Ministerul Educației al Republicii Moldova

Universitatea Tehnică a Moldovei

Catedra Informatica Aplicată

**RAPORT**

Lucrarea de laborator nr.4-5

*Tip abstract de date. Arbore binar*

A efectuat:

st. C-171 D.Melniciuc

A verificat:

dr., conf.univ., catedra IA M. Kulev

Chișinău 2018

**Lucrarări de laborator nr.4-5**

**Tema:** Implementarea tipului de date abstract “Arbore binar” in C

**Scopul lucrării:** obținerea deprinderilor practice de implementare a unui tip de date abstract (TDA) in limbajul C si anume a TDA “Arbore binar”

**Condiția problemei (sarcina de lucru)** [1]**:** De scris trei fișiere in limbajul C pentru implementarea (2 fișiere) si utilizarea (1 fișier – program cu funcția Main ) a TDA “Lista simplu înlănțuită”:

1. Fișierul antet cu extensia .h care descrie structura de date a elementului tabloului (conform variantei din lucrare precedenta) și prototipurile funcțiilor care asigură operațiunile de prelucrare a tabloului de structuri.

2. Fișier cu extensia .cpp (sau .c) care conține codurile (implementările) tuturor funcțiilor declarate în fișierul antet.

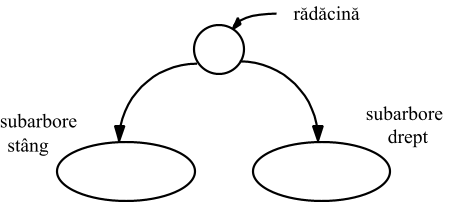
3. Fișierul utilizatorului - programul cu funcția main () pentru prelucrarea tabloului de structuri cu afișarea meniului de opțiuni pe ecran, si anume: alocarea memoriei dinamice pentru tablou, introducerea informației despre elementele tabloului de la tastatura, afișarea informației despre elementele tabloului pe ecran, căutarea elementului tabloului după unui câmp al structurii, modificarea câmpurilor elementului tabloului, interschimbarea a 2 elemente tabloului, sortarea elementelor tabloului după unui câmp, adăugarea unui element nou la sfârșitul tabloului, adăugarea unui element nou la începutul tabloului, inserarea unui element nou in tabloul la poziția indicata, ștergerea unui element al tabloului, scrierea informației despre elementele tabloului în fișier, citirea informației despre elementele tabloului din fișier, eliberarea memoriei alocate pentru tablou, ieșire din program.

**Varianta 5:** Marfa

**Mersul lucrării:**

**Noțiuni principale din teorie și metode folosite:**

În [informatică](http://ro.wikipedia.org/wiki/Informatic%C4%83), un **arbore binar** este un [arbore](http://ro.wikipedia.org/wiki/Arbore_(teoria_grafurilor)) în care fiecare nod are cel mult doi succesori (fii). De obicei, succesorii se numesc „nodul stânga” și „nodul dreapta”. Arborii binari sunt folosiți mai ales drept [arbori binari de căutare](http://ro.wikipedia.org/wiki/Arbore_binar_de_c%C4%83utare) sau și la [structurile de date](http://ro.wikipedia.org/wiki/Structur%C4%83_de_date) de tip [*heap*](http://ro.wikipedia.org/w/index.php?title=Heap&action=edit&redlink=1).



Implementare:

typedef struct arb

{

int info;

struct carte \*left, \*right;

}arb;

**Analiza datelor:**

**Functia main:**

**Descrierea**:Functia principala a programului.

1. date de intrare: nu sunt.

b) date de ieșire: nu sunt.

c) date intermediare:

command – variabilă simplă de tip întreg, opțiunea aleasă de utilizator;

m,m1,m2,m3,f - variabilă simplă de tip întreg, determină dacă funcția s-a efectuat cu succes;

t - variabilă locala de tip pointer la nod;

fname-variabila de tip pointer la char,denumirea fisierului pentru introducere

**int enq(nod\* inf):**

**Descrierea:**Functia pentru inserarea unui element in coada.

1. parametri primiți:

inf – pointer de tip nod, adresa nodului care se introduce în coadă;

1. date returnabile: functia returnaza 1 daca s-a locat memorie -1 invers;
2. date locale:   
   c– pointer de tip coadă, memoria elementului adăugat;

**nod\* deq(void):**

**Descriere:**Functia pentru eliminarea unui element din coada astfel obtinindui adresa lui.

1. parametri primiți: nu sunt;
2. date returnabile:  
   inf – pointer de tip nod (arbore), elementul șters;
3. date locale:  
   c – pointer de tip coadă, adresa în care se stochează elementul ce urmează să fie șters;

**int push(nod\* inf):**

**Descriere:**Functia pentru inserarea unui element in stiva.

1. parametri primiți:

inf – pointer de tip nod, adresa nodului care se introduce în stiva;

1. date returnabile: functia returnaza 1 daca s-a locat memorie -1 invers;date locale:   
   c– pointer de tip stivă, memoria elementului adăugat;

