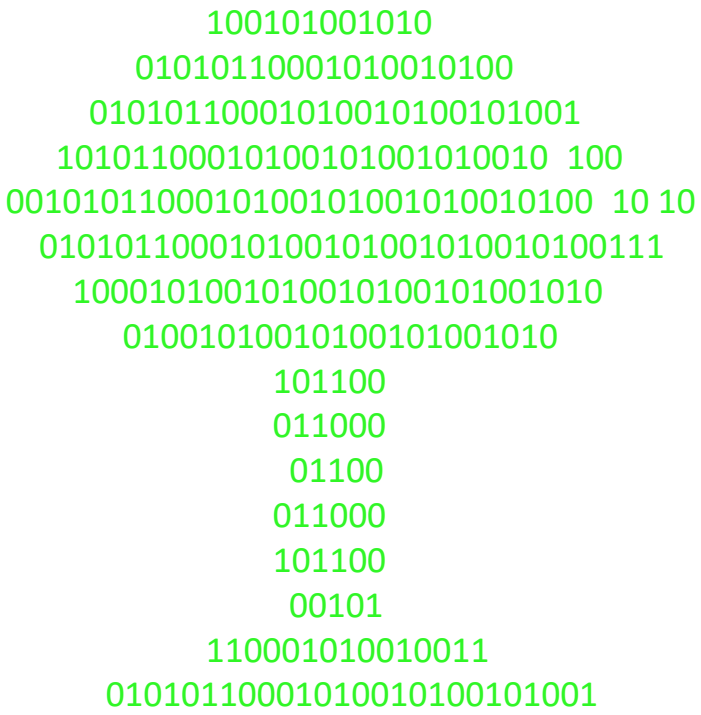
Structuri de date

Curs, CTI – An I


100101001010
0101011000101001010
0101011000101001010010100
1010110001010010100101001
10101100010100101001010010 100
00101011000101001010010100101001 10 10
0101011000101001010010100101001111
1000101001010010100101001010
010010100101001010010
101100
011000
01100
011000
101100
00101
110001010010011
0101011000101001010010100

Stive

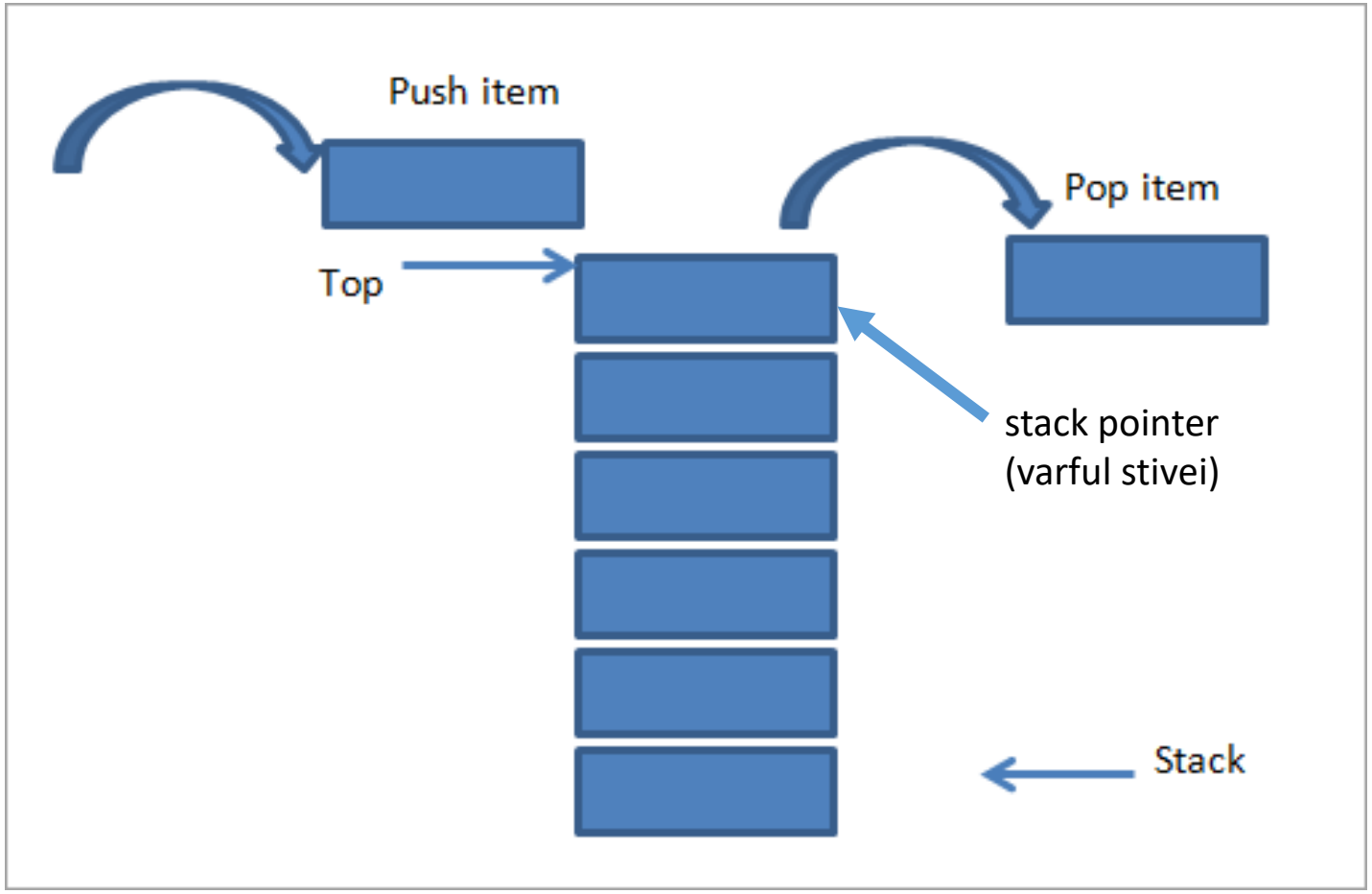
STIVE

- o lista specializata organizata pe principiul **LIFO** (Last In First Out) / **FILO** (First In Last Out):
  - Inserarea si stergerea se realizeaza numai la un (acelasi) capat al listei

