Γραμμικές Δομές Δεδομένων

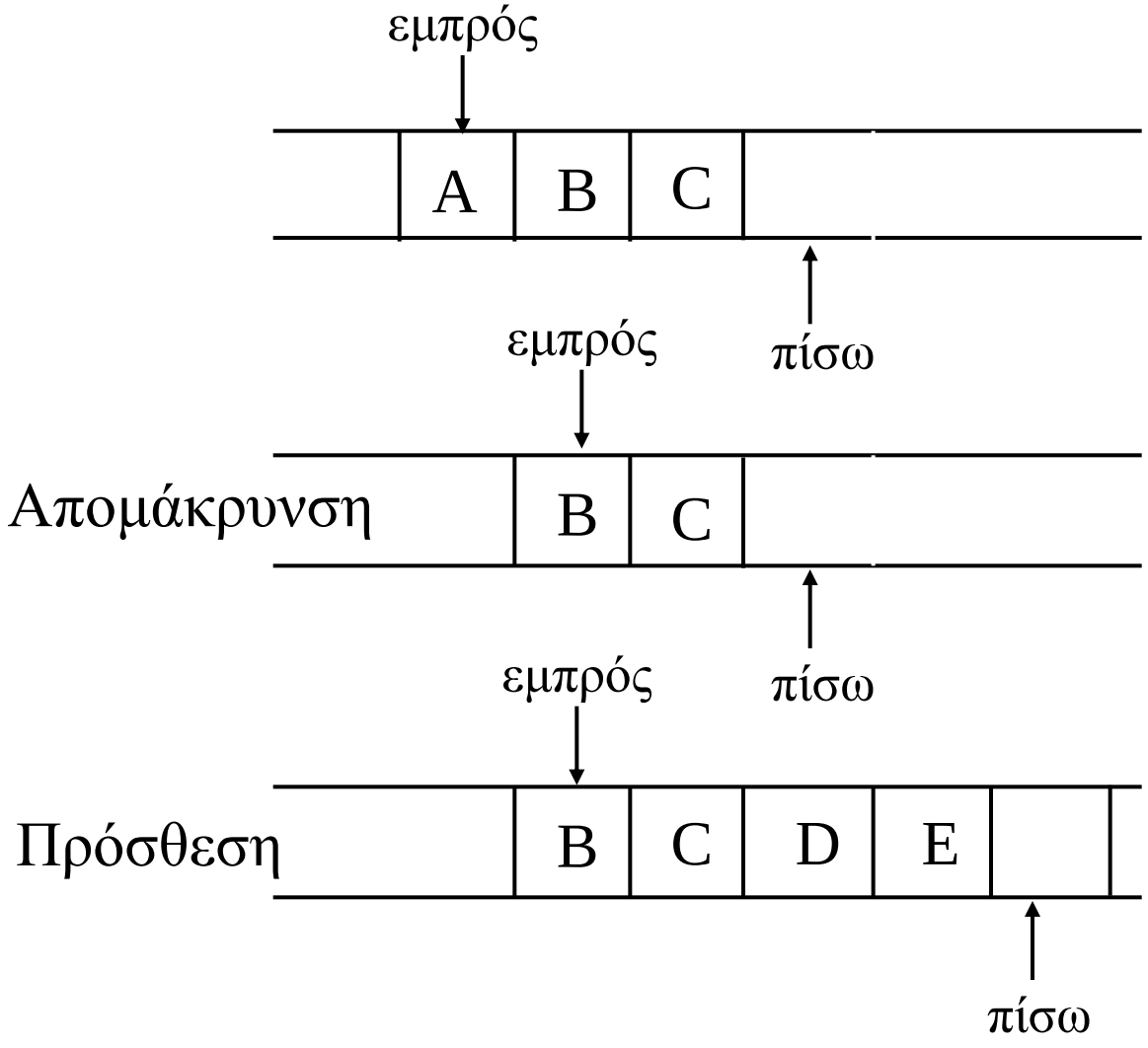
- Συλλογή δεδομένων στη σειρά (ολική διάταξη). Προσθέτουμε δεδομένα στη Δομή (μεγαλώνει) ή αφαιρούμε δεδομένα (μικραίνει)
- Αν περιορίσουμε τις πράξεις
  - Στην Αρχή μόνο – Στοίβα (stack)
  - Είσοδος αρχή & έξοδος τέλος - Ουρά (queue)

