Διάλεξη 05: Αφηρημένοι Τύποι Δεδομένων

Στην ενότητα αυτή θα μελετηθούν τα εξής επιμέρους θέματα:

- Αφηρημένοι Τύποι Δεδομένων (ΑΤΔ)
- Οι ΑΤΔ Στοίβα και Ουρά
- Υλοποίηση των ΑΤΔ Στοίβα και Ουρά

Διδάσκων: Παναγιώτης Ανδρέου

Αφηρημένοι Τύποι Δεδομένων

- **Τύπος Δεδομένων:** ένας **τύπος** μαζί με ένα σύνολο **πράξεων** για τη **δημιουργία** και **επεξεργασία** δεδομένων του τύπου (int, char, float)
- **Αφηρημένος Τύπος Δεδομένων (ΑΤΔ):** μαθηματικό μοντέλο που αποτελείται από
  - ένα ή περισσότερα **πεδία ορισμού** και
  - ένα σύνολο **πράξεων** για επεξεργασία των πεδίων ορισμού.
- Όταν μιλούμε για ένα ΑΤΔ μας ενδιαφέρει η **προδιαγραφή** του και πως θα τον χρησιμοποιήσουμε. **Δεν μας ενδιαφέρει ο τρόπος υλοποίησής του μέσα στη μηχανή.** (Η υλοποίηση ενός ΑΤΔ μπορεί να αλλάξει χωρίς να επηρεάσει την ορθότητα προγραμμάτων που τον χρησιμοποιούν.)

Αφηρημένοι Τύποι Δεδομένων (συν.)

Παραδείγματα:

1. Ο **τύπος** *int* μαζί με τις πράξεις +, *, /, =, είναι ένας τύπος δεδομένων.
   - Για να τον χρησιμοποιήσουμε πρέπει να γνωρίζουμε το σύνολο των πράξεων.
   - Ο τρόπος αναπαράστασής του στον υπολογιστή δεν μας ενδιαφέρει
2. Ο **τύπος** *double* μαζί με τις πράξεις +, *, /, =, είναι ένας τύπος δεδομένων.
3. Ο **τύπος** *float* μαζί με τις πράξεις +, *, /, =, είναι ένας τύπος δεδομένων.

Αφηρημένοι Τύποι Δεδομένων (συν.)

4. Ο **<u>Αφηρημένος Τύπος Δεδομένων (ΑΤΔ)</u>** *Ουρά Προτεραιότητας, το οποίο* είναι ένα σύνολο στοιχείων τύπου *key* (π.χ., key=int ή key=char ή key = (int,int)), συνοδευόμενο από τις πιο κάτω πράξεις.
   - *δημιούργησε την άδεια ουρά προτεραιότητας, q,*
   - *έλεγξε αν η ουρά q είναι άδεια,*
   - *βάλε το στοιχείο k στην ουρά q,*
   - *αφαίρεσε και επίστρεψε το μικρότερο στοιχείο της q (σύμφωνα με τη γραμμική διάταξη της ουράς).*

- **Ένας ΑΤΔ μπορεί να υλοποιηθεί με πολλούς τρόπους,** π.χ. μια ουρά προτεραιότητας μπορεί να υλοποιηθεί από δομές λίστας, δενδρικές δομές κλπ.

Λίστες

- **Λίστα**: μια ακολουθία στοιχείων (π.χ., Queue ή Stack όπως θα δούμε στην συνέχεια)

$$\Lambda = \alpha_1, \alpha_2, \ldots, \alpha_n$$

- Αναφερόμαστε στα στοιχεία της λίστας ως **κόμβους**. Με $\Lambda[i]$ θα αναφερόμαστε στο i-οστό στοιχείο της λίστας.
- **Μήκος** μιας λίστας Λ ονομάζεται ο αριθμός των στοιχείων της και συμβολίζεται ως $|\Lambda|$.
- Αν $|\Lambda| = 0$ τότε αναφερόμαστε στην **κενή λίστα** την οποία συμβολίζουμε ως $\langle\rangle$.

Λίστες (συν.)

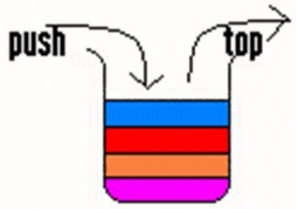
- Συνοδεύοντας λίστες με ένα σύνολο πράξεων μπορούμε να ορίσουμε **αφηρημένους τύπους δεδομένων**. Χρήσιμες πράξεις περιλαμβάνουν τις πιο κάτω:
  - Δημιουργία λίστας
  - Εισαγωγή νέου κόμβου στη λίστα
  - Εξαγωγή κόμβου από τη λίστα
  - Εύρεση κόμβου με ορισμένη ιδιότητα
  - Διάταξη της λίστας σύμφωνα με κάποια σχέση

Λίστες (συν.)

- Οι πιο σημαντικές πράξεις στον ορισμό ενός ΑΤΔ-λίστας είναι η **εισαγωγή** και η **εξαγωγή** κόμβων στα **άκρα της λίστας**.
- Με βάση την προδιαγραφή αυτών των πράξεων, διακρίνουμε **δύο βασικούς τύπους λίστας** που έχουν πολλές και σημαντικές εφαρμογές σε κλάδους επιστημών που χρησιμοποιούν υπολογιστικές μεθόδους. Είναι οι ακόλουθες:
  - Η ***στοίβα* (stack)** που έχει μόνο ένα άκρο προσιτό για εισαγωγές και εξαγωγές κόμβων. ***(LIFO – Last In First Out)***
  - Η ***ουρά* (queue)** όπου γίνονται εισαγωγές στο ένα άκρο και εξαγωγές από το άλλο. ***(FIFO – First In First Out)***
- Υπάρχουν και άλλοι ΑΤΔ-λίστας μικρότερης πρακτικής σημασίας, όπως: ουρά με δύο άκρα, πολλαπλή στοίβα, κλπ.

ΑΤΔ Στοίβα (stack)

- Ορίζουμε μια στοίβα S ως μια λίστα συνοδευόμενη από τις πιο κάτω πράξεις:



makeEmpty()	δημιούργησε την κενή στοίβα ⟨⟩.
isEmpty(S)	επέστρεψε τη λογική τιμή που εκφράζει το αν η S είναι κενή.
push(x)	εισήγαγε τον κόμβο x στη στοίβα S.
pop()	διέγραψε τον κόμβο κορυφής της S.
top()	δώσε τον κόμβο κορυφής της S.
size()	επιστρέφει το πλήθος των στοιχείων της S.

ΑΤΔ Στοίβα (stack) (συν.)

Οι πράξεις αυτές προδιαγράφονται από τους εξής κανόνες:

isEmpty(makeEmpty) = true

isEmpty(push(x)) = false

pop(makeEmpty) = error

πολιτική LIFO

last in, first out

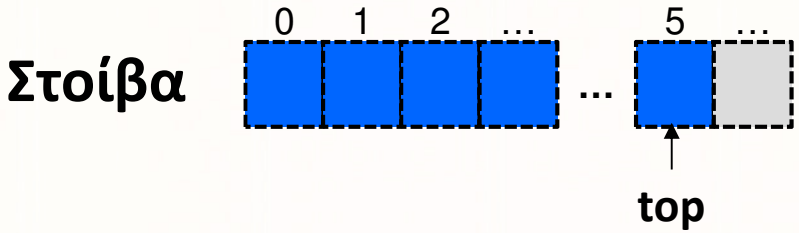
pop(push(x)) = S

top(makeEmpty()) = error

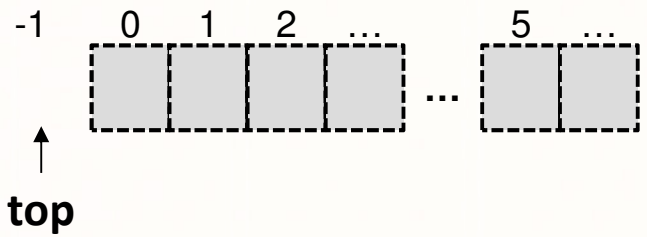
top(push(x)) = x

Αναπαράσταση Αλγορίθμων Στοίβας

- Με χρήση μίας μεταβλητής top και μίας συλλογής από στοιχεία



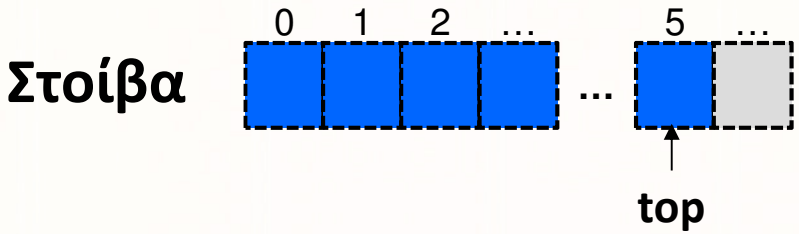

```
void makeEmpty(){
   // Θέσε top = -1;
   // Άδειασε τα στοιχεία της στοίβας.
}
```



