Liste multiplu inlantuite. Structuri multilista

Cu ajutorul listelor, se pot implementa structuri de date dinamice foarte complexe.

Aplicatie1

Sa se scrie un program care realizeaza evidenta datelor de stare civila a unui grup de persoane; Pentru fiecare persoana se retin numele si varsta. Programul principal va primi si executa, in mod repetat, urmatoarele comenzi:

a = Adauga in evidenta o noua persoana a carei date se citesc
t = Cauta un nume in evidenta
s = Elimina o persoana din evidenta
n = Afiseaza alfabetic persoanele din evidenta
v = Afiseaza persoanele in ordinea virstei
x = terminare program

Din analiza problemei, rezulta ca este necesara o structura de date dinamica (numarul de persoane in evidenta variaza in timpul executiei programului). De asemenea, trebuie facute, la momente aleatoare de timp, afisari ale persoanelor din evidenta, ordonate dupa unul din cele 2 criterii diferite (nume si varsta). Implementarea evidentei sub forma de lista simplu inlantuita ordonata dupa nume sau lista simplu inlantuita ordonata dupa varsta este nesatisfacatoare, deoarece s-ar impune o operatie de reordonare a listei de fiecare data cand se comanda afisarea dupa celalalt criteriu.

O solutie este ca persoanele sa fie pastrate intr-o lista multuplu inlantuita. Aceasta lista are in noduri informatiile despre persoane, noduri care sunt inlantuite simultan in 2 liste ordonate, una dupa nume si una dupa varsta. Se subliniaza faptul ca aceleasi noduri fizice fac parte simultan din ambele liste ! (informatia despre fiecare persoana apare o singura data). Altfel, daca informatiile ar fi pastrate in 2 liste ordonate independente, redundanta informatiilor care se introduce astfel complica operatiile de adaugare si stergere a unei persoane in evidenta (aceste operatii ar trebui facute pe ambele liste). Structura de date rezultata este prezentata in exemplul din figura de mai jos.

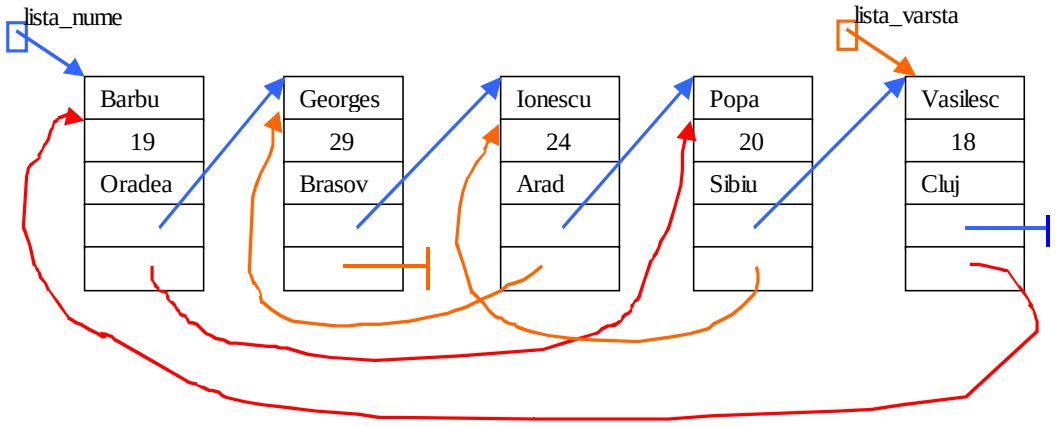

Figure 1. Lista multiplu inlantuita

In exemplul din figura, exista 5 noduri, care au doua campuri de inlantuire, unul care indica urmatorul nume si unul care indica urmatorul ca varsta. Prin aceste campuri de inlantuire, nodurile sunt inlantuite in 2 liste ordonate, `lista_nume` si `lista_varsta`. Traversand structura de date prin lista_nume urmand campul de inlantuire dupa nume rezulta ordinea nodurilor “Barbu” “ Georgescu“ ”Ionescu” “Popa” “Vasilescu”. Traversand structura de date prin lista_varsta urmand campul de inlantuire dupa varsta, rezulta ordinea nodurilor “Vasilescu” “Barbu” “Popa” “Ionescu” “Georgescu”.