Επισκόπηση Μαθήματος

- ΑΤΔ Ουρά
- Εναλλακτικοί Σχεδιασμοί
- Προγραμματισμός με Ενότητες
  - Interface.h
  - Implementation.c

Ουρές (Queues)

FIFO



Σύμβαση:

Εμπρός: η πρώτη (κατειλημμένη) θέση για απομάκρυνση

Πίσω: η πρώτη (ελεύθερη) θέση για πρόσθεση

Ορισμός

Η ουρά είναι μια πεπερασμένη συλλογή δεδομένων τύπου Τ σε μια γραμμική ακολουθία.

Βασικές πράξεις

Κενή

Πρόσθεση

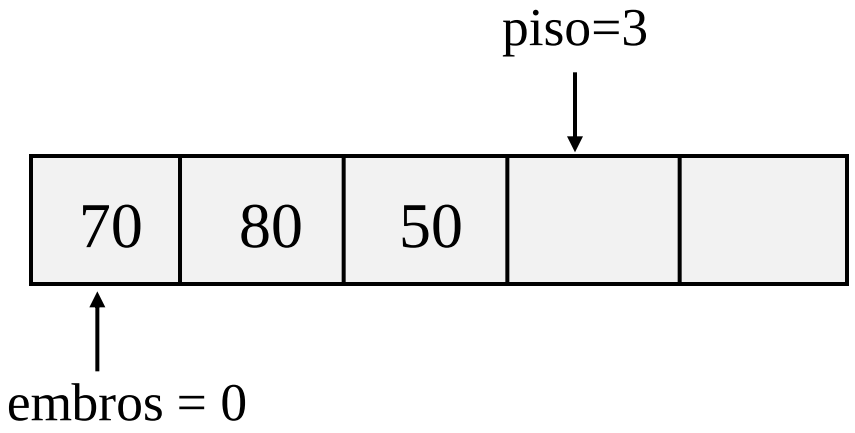
Απομάκρυνση

Πλήρης (ως βοηθητική συνάρτηση)

Αρχικοποίηση (Δημιουργία)

Σχεδιασμός Υλοποίησης με πίνακα

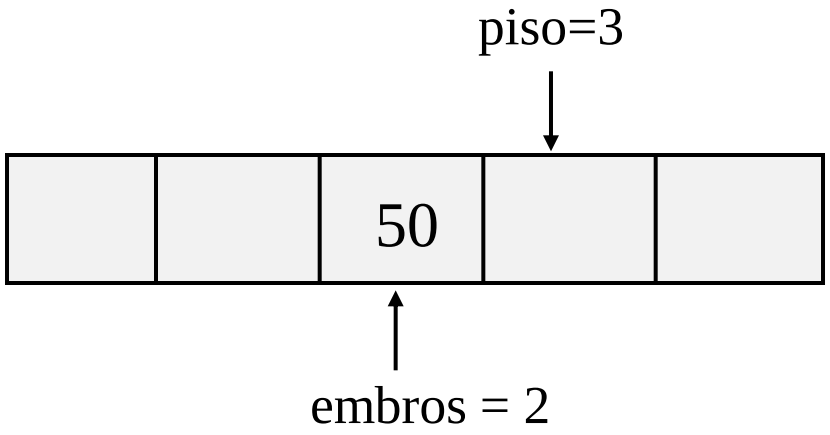
(χωρίς μετακινήσεις στοιχείων)



