Exemple de probleme cu tablouri, liste si arbori binari

Cuprins:

1. Exemple de probleme cu tablouri, liste dinamice si arbori binari
2. Desfășurarea lucrării

1. Exemple de probleme cu tablouri, liste dinamice si arbori binari

Problema 1

Se cere să se realizeze un program ce execută decodificarea unui text care conține doar vocalele alfabetului. Șirul de biți asociat unui simbol are lungime variabilă, astfel: A: 0, E: 11, I: 101, O: 1000 și U: 10010. Șirul de biți este introdus de la tastatură (un bit pe o line separata) și textul decodificat va fi scris pe ecran (câte un caracter pe o linie separată), Introducerea unui 2 (care nu se va afișa) la intrare va termina secvența introdusa.

Exemplu:

Intrare
0
1
0
0
1
0
1
1
2

Ieșire
A
U
E

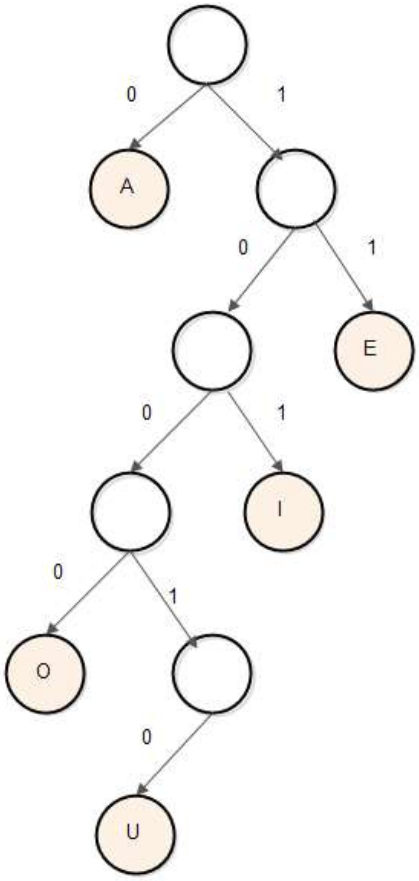
Soluție propusă:

Rezolvarea poate folosi un arbore de codare care are în nodurile terminale (fără descendenți) simbolurile codate. Determinarea unui simbol se va face prin parcurgerea arborelui din rădăcină în stânga sau în dreapta conform bitului citit de la intrare, până la găsirea unui nod terminal. Figura 1.1 arata modul de construire a arborelui de codare.

Având în vedere că numărul de noduri este cunoscut, arborele poate fi implementat static, printr-un tablou ca în figura 1.2. Arborele binar are 6 niveluri și numărul de noduri este $2^0 + 2^1 + 2^2 + 2^3 + 2^4 + 2^5 = 63$. În nodurile frunză se vor scrie simbolurile codate, iar în restul nodurilor se va scrie caracterul ‘ ‘. Un nod este nod terminal atunci când are cei doi descendenți egali cu caracterul ‘ ‘.

Se presupune că șirul de la intrare reprezintă o codificare corectă.

Figura 1.3 descrie organigrama programului. Elementele primite la intrare se introduc într-o coadă (nu se cunoaște dimensiunea șirului de biți codificat). După citirea șirului de biți, acesta se va prelucra prin extragerea din coada, bit cu bit.



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

Figura 1.1. Arborele de codare

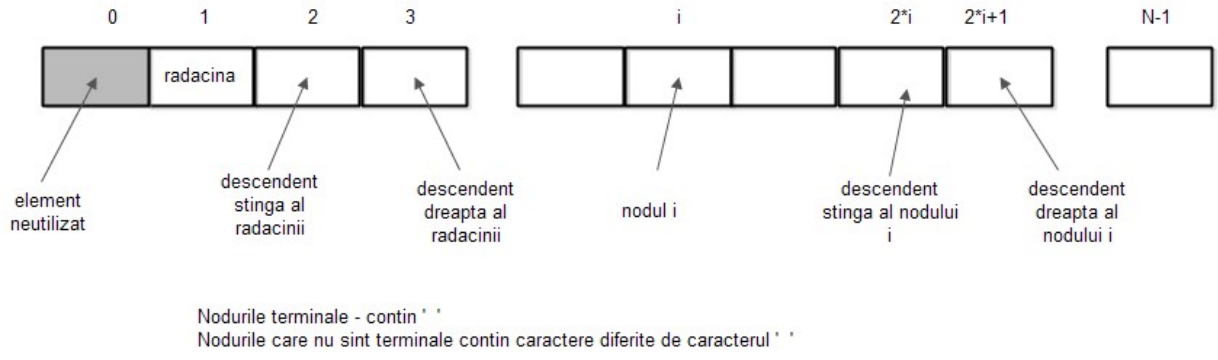


Figura 1.2. Reprezentarea statica a arborelui

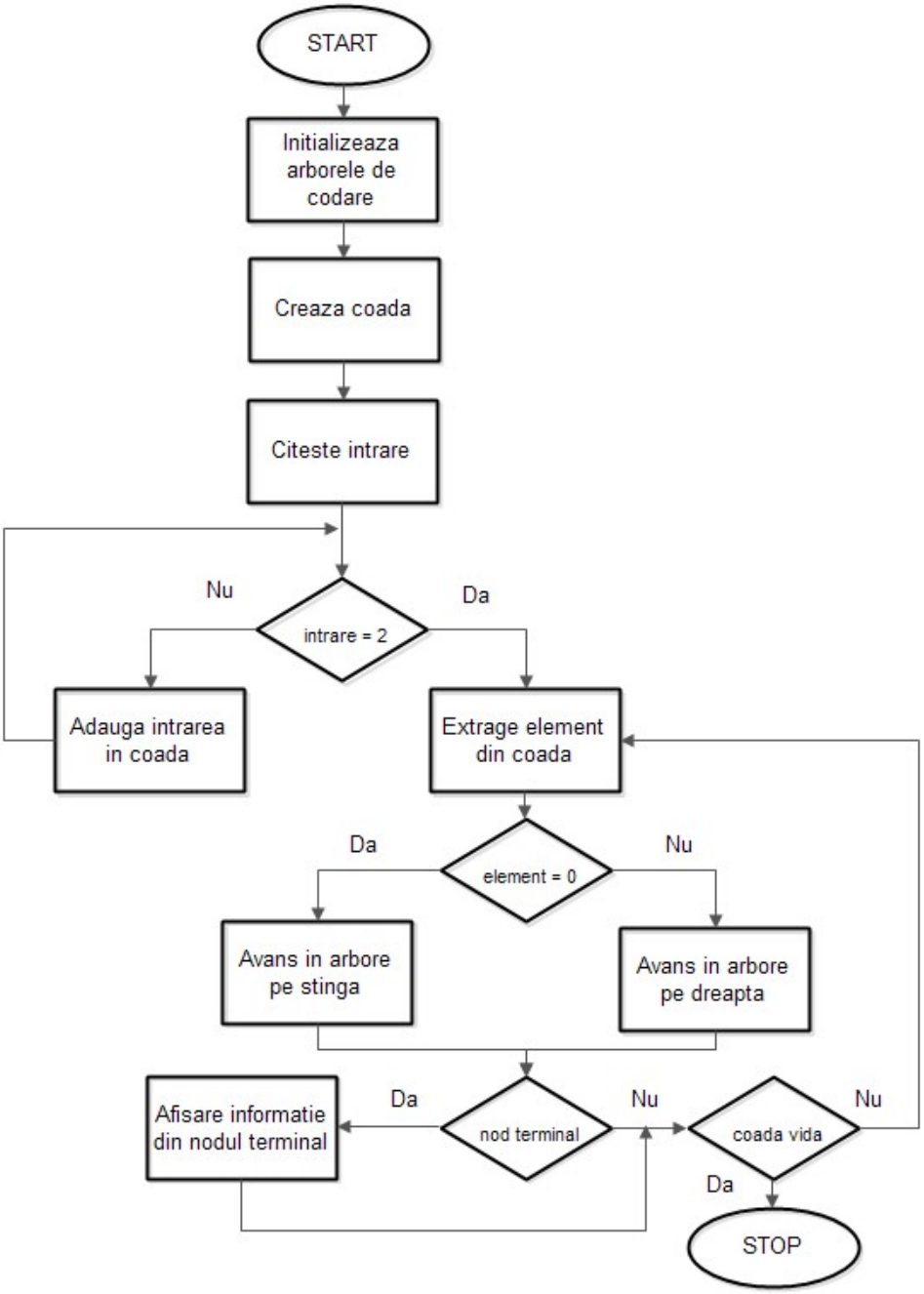


Figura 1.3. Organigrama programului

Codul asociat este următorul:

```
#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(șizeof(struct coada*));
   rear= (struct coada*)malloc(șizeof(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 2

Să se realizeze un program care afișează, în ordine descrescătoare elementele unei mulțimi de numere naturale nenule introduse de la tastatură. Elementele identice se vor afișa o singură dată. Introducerea unui 0 (care nu se va afișa) la intrare va termina secvența introdusă.

Exemplu:

Intrare:
2
4

6
8
2
3
8
7
5
0

Ieșire
8
7
6
5
4
3
2

Soluție propusă:

Se vor introduce elementele de la intrare într-o coadă dinamică (deoarece nu se cunoaște numărul elementelor). După terminarea introducerii secvenței de intrare, se va crea un arbore binar de căutare. Elementele identice vor fi ignorate în funcția de creare a arborelui. Afișarea în ordine descrescătoare se va face folosind o funcție de parcurgere a arborelui binar – în post ordine (RDL) conform figurii 2.1.

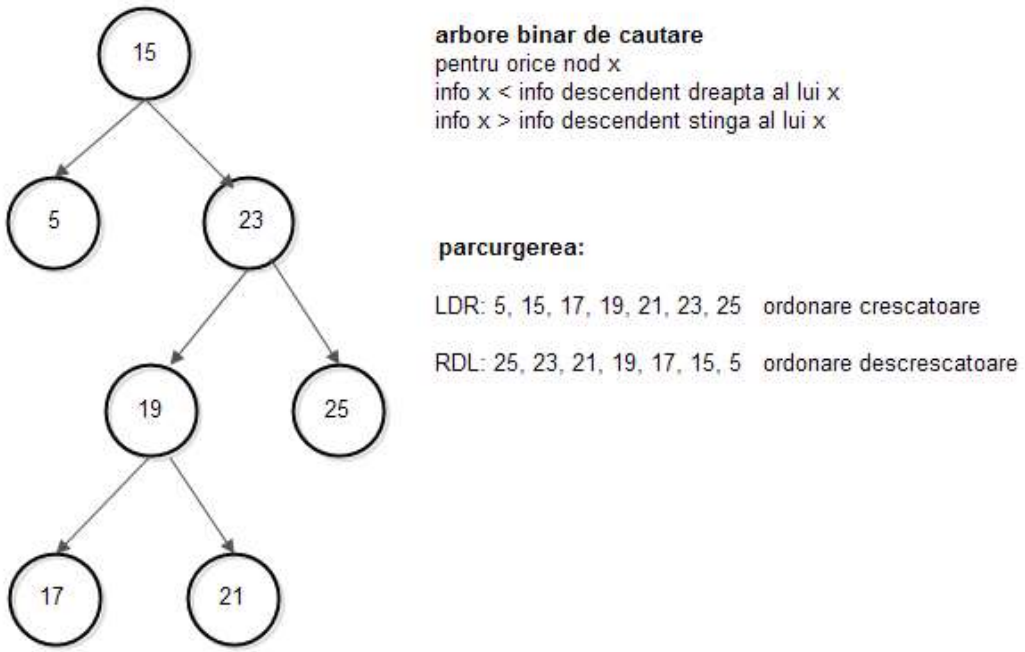


Figura 2.1. Parcurgerile în ordine (LDR) și post ordine (RDL)

Coada și arborele sunt implementate dinamic. Funcțiile pentru arborele binar sunt recursive.
Figura 2.2 indică structura nodurilor, iar figura 2.3 prezintă organigrama programului.

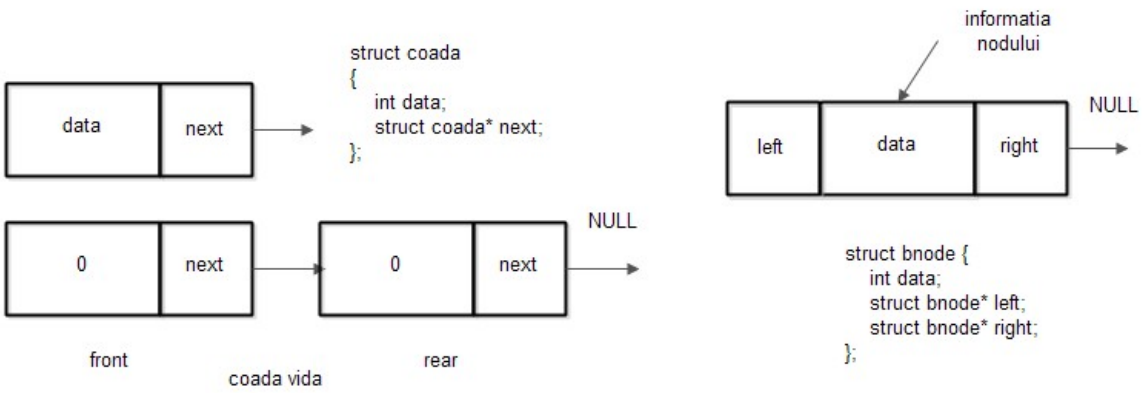


Figura 2.2. Structura nodurilor

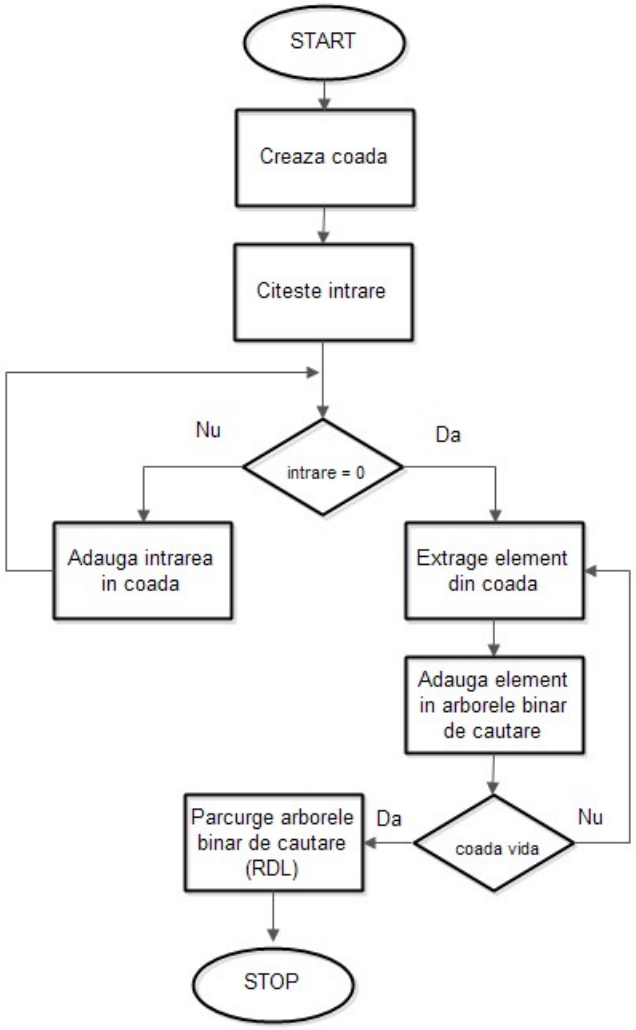


Figura 2.3. Organigrama programului

Codul propus este următorul:

```
#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(șizeof(struct coada*));
   rear= (struct coada*)malloc(șizeof(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 sunt 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 în post ordine (RDL)
void rdl(struct bnode* r)
{
   if(r!=NULL)
   {
      rdl(r->right);
      printf("%d\n", r->data);
      rdl(r->left);
   }
}
```

Problema 3

Să se realizeze un program care preia de la intrare șiruri de caractere și scrie la ieșire un șir de numere naturale nenule care reprezintă ordinea alfabetică a șirurilor de la intrare. Fiecare șir va fi introdus pe o linie separată. Nu se vor da la intrare șiruri identice. Introducerea șirului "stop" (care nu se va afișa) la intrare va termina secvența introdusă. Șirurile de caractere nu conțin spatii și au maxim zece de elemente.

Exemplu:

Intrare:

Popescu
Ionescu
Vașilescu
Avramescu
Florescu

stop

Ieșire:

4
3
5
1
2

Soluție propusă:

Elementele de la intrare sunt introduse într-o listă (inserare la sfârșit). Lista este creata dinamic, cu un nod fals *prim*. Elementele sunt apoi citite din listă și inserate într-un arbore binar de căutare (după câmpul *nume*). Nodurile pentru listă și arbore sunt ilustrate în figura 3.1. Pentru arbore, exista un câmp, *index*, pentru a se stoca poziția elementului *nume*.

Adăugarea în listă se face la sfârșitul acesteia, ca în figura 3.2.

După construirea arborelui, acesta se parcurge în ordine (LDR) și se actualizează câmpul *index* pentru fiecare nod. Datorita proprietății de ordonare crescătoare a parcurgerii LDR pentru un arbore binar de căutare, câmpul *index* va fi actualizat în ordinea alfabetica a câmpului *nume*.

După completarea câmpurilor *index*, se citește din nou lista (de la primul element) și se caută nodul în arbore. După găsirea acestuia, se afișează la ieșire câmpul *index* al nodului găsit.

Figura 3.3 prezintă organigrama programului.

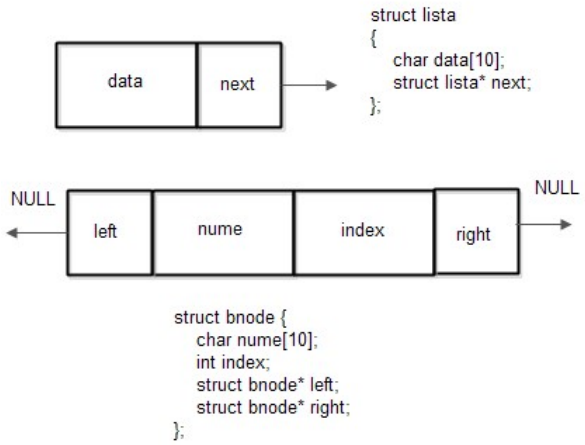


Figura 3.1. Nodurile listei și arborelui

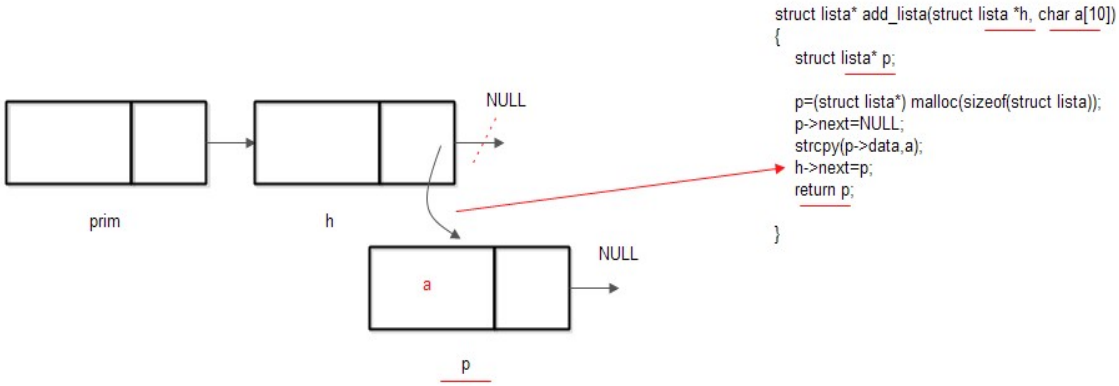


Figura 3.2. Adăugarea în listă

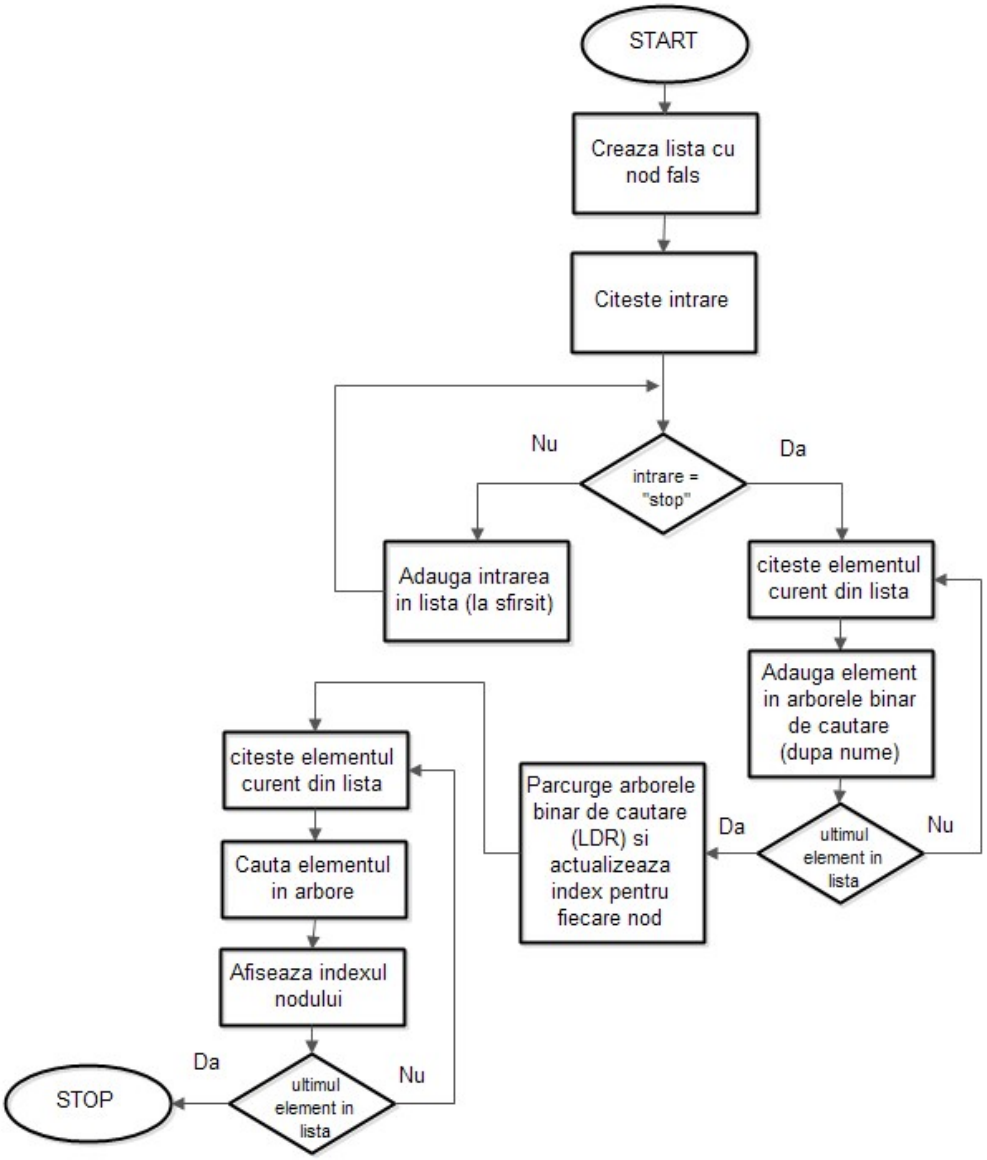


Figura 3.3. Organigrama programului

Codul propus este următorul:

```
#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 în arbore
// se cauta dupa cimpul nume, incepind cu nodul r
// intoarce nodul găsit
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 în ordine arborele binar
// actualizeaza cimpul index, în 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 în 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
// și avanseaza la urmatorul element
void read_lista(char x[10])
{
   strcpy(x, current->data);
   current=current->next;
}
```

Problema 4

Să se scrie un program care preia de la tastatură, pe linii separate, perechi de numere naturale (index) și șiruri numerice (nume). Programul va afișa la ieșire, pe câte o linie, șirurile de caractere în ordinea indexului asociat. Nu se pot da la intrare perechi identice și fiecare șir are un index unic. Introducerea unui 0 (care nu se va afișa) la intrare va termina secvența introdusă. Șirurile de numere au maxim zece caractere.

Exemplu:

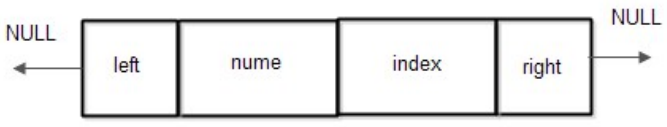
Intrare:
4
Ion
1
Vașile
8
Adrian
2
Marius

Ieșire:

Vașile
Marius
Ion
Adrian

Soluție propusă:

Se vor citi de la tastatura perechi *(index, nume*) și se va crea un arbore binar de căutare după cheia *index*. Nu e necesară o listă, se pot pune elementele direct în arbore. Apoi se traversează arborele în ordine (LDR) și se afișează câmpul *nume*. Structura nodurilor din arbore este ilustrată în figura 4.1. În figura 4.2 este prezentată organigrama programului.




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

Figura 4.1. Structura nodului arborelui

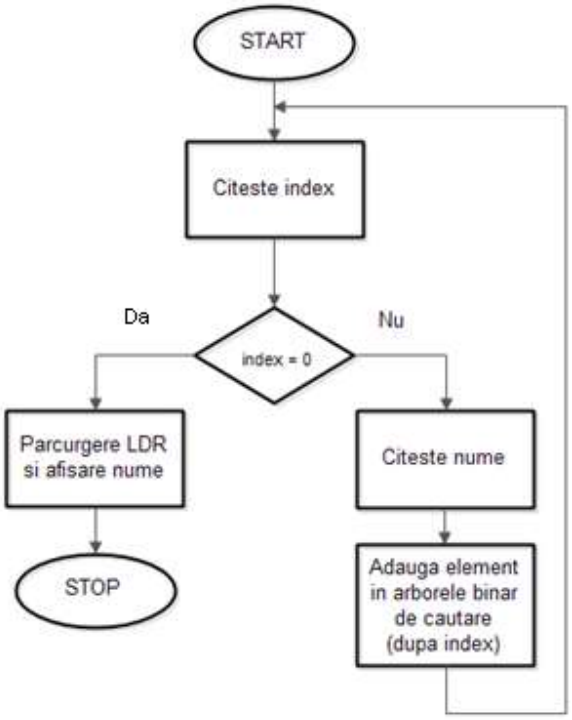


Figura 4.2. Organigrama programului

Codul propus este următorul:

```
#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(șizeof(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 și 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);
  }
}
```

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, 3 si 4, modul de implementare statica si dinamica a unui arbore binar si procedurile de inserare, căutare si traversare asociate unui arbore binar

Tema: Sa se propună alte soluții pentru problemele 1, 2, 3 si 4 si sa se evalueze in VPL soluțiile propuse.