# COADA

## I. INTRODUCERE TEORETICA

Un alt principiu de gestionare a listelor simplu inlantuite este principiul FIFO (First In First Out). Conform acestui principiu, primul element introdus in lista este si primul care este scos din lista.

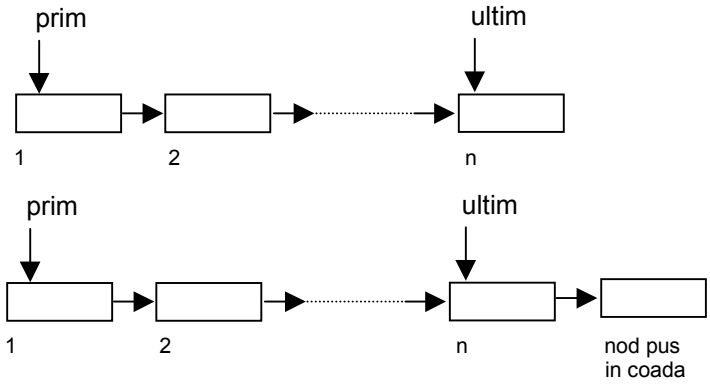
Despre o lista gestionata in acest fel se spune ca formeaza o **coada**. Cele doua capete ale listei simplu inlantuite care implementeaza o coada sunt si capetele cozii.

Asupra cozilor se definesc trei operatii, ca si asupra stivelor:

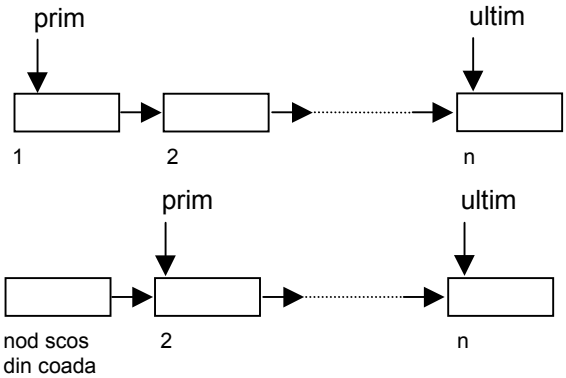
**a. pune un element in coada;**
**b. scoate un element din coada;**
**c. stergerea (vidarea) unei cozi.**

Pentru a respecta principiul FIFO, in DESFASURAREA LUCRARII, vom pune un element in coada folosind functia **adauga** si vom scoate un element din coada folosind functia **spn**. Deci, la un capat al cozii se pun elementele in coada, iar la celalalt capat se scot elementele din coada.

**Operatia de punere in coada**




**Operatia de scoatere din coada**




Cozile definite ca mai sus corespund celor din viata de toate zilele si din aceasta cauza ele se folosesc frecvent in problemele de simulare a fenomenelor reala.

## II. DESFASURAREA LUCRARII

Presupunem o mica filiala a unei banci cu un singur ghiseu, care se deschide la ora 8. Filiala se inchide la ora 16, dar publicul aflat la coada este deservit in continuare. Dupa ora 16 nici o persoana nu mai are voie sa intre in sediul filialei ca sa se aseze la coada la ghiseu. In acest scop se realizeaza o simulare pe calculator a situatiei existente care sa stabileasca o medie a orelor suplimentare efectuate zilnic pe o perioada de o luna.

Pentru a simula acest fenomen se au in vedere operatiile efectuate la ghiseu, dupa cum urmeaza:

| **Operatie** | **Timp de executie:** |
|---|---|
| 1 - depunere | 5 minute |
| 2 - restituire fara confirmare | 7 minute |
| 3 - depunere pe un carnet nou | 10 minute |
| 4 - restituire cu confirmare | 20 minute |
| 5 - lichidare | 25 minute |

Operatiile care se efectueaza nu sunt uniform repartizate. In medie, operatiile 1 si 2 se executa cel mai frecvent, iar operatia 5 cel mai rar. In medie, s-a constatat ca operatiile 3 si 4 se solicita triplu fata de operatia 5, iar operatiile 1 si 2 sunt de 7 ori mai solicitate decat operatia 5.

Avand in vedere acest fapt, consideram o metoda de simulare a operatiilor care se executa la ghiseu bazata pe numere pseudoa-leatoare. Daca luam in seama operatia 5, ca unitate, inseamna ca pentru operatiile 1 si 2 vom alege ponderea 7, iar pentru operatiile 3 si 4 ponderea 3:

| **Operatie** | **Pondere** |
|---|---|
| operatia 1 | 7 |
| operatia 2 | 7 |
| operatia 3 | 3 |
| operatia 4 | 3 |
| operatia 5 | 1 |
| **Total** | **21** |

Se vor genera numere pseudoaleatoare situate in intervalul [1, 21], iar operatia se va defini astfel:
Fie r un numar pseudoaleator din intervalul [1, 21]:
- daca 1 <= r <= 7, atunci se realizeaza operatia 1;
- daca 8 <= r <= 14, atunci se realizeaza operatia 2;
- daca 15 <= r <= 17, atunci se realizeaza operatia 3;
- daca 18 <= r <= 20, atunci se realizeaza operatia 4;
- daca r = 21, atunci se realizeaza operatia 5.

Numerele pseudoaleatoare pot fi generate folosind functiile **random** si **srand** din biblioteca limbajelor C si C++.

Functia *random* are prototipul:

**int random(int n);**

Ea returneaza un numar pseudoaleator aflat in intervalul [0,n), numar natural mai mic decat *n*.

Numerele pseudoaleatoare se genereaza printr-un proces de calcul iterativ. Acest proces porneste cu o valoare initiala care poate fi definita de programator apeland functia **srand**. Aceasta valoare initiala se numeste samanta sirului de numere pseudoa-leatoare.

Functia *srand* are prototipul:

**void srand(unsigned n);**

unde *n* este valoarea la care se seteaza samanta dupa apelul functiei srand.

Ambele functii au prototipul in fisierul stdlib.h.

Programul de simulare a problemei indicate mai sus construieste o coada de asteptare cu persoanele care sosesc la agentie in intervalul de timp indicat mai sus.

Apeland functia *random* se determina operatia solicitata de fiecare persoana aflata la coada. Se scot elementele din coada respectand principiul FIFO si procesul continua pana la ora 16 cand se sisteaza operatia de adaugare. Se determina timpul necesar pentru realizarea operatiilor solicitate de persoanele aflate la coada.

Ramane de precizat modul in care vin persoanele la agentie. Vom presupune ca intervalul de timp intre doua persoane care vin la agentie este aleator si ca acesta este de maximum 15 minute. In acest caz se poate apela functia random cu parametrul 15, iar valoarea:

**random(15) + 1**

va reprezenta intervalul la care soseste persoana urmatoare la agentie.

Programul va contine o variabila globala *timpcrt* a carei valoare este numarul minutelor scurse de la ora 8 si pana in momentul in care a sosit ultima persoana la agentie.

Avem 8 ore de lucru la ghiseu, deci:

**timpcrt <= 80 * 60 = 480 minute**

O alta variabila care numara minutele rezultate din deservirea persoanelor care s-au aflat la coada la ghiseu este *timpghiseu*. Valorile acestor variabile satisfac relatia:

**timpghiseu <= timpcrt**

Cand o persoana se aseaza la coada, atunci **timpcrt** se mareste cu timpul dat de expresia:

**random(15) + 1**

Cand o persoana se scoate din coada (a fost deservita), **timpghiseu** se mareste cu timpul necesar pentru operatia solicitata de persoana respectiva, operatie care se defineste cu ajutorul expresiei:

**random(21) + 1**

Deoarece *timpcrt* defineste timpul curent (numarul de minute care s-au scurs incepand cu ora 8 si pana in momentul in care ultima persoana s-a asezat la coada), rezulta ca persoana din fata poate fi deservita in doua moduri:

- Inainte de inchiderea filialei
(daca **timpghiseu + timp necesar pentru operatia ei <= timpcrt**, altfel se adauga o noua persoana la coada daca **timpcrt <= 480**);

- Dupa inchiderea filialei
(daca **timpcrt > 480**).

**Observatie:** Fiecare punct al DESFASURARII LUCRARII va fi scris intr-un fisier separat cu extensia **.cpp**

In fisierul programului principal va fi definita structura:

```
typedef struct tnod
        {
         int timpoperatie;
         struct tnod *urm;
        } TNOD;
```

1. Sa se scrie o functie:

**int incnod(TNOD *p)**

care incarca variabila **timpoperatie** din structura nodului curent cu o valoare r stabilita ca mai sus, in descrierea programului (se determina aleator operatia si se pastreaza in nod timpul necesar operatiei respective). Functia incarca nodul curent numai daca timpul curent nu depaseste pe cel admis, returnand valoarea 1; altfel returneaza valoarea -1.

O posibila solutie:

```
int incnod(TNOD *p)
/* - incarca nodul curent daca timpul curent nu depaseste pe cel admis si returneaza 1;
   - altfel returneaza -1. */
{
 extern int timpcrt;
 int r;

 /* determina timpul, in minute, la care soseste persoana la agentie */
 timpcrt += random(INTERVAL) + 1;
 if(timpcrt > MAXTIMP)  return -1; /*agentie inchisa */

 /* determina operatia si pastreaza in nod timpul necesar operatiei respective */
 r = random(21) + 1;
 if(r <= 7)   r = 5;
 else if(r <= 14)   r = 7;
      else  if(r <= 17)   r = 10;
                else  if(r <= 20)   r = 20;
                          else r = 25;
 p -> timpoperatie = r;
 return 1;
} /* sfarsit incnod */
```

2. Sa se scrie o functie:

**void elibnod(TNOD *p)**

care elibereaza nodul din lista (coada) spre care pointeaza un pointer p.

3. Sa se scrie o functie:

**TNOD *adauga()**

care adauga un nod in coada si returneaza pointerul spre nodul adaugat sau zero daca nu s-a realizat adaugarea. Daca este necesar, se vor apela si functiile definite la punctele anterioare.

O posibila solutie:

```
TNOD *adauga()
/* - adauga un nod la o lista simplu inlantuita;
   - returneaza pointerul spre nodul adaugat sau zero daca nu s-a realizat adaugarea. */
{
 extern TNOD *prim, *ultim;
 TNOD *p;
 int n;

 n = sizeof(TNOD);
 if(((p = (TNOD *)malloc(n)) != 0) && (incnod(p) == 1))
   {
    if(prim == 0)
          prim = ultim = p;
    else
```

```
            {
             ultim -> urm=p;
             ultim = p;
             }
    p -> urm=0;
    return p;
   }
 if(p == 0)
   {
    printf("memorie insuficienta\n");
    exit(1);
   }
 elibnod(p);
 return 0;
}
```

4. sa se scrie o functie

**void spn()**

care sterge primul nod al listei (cozii).

O posibila solutie:

```
void spn() /* sterge primul nod al listei (cozii) */
{
 extern TNOD *prim, *ultim;
 TNOD *p;

 if(prim == 0)
            return;
 p = prim;
 prim = prim -> urm;
 elibnod(p);
 if(prim == 0)
            ultim = 0;
} /* sfarsit spn */
```

5. Sa se deschida un fisier in care va fi introdus programul al carui scop a fost definit la inceputul DESFASURARII LUCRARII. Programul creeaza o coada apeland functia **adauga**. Se va introduce modelul structurii TNOD (a se vedea mai sus) si se vor declara variabilele globale **prim** si **ultim** ca pointeri de tip TNOD.

Deoarece numarul de zile lucratoare variaza de la o luna la alta, o variabila z va fi initializata de la tastatura cu o valoare reprezentand acest numar de zile. Pogramul va afisa atat numarul de minute suplimentare facute zilnic cat si numarul total de minute (sau ore) cumulate la sfarsitul lunii.

RASPUNSURI

2. ELIBNOD.CPP

```
void elibnod(TNOD *p)  /* elibereaza zona ocupata de  nodul listei spre care pointeaza p */
{
 free(p);
} /* sfarsit elibnod */
```

5. LAB_TP3.CPP

```
# include <stdio.h>
# include <stdlib.h>
# include <alloc.h>
# include <conio.h>
# include <math.h>

typedef struct tnod
                      {
                       int timpoperatie;
                       struct tnod *urm;
                      } TNOD;

# define MAXTIMP 480
# define INTERVAL 15

# include "incnod.cpp"
# include "elibnod.cpp"
# include "adauga.cpp"
# include "spn.cpp"

TNOD *prim, *ultim;
int timpcrt;

void main() /* simuleaza cozile de la o banca pe o perioada de un an */
{
 int timpghiseu;
 int i, z;
 double t = 0.0;

 clrscr();

 /* se lucreaza si sambata => z este aproximativ 26 */
 printf("\nNumarul de zile lucratoare ale lunii ?  ");
 scanf("%d",&z);
 for(i = 1; i <= z; i++)
    {
   //  srand(i*10); /* seteaza samanta sirului pseudoaleator */

     /* initializari la deschiderea agentiei */
     timpcrt = timpghiseu = 0;
     prim = ultim = 0; /* coada este vida */

     /* ultima persoana sosita se pune la coada */
     while(adauga())
       /* - se deservesc persoanele aflate la coada;
```

```
            - trebuie ca:
                    1.  prim != 0  (altfel nu exista coada);
                    2.  timpghiseu <= timpcrt  */
           while(prim !=0 && timpghiseu <= timpcrt)
            {
             /* se deserveste prima persoana din coada */
             timpghiseu += prim -> timpoperatie;
             spn(); /* se elimina din coada persoana deservita */
            }

    /* - se inchide agentia;
          - se deservesc in continuare persoanele
                    aflate la coada la ghiseu. */
    while(prim != 0)
     {
          timpghiseu += prim -> timpoperatie;
          spn();
     }

    /* se afiseaza timpul suplimentar in minute */
   if((timpghiseu - 480) > 0)
   {
    printf("\nziua %d -", i);
    printf("timp peste ora 16: %d minute", timpghiseu - 480);

    t = t + timpghiseu - 480; /* se aduna la timpul total */

    /* vizualizam un ecran de date */
    if((i % 22) == 0)
           {
            printf("\nActionati o tasta pentru a continua!");
            getch();
           }
    }
   else  printf("\nziua %d - timp peste ora 16: 0 minute", i);
   } /* sfarsit for */
   getch();
   printf("\n\n SFASIT DE LUNA\n");
   printf("\nTimpul total cumulat pe luna este: %g minute", t);
   printf("\nsau: %g ore\n", floor(t/60));
   getch();
} /* sfarsit main */
```