Αρχικά κενή
embros==0
piso ==0

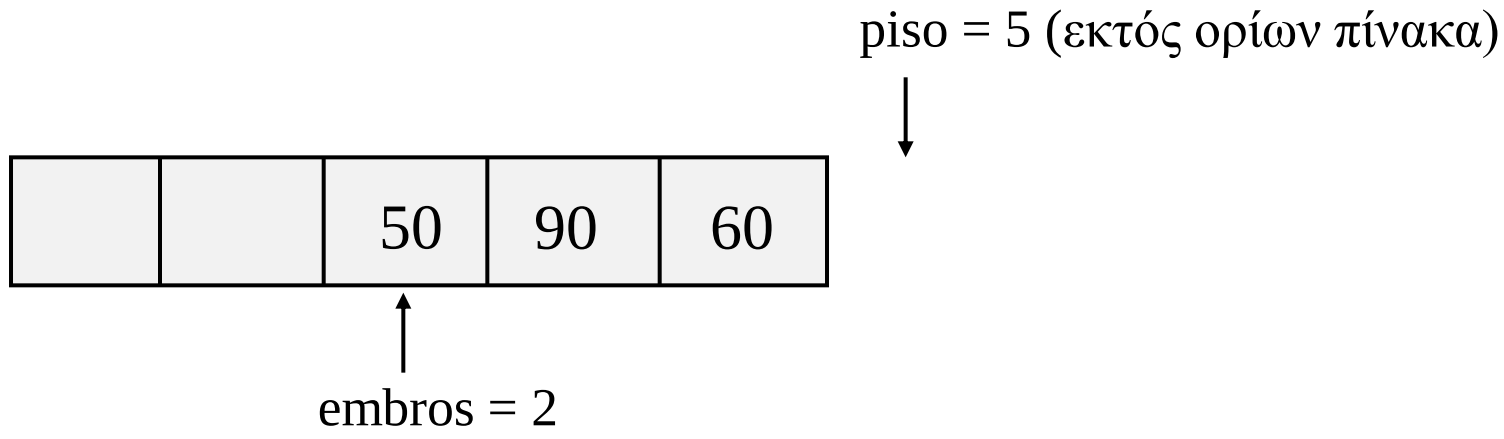
Είσοδος 3
Στοιχείων
embros==0
Piso==3

Αν απομακρύνουμε δύο στοιχεία, τότε έχουμε :

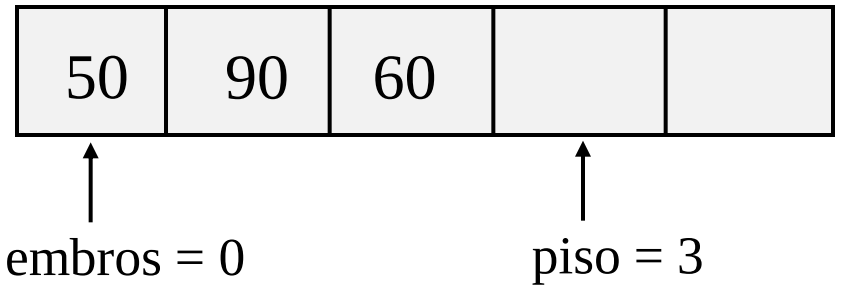


Απομάκρυνση 2
Στοιχείων
embros==2
piso==3

Η πρόσθεση των 90 και 60 θα τροποποιήσει το σχήμα στο παρακάτω :



Χώρος υπάρχει. Πώς θα γίνει η πρόσθεση ενός επιπλέον στοιχείου;



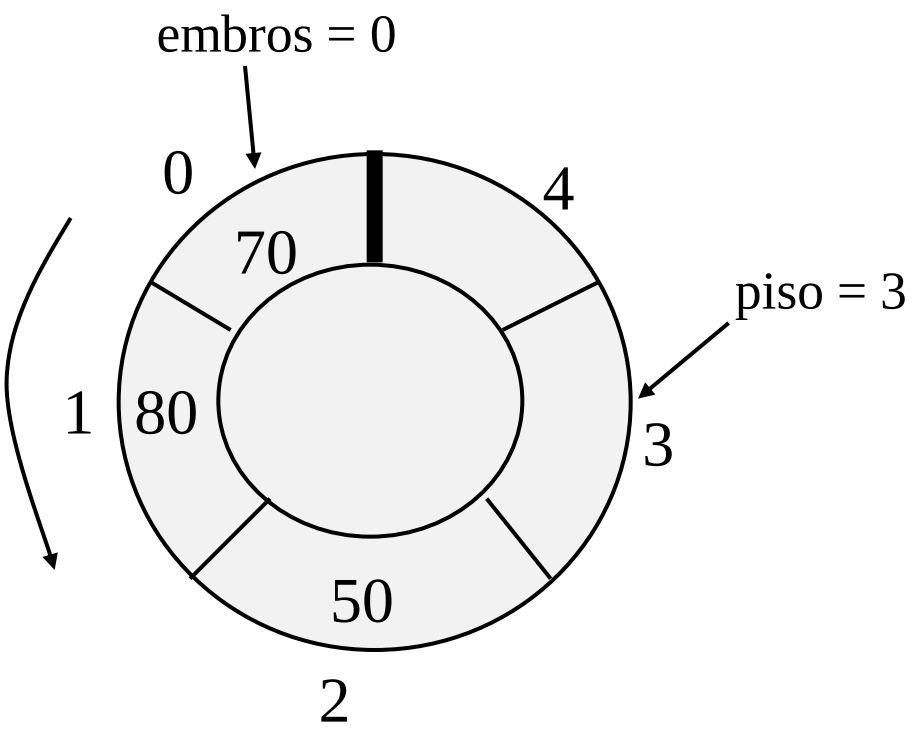
Με Μετατόπιση.
Κάθε στοιχείο μετατοπίζεται κατά n θέσεις O(n)

