Exemple de probleme cu stive si cozi implementate dinamic

Cuprins:

1. Exemple de probleme cu stive si cozi dinamice

Problema 1

Dacă la intrare se dă un șir de numere naturale, să se afișeze la ieșire grupuri de câte trei elemente inversate între ele. Elementele de la intrare se citesc pe o singură linie, elementele de la ieșire se afișează pe o singură linie. Dacă șirul de la intrare nu are un număr întreg de grupuri de câte trei elemente, atunci ultimul grup se va afișa neinversat. Introducerea unui 0 (care nu se va afișa la ieșire) la intrare va termina secvența introdusă.

Exemplu:

Intrare
1
2
3
4
5
0

Ieșire
3
2
1
4
5

Soluție propusă – cu explicații:

Se va crea o listă în care se vor insera (la sfârșit) elementele de la intrare. Apoi se va parcurge lista și se vor pune elementele într-o stivă. După introducerea a trei elemente în stivă, aceasta se va descărca la ieșire. Elementele rămase în listă se vor scoate la ieșire separate. În figura 1.1 sunt ilustrate lista și stiva. Organigrama asociată este prezentată în figura 1.2. Figurile 1.3, 1.4 și 1.5 explică modul de operare cu lista și stiva.

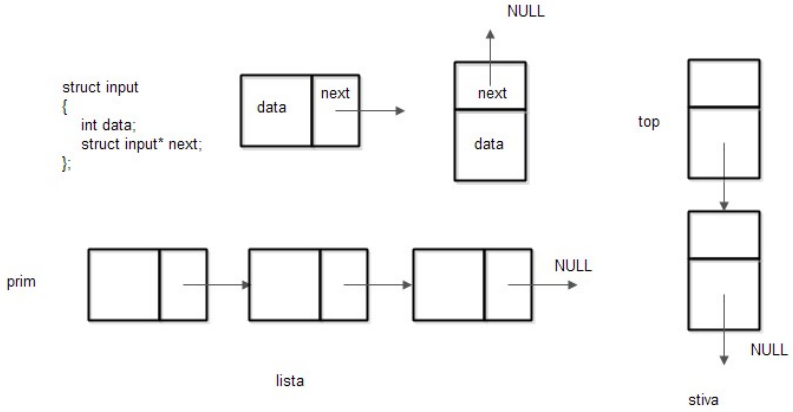


Figura 1.1. Lista și stiva

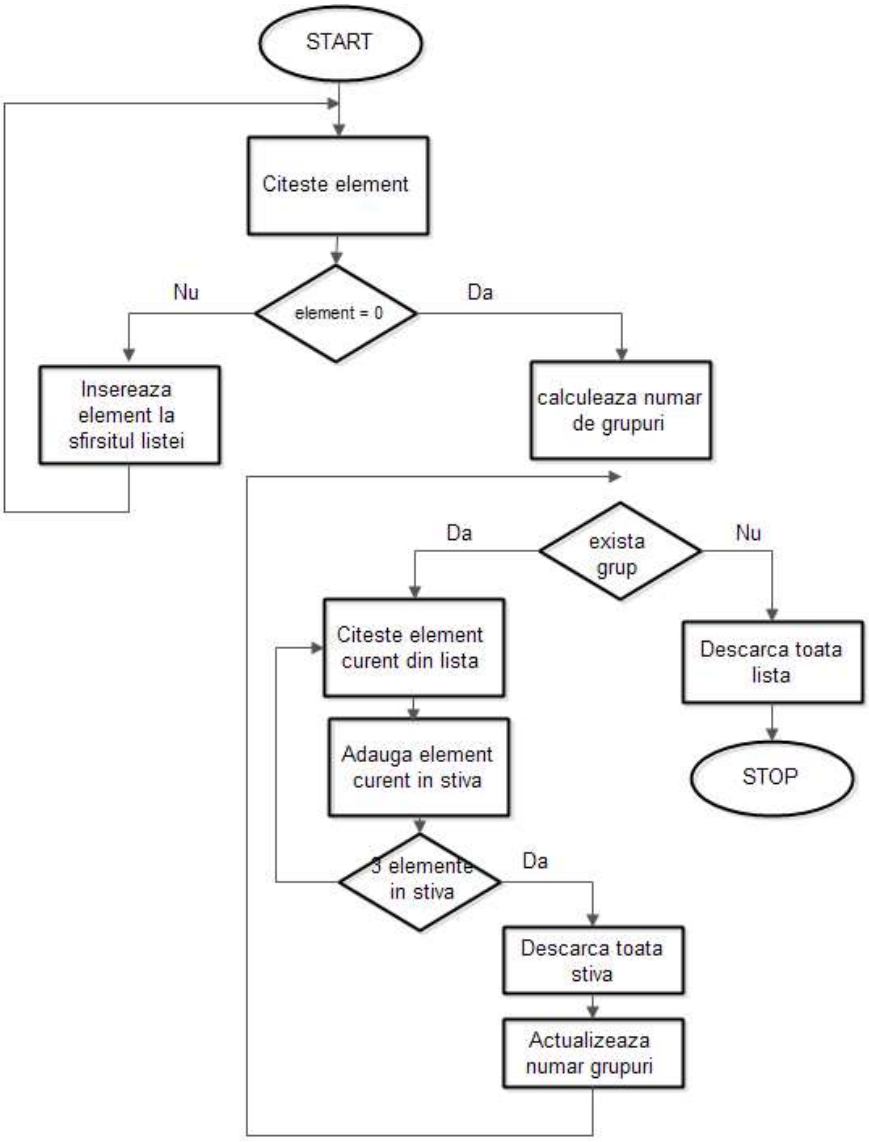
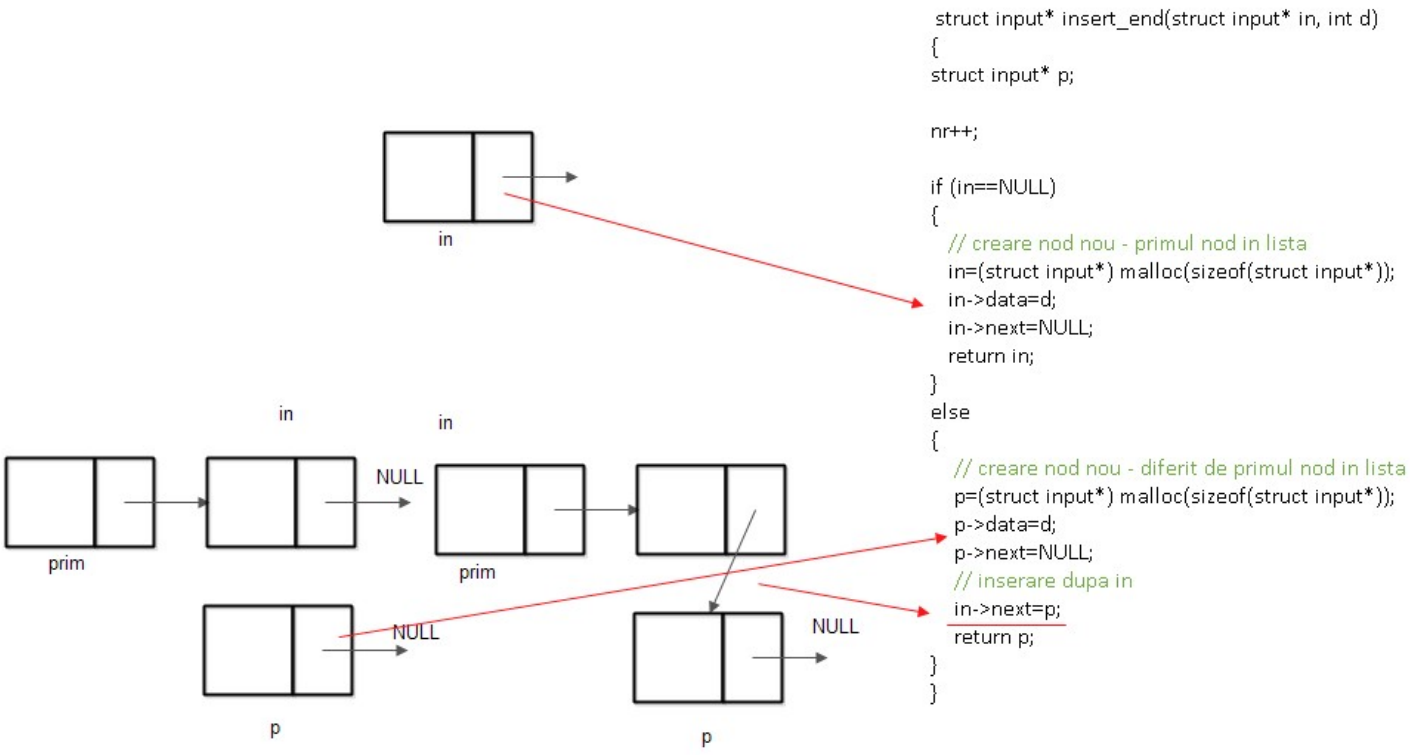


Figura 1.2. Organigrama programului

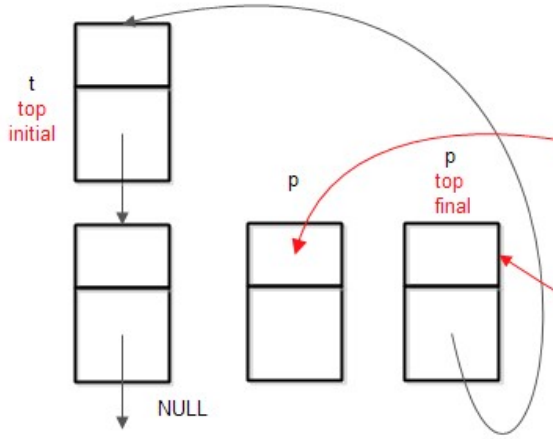



```
struct input* insert_end(struct input* in, int d)
{
struct input* p;

nr++;

if (in==NULL)
{
  // creare nod nou - primul nod in lista
  in=(struct input*) malloc(sizeof(struct input*));
  in->data=d;
  in->next=NULL;
  return in;
}
else
{
   // creare nod nou - diferit de primul nod in lista
   p=(struct input*) malloc(sizeof(struct input*));
   p->data=d;
   p->next=NULL;
   // inserare dupa in
   in->next=p;
   return p;
}
}
```

Figura 1.3. Inserarea la sfârșitul listei




```
struct input* add_stack(struct input* t, int d)
{
  struct input* p;

// stiva vida - se creaza primul element
  if(t==NULL)
  {
     t=(struct input*) malloc(sizeof(struct input*));
     t->data=d;
     t->next=NULL;
  }
  else
// stiva nevida - se insereza inainte de top
  {
     p=(struct input*) malloc(sizeof(struct input*));
     p->data=d;
     p->next=t;
     t=p;
  }
return t;
}
```

Figura 1.4. Inserarea în stivă

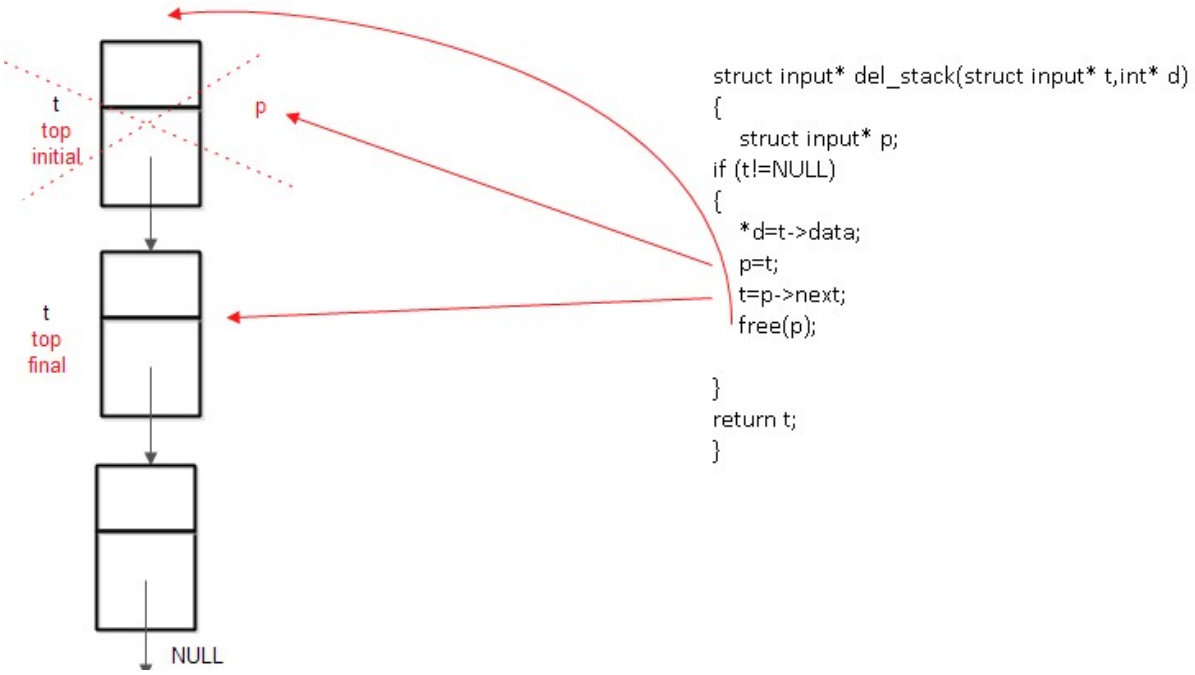


Figura 1.5. Extragerea din stivă

Codul propus este următorul:

```
#include <stdio.h>
#include <stdlib.h>

struct input
{
   int data;
   struct input* next;
};

// functie de inserare la sfirșitul listei
struct input* insert_end(struct input* în, int d);
// functie de adaugare în stiva
struct input* add_stack(struct input* t, int d);
// functie de stergere din stiva
struct input* del_stack(struct input* t,int* d);

// numar de elemente la intrare
int nr;

int main() {

   struct input* intrare=NULL;
```

```
  struct input* prim=NULL;
  struct input* top=NULL;
  int a=-1;
  int first=1;

  // contor de elemente în stiva
  int cnt;
  // numar de grupuri
  int nrg;
  int x;

  struct input* p;
  nr=0;
  nrg=0;

  while (a!=0)
  {
     scanf("%d",&a);
     if (a!=0) intrare=insert_end(intrare,a);
    // lista este creata fara nod fals, se va retine pointerul la primul element
     if (first==1) { prim=intrare; first=0;}
  }
  p=prim;
  cnt=0;
  while (nrg < nr/3)
  {
     // inserare în stiva
    top=add_stack(top,p->data);
     cnt++;
    // avans în lista
     p=p->next;
     if (cnt % 3==0) {
        // actualizare numar grup
        nrg++;
        // descarca stiva
        while (top != NULL) {
           top = del_stack(top,&x);
           printf("%d\n",x);
        }
     }
     }

// descarca lista
  while(p!=NULL)
  {
     printf("%d\n",p->data);
```

```
      p=p->next;
   }
   return 0;
}

// insereaza element la sfirșitul listei
// intrare:  pointer la nodul dupa care se insereaza, informatia noua
// intoarce pointer la ultimul nod din lista

struct input* insert_end(struct input* în, int d)
{
struct input* p;

nr++;

if (în==NULL)
{
  // creare nod nou – primul nod în lista
  în=(struct input*) malloc(șizeof(struct input*));
  în->data=d;
  în->next=NULL;
  return în;
}
else
{
   // creare nod nou – diferit de primul nod în lista
  // cautare sfirșit lista (nu e cazul, functia va fi apelata cu pointerul dupa care se va insera)
  //while(în->next!=NULL) în = în->next;
  // creare nod nou
   p=(struct input*) malloc(șizeof(struct input*));
   p->data=d;
   p->next=NULL;
   // inserare dupa în
   în->next=p;
   return p;
}
}

// adaugare în stiva
// intrare: virful stivei, informatia noului nod
// ieșire: virful stivei (nou)

struct input* add_stack(struct input* t, int d)
{
  struct input* p;
```

```
// stiva vida – se creaza primul element
  if(t==NULL)
  {
     t=(struct input*) malloc(șizeof(struct input*));
     t->data=d;
     t->next=NULL;
  }
  else
// stiva nevida – se insereza inainte de top
  {
     p=(struct input*) malloc(șizeof(struct input*));
     p->data=d;
     p->next=t;
     t=p;
  }
return t;
}

// extragere din stiva
// intrare: virful stivei
// ieșire: virful stivei (nou), informatia din nod

struct input* del_stack(struct input* t,int* d)
{
   struct input* p;
if (t!=NULL)
{
   *d=t->data;
   p=t;
   t=p->next;
   free(p);

}
return t;
}
```

