



# COZI

Șl. Dr. Ing. Șerban Radu

Departamentul de Calculatoare

Facultatea de Automatică și Calculatoare



# Introducere

- O coadă ("Queue"), numită și listă FIFO ("First In First Out") este o listă la care adăugarea se face pe la un capăt (la sfârșitul cozii), iar extragerea se face de la celalalt capăt (de la începutul cozii)
- Ordinea de extragere din coadă este aceeași cu ordinea de introducere în coadă



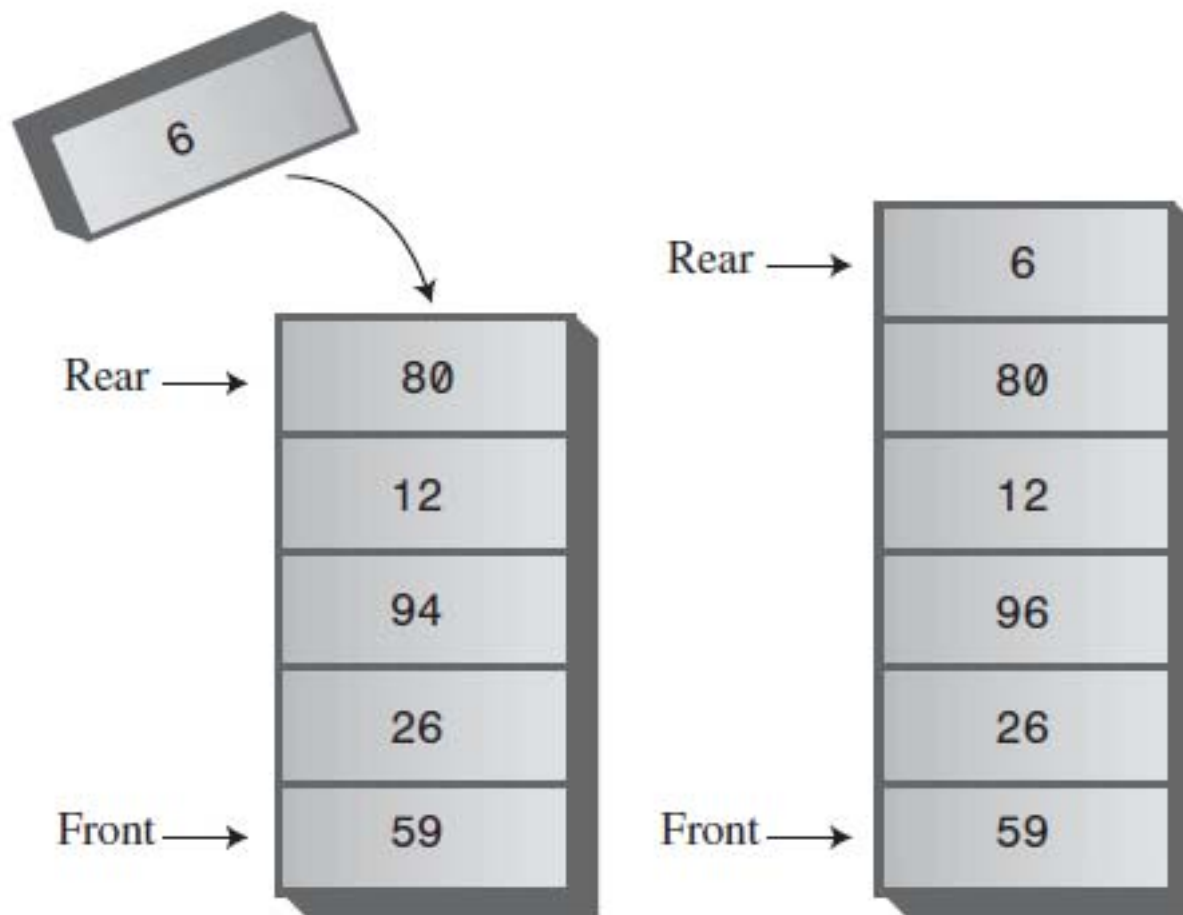
# Aplicații ale cozilor

- Există cozi care funcționează în cadrul sistemului de operare:
  - O coadă asociată imprimantei, în care aplicațiile care au ceva de tipărit așteaptă ca imprimanta să devină disponibilă
  - O coadă în care sunt păstrate apăsările tastelor, atunci când este folosită tastatura