Μπορούμε καλύτερα!

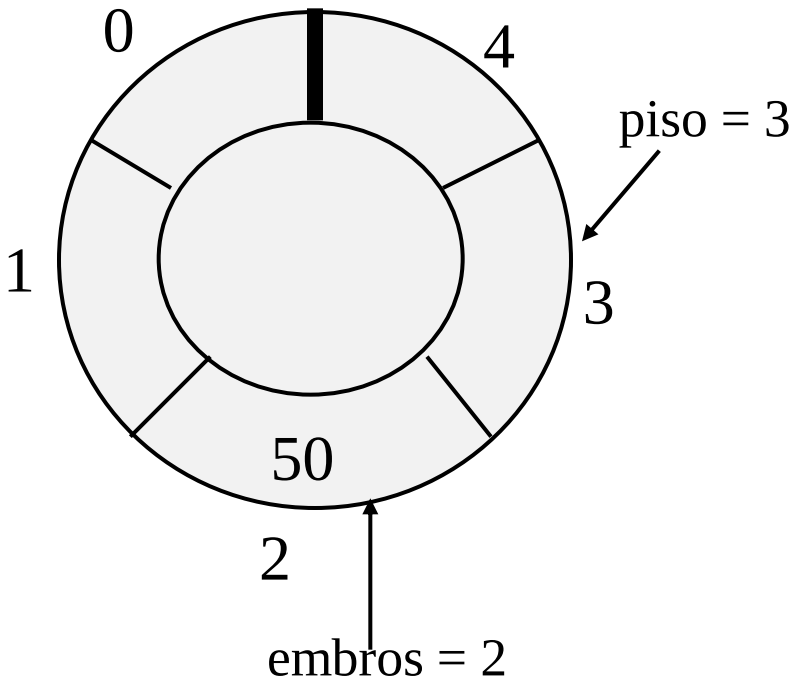
Η μετακίνηση των στοιχείων της ουράς μπορεί να αποφευχθεί αν φανταστούμε ότι ο πίνακας που φιλοξενεί την ουρά είναι **κυκλικός (circular)**, με το πρώτο στοιχείο του να «έπεται» του τελευταίου.

Πρόσθεση
70, 80, 50

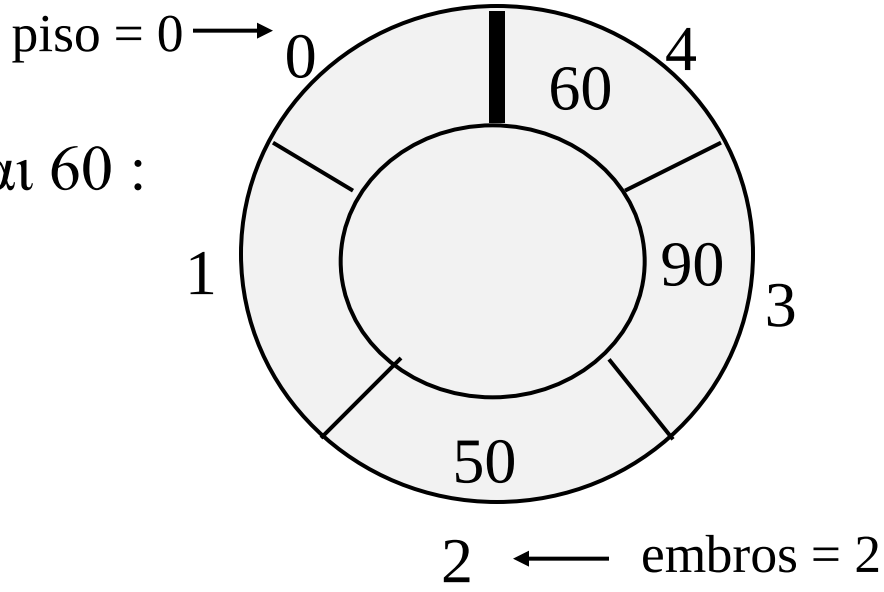
Ο piso προσδιορίζει την πρώτη διαθέσιμη θέση
Ο embros την πρώτη κατειλημμένη