**nod \*pop(void):**

**Descriere:**Functia pentru eliminarea unui element din stiva astfel obtinindui adresa lui.

1. parametri primiți: nu sunt;
2. date returnabile:  
   inf – pointer de tip nod, elementul șters;
3. date locale:  
   c – pointer de tip stivă, adresa în care se stochează elementul ce urmează să fie șters;

**int show\_q(void):**

**Descriere:**Functri pentru afisarea liste de elemente cu ajutorul cozii in largime

1. parametri primiți: nu sunt;
2. date returnabile:   
   functia returneaza 1 daca elementele au fost gasite,0 daca lista este vida,
3. p,c – pointer de tip nod, variabile pentru prelucrarea listei;

**int show\_s(void):**

**Descriere:**Functri pentru afisarea liste de elemente cu ajutorul stivei in adincime

1. parametri primiți: nu sunt;
2. date returnabile:   
   functia returneaza 1 daca elementele au fost gasite,0 daca lista este vida,
3. p,c – pointer de tip nod, variabile pentru prelucrarea listei;

**int size\_q(void):**

**Descriere:**Functia pentru afisarea la ecran la marimii arborelui cu ajutorul cozii

1. parametri primiți: nu sunt;
2. date returnabile:   
   n – variabilă simplă de tip întreg, mărimea arborelui;
3. date intermediare:  
   c,p – pointeri de tip nod, variabile pentru prelucrarea listei;

**int size\_s(void):**

**Descriere:**Functia pentru afisarea la ecran la marimii arborelui cu ajutorul stivei

1. parametri primiți: nu sunt;
2. date returnabile:   
   n – variabilă simplă de tip întreg, mărimea arborelui;
3. date intermediare:  
   c,p – pointeri de tip nod, variabile pentru prelucrarea listei;

**int create\_q(void):**

**Descriere:**Functia pentru crearea arborelui cu ajutorul cozii.

1. parametri primiți: nu sunt;
2. date returnabile: returneaza 1 daca elementul a fost creat cu succes,0 invers;
3. date intermediare:  
   f – variabilă simplă de tip întreg, răspunsul utilizatorului.

c,p – pointeri de tip carte(arbore), variabile pentru prelucrarea listei;

**int create\_s(void):**

**Descriere:**Functia pentru crearea arborelui cu ajutorul stivei.

1. parametri primiți: nu sunt;
2. date returnabile: returneaza 1 daca elementul a fost creat cu succes,0 invers;
3. date intermediare:  
   f – variabilă simplă de tip întreg, răspunsul utilizatorului.

c,p – pointeri de tip carte(arbore), variabile pentru prelucrarea listei;

**nod\* search\_q(char\* fname):**

**Descriere:**Functia pentru cautarea unui element dupa un cimp al structurii cu ajutorul cozii.

1. parametri primiți:fname variabila de tip ponter la char,denumirea elementrului pentru cautare;
2. date returnabile:   
   p - pointer de tip nod,adresa elementului găsit;
3. date locale:  
   p,c– pointer de tip nod, variabile pentru prelucrarea listei;

**nod\* search\_s(char\* fname):**

**Descriere:**Functia pentru cautarea unui element dupa un cimp al structurii cu ajutorul stivei.

1. parametri primiți:fname variabila de tip ponter la char,denumirea elementrului pentru cautare;
2. date returnabile:   
   p - pointer de tip nod,adresa elementului găsit;
3. date locale:  
   p,c– pointer de tip nod, variabile pentru prelucrarea listei.

**int freemem(void):**

1. parametri primiți: nu sunt;
2. date returnabile: nu sunt.
3. date intermediare:  
   l – variabilă simplă de tip întreg, verifică dacă elementul s-a introdus în stivă;

c,p – pointeri de tip carte(arbore), variabile pentru prelucrarea listei;

**nod\* createRSD(void):**

**Descriere:**Functia pentru crearea arborelui recursiv RSD.

1. parametri primiți: nu sunt;
2. date returnabile:   
   c - pointer de tip nod,adresa rădăcinii;
3. date locale:  
   f – variabilă simplă de tip întreg, răspunsul utilizatorului;

**funcțiile void show(carte \*c):**

**Descriere:**Functia pentru afisarea elemntelor cu ajutorul recursei(RSD,RDS,SRD,DSR,SDR,DSR)

1. parametri primiți:  
   c-pointer de tip carte(arbore), adresa elementului care va fi afișat;
2. date returnabile: nu sunt;
3. date locale: nu sunt;

**nod\* searchRSD(nod \*c,):**

**Descriere:**Functia pentru cautarea unui elementru cu ajutorul recursiei.

1. parametri primiți:  
   c-pointer de tip nod, adresa rădăcinii;  
   date returnabile:   
   c-pointer de tip carte, adresa elementului găsit;
2. date locale: nu sunt;

**int sizeRSD(carte \*c):**

**Descriere:**Functia pemtri afisareala ecran a mariii arborelui cu ajutorul recuriei.

1. parametri primiți:  
   c-pointer de tip nod, adresa rădăcinii;
2. date returnabile: mărimea arborelui;
3. date locale: nu sunt;

**int highRSD(carte \*c):**

**Descriere:**Functia pentru afsarea la ecran a naltimii arborelui cu ajutorul recursiei.

1. parametri primiți:  
   c-pointer de tip nodmadresa rădăcinii;
2. date returnabile:   
   r,l – variabile simple de tip întreg, înălțimea arborelui;
3. date locale: nu sunt;