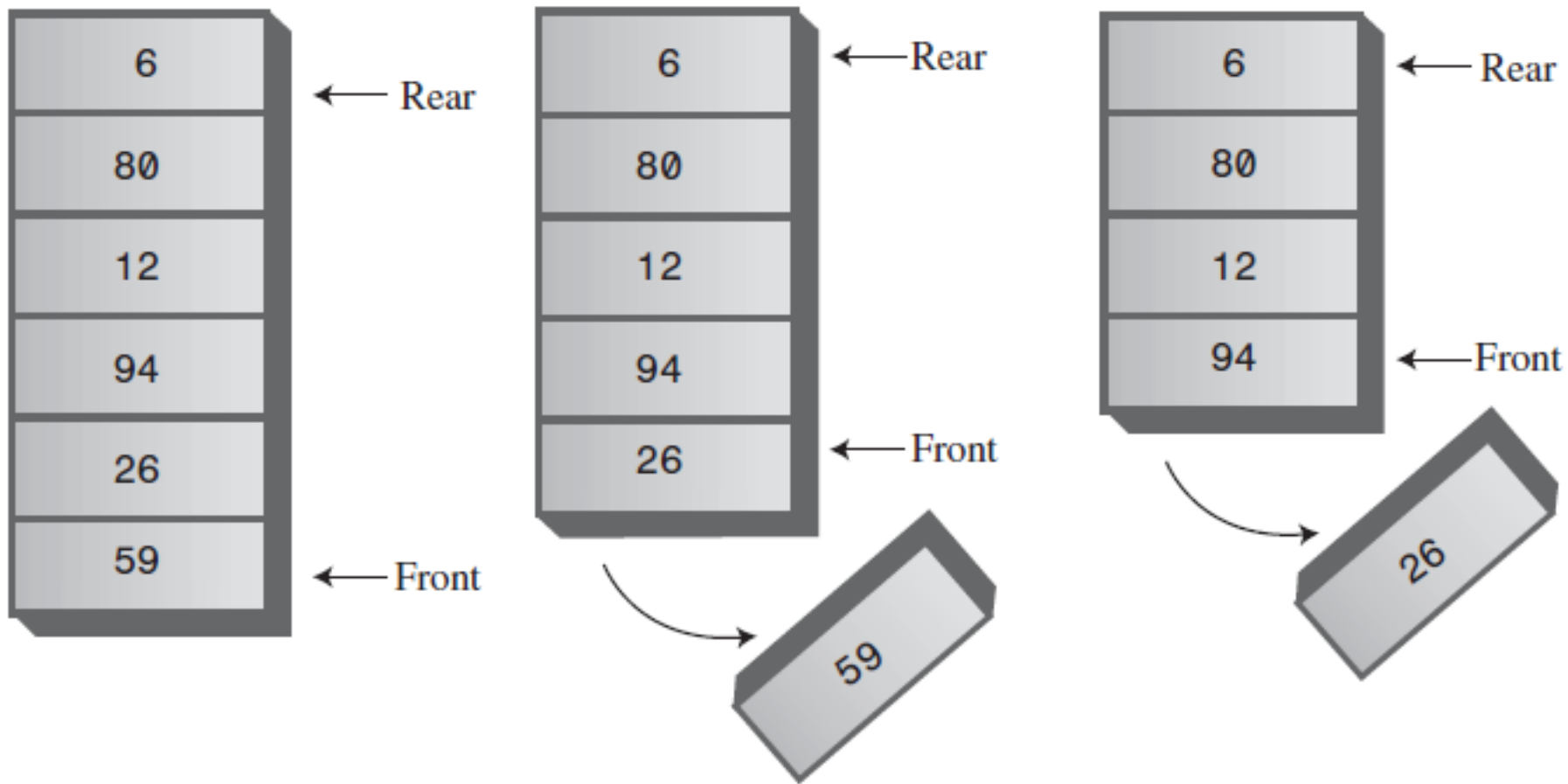


# Operații cu cozi

- Inițializare coadă (initqueue)
- Test coadă goală (emptyqueue)
- Adaugă un obiect la coadă (enqueue)
- Scoate un obiect din coadă (dequeue)
- Obține valoarea primului obiect din coadă, fără a-l scoate din coadă (peek)
- Vezi demonstrația Queue



New item inserted at rear of queue



Two items removed from front of queue




# Implementarea cozilor

- Cozile pot fi implementate în două moduri
  - Static, folosind tablouri
  - Dinamic, folosind pointeri
- Implementarea sub formă de tablou are dezavantajul lungimii finite a cozii, pentru că se declară de la început dimensiunea maximă a cozii


# Exemplu de implementare statică

- Se citește un cuvânt de la tastatură, care se introduce într-o coadă, iar apoi se citește din coadă cuvântul introdus și se afișează pe ecran






```
#include <stdio.h>
#include <string.h>
#include <conio.h>
struct queue {
    int cap, coada, lungime;
    char element[100];
} c;
```



```
int main() {  
    char sir[20];  
    int i;  
    printf("Introduceti un cuvant\n");  
    gets(sir);  
    c.cap = 0;  
    c.coada = 0;  
    c.lungime = 0;  
    for (i = 0; i < strlen(sir); i++) {  
        c.element[c.coada] = sir[i];  
        c.coada++;  
        c.lungime++;  
    }  
}
```



```
printf("Cuvantul citit din coada este\n");
do {
    printf("%c ", c.element[c.cap]);
    c.cap++;
    c.lungime--;
} while (c.coada != c.cap);
getch();
}
```




# Exemplu de implementare dinamică

- Se consideră un depou de locomotive cu intrarea prin fața depoului și cu ieșirea prin spatele depoului
- Depoul de locomotive conține o singură linie de cale ferată
- Să se implementeze un program pentru managementul locomotivelor din depou

# Cerințe program

- O locomotivă se identifică prin codul său asociat, o valoare de tip întreg
- Se procesează următoarele comenzi:
  - I – intrarea unei locomotive
  - E – ieșirea unei locomotive
  - L – listarea locomotivelor din depou
  - S – sfârșitul programului



```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
struct locomotive {
    int cod;
    locomotive *next;
} *p, *prim, *ultim;
void intrare ();
void iesire ();
void listare ();
```


```
int main() {
    char comanda;
    prim = NULL;
    do {
        printf("Introduceti operatia : ");
        comanda = getche();
        printf("\n");
        if ((comanda == 'I') || (comanda == 'i')) intrare();
        if ((comanda == 'E') || (comanda == 'e')) iesire();
        if ((comanda == 'L') || (comanda == 'l')) listare();
        if ((comanda == 'S') || (comanda == 's')) exit(0);
    } while ((comanda != 'S') || (comanda != 's'));
}
```

```
void intrare() {
    int code;
    printf ("Introduceti codul locomotivei care intra : ");
    scanf ("%d", &code);
    if (prim == NULL) {
        prim = (locomotive*) malloc(sizeof(locomotive));
        prim->cod = code;    prim->next = NULL;
        ultim = prim;
    }
    else {
        p = (locomotive*) malloc(sizeof(locomotive));
        p->cod = code;    p->next = NULL;
        ultim->next = p;
        ultim = p;
    } }
```



```
void iesire(){
    if (prim == NULL)
        printf("Depoul e gol; repetati comanda mai tarziu\n");
    else {
        printf("Iese locomotiva cu codul %d\n", prim-
>cod);
        p = prim;
        //se salveaza adresa primului element
        prim = p->next;
        //urmatorul element dupa primul devine primul
        //element

        free (p);
    }
}
```



```
void listare() {  
    if (prim == NULL) printf("Depoul e gol\n");  
    else {  
        printf("Locomotivele din depou sunt:\n");  
        p = prim;  
        do {  
            printf("%d\n", p->cod);  
            p = p->next;  
        } while (p !=NULL);  
    }  
}
```

# Cozi cu priorități

- O coadă cu priorități reprezintă o structură de date mai specializată decât o stivă sau o coadă
- La fel ca o coadă, coada cu priorități are o parte din față și o parte din spate, iar elementele sunt șterse din față

# Cozi cu priorități

- Într-o coadă cu priorități, elementele sunt ordonate după valoarea unei chei, astfel încât elementul cu cheia minimă (sau, în unele implementări, cel cu cheia maximă) se află întotdeauna în față
- Elementele sunt inserate în pozițiile corespunzătoare, astfel încât ordinea să fie menținută



# Aplicații ale cozilor cu priorități

- În sistemele de operare în timp real, programele sunt așezate într-o coadă de priorități, astfel încât programul cu prioritatea cea mai mare este cel care va primi cel dintâi procesorul pentru o cuantă de timp, pentru a se putea executa



# Aplicații ale cozilor cu priorități

- În multe situații, se dorește accesul la elementul cu valoarea cea mai mică, valoarea putând reprezenta cel mai ieftin sau cel mai scurt mod de a efectua o activitate
- Elementul cu cheia cea mai mică are prioritatea cea mai mare
- Vezi demonstrația PriorityQ



# Concluzii

- O coadă permite accesul la primul element inserat
- Operațiile principale asupra cozilor sunt inserarea unui element în spatele cozii și ștergerea elementului din fața cozii
- O coadă cu priorități permite accesul la cel mai mic (sau uneori cel mai mare) dintre elemente

# Concluzii

- Operațiile principale asupra cozilor cu priorități sunt inserarea unui element în ordine sortată și ștergerea elementului cu cheia minimă
- O coadă poate fi implementată utilizând tablouri (array-uri) sau liste înlănțuite