Απομάκρυνση των 70 και 80



Πρόσθεση των 90 και 60 :



Μετακίνηση του piso :

Αν piso == (plithos-1) τότε plithos

piso = 0

διαφορετικά

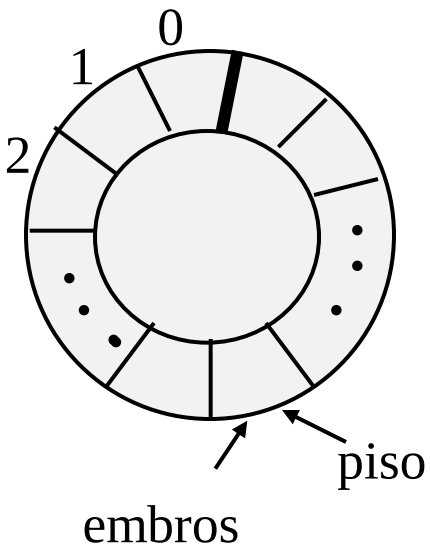
piso = piso+1

ή piso = (piso+1) % plithos

Μετακίνηση του embros :

embros = (embros+1) % plithos

- Κενή Ουρά

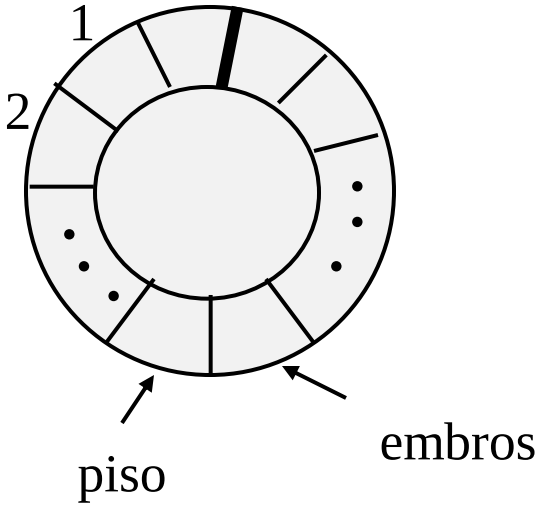


Κριτήριο κενής ουράς ;;;

embros = = piso

Το ίδιο για κριτήριο γεμάτης !!!

- Πλήρης Ουρά



Α' ΛΥΣΗ με μια κενή θέση

Συνθήκη για την γεμάτη ουρά

(piso+1) % plithos == embros

Για κενή
embros == piso

Προγραμματισμός Σχεδιασμού

```
#define plithos …
typedef  ... typos_stoixeiou;

typedef struct {
     int embros, piso;
     typos_stoixeiou pinakas [plithos];
} typos_ouras;

typos_ouras oura ;
```

Η κλήση της δημιουργίας μιας κενής ουράς θέτει στους embros και piso την αρχική τιμή 0.

```
void dimiourgia(typos_ouras *ouraPtr)
/* Προ: Καμία
    Μετά: Δημιουργία κενής ουράς. */
{
     ouraPtr->embros=0;
     ouraPtr->piso=0;
}
```

```
int keni (typos_ouras oura){
/* Προ: Δημιουργία
   Μετά: Η συνάρτηση επιστρέφει 1 αν η oura
     είναι κενή αλλιώς επιστρέφει 0. */

     return ((oura.embros)==(oura.piso));
}

int gemati(typos_ouras oura){ // μία κενή θέση
/* Προ: Δημιουργία
   Μετά: Αν η ουρά είναι γεμάτη τότε η
συνάρτηση επιστρέφει 1 αλλιώς επιστρέφει 0 */

    return((oura.piso+1)%plithos== oura.embros)
}
```