**nod\* freememSRD(carte \*c):**

1. parametri primiți:  
   c-pointer de tip carte(arbore), adresa rădăcinii;
2. date returnabile: nu sunt;
3. date locale: nu sunt;

**Codul (textul) programului in limbajul C:**

***Fișierul lab.h:***

typedef struct nod

{

char name[40];

char distrib[40];

int ID;

int termen;

float pret;

struct nod \*left,\*right;

}nod;

nod \*root=NULL;

// coada

typedef struct elq{

nod \*adrnod;

struct elq \*next;

}elq;

elq \*first=NULL;

elq \*last=NULL;

// stiva

typedef struct els{

nod \*adrnod;

struct els\* prev;

}els;

els \*top=NULL;void find\_error(int ord);

int enq(nod \*inf);

nod\* deq();

int push(nod \*inf);

nod\* pop();

int creat\_q();

int creat\_s();

nod\* creat\_rsd();

int show\_q();

int show\_s();

void show\_rsd(nod \*c);

void show\_rds(nod \*c);

void show\_srd(nod \*c);

void show\_drs(nod \*c);

void show\_sdr(nod \*c);

void show\_dsr(nod \*c);

nod\* search\_q(char \*fname);

nod\* search\_s(char \*fname);

nod\* search\_rsd(nod\* c, char \*fname);

int size\_q();

int size\_s();

int size\_rsd(nod \*c);

int read\_file(char \*fname);

int write\_file(char \*fname);

void print(nod \*c,int k=0);

void freemem\_rsd(nod \*c);

int freemem\_q();

int freemem\_s();

int height\_rsd(nod \*c);

***Fișierul: lab.cpp:***

int inq(nod \*x)

{

elq \*c;

c=(elq\*)malloc(sizeof(elq));

if(!c) return 0;

if(!first){

first = c;

}

else last->next=c;

last=c;

c->next=NULL;

c->adrnod=x;

return 1;

}

nod\* delq()

{

elq \*c=first;

nod \*x;

if(c == last){

first=last=NULL;

} else first=c->next;

x=c->adrnod;

free(c);

return x;

}

int push(nod \*x)

{

els \*c;

c=(els\*)malloc(sizeof(els));

if(!c) return 0;

c->prev=top;

c->adrnod=x;

top=c;

return 1;

}

nod\* pop()

{

els \*c=top;

nod \*x;

top=c->prev;

x=c->adrnod;

free(c);

return x;

}

int create\_q()

{

int f;

nod \*c,\*p;

first=last=NULL;

printf("\nDoriti sa creati radacina arborelui (1/0)?: ");

scanf("%d",&f);

if(f)

{

c=(nod\*)malloc(sizeof(nod));

if(!c) return -1;

printf("\nIntroduceti datele radacinei:\n");

printf("Denumire: ");

scanf("%s", &c->name);

printf("Importator: ");

scanf("%s", &c->distrib);

printf("ID: ");

scanf("%d", &c->ID);

printf("Livrare(zile): ");

scanf("%d", &c->termen);

printf("pret: ");

scanf("%f", &c->pret);

if(!inq(c)) return -2;

root = c;

}

else root = NULL;

while(first)

{

p = delq();

printf("\nDoriti sa creati copilul sting al nodului %s (1/0)?: ",p->name);

scanf("%d",&f);

if(f)

{

c=(nod\*)malloc(sizeof(nod));

if(!c) return -3;

printf("\nIntroduceti datele copilului sting\n");

printf("Denumire: ");

scanf("%s", &c->name);

printf("Importator: ");

scanf("%s", &c->distrib);

printf("ID: ");

scanf("%d", &c->ID);

printf("Livrare(zile): ");

scanf("%d", &c->termen);

printf("pret: ");

scanf("%f", &c->pret);

p->left=c;

if(!inq(c)) return -2;

} else p->left=NULL;

printf("\nDoriti sa creati copilul drept al nodului %s (1/0)?: ",p->name);

scanf("%d",&f);

if(f)

{

c=(nod\*)malloc(sizeof(nod));

if(!c) return -3;

printf("\nIntroduceti datele copilului drept\n");

printf("Denumire: ");

scanf("%s", &c->name);

printf("Importator: ");

scanf("%s", &c->distrib);

printf("ID: ");

scanf("%d", &c->ID);

printf("Livrare(zile): ");

scanf("%d", &c->termen);

printf("pret: ");

scanf("%f", &c->pret);

p->right = c;

if(!inq(c)) return -2;

}

else p->right = NULL;

}

return 1;

}

int show\_q()

{

nod \*p,\*c;

first=last=NULL;

if(!root) return 0;

p=root;

if(!inq(p)) return -2;

while(first){

p=delq();

printf("\nNodul (%s)\n",p->name);

printf("Denumire: %s\n", p->name);

printf("Importator: %s\n", p->distrib);

printf("ID: %d\n", p->ID);

printf("Livrare(zile): %d\n", p->termen);

printf("Pret: %f\n", p->pret);

c=p->left;

if(c)

if(!inq(c)) return -2;

c=p->right;

if(c)