Se defineste tipul structurat persoana pentru tipul nodurilor listei. Pe langa campurile de date (nume, varsta), aceasta structura contine doua campuri de inlantuire: `urm_nume` si `urm_varsta`. Campul `urm_nume` indica adresa urmatorului nod care are numele alfabetic dupa numele din nodul curent, iar campul `urm_varsta` indica urmatorul nod care are varsta mai mare decat varsta din nodul current.

Pentru accesul la structura de date, sunt necesari 2 pointeri spre cele 2 inceputuri ale listei - un pointer spre capul listei ordonata dupa nume, variabila globala `lista_nume`, si un pointer spre capul listei dupa varsta in variabila globala `lista_varsta`.

```
typedef struct pers
 {
    char *nume;
    int virsta;
    struct pers *urm_nume;
    struct pers *urm_varsta;
  } persoana;

persoana *lista_nume =NULL, *lista_varsta =NULL;
```

Dintre operatiile pe structura de date corespunzatoare evidentei persoanelor, operatiile de cautare si afisare se efectueaza ca operatii obinuite pe liste simplu inlantuite.

Functia de cautare a unei persoane cu nume dat parcurge nodurile incepand cu nodul indicat de inceputul `lista_nume`, avansand pe inlantuirea `urm_nume` intr-o lista ordonata:

```
persoana *cauta( char *n) {
       persoana *p;
       for( p = lista_nume; p !=NULL ; p =p->urm_nume)
             if( strcmp(p->nume,n ) == 0) return p;
       return NULL;
 }
```

Functiile de afisare in ordine alfabetica a numelor sau in ordine crescatoare a varstei sunt simple traversari ale listelor indicate de lista_nume, respective lista_varsta:

```
void afis_nume( void ) {
   persoana *p;
   for( p =lista_nume; p != NULL ; p =p->urm_nume)
        printf("%s %d\n",p->nume, p->virsta);
 }

 void afis_varsta( void ) {
```

```
    persoana *p;
    for( p =lista_varsta; p != NULL ; p =p->urm_varsta)
        printf("%s %d\n",p->nume, p->virsta);
}
```

Pentru adaugarea unei noi persoane cu numele `n` si varsta `v` in evidenta, se procedeaza in felul urmator: daca exista deja o persoana cu numele `n`, se afiseaza un mesaj de eroare si nu se mai adauga o noua inregistrare cu acest nume. Altfel, se aloca memoie pentru un nou nod, in care se copiaza informatiile referitoare la persoana. In final, acest nou nod trebuie inlantuit in structura de date existenta. Trebuie gasit locul de insertie a noului nod in lista ordonata dupa nume si in cea ordonata dupa varsta si realizate legaturile. Aceste operatii vor fi facute de functiile `adauga_nume` si `adauga_varsta`.