Operatii de baza pe TAD Stiva (Stack):

- Push: adaugare element
- Pop: extragere element
- Top: returnarea valorii primului element
- Init: initializeaza stiva (stiva vida)
- IsEmpty: intoarce 1 (true) daca stiva este goala si 0 (false) daca are elemente

STIVE



STIVE

Utilizare:

- Inversarea unei secvente (cuvant, numere, etc.)
- Verificare daca un cuvant este palindrom
- Mecanismul de “undo” (modificarile sunt pastrate intr-o stiva)
- In algoritmi de tip backtracking (explorarea spatiului solutiilor):
  - ex. Gasirea drumului intr-un labirint – variantele de explorare se stocheaza intr-o stiva
- Evaluarea expresiilor aritmetice
  - utilizand forma pre-fixata/post-fixata

STIVE

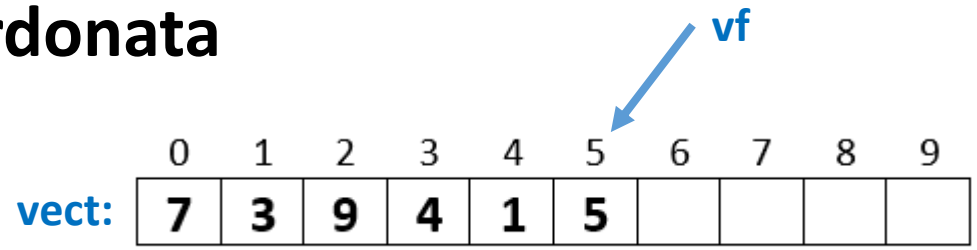
Reprezentarea in memorie:

A. Stiva ordonata (alocata static)

A. Stiva dinamica (alocata dinamic)

STIVE

A. Stiva ordonata

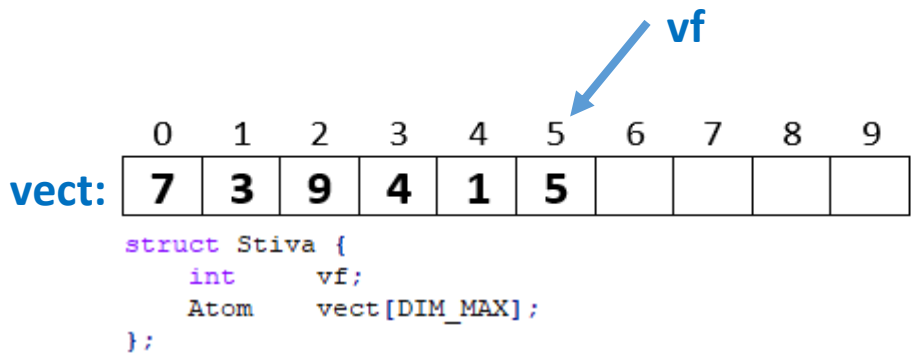



```
#define DIM_MAX 10

typedef ... Atom                    //ex: typedef int Atom

struct Stiva {
    int     vf;                     //stack pointer
    Atom    vect[DIM_MAX];          //tabloul ce contine elementele stivei
};
```

STIVE



A. Stiva ordonata

Operatii in stiva ordonata

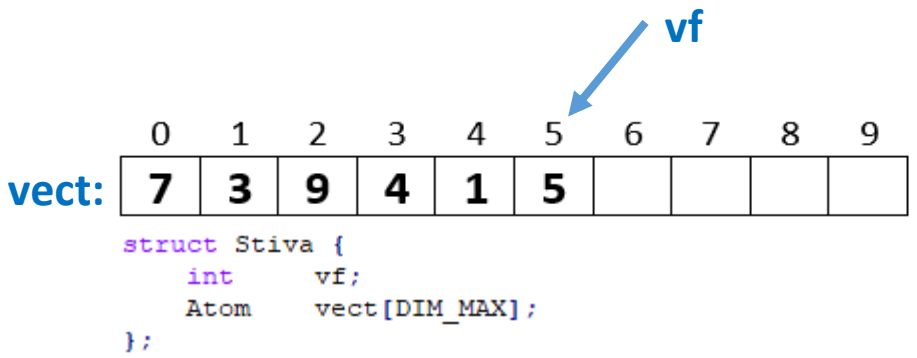
```
InitStack(s)
        s.vf := -1
End
```

```
IsFull(s)
        if(s.vf = DIM_MAX-1)
                return true
        end-if
        return false
End
```

```
IsEmpty(s)
        if(s.vf = -1)
                return true
        end-if
        return false
End
```

```
Top(s)
        return s.vect[s.sp]
End
```

STIVE



A. Stiva ordonata

Operatii in stiva ordonata

```
Push(s, val)
        s.vf ++
        s.vect[s.vf] := val
End
```

```
Pop(s)
        s.vf--
End
```

Atentie la mecanismul de tratare a erorilor!

V1: lasat in seama utilizatorului stivei (ca mai sus)

V2: integrat in implementarea operatiilor Top/Pop/Push

- returnare valoare de success la realizarea operatiei
- mesaj de eroare

STIVE

```
struct Stiva {
    int      vf;
    Atom     vect[DIM_MAX];
};
```

A. Stiva ordonata

Prototipurile functiilor in C++

```
void InitStack(Stiva &s);
bool isEmpty(const Stiva &s);
Atom Top(const Stiva &s);
void Push(Stiva &s, Atom val);
void Pop(Stiva &s);
```

Transmiterea prin referinta pentru a evita copierea parametrului (s . vect poate fi mare)!!!

```
InitStack(s)
        s.vf := -1
End
```

```
IsEmpty(s)
        if(s.vf = -1)
                return true
        end-if
        return false
End
```

```
IsFull(s)
        if(s.vf = DIM_MAX-1)
                return true
        end-if
        return false
End
```

```
Top(s)
        return s.vect[s.sp]
End
```

```
Push(s, val)
        s.vf ++
        s.vect[s.vf] := val
End
```

```
Pop(s)
        s.vf--
End
```

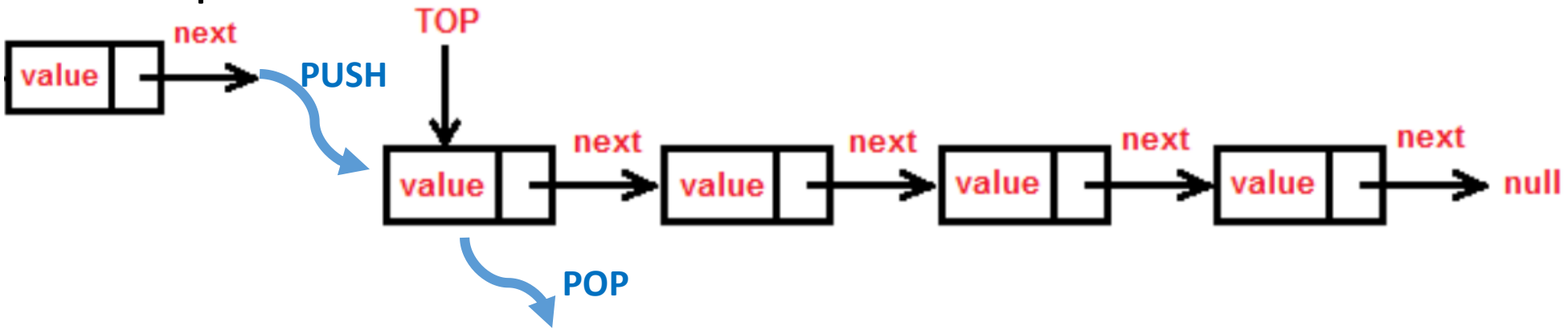
STIVE

```
struct Element {
    Atom    value;
    Element *next;
};

typedef Element* Stiva;
```

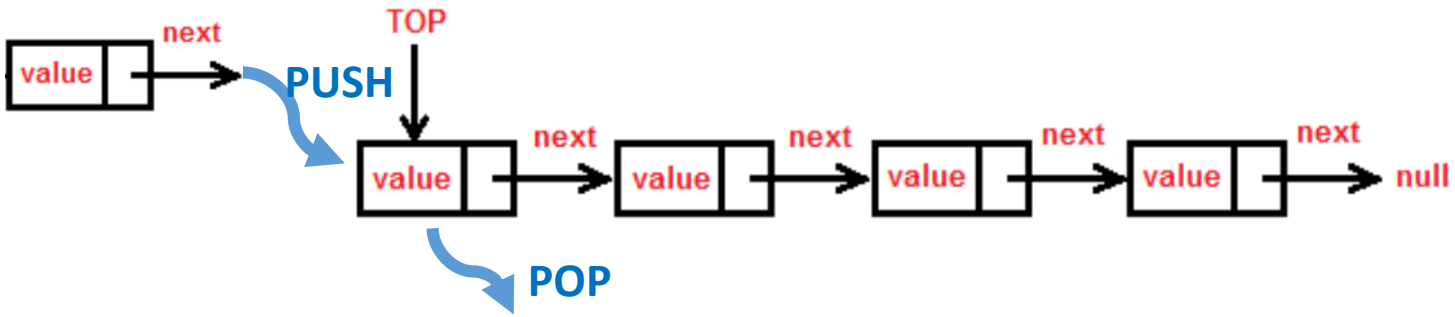
A. Stiva dinamica

- lista liniara simplu inlantuita



- PUSH – inserare in fata listei
- POP – stergerea primului element
- TOP – consultarea primului element

STIVE



A. Stiva dinamica

Operatii in stiva dinamica

```
InitStack(s)
        s := 0
End
```

```
IsEmpty(s)
        if(s = 0)
                return true
        end-if
        return false
End
```

```
Top(s)
        return value(s)
End
```

```
Push(s, val)
        p := create_elem(val)
        value(p) := val
        next(p) := s
        s := p
End
```

```
Pop(s)
        p := s
        s := next(s)
        delete(p)
End
```

STIVE

A. Stiva dinamica

Operatii in stiva dinamica

Atentie la mecanismul de tratare a erorilor!
V1: lasat in seama utilizatorului stivei (ca in dreapta)

```
void InitStack(Stiva &s);
bool isEmpty(Stiva s);
Atom Top(Stiva s);
void Push(Stiva &s, Atom val);
void Pop(Stiva &s);
```

V2: integrat in implementarea operatiilor Top/Pop/Push
- returnare valoare de success la realizarea operatiei

```
bool Top(Stiva &s, Atom &val);
bool Push(Stiva &s, Atom val);
bool Pop(Stiva &s);
```

```
InitStack(s)
        s := 0
End
```

```
IsEmpty(s)
        if(s = 0)
                return true
        end-if
        return false
End
```

```
Top(s)
        return value(s)
End
```

```
Push(s, val)
        p := create_elem(val)
        value(p) := val
        next(p) := s
        s := p
End
```

```
Pop(s)
        p := s
        s := next(s)
        delete(p)
End
```

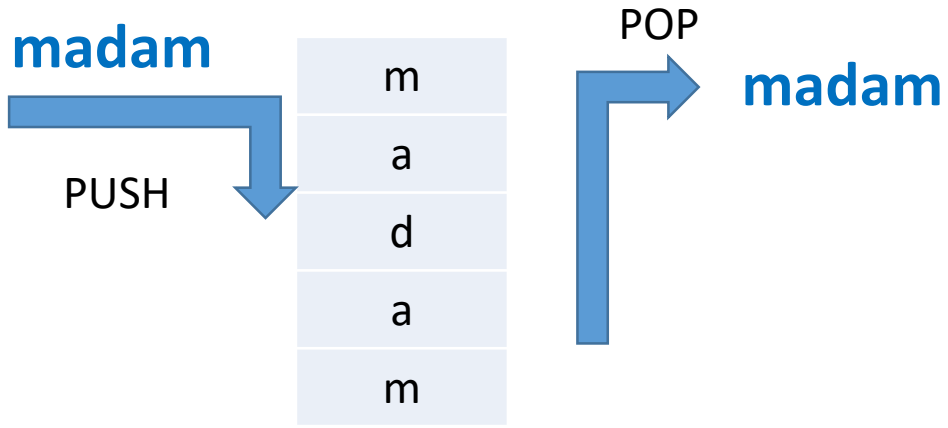
STIVE - utilizare

Biologie moleculara: Many molecular lengths between 4 and 8 nucleotides are palindromic as they correspond to nitrogenous sequences that read the same forwards as they do backward.

Verificare palindrom

abcdeedcba **123454321**

secventa care citita invers este identica cu cea originala



STIVE - utilizare

Evaluarea expresiilor aritmetice

Ideea:

- rearanjarea expresiei a.i. sa nu contina paranteze,
- ordinea in care se efectueaza operatiile sa fie clara si evaluarea usor de facut pe calc.

Forma prefixata	**Forma infixata**	**Forma postfixata**
+ab	a+b	ab+
forma poloneza		*forma poloneza inversa*

STIVE - utilizare

Forma postfixata

ab+

Evaluarea expresiilor aritmetice

Forma postfixata:

- Operatorii apar in ordinea in care se executa operatiile la evaluarea expresiei
- Operatorii apar in urma operanzilor
- Evaluarea se face parcurgand expresia stg -> drt si executand operatiile (tinand cont de precedenta lor)

Definitie

- Pentru orice operand (constanta sau variabila) *E*, *E* este forma poloneza a operandului *E*
- Daca *E* este o expresie de forma *E1 op E2* , f.p. a expresiei este *E1' E2' op* unde *E1'* si *E2'* sunt respectiv f.p. ale expresiilor *E1* si *E2*
- Daca *E* este o expresie de forma (*E1*). f.p. a expresiei *E1* este de asemenea f.p. a expresiei *E*

STIVE - utilizare

Forma postfixata
ab+

Evaluarea expresiilor aritmetice

Forma postfixata:

Expresie	Forma postfixata
E (operand sau operator)	E' = E
E1 *op* E2	E1' E2' *op*
(E1)	E1'

Expresie	Forma postfixata
a+b	ab+
4+5*5	455*+
4*2+3	42*3+
4*(2+3)	423+*
5+(2*3+4)/(6-4)	

STIVE - utilizare

Forma postfixata

ab+

Evaluarea expresiilor aritmetice

Forma postfixata – evaluare

- Se foloseste o **stiva** de operanzi
- Se parcurge expresia
  - Daca se intalneste operand -> se adauga in stiva
  - Daca se intalneste operator
    - se extrag 2 operanzi din stiva,
    - se efectueaza operatia,
    - se depune rezultatul in stiva

STIVE - utilizare

Forma postfixata
ab+

Evaluarea expresiilor aritmetice

Forma postfixata:

```
// Expresia se afla intr-un tablou s cu elemente simboluri de
// tip operand sau operator binar
i=0
while (nu s-a terminat sirul s) do
    if (s[i] este operand) then
        depune s[i] in stiva
    else    if (s[i] este operator) then
                extrage din stiva doua simboluri t1 si t2;
                executa operatia s[i] asupra lui t1 si t2;
                depune rezultatul in stiva
        else semnalizeaza eroare
        endif
    endif
    i=i+1
end-while
```

STIVE - utilizare

Forma postfixata

ab+

Evaluarea expresiilor aritmetice

Trecerea din forma **infixata** in forma **postfixata**:

- Se foloseste o **stiva** pentru stocarea temporara a operatorilor
- Se parcurge expresia infixata
  - Daca se intalneste operand → se adauga in expresia postfixata
  - Daca se intalneste operator
    - se scot din stiva toti operanzii cu precedenta mai mare sau egala, pana la ‘(’ si se adauga in expresia postfixata (exclusiv ‘(’)
    - se adauga in stiva operatorul curent
  - Daca se intalneste ‘(’ → se adauga in stiva
  - Daca se intalneste ‘)’
    - se scot din stiva toti operatorii pana la ‘(’ si se adauga la expresia postfixata
    - se scoate din stiva ‘(’

STIVE - utilizare

Forma postfixata
ab+

Evaluarea expresiilor aritmetice

Trecerea din forma **infixata** in forma **postfixata**:

- Se foloseste o **stiva** pentru stocarea temporara a operatorilor

EXEMPLU

5 + (2 * 3 + 4) / (6 - 4)

100101001010
0101011000101001010 0
0101011000101001010010 1001
1010110001010010100101001 0 100
0010101100010100101001010010100 10 10
01010110001010010100101001010010100111
1000101001010010100101001010
010001010010100101001010
101100
011000
01100
011000
101100
00101
110001010010011
0101011000101001010010100 1001

Cozi

Tipul abstract de dată – Coadă (En. Queue)

Coadă la un magazin Apple din New York (2011)

Drum cu sens unic și o singură bandă

Coada – tip abstract de date (TAD) de tip FIFO (sau LILO)

TAD Coadă

O coadă este o listă specială organizată pe principiul FIFO (First In – First Out) sau primul sosit – primul servit:

- inserarea se realizează numai la sfârşitul listei
- ştergerea se realizează numai de la începutul listei

Operaţii pe structuri de tipul coadă

- Put: un atom (nod) este inserat la sfârşitul cozii (echivalent *Insert*, *Enqueue*);
- Get: primul atom (nod) al cozii este îndepărtat (echivalent *Delete*, *Dequeue*);
- Front: returnează informația primului atom (nod) al cozii fără să-l șteargă (echivalent *Top*, *Peak*);
- Init: inițializează coada (coadă vidă);
- IsEmpy: întoarce 1 (true) dacă coada este goală și 0 (false) dacă coada conține atomi.

TAD Coadă

Exemple de utilizare a cozilor:

- Playlist
- Bufferul (zona tampon) de date la iPod
- Comenzile de tipărire în coada de aşteptare a imprimantei
- Comenzile clienţilor pe un portal Web

Există situații speciale în care principiul FIFO este încălcat (de pildă cererile de întreruperi către sistemul de operare, sau modelarea cazului în care o persoană cu nevoi speciale ”sare” peste coadă) -> *Coada de priorități* pe care o vom dicuta peste câteva săptamâni.

Implementare (similar listelor sau stivelor)

- Dinamic
- Static

Coada – Implementare dinamică

Deoarece operațiile de inserare (put), ștergere (get) și consultare sunt permise doar la unul din cele două capete ale cozii, în cazul implementării dinamice este nevoie de doi pointeri către primul și ultimul atom (nod) din structură:

- ✓ head (primul nod inserat) - astfel încât să *obţinem /(get, front)* primul element din coadă;
- ✓ tail (ultimul nod inserat) - astfel încât să putem *insera/put* un nou element în coadă.

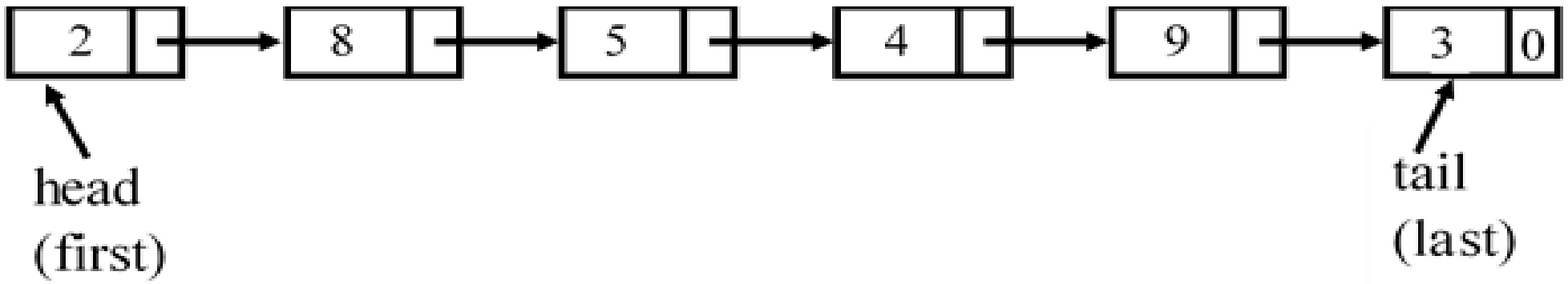


Fig. C1

Coada – Implementare dinamică

Definiție

```
typedef  int atom;

struct Element
{
   Atom info;
   Element *succ;
};

struct Queue
{
   Element *head, *tail;//pointeri la primul
                        //și ultimul element
} ;
```

Funcții

```
Queue InitQ(void);
int IsEmptyQ(Queue q);
void Put(Queue &q, Atom x);
Atom Get(Queue &q);
Atom Front(Queue q);
```

Coada – Implementare dinamică

```
Queue InitQ(void)
{
    Queue q;
    q.head = q.tail = 0;//ambii pointeri sunt nuli
   return q;
}

int IsEmptyQ(Queue q)
{
   return (q.head == 0 && q.tail==0);//testez dacă ambii pointeri sunt nuli
}
```

Coada – Implementare dinamică

Put – inserarea unui nod în coada din figura C1

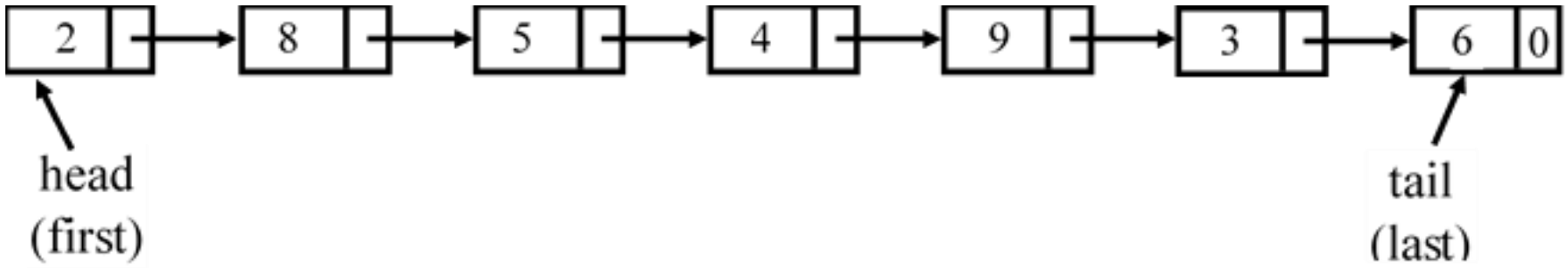
Queue q;

p<- get_sp();// se alocă memorie pentru un nou Element

info(p)<-val;// 6 in cazul dat

succ(p)<-0;// va fi ultimul element

succ(q.tail)<-p



Ultimul nod a fost cel cu info 3 și s-a inserat nodul de info 6.

Coada – Implementare dinamică

Get – extragerea (ștergerea) unui nod din coada din figura C1

Queue q;

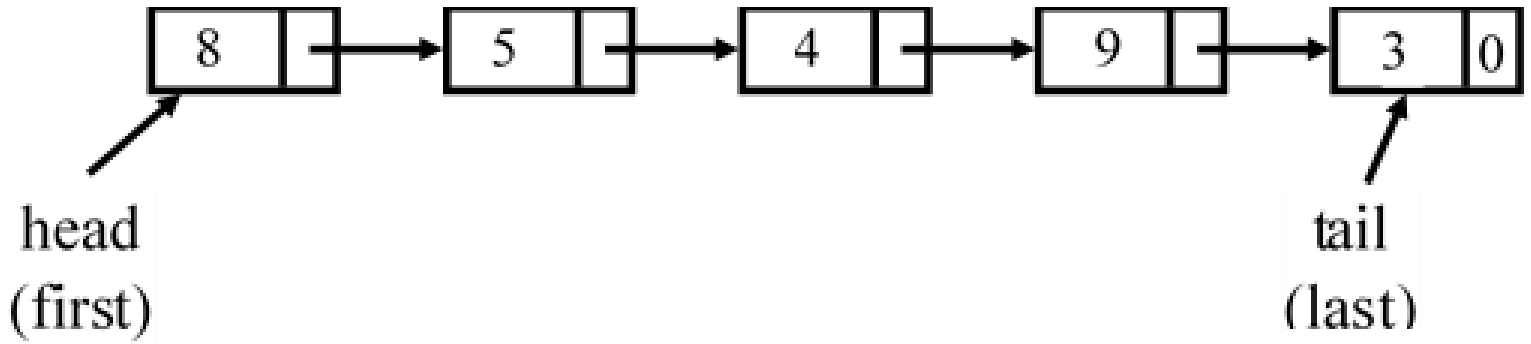
p<- q.head;// pointer spre primul nod ce urmează a fi șters

aux<-info(p);//valoarea ce va fi returnata

q.head<-succ(p);// noul head

free_sp(p);// dealocare memorie

return aux;



S-a extras elementul de info 2 și elementul de info 8 a devenit head.

Coada – Implementare dinamică

- Put și Get modifică coada, Front nu (similar cu Push, Pop și Top de la stivă);
- Toate operațiile sunt de complexitate O(1) (similar stivei);
- Nu are sens să implementăm o funcție de "parcurgere" a cozii, nu este o operație specifică (similar stivei);
- Discuție privind prototipurile

```
Atom Get(Queue &q); Atom Front(Queue q);
```

- Cum tratăm cazul când coada este vidă și totuși se apelează una dintre aceste funcții?

```
Ex:  int Get(Queue &q,Atom &v);
```

Funcția întoarce un cod de eroare (1 dacă coada este vidă) iar v trebuie inițializată înainte de apel și valoarea se întoarce prin referință.

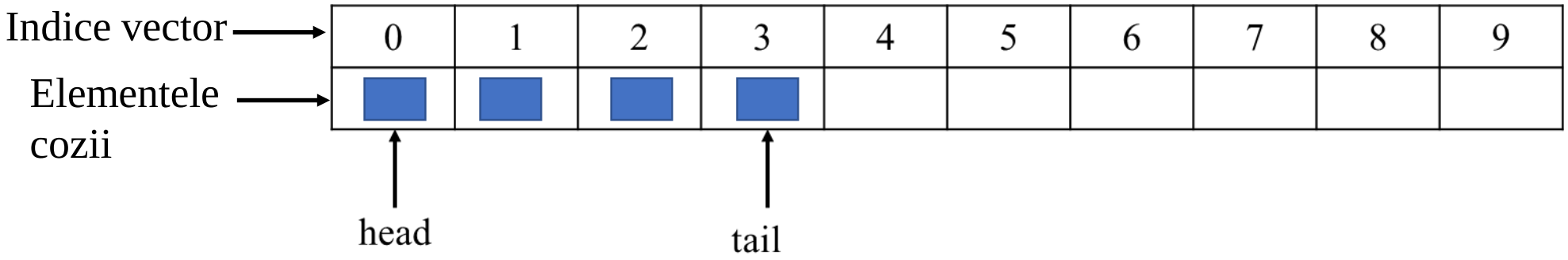
Alte soluții?

Coada – Implementare statică

O coadă poate fi implementată static pe un spațiu de memorare de tip tablou (vector).

Pentru a păstra timpul de execuţie constant O(1) atât pentru operaţia *Put* cât şi pentru *Get*, trebuie să nu mutăm elementele cozii în vector.

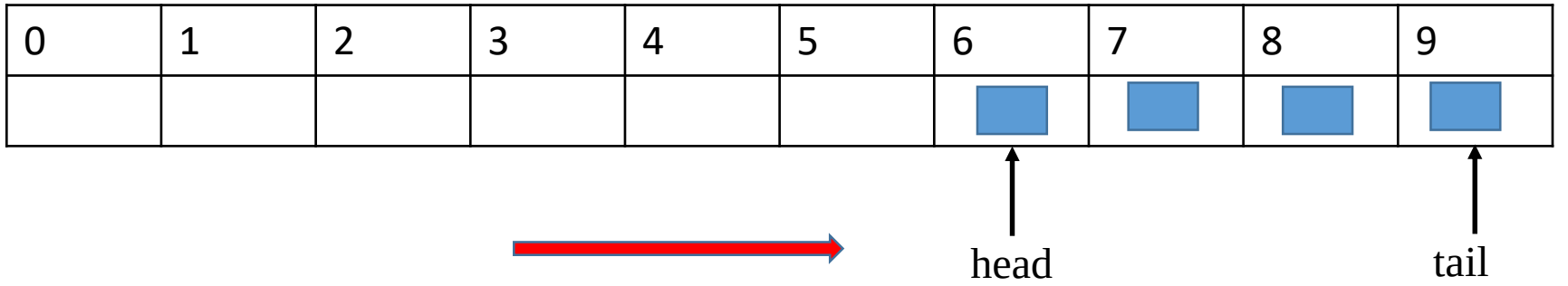
Se utilizează doi indici pe vector: unul pe începutul cozii (head) şi unul pe sfârşitul ei (tail).



Inserând și extrăgând elemente din coadă se poate ajunge la stuația următoare:

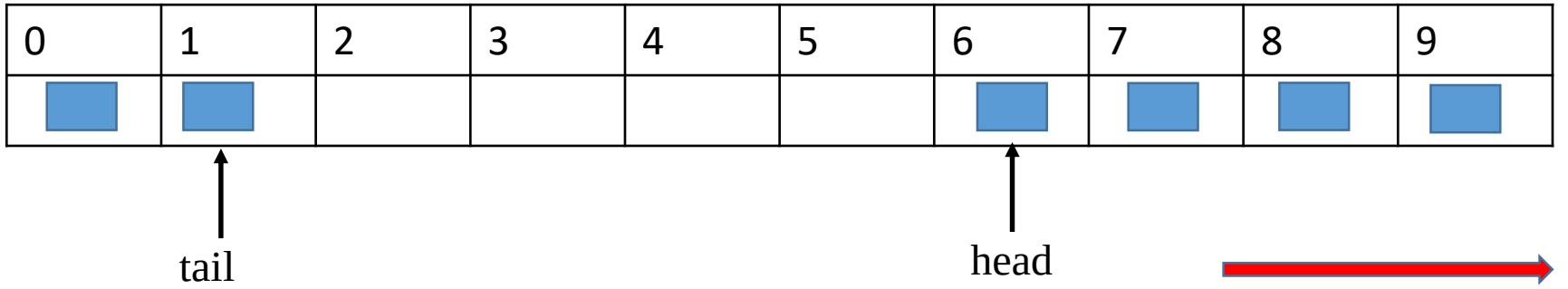
Coada – Implementare statică

Indicii se deplasează spre dreapta:



Deși mai sunt locuri libere în vector, indicele tail este pe poziția maximă și nu se mai pot insera elemente în coadă.

Soluția: să se realizeze o coadă circulară astfel încât tail să poată trece pe poziția 0.



Coada – Implementare statică

Totuși, pentru comoditatea implementării condițiilor de coadă plină, coadă goală și a trecerii celor doi indici de pe poziția maximă (9 în cazul nostru) pe prima poziție (0 în cazul unui vector), tail va indica prima poziție liberă în care se poate insera un element și NU poziția pe care s-a inserat ultimul element.

Astfel, o poziție rămâne neocupată când coada va fi plină!

Condiția de coada vidă: head ≡ tail

Condiția de coadă plină: head= următoarea valoare a lui tail după eventuală inserare

```
typedef  int Atom;
#define dimmax 10 //dimensiune vector
struct Queue
{
   int head, tail;
   Atom vect[dimmax];
};
```

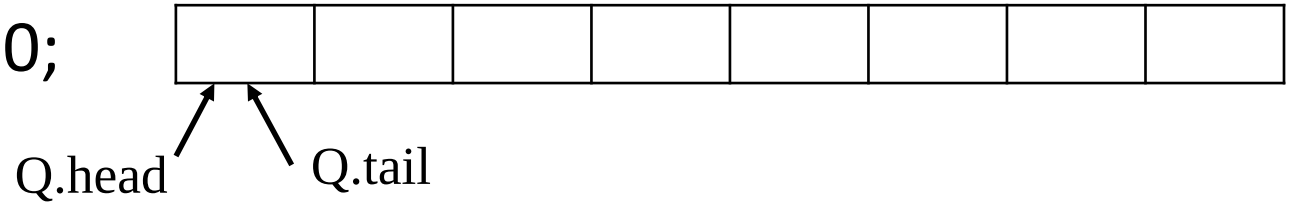
```
int NextPos(int index)
{
   if (index<dimmax - 1)//altă soluție??
      return index + 1;
   else return 0;

}//index va fi Q.head sau Q.tail dupa caz
```

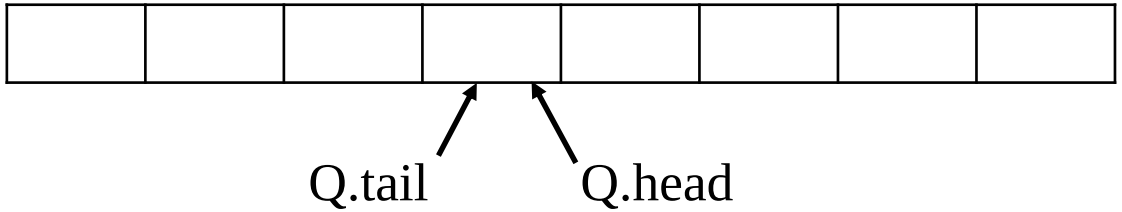
Coada – Implementare statică

```
Queue Q;
```

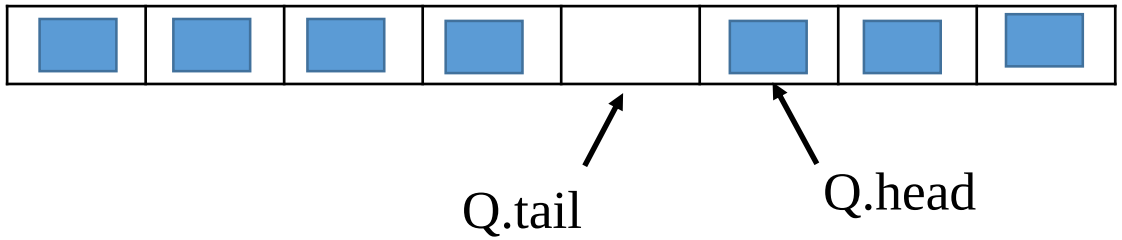
Inițializarea: Q.head = Q.tail = 0;



Coadă goală: se returnează rezultatul (Q.head == Q.tail)



Coadă plină: NextPos(Q.tail)==Q.head



Coada – Implementare statică

Prototipurile pentru coada alocată static

```
int IsEmpty(const queue &Q);
void InitQ(queue &Q);
void Put(queue &Q, Atom a);
Atom Get(queue &Q);
Atom Front(const queue &Q);
```

Link pentru simulare structuri de date:
https://www.cs.usfca.edu/~galles/visualization/Algorithms.html

Stive și cozi – Implementări în Java și C++

La curs și laborator, au fost prezentate proprietățile și implementările standard pentru stive și cozi.

Ulterior, veți întâlni implementări ușor diferite în alte medii de programare.

Operații stivă

JAVA

1. **Object push(*Object element*)** : Pune un element în vârful stivei.
2. **Object pop()** : Extrage și returnează elemental din vârful stivei. Daca se apelează când stiva este goală aruncă exceptie de tipul ‘EmptyStackException’.
3. **Object peek()** : Returnează elemental din vârful stivei, dar nu-l șterge.
4. **boolean empty()** : returnează true dacă stiva este goală și false în caz contrar.
5. **int search(*Object element*)** : Determină dacă un element există în stivă și dacă da, returnează poziția acelui element. Dacă elemental nu se află în stivă intoarce -1.

Operatii stivă, C++

1. **empty** : returnează true dacă stiva este goală și false în caz contrar
2. **size**: returnează dimensiunea stivei
3. **top**: Accesează valoarea din vârful stivei fără să șteargă
4. **push**: Insert element
5. **pop**: Sterge elemental din varful stivei, fără sa-l returneze
6. **swap:** Swap contents

```
int main()
{
   // Empty stack
   stack<int> mystack;
   mystack.push(0);
   mystack.push(1);
   mystack.push(2);

   // Printing content of stack
   while (!mystack.empty()) {
      cout << ' ' << mystack.top(); //afisează ce este în vârful stivei
      mystack.pop(); //șterge ce este în vârful stivei
   }
}
```

```
//calculeaza suma elementelor unei stive
sum=0
while (!mystack.empty()) {
        sum = sum + mystack.top();
        mystack.pop();
  }
```

Cozi - Java

- **add()-** is used to add elements at the tail of queue.
- **peek()-** is used to view the head of queue without removing it. It returns Null if the queue is empty.
- **element() –** it is similar to peek(). It throws *NoSuchElementException* when the queue is empty.
- **remove()-** removes and returns the head of the queue. It throws *NoSuchElementException* when the queue is empty.
- **poll()-** removes and returns the head of the queue. It returns null if the queue is empty.
- **size()-** return the no. of elements in the queue.

Cozi – C++

- **empty:** Test whether container is empty
- **size:** Return size
- **front:** Access next element
- **back:** Access last element
- **push:** Insert element
- **pop:** Remove next element
- **swap:** Swap contents