{

if(!inq(c)) return -2;

}

}

return 1;

}

nod\* search\_q(char \*fname)

{

nod \*p,\*c;

first=last=NULL;

if(!root) return NULL;

p=root;

if(!inq(p)) return NULL;

while(first){

p=delq();

if(!strcmp(p->name,fname)) return p;

c=p->left;

if(c){

if(!inq(c)) return NULL;

}

c=p->right;

if(c){

if(!inq(c)) return NULL;

}

}

return NULL;

}

int size\_q()

{

int s=0;

nod \*p,\*c;

first=last=NULL;

if(!root) return 0;

p=root;

if(!inq(p)) return -2;

while(first){

p=delq();

s++;

c=p->left;

if(c)

if(!inq(c)) return -2;

c=p->right;

if(c)

{

if(!inq(c)) return -2;

}

}

return s;

}

int freemem\_q()

{

nod \*p,\*c;

first=last=NULL;

if(!root) return 0;

p=root;

if(!inq(p)) return -2;

while(first)

{

p=delq();

c=p->left;

if(c)

if(!inq(c)) return -2;

c=p->right;

if(c)

if(!inq(c)) return -2;

free(p);

}

return 1;

}

int height\_q(nod \*c)

{

int l=0,r=0;

if(!c) return -1;

l=1+height\_q(c->left);

r=1+height\_q(c->right);

if(l>r) return l;

else return r;

}

int creat\_s()

{

nod \*p, \*c;

int f;

root=NULL;

top=NULL;

printf("\nDoriti sa creati radacina arborelui (1/0)? : ");

fflush(stdin); scanf("%d",&f);

if(f)

{

c=(nod\*)malloc(sizeof(nod));

if(!c) return -1;

printf("\nIntroduceti datele \n");

printf("Denumire: ");

scanf("%s", &c->name);

printf("Importator: ");

scanf("%s", &c->distrib);

printf("ID: ");

scanf("%d", &c->ID);

printf("Livrare(zile): ");

scanf("%d", &c->termen);

printf("pret: ");

scanf("%f", &c->pret);

if(!push(c)) return -5;

root=c;

while(top)

{

p=pop();

printf("\nDoriti sa creati copilul drept al nodului %s (1/0)?: ",p->name);

fflush(stdin);

scanf("%d",&f);

if(f)

{

c=(nod\*)malloc(sizeof(nod));

if(!c) return -3;

printf("\nIntroduceti datele\n");

printf("Denumire: ");

scanf("%s", &c->name);

printf("Importator: ");

scanf("%s", &c->distrib);

printf("ID: ");

scanf("%d", &c->ID);

printf("Livrare(zile): ");

scanf("%d", &c->termen);

printf("pret: ");

scanf("%f", &c->pret);

p->right = c;

if(!push(c)) return -5;

} else p->right = NULL;

printf("\nDoriti sa creati copilul sting al nodului %s (1/0)?: ",p->name);

fflush(stdin);

scanf("%d",&f);

if(f)

{

c=(nod\*)malloc(sizeof(nod));

if(!c) return -3;

printf("\nIntroduceti datele\n");

printf("Denumire: ");

scanf("%s", &c->name);

printf("Importator: ");

scanf("%s", &c->distrib);

printf("ID: ");

scanf("%d", &c->ID);

printf("Livrare(zile): ");

scanf("%d", &c->termen);

printf("pret: ");

scanf("%f", &c->pret);

p->left=c;

if(!push(c)) return -5;

}else p->left=NULL;

}

}

return 1;

}

int show\_s()

{

nod \*p,\*c;

top=NULL;

if(!root) return 0;

p=root;

if(!push(p)) return -5;

while(top){

p=pop();

printf("Nodul (%s)\n",p->name);

printf("Denumire: %s\n", p->name);

printf("Importator: %s\n", p->distrib);

printf("ID: %d\n", p->ID);

printf("Livrare(zile): %d\n", p->termen);

printf("Pret: %f\n\n", p->pret);

c=p->right;

if(c!=NULL){

if(!push(c)) return -5;

}

c=p->left;

if(c!=NULL){

if(!push(c)) return -5;

}

}

printf("\n");

return 1;

}

int size\_s()

{

nod \*p,\*c;

int s=0;

top=NULL;

if(!root) return 0;

p=root;

if(!push(p)) return -5;

while(top)

{

p=pop();

s++;

c=p->right;

if(c!=NULL)

if(!push(c)) return -5;

c=p->left;

if(c!=NULL)

if(!push(c)) return -5;

}

return s;

}

nod\* search\_s(char \*fname)

{

nod \*p,\*c;

top=NULL;

if(!root) return NULL;

p=root;

if(!push(p)) return NULL;

while(top)

{

p=pop();

if(!strcmp(fname,p->name)) return p;

c=p->right;

if(c!=NULL)

if(!push(c)) return NULL;

c=p->left;

if(c!=NULL)

if(!push(c)) return NULL;

}

return NULL;

}

int freemem\_s()

{

nod \*p,\*c;

top=NULL;

if(!root) return 0;

p=root;

if(!push(p)) return -5;

while(top)

