



# Structuri de Date

---

Anul universitar 2019-2020  
Prof. Adina Magda Florea



# Curs Nr. 4

---

## Stive. Cozi

- Definitie. Operatii de baza
- Implementare stiva
- Implementare coada
- Forme ale expresiilor aritmetice

# Colectii cu disciplina de prelucrare dictata de ordinea inserarii elementelor

---

**Stiva:** ultimul venit – primul servit  
(**L**ast **I**n **F**irst **O**ut – **LIFO**)



**Coadă:** primul venit – primul servit  
(**F**irst **I**n **F**irst **O**ut – **FIFO**)

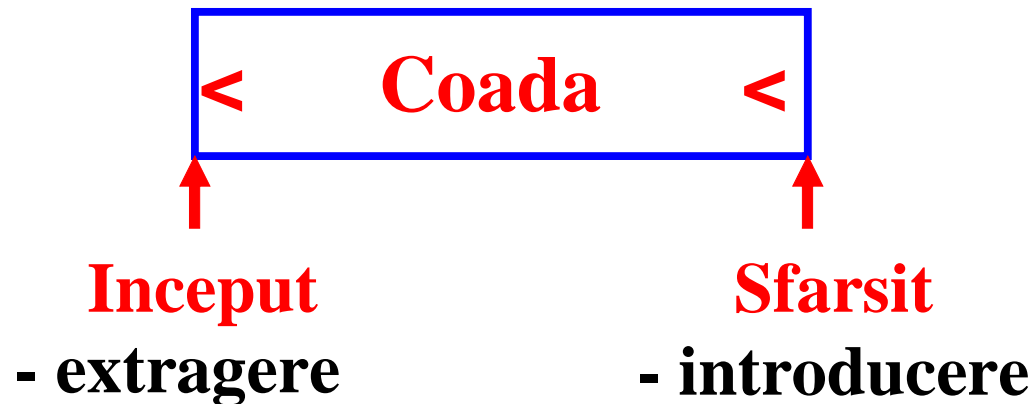


# Colectii cu disciplina de prelucrare dictata de ordinea inserarii elementelor

---

**Cooda:** primul venit – primul servit  
(**F**irst **I**n **F**irst **O**ut – **FIFO**)

*Analogii:* cereri rezolvate in ordinea inregistrarii

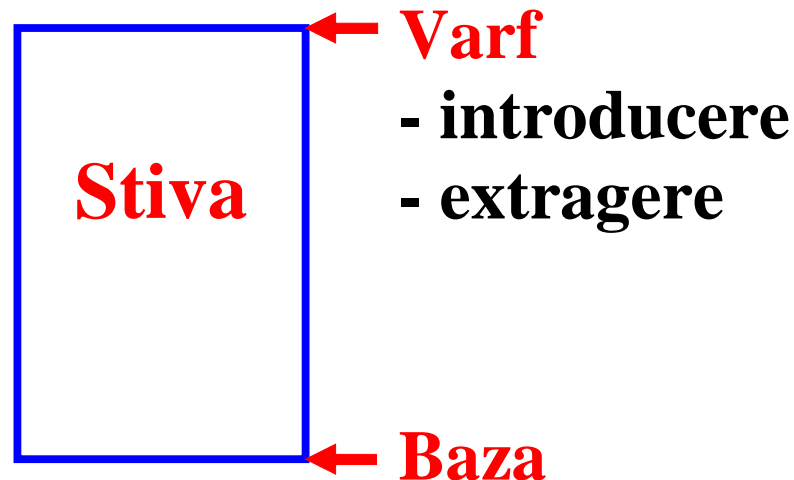


# Colectii cu disciplina de prelucrare dictata de ordinea inserarii elementelor

---

**Stiva:** ultimul venit – primul servit  
(**L**ast **I**n **F**irst **O**ut – **LIFO**)

*Analogii:* tub de medicamente, dosare puse  
teanc in ordinea primirii si parcurse la finalul  
zilei in ordinea in care sunt asezate etc.