Η πρόσθεση ενός στοιχείου στην ουρά γίνεται στη θέση που δείχνει ο δείκτης piso, ο οποίος **μετά** την πρόσθεση προχωρά μια θέση.

```
void prosthesi(typos_ouras *ouraPtr,
                  typos_stoixeiou stoixeio,
                  int * full){
/* Προ : Δημιουργία ουράς και η ουρά δεν είναι γεμάτη.
   Μετά: το stoixeio προστίθεται στην ουρά */

      if (gemati(*ouraPtr))
                  *full = 1;
      else{
            *full = 0;
            ouraPtr->pinakas[ouraPtr->piso] = stoixeio;
            ouraPtr->piso = (ouraPtr->piso+1)%plithos;
       }
}
```

Η απομάκρυνση ενός στοιχείου γίνεται από εμπρός, δηλαδή το στοιχείο που δείχνει ο δείκτης embros και αυξάνεται κατά ένα (%).

```
void apomakrynsi( typos_ouras *ouraPtr,
                  typos_stoixeiou *stoixeioPtr,
                  int *empty)
/* Προ :Δημιουργία και η ουρά δεν είναι κενή.
   Μετά:Ανάκτηση και απομάκρυνση του πρώτου στοιχείου
        της ουράς. Η τιμή του καταχωρείται στο stoixeio.
*/

      if (keni(*ouraPtr))
         *empty = 1;
      else{
        *empty = 0;
        *stoixeioPtr=ouraPtr->pinakas[ouraPtr->embros];
         ouraPtr->embros=(ouraPtr->embros+1) % plithos;
       }
}
```

Β' Υλοποίηση ουράς με λογική μεταβλητή για Κενή

Στην υλοποίηση αυτή χρησιμοποιείται μια λογική μεταβλητή προκειμένου να ελεγχθεί αν μια ουρά είναι κενή ή γεμάτη, ώστε να μην αφήνουμε κενή θέση. `empros==piso` και στις δυο περιπτώσεις, αλλά χρησιμοποιούμε την `adeia` για διαφοροποίηση. Οι δηλώσεις είναι τώρα οι εξής:

```
#define plithos  …
typedef  ...  typos_stoixeiou;

typedef struct {
     int embros, piso, adeia;
     typos_stoixeiou pinakas[plithos];
} typos_ouras;
```

Λαμβάνοντας υπόψη τις παραπάνω δηλώσεις, οι υλοποιήσεις των βασικών πράξεων για τον ΑΤΔ ουρά τροποποιούνται ως εξής:

```
void dimiourgia(typos_ouras *ouraPtr){
/* Προ : Καμμία.
   Μετά: Η oura είναι μια κενή ουρά. */

     ouraPtr->embros=0;
     ouraPtr->piso=0;
     ouraPtr->adeia=1;
}
```

```
int keni(typos_ouras oura){
/* Προ : Δημιουργία ουράς
   Μετά: Η συνάρτηση επιστρέφει 1 αν η oura
         είναι κενή αλλιώς επιστρέφει 0. */

     return (oura.adeia);
}
```

```
int gemati(typos_ouras oura){
/* Προ : Δημιουργία ουράς.
   Μετά: Αν η ουρά είναι γεμάτη τότε
     επιστρέφει 1 διαφορετικά επιστρέφει 0 */

     return (        (oura.embros==oura.piso)
                &&   !keni(oura));
}
```

```
void prosthesi(typos_ouras *ouraPtr,
               typos_stoixeiou stoixeio,
               int *full){
/* Προ : Δημιουργία ουράς.
   Μετά: Εισάγεται η τιμή του stoixeio. */

   if (gemati(*ouraPtr))
     *full=1;
   else{
     *full =0;
     ouraPtr->pinakas[ouraPtr->piso]=stoixeio;
     ouraPtr->piso=(ouraPtr->piso+1) % plithos;
     ouraPtr->adeia=0;
    }
}
```

```
void apomakrynsi( typos_ouras *ouraPtr,
                  typos_stoixeiou *stoixeio,
                  int *empty){
/* Προ : Δημιουργία ουράς.
   Μετά: Απομάκρυνση του πρώτου στοιχείου.
*/

   if (keni(*ouraPtr))
      *empty=1;
   else{
      *empty=0;
      *stoixeio=ouraPtr->pinakas[ouraPtr->embros];
      ouraPtr->embros=((ouraPtr->embros+1)% plithos);
      ouraPtr->adeia=(ouraPtr->embros==ouraPtr->piso);
   }
}
```

