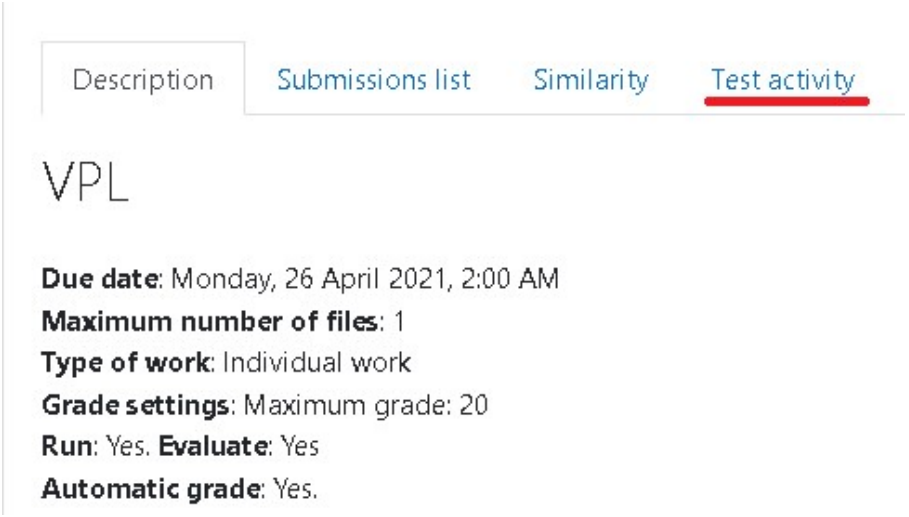
Exemple de probleme cu liste si arbori binari

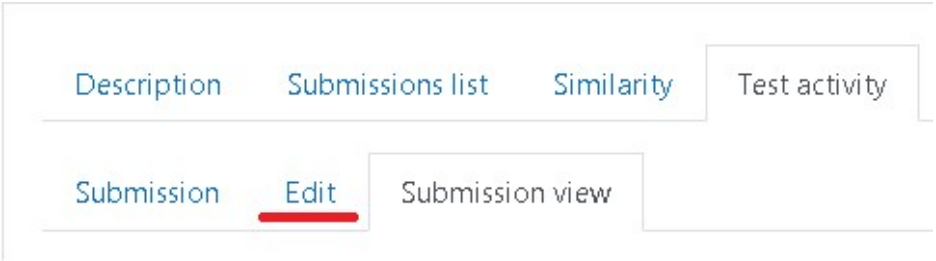
Cuprins:

1. Modul de utilizare a resursei VPL în Moodle (de către student)

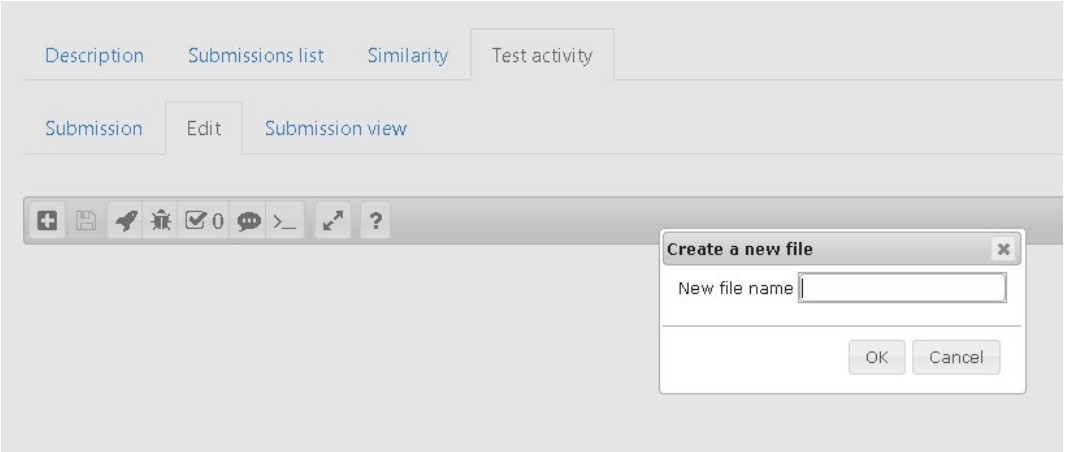
1) Se accesează resursa VPL – click pe link-  VPL
2) Se accesează meniul **Test activity**



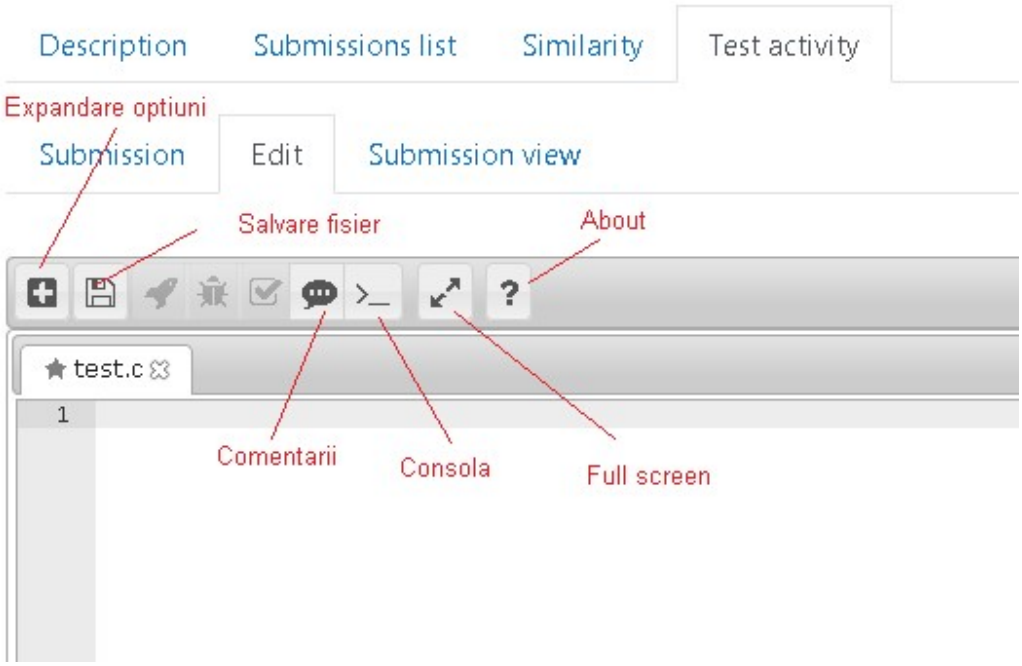
3) Din meniul **Test activity** se selectează sub-meniul **Edit**