# ADT Stiva - Operatii de baza

---

**Stiva: (Last In First Out – LIFO):**

- Initializare stiva (InitStack)
- Test stiva vida (IsEmptyStack)
- Adauga element in stiva (Push)
- Extrage element din stiva (Pop)
- Obtine elementul din varful stivei – fara a modifica continutul acesteia (Top)
- Elibereaza spatiul ocupat de intreaga structura (DistrS)

# ADT Coada - Operatii de baza

---

**Coada: (Last In First Out – LIFO)**

- Initializare coada (InitQueue)
- Test coada vida (IsEmptyQueue)
- Adauga element in coada (Enqueue)
- Extrage element din coada (Dequeue)
- Obtine primul element din coada – fara a modifica continutul acesteia (PrimQ/FrontQ)
- Elibereaza spatiul ocupat de intreaga structura (DistrQ)

## Alte operatii

---

- Test stiva/coada plina
- Numar elemente din stiva/coada
- Muta primul element din stiva/coada sursa in stiva/coada destinatie
- Transfera continut stiva/coada sursa in stiva/coada destinatie



# Implementare stivă

---

## ■ Vector

```
typedef int Item;
typedef struct {
    Item    *elements;
    int     top;
    int     maxSize; } TStack;
```

```
void InitStack(TStack *s, int maxSize)
```

```
int IsEmptyStack(TStack *s)
```

```
void Push(TStack *s, Item elem)
```

```
Item Pop(TStack *s)
```

```
int IsFullStack(TStack *s)
```

# Implementare stivă

---

## ■ Lista

```
typedef struct cel {  
    Item    elem;  
    struct  cel  *next;} StackCel,  
                                *TStack;
```

```
TStack InitStack()
```

```
int IsEmptyStack(TStack s)
```

```
TStack Push(TStack s, Item el)
```

```
TStack Pop(TStack s, Item *el)
```

```
int IsFullStack(TStack s)
```

# Stiva implementata cu vector (1)

---

```
typedef int Item;
typedef struct {
    Item    *elements;
    int     top;
    int     maxSize; } TStack;

void InitStack(TStack *s, int maxSize)
{ Item *newst;
  newst = (Item*)malloc(sizeof(Item)*maxSize);
  if (newst == NULL) { printf("Eroare\n");
                      exit(1);}

  s->elements = newst;
  s->maxSize = maxSize;
  s->top = -1;
}
```

# Stiva implementata cu vector (2)

---

```
void DistrS(TStack *s)
{
    free(s->elements);
    s->elements = NULL;
    s->maxSize = 0;
    s->top = -1;
}
```

```
int IsEmptyStack(TStack *s)
{
    return s->top < 0;
}
```

```
int IsFullStack(TStack *s)
{
    return s->top >= s->maxSize - 1;
}
```

# Stiva implementata cu vector (3)

---

```
void Push(TStack *s, Item elem)
{   if (IsFullStack(s))
    {   printf("Stiva plina\n"); exit(1); }
    s->elements[++s->top] = elem; }
```

```
Item Pop(TStack *s)
{   if (IsEmptyStack(s))
    {   printf("Stiva goala\n"); exit(1); }
    return s->elements[s->top--]; }
```

# Stiva implementata cu lista (1)

---

```
typedef struct cel {  
    Item    elem;  
    struct  cel  *next;} StackCel, *TStack;
```

```
TStack InitStack()  
{ TStack s;  
    return s=NULL;  
}
```

```
int IsEmptyStack(TStack s)  
{ return s==NULL;}
```

## Stiva implementata cu lista (2)

---

```
TStack Push(TStack s, Item el)
{
    TStack t;
    t = (TStack) malloc(sizeof(StackCel));
    if (t == NULL)
    {
        printf("memorie insuficienta \n");
        return NULL;
    }
    t->elem = el; t->next = s;
    return t;
}
```

## Stiva implementata cu lista (3)

---

```
TStack Pop(TStack s, Item *el)
{
    TStack t;
    if(s == NULL)
        {printf("stiva vida \n"); return NULL;}
    *el = s->elem;
    t = s; s = s->next; free(t);
    return s;
}
```



# Implementare coadă (FIFO)

---

## ■ Vector

```
typedef struct {
    Item    *elements;
    int      front; /* capul cozii */
    int      count; /* nr elemente din coada */
    int      maxSize; } TQueue;

void InitQ(TQueue *q, int maxSize)
int IsEmptyQueue(TQueue *q)
void Enqueue(TQueue *q, Item elem)
Item Dequeue(TQueue *q)
int IsFullQueue(TQueue *q)
```

# Implementare coadă (FIFO)

---

## ■ Lista

```
typedef struct cel {  
    Item elem;  
    struct cel *next;} QueueCel, *AQueue;  
typedef struct Queue {AQueue front, rear;}  
                        TQueue;
```

```
TQueue InitQueue()
```

```
int IsEmptyQueue(TQueue q)
```

```
TQueue Enqueue(TQueue q, Item el)
```

```
TQueue Dequeue(TQueue q, Item *el)
```

```
int IsFullQueue(TQueue q)
```

# Coada implementata cu vector (1)

---

```
typedef struct {
    Item    *elements;
    int      front; /* capul cozii */
    int      count; /* nr elemente din coada */
    int      maxSize; } TQueue;

void InitQ(TQueue *q, int maxSize)
{ Item *newq;
  newq = (Item*)malloc(sizeof(Item)*maxSize);
  if (newq == NULL) { printf("Eroare\n");
                      exit(1);}

  q->elements = newq;
  q->maxSize = maxSize;
  q->front = 0;
  q->count = 0;
}
```

## Coada implementata cu vector (2)

---

```
void Enqueue(TQueue *q, Item elem)
{ int newElementIndex;
  if (q->count >= q->maxSize)
    { printf("Coada plina\n"); exit(1); }
  newElementIndex =
    (q->front + q->count) % q->maxSize;
  queue->elements[newElementIndex] = elem;
  queue->count++; }
```

# Coada implementata cu vector (3)

---

```
Item Dequeue(TQueue *q)
{ int oldElem;
  if (q->count <= 0)
    { printf("Eroare\n"); exit(1); }
  oldElem = q->elements[q->front];
  q->front++;
  q->front %= q->maxSize;
  q->count--;
  return oldElem;
}
```

# Coada implementata cu lista (1)

---

```
typedef struct cel {  
    Item elem;  
    struct cel *next;} QueueCel, *AQueue;  
typedef struct Queue {AQueue front, rear;}  
                        TQueue;
```

```
TQueue InitQueue()  
{ TQueue q;  
  q.front = q.rear = NULL; return q; }
```

```
Int IsEmptyQueue(TQueue q)  
{ return q.front == NULL;}
```

## Coada implementata cu lista (2)

---

```
TQueue Enqueue(TQueue q, Item el)
{
    AQueue p;
    p = (AQueue)malloc(sizeof(QueueCel));
    if(p==NULL){printf("Eroare\n");
                exit(1);}
    p->elem = el; p->next = NULL;
    if(q.front == NULL)
        q.front = q.rear = p;
    else { q.rear->next = p;
          q.rear = p; }
    return q;
}
```

## Coada implementata cu lista (3)


---

```
TQueue Dequeue(TQueue q, Item *el)
{
    AQueue p; Item t;
    if(q.front == NULL)
    {
        printf("coada vida\n"); exit(1);
    }
    *el = q.front->elem;
    if(q.front == q.rear)
    {
        free(q.front); q.front = q.rear = NULL;
    }
    else
    {
        p = q.front;
        q.front = q.front->next;
        free(p);
    }
    return q;
}
```




# Mutarea unui element

**Cooda:**

Destinatie	Sursa
< 1, 2, 3 <	< <b>20</b> , 13, 8, 5 <
	
< 1, 2, 3, <b>20</b> <	< 13, 8, 5 <

**Stiva :**

Destinatie	Sursa
1, 2, 3 :	20, 13, 8, <b>5</b> :
	
1, 2, 3, <b>5</b> :	20, 13, 8 :

Echivalenta cu **extragere** urmata de **introducere**.

# Transfer continut Coada / Stiva

## Concatenare cozi

Destinatie	Sursa
< 1, 2, 3 <	< 13, 8, 5 <
⇓	
< 1, 2, 3, 13, 8, 5 <	<<

Rastoarna stiva **sursa** in stiva destinatie

Destinatie	Sursa
11, 22 :	21, 22, 23 :
⇓	
11, 22, 23, 22, 21 :	:

Pot fi realizate prin **mutari repetate**.

# Suprapunere Stive

Destinatie	Sursa
11, 22 :	21, 22, 23 :
⇓	
11, 22, 21, 22, 23 :	:

Poate fi realizata prin **mutari repetate**, folosind o **stiva auxiliara**:

```
{ initializeaza stiva aux;  
  cat timp sursa nevida  
  { muta element din sursa in aux; }  
  cat timp aux nevida  
  { muta element din aux in destinatie; }  
  distruge stiva aux;  
}
```

# Transfer stiva noua

**Transfer intr-o noua stiva** a elementelor mai mici decat o valoare de referinta, pastrand ordinea relativa a elementelor.

**Exemplu:**

sursa | 5, 10, 27, 3, 8, 21, 4 :



**s\_mici** | 4, 8, 3, 5 : (ordine inversa!)

**s\_mari** | 21, 27, 10 :



sursa | 10, 27, 21 :

**mici** | 5, 3, 8, 4 :

# Forma expresiilor aritmetice

---

- Forma infixata
- Forma prefixata (notatie poloneza)
- Forma postfixata (notatie poloneza inversa)

Utile la compilatoare si interpretoare

- Transformare din forma infixata in forma postfixata
- Evaluare expresie in forma postfixata

# Evaluare expresie in forma postfixata

*Folosim o stiva*

**pentru** fiecare atom X din expresia postfixata **repeta**

**daca** X este operator **atunci**

op1 = pop (s)

op2 = pop(s)

rez = aplica X pe op1 si op2

push(rez)

**altfel**

**daca** X este operand **atunci**

push(s,X)

rez = pop(s)

**ABC\*DEF^ /G\* -H\* +**

**A+ (B\*C-(D/E^F)\*G)\*H,**

$A + (B * C - (D / E ^ F) * G) * H,$

Input      Stiva      Output

1.		(	
2.	A	(	A
3.	+	(+	A
4.	(	(+(	A
5.	B	(+(	AB
6.	*	(+(*	AB
7.	C	(+(*	ABC
8.	-	(+(-	ABC*
9.	(	(+(-(	ABC*
10.	D	(+(-(	ABC*D
11.	/	(+(-(/	ABC*D
12.	E	(+(-(/	ABC*DE
13.	^	(+(-(/^	ABC*DE
14.	F	(+(-(/^	ABC*DEF
15.	)	(+(-	ABC*DEF^/
16.	*	(+(-*	ABC*DEF^/
17.	G	(+(-*	ABC*DEF^/G
18.	)	(+	ABC*DEF^/G*-
19.	*	(+*	ABC*DEF^/G*-
20.	H	(+*	ABC*DEF^/G*-H
21.	)	Empty	ABC*DEF^/G*-H*+

Transformare  
expresie in  
forma postfixata  
- exemplu

# Transformare expresie in forma postfixata

**cat timp** mai sunt atomi X in input **repeta**

**daca** X este operand (numar sau o variabila) **atunci**

        ■ scrie X in output

**daca** X este un operator **atunci**

**cat timp** exista un operator in varful stivei cu precedenta  
            mai mare sau egala cu X **si** varful stivei nu este “)”

**repeta**

            ■ pop operator din stiva de operatori si pune-l in Y

    ■ push X in stiva

**daca** X = “(“ **atunci** push X in stiva

**daca** X = “)” **atunci**

**cat timp** op varf stiva < > “(“ **repeta**

            ■ pop operator din stiva si pune-l in Y

**daca** varf stiva = “(“ **atunci**

            pop stiva si ignora “(“