Problema 2

Să se realizeze un program care ține evidența unor procese (identificate prin numere naturale strict pozitive) organizate pe trei niveluri de prioritate (1 – prioritate maximă, 2, 3- prioritate minimă). Programul citește de la tastatură pe linii separate procesul și prioritatea asociată acestuia. Programul va afișa procesele în ordinea priorității și a intrării. Introducerea unui 0 ca identificator de proces (care nu se va afișa la ieșire) la intrare va termina secvența introdusă.

Exemplu:

Intrare:
1
1
3
2
5
1
4
3
0

Ieșire:
1
5
3
4

Soluție propusă – cu explicații:

Se vor crea 3 cozi (câte una pentru fiecare prioritate) folosindu-se două noduri false ca în figura 2.1.

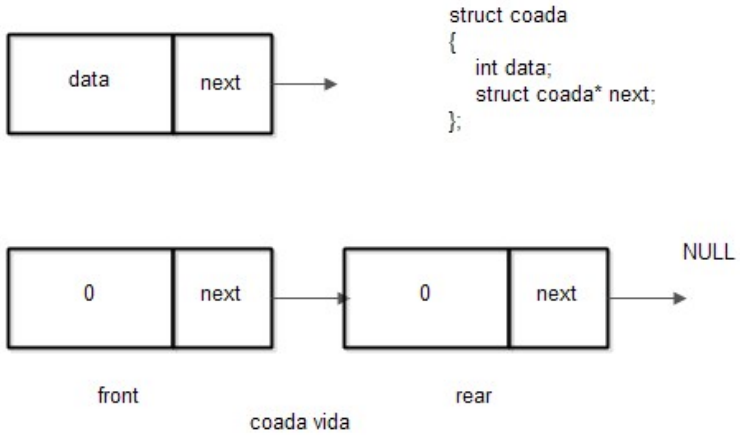


Figura 2.1. Crearea unei cozi cu două noduri false

Se citesc de la tastatura perechi (proces, prioritate) și se adaugă procesele în cozile de prioritate asociate.

Organigrama programului este ilustrată în figura 2.2.

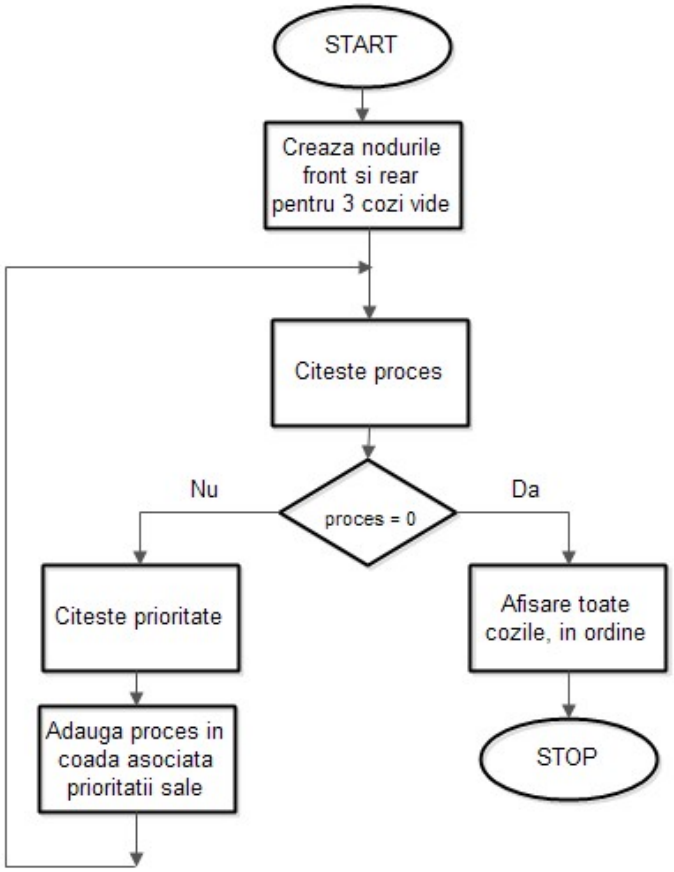


Figura 2.2. Organigrama programului

Funcțiile de adăugare în coadă și de ștergere din coadă sunt prezentate în figurile 2.3 și 2.4.

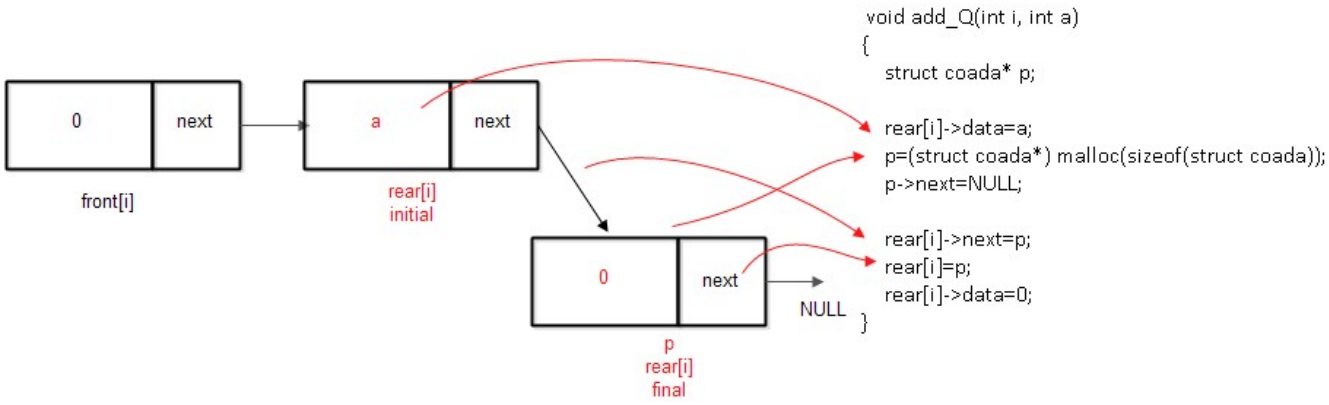


Figura 2.3. Adăugare în coadă (înainte de rear)

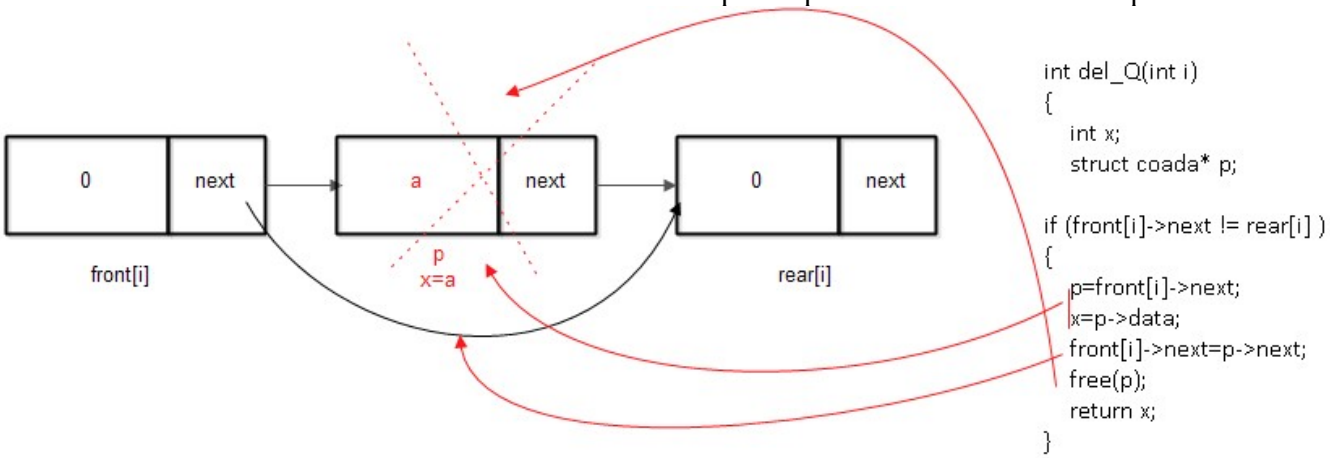


Figura 2.4. Ștergere din coadă (după front)

Codul propus este următorul:

```
#include <stdio.h>
#include <stdlib.h>

struct coada
{
   int data;
   struct coada* next;
};

void add_Q(int i, int a);
int del_Q(int i);

// trei cozi
struct coada* front[3];
struct coada* rear[3];


int main() {

int i;
int proc = -1;
int prio = 0;
int a;

// crearea cozilor de prioritate – vide


for (i=0; i<3; i++)
{
```

```
   front[i]= (struct coada*)malloc(sizeof(struct coada*));
   rear[i]= (struct coada*)malloc(sizeof(struct coada*));

   front[i]->next=rear[i];
   rear[i]->next=NULL;
   front[i]->data=0;
   rear[i]->data=0;
}
   while (proc != 0)
   {
      scanf("%d",&proc);
      if (proc!=0)
      {
         scanf("%d",&prio);
          // adaugare în coada prio-1 (prioritatile se numara de la 1)
         add_Q(prio-1,proc);
      }
   }

   for (i=0; i<3; i++)
   {
      while(front[i]->next != rear[i]) // test coada vida
      {
         // extragere din coada i
          a=del_Q(i);
         printf("%d\n",a);
      }
   }

   return 0;
}

void add_Q(int i, int a)
{
   struct coada* p;

   rear[i]->data=a;
   p=(struct coada*) malloc(sizeof(struct coada));
   p->next=NULL;
   rear[i]->next=p;
   rear[i]=p;
   rear[i]->data=0;
}

int del_Q(int i)
{
```

```
   int x;
   struct coada* p;

if (front[i]->next != rear[i] )
{
   p=front[i]->next;
   x=p->data;
   front[i]->next=p->next;
   free(p);
   return x;
}
return 0;
}
```

Problema 3

Să se realizeze un program care primește la intrare (tastatură) un șir de numere întregi (câte un număr pe o linie separată) și afișează la ieșire elementele de forma 3k, 3k+1 și 3k+2 (în aceasta ordine), k natural. Se va păstra ordinea relativă între elemente. Introducerea unui 0 (care nu se va afișa) la intrare va termina secvența introdusă.

Exemplu:

Intrare:

20
4
5
8
15

Ieșire

15
4
20
5
8

Soluție propusă:

Problema se va rezolva similar problemei 2. Se vor defini cozi asociate restului 0, 1 și 2.
Funcțiile de creare a cozilor, de adăugare și extragere din coada rămân aceleași ca la problema 2.
Organigrama asociată problemei 2 se va modifica ca în figura 3.1.

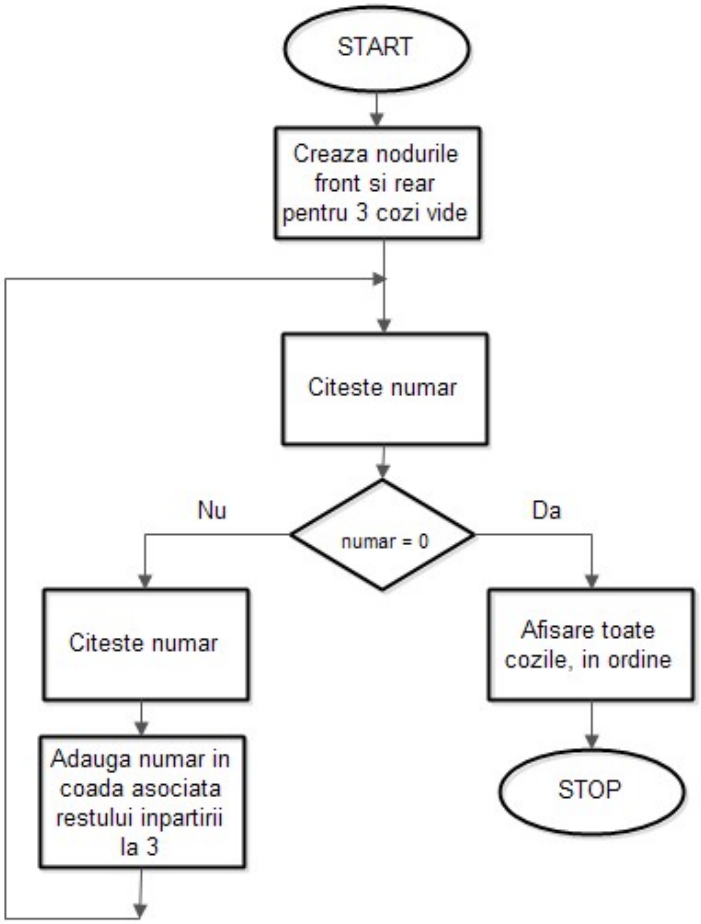


Figura 3.1. Organigrama programului

Codul asociat este următorul:

```
#include <stdio.h>
#include <stdlib.h>

struct coada
{
   int data;
   struct coada* next;
};

void add_Q(int i, int a);
int del_Q(int i);


struct coada* front[3];
struct coada* rear[3];
```

```
int main() {
   int i;
   int în = -1;
   int a;

   for (i=0; i<3; i++)
   {
      front[i]= (struct coada*)malloc(șizeof(struct coada*));
      rear[i]= (struct coada*)malloc(șizeof(struct coada*));

      front[i]->next=rear[i];
      rear[i]->next=NULL;
      front[i]->data=0;
      rear[i]->data=0;
   }

   while (în != 0)
   {
      scanf("%d",&în);
      if (în!=0)
      {
         add_Q(în % 3,în);
      }
   }

   for (i=0; i<3; i++)
   {
      while(front[i]->next != rear[i])
      {
         a=del_Q(i);
         printf("%d\n",a);
      }
   }

   return 0;
}

void add_Q(int i, int a)
{
   struct coada* p;

   rear[i]->data=a;
   p=(struct coada*) malloc(șizeof(struct coada));
   p->next=NULL;
   rear[i]->next=p;
```

```
   rear[i]=p;
   rear[i]->data=0;
}

int del_Q(int i)
{
   int x;
   struct coada* p;

   if (front[i]->next != rear[i] )
   {
      p=front[i]->next;
      x=p->data;
      front[i]->next=p->next;
      free(p);
      return x;
   }
   return 0;
}
```

2. Desfasurarea lucrarii

a) Se va crea un proiect CLion in care se va rula cu depanare codul asociat problemei 1.
b) Se va rula si evalua codul dezvoltat anterior in CLion in mediul VPL
c) Se vor studia organigramele asociate problemelor 1, 2 si 3 si modul de implementare dinamica a stivei si cozii

Tema: Sa se propună alte soluții pentru problemele 1, 2 si 3 si sa se evalueze in VPL soluțiile propuse.