Βασικές Δομές Δεδομένων

Στην ενότητα αυτή θα μελετηθούν τα εξής επιμέρους θέματα:

Αφηρημένοι Τύποι Δεδομένων

Οι ΑΤΔ Στοίβα και Ουρά

Υλοποίηση των ΑΤΔ Στοίβα και Ουρά με Διαδοχική και Δυναμική Χορήγηση Μνήμης

Αφηρημένοι Τύποι Δεδομένων

- **Τύπος Δεδομένων:** μια **συλλογή αντικειμένων** με μια κοινή σχέση μαζί με ένα σύνολο **πράξεων** για τη δημιουργία και επεξεργασία των αντικειμένων.

- **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

Παραδείγματα:

1. Ο τύπος *int* μαζί με τις πράξεις +, *, /, =, είναι ένας τύπος δεδομένων. Για να τον χρησιμοποιήσουμε χρειάζεται να γνωρίζουμε το σύνολο των σχετικών πράξεων. Ο τρόπος αναπαράστασής του στον υπολογιστή δεν μας ενδιαφέρει.
2. Ουρά Προτεραιότητας: ένα *σύνολο στοιχείων τύπου key* (π.χ. key = (int,int)) πάνω στο οποίο έχει ορισθεί μια γραμμική διάταξη, 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

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

- 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 $\langle\rangle$.

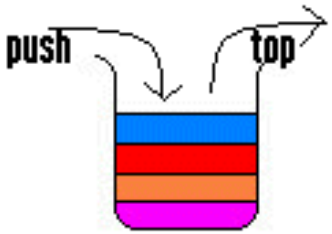
- Συνοδεύοντας λίστες με ένα σύνολο πράξεων μπορούμε να ορίσουμε αφηρημένους τύπους δεδομένων. Χρήσιμες πράξεις περιλαμβάνουν τις πιο κάτω:
  - Δημιουργία λίστας
  - Εισαγωγή νέου κόμβου στη λίστα
  - Εξαγωγή κόμβου από τη λίστα
  - Εύρεση κόμβου με ορισμένη ιδιότητα
  - Διάταξη της λίστας σύμφωνα με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

Λίστες

- 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.
  - Η ***ουρά*** (queue) όπου γίνονται εισαγωγές στο ένα άκρο και εξαγωγές από το άλλο.
- Υπάρχουν και άλλοι ΑΤΔ-λίστας μικρότερης πρακτικής σημασίας, όπως
  - ουρά με δύο άκρα,
  - πολλαπλή στοίβα, κλπ

Στοίβες

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



MakeEmptyStack()	δημιούργησε την κενή στοίβα ⟨⟩.
IsEmptyStack(S)	επέστρεψε τη λογική τιμή που εκφράζει το αν η S είναι κενή.
Push(x,S)	εισήγαγε τον κόμβο x στην κορυφή της στοίβας S
Pop(S)	διέγραψε τον κόμβο κορυφής της S.
Top(S)	δώσε τον κόμβο κορυφής της S.

Στοίβες

Οι πράξεις αυτές προδιαγράφονται από τους εξής κανόνες:

IsEmptyStack (MakeEmptyStack) = true

IsEmptyStack(Push(x,S)) = false

Pop(MakeEmptyStack()) = error

Pop(Push(x,S)) = S

Top(MakeEmptyStack()) = error

Top(Push(x,S)) = x

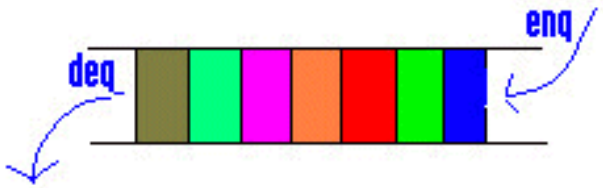
πολιτική LIFO

last in, first out

Ουρές

- Από μια ουρά Q πρώτο διαγράφεται το στοιχείο που εισήχθηκε πρώτο στην ουρά. Νέες εισαγωγές γίνονται στο πίσω άκρο.
- Ορίζουμε μια ουρά ως μια λίστα συνοδευόμενη από τις πιο κάτω πράξεις:

MakeEmptyQueue()	δημιούργησε την κενή ουρά <>.
IsEmptyQueue(Q)	επέστρεψε τη λογική τιμή που εκφράζει το αν η Q είναι κενή
EnQueue (x,Q)	εισήγαγε τον κόμβο x στην ουρά Q
DeQueue(Q)	διέγραψε τον κόμβο εξόδου της Q
Top(Q)	δώσε τον κόμβο εξόδου της Q.



