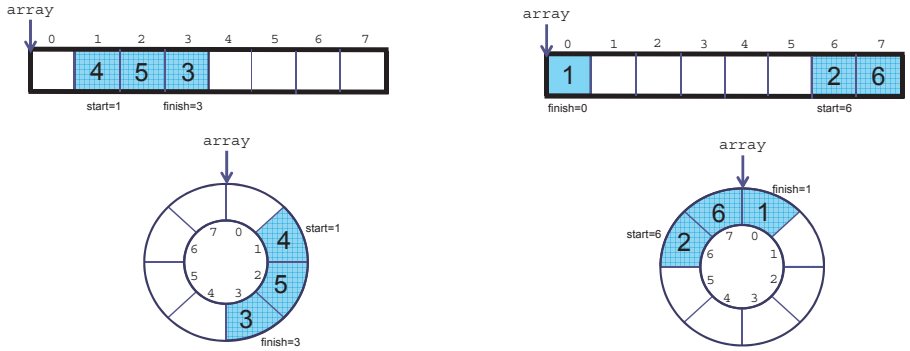
Α. Θεωρία
3. Κυκλική Ουρά
1. Η ανάγκη για την Κυκλική Ουρά

- Η πράξη της απομάκρυνσης απαιτεί την μετακίνηση των στοιχείων της ουράς κατά μία θέση αριστερά, το οποίο κρίνεται μη αποδοτικό.
- Μπορούμε να επιτύχουμε βελτίωση στο χρόνο αν βάλουμε άλλη μία μεταβλητή που να δείχνει την αρχή της ουράς και να έχουμε ως όρια της ουράς αυτές τις δύο μεταβλητές (χωρίς να κάνουμε μετακίνηση των στοιχείων της ουράς). Για παράδειγμα:



Α. Θεωρία
3. Κυκλική Ουρά
1. Η ανάγκη για την Κυκλική Ουρά

- Για να ξεπεραστεί το (νέο) πρόβλημα, ότι κάποια στιγμή θα φτάσουμε στο τέλος του πίνακα (οπότε και δεν θα μπορούμε να εισάγουμε ένα νέο στοιχείο στον πίνακα)
  - Εισάγεται η έννοια της κυκλικής ουράς!
  - Όπου πλέον όταν φτάσουμε στο τέλος του πίνακα οι νέες εισαγωγές θα γίνονται στην αρχή του πίνακα
  - Δηλαδή βλέπουμε τον πίνακα ως κύκλο (όπου η επόμενη θέση της N-1 είναι η 0)

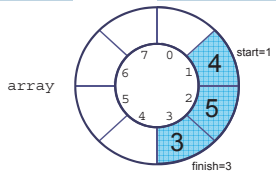


Α. Θεωρία
3. Κυκλική Ουρά
2. Υλοποίηση σε C: Δηλώσεις

Οι **δηλώσεις** σε C είναι οι ακόλουθες:
- Η ουρά είναι μία δομή (struct) με τα εξής στοιχεία:
  - Ένας **πίνακας** με **QUEUE_SIZE** στοιχεία
  - Δύο ακέραιες μεταβλητές (**start** και **finish**) που δείχνουν αντίστοιχα την αρχή και το τέλος της ουράς με τιμές (με start=finish=-1 αν η ουρά είναι άδεια).

```
#define QUEUE_SIZE 10       /* Megethos pinaka ouras */
typedef int elem;           /* typos dedomenwn ouras */
struct queue{
   elem array[QUEUE_SIZE]; /* pinakas stoixeiwn     */
   int start;              /* arxi tis ouras        */
   int finish;             /* telos tis ouras       */
};
typedef struct queue QUEUE; /* Sinwnimo tis ouras    */
```

Αναπαράσταση:

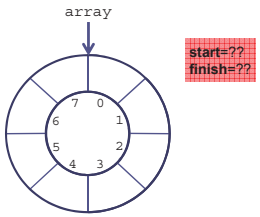


Α. Θεωρία
3. Κυκλική Ουρά
3. Υλοποίηση σε C: Αρχικοποίηση Ουράς

Η αρχικοποίηση γίνεται θέτοντας την αρχή και το τέλος της ουράς ίσο με -1

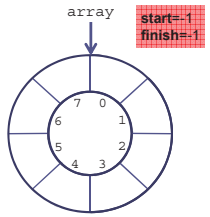
```
/* QU_init(): arxikopoiei tin oura */
void QU_init(QUEUE *q)
{
        q->start =-1;
        q->finish=-1;
}
```

ΠΡΙΝ:



QUEUE_SIZE=8

ΜΕΤΑ:



QUEUE_SIZE=8

Α. Θεωρία
3. Κυκλική Ουρά
4. Υλοποίηση σε C: Έλεγχοι – Κενή Ουρά

Ο **έλεγχος** αν η ουρά είναι **κενή** γίνεται βλέποντας αν η μεταβλητή `start` (και η μεταβλητή `finish`) είναι ίση με `-1`

- Αρκεί να ελέγξουμε τη μία από τις δύο (αφού μεριμνάμε ότι η υλοποίηση των βασικών πράξεων είναι ορθή και αυτός που χρησιμοποιεί την ουρά δεν έχει πρόσβαση στην υλοποίηση)

```
/* QU_empty(): epistrefei TRUE/FALSE
 *          analoga me to an i oura einai adeia */
int QU_empty(QUEUE q)
{
   return q.finish==-1;
}
```


Α. Θεωρία
3. Κυκλική Ουρά
5. Υλοποίηση σε C: Έλεγχοι – Γεμάτη Ουρά

Ο **έλεγχος** αν η ουρά είναι **γεμάτη** γίνεται βλέποντας αν η μεταβλητή finish έχει τιμή 1 λιγότερο από τη μεταβλητή start (δηλαδή `start==finish+1`)

- Μόνο στην περίπτωση που `finish=QUEUE_SIZE-1` τότε πρέπει να ελέγξουμε αν `start==0`

Αυτό μπορεί να γίνει με τον παρακάτω κομψό κώδικα:

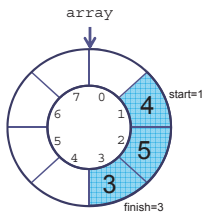
```
/* QU_full(): epistrefei TRUE/FALSE
 *          analoga me to an i oura einai gemati */
int QU_full(QUEUE q)
{
   return q.start==(q.finish+1)%QUEUE_SIZE;
}
```

Υπενθύμιση:
- Ο τελεστής % επιστρέφει το υπόλοιπο της διαίρεσης δύο αριθμών (Γλώσσα C – Μάθημα 4).
- Εδώ τον χρησιμοποιούμε ώστε με μία εντολή να προκύπτει το start είτε ίσο με την επόμενη θέση, είτε ίσο με 0.

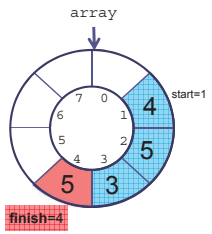
Α. Θεωρία
3. Κυκλική Ουρά
6. Υλοποίηση σε C: Προσθήκη Στοιχείου

Η **προσθήκη** ενός στοιχείου στην ουρά γίνεται (εφόσον αυτή δεν είναι γεμάτη) ως εξής:
- Το `finish` παίρνει τιμή την επόμενη θέση του πίνακα:
  - Την τιμή `finish+1` (αν το `finish < QUEUE_SIZE-1`)
  - Την τιμή `0` (αν το `finish = QUEUE_SIZE-1`)
- Το στοιχείο αποθηκεύεται στην θέση `finish`.

ΠΡΙΝ:



ΜΕΤΑ (π.χ. προσθήκη του «5»):



Α. Θεωρία
3. Κυκλική Ουρά
6. Υλοποίηση σε C: Προσθήκη Στοιχείου

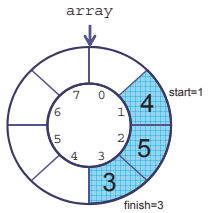
```
/* QU_enqueue(): Eisagei to x stin oura q
 *       epistrefei TRUE: se periptwsi epitixias
 *                  FALSE: se periptwsi apotixias */
int QU_enqueue(QUEUE *q,elem x)
{
   if (QU_full(*q))
      return FALSE;
   else
   {
      if (QU_empty(*q))
      {
         q->start=0;
         q->finish=0;
      }
      else
      {
         q->finish=(q->finish+1)%QUEUE_SIZE;
      }
      q->array[q->finish]=x;
      return TRUE;
   }
}
```

A. Θεωρία
3. Κυκλική Ουρά
6. Υλοποίηση σε C: Απομάκρυνση Στοιχείου

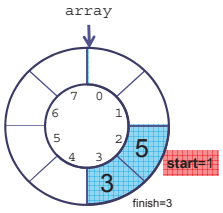
Η **απομάκρυνση** (του πρώτου) στοιχείου από την ουρά γίνεται (εφόσον αυτή δεν είναι άδεια):
- Αποθηκεύοντας το στοιχείο της θέσης start (ώστε να επιστραφεί)

και:
- Αν το start είναι < από QUEUE_SIZE-1, τότε θέτουμε start=start+1
- Αν το start είναι = με QUEUE_SIZE-1 τότε θέτουμε start=0

ΠΡΙΝ:



ΜΕΤΑ *x=4



A. Θεωρία
3. Κυκλική Ουρά
6. Υλοποίηση σε C: Απομάκρυνση Στοιχείου

```
/* QU_dequeue(): Kanei apomakrinsi tou prwtou stoixeiou tis ouras
 *        epistrefei TRUE: se periptwsi epitixias
 *                         FALSE: se periptwsi apotixias */
int QU_dequeue(QUEUE *q,elem *x)
{

   if (QU_empty(*q))
      return FALSE;
   else
   {
      *x=q->array[q->start];
      if (q->start == q->finish) /* H oura adeiase */
      {
         q->start=-1;
         q->finish=-1;
      }
      else
      {
         q->start=(q->start+1)%QUEUE_SIZE;
      }
      return TRUE;
   }
}
```


A. Θεωρία
3. Κυκλική Ουρά
7. Υλοποίηση σε C: Παράδειγμα

- Μεταγλωττίστε και εκτελέστε το project circular_queue.dev στο οποίο:
  - Το queue.h έχει τα πρωτότυπα των συναρτήσεων και την δήλωση της δομής
  - Το queue.c έχει τα σώματα των συναρτήσεων
  - Το main.c έχει ένα πρόγραμμα που επιδεικνύει την χρήση μίας ουράς ακεραίων.

- «Παίξτε» με το πρόγραμμα ώστε να γίνει πλήρως κατανοητή η λειτουργία της ουράς.

B. Ασκήσεις
Εφαρμογή 1: Ταμεία Τράπεζας

- Κατασκευάστε ένα πρόγραμμα το οποίο θα προσομοιώνει τη λειτουργία μιας τράπεζας με δύο ταμεία
- Κάθε ταμείο (από τα δύο) θα είναι μια ουρά συμβολοσειρών 80 χαρακτήρων. Κάθε ουρά θα χωράει το πολύ 5 πελάτες. Στην συμβολοσειρά θα αποθηκεύεται το επώνυμο του πελάτη.
- Το κυρίως πρόγραμμα θα υλοποιεί τις εξής ενέργειες (μέσα από ένα μενού επιλογών):
  - Άφιξη Πελάτη. Θα διαβάζεται από την είσοδο το επώνυμο του πελάτη και έπειτα: Αν «χωράει» στην πρώτη ουρά, τότε θα τοποθετείται σε αυτήν, αλλιώς θα τοποθετείται στην δεύτερη ουρά. Αν δεν χωράει σε καμία ουρά, τότε ο πελάτης δεν θα γίνεται δεκτός.
  - Αναχώρηση Πελάτη. Θα επιλέγεται τυχαία ένα ταμείο που έχει πελάτη και θα απομακρύνεται ο 1ος πελάτης του ταμείου. Στην οθόνη θα τυπώνεται το επώνυμό του και το ταμείο από το οποίο εξυπηρετήθηκε.
  - Τέλος Προγράμματος. Θα γίνεται εφικτή, μόνο αν και τα δύο ταμεία είναι άδεια. Σε αντίθετη περίπτωση θα βγάζει μήνυμα για το ποια ταμεία έχουν ακόμη πελάτη.

Υπόδειξη:
- Δεδομένου ότι η κυκλική ουρά είναι «πιο γρήγορη» συνίσταται χρησιμοποιήσετε αυτή, αντί για την απλή ουρά.

Β. Ασκήσεις
Εφαρμογή 2: Ουρά Προτεραιότητας

- Η ουρά προτεραιότητας είναι μία δομή δεδομένων, όπου κάθε στοιχείο που πρόκειται να εισαχθεί στην ουρά έχει και έναν πρόσθετο ακέραιο αριθμό που καθορίζει την προτεραιότητά του (όσο μεγαλύτερη η τιμή του αριθμού, τόσο μεγαλύτερη και η προτεραιότητα).
- Ένας τρόπος για να υλοποιήσουμε την δομή δεδομένων «Ουρά Προτεραιότητας» είναι:
  - Να τοποθετήσουμε έναν ακόμη ακέραιο αριθμό στο στοιχείο της δομής που καθορίζει την προτεραιότητα.
  - Η εισαγωγή του στοιχείου να γίνεται στην σωστή θέση της ουράς (με βάση την προτεραιότητα)
- Κατασκευάστε ένα πρόγραμμα επίδειξης της λειτουργίας της ουράς προτεραιότητας, τροποποιώντας κατάλληλα το project που κατασκευάσαμε για την κυκλική ουρά.

Υπόδειξη:
- Δεδομένου ότι η κυκλική ουρά είναι «πιο γρήγορη» συνίσταται χρησιμοποιήσετε αυτή, αντί για την απλή ουρά.

Β. Ασκήσεις
Εφαρμογή 3: Εκτύπωση Στοιχείων Ουράς

- Θεωρείστε την κυκλική ουρά ακεραίων που κατασκευάσαμε στο μάθημα.
- Θέλουμε να κατασκευάσουμε την δευτερεύουσα πράξη

```
void QU_print(QUEUE *q)
```

  - Η οποία θα τυπώνει τα στοιχεία της ουράς.

- Επεκτείνετε το project που κατασκευάσαμε για την κυκλική ουρά ώστε να ενσωματώνει και αυτήν τη συνάρτηση.
  - Η συνάρτηση αυτή πρέπει να δηλωθεί στην main, καθώς θεωρείται δευτερεύουσα πράξη.
  - Προσοχή όμως! Η συνάρτηση αυτή θεωρείται δευτερεύουσα οπότε ΔΕΝ πρέπει να έχει πρόσβαση στα μέλη της δομής.
  - Υπόδειξη: Μπορείτε όμως να ορίσετε βοηθητικές δομές (όπως π.χ. μία προσωρινή ουρά)

- Στο κυρίως πρόγραμμα και συγκεκριμένα, στην επιλογή για την εκτύπωση της ουράς, να καλείται αυτή η συνάρτηση.

Β. Ασκήσεις
Εφαρμογή 4: Αντιστροφή Ουράς

- Επεκτείνετε το project της εφαρμογής 3, έτσι ώστε να ορίζεται και η εξής δευτερεύουσα πράξη στις δηλώσεις της ουράς:

```
void QU_reverse(QUEUE *q)
```

  - Η συνάρτηση αυτή θα αντιστρέφει τα στοιχεία της ουράς (το πρώτο στοιχείο να γίνεται τελευταίο, το δεύτερο προτελευταίο, κ.οκ.)
  - Συνίσταται να χρησιμοποιήσετε μία στοίβα!