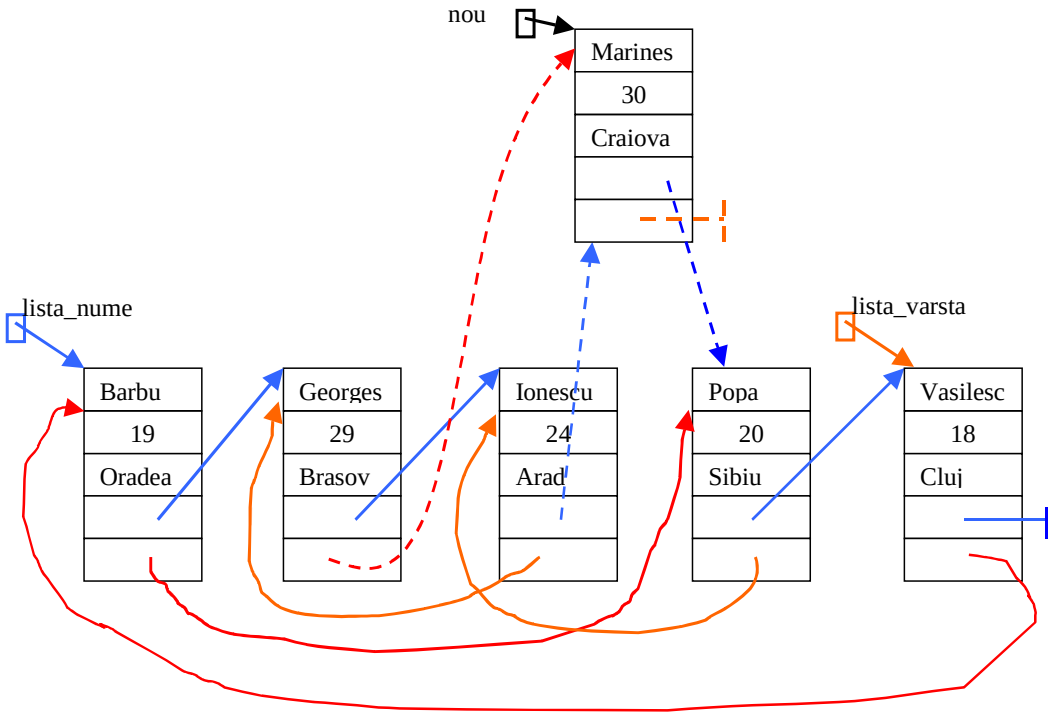

Figure 2. Realizarea inlantuirilor la adaugarea unei noi persoane in evidenta

```
void adauga(char *n, int v) {
       persoana *t;
       if(( t = cauta( n )) != NULL) {
                  printf("Eroare: %s exista deja \n", n);
            return;
       }
       else
       if(( t = (persoana*)malloc(sizeof(persoana))) == NULL ||
           ( t->nume = (char*)malloc(strlen(n)+1)) == NULL )  {
                  printf("Eroare : memorie insuficienta\n");
                  exit( 1 );
            }
            else {
                  strcpy( t->nume,n);
                  t->virsta = v;
                  adauga_nume(t);
                  adauga_varsta(t);
```

```
            }
 }
```

Cele 2 functii , `adauga_nume` si `adauga_varsta`, realizeaza inlantuirile nodului nou in cele 2 liste, `lista_varsta` respectiv `lista_nume`. Fiecare dintre aceste doua functii e o operatie simpla de insertie a unui nou nod in lista ordonata respectiva.

```
void adauga_nume(persoana *nou) {
   persoana *q1,*q2;

    for(q1=q2=lista_nume; q1!=NULL && strcmp(q1->nume, nou-
>nume)<0;
                                    q2=q1, q1=q1->urm_nume);

    nou->urm_nume = q1;
    if(q1 == q2 ) lista_nume=nou;
    else {
      q2->urm_nume = nou;
     }
  }

 void adauga_varsta( persoana *nou) {
   persoana *q1,*q2;

   for(q1=q2=lista_varsta; q1!=NULL && q1->virsta<nou->virsta;
                                    q2=q1,q1=q1->urm_varsta);

   nou->urm_varsta = q1;
   if(q1 == q2) lista_varsta=nou;
   else {
      q2->urm_varsta = nou;
      }
  }
```

Pentru scoaterea din evidenta a unei persoana cu nume dat, nodul trebuie scos din inlantuirile din ambele liste. In actuala implementare, scoaterea unei persoana in evidenta se face mai greoi, implicand doua operatii independente de cautare si stergere din cele 2 liste: dupa ce nodul a fost gasit (si scos) in lista dupa nume, mai trebuie facuta o cautare in lista dupa varsta, pentru a-l putea elimina si din aceasta . Aceasta a doua cautare e necesara pentru a identifica nodul predecessor in lista dupa varsta .

```
 void elimin( char *s){
  scot_nume( s );
  scot_varsta( s );
}


void scot_nume( char *l) {
   persoana *q1,*q2;

   for(q1=q2=lista_nume; q1 != NULL && strcmp( q1->nume,l)<0;
q2=q1, q1=q1->urm_nume);

   if( q1 != NULL && strcmp (q1->nume,l) ==0)
         if(q1 == q2 ) lista_nume=lista_nume->urm_nume;
```

```
        else {
                q2->urm_nume = q1->urm_nume;
                }
   else {
        printf(" Eroare: %s nu apare in evidenta\n",l);
        }
}


void scot_varsta( char *l) {
   persoana *q1,*q2;

   for(q1=q2=lista_varsta; q1 != NULL &&
            strcmp( q1->nume,l)!=0; q2=q1, q1=q1->urm_varsta);

   if( q1 != NULL && strcmp (q1->nume,l) ==0)
      if(q1 == q2 )  lista_varsta= lista_varsta->urm_varsta;
        else {
              q2->urm_varsta = q1->urm_varsta;
               }
   else {
        printf(" Eroare: %s nu apare in evidenta\n",l);
          }
}
```

Functiile prezentate pot fi apelate dintr-un program principal care functioneaza pe baza unui meniu din care utilizatiorul poate alege comenzile in mod interactiv.

Aplicatie2

Sa se scrie un program care realizeaza evidenta datelor privitoare la intreprinderile si salariatii unui judet. Pentru fiecare intreprindere, se retin denumirea, profilul activitatii, si lista persoanelor angajate. Pentru fiecare persoana se retin numele si varsta. Programul principal va primi si executa, in mod repetat, urmatoarele comenzi:

i = Adauga in evidenta o noua intreprindere a carei date se citesc
a = Adauga un nou angajat la o anumita intreprindere
s = Afiseaza alfabetic persoanele angajate la o anumita intreprindere
l = Afiseaza alfabetic toate intreprinderile
x = Terminare program

Structura de date avantajoasa in acest caz este o lista de intreprinderi, fiecare intreprindere continand o lista a persoanelor angajate.
Un exemplu de continut al acestei structuri de date este dat in figura urmatoare:

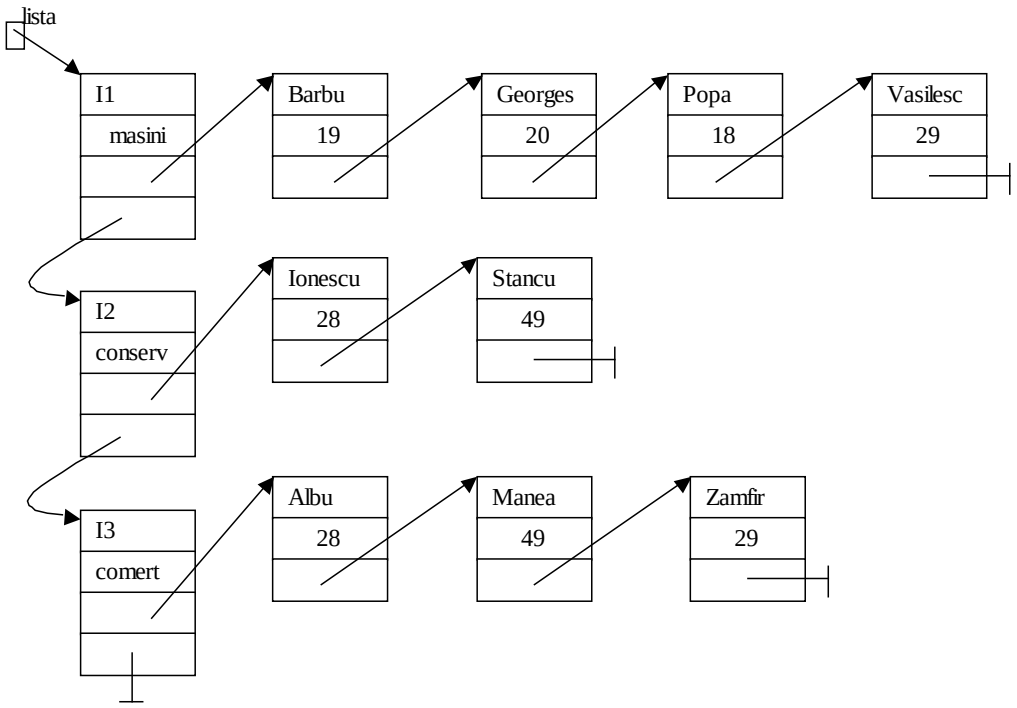

Figure 3. Structura multilista

Se observa ca sunt necesare doua tipuri de noduri: un tip de nod pentru lista intreprinderilor (lista principala) si un alt tip de nod pentru persoanele din listele de salariati.

Nodurile din listele de salariati sunt de tip structurat `persoana`, definit in felul urmator:

```
typedef struct pers
 {
    char *nume;
    int varsta;
    struct pers *urm;
   } persoana;
```

Nodurile din lista de intreprinderi sunt de tipul structurat `intreprindere`. Acest tip contine printre campurile sale de date, pe langa denumirea si profilul de activitate, si un pointer la lista salariatilor intreprinderii, campul `salariati` de tip pointer la `persoana`. Campul `urm` este campul de inlantuire obisnuita in lista de intreprinderi.

```
typedef struct intr {
      char * nume;
      char * profil;
      persoana * salariati;
      struct intr * urm;
} intreprindere;
```

Pentru a accesa intreaga structura de date, este nevoie de un pointer la primul nod al listei principale, lista de intreprinderi:

```
intreprindere * lista=NULL;
```

Pentru problema data, aceasta structura de tip multilista este mult mai avantajoasa decat, de exemplu, memorarea tuturor salariatilor intr-o lista unica de persoane, in care informatiile privind locul de munca al fiecarei persoane ar fi incluse direct in nodurile listei de persoane (informatiile referