Δομές Δεδομένων (Data Structures)

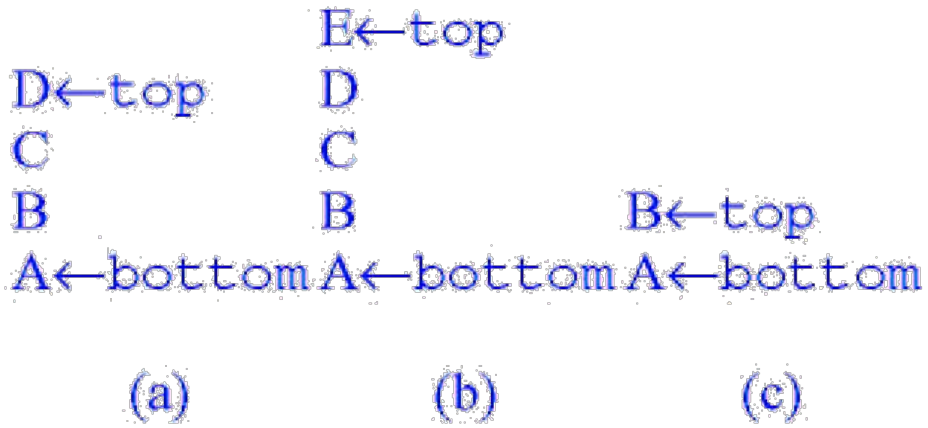
Στοίβες – Ουρές

- Στοίβες: Βασικές Έννοιες.
- Ουρές: Βασικές Έννοιες.
- Βασικές Λειτουργίες.
- Παραδείγματα.

Στοίβες

- **Δομή τύπου LIFO: Last In - First Out**
  (τελευταία εισαγωγή – πρώτη εξαγωγή)

- **Γραμμική λίστα όπου: Εισαγωγή και διαγραφή μόνο στο ένα άκρο της λίστας (στην αρχή)**



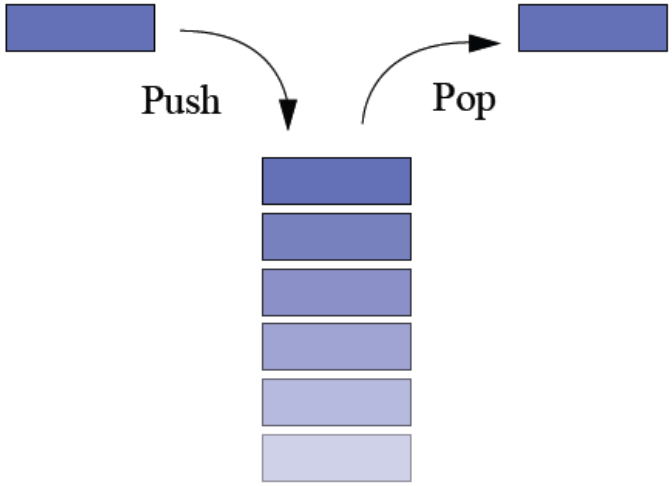
Στοίβες - Εφαρμογές

- **Άμεσες εφαρμογές:**
  - **Web browser: Το ιστορικό των σελίδων που επισκεπτόμαστε.**
  - **Επεξεργαστή κειμένου: Η λίστα αναίρεσης (undo).**
  - **Java Virtual Machine: Η σειρά των κλήσεων των διαφορετικών μεθόδων.**
- **Έμμεσες εφαρμογές:**
  - **Βοηθητικές σε πιο γενικές Δομές δεδομένων.**

Στοίβες

- **Λειτουργίες:**
  - Create: Δημιουργία στοίβας.
  - IsEmpty: (boolean) Επιστρέφει true αν η στοίβα είναι άδεια, false αλλιώς.
  - Top: Επιστροφή κορυφαίου στοιχείου.
  - Push (or Add): Ένθεση στοιχείου.
  - Pop (or Delete): Διαγραφή κορυφαίου στοιχείου.

- **Προσοχή:** Η πρόσβαση είναι δυνατή μόνο στο κορυφαίο στοιχείο της ακολουθίας.



