



Διάλεξη 9: 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 με Στατική Δέσμευση Μνήμης

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 Αφηρημένος Τύπος Δεδομένων (ΑΤΔ) 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 ,*
- *έλεγε αν η ουρά q είναι άδεια,*
- *βάλει το στοιχείο k στην ουρά q ,*
- *αφαίρεσε και επέστρεψε 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)

$$L = \alpha_1, \alpha_2, \dots, \alpha_n$$

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

Λίστες (συν.)

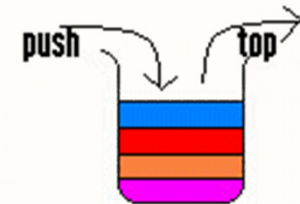
- Συνοδεύοντας λίστες με ένα σύνολο πράξεων μπορούμε να ορίσουμε **αφηρημένους τύπους δεδομένων**. Χρήσιμες πράξεις περιλαμβάνουν τις πιο κάτω:
 - Δημιουργία λίστας
 - Εισαγωγή νέου κόμβου στη λίστα
 - Εξαγωγή κόμβου από τη λίστα
 - Εύρεση κόμβου με ορισμένη ιδιότητα
 - Διάταξη της λίστας σύμφωνα με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



MakeEmptyStack() δημιουργήσε την κενή στοίβα $\langle \rangle$.

IsEmptyStack(S) επέστρεψε τη λογική τιμή που εκφράζει το αν η S είναι κενή.

Push(x,S) εισήγαγε τον κόμβο x στη στοίβα S .

Pop(S) διέγραψε τον κόμβο κορυφής της S .

Top(S) δώσε τον κόμβο κορυφής της S .

ΑΤΔ Στοίβα (stack) (συν.)

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

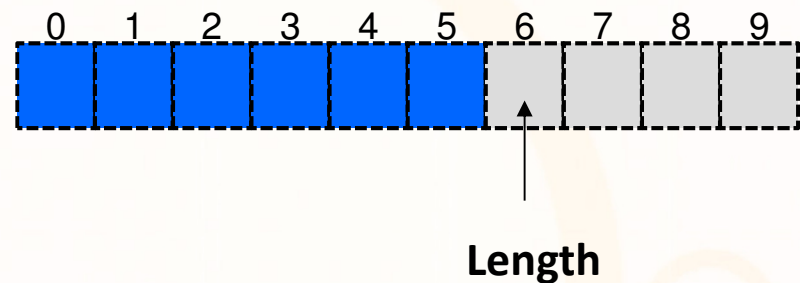
πολιτική LIFO
last in, first out

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 (πχ `struct node student;`)
- **Δυναμική:** Δέσμευση μνήμης κατά την διάρκεια της εκτέλεσης (`struct node *student` με `malloc()` και `free()`)

Στοίβα με Στατική Δέσμευση 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 $\alpha_1, \alpha_2, \dots, \alpha_n$ χρειαζόμαστε ένα πίνακα A 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).



Στοίβας με Στατική Δέσμευση Μνήμης: Υλοποίηση

- Ο τύπος δεδομένων για τη **Στοίβα** με **Στατική** Δέσμευση Μνήμης είναι:

```
typedef struct {  
    type list[ size ]; //π.χ., size=10  
    int Length;  
} STACK;
```

Στατική Δέσμευση size
θέσεων μνήμης

- Υλοποίηση πράξεων:

```
void MakeEmpty(STACK *S){  
    S->Length = 0;  
}
```

```
int IsEmpty(STACK *S){  
    return (S->Length == 0);  
}
```

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

```
void Push(type x, STACK *S){  
    if ((S->Length) < size) {  
        S->list[S->Length]= x;  
        (S->Length)++;  
    }  
}
```

```
void Pop(STACK *S){  
    if ( !IsEmpty(S) )  
        (S->Length)--;  
}
```

ΑΤΔ Λίστα 2 : Ουρές (queues)

- Νέες εισαγωγές γίνονται στο πίσω άκρο.
- Εξαγωγές από το μπροστινό άκρο
- Ορίζουμε μια ουρά ως μια λίστα συνοδευόμενη από τις πιο κάτω πράξεις:



MakeEmptyQueue()

δημιούργησε την κενή ουρά $\langle \rangle$.

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 .

ΑΤΔ Λίστα 2 : Ουρές (queues) (συν.)

- Οι πράξεις αυτές προδιαγράφονται από τους εξής κανόνες

$\text{IsEmptyQueue}(\text{MakeEmptyQueue}()) = \text{true}$

$\text{IsEmptyQueue}(\text{EnQueue}(x, Q)) = \text{false}$

$\text{DeQueue}(\text{MakeEmptyQueue}()) = \text{error}$

πολιτική FIFO
first in, first out

$\text{Top}(\text{MakeEmptyQueue}()) = \text{error}$

$\text{Top}(\text{EnQueue}(x, Q)) = \begin{cases} x & \text{if } \text{IsEmptyQueue}(Q) \\ \text{Top}(Q) & \text{else} \end{cases}$

Προτού να
εισάγουμε το x



Ουρά (απλή) με Στατική Δέσμευση Μνήμης: Υλοποίηση

- Ο τύπος δεδομένων για τη **Ουρά** με **Στατική** Δέσμευση Μνήμης είναι:

```
typedef struct {  
    type list[ size ];  
    int front;  
    int rear;  
} QUEUE;
```

- Υλοποίηση πράξεων:

```
void MakeEmpty(QUEUE *Q){  
    Q->front = Q->rear = 0;  
}
```

```
int IsEmpty(QUEUE *Q){  
    return (Q->front == Q->rear);  
}
```

```
type Top( QUEUE *Q ){  
    if ( !IsEmpty(Q) )  
        return Q->list[Q->front];  
}
```

Ουρά (απλή) με Στατική Δέσμευση Μνήμης: Υλοποίηση

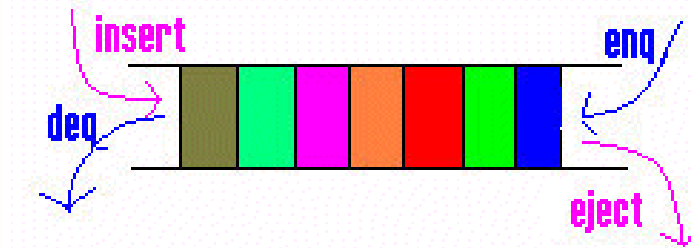
- Υλοποίηση πράξεων (συν.):

```
void Enqueue(type x, QUEUE *Q){  
    if ((Q->rear) < size) {  
        Q->list[Q->rear] = x;  
        Q->rear++;  
    }  
}
```

```
void Dequeue(QUEUE *Q){  
    if (!IsEmpty(Q) && (Q->front) < (Q->rear) )  
        Q->front += 1;  
}
```


ΑΤΔ Λίστα 3 : 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: **MakeEmptyQueue()**, **IsEmptyQueue(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)

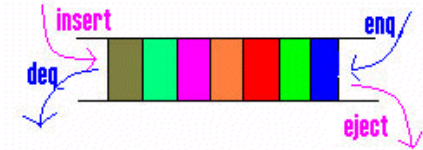
δώσε τον κόμβο στο μπροστινό άκρο της Q

Rear(Q)

δώσε τον κόμβο στο πίσω άκρο της Q

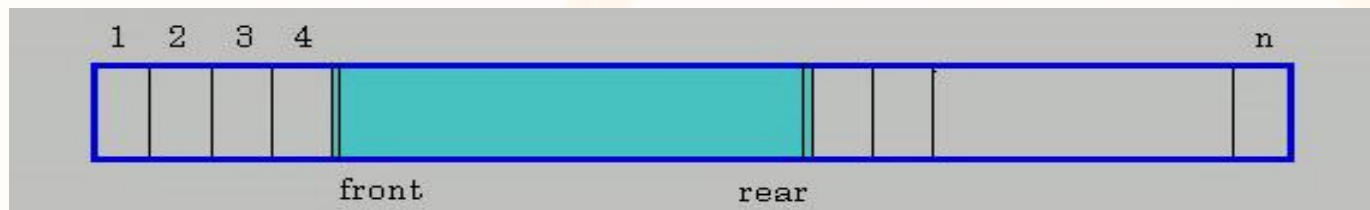
Ουρές με Δύο Άκρα με Στατική Δέσμευση Μνήμης

- Για την παράσταση μιας ουράς με στοιχεία $\alpha_0, \alpha_1, \dots, \alpha_{n-1}$ 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.



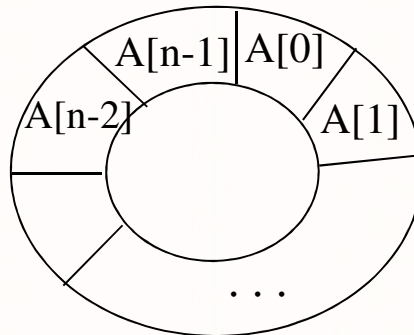
- Έτσι χρησιμοποιούμε μια εγγραφή με τρία πεδία
 1. ένα πίνακα $A[n]$,
 2. μια μεταβλητή **front**, τύπου *ακέραιος*, που συγκρατεί τη θέση που βρίσκεται αμέσως πριν τη θέση εξόδου, και
 3. μια μεταβλητή **rear**, τύπου *ακέραιος*, που συγκρατεί τη θέση του τελευταίου στοιχείου της ουράς.

Μέγεθος ουράς: $\text{rear} - \text{front} + 1$ (πχ $\text{rear} = 2, \text{front} = 0 \Rightarrow \text{length} = 3$)



ΑΤΔ Λίστα 4 : 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.