Γ' Υλοποίηση ουράς με μετρητή

Στην περίπτωση αυτή χρησιμοποιείται ένας μετρητής για τον υπολογισμό των θέσεων στην ουρά. Καλύτερη χρήση του επιπλέον int. Μπορούμε να προσθέσουμε την πράξη megethos. Οι δηλώσεις για μια ουρά έχουν τη μορφή :

```
#define plithos ...
typedef  ...  typos_stoixeiou;
typedef
typedef struct {
     int embros, piso, metritis;
     typos_stoixeiou pinakas[plithos];
} typos_ouras;
typos_ouras    oura;
```

Στη συνέχεια παρουσιάζονται οι υλοποιήσεις των βασικών πράξεων για μια ουρά:

```
void dimiourgia(typos_ouras *ouraPtr){
     ouraPtr->metritis=0;
     ouraPtr->embros=0;
     ouraPtr->piso=0;
}
```

```
int keni(typos_ouras oura){
/*Προ : Η ουρά έχει δημιουργηθεί.
  Μετά : Επιστρέφει 1 αν η ουρά είναι κενή
         και 0 αν όχι*/

     return (oura.metritis == 0);
}
```

```
int gemati(typos_ouras oura){
/*Προ : Η oura έχει δημιουργηθεί.
  Μετά: Η συνάρτηση επιστρέφει 1 αν η oura
        είναι γεμάτη και 0 όχι*/


     return (oura.metritis == plithos);
}
```

```
void prosthesi(typos_ouras *ouraPtr,
               typos_stoixeiou stoixeio,
               int *full){
/* Προ: Η oura έχει δημιουργηθεί.
   Μετά: Το stoixeio έχει εισαχθεί στην ουρά
*/

  if (gemati(*ouraPtr))
     *full = 1;
  else{
     *full=0;
     ouraPtr->metritis++;
     ouraPtr->pinakas[ouraPtr->piso]=stoixeio;
     ouraPtr->piso=(ouraPtr->piso+1) % plithos;
  }
}
```

```
void apomakrynsi(typos_ouras *ouraPtr,
                 typos_stoixeiou *stoixeioPtr,
                  int *empty){
/* Προ : Η oura δεν είναι κενή.
   Μετά : Το πρώτο στοιχείο της ουράς απομακρύνθηκε και η
          τιμή του καταχωρήθηκε στο stoixeio
*/

   if (keni(*ouraPtr))
      *empty=1;
   else{
      *empty = 0;
      ouraPtr->metritis--;
      *stoixeioPtr=ouraPtr->pinakas[ouraPtr->embros];
      ouraPtr->embros=(ouraPtr->embros+1)%plithos;
   }
}
```

Εφαρμογή : Προσομοίωση ουράς αναμονής

Μελέτη συμπεριφοράς μιας ουράς σε ένα ταμείο

Δεδομένα προσομοίωσης

- Πιθανότητα άφιξης ενός πελάτη εντός ενός λεπτού (0,1)
- Χρόνος (σταθερός) εξυπηρέτησης ενός πελάτη
- Συνολικός χρόνος προσομοίωσης

Απεικόνιση των αντικειμένων (δεδομένων).

Η προσομοίωση εκτελεί Ν επαναλήψεις, όπου Ν είναι η διάρκεια (σε λεπτά) της προσομοίωσης. Ο **ΧΡΟΝΟΣ** απεικονίζεται με ακέραιο (int) με αρχική τιμή 0 και αυξάνει κατά 1 σε κάθε επανάληψη. Στόχος της προσομοίωσης είναι ο υπολογισμός της **μέσης τιμής του χρόνου αναμονής**. Μετράμε το πλήθος των πελατών που εξυπηρετήθηκαν και τον συνολικό χρόνο αναμονής τους.

Για κάθε **ΠΕΛΑΤΗ** προσθέτουμε στην ουρά τον **χρόνο που εισήλθε** (int). Ο χρόνος αναμονής είναι η διαφορά χρόνου εξόδου και χρόνου εισόδου.

Ο **ΤΑΜΙΑΣ** απεικονίζεται με τον χρόνο εξυπηρέτησης που απαιτείται για κάθε πελάτη (int). Ίδιος (xronos_eksipiretisis) για όλους τους πελάτες.
Όταν ένας **ΠΕΛΑΤΗΣ** απομακρύνεται από την ουρά ο ταμίας είναι απασχολημένος για xronos_eksipiretisis λεπτά. Μια μεταβλητή που απεικονίζει τον enapomenon_xronos=xronos_eksipiretisis, ελαττώνεται κατά 1 σε κάθε επανάληψη. Όταν ο μετρητής αυτός λαμβάνει την τιμή 0, ο ταμίας είναι ελεύθερος και απομακρύνεται από την ουρά ο επόμενος πελάτης (αν υπάρχει, αλλιώς αδρανής).

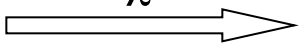
Σε κάθε επανάληψη εξετάζουμε α) αν έχουμε νέα είσοδο πελάτη και β) αν ο ταμίας είναι ελεύθερος για να απομακρυνθεί ο επόμενος πελάτης από την ουρά.

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

#include "oura.h"

int main(void)
{
   typos_ouras oura; /* ουρά πελατών */
   float pithanotita_aphiksis;
          /* πιθανότητα άφιξης πελάτη σε ένα λεπτό */
   unsigned int xronos_eksipiretisis;
          /* χρόνος για την εξυπηρέτηση ενός πελάτη */
   unsigned int xronos_prosomoiosis;
          /* συνολικός χρόνος προσομοίωσης */
```

συνέχεια

```
unsigned int xronos;
            /* το ρολόϊ της προσομοίωσης */
unsigned int enapomenon_xronos;
            /* χρόνος που απομένει για το τέλος της
                  εξυπηρέτησης  ενός πελάτη */

unsigned int arithmos_pelaton;
      /* πλήθος των πελατών που εξυπηρετήθηκαν */

unsigned int xronos_anamonis;
            /*συνολικός χρόνος αναμονής */
unsigned int xronos_eisodou;
            /* η ώρα που εισήλθε ο πελάτης στην ουρά */

float mesos_xronos; /* μέσος χρόνος αναμονής */

float random;
```

συνέχεια



```
scanf("%d %f %d",
                  &xronos_prosomoiosis,
                  &pithanotita_aphiksis,
                  &xronos_eksipiretisis);

printf("Η προσομοίωση θα διαρκέσει: “);
printf("%d λεπτά.\n",xronos_prosomoiosis);
printf("Η πιθανότητα άφιξης πελάτη");
printf("%4.2f.\n",pithanotita_aphiksis);
printf("Η διάρκεια εξυπηρέτησης πελάτη:");
printf("%d λεπτά\n",xronos_eksipiretisis);
```

συνέχεια

```
dimiourgia (&oura);

xronos = 0;
enapomenon_xronos = 0; /* χρόνος ταμία */

arithmos_pelaton = 0;
xronos_anamonis = 0;
srand(time(NULL));
```

συνέχεια

```
while (xronos < xronos_prosomoiosis)
{     random=((float) rand())/(float) RAND_MAX ;

      if (random < pithanotita_aphiksis )
            prosthesi(&oura, xronos);

      if (enapomenon_xronos == 0){ /* ελεύθερος ταμίας*/
          if (!keni(oura)){        /* υπάρχει πελάτης */
            apomakrynsi(&oura, &xronos_eisodou);
            enapomenon_xronos =xronos_eksipiretisis;

            xronos_anamonis += (xronos - xronos_eisodou);
            arithmos_pelaton++;
          }
      } else if (enapomenon_xronos > 0)
            enapomenon_xronos--;

       xronos++;
}  /* while */
```

συνέχεια

```
if (arithmos_pelaton == 0)
      mesos_xronos = 0.0;
else
      mesos_xronos =(float)xronos_anamonis /
(float)arithmos_pelaton;

printf("Εξυπηρετήθησαν %5d πελάτες”, arithmos_pelaton);

printf(“Ο μέσος χρόνος αναμονής ήταν %4.2f λεπτά.\n",
            mesos_xronos);

}
```