Ουρές

- Οι πράξεις αυτές προδιαγράφονται από τους εξής κανόνες

IsEmptyQueue(MakeEmptyQueue()) = true

IsEmptyQueue (EnQueue(x,Q)) = false

DeQueue(MakeEmptyQueue()) = error

DeQueue (EnQueue(x,Q)) =
if IsEmptyQueue(Q) then Q
else EnQueue(x, DeQueue(Q))

Top (MakeEmptyQueue ()) = error

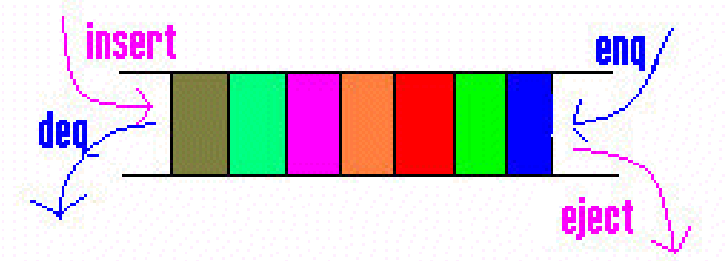
Top(EnQueue (x,Q)) = if IsEmptyQueue(Q) then x
else Top(Q)

πολιτική FIFO

first in, first out

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: MakeEmptyDeQue(), IsEmptyDeQue(Q)

Insert(x, Q)	εισήγαγε το στοιχείο x στο μπροστινό άκρο της Q
Eject(Q)	διέγραψε τον κόμβο στο πίσω άκρο της Q
EnQueue (x,Q)	εισήγαγε τον στοιχείο x στο πίσω μέρος της Q
DeQueue(Q)	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

Αναπαράσταση Βασικών ΑΤΔ

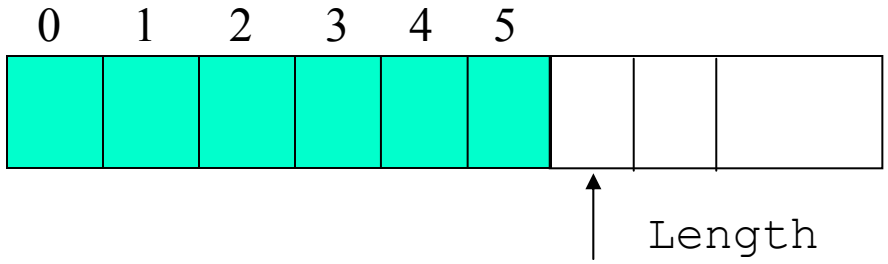
- Οι πιο πάνω ΑΤΔ μπορούν να υλοποιηθούν με διάφορες δομές δεδομένων χρησιμοποιώντας είτε στατική είτε δυναμική χορήγηση μνήμης.

1. Αναπαράσταση Στοίβας με Διαδοχική Χορήγηση Μνήμης

- 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 $\alpha_1, \alpha_2, ..., \alpha_n$ χρειαζόμαστε ένα πίνακα Α στον οποίο θα αποθηκεύσουμε τα στοιχεία της στοίβας, $A[i-1] = \alpha_i$. 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).



Αναπαράσταση ΒασικώνΑΤΔ

Ο τύπος δεδομένων που χρειάζεται για τη δομή είναι:

```
typedef struct Stack{
   type list[size];
   int Length;
} stack
```

Υλοποίηση πράξεων:

```
MakeEmpty(stack *S){
   (*S).Length = 0;
}
```

```
Push(type x, stack *S){
  if (*S).Length <size
    (*S).list[(*S).Length]= x;
    (*S).Length++; }
```

```
int IsEmpty(stack *S){
   return ((*S).Length == 0);
}
```

```
Pop(stack *S){
   if !IsEmpty(S)
        (*S).Length--;
}
```

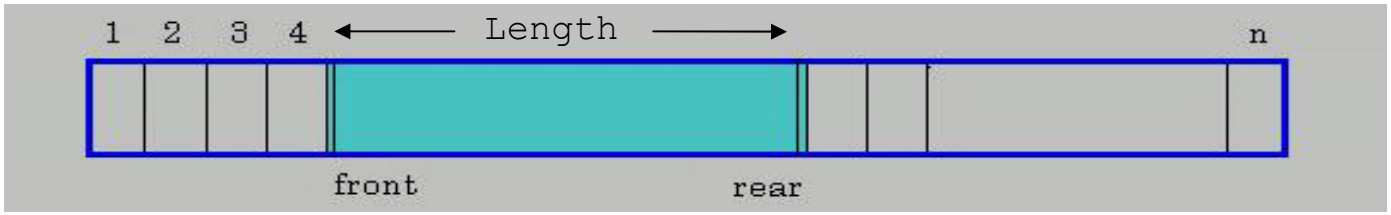
```
type Top(stack *S){
   if !IsEmpty(S)
    return
     (*S).list[(*S).Length-1];
}
```

Αναπαράσταση Βασικών ΑΤΔ

2. Παράσταση Ουράς σε Διαδοχική Μνήμη

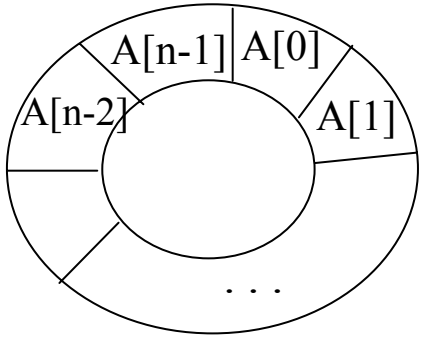
- Για την παράσταση μιας ουράς με στοιχεία $\alpha_0, \alpha_1, ..., \alpha_{n-1}$ χρειαζόμαστε ένα πίνακα A στον οποίο θα αποθηκεύσουμε τα στοιχεία της ουράς, $A[i\text{-}1] = \alpha_i$, και δύο δείκτες που προσδιορίζουν τα δύο προσιτά άκρα της ουράς.

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

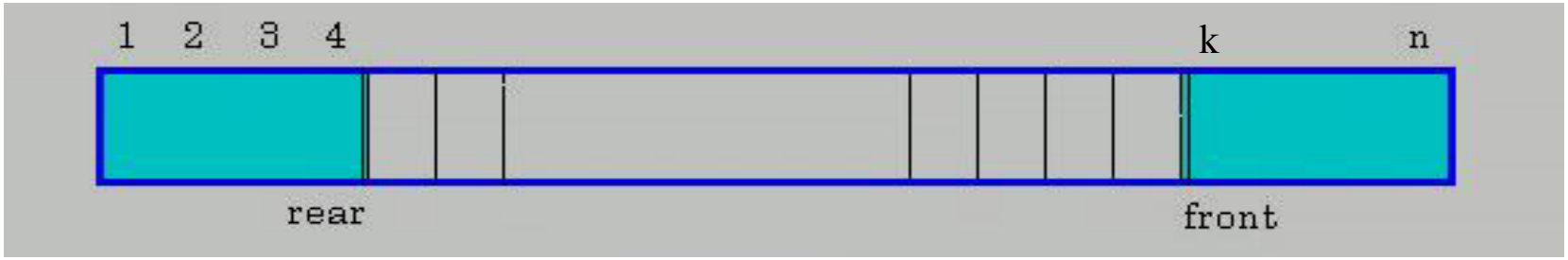


Αναπαράσταση Βασικών ΑΤΔ

- Για λόγους χώρου μνήμης θα 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.



Αναπαράσταση Βασικών ΑΤΔ

Ο τύπος δεδομένων που χρειάζεται για τη δομή είναι:

```
typedef struct Queue{
   type list[size];
   int front;
   int Length;
} queue
```

Υλοποίηση πράξεων:

```
MakeEmpty(queue *Q){
   (*Q).Length = (*Q).front = 0;
}
int IsEmpty(queue *Q){
   return ((*Q).Length == 0);
}
```

Αναπαράσταση Βασικών ΑΤΔ

```
Enqueue(type x, queue *Q){
   if ((*Q).Length < size)
        (*Q).list[(*Q).front ⊕ (*Q).Length] = x;
        (*Q).Length++;
}

Dequeue(queue *Q){
   if (!IsEmpty (Q))
        (*Q).Length--;
        (*Q).front = (*Q).front ⊕ 1;
}

type Top(queue *Q){
   if (!IsEmpty(Q)
        return (*Q).list[(*Q).front];
}
```

Γράφουμε

$a \oplus b$ για $(a + b) \bmod size$

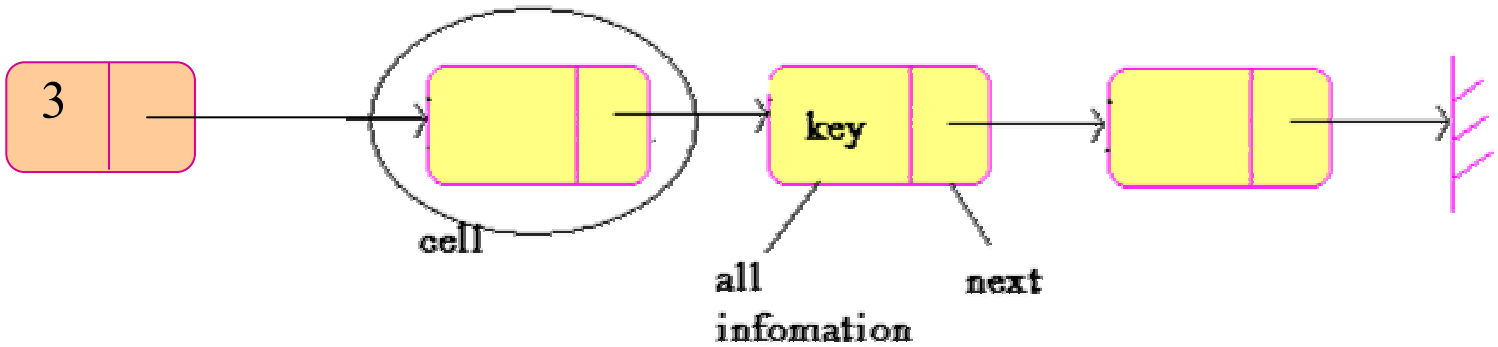
και

$a \ominus b$ για $(a - b) \bmod size$

Αναπαράσταση Βασικών ΑΤΔ

3. Παράσταση Στοίβας σε Συνδετική Μνήμη

- Για την παράσταση μιας στοίβας με στοιχεία $\alpha_1, \alpha_2, ..., \alpha_n$ χρησιμοποιούμε μια συνδεδεμένη λίστα από κόμβους.
- Κάθε κόμβος αποτελείται από ένα στοιχείο (στοιχεία της στοίβας) και από ένα δείκτη (προς τον επόμενο κόμβο της στοίβας). Η κορυφή της στοίβας είναι ο πρώτος κόμβος της λίστας,
- Χρησιμοποιούμε μια μεταβλητή για να φυλάγουμε στοιχεία σχετικά με τη στοίβα π.χ. μέγεθος και δείκτη προς την κορυφή της στοίβας.



Αναπαράσταση Βασικών ΑΤΔ

- Συνεπώς απαιτούνται οι παρακάτω δηλώσεις κόμβων:

```
typedef struct node {
    type          data;
    struct node *next;
} NODE;
```

```
typedef struct stack {
    NODE  *top;
    int   size;
} STACK;
```

- Υπολείπεται η υλοποίηση των βασικών πράξεων στοίβας για αναπαράσταση με συνδεδεμένες λίστες.

- Βασικό ζητούμενο: όλες οι πράξεις να εκτελούνται σε χρόνο O(1).

Αναπαράσταση Βασικών ΑΤΔ

- Υλοποίηση πράξεων:

```
int IsEmptyStack(STACK *S)
    return ((*S).size == 0)

Pop(STACK *S)
    if ((*S).size) > 0)
          p = (*S).top;
          (*S).top = (*p).next;
          free(p);
          (*S).size--;

Push(STACK *S, type x)
    p = (NODE *)malloc(sizeof(NODE));
    (*p).data = x;
    (*p).next =(*S).top;
    (*S).top = p;
    (*S).size ++;
```

Αναπαράσταση ΒασικώνΑΤΔ

4. Παράσταση Ουράς σε Συνδετική Μνήμη

- Όπως και στην περίπτωση της στοίβας, κάθε κόμβος αποτελείται από ένα στοιχείο και ένα δείκτη (που δείχνει προς τον επόμενο κόμβο).
- Σε αυτή την περίπτωση χρησιμοποιούμε δύο δείκτες για υλοποίηση μιας ουράς, ο κάθε ένας από τους οποίους δείχνει στο κάθε ένα από τα προσιτά άκρα της ουράς.

```
struct queue {
    int size
    node *front;
    node *back;
}
```