{

p=pop();

c=p->right;

if(c!=NULL)

if(!push(c)) return -5;

c=p->left;

if(c!=NULL)

if(!push(c)) return -5;

free(p);

}

return 1;

}

int height\_s(nod \*c)

{

int l=0,r=0;

if(!c) return -1;

l=1+height\_s(c->left);

r=1+height\_s(c->right);

if(l>r) return l;

else return r;

}

nod\* creat\_rsd(nod\* parent)

{

nod \*c;

int f;

c=(nod\*)malloc( sizeof(\*c) );

if(!c) return NULL;

printf("\nIntroduceti datele\n");

printf("Denumire: ");

fflush(stdin); scanf("%s", &c->name);

printf("Importator: ");

scanf("%s", &c->distrib);

printf("ID: ");

scanf("%d", &c->ID);

printf("Livrare(zile): ");

scanf("%d", &c->termen);

printf("pret: ");

scanf("%f", &c->pret);

printf("\nDoriti sa creati copilul sting al nodului %s (1/0)?: ",c->name);

fflush(stdin); scanf("%d",&f);

if(f) c->left=creat\_rsd(c);

else c->left=NULL;

printf("\nDoriti sa creati copilul drept al nodului %s (1/0)?: ",c->name);

fflush(stdin); scanf("%d",&f);

if(f) c->right=creat\_rsd(c);

else c->right=NULL;

return c;

}

void show\_rsd(nod \*c)

{

if(!c) return;

printf("Nodul (%s)\n",c->name);

printf("Denumire: %s\n", c->name);

printf("Importator: %s\n", c->distrib);

printf("ID: %d\n", c->ID);

printf("Livrare(zile): %d\n", c->termen);

printf("Pret: %f\n\n", c->pret);

show\_rsd(c->left);

show\_rsd(c->right);

}

void show\_rds(nod \*c)

{

if(!c)

return;

printf("Nodul (%s)\n",c->name);

printf("Denumire: %s\n", c->name);

printf("Importator: %s\n", c->distrib);

printf("ID: %d\n", c->ID);

printf("Livrare(zile): %d\n", c->termen);

printf("Pret: %f\n\n", c->pret);

show\_rds(c->right);

show\_rds(c->left);

}

void show\_srd(nod \*c)

{

if(!c) return;

show\_srd(c->left);

printf("Nodul (%s)\n",c->name);

printf("Denumire: %s\n", c->name);

printf("Importator: %s\n", c->distrib);

printf("ID: %d\n", c->ID);

printf("Livrare(zile): %d\n", c->termen);

printf("Pret: %f\n\n", c->pret);

show\_srd(c->right);

}

void show\_drs(nod \*c){

if(!c){

return;

}

show\_drs(c->right);

printf("Nodul (%s)\n",c->name);

printf("Denumire: %s\n", c->name);

printf("Importator: %s\n", c->distrib);

printf("ID: %d\n", c->ID);

printf("Livrare(zile): %d\n", c->termen);

printf("Pret: %f\n\n", c->pret);

show\_drs(c->left);

}

void show\_sdr(nod \*c)

{

if(!c)

return;

show\_sdr(c->left);

show\_sdr(c->right);

printf("Nodul (%s)\n",c->name);

printf("Denumire: %s\n", c->name);

printf("Importator: %s\n", c->distrib);

printf("ID: %d\n", c->ID);

printf("Livrare(zile): %d\n", c->termen);

printf("Pret: %f\n\n", c->pret);

}

void show\_dsr(nod \*c)

{

if(!c)

return;

show\_dsr(c->right);

show\_dsr(c->left);

printf("Nodul (%s)\n",c->name);

printf("Denumire: %s\n", c->name);

printf("Importator: %s\n", c->distrib);

printf("ID: %d\n", c->ID);

printf("Livrare(zile): %d\n", c->termen);

printf("Pret: %f\n\n", c->pret);

}

void freemem\_rsd(nod \*c)

{

if(!c) return;

freemem\_rsd(c->left);

freemem\_rsd(c->right);

free(c);

}

nod\* search\_rsd(nod\* c, char \*fname)

{

nod\* t=NULL;

if(!c) return NULL;

if (!strcmp(c->name,fname)) return c;

t=search\_rsd(c->left,fname);

if (t) return t;

t=search\_rsd(c->right,fname);

if (t) return t;

return NULL;

}

int size\_rsd(nod \*c)

{

int n=0;

if(!c)

return n;

return 1 + size\_rsd(c->left) + size\_rsd(c->right);

}

int height\_rsd(nod \*c)

{

int l=0,r=0;

if(!c) return -1;

l=1+height\_rsd(c->left);

r=1+height\_rsd(c->right);

if(l>r) return l;

else return r;

}

***Fișierul main.c:***

int main()

{

nod \*t=NULL;

char fname[100];

int gxr = 0;

do{

start:

system("cls");

printf("\n |========================================================================================|\n");

puts(" | Coada (query) \t | Stiva (stack) | Recursie |");

printf(" |==============================|=============================|===========================|\n");

printf(" | [1] Crearea arborelui \t | [11] Crearea arborelui | [21] Crearea RSD |\n");

printf(" | [2] Afisarea arborelui \t | [12] Afisarea arborelui | [22] Afisarea RSD |\n");

printf(" | [3] Cautarea unui nod \t | [13] Cautarea unui nod | [23] Afisarea RDS |\n");

printf(" | [4] Marimea arborelui \t | [14] Marimea arberelui | [24] Afisarea SRD |\n");

printf(" | [5] Inaltimea arborelui\t | [15] Inaltimea arberelui | [25] Afisarea DRS |\n");

printf(" | [6] Eliberarea memoriei\t | [16] Eliberarea memoriei | [26] Afisarea SDR |\n");

printf(" | \t | | [27] Afisarea DSR |\n");

printf(" | \t | | [28] Marimea arborelui |\n");

printf(" | \t | | [29] Inaltimea arborelui |\n");

printf(" | \t | | [30] Cautarea unui nod |\n");

printf(" | \t | | [31] Eliberarea memoriei |\n");

printf(" |========================================================================================|\n\n >> ");

scanf("%d",&gxr);

puts(" ");

switch (gxr)

{

case 1: {

create\_q();

system("PAUSE");

break;

}

case 2: {

show\_q();

system("PAUSE");

break;

}

case 3: {

printf(" Introduceti numele de cautat: ");

fflush(stdin);

scanf("%s",&fname);

t=search\_q(fname);

if(t){

printf("Nodul (%s)\n",t->name);

printf("Importator: %s \n", t->distrib);

printf("ID: %d \n", t->ID);

printf("Livrare(zile): %d \n", t->termen);

printf("Pret: %f \n", t->pret);

} else {

printf("[-] Nu exista asa nume!\n");}

system("PAUSE");

break;

}

case 4: {

printf(" Nodurile arborelui = %d\n",size\_q());

system("PAUSE");

break;

}

case 5: {

printf("Inaltimea arborelui este %d \n", height\_q(root));

system("PAUSE");

break;

}

case 6: {

freemem\_q();

root=NULL;

printf("[+] Eliberarea memoriei a avut loc cu succes!\n");

system("PAUSE");

break;

}

case 11: {

creat\_s();

system("PAUSE");

break;

}

case 12: {

show\_s();

system("PAUSE");

break;

}

case 13: {

printf("Introduceti numele de cautat: ");

fflush(stdin);

scanf("%s",&fname);

t=search\_s(fname);

if(t){

printf("Nodul (%s)\n",t->name);

printf("Denumire: %s\n", t->name);

printf("Importator: %s\n", t->distrib);

printf("ID: %d\n", t->ID);

printf("Livrare(zile): %d\n", t->termen);

printf("Pret: %f\n\n", t->pret);

} else printf("[-] Nu exista asa nume!\n");

system("PAUSE");

break;

}

case 14: {

printf("Nodurile arborelui = %d\n",size\_s());

system("PAUSE");

break;

}

case 15: {

printf("Inaltimea arborelui = %d \n", height\_s(root));

system("PAUSE");

break;

}

case 16: {

freemem\_s();

root=NULL;

printf("[-] Eliberarea memoriei a avut loc cu succes!\n");

system("PAUSE");

break;

}

case 21: {

root = creat\_rsd(NULL);

system("PAUSE");

break;

}

case 22: {

show\_rsd(root);

system("PAUSE");

break;

}

case 23: {

show\_rds(root);

system("PAUSE");

break;

}

case 24: {

show\_srd(root);

system("PAUSE");

break;

}

case 25: {

show\_drs(root);

system("PAUSE");

break;

}

case 26: {

show\_sdr(root);

system("PAUSE");

break;

}

case 27: {

show\_dsr(root);

system("PAUSE");

break;

}

case 28: {

printf(" Nodurile arborelui = %d\n",size\_rsd(root));

system("PAUSE");

break;

}

case 29: {

printf("Inaltimea arborelui este %d \n", height\_rsd(root));

system("PAUSE");

break;

}

case 30: {

printf(" Introduceti numele de cautat: ");

fflush(stdin);

scanf("%s",&fname);

t=search\_rsd(root,fname);

if(t){

printf("Nodul (%s)\n",t->name);

printf("Denumire: %s\n", t->name);

printf("Importator: %s \n", t->distrib);

printf("ID: %d \n", t->ID);

printf("Livrare(zile): %d \n", t->termen);

printf("Pret: %f \n", t->pret);

} else {

printf(" Nu exista asa nume!\n");}

system("PAUSE");

break;

}

case 31: {

freemem\_rsd(root);

root=NULL;

printf("[+] Eliberarea memoriei a avut loc cu succes!\n");

system("PAUSE");

break;

}

case 0:{

finish:

system("pause");

system("cls");

return 0;

}

default:{

int zx = 0;

printf("[-] Gresit\n");

puts("[1] Continuam programul\n[2] Iesim\n>> ");

scanf("%d", &zx);

if (zx == 1) goto start;

else goto finish;

system("cls");

break;

}

}

}

while(gxr!=0);