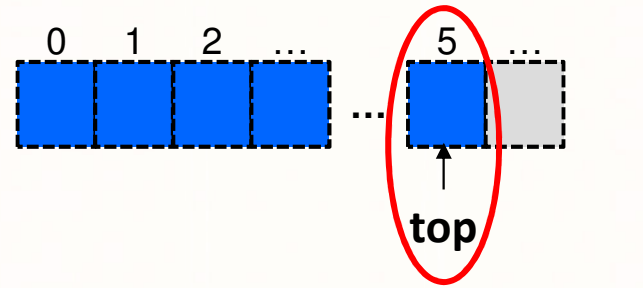

```
public boolean isEmpty(){
   // Αν το top = -1 επέστρεψε true αλλιώς false.
}

public int size() {
  // Επέστρεψε το top+1.
}
```

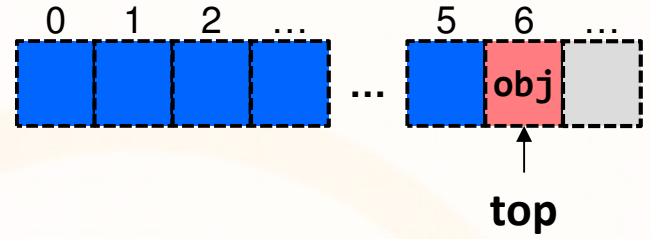
Αναπαράσταση Αλγορίθμων Στοίβας (συν.)

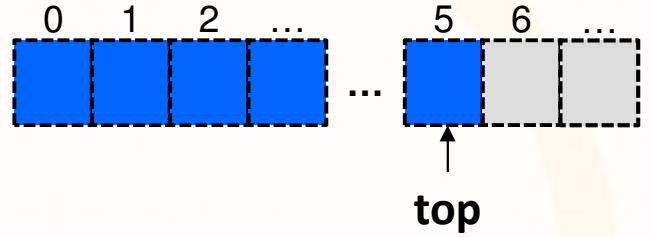
- Με χρήση μίας μεταβλητής top και μίας συλλογής από στοιχεία




```
public E top() {
 // Αν η στοίβα δεν είναι άδεια επέστρεψε το
 // στοιχειό στη θέση top, αλλιώς null.
}
public void push(obj) {
 // Αύξησε τη θέση top κατά 1.
 // Πρόσθεσε το καινούριο στοιχείο στην θέση top.
}
public void pop() {
 // Αν η στοίβα δεν είναι άδεια τότε διέγραψε
 // το στοιχειό στη θέση top.
 // Μείωσε τη θέση top κατά 1.
}
```





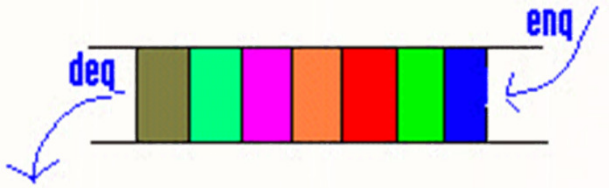


Παράδειγμα Χρήσης Στοίβας

Ενέργεια	Έξοδος	S
push(5)	–	{5}
push(3)	–	{3, 5}
pop()	– (3)	{5}
push(7)	–	{7, 5}
pop()	– (7)	{5}
top()	5	{5}
pop()	– (5)	{}
top()	null	{}
isEmpty()	true	{}
push(9)	–	{9}
push(7)	–	{7, 9}
size()	2	{7, 9}
push(3)	–	{3, 7, 9}
push(5)	–	{5, 3, 7, 9}
pop()	– (9)	{3, 7, 9}

ΑΤΔ Λίστα 2 : Ουρά (Queue)

- Νέες εισαγωγές γίνονται στο πίσω άκρο.
- Εξαγωγές από το μπροστινό άκρο



- Ορίζουμε μια ουρά Q ως μια λίστα συνοδευόμενη από τις πιο κάτω πράξεις:

makeEmpty()	δημιούργησε την κενή ουρά <>.
isEmpty()	επέστρεψε τη λογική τιμή που εκφράζει το αν η Q είναι κενή.
enQueue (x)	εισήγαγε τον κόμβο x στην ουρά Q.
deQueue()	διέγραψε τον κόμβο εξόδου της Q
top(Q)	δώσε τον κόμβο εξόδου της Q.
size()	επιστρέφει το πλήθος των στοιχείων της Q.

ΑΤΔ Λίστα 2 : Ουρά (Queue) (συν.)

- Οι πράξεις αυτές προδιαγράφονται από τους εξής κανόνες

isEmpty(makeEmpty( )) = true

isEmpty(enQueue(x)) = false

deQueue(makeEmpty( )) = error

πολιτική FIFO

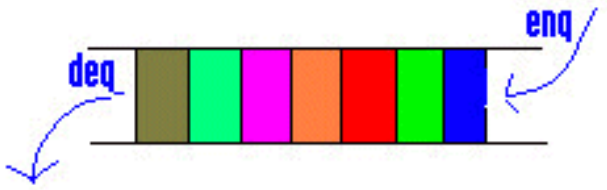
first in, first out

top (makeEmpty( )) = error

top(enQueue (x,Q)) = if isEmpty(Q) then x

else top(Q)

Προτού να εισάγουμε το x

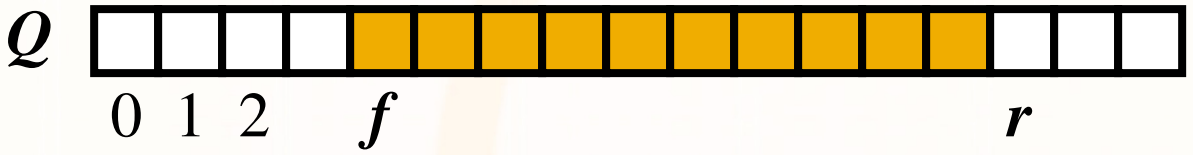


Αναπαράσταση Ουράς

Αναπαράσταση Ουράς με χρήση Πίνακα

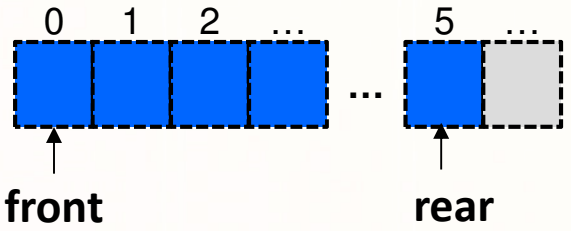
Για την αναπαράσταση μιας ουράς με στοιχεία $\alpha_0, \alpha_1, \ldots, \alpha_{n-1}$ χρειαζόμαστε ένα πίνακα A στον οποίο θα αποθηκεύσουμε τα στοιχεία της ουράς, $A[i\text{-}1] = \alpha_i$, και δύο δείκτες που προσδιορίζουν τα δύο προσιτά άκρα της ουράς.

- Έτσι χρησιμοποιούμε μια εγγραφή με τρία πεδία
  1. ένα πίνακα A[n],
  2. μια μεταβλητή front, τύπου *ακέραιος*, που συγκρατεί τη θέση εξόδου, και
  3. μια μεταβλητή rear, τύπου *ακέραιος*, που συγκρατεί τη θέση εισόδου.

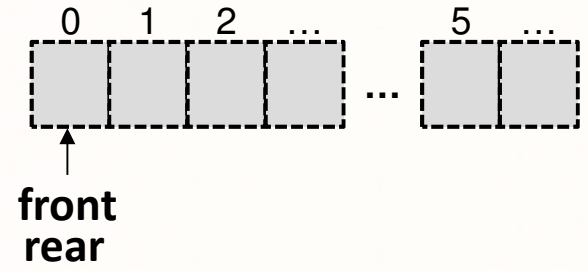


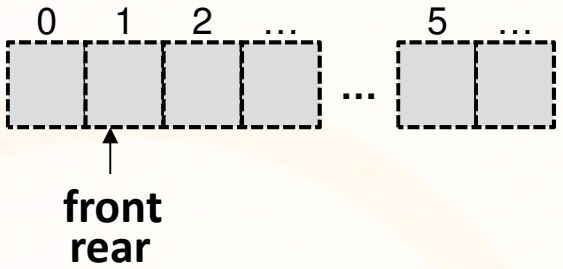
Αναπαράσταση Αλγορίθμων Ουράς

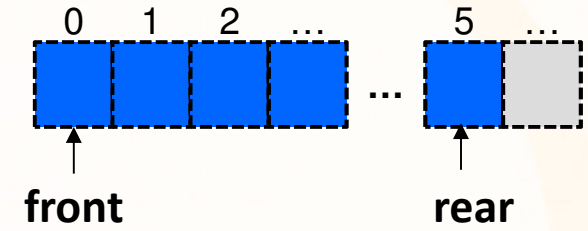
- Με χρήση δύο μεταβλητών front,rear και μίας συλλογής από στοιχεία




```
void makeEmpty(){
   // Θέσε front=rear= 0;
   // Άδειασε τα στοιχεία της ουράς.
}

public boolean isEmpty(){
   // Αν το front==rear επέστρεψε true
   // αλλιώς false.
}

public int size() {
  // Επέστρεψε το (rear – front + 1).
}
```

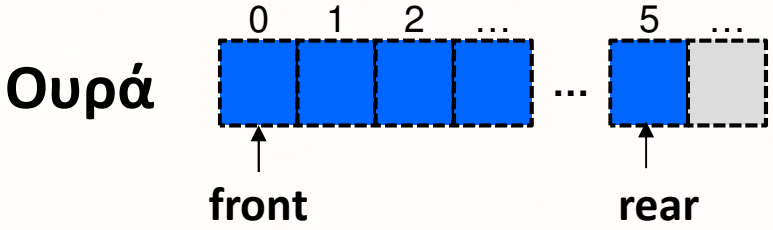




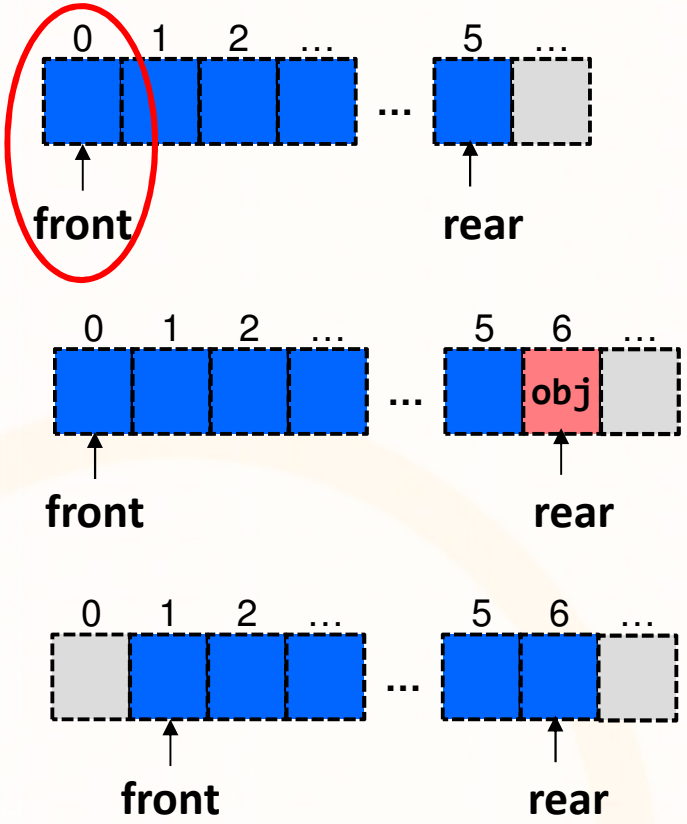


rear-front+1= 5-0+1= 6

Αναπαράσταση Αλγορίθμων Ουράς (συν.)




```
public E top() {
 // Αν η ουρά δεν είναι άδεια επέστρεψε το
 // στοιχειό στη θέση front, αλλιώς null.
}
public void enQueue(obj) {
 // Αύξησε τη θέση rear κατά 1.
 // Πρόσθεσε το καινούριο στοιχείο στην θέση rear.
}
public void deQueue() {
 // Αν η ουρά δεν είναι άδεια τότε διέγραψε
 // το στοιχειό στη θέση front.
 // Αύξησε τη θέση front κατά 1.
}
```



Παράδειγμα Χρήσης Ουράς

Ενέργεια	Έξοδος	Q
enqueue(5)	–	{5}
enqueue(3)	–	{5, 3}
dequeue()	– (5)	{3}
enqueue(7)	–	{3, 7}
dequeue()	– (3)	{7}
top()	7	{7}
dequeue()	– (7)	{}
dequeue()	– (null)	{}
isEmpty()	true	{}
enqueue(9)	–	{9}
enqueue(7)	–	{9, 7}
size()	2	{9, 7}
enqueue(3)	–	{9, 7, 3}
enqueue(5)	–	{9, 7, 3, 5}
dequeue()	– (9)	{7, 3, 5}

Υλοποίηση ΑΤΔ

- Οι προαναφερθείς ΑΤΔ μπορούν να υλοποιηθούν με διάφορες δομές δεδομένων χρησιμοποιώντας είτε **στατική** είτε **δυναμική χορήγηση** μνήμης.

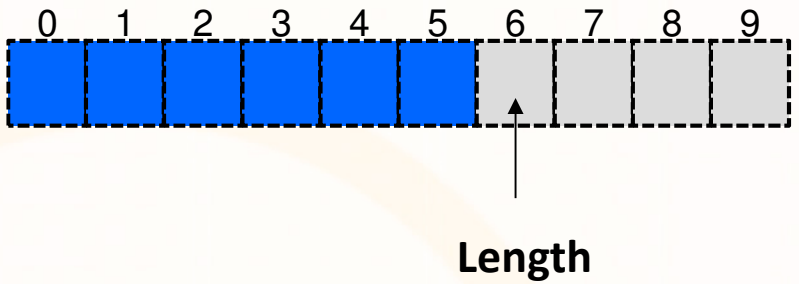
- **Στατική**: Δέσμευση μνήμης πριν την εκκίνηση προγράμματος
  - π.χ., δημιουργία πίνακα με *N* θέσεις

- **Δυναμική:** Δέσμευση μνήμης κατά την διάρκεια της εκτέλεσης
  - Χρησιμοποιώντας συλλογές (Collections) οι οποίες μεταβάλλουν δυναμικά το μέγεθός τους (π.χ., ArrayList, Vector)
  - Χρησιμοποιώντας δείκτες αναφοράς (pointers) σε αντικείμενα

Στοίβα με Στατική Δέσμευση Μνήμης (πίνακας)

- Ο "πιο απλός τρόπος" είναι η χρήση μονοδιάστατου πίνακα. Χρειάζεται να γνωρίζουμε από την αρχή το μήκος της λίστας.
- Για την παράσταση στοίβας με στοιχεία α_1, α_2, ..., α_n χρειαζόμαστε ένα πίνακα A στον οποίο θα αποθηκεύσουμε τα στοιχεία της στοίβας, A[i-1] = α_i. Πρέπει να γνωρίζουμε ανά πάσα στιγμή που βρίσκεται η κορυφή της στοίβας.

- Έτσι χρησιμοποιούμε μια *εγγραφή με δύο πεδία*
  1. ένα πίνακα `A[0..n-1]`, και
  2. μια μεταβλητή `Length` τύπου *ακέραιος* (που συγκρατεί τη θέση κορυφής).



- **Πρόβλημα:** δεν μπορούμε να προσθέσουμε περισσότερα από ***n*** στοιχεία ➔ Δεν θα χρησιμοποιήσουμε αυτή την προσέγγιση

Διαπροσωπεία (Interface) Στοίβας

```
public interface IStack<E> {

    public void makeEmpty();

    public boolean isEmpty();

    public int size();

    public E top();

    public void push(E obj);

    public void pop();

}
```

Στοίβα με Δυναμική Δέσμευση Μνήμης (ArrayList)

- Με χρήση ArrayList και μίας μεταβλητής top

```
import java.util.ArrayList;
public class Stack<E> implements IStack<E> {
  private ArrayList<E> stack;
  private int top;

  public Stack() {
    stack = new ArrayList<E>();
    top = -1;
  }

  public void makeEmpty(){
    top=-1;
    stack.clear();
  }

  public boolean isEmpty(){
    return (top == -1);
  }

  public int size() {
    return top+1;
  }
```

```
  public E top() {
    return isEmpty()?
           null : stack.get(top);
  }

  public void push(E obj) {
    stack.add(obj);
    top++;
  }

  public E pop() {
    if (!isEmpty()) {
      E temp = stack.get(top);
      stack.remove(top);
      top--;
      return temp;
    }
    return null; // stack is empty
  }
```

π.χ., `Stack<Integer> stack = new Stack<Integer>();`

Στοίβα με Δυναμική Δέσμευση Μνήμης 2 (ArrayList)

- Με χρήση μόνο ArrayList και της μεθόδου size()

```
import java.util.ArrayList;
public class Stack<E>
             implements IStack<E> {
  private ArrayList<E> stack;

  public Stack() {
    stack = new ArrayList<E>();
  }

  public void makeEmpty(){
    stack.clear();
  }

  public boolean isEmpty(){
    return stack.size()==0;
  }

  public int size() {
    return stack.size();
  }
```

```
public E top() {
  return isEmpty()?
        null : stack.get(stack.size()-1);
}

public void push(E obj) {
  stack.add(obj);
}

public E pop() {
  if (!isEmpty()) {
    E temp = stack.get(stack.size()-1);
    stack.remove(stack.size()-1);
    return temp;
    }
    return null; // stack is empty
}
```

Διαπροσωπεία (Interface) Ουράς

```
public interface IQueue<E> {

      public void makeEmpty();

      public boolean isEmpty();

      public int size();

      public E top();

      public void enQueue(E obj);

      public void deQueue();

}
```

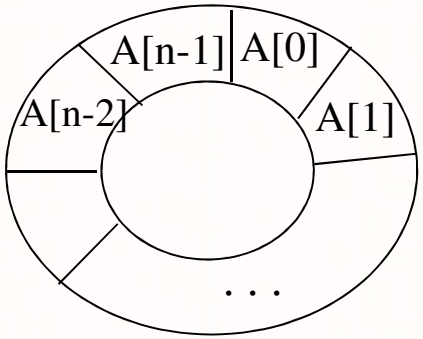
Ουρά με Δυναμική Δέσμευση Μνήμης (ArrayList)

- Με χρήση μόνο ArrayList και της μεθόδου size()

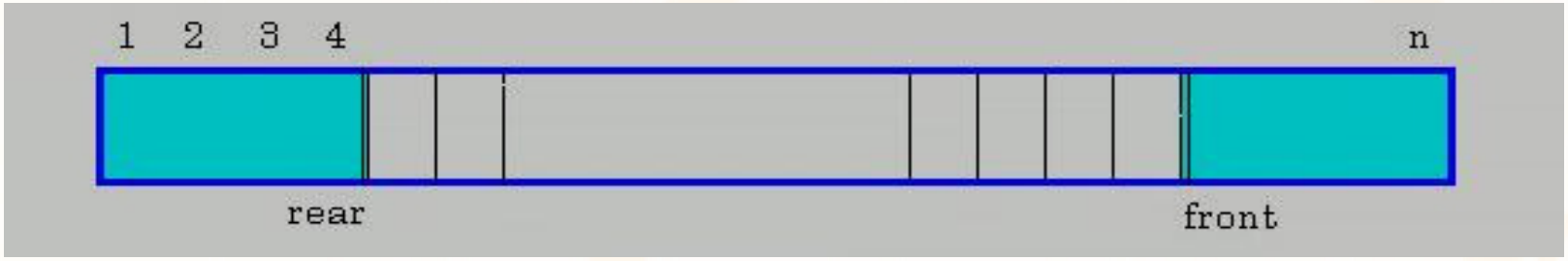
```
import java.util.ArrayList;

public class Queue<E> implements IQueue<E>
{
  private ArrayList<E> queue;

  public Queue(){
    queue = new ArrayList<E>();
  }

  public void makeEmpty() {
    queue.clear();
  }

  public boolean isEmpty() {
    return queue.size()==0;
  }

  public int size() {
    return queue.size();
  }

  public E top() {
    if(!isEmpty()) {
      return queue.get(0);
    }
    return null;
  }

  public void enQueue(E obj) {
    queue.add(obj);
  }

  public void deQueue() {
    if(!isEmpty()) {
      queue.remove(0);
    }
  }
}
```

ΑΤΔ Λίστα 3 : Κυκλική Ουρά

- Για λόγους χώρου μνήμης μπορούμε να πραγματοποιήσουμε την ουρά με μια ***κυκλική*** διάταξη των λέξεων της μνήμης. Δηλαδή θα θεωρούμε ότι η περιοχή μνήμης δεν αρχίζει με τη λέξη **A[0]** και τελειώνει με τη λέξη **A[n-1],** αλλά ότι μετά την **A[n-1]** ακολουθεί η **A[0].**

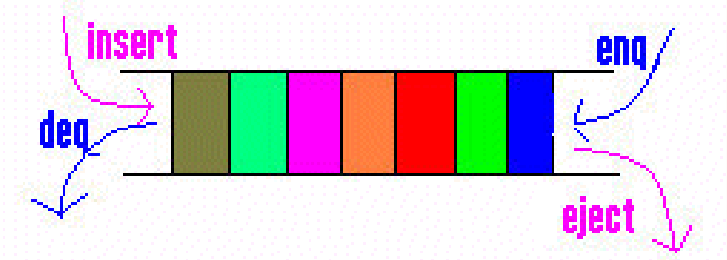


- Έτσι μετά από μια ακολουθία εισαγωγών και εξαγωγών η ουρά μας πιθανόν να έχει την πιο κάτω μορφή όπου θεωρούμε ότι **η αρχή της ουράς βρίσκεται στη θέση k** και το τέλος της ουράς στη θέση 4.



ΑΤΔ Λίστα 4 : Ουρές με Δύο Άκρα

- Ο ΑΤΔ ‘ουρά με δύο άκρα είναι παρόμοιος με το ΑΤΔ ουρά, με τη διαφορά ότι έχει δύο άκρα και επιτρέπει εισαγωγές και εξαγωγές και στα δύο.

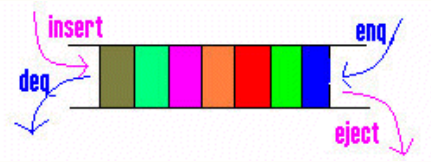


- Μια ουρά δύο άκρων ορίζεται ως μια λίστα συνοδευόμενη από τις πιο κάτω πράξεις: makeEmpty(), isEmpty()

insert(x)	εισήγαγε το στοιχείο x στο μπροστινό άκρο της Q
eject()	διέγραψε τον κόμβο στο πίσω άκρο της Q
enQueue (x)	εισήγαγε τον στοιχείο x στο πίσω μέρος της Q.
deQueue()	διέγραψε τον κόμβο στο μπροστινό άκρο της Q
front(Q)	δώσε τον κόμβο στο μπροστινό άκρο της Q
rear(Q)	δώσε τον κόμβο στο πίσω άκρο της Q

Ουρές με Δύο Άκρα με Στατική Δέσμευση Μνήμης

- Για την παράσταση μιας ουράς με στοιχεία $\alpha_0, \alpha_1, \ldots, \alpha_{n-1}$ χρειαζόμαστε
  - ένα πίνακα *A* στον οποίο θα αποθηκεύσουμε τα στοιχεία της ουράς,
  - Θα αναφερόμαστε στο στοιχείο *A*[*i*-1] σαν α_i ,
  - δύο δείκτες που προσδιορίζουν τα δύο προσιτά άκρα της ουράς.



- Έτσι χρησιμοποιούμε μια εγγραφή με τρία πεδία
  1. ένα πίνακα `A[n]`,
  2. μια μεταβλητή `front`, τύπου *ακέραιος*, που συγκρατεί τη θέση που βρίσκεται αμέσως πριν τη θέση εξόδου, και
  3. μια μεταβλητή `rear`, τύπου *ακέραιος*, που συγκρατεί τη θέση του τελευταίου στοιχείου της ουράς.

  Μέγεθος ουράς: rear-front+1 (πχ rear=2, front=0 => length=3)