Υλοποίηση Στοίβας: Με Πίνακα

- Απλός τρόπος υλοποίησης
- Ορίζουμε μεταβλητή που ακολουθεί το πάνω στοιχείο της στοίβας. Προσθέτουμε στοιχεία από αριστερά προς τα δεξιά

Algorithm *size()*
return *t + 1*
Algorithm *pop()*
if *isEmpty() then*
throw *EmptyStackException*
else
***t* ← *t* − 1**
return *S[t + 1]*

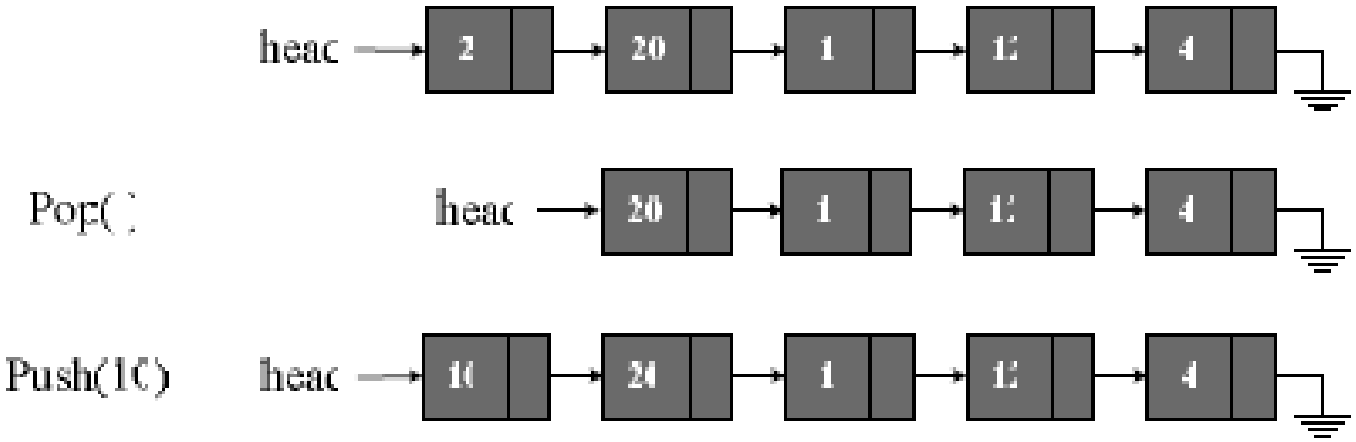
```
int S[nMax], top = -1;
int isEmpty() {
   return(top == -1); }

int isFull() {
   return(top >= nMax-1); }
```

```
int Push(int x) {
   if (isFull()) return(-1);
   S[++top] = x; return(0);    }
int Pop(int *x) {
   if (isEmpty()) return(-1);
   *x = S[top--]; return(0);   }
```

Υλοποίηση Στοίβας: Με Λίστα

- Εισαγωγή και Διαγραφή στην αρχή.
- Η θέση του κορυφαίου στοιχείου σε μεταβλητή **head**.
- Βασικές Λειτουργίες: Ένθεση (**Push**) / Διαγραφή (**Pop**)



Υλοποίηση Στοίβας: Με Λίστα

- **Πολυπλοκότητα σε χρόνο O(1).**
- **Πρέπει πάντα να ελέγχουμε αν top ≤ maxCapacity.**
- **Απαιτήσεις χώρου: για δυναμική υλοποίηση Θ(n).**
- **Λιγότερη εύκολη υλοποίηση από ότι με πίνακα.**

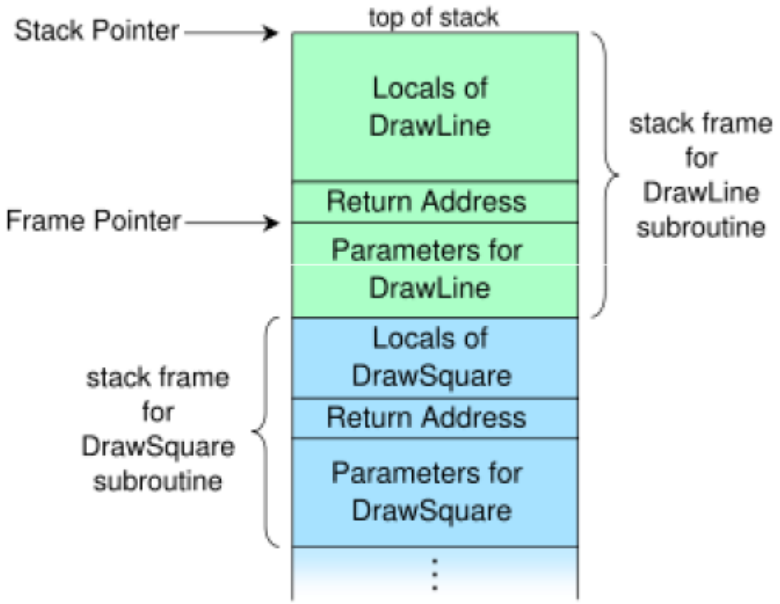
- **Υλοποίηση: Με πίνακα ή με λίστα;**
  - ✓ **Με λίστα εκτός αν γνωρίζουμε εκ των προτέρων τον μέγιστο αριθμό στοιχείων και θέλουμε μια εύκολη και γρήγορη λύση.**

Στοίβες – Παράδειγμα

❑ **Στοίβα Κλήσεων Συναρτήσεων (call stack ή execution stack ή run-time stack)**
Διαχειρίζεται τις κλήσεις των δομημένων σε συναρτήσεις (ρουτίνες) προγραμμάτων.

➢ Στην στοίβα κλήσεων καταγράφεται πληροφορία σχετική με τις ενεργές ρουτίνες ενός προγράμματος. Ενεργές θεωρούνται οι ρουτίνες που έχουν κληθεί, αλλά δεν έχουν ακόμη επιστρέψει.

➢ Η κλήση μιας ρουτίνας προκαλεί την ώθηση στη στοίβα κλήσεων μιας δομής με πληροφορία (activation record) που μπορεί να περιέχει:
– Την διεύθυνση επιστροφής (στην καλούσα ρουτίνα).
– Την τιμή άλλων βασικών καταχωρητών σε σχέση με την καλούσα ρουτίνα.
– Χώρο για την αποθήκευση των τοπικών μεταβλητών της ρουτίνας.
– Χώρο για την αποθήκευση των παραμέτρων κλήσης της ρουτίνας.
– Χώρο για τον υπολογισμό εκφράσεων.

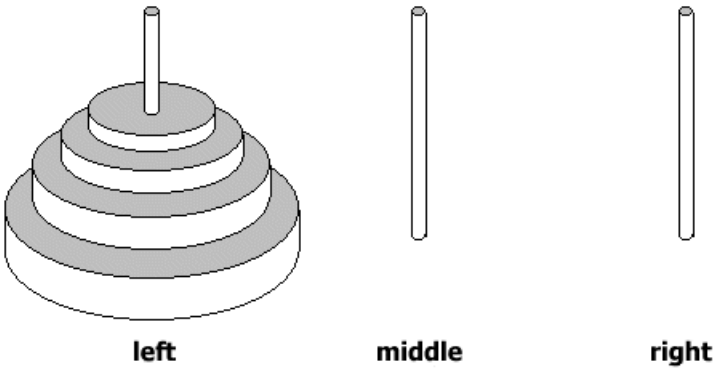


Στοίβες – Πύργοι του Hanoi

- **Ινδικό Τελετουργικό (ή μύθος;) [E. Lucas 1883].**

Σύμφωνα με κάποιο μύθο σε έναν ναό της Άπω Ανατολής, οι ιερείς προσπαθούσαν να μεταφέρουν μια στοίβα χρυσών δίσκων από ένα στύλο σε ένα άλλο. Ο αρχικός στύλος έχει 64 δίσκους τοποθετημένους σε φθίνουσα σειρά μεγέθους από κάτω προς τα πάνω. Η μεταφορά των δίσκων από τον πρώτο στύλο στον τρίτο πρέπει να γίνει με τους εξής κανόνες:

1) Κάθε φορά μεταφέρεται μόνο ένας δίσκος.
2) Σε καμιά περίπτωση δεν μπορεί ένας μεγαλύτερος δίσκος να τοποθετηθεί πάνω σε ένα μικρότερο .
3) Ο δεύτερος στύλος μπορεί να χρησιμοποιηθεί για προσωρινή τοποθέτηση.



Στοίβες – Πύργοι του Hanoi

- **Αναδρομική Επίλυση:**

Έστω *n* δίσκοι στον πύργο 1 → μετακινούμε τους n-1 στον πύργο 3 και μετά τον μεγαλύτερο στον 2 → μετακινούμε τους n-2 στον πύργο 1 και μετά τον μεγαλύτερο στον 2 → …

Πιθανοί συνδυασμοί (κινήσεις που απαιτούνται για τη μεταφορά):

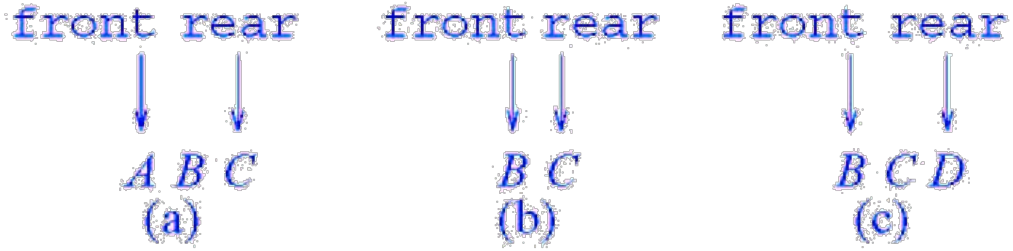
3 δίσκων : 7 κινήσεις δηλ 2^3- 1 κινήσεις.
4 δίσκων: 15 κινήσεις δηλ 2^4- 1 κινήσεις.
…
64 δίσκων: 2^{64}-1 κινήσεις .

Γενικά για *n* δίσκους: 2^n -1 κινήσεις.

Ουρές

- Δομή **τύπου FIFO:** **First In - First Out**
  (πρώτη εισαγωγή – πρώτη εξαγωγή).

- **Γραμμική λίστα όπου: Εισαγωγή και διαγραφή από τα δύο διαφορετικά άκρα της λίστας.**
- **Αρχή (front) και Τέλος (rear).**



Ουρές - Εφαρμογές

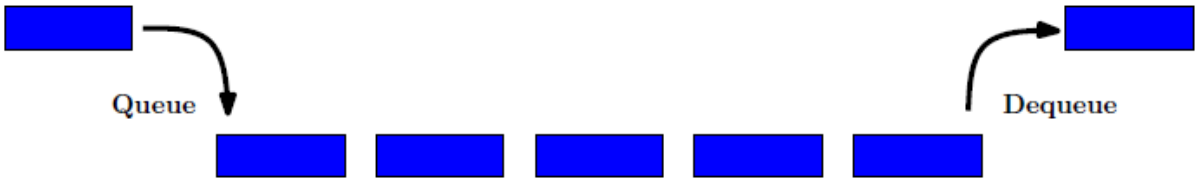
- **Λειτουργεί όπως η ουρά σε μια δημόσια υπηρεσία (π.χ. τράπεζα)**

- **Άμεσες εφαρμογές:**
  - **Προσπέλαση κοινόχρηστων πόρων.**
  - **Χρήση επεξεργαστών σε πολύ-επεξεργαστικά περιβάλλοντα.**
  - **Routing buffers.**

- **Έμμεσες εφαρμογές:**
  - **Βοηθητικές σε πιο γενικές Δομές δεδομένων.**

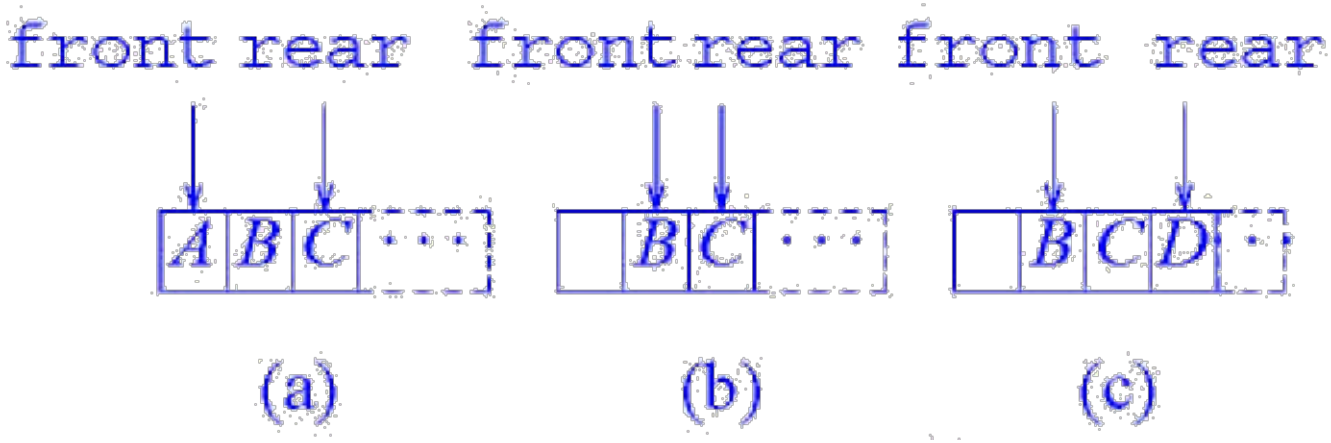
Ουρές

- **Λειτουργίες:**
  - Create: Δημιουργία στοίβας.
  - Add (or Enqueue or Queue)(): Εισαγωγή στοιχείου στο τέλος της ουράς.
  - Delete (or. Dequeue)(): Διαγραφή στοιχείου στην κορυφή της ουράς.
  - First(): Επιστροφή στοιχείου τέλους της ουράς.
  - Last(): Επιστροφή στοιχείου τέλους της ουράς.



Υλοποίηση Ουράς: Με Πίνακα

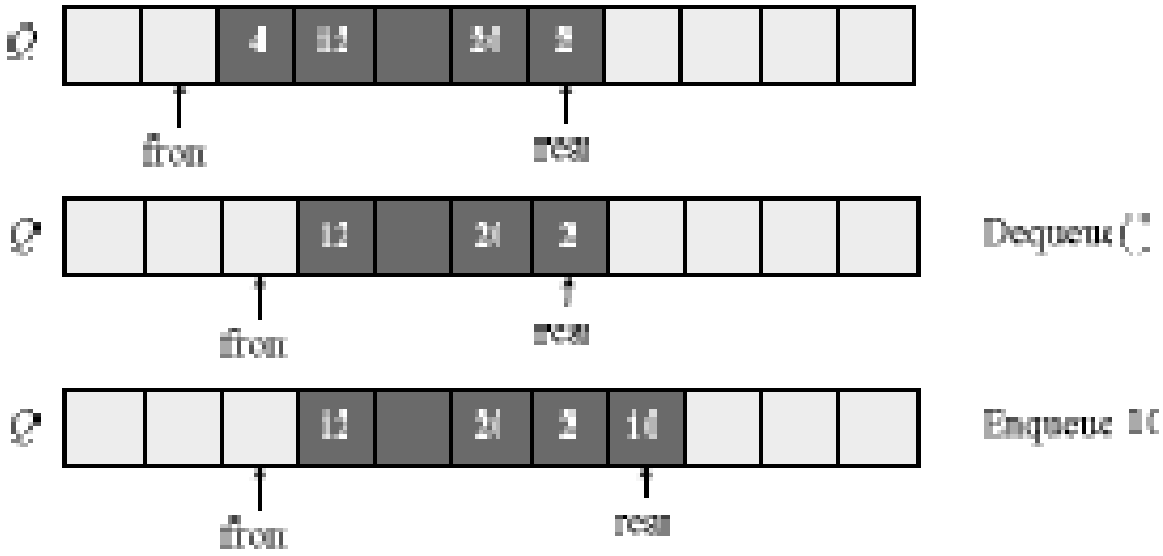
- Δημιουργούμε πίνακα με μεταβλητές για το πρώτο (π.χ. front) και το τελευταίο (π.χ. rear) στοιχείο.




```
location(i)=front+i-1
```

Υλοποίηση Ουράς: Με Πίνακα

- **Βασικές Λειτουργίες: Ένθεση (Enqueue) / Διαγραφή (Dequeue)**

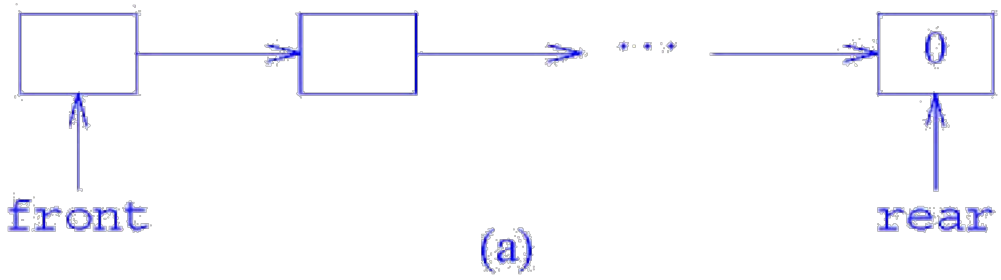


Υλοποίηση Ουράς: Με Λίστα

- Πρώτο στοιχείο σε front τελευταίο σε rear.
- Εισαγωγή από το τέλος (τελευταίο στοιχείο).
- Διαγραφή από αρχή (πρώτο στοιχείο)

Υλοποίηση Ουράς: Με Λίστα

- **Παράδειγμα:**

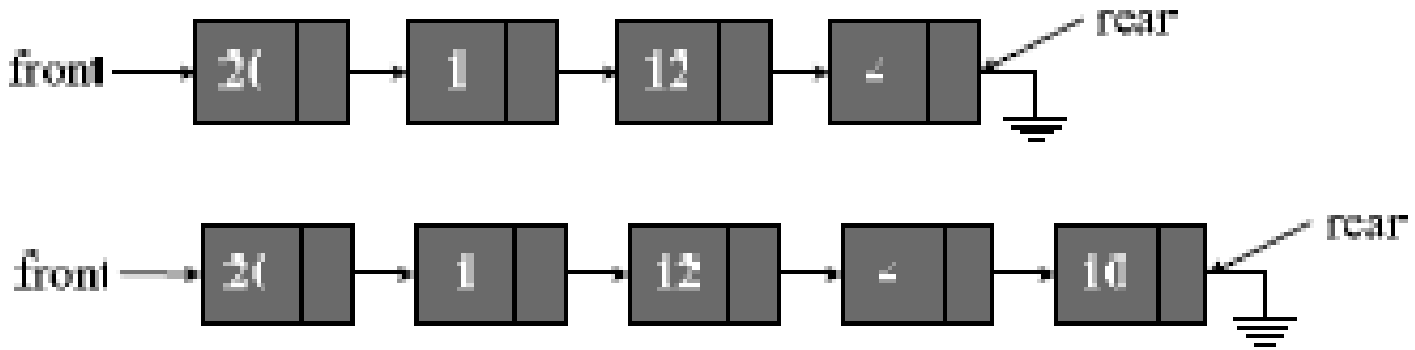



```
class LinkedQueue {
// FIFO objects
   public:
      LinkedQueue()
         {front = rear = 0;}; // constructor
      ~LinkedQueue(); // destructor
      int IsEmpty() {return ((front) ? 0 : 1);}
      int IsFull();
      int First(type& x); // return first element of queue
      int Last(type& x);  // return last element of queue
      int operator +(type x); // add x to queue
      int operator -(type& x); // delete x from queue
         // First,+,- return 0 on failure, 1 on success
   private:
      Node<type> *front, *rear;
}
```

Ουρές - Ένθεση

- Εισαγωγή - Ένθεση κόμβου.

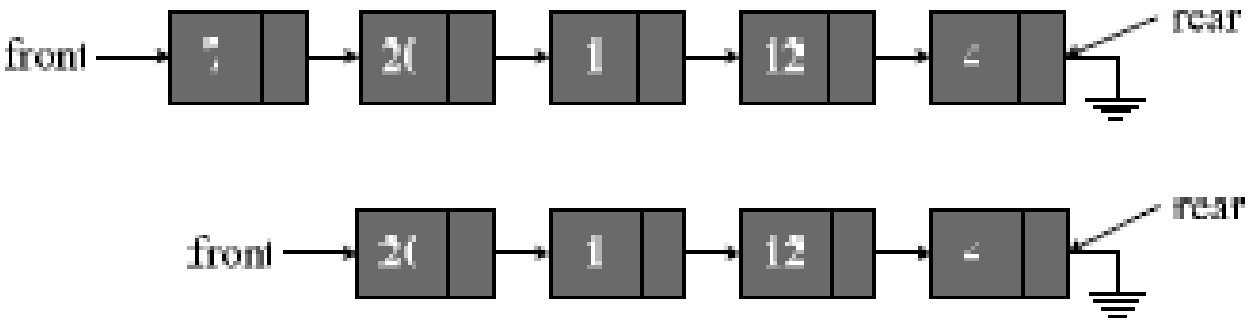
```
Enqueue(Node *p) {
    if (front) rear->next = p;
    else front = p;
    rear = p;
    p->next = NULL;   }
```