4) În **Edit** se creează un fișier nou (dacă nu era deja creat). Fișierul trebuie să aibă extensia .C – dacă programul este scris in limbaj C sau o extensie corespunzătoare dacă programul se scrie în alt limbaj. Numele fișierului se va scrie în fereastra **Create a new file.** Comanda se va termina apăsând butonul **OK.**



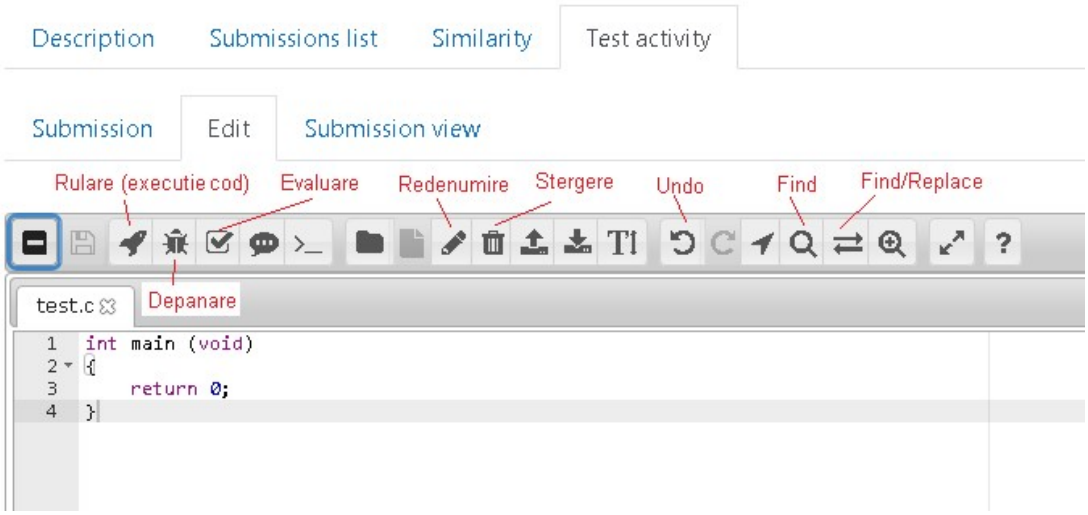
5) În fereastra de editare se va scrie programul care trebuie salvat prin apăsarea butonului de **Salvare fișier**. Trimiterea fișierului (submission) se va face automat la salvarea fișierului. Se pot face oricâte salvări și se va considera ultima variantă salvată.



Există și alte butoane: **Expandare opțiuni**, **Comentarii**, **Consola** (pentru a introduce intrările în program și a vizualiza ieșirile acestuia**), Full screen** și **About**.

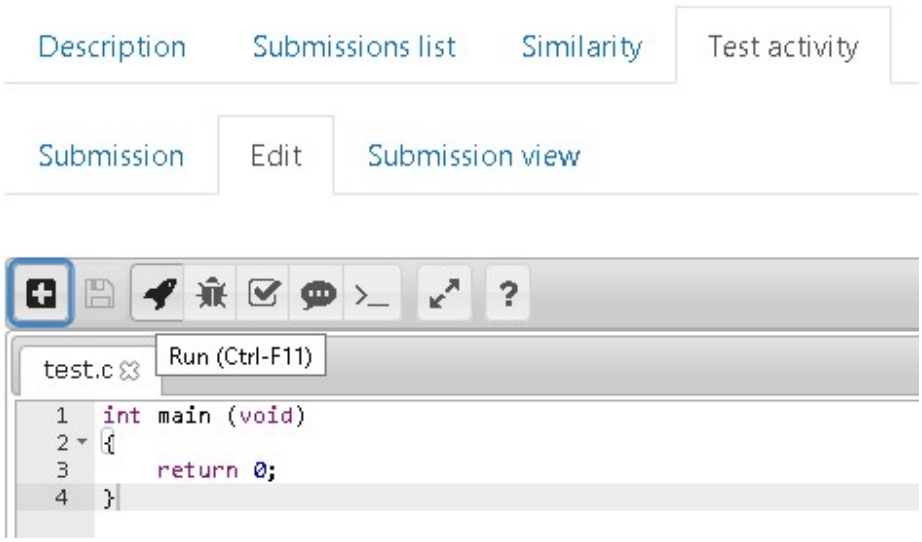
6) După editarea codului se va salva fișierul. Orice modificare în program va activa butonul de **Salvare fisier**.

Codul poate fi rulat sau evaluat (dacă s-a definit un fișier de testare automata).