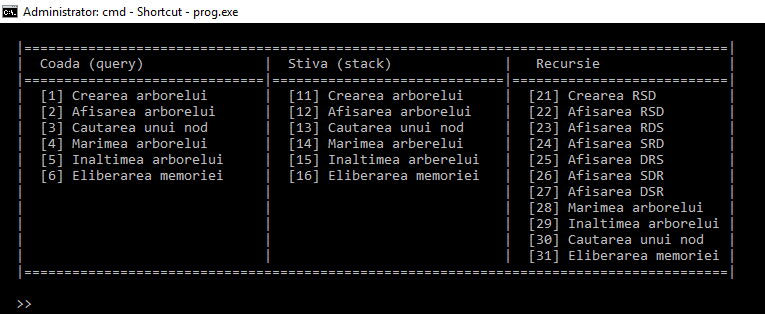
getch();

return 0;

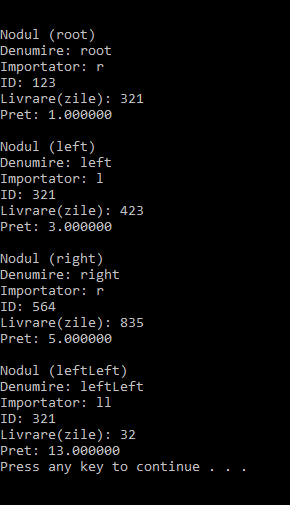
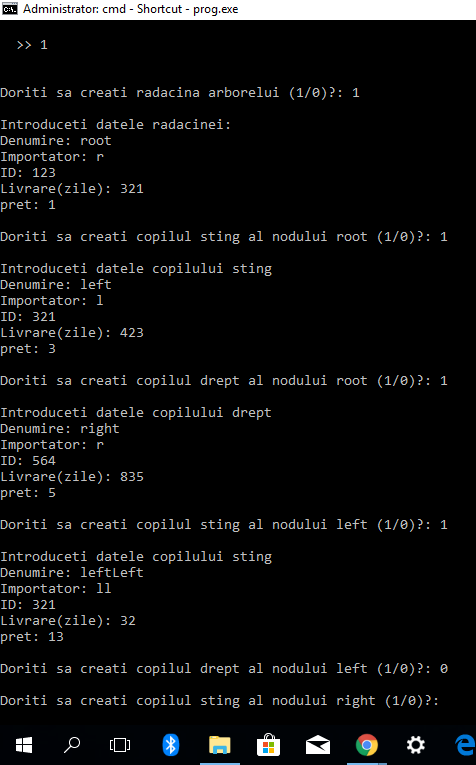
}

***Rezultatele:***

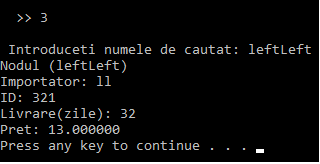
*Meniul principal:*



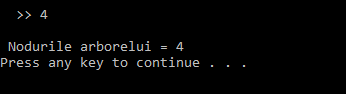
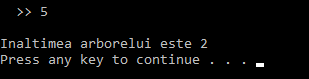
*Introducerea prin coada: Afisarea datelor:*



*Cautarea datelor:*

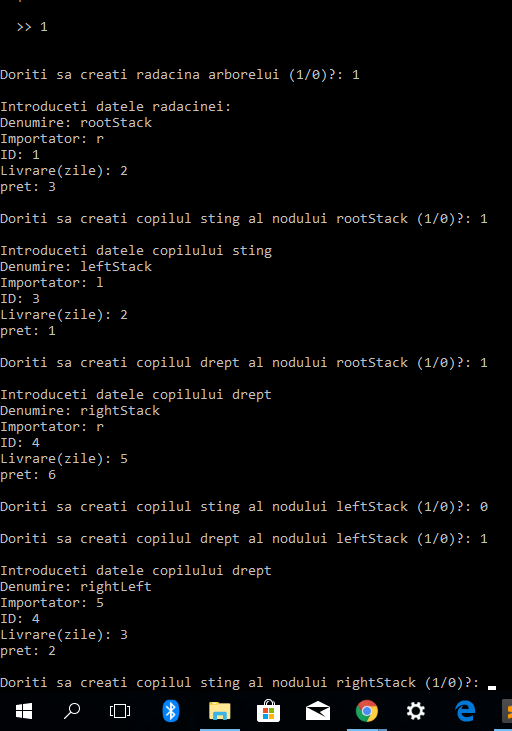


*Marimea arborelui: Inaltimea arborelui:*

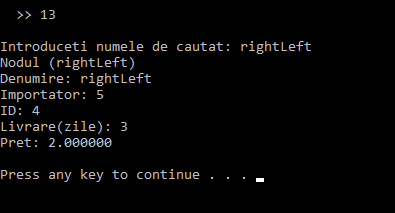


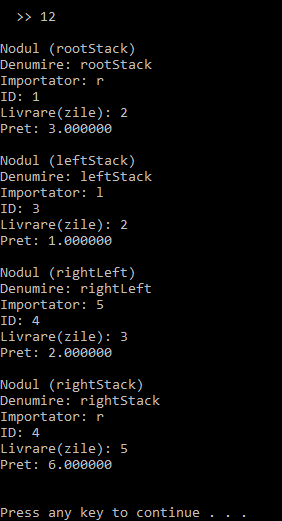
**Stiva**

*Introducerea prin stiva:*

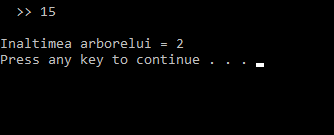
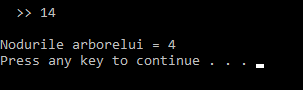


