Problema 1

Sa se scrie un program care afiseaza tipurile de produse si produsele dintr-un magazin. Tipurile de produse si produsele sint introduse de la tastatura in forma: TipProdus, Produs (pe linii separate). Pot fi introduse tipuri si produse identice, dar se vor afisa o singura data. Afisarea se va face in ordinea introducerii la intrare. Introducerea unui 0 ca TipProdus (care nu se va afisa la iesire) la intrare va termina secventa introdusa. Numele de tipuri de produse si de produse nu contin spatii si au maxim 100 de caractere.

Exemplu:

Intrare:

```
Electronice
Telefon
Electronice
Calculator
Electrocasnice
Frigider
Altele
Hirtie
Electronice
Tableta
Electronice
Telefon
0
```

Iesire

```
Electronice
Telefon
Calculator
Tableta
Electrocasnice
Frigider
Altele
Hirtie
```

Solutie propusa – cu explicatii:

Problema nu specifica numarul maxim de tipuri de produse sau de produse. Rezolvarea poate folosi o structura de doua liste simplu inlantuite ca in figura 1. Exista un nod fals, *headerTip*, care va fi folosit pentru a lega toate celelelte noduri. Pe verticala se vor lega nodurile care contin tipul de produs, iar pe orizontala se vor lega nodurile care contin produse de acelasi tip. In figura 1 s-a exemplificat cazul a 2 tipuri de produse, TP1 cu produsul P1 si TP2 cu produsele P2 si P3. De asemenea, sint indicate si declaratiile corespunzatoare nodurilor.

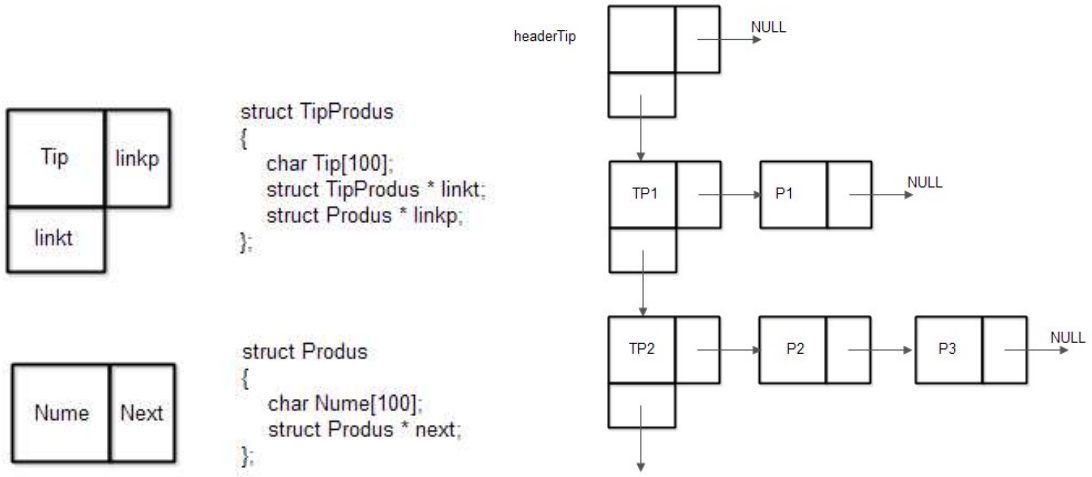
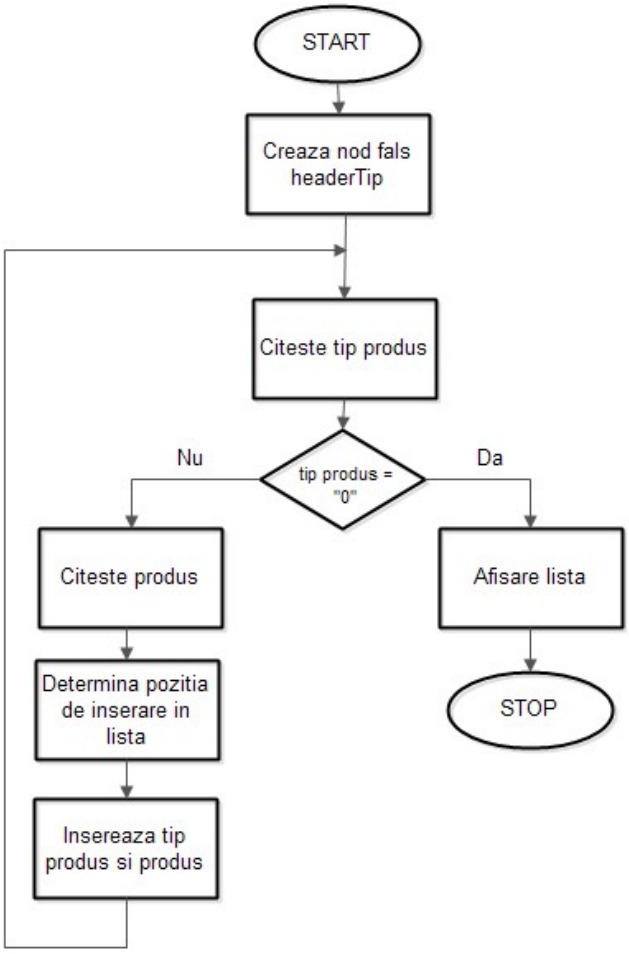


Figura 1. Structura listelor si declaratiile de tip

Organigrama programului este prezentata in figura 2.



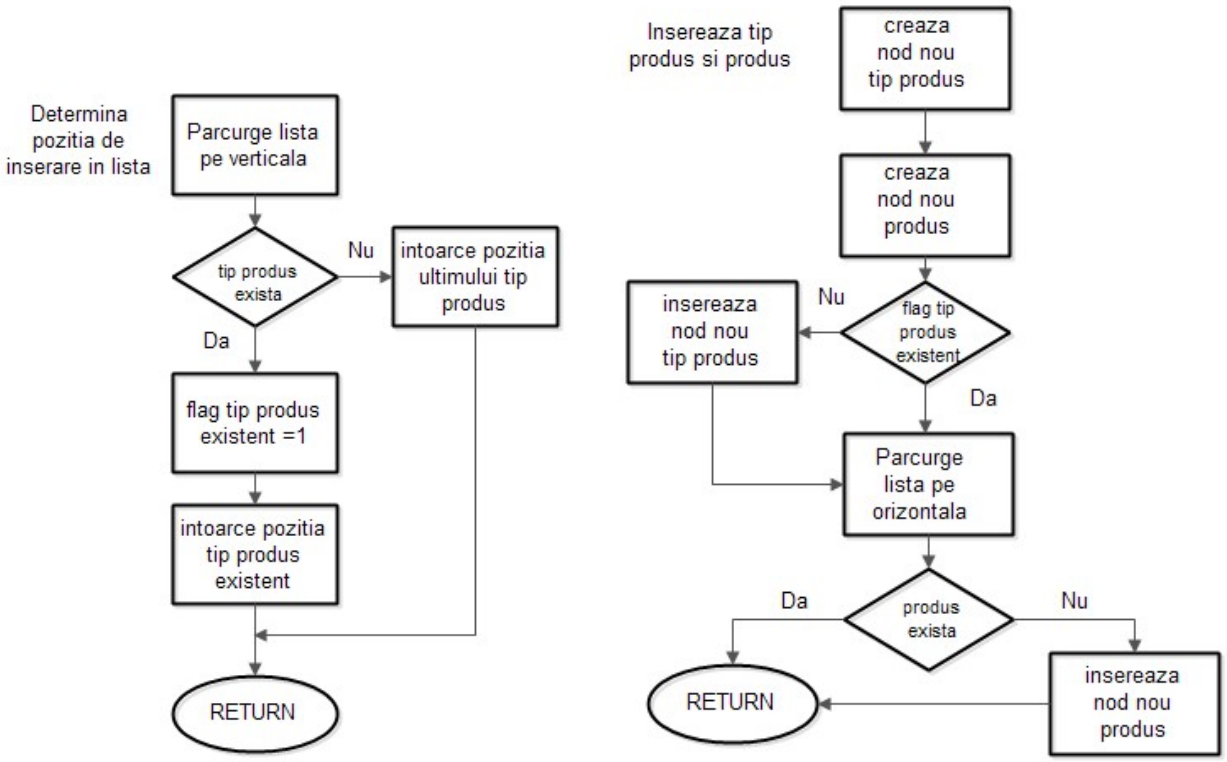


Figura 2 Organigrama programului

Codul asociat este urmatorul:

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

struct TipProdus
{
   char Tip[100];
   struct TipProdus * linkt;
   struct Produs * linkp;
};

struct Produs
{
   char Nume[100];
   struct Produs * next;
};
```

```
// se declara prototipurile functiilor utilizate
struct TipProdus* NewTip(char tipP[]);
struct Produs* NewProd(char P[]);

struct TipProdus* EndTip(char tipP[], int* res);
void insertTipProd(char tipP[],char P[]);

void Afisare(void);

// nodul fals
struct TipProdus *headerTip;
// variabila result indica daca tipul de produs exista deja
int result;

int main()
{
// creare nod fals
headerTip=NewTip("");
// tablouri pentru numele tip produs si produs
char Tip[100];
char Prod[100];

while (1)
{
   // citeste tip produs si produs
   scanf("%s",Tip);
   if (strcmp(Tip, "0")==0) break;
   scanf("%s",Prod);
   // cauta pozitiile in liste si insereaza daca e cazul
   insertTipProd(Tip,Prod);
}
   // afisare liste
   Afisare();

   return 0;
}

// crearea nod nou tip produs
// parametru de intrare numele tipului de produs
// intoarce un pointer la nodul nou

struct TipProdus* NewTip(char tipP[])
{
  struct TipProdus * t;
```

```
   t =(struct TipProdus *) malloc(sizeof(struct TipProdus));
   strcpy(t->Tip, tipP);
   t->linkp=NULL;
   t->linkt=NULL;

   return t;

}

// crearea nod nou produs
// parametru de intrare numele produsului
// intoarce un pointer la nodul nou

struct Produs* NewProd(char P[])
{
   struct Produs * t;

   t =(struct Produs *) malloc(sizeof(struct Produs));
   strcpy(t->Nume, P);
   t->next=NULL;

   return t;
}

// cauta pozitia in lista de tipuri de produse
// parametru de intrare numele tipului de produs
// intoarce un pointer la ultimul nod din lista de tipuri de produse si res = 0, daca tipul de produs nu exista
// intoarce un pointer la  nod tipului de produs existent din lista de tipuri de produse si res = 1, daca tipul de
// produs  exista

struct TipProdus* EndTip(char tipP[],int* res)
{
   struct TipProdus *t;
   *res = 0;

   t=headerTip;
   while (t->linkt != NULL)
   {
      t=t->linkt;
      if (strcmp(t->Tip,tipP)==0)
      {
       *res=1;
      break;
      }
   }
```

```
    return t;
}

// insereaza tip nou de produs – la sfirsitul liste de tipuri de produse
// insereaza produs nou – la sfirsitul listei de produse associate tipului de produs
// parametrii de intrare : tip produs, produs

void insertTipProd(char tipP[], char P[])
{
    struct TipProdus *t;
    struct TipProdus *p;
    struct Produs *r;
    struct Produs *h;
    int exista =0;

   // t – pointer la pozitia in lista de tipuri de produse
    t = EndTip(tipP, &result);
   // h – pointer produsul nou
    h = NewProd(P);
    // test existenta tip produs
    if (result == 0)
    {
        p = NewTip(tipP);
        t->linkt = p;
    }
    else p=t;
    // p – pointer la tipul de produse in lista caruia se va aduga (daca e cazul) un nou produs
    r=p->linkp;
    // r – pointer la inceputul listei de produse asociat tipului de produs
    if (r == NULL) p->linkp = h;
    else {
        while (r->next!= NULL)
        {
            if (strcmp(r->Nume,P)==0) {exista=1; break;}
                r = r->next;
        }
       // exista – flag ce indica faptul ca produsul exista
        if (exista==0)
        {
            if (strcmp(r->Nume,P)!=0) r->next = h;
        }
    }
}

// afisare liste – vericala , orizontala
```

```
void Afisare(void)
{
  struct TipProdus* t;
  struct Produs* p;

  t=headerTip->linkt;

  while (t != NULL)
  {
     printf("%s\n",t->Tip);
     p=t->linkp;
     while (p!=NULL)
     {
        printf("%s\n",p->Nume);
        p=p->next;
     }
     t=t->linkt;
  }
}
```

Functia de determinare a pozitiei de inserare in lista de tipuri de produse, *EndTip*, este explicata in figura 3.

Functia are ca parametri tipul de produs si intoarce un pointer la nodul dupa care se va insera noul tip de produse si un flag, *res*, care indica daca tipul de produs exista sau nu in lista de tipuri de produse. Cautare se va face pina la sfirsitul listei.

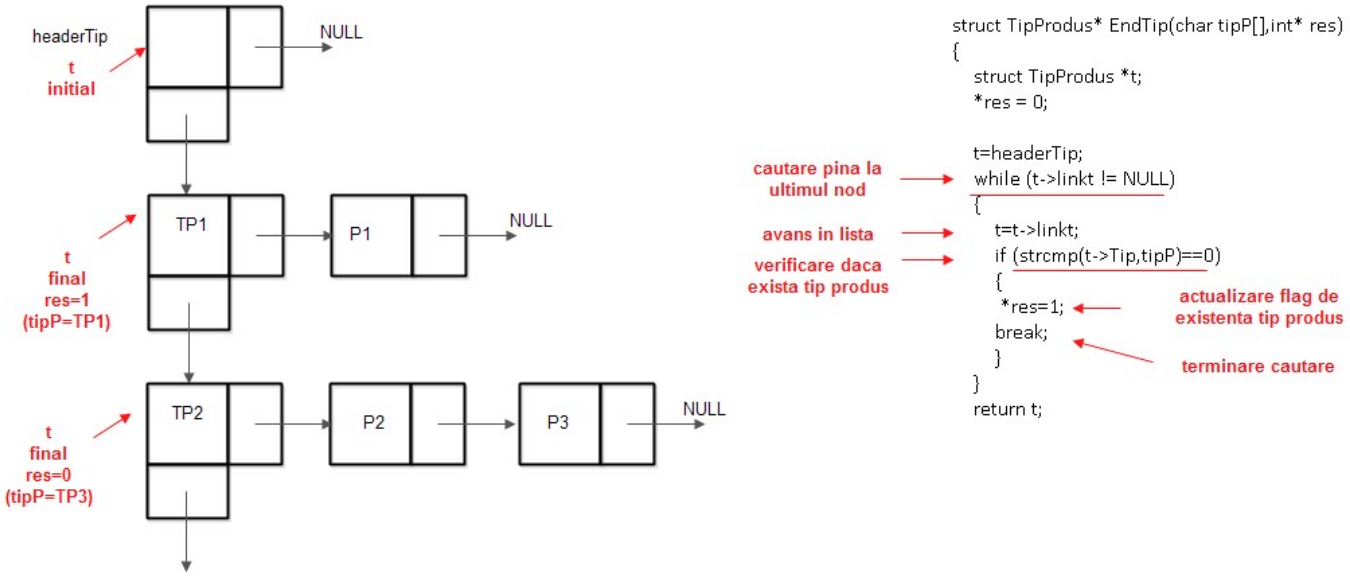


Figura 3. Determinarea pozitiei de inserare in lista de tipuri de produse

In figura 4 se arata modul de functionare a functiei de inserare a tipului de produs si a produsului, *insertTipProd*.

Functia are ca parametri tipul de produs si produsul. Functia apeleaza funcia anterioara, *EndTip* , de determinare a pointerului pentru tipul de produs si verifica daca exista tipul de produs. Daca tipul de produs exista, se cauta in lista de produse asociata produsul. Daca acesta nu exista se va insera produsul nou la sfirsitul listei orizontale, iar daca exista nu se va insera nimic.

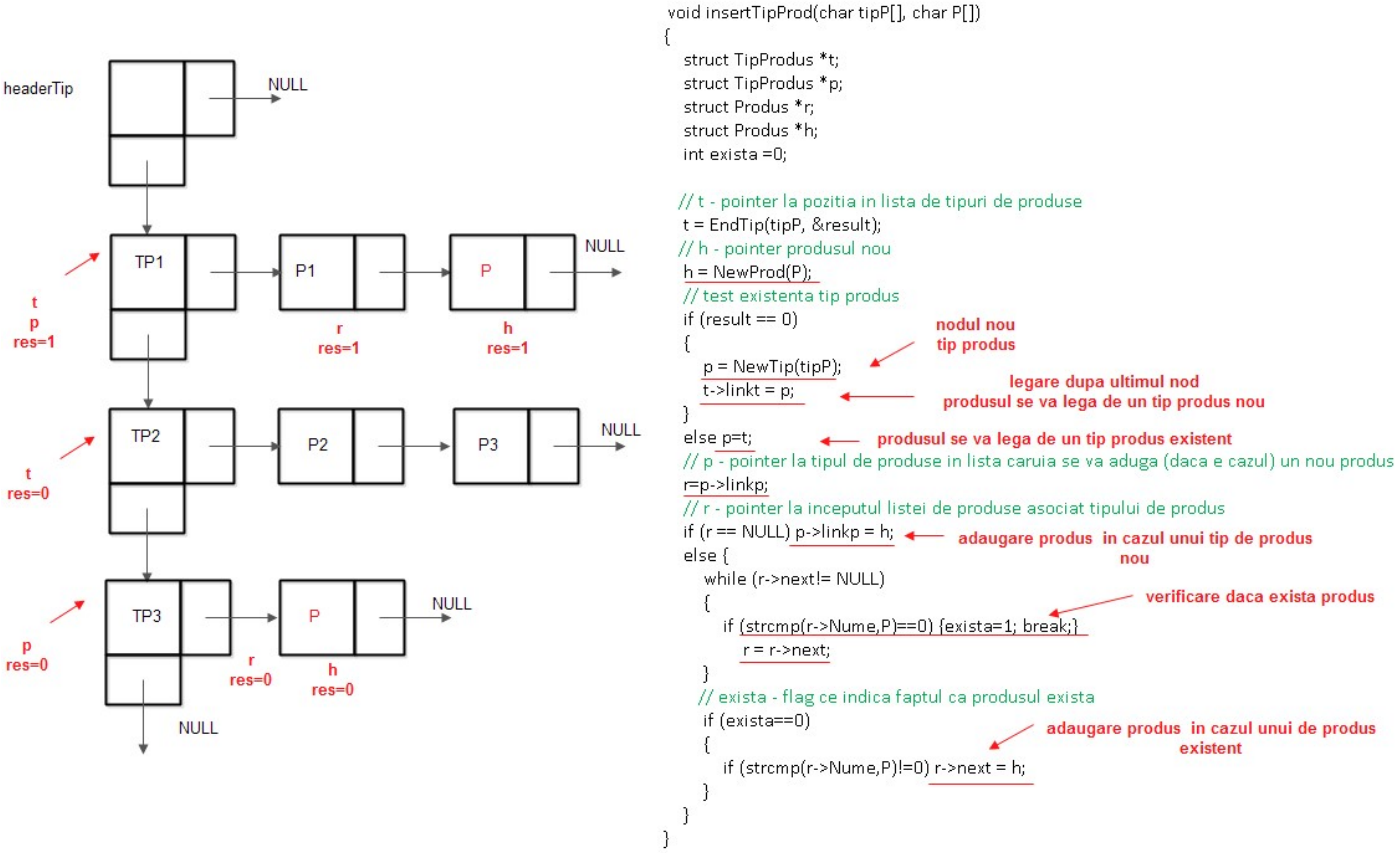


Figura 4. Inserarea tipului de produs si a produsului

Daca tipul de produs nu exista, se va crea si insera in lista verticala un nod nou si se va insera in lista orizontala noul produs.

Problema 2

Sa se scrie un program care afiseaza marcile de masini si modelul acestora dintr-o parcare cu plata.. Marca si modelul sint introduse de la tastatura in forma: Marca, Model (pe linii separate). Pot fi introduce marci si modele identice, dar se vor afisa o singura data. Afisarea se va face in ordine alfabetica. Introducerea unui 0 (care nu se va afisa la iesire) la intrare va termina secventa introdusa.

Exemplu:

Intrare:

Volkswagen
Golf
Volkswagen
Polo
Ford
Mondeo
Skoda
Fabia
Ford
Focus
Ford
Mondeo
Skoda
Octavia
0

Iesire:

Ford
Focus
Mondeo
Skoda
Fabia
Octavia
Volkswagen
Golf
Polo

Solutie propusa – cu explicatii:

Problema se poate rezolva prin modificarea functiilor de inserare din problema 1. Se vor crea liste ordonate. Functia de determinare a pozitiei de inserare in lista de tipuri de produse, *EndTip*, este explicata in figura 1. In figura 2 se arata modul de functionare a functiei de inserare a tipului de produs si a produsului, *insertTipProd*. Pentru functiile de inserare, codul asociat este urmatorul:

```
struct TipProdus* EndTip(char tipP[],int* res)
{
   struct TipProdus *t;
   *res = 0;

   t=headerTip;
   while (t->linkt != NULL)
   {
      if (strcmp(t->linkt->Tip,tipP) < 0)
      {
         t=t->linkt;
      }
      else
      if (strcmp(t->linkt->Tip,tipP)==0)
      {
         t=t->linkt;
         *res=1;
         break;
      }
      else
         break;

   }
   return t;
}

void insertTipProd(char tipP[], char P[])
{
   struct TipProdus *t;
   struct TipProdus *p;
   struct Produs *r, *r1;
   struct Produs *h;
   int exista =0;


   t = EndTip(tipP, &result);
   h = NewProd(P);

   if (result == 0)
   {
      p = NewTip(tipP);
      p->linkt=t->linkt;
      t->linkt = p;
   }
   else p=t;
```

```
  r=p->linkp;
  if (r == NULL) p->linkp = h;
  else {
     while (r!= NULL)
     {
        if (strcmp(r->Nume,P) == 0) {exista=1; break;}
        else
        if (strcmp(r->Nume,P) < 0) { r1=r; r = r->next;}
        else
        { r1=NULL; break;}
     }

     if (exista==0)
     {
        if (r1!=NULL) { h->next=r1->next; r1->next = h;}
        else { h->next=r; p->linkp=h;}
     }
  }
}
```

```
struct TipProdus* EndTip(char tipP[],int* res)
{
   struct TipProdus *t;
   *res = 0;

   t=headerTip;
   while (t->linkt != NULL)
   {
      if (strcmp(t->linkt->Tip,tipP) < 0)
      {
         t=t->linkt;
      }
      else
      if (strcmp(t->linkt->Tip,tipP)==0)
      {
         t=t->linkt;
         *res=1;
         break;
      }
      else
         break;

   }
   return t;
}
```

cautare nod cu info mai mare decit info nodului de inserat

nod tip produs identic: se actualizeaza res si se termina cautarea

termina cautarea, intoarce un pointer dupa care se va insera npul nod tip produs (res = 0) sau un pinter la un nod existent in care se va actualiza lista orizontala (res=1)

Figura 1. Determinarea pozitiei de inserare in lista ordonata de tipuri de produse

```
void insertTipProd(char tipP[], char P[])
{
   struct TipProdus *t;
   struct TipProdus *p;
   struct Produs *r, *r1;
   struct Produs *h;
   int exista =0;


   t = EndTip(tipP, &result);
   h = NewProd(P);

   if (result == 0)
   {
      p = NewTip(tipP);
      p->linkt=t->linkt;
      t->linkt = p;
   }
   else p=t;

   r=p->linkp;
   if (r == NULL) p->linkp = h;
   else {
      while (r!= NULL)
      {
         if (strcmp(r->Nume,P) == 0) {exista=1; break;}
         else
         if (strcmp(r->Nume,P) < 0) { r1=r; r = r->next;}
         else
         { r1=NULL; break;}
      }

      if (exista==0)
      {
         if (r1!=NULL) { h->next=r1->next; r1->next = h;}
         else { h->next=r; p->linkp=h;}
      }
   }
}
```

nod existent

cautare nod cu info mai mare decit info nodului de inserat; r nodul gasit, r1 nodul anterior lui r se va insera dupa r1

primul nod are info mai mare ca nodul de inserat se va insera la inceputul listei orizontale

inserare dupa r1

inserare la inceputul listei orizontale

Figura 2. Inserarea tipului de produs si a produsului

Problema 3

Daca la intrare se da un sir de numere naturale, sa se afiseze la iesire grupuri de cite 3 elemente inversate intre ele. Elementele de la intrare se citesc pe o singura linie, iar elementele de la iesire se afiseaza pe o singura linie. Daca sirul de la intrare nu are un numar intreg de grupuri de cite 3 elemente atunci ultimul grup se va afisa neinversat. Introducerea unui 0 (care nu se va afisa la iesire) la intrare va termina secventa introdusa.

Exemplu:

Intrare

1
2
3
4
5
0

Iesire
3
2
1
4
5

Solutie propusa – cu explicatii:

Se va crea o lista in care se vor insera (la sfirsit) elementele de la intrare. Apoi se va parcurge listasi se vor pune elementele in stiva. Dupa introducerea a 3 elemente in stiva, aceasta se va descarca la iesire. Elementele ramase in lista , se vor scoate la iesire separate. In figura 1 este ilustrata lista si stiva.

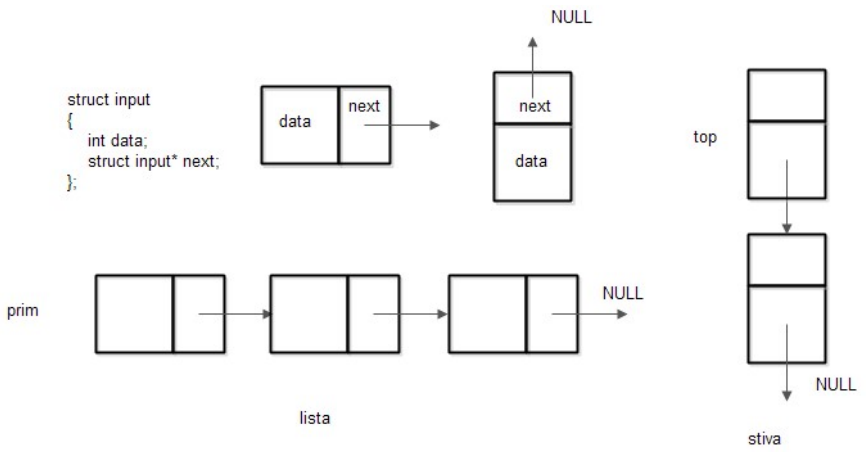


Figura 1. Lista si stiva

Organigrama asociata este prezentata in figura 2.

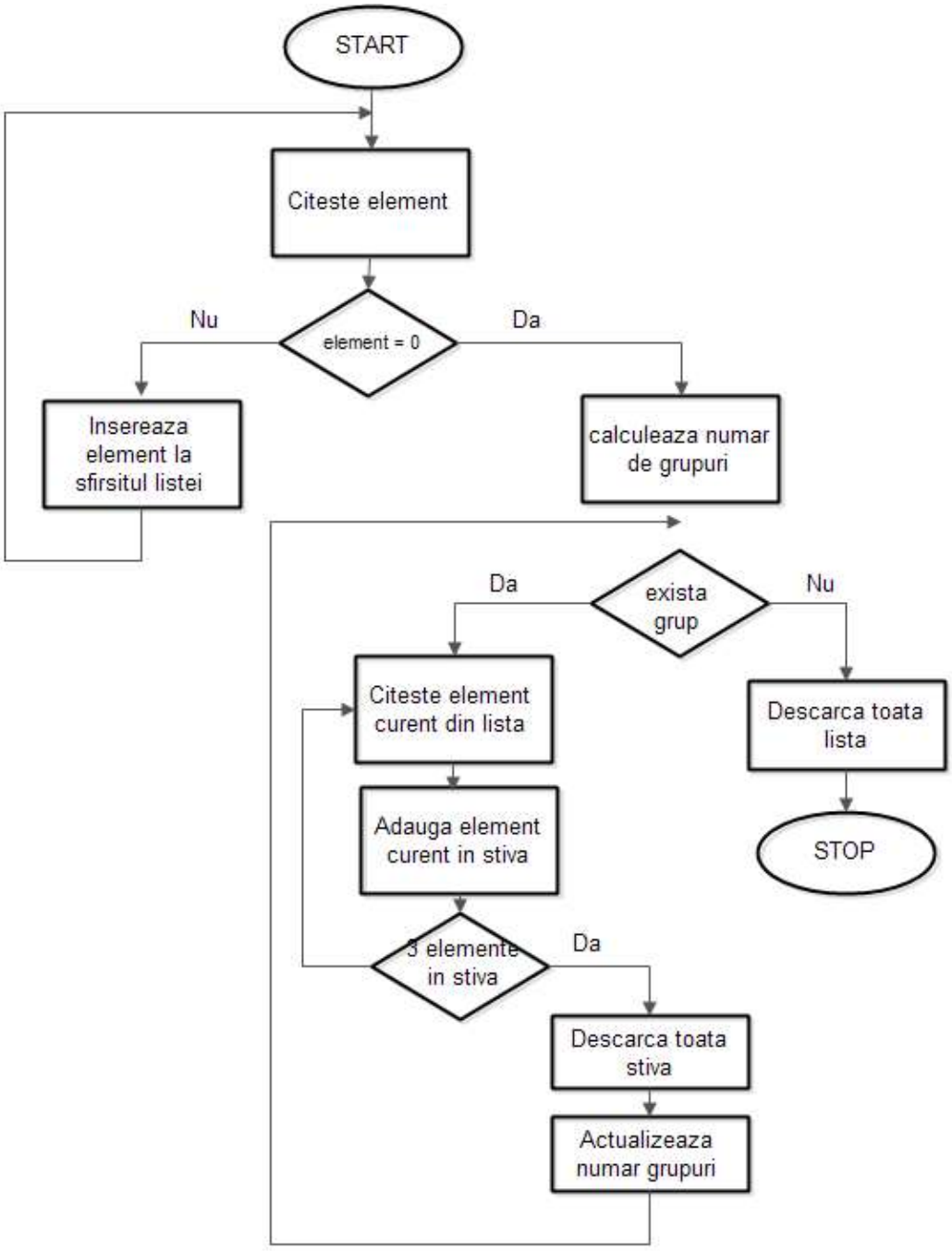
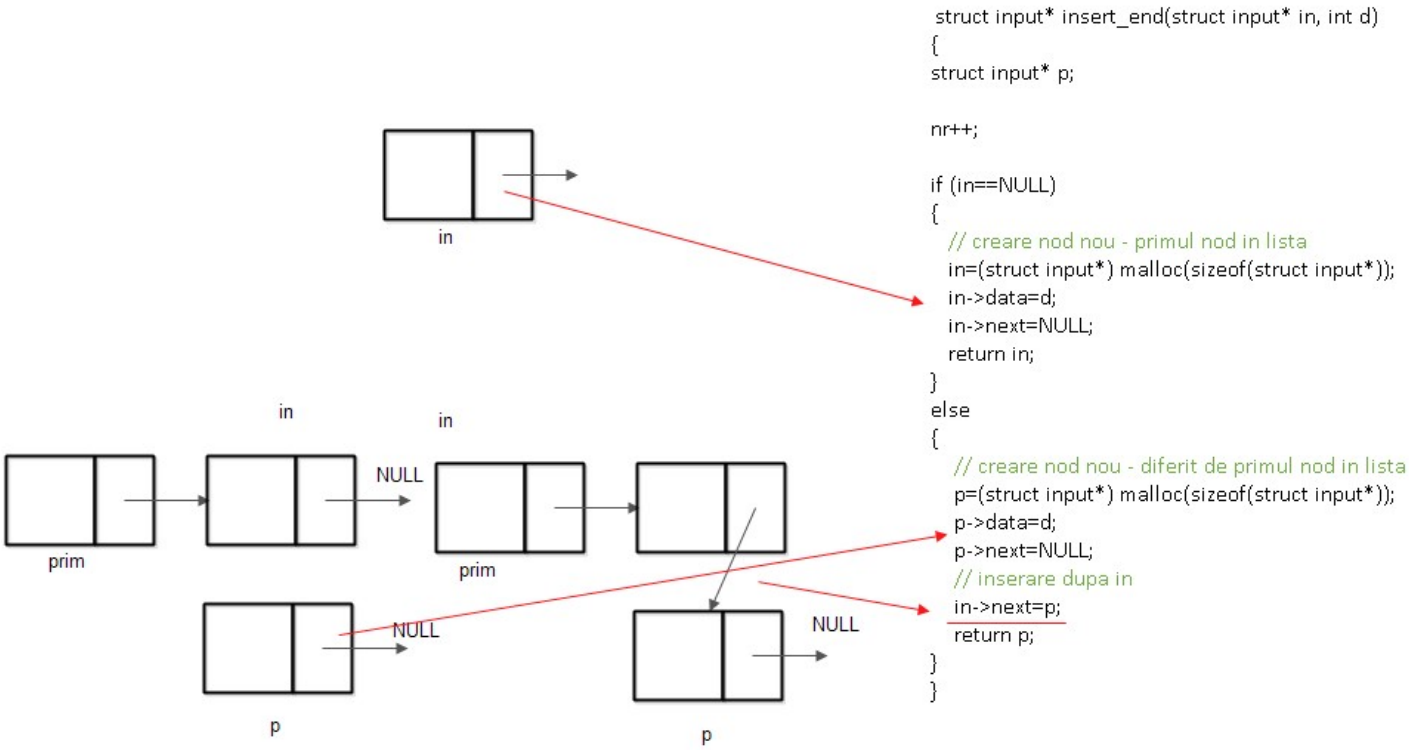


Figura 2. Organigrama programului

Figurile 3, 4 si 5 explica modul de operare cu lista si stiva.

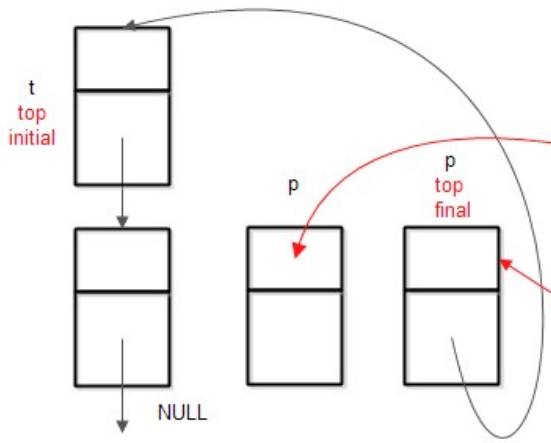



```
struct input* insert_end(struct input* in, int d)
{
struct input* p;

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 3. Inserarea la sfirsitul 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 4. Inserarea in stiva

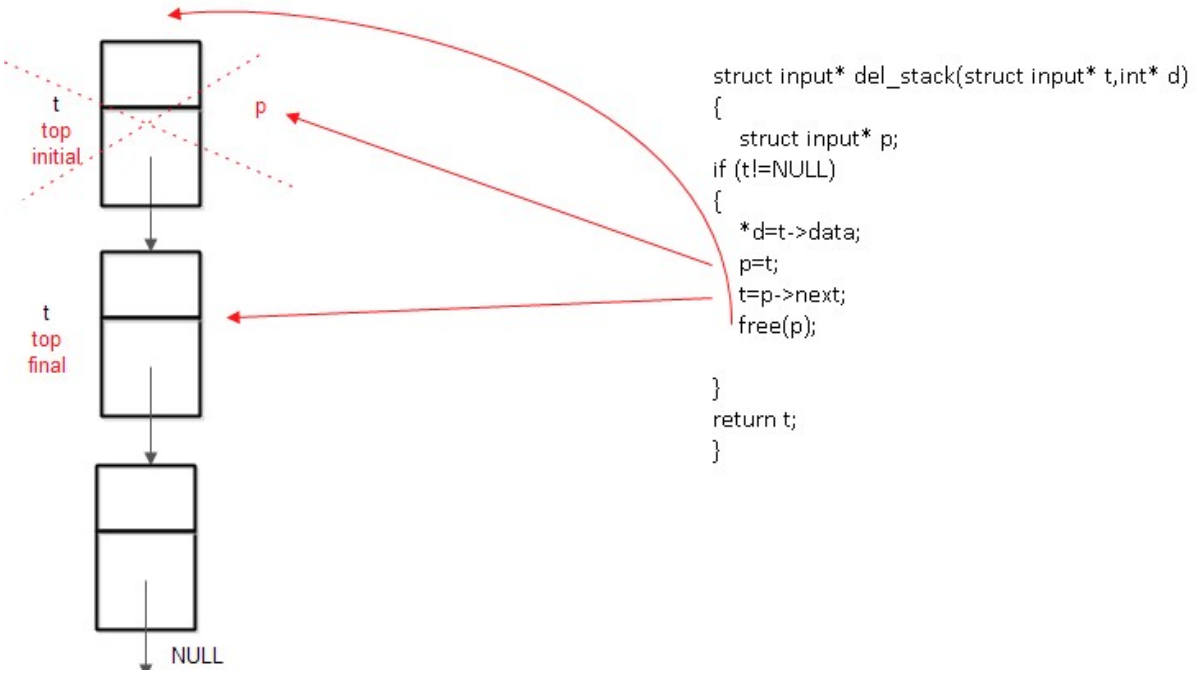


Figura 5. Extragerea din stiva

Codul propus este urmatorul:

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

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


// functie de inserare la sfirsitul listei
struct input* insert_end(struct input* in, int d);
// functie de adaugare in 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 in 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 in stiva
  top=add_stack(top,p->data);
   cnt++;
  // avans in 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 sfirsitul listei
// intrare:  pointer la nodul dupa care se insereaza, informatia noua
// intoarce pointer la ultimul nod din lista

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
  // cautare sfirsit lista (nu e cazul, functia va fi apelata cu pointerul dupa care se va insera)
  //while(in->next!=NULL) in = in->next;
  // creare nod nou
   p=(struct input*) malloc(sizeof(struct input*));
   p->data=d;
   p->next=NULL;
   // inserare dupa in
   in->next=p;
   return p;
}
}

// adaugare in stiva
// intrare: virful stivei, informatia noului nod
// iesire: 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(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;
}

// extragere din stiva
// intrare: virful stivei
// iesire: 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 4

Sa se realizeze un program care tine evidenta unor procese (identificate prin numere naturale strict pozitive) organizate pe 3 niveluri de prioritatate (1 – prioritate maxima 2, 3- prioritate minima). Programul citeste de la tastatura pe linii separate procesul si prioritatea asociata acestuia. Programul va afisa procesele in ordinea prioritatii si a intrarii. Introducerea unui 0 ca identificator de process (care nu se va afisa la iesire) la intrare va termina secventa introdusa.

Exemplu:

Intrare:
1
1
3
2
5
1
4
3
0

Iesire:
1
5
3
4

Solutie propusa – cu explicatii:

Se vor crea 3 cozi (cite una pentru fiecare prioritate) folosindu-se 2 noduri false ca in figura 1.

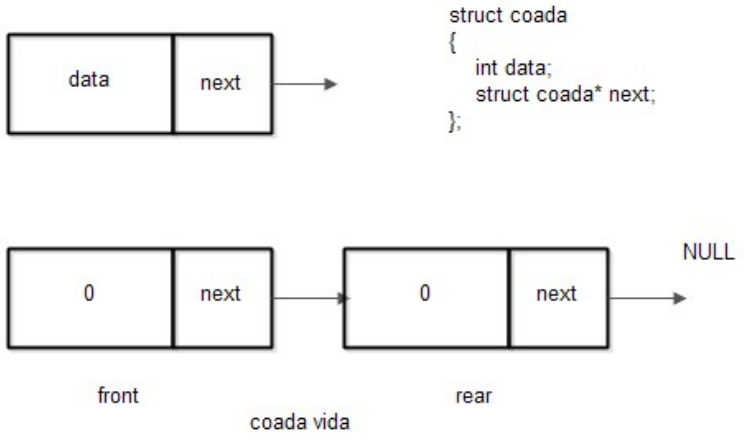


Figura 1. Crearea unei cozi cu doua noduri false

Se citesc de la tastatura perechi (proces, prioritate) si se adauga procesele in cozile de prioritate associate.

Organigrama programului este ilustrata in figura 2.

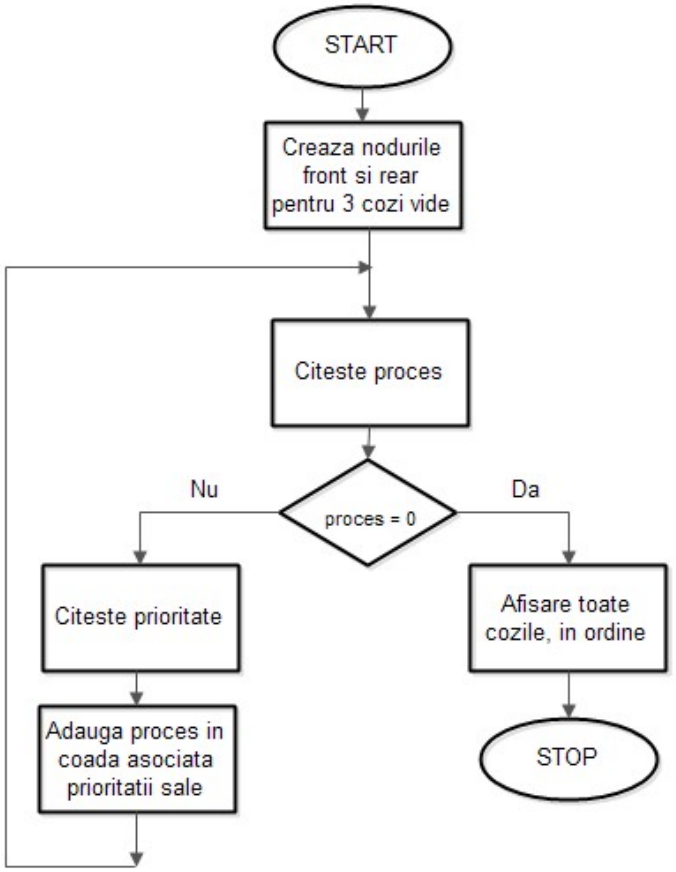


Figura 2. Organigrama programului

Functiile de adugare in coad si de stergere din coada sint prezentate in figurile 3 si 4.

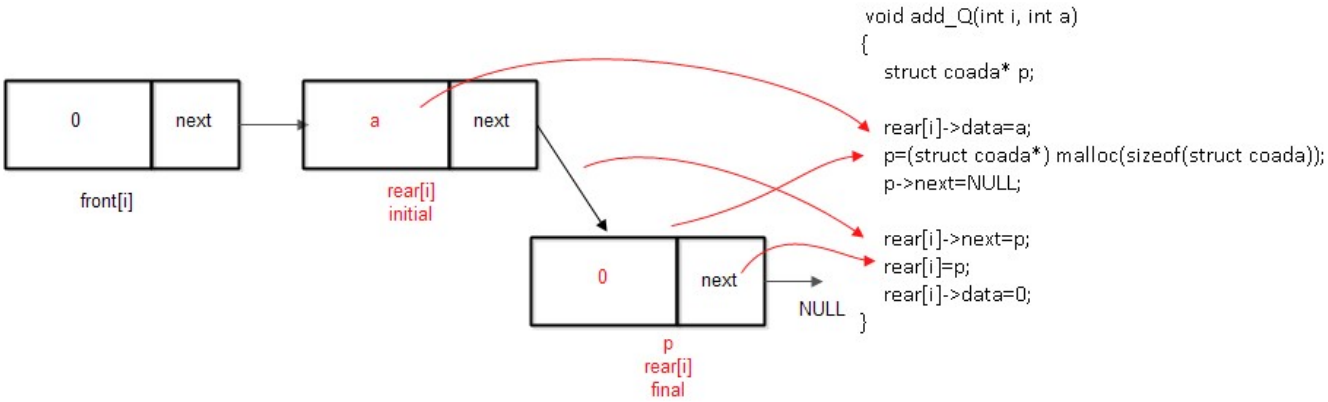


Figura 3. Adaugare in coada (inainte de rear)

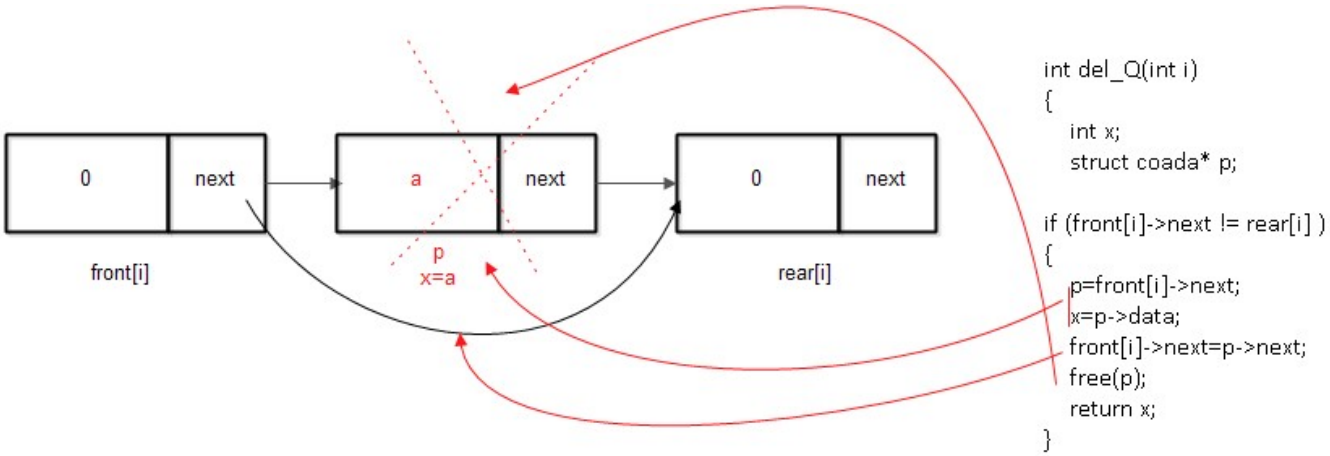


Figura 4. Stergere din coada (dupa front)

Codul propus este urmatorul:

```
#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 in 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 5

Sa se realizeze un program care primeste la intrare (tastatura) un sir de numere intregi (cite un numar pe o linie separata) si afiseaza la iesire elementele de forma 3k, 3k+1 si 3k+2 (in aceasta ordine), k natural. Se va pastra ordinea relativa intre elemente. Introducerea unui 0 (care nu se va afisa) la intrare va termina secventa introdusa.

Exemplu:

Intrare:

```
20
4
5
8
15
0
```

Iesire

```
15
4
20
5
8
```

Solutie propusa:

Problema se va rezolva similar problemei 4. Se vor defini cozi asociate restului 0, 1 si 2.

Functiile de creare a cozilor, de adaugare si extragere din coada ramin aceleasi ca la problema 4.

Organigrama asociata problemei 4 se va modifica ca in figura 1.

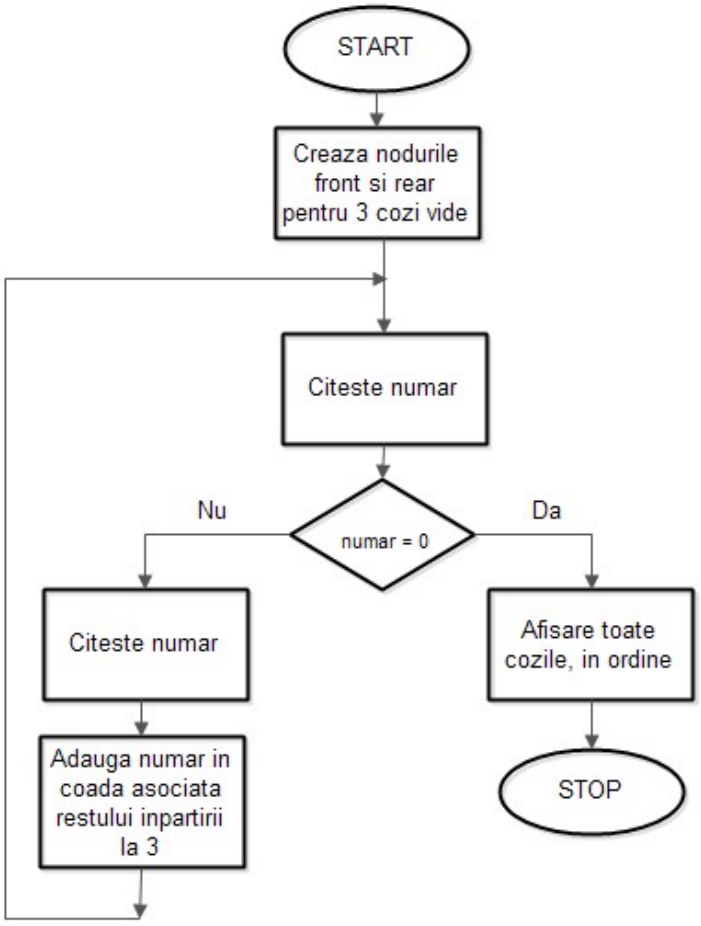


Figura 1. Organigrama programului

Codul asociat este urmatorul:

```
#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 in = -1;
   int a;

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

   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(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 6

Se cere sa se realizeze un program ce realizeaza decodificarea unui text care continue doar vocalele alfabetului. Sirul de biti asociat unui simbol are lungime variabila astfel: A: 0, E: 11, I: 101, O: 1000 si U: 10010.
Sirul de biti este introdus de la tastatura (un bit pe o line separata) si textul decodificat va fi scris pe ecran (cite un caracter pe o linie separata), Introducerea unui 2 (care nu se va afisa) la intrare va termina secventa introdusa.

Exemplu:

Intrare
0
1
0
0
1
0
1
1
2

Iesire
A
U
E

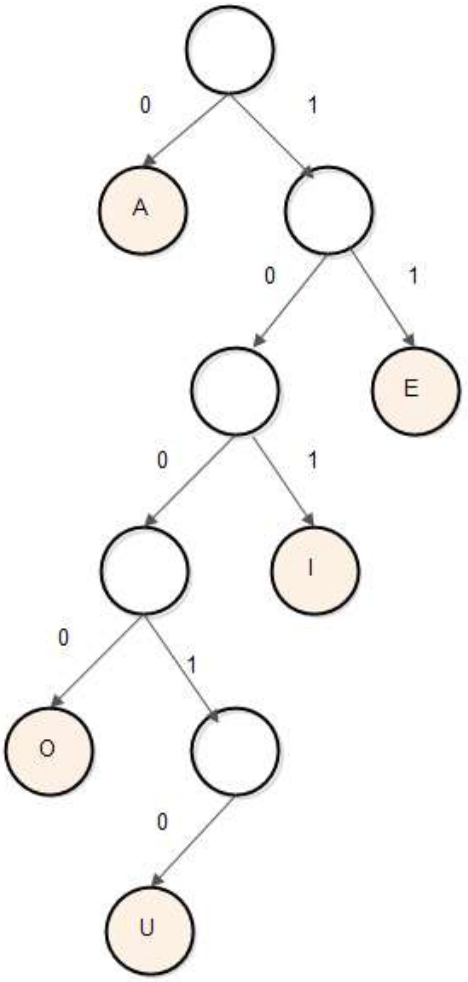
Solutie propusa:

Rezolvarea poate folosi un arbore de codare care are in nodurile terminale (fara descendenti) simbolurile codate. Determinarea unui simbol se va face prin parcurgerea arborelui din radacina in stinga sau in dreapta conform bitului citit de la intrare, pina la gasirea unui nod terminal. Figura 1 arata modul de construire al arborelui de codare.

Avind in vedere ca numarul de noduri este cunoscut, arborele poate fi implemnetat static, printr-un tablou ca in figura 2. Arborele binar are 6 niveluri si numarul de noduri este $2^0 + 2^1 + 2^2 + 2^3 + 2^4 + 2^5 = 63$. In nodurile frunza se vor scrie simbolurile codate, iar in restul nodurilor se va scrie caracterul ‘ ‘. Un nod este nod terminal atunci cind are cei 2 descendenti egali cu caracterul ‘ ‘.

Se presupune ca sirul de la intrare reprezinta o codificare corecta.

Figura 3 descrie organigrama programului. Elementele primate la intrare se introdic intr-o coada (nu se cunosate dimensiunea sirului de biti codificat). Dupa citirea sirului de biti, acesta se va prelucra prin extragerea din coada, bit cu bit.



Simbol	Cod
A	0
E	11
I	101
O	1000
U	10010

Figura 1. Arborele de codare

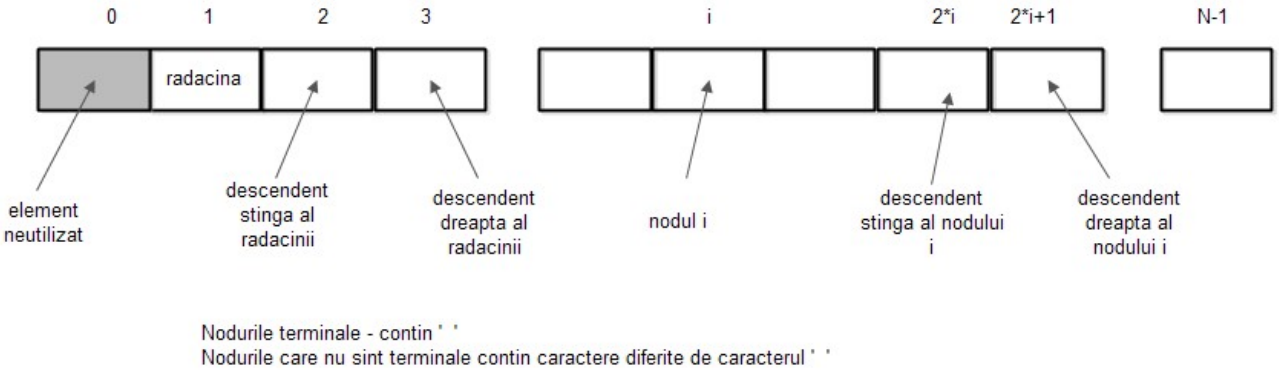


Figura 2. Reprezentarea statica a arborelui

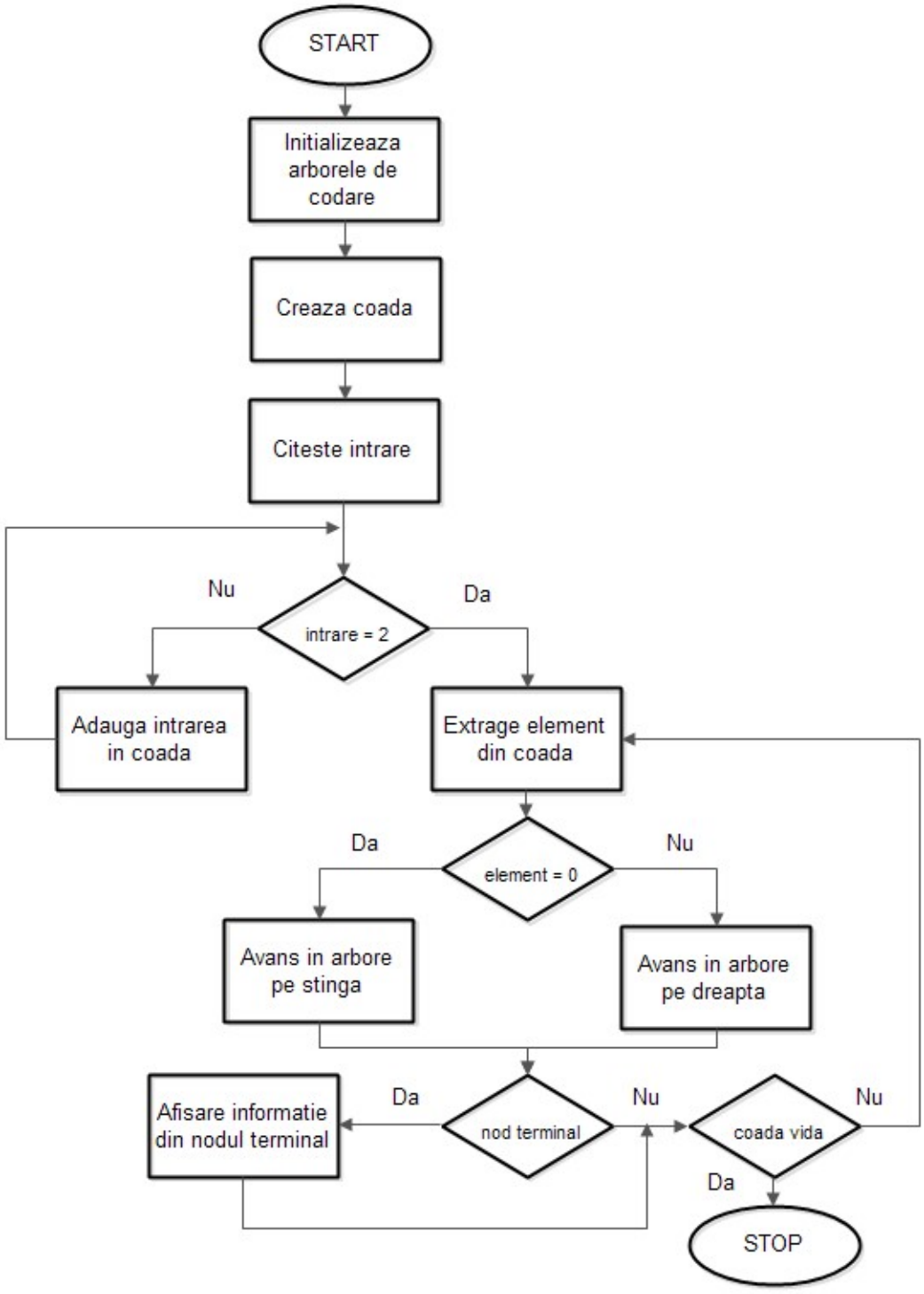


Figura 3. Organigrama programului

Codul asociat este urmatorul:

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

char static_tree[64];
struct coada
{
   int data;
   struct coada* next;
};

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

struct coada* front;
struct coada* rear;

int main() {

 int i;
 int c=-1;
 int a;

 for (i=1; i<64; i++) static_tree[i]=' ';

   static_tree[2]='A';
   static_tree[7]='E';
   static_tree[13]='I';
   static_tree[24]='O';
   static_tree[50]='U';

   front= (struct coada*)malloc(sizeof(struct coada*));
   rear= (struct coada*)malloc(sizeof(struct coada*));

   front->next=rear;
   rear->next=NULL;
   front->data=2;
   rear->data=2;

   while(c!=2)
   {
      scanf("%d",&c);
      if (c!=2) add_Q(c);
```

```
   }
   while(front->next != rear)
   {
      i=1;
      while( (static_tree[2*i] != ' ') || (static_tree[2*i+1] != ' ') )
      {
         a=del_Q();
         if (a<0) break;
         if (a==0) i=2*i;
         else i=2*i+1;
         if(i>=32 && i<=63) break;
      }
      printf("%c\n",static_tree[i]);
   }
   return 0;
}

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

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

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

   if (front->next != rear )
   {
      p=front->next;
      x=p->data;
      front->next=p->next;
      free(p);
      return x;
   }
   return -1;
}
```

Problema 7

Sa se realizeze un program care primeste un sir de numere naturale nenule, care pot fi si identice, de la tastatura (pe linii separate) va afisa la iesire secvente de numere sortate care contin numarul curent introdus si toate numerele anterior introduse. Introducerea unui 0 (care nu se va afisa) la intrare va termina secventa introdusa. La iesire se va afisa virgula dupa fiecare numar din secventele sortate.

Exemplu:

Intrare
1
5
8
2
20
3
2
0

Iesire
1,
1, 5,
1, 5, 8,
1, 2, 5, 8,
1, 2, 5, 8, 20,
1, 2, 3, 5, 8, 20,
1, 2, 2, 3, 5, 8, 20,

Solutie propusa:

Ideea de baza este aceea ca un arbore binar de cautare parcurs in ordine – produce o secventa ordonata crescator. Elementele de la intrare se vor pune intr-o coada implementata dinamic (nu se cunoaste numarul de elemente la intrare), cu doua noduri false (ca in problemele anterioare). Cu elementele din coada, se va crea un arbore binar de cautare. Dupa creare, acesta se va parcurge in ordine (LDR) si se va afisa fiecare nod.
Arborele va fi creat dinamic Functiile de creare si traversare sint recursive.
Figura 1 indica structura nodurilor pentru coada si arbore.

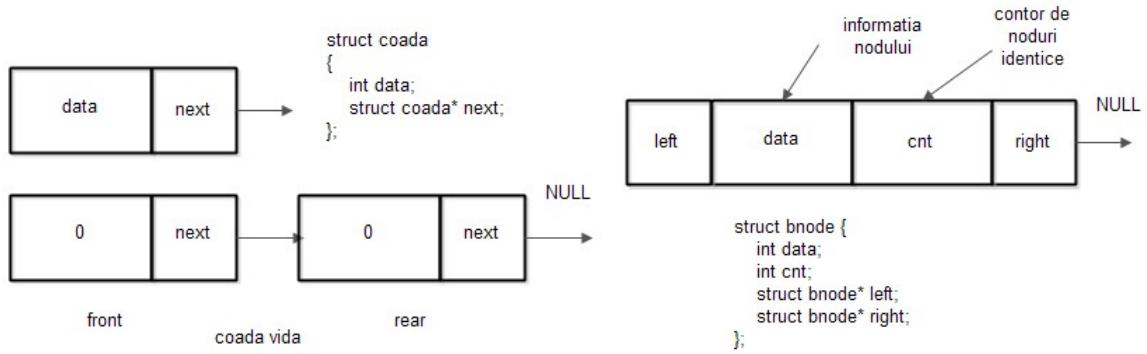


Figura 1. Structura nodurilor

Figura 2 prezinta organigrama programului.

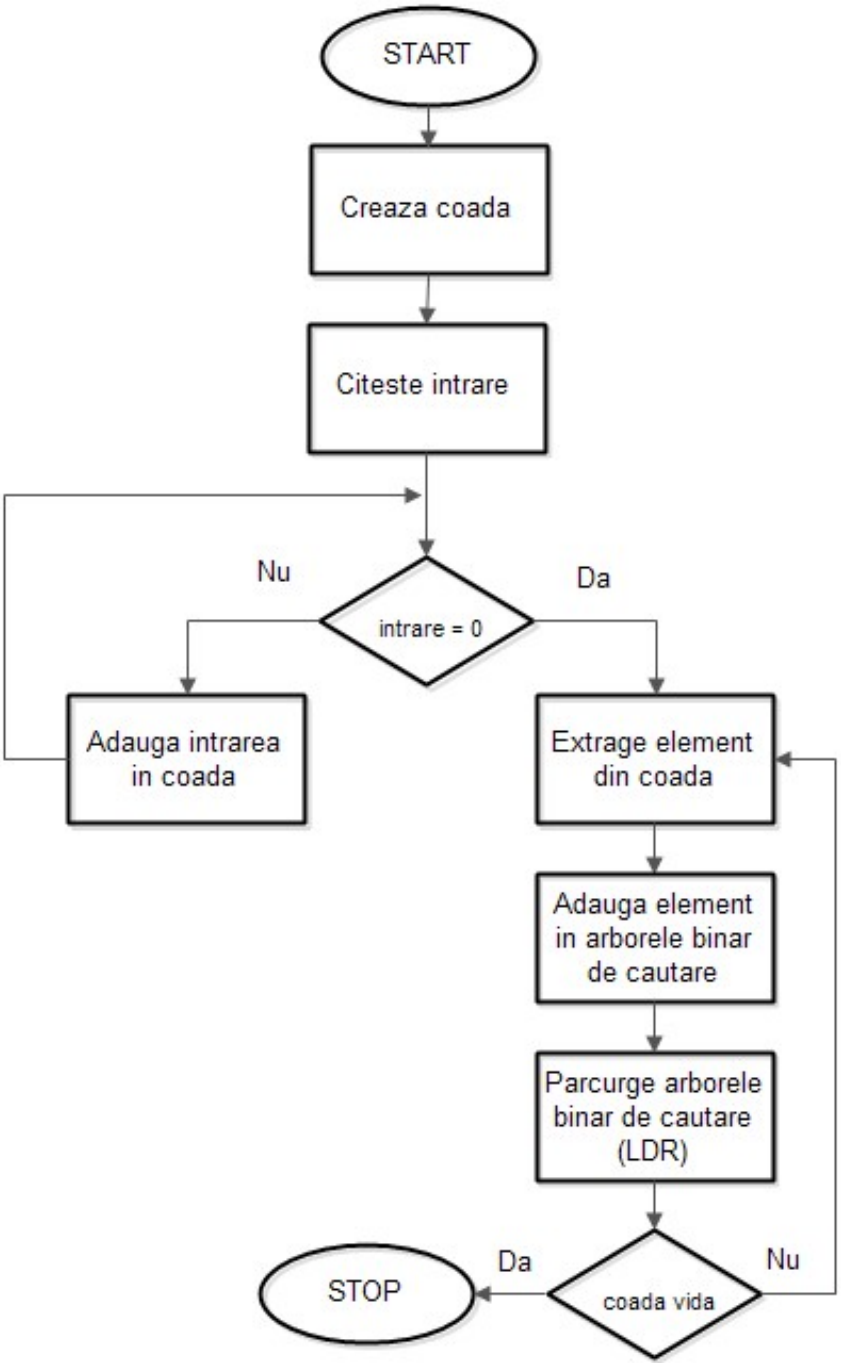


Figura 2. Organigrama programului

Codul propus este urmatorul:

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

// structura nodului arborelui binar
struct bnode {
   int data;
   int cnt;
   struct bnode* left;
   struct bnode* right;
};

// prototipurile functiilor pentru arbore
struct bnode* new_node_abc(int a);
struct bnode* build_abc(struct bnode*r, int a);
void ldr(struct bnode* r);

// nodul din coada
struct coada
{
   int data;
   struct coada* next;
};

// prototipurile functiilor pentru arbore
void add_Q(int a);
int del_Q(void);

// nodurile false
struct coada* front;
struct coada* rear;

int main() {

   int in=-1;
//nodul radacina
   struct bnode* root=NULL;
// nodurile false ale cozii
   front= (struct coada*)malloc(sizeof(struct coada*));
   rear= (struct coada*)malloc(sizeof(struct coada*));
// initializarea cozii
   front->next=rear;
   rear->next=NULL;
```

```
    front->data=0;
    rear->data=0;

    int a;

    while(in!=0)
    {
        scanf("%d", &in);
        if (in==0) break;
        add_Q(in);
    }
    while(front->next != rear)
    {
        a=del_Q();
        root=build_abc(root,a);
        ldr(root);
        printf("\n");
    }

    return 0;
}

// creare nod nou in arbore
struct bnode* new_node_abc(int a)
{
    struct bnode* r;

    r= (struct bnode*) malloc(sizeof(struct bnode));
    r->data=a;
    r->cnt=1;
    r->left=NULL;
    r->right=NULL;

    return r;
}

// construire arbore binar de cautare
struct bnode* build_abc(struct bnode*r, int a)
{
    if (r==NULL) r = new_node_abc(a);
    else
    {
        if (a < r->data ) r->left=build_abc(r->left,a);
        if (a > r->data ) r->right=build_abc(r->right,a);
        if (a==r->data) r->cnt=r->cnt+1;
```

```
    }
    return r;
}

// parcurgere in ordine (LDR) – afisare info nod
void ldr(struct bnode* r)
{
    int i;
    if(r!=NULL)
    {
        ldr(r->left);
        for (i=0; i < r->cnt; i++) printf("%d,", r->data);
        ldr(r->right);
    }
}

// adaugare in coada
void add_Q(int a)
{
    struct coada* p;

    rear->data=a;
    p=(struct coada*) malloc(sizeof(struct coada));
    p->next=NULL;
    rear->next=p;
    rear=p;
    rear->data=0;
}
// extragere din coada
int del_Q(void)
{
    int x;
    struct coada* p;

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

Problema 8

Sa se realizeze un program care afiseaza, in ordine descrescatoare elementele unei multimi de numere naturale nenule introduse de la tastatura. Elementele identice se vor afisa o singura data. Introducerea unui 0 (care nu se va afisa) la intrare va termina secventa introdusa.

Exemplu:

Intrare:
2
4
6
8
2
3
8
7
5
0

Iesire
8
7
6
5
4
3
2

Solutie propusa:

Se vor introduce elementele de la intrare intr-o coada dinamica (deoarece nu se cunoaste numarul elementelor). Dupa terminarea introducerii secventei de intrare, se va crea un arbore binar de cautare. Elementele identice vor fi ignorate in functia de creare a arborelui. Afisarea in ordine descrescatoare se va face folosind o functie de parcurgere a arborelui binar – in post ordine (RDL) conform figurii 1.

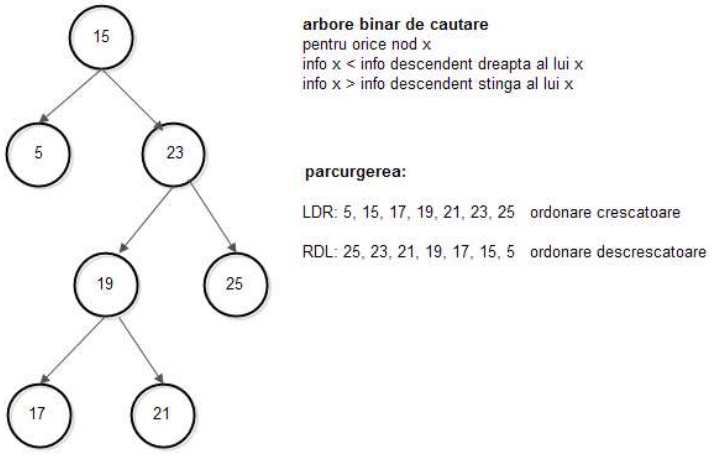


Figura 1. Parcurgerile in ordine (LDR) si post ordine (RDL)

Coada si arborele sint implementate dinamic. Functiile pentru arborele binar sint recursive. Figura 2 indica structura nodurilor, iar figura 3 prezinta organigrama programului.

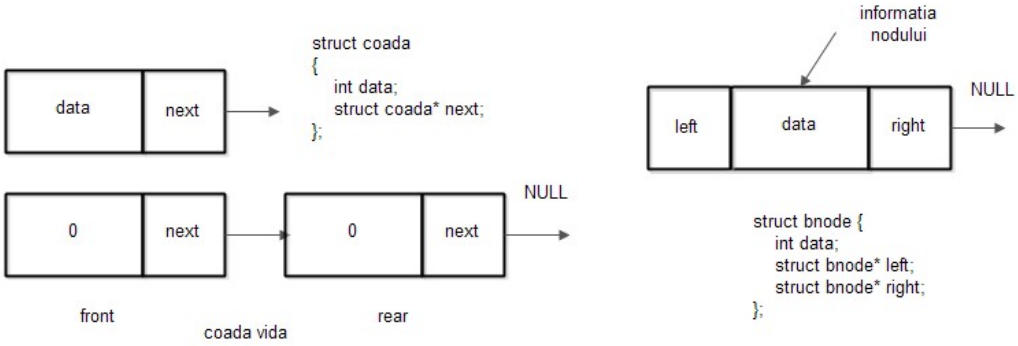


Figura 2. Structura nodurilor

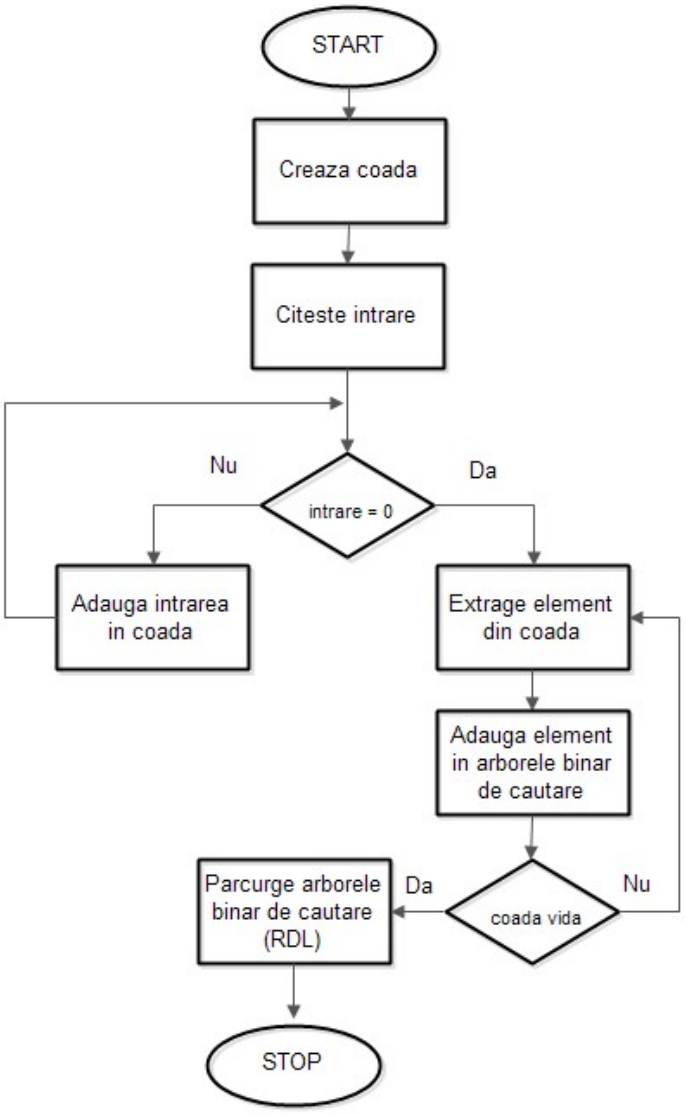


Figura 3. Organigrama programului

Codul propus este urmatorul:

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


// nodul arborelul binar
struct bnode {
   int data;
   struct bnode* left;
   struct bnode* right;
};
// nodul cozii
struct coada
{
   int data;
   struct coada* next;
};
// functii pentru coada
void add_Q(int a);
int del_Q(void);
// functii pentru arbore
struct bnode* build_abc(struct bnode*r, int a);
void rdl(struct bnode* r);

// nodurile false
struct coada* front;
struct coada* rear;

int main() {

   int c=-1;

   // radacina arborelui
  struct bnode* root=NULL;

   front= (struct coada*)malloc(sizeof(struct coada*));
   rear= (struct coada*)malloc(sizeof(struct coada*));

  // coada vida
   front->next=rear;
   rear->next=NULL;
   front->data=0;
   rear->data=0;
```

```
   while(c!=0)
   {
      scanf("%d",&c);
      if (c==0) break;
      add_Q(c);
   }

   while(front->next != rear)
   {
      c=del_Q();
      root=build_abc(root,c);
   }

   rdl(root);

   return 0;
}

// crearea arborelui binar de cautare
struct bnode* build_abc(struct bnode*r, int a)
{
   if (r==NULL)
   {
      r= (struct bnode*) malloc(sizeof(struct bnode));
      r->data=a;
      r->left=NULL;
      r->right=NULL;
   }
   else
   {
     // nodurile identice sint inserate o singura data
      if (a < r->data ) r->left=build_abc(r->left,a);
      if (a > r->data ) r->right=build_abc(r->right,a);
   }

   return r;
}

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

   rear->data=a;
   p=(struct coada*) malloc(sizeof(struct coada));
```

```
   rear->next=p;
   rear=p;
   rear->data=0;
}

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

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

// parcurgerea in post ordine (RDL)
void rdl(struct bnode* r)
{
   if(r!=NULL)
   {
      rdl(r->right);
      printf("%d\n", r->data);
      rdl(r->left);
   }
}
```

Problema 9

Sa se realizeze un program care preia de la intrare siruri de caractere si scrie la iesire un sir de numere naturale nenule care reprezinta ordinea alfabetica a sirurilor de la intrare. Fiecare sir va fi introdus pe o linie separata. Nu se vor da la intrare siruri identice. Introducerea sirului "stop" (care nu se va afisa) la intrare va termina secventa introdusa. Sirurile de caractere nu contin spatii si au maxim 10 de elemente.

Exemplu:

Intrare:

Popescu
Ionescu
Vasilescu
Avramescu
Florescu
stop

Iesire:

4
3
5
1
2

Solutie propusa:

Elementele de la intrare sint introduse intr-o lista (inserare la sfirsit). Lista este creata dinamic, cu un nod fals *prim*. Elementele sint apoi citite din lista si inserate intr-un arbore binar de cautare (dupa cimpul *nume*). Nodurile pentru lista si arbore sint ilustrate in figura1. Pentru arbore, exista un cimp, *index*, pentru a se stoca pozitia elementului *nume*.

Adaugare in lista se face la sfirsitul acesteia , ca in figura 2.

Dupa construirea arborelui, acesta se parcurge in ordine (LDR) si se actualizeaza cimpul *index* pentru fiecare nod. Datorita proprietatii de ordonare crescatoare a parcurgerii LDR pentru un arbore binar de cautare, cimpul *index* va fi actualizat in ordinea alfabetica a cimpului *nume*.

Dupa completarea cimpurilor *index*, se citeste din nou lista (de la primul element) si se cauta nodul in arbore. Dupa gasirea acestuia, se afiseaza la iesire cimpul *index* al nodului gasit.

Figura 3 prezinta organigrama programului.

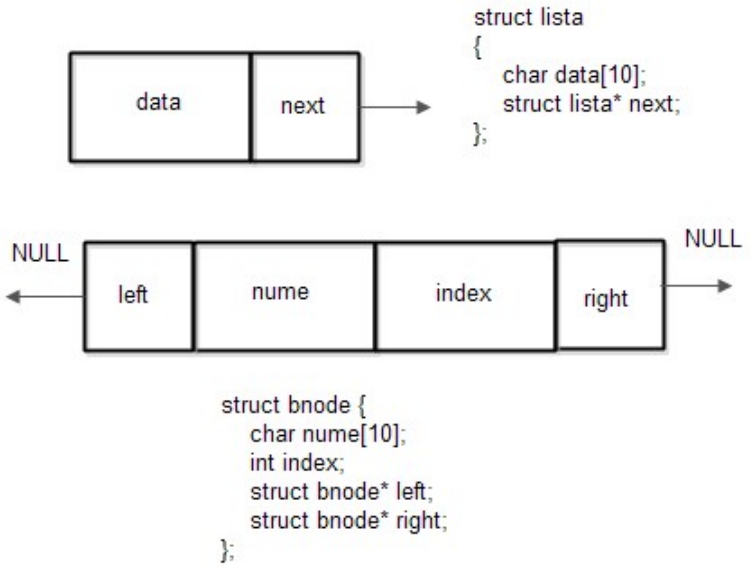


Figura 1. Nodurile listei si arborelui

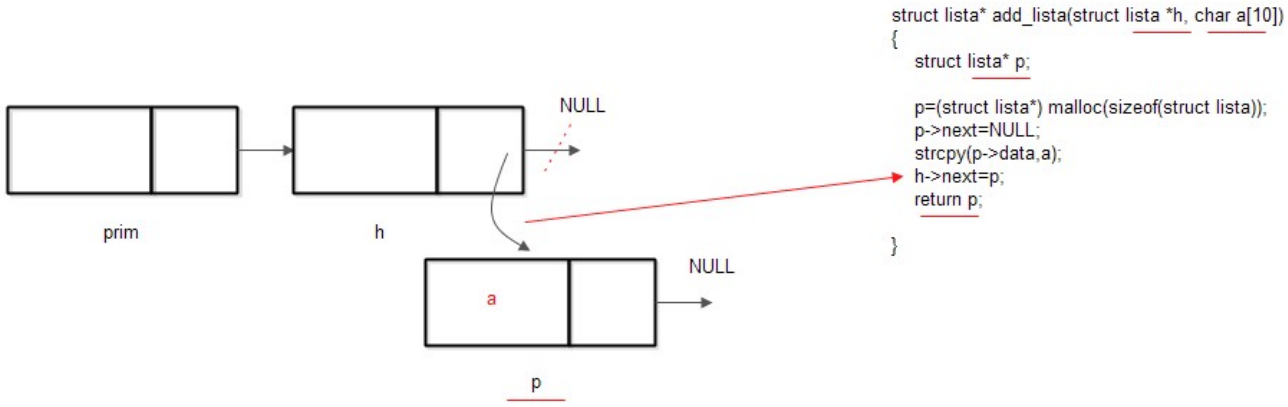


Figura 2. Adaugarea in lista

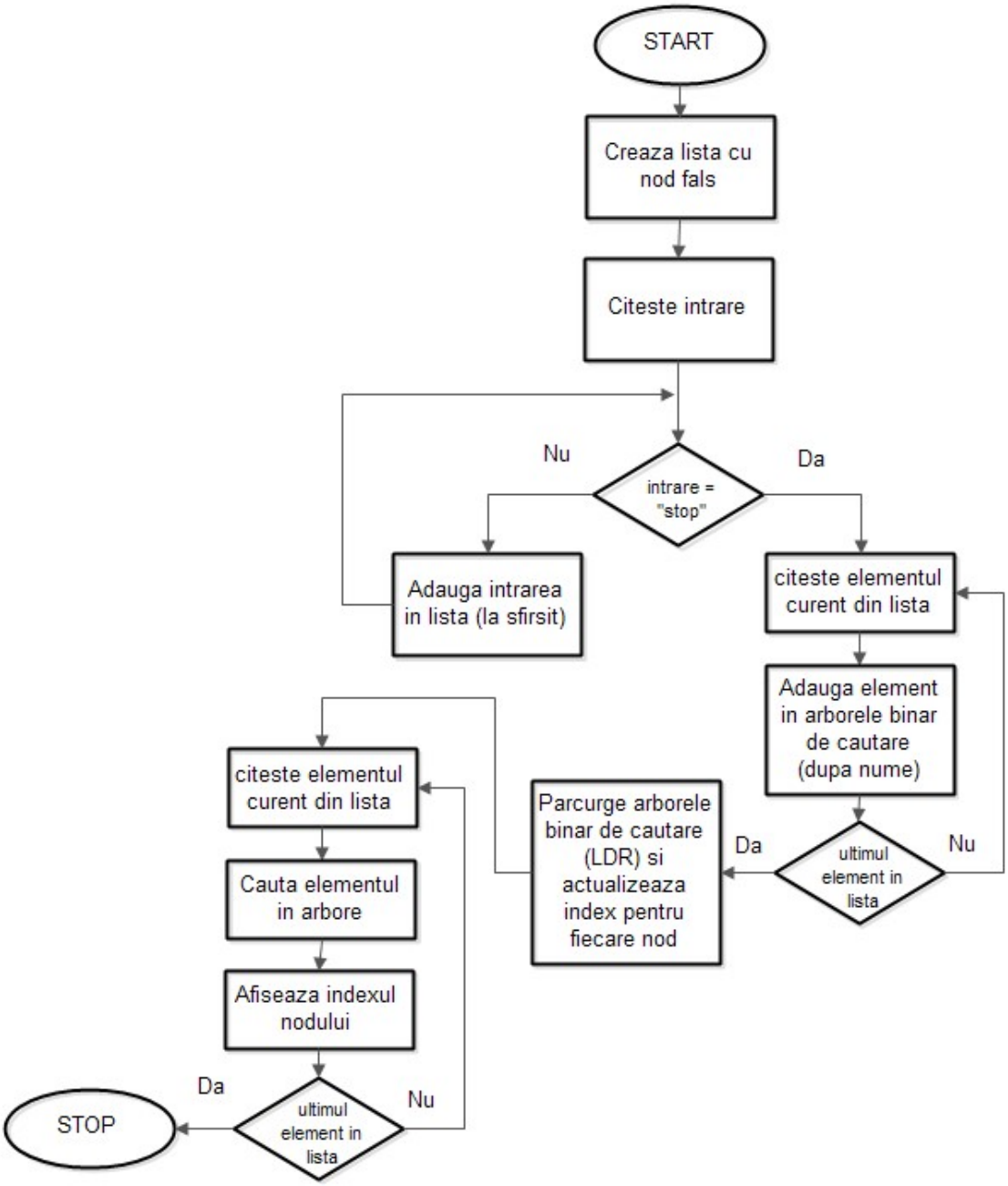


Figura 2. Organigrama programului

Codul propus este urmatorul:

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

// nodul arborelui
struct bnode {
```

```
   char nume[10];
   int index;
   struct bnode* left;
   struct bnode* right;
};

// nodul listei
struct lista
{
   char data[10];
   struct lista* next;
};

// functii pentru lista
struct lista* add_lista(struct lista* h, char a[10]);
void read_lista(char x[10]);

// functii pentru arborele binar de cautare
struct bnode* build_abc(struct bnode*r, char a[10]);
void ldr(struct bnode* r);
struct bnode* cauta_abc(struct bnode*r, char a[10]);

// nod fals lista
struct lista* prim;
// nodul current (dupa care se va insera)
struct lista* current;

// index pentru nodurile arborelui
int nr=0;
int main() {

   char c[10]="";
   struct bnode* root=NULL;
   struct bnode* r=NULL;
// creare nod fals
   prim= (struct lista*)malloc(sizeof(struct lista*));
   prim->next=NULL;
   strcpy(prim->data,"");
   current=prim;

     while(strcmp(c,"stop") != 0)
   {
      scanf("%s",c);
      if (strcmp(c,"stop") == 0) break;
      current = add_lista(current, c);
```

```
    }
    current=prim->next;
    while(current != NULL)
    {
       if (current == NULL) break;
       read_lista(c);
       root=build_abc(root,c);
    }

    ldr(root);

    current=prim->next;

    while(current != NULL)
    {
       if (current == NULL) break;
       read_lista(c);
       r=cauta_abc(root,c);
       if (r!=NULL) printf("%d\n",r->index);
    }
    return 0;
}

// construire arbore binar de cautare dupa nume
struct bnode* build_abc(struct bnode*r, char a[10])
{
    if (r==NULL)
    {
       r= (struct bnode*) malloc(sizeof(struct bnode));
       strcpy(r->nume,a);
      // index inca neactualizat
       r->index=0;
       r->left=NULL;
       r->right=NULL;
    }
    else
    {
       if ( strcmp(a, r->nume) < 0 ) r->left=build_abc(r->left,a);
       if ( strcmp(a, r->nume) > 0 ) r->right=build_abc(r->right,a);
    }
    return r;
}
// cautare in arbore
// se cauta dupa cimpul nume, incepind cu nodul r
// intoarce nodul gasit
```

```
struct bnode* cauta_abc(struct bnode*r, char a[10])
{
   if (r==NULL) return NULL;
   if (strcmp(r->nume,a) == 0) return r;
   if (strcmp(a,r->nume) < 0) return cauta_abc(r->left,a);
   if (strcmp(a,r->nume) > 0) return cauta_abc(r->right,a);
}

// parcurge in ordine arborele binar
// actualizeaza cimpul index, in ordinea crescatoare a cimpului nume
void ldr(struct bnode* r)
{
   if(r!=NULL)
   {
      ldr(r->left);
      // primul element are indexul 1
      nr++;
      r->index=nr;
      ldr(r->right);
   }
}

// adauga in lista, dupa nodul h
struct lista* add_lista(struct lista *h, char a[10])
{
   struct lista* p;

   p=(struct lista*) malloc(sizeof(struct lista));
   p->next=NULL;
   strcpy(p->data,a);
   h->next=p;
   return p;

}
// citeste elemental curent din lista
// si avanseaza la urmatorul element
void read_lista(char x[10])
{
   strcpy(x, current->data);
   current=current->next;
}
```

Problema 10

Sa se scrie un program care preia de la tastaura, pe linii separate, perechi de numere naturale (index) si siruri numerice (nume). Programul va afisa la iesire, pe cite o linie, sirurile de caractere in ordinea indexului asociat. Nu se pot da la intrare perechi identice si fiecare sir are un index unic. Introducerea unui 0 (care nu se va afisa) la intrare va termina secventa introdusa. Sirurile de numere au maxim 10 caractere.

Exemplu:

Intrare:

4
Ion
1
Vasile
8
Adrian
2
Marius
0

Iesire:

Vasile
Marius
Ion
Adrian

Solutie propusa:

Se vor citi de la tastatura perechi *(index, nume*) si se va crea un arbore binar de cautare dupa cheia *index*. Nu e necesara o lista, se pot pune elementele direct in arbore. Apoi se traverseaza arborele in ordine (LDR) si se afiseaza cimpul *nume*. Structura nodurilor din arbore este ilustrata in figura 1. In figura 2 este prezentata organigrama programului.


```
struct bnode {
    char nume[10];
    int index;
    struct bnode* left;
    struct bnode* right;
};
```

Figura 1. Structura nodului arborelui

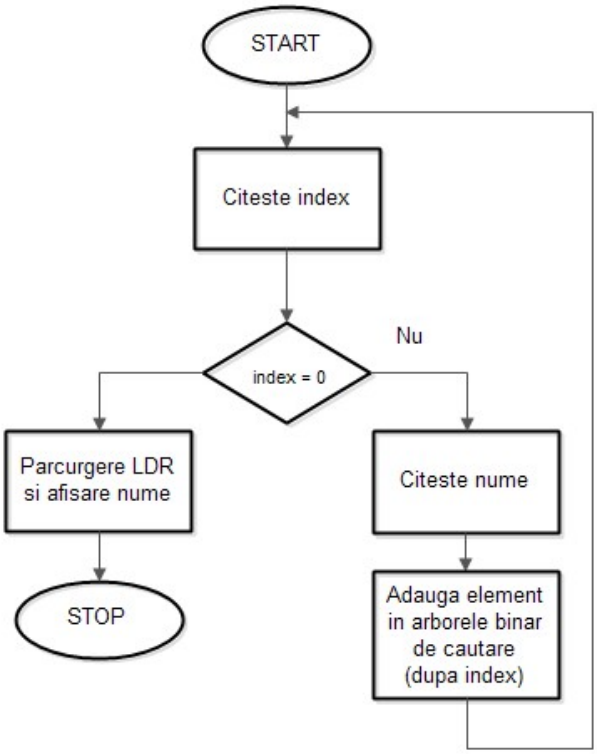


Figura 2. Organigrama programului

Codul propus este urmatorul:

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

// structura nodului din arbore
struct bnode {
   char nume[10];
   int index;
   struct bnode* left;
   struct bnode* right;
};

// functii pentru arbore
struct bnode* build_abc(struct bnode*r, int n, char a[10]);
void ldr_nume(struct bnode* r);

int main() {

   int index = -1;
   char nume[10];
```

```
   struct bnode* root=NULL;

   while(index != 0)
   {
      scanf("%d",&index);
      if (index == 0) break;
      scanf("%s",nume);
      root=build_abc(root,index,nume);
   }
   ldr_nume(root);
   return 0;
}

// construirea arborelui binar de cautare
// dupa cimpul index
struct bnode* build_abc(struct bnode*r, int n, char a[10])
{
   if (r==NULL)
   {
      r= (struct bnode*) malloc(sizeof(struct bnode));
      strcpy(r->nume,a);
      r->index=n;
      r->left=NULL;
      r->right=NULL;
   }
   else
   {
      if (n < r->index) r->left=build_abc(r->left,n,a);
      if (n > r->index) r->right=build_abc(r->right,n,a);
   }
   return r;
}

// parcurgere LDR si afisare cimp nume
void ldr_nume(struct bnode* r)
{
   if(r!=NULL)
   {
      ldr_nume(r->left);
      printf("%s\n",r->nume);
      ldr_nume(r->right);
   }
}
```