Ουρές - Διαγραφή

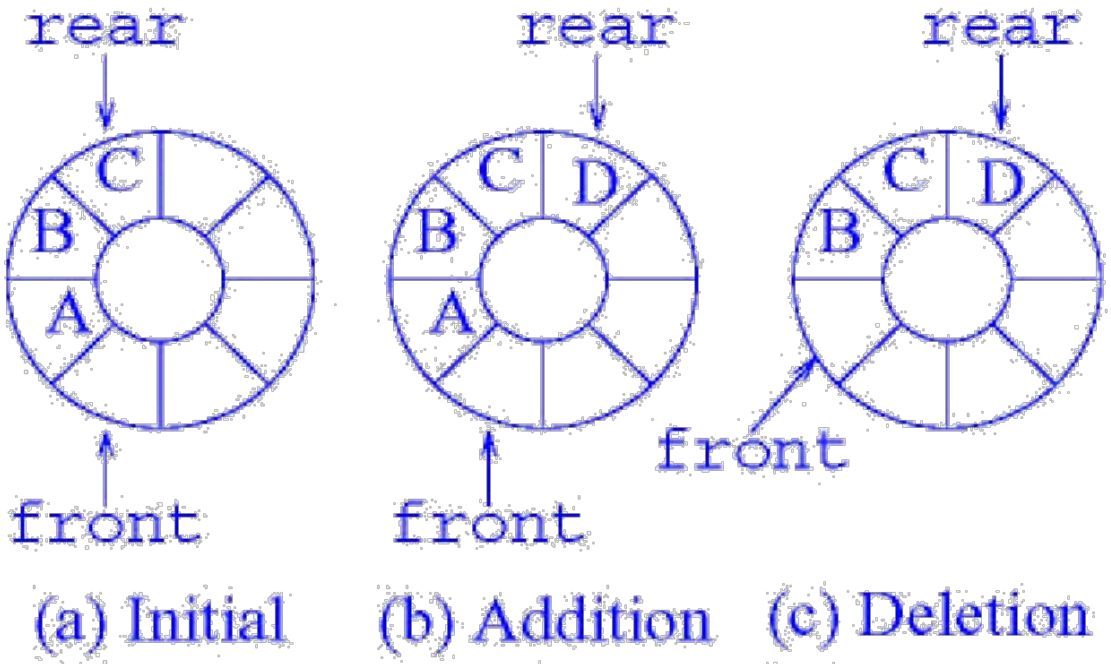
- Διαγραφή κόμβου.

```
Dequeue ( ) {
    q = front; x = q->elem;
    front = front->next;
    free(q); return(x); }
```



Κυκλικές Ουρές

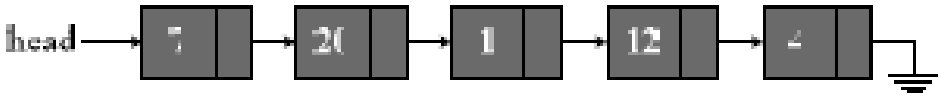
- **Κυκλικές ουρές:**




```
location(i)=(front+i)%MaxSize
```

Ουρές Προτεραιότητας (Priority Queues)

- Κάθε κόμβος της ουράς έχει προτεραιότητα *P*
- Η σειρά διαγραφής ή/και ένθεσης καθορίζεται από την προτεραιότητα (μεγαλύτερη ή μικρότερη) (π.χ. ουρά Ελληνικής δημόσιας υπηρεσίας;)
- Εφαρμογές: Βασικό συστατικό πολλών ΔΔ και αλγορίθμων
  - Heapsort.
  - Γενικά ταξινόμηση με επιλογή.
  - Αλγόριθμος του Dijkstra



Βιβλιογραφία

- *Δομές Δεδομένων, Αλγόριθμοι και Εφαρμογές στη C++*, S. Sahni, Εκδόσεις Τζιόλα, 2004.
- Δομές Δεδομένων, Έννοιες, Τεχνικές και Αλγόριθμοι, Γ.Φ. Γεωργακόπουλος., Πανεπιστημιακές Εκδόσεις Κρήτης, Ηράκλειο 2002.
- *Δομές Δεδομένων*, Π. Μποζάνης, Εκδόσεις Τζιόλα 2006.
- **Πύργοι του Hanoi:** *http://en.wikipedia.org/wiki/Tower_of_Hanoi*