*Afisarea: Cautare:*

**

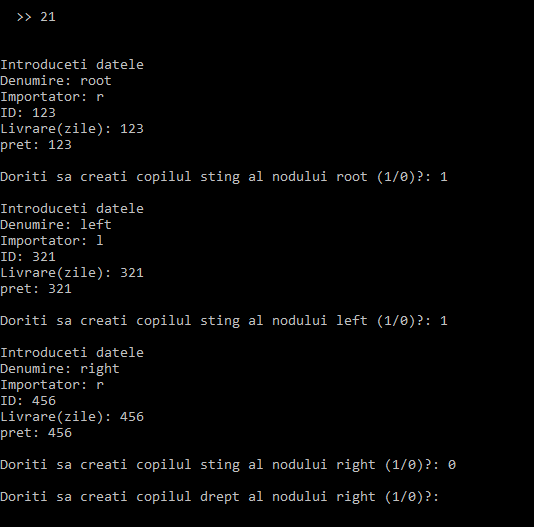


*Marimea arborelui: Inaltimea arborelui:*

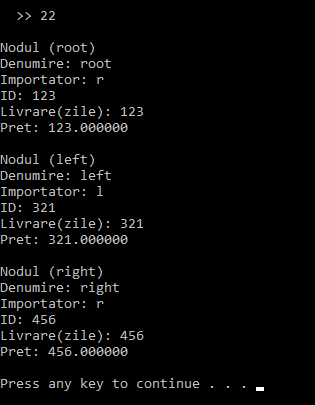
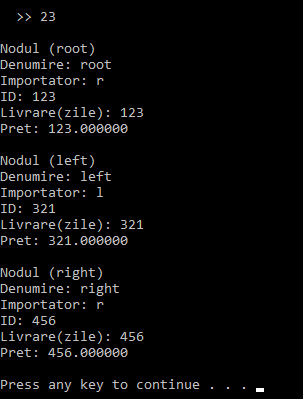


***Recursia:***

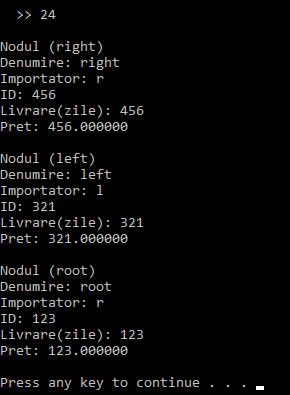
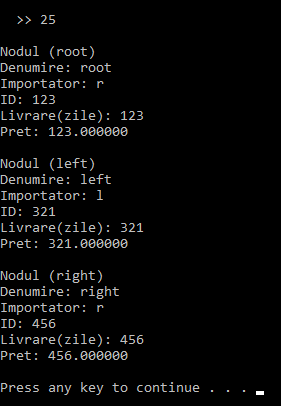
*Introducerea:*



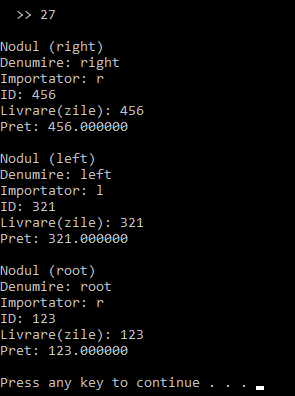
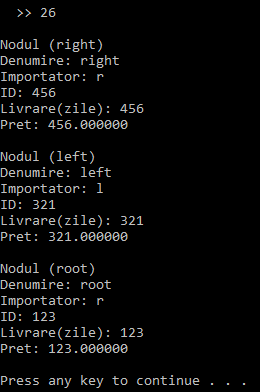
*Afisarea RSD Afisarea RDS*



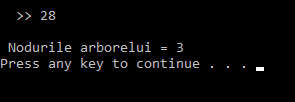
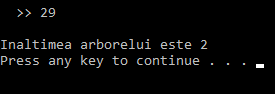
*Afisarea SRD Afisarea DRS*



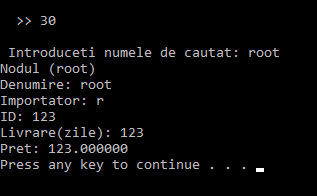
*Afisarea SDR Afisarea DSR*



*Marimea Inaltimea*

**

*Cautarea*

******

***Concluzii:***

În urma efectuării acestui laborator, am elaborat un program de prelucrare a unui arbore binar și am însușit modul de modul de repartizare a codului în fișiere, ceea ce permite ca codul să fie mai ușor de citit și redactat. Am elaborat un meniu prin care utilizatorul poate comunica cu programul elaborat și am implementat cele mai utile funcții de prelucrare a arborelui binar.

**Bibliografie:**

1. Conspectul prelegerilor cursului Programarea Calculatoarelor. Lector dr., conf. univ. M.Kulev. Chișinău: UTM, 2018.