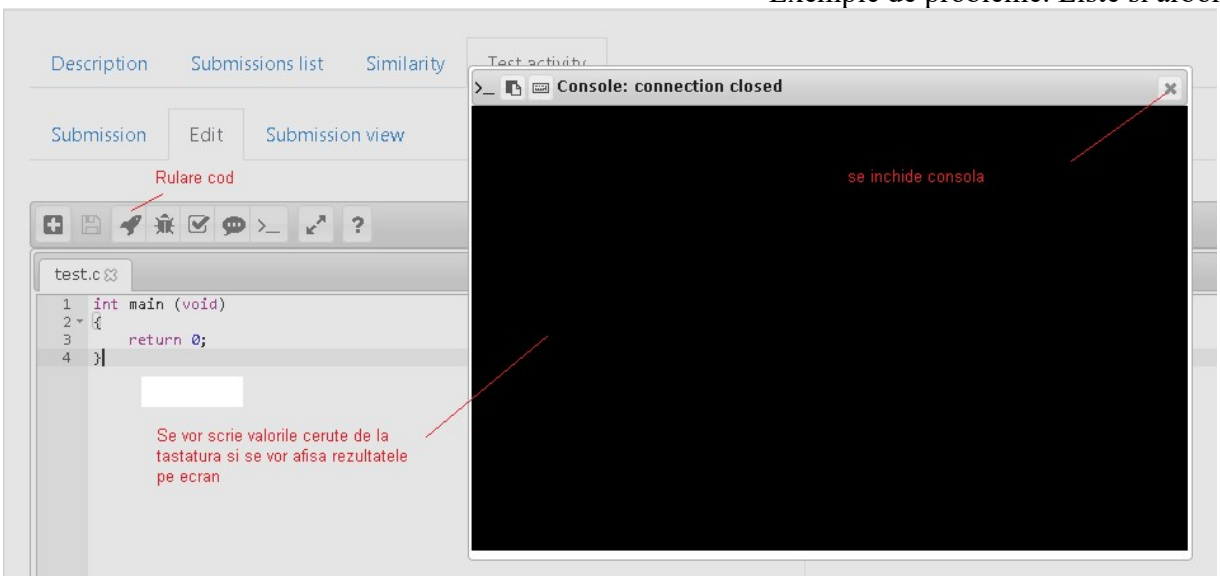


Se recomandă depanarea în CLion, apoi copierea codului din CLion în VPL.

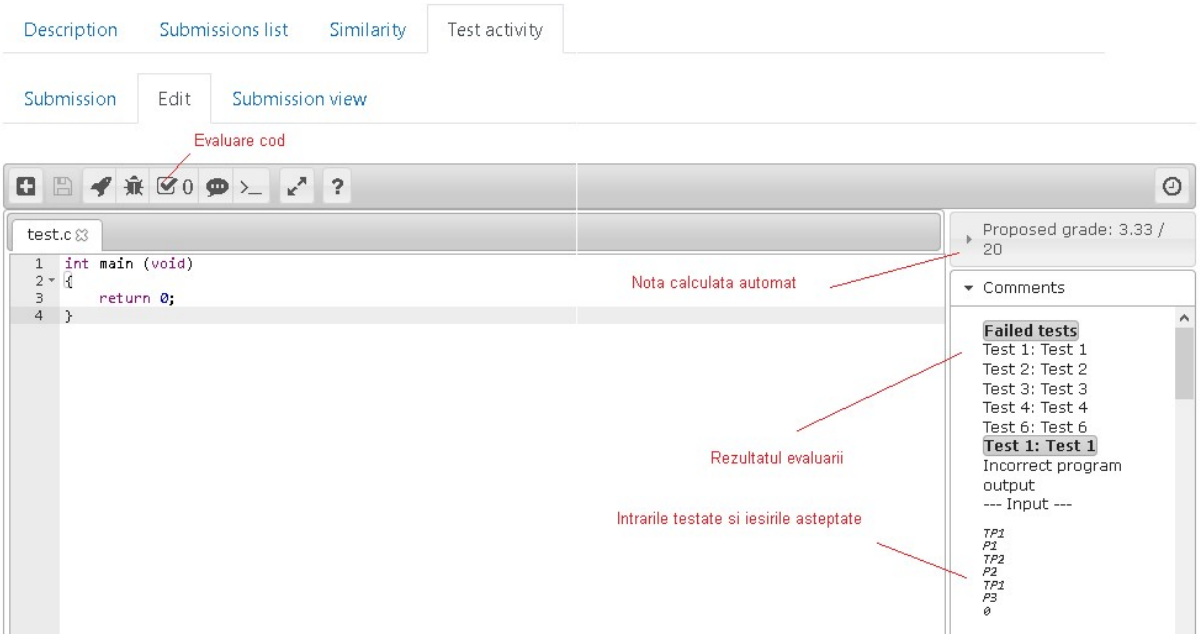
Există comenzi de editare, de căutare în fișier, de redenumire a fișierului etc. Comenzile pot fi identificate ușor plasând mouse-ul pe butonul dorit (semnificația butonului se va afișa automat).



7) Codul se execută și se introduc valori de intrare (dacă e cazul). Se vizualizează ieșirile obținute.

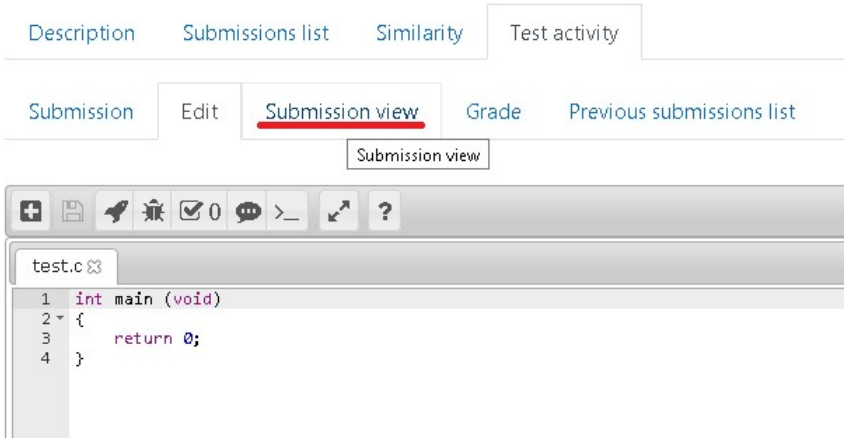


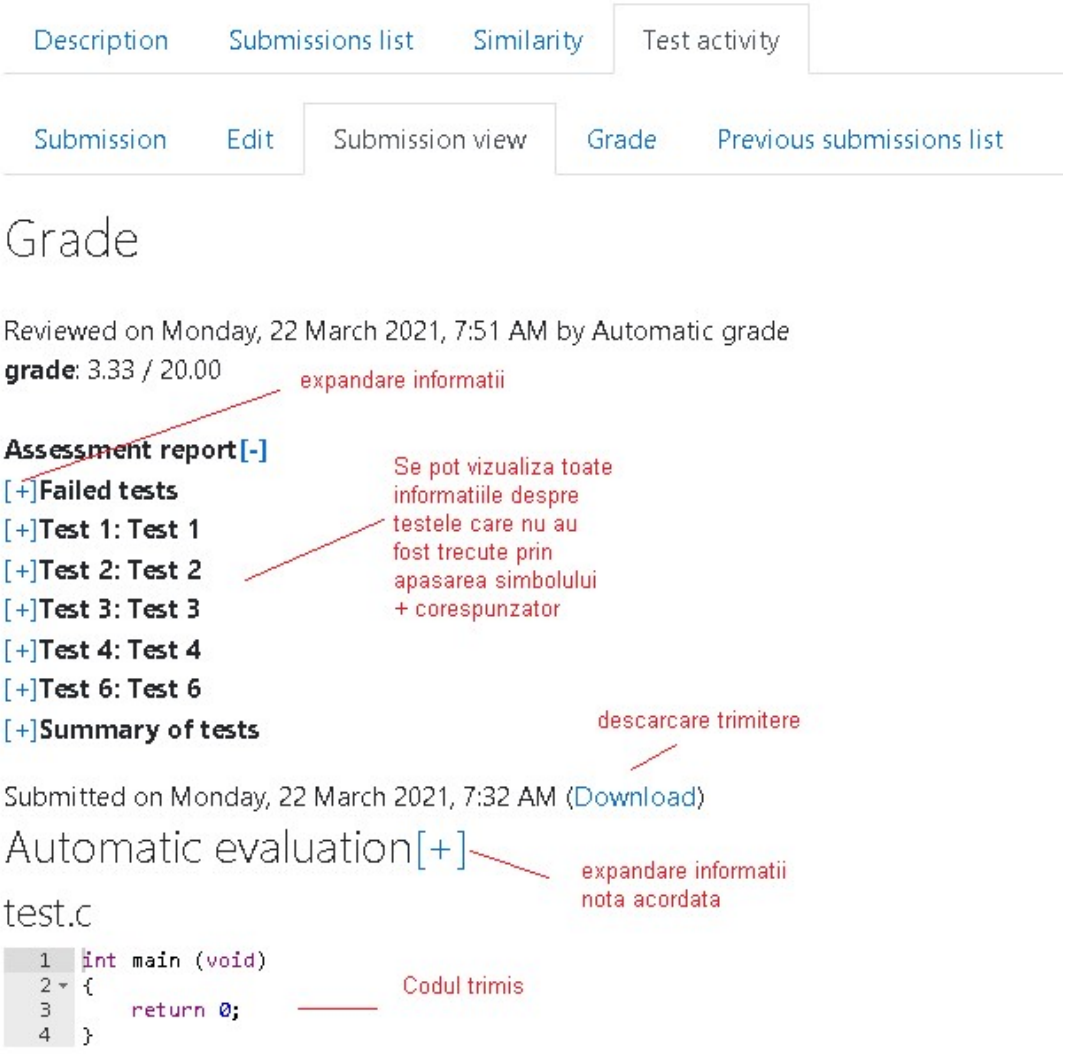
8) Codul poate fi evaluat automat:



În fereastra din dreapta va apărea rezultatul evaluării. Pentru testele care nu au fost trecute se vor afișa intrările testate și ieșirile așteptate. Aceste informații pot fi utile pentru a se corecta codul.

9) Trimiterea și rezultatul obținut pot fi vizualizate din meniul **Submission view**





Se pot vizualiza toate informațiile despre trimitere (codul trimis, testele care nu au trecut si nota acordata).

Observație:

Nota acordată automat poate fi modificată (scăzută) de către profesor în următoarele situații:

- Rezultatele au fost obținute prin "hard-codare", nu prin implementarea unui algoritm
- Nu s-au respectat anumite cerințe impuse de subiect (care nu pot fi testate automat)

10) Editarea (modificarea codului), rularea acestuia și evaluarea pot fi repetate până la obținerea rezultatului dorit.

2. Exemple de probleme cu liste dinamice

Problema 1

Să se scrie un program care afișează tipurile de produse și produsele dintr-un magazin. Tipurile de produse și produsele sunt introduse de la tastatură în forma: TipProdus, Produs (pe linii separate). Pot fi introduse tipuri și produse identice, dar se vor afișa o singură dată. Afișarea se va face în ordinea introducerii la intrare. Introducerea unui 0 ca TipProdus (care nu se va afișa la ieșire) la intrare va termina secvența introdusă. Numele de tipuri de produse și de produse nu conțin spații și au maxim 100 de caractere.

Exemplu:

Intrare:

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

Ieșire

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

Soluție propusă – cu explicații:

Problema nu specifică numărul maxim de tipuri de produse sau de produse. Rezolvarea poate folosi o structură de două liste simplu înlănțuite ca în figura 1. Există un nod fals, *headerTip*, care va fi folosit pentru a lega toate celelalte noduri. Pe verticală se vor lega nodurile care conțin tipul de produs, iar pe orizontală se vor lega nodurile care conțin produse de același tip. În figura 1.1 s-a exemplificat cazul a două tipuri de produse, TP1 cu produsul P1 și TP2 cu produsele P2 și P3. De asemenea, sunt indicate și declarațiile corespunzătoare nodurilor.

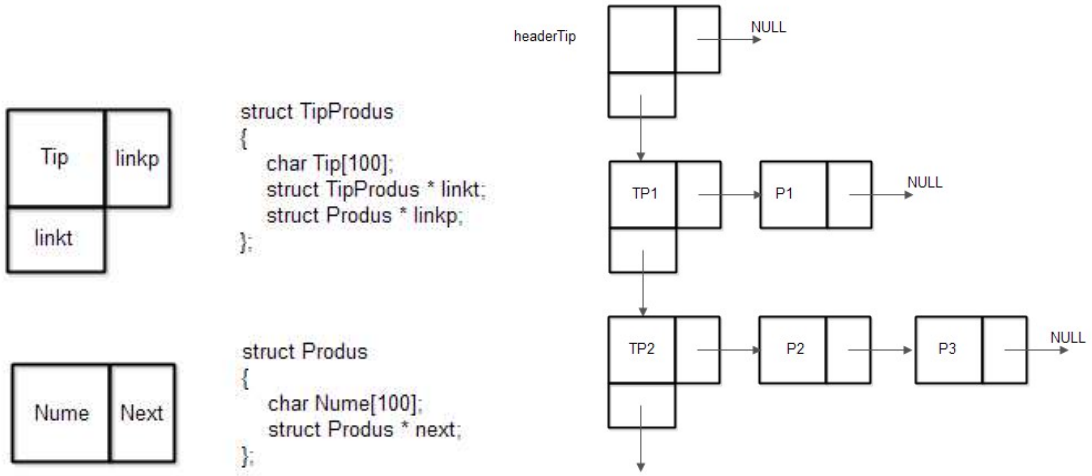
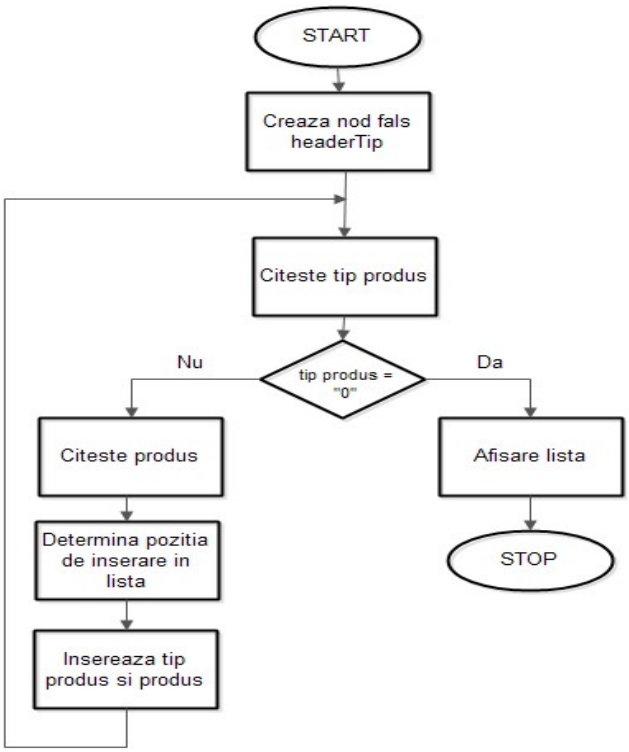


Figura 1.1. Structura listelor și declarațiile de tip

Organigrama programului este prezentată în figura 1.2.



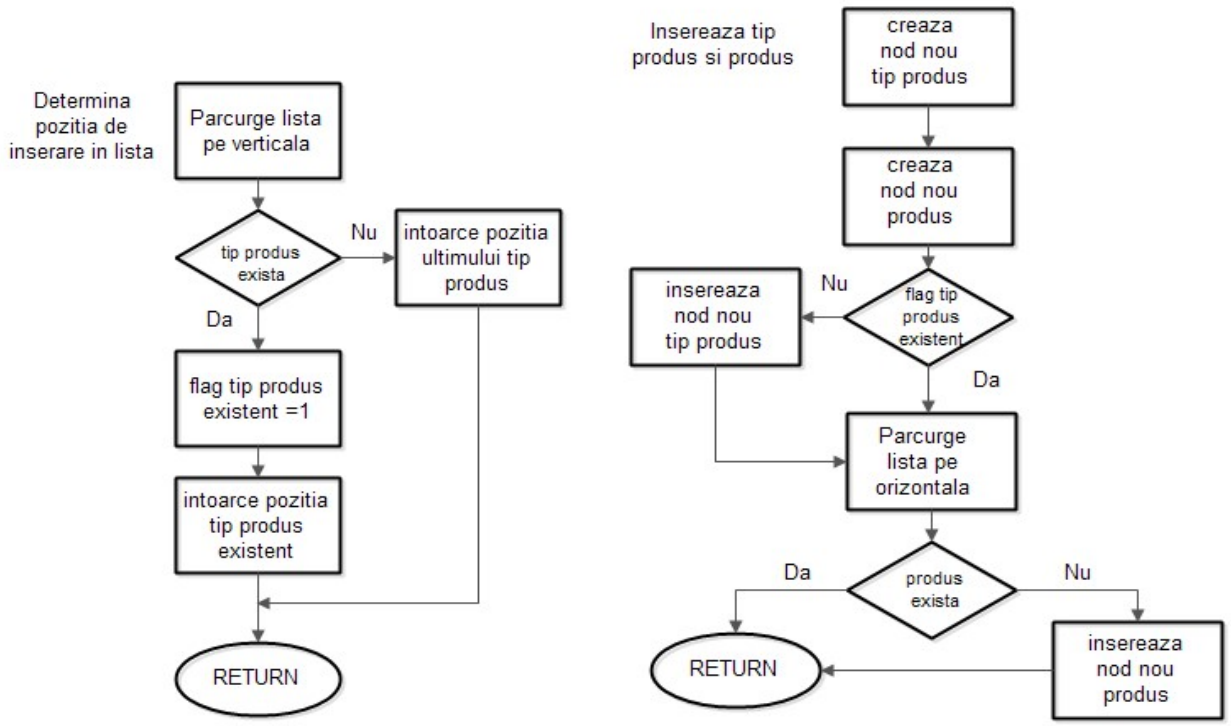


Figura 1.2 Organigrama programului

Funcția de determinare a poziției de inserare în lista de tipuri de produse, *EndTip*, este explicată în figura 1.3.

Funcția are ca parametri tipul de produs și întoarce un pointer la nodul după care se va insera noul tip de produse și un flag, *res*, care indică dacă tipul de produs există sau nu în lista de tipuri de produse. Căutarea se va face până la sfârșitul listei.

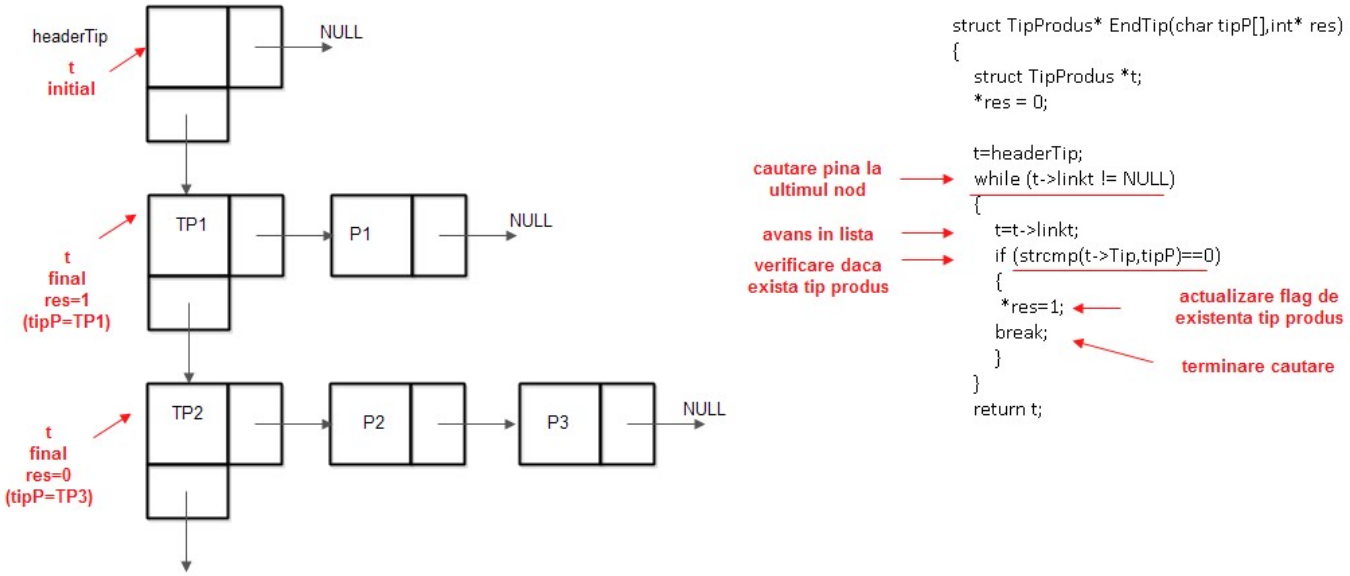


Figura 1.3. Determinarea poziției de inserare în lista de tipuri de produse

In figura 1.4 se arată modul de funcționare a funcției de inserare a tipului de produs și a produsului, *insertTipProd*. Funcția are ca parametri tipul de produs și produsul. Funcția apelează funcția anterioară, *EndTip*, de determinare a pointerului pentru tipul de produs și verifică dacă există tipul de produs. Dacă tipul de produs există, se caută în lista de produse asociată produsul. Dacă acesta nu există se va insera produsul nou la sfârșitul listei orizontale, iar dacă există nu se va insera nimic.

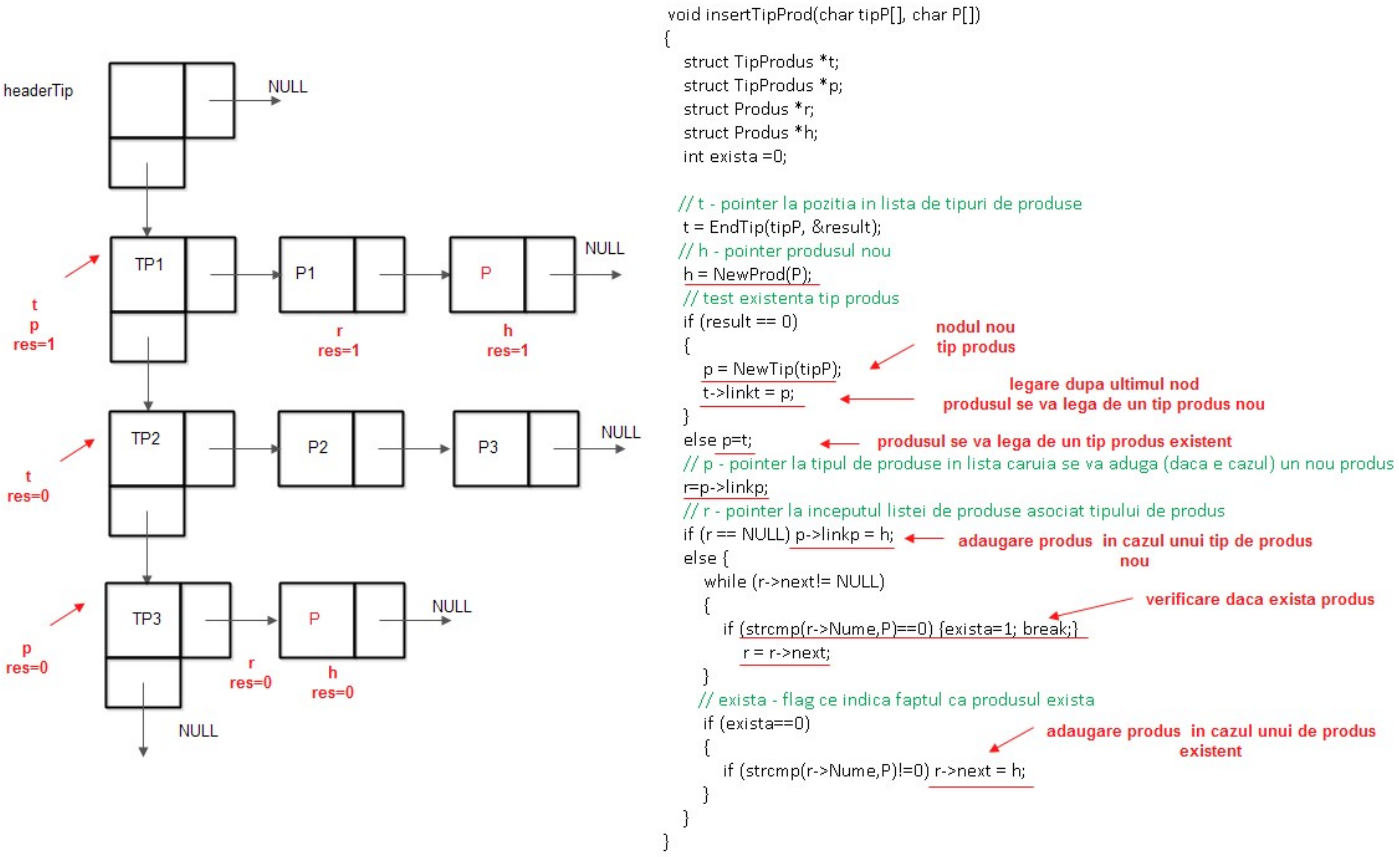


Figura 1.4. Inserarea tipului de produs și a produsului

Dacă tipul de produs nu există, se va crea și insera în lista verticală un nod nou și se va insera în lista orizontală noul produs.

Codul asociat este următorul:

```
#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 și produs
char Tip[100];
char Prod[100];



while (1)
{
   // citeste tip produs și produs
   scanf("%s",Tip);
   if (strcmp(Tip, "0")==0) break;
   scanf("%s",Prod);
   // cauta pozitiile în liste și 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(șizeof(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(șizeof(struct Produs));
   strcpy(t->Nume, P);
   t->next=NULL;

   return t;
}
// cauta pozitia în lista de tipuri de produse
// parametru de intrare numele tipului de produs
// intoarce un pointer la ultimul nod din lista de tipuri de produse și res = 0, daca tipul de produs nu exista
// intoarce un pointer la  nod tipului de produs existent din lista de tipuri de produse și 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 sfirșitul liste de tipuri de produse
// insereaza produs nou – la sfirșitul 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 în 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 în 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;
  }
}
```

Problema 2

Să se scrie un program care afișează mărcile de mașini și modelul acestora dintr-o parcare cu plată. Marca și modelul sunt introduse de la tastatură în forma: Marcă, Model (pe linii separate). Pot fi introduse mărci și modele identice, dar se vor afișa o singură data. Afișarea se va face în ordine alfabetică. Introducerea unui 0 (care nu se va afișa la ieșire) la intrare va termina secvența introdusă.

Exemplu:

Intrare:

Volkswagen
Golf
Volkswagen
Polo
Ford
Mondeo
Skoda
Fabia
Ford
Focus
Ford

Mondeo
Skoda
Octavia
0

Ieșire:

Ford
Focus
Mondeo
Skoda
Fabia
Octavia
Volkswagen
Golf
Polo

Soluție propusă – cu explicații:

Problema se poate rezolva prin modificarea funcțiilor de inserare din problema 1. Se vor crea liste ordonate. Funcția de determinare a poziției de inserare în lista de tipuri de produse, *EndTip*, este explicată în figura 2.1.

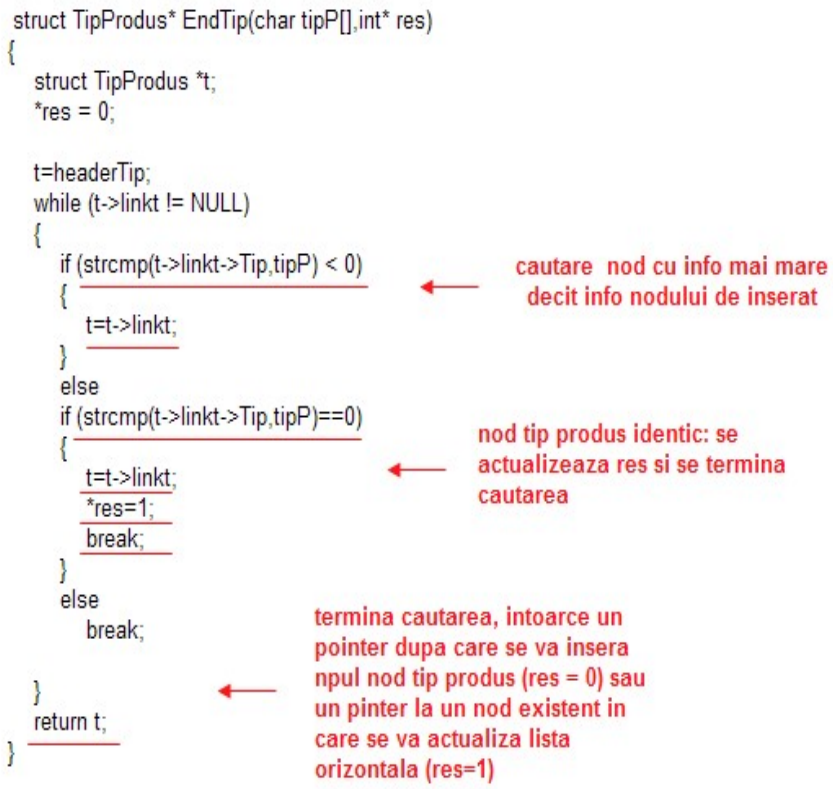


Figura 2.1. Determinarea poziției de inserare în lista ordonată de tipuri de produse

În figura 2.2 se arată modul de funcționare a funcției de inserare a tipului de produs și a produsului, *insertTipProd*.

```
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 2.2. Inserarea tipului de produs și a produsului

Pentru funcțiile de inserare, codul asociat este următorul:

```
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;}
    }
  }
}
```

Problema 3

Să se realizeze un program care primește un șir de numere naturale nenule, care pot fi și identice, de la tastatură (pe linii separate) și va afișa la ieșire secvențe de numere sortate care conțin numărul curent introdus și toate numerele anterior introduse. Introducerea unui 0 (care nu se va afișa) la intrare va termina secvența introdusa. La ieșire se va afișa virgulă după fiecare număr din secvențele sortate.

Exemplu:

Intrare
1
5
8
2
20
3
2
0

Ieșire
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,

Soluție propusă:

Ideea de bază este aceea că un arbore binar de căutare parcurs în ordine – produce o secvență ordonată crescător. Elementele de la intrare se vor pune într-o coadă implementată dinamic (nu se cunoaște numărul de elemente la intrare), cu două noduri false (ca în problemele anterioare). Cu elementele din coadă, se va crea un arbore binar de căutare. După creare, acesta se va parcurge în ordine (LDR) și se va afișa fiecare nod.

Arborele va fi creat dinamic Funcțiile de creare și traversare sunt recursive.

Figura 3.1 indică structura nodurilor pentru coadă și arbore.

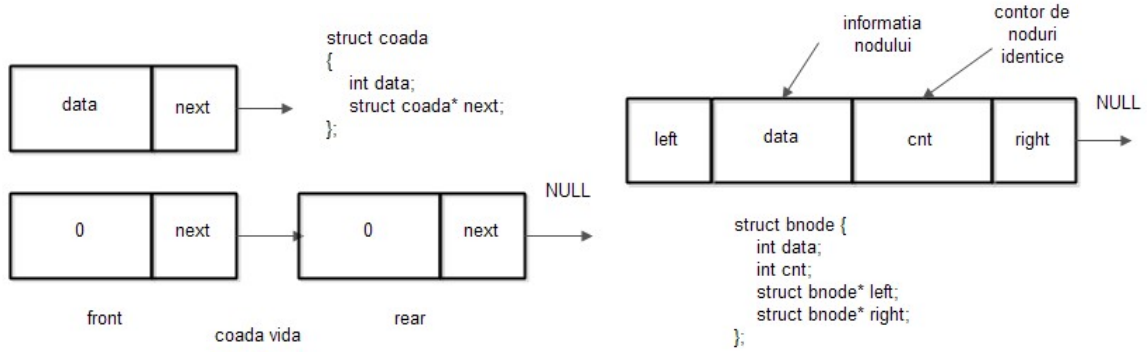


Figura 3.1. Structura nodurilor

Figura 3.2 prezintă organigrama programului:

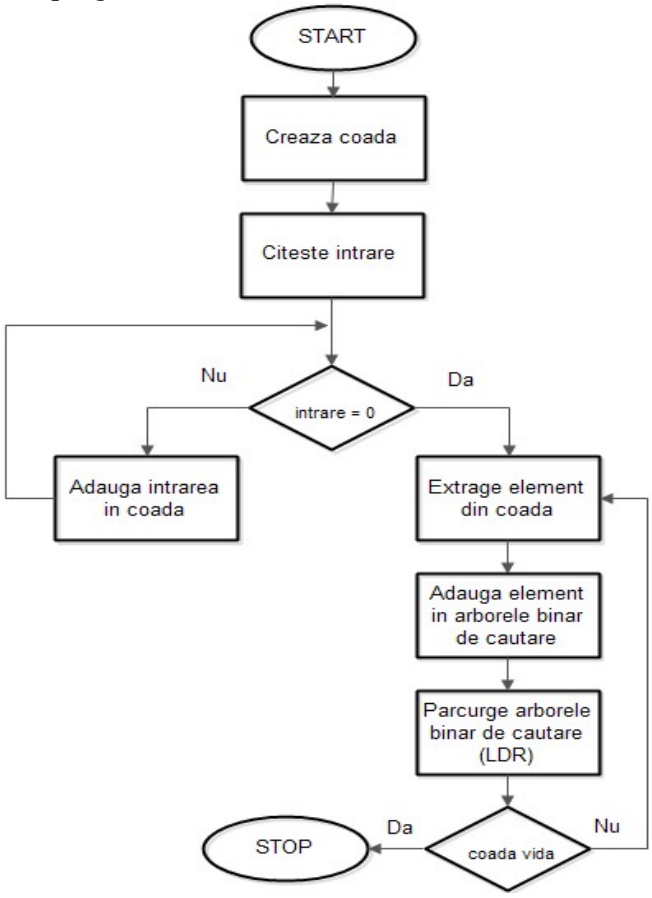


Figura 3.2. Organigrama programului

Codul propus este următorul:

```
#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 în=-1;
//nodul radacina
   struct bnode* root=NULL;
// nodurile false ale cozii
   front= (struct coada*)malloc(șizeof(struct coada*));
   rear= (struct coada*)malloc(șizeof(struct coada*));
// initializarea cozii
   front->next=rear;
   rear->next=NULL;
   front->data=0;
   rear->data=0;
   int a;

   while(în!=0)
```

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

// creare nod nou în 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;
}
void ldr(struct bnode* r)       // parcurgere în ordine (LDR) – afisare info nod
{
   int i;
   if(r!=NULL)
   {
      ldr(r->left);
```

```
      for (i=0; i < r->cnt; i++) printf("%d,", r->data);
      ldr(r->right);
   }
}

void add_Q(int a)                  // adaugare în coada
{
   struct coada* p;
   rear->data=a;
   p=(struct coada*) malloc(șizeof(struct coada));
   p->next=NULL;
   rear->next=p;
   rear=p;
   rear->data=0;
}

int del_Q(void)                           // extragere din coada
{
   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;
}
```

3. Desfășurarea lucrării

a) Se va crea un proiect CLion in care se va rula cu depanare codul asociat problemei 1.
b) Se va experimenta modul de lucru in mediul VPL (creare fișier nou, editare, compilare, execuție si evaluare) pentru codul dezvoltat anterior in CLion
c) Se vor studia organigramele asociate problemelor 1, 2 si 3 si modul de implementare dinamica a listelor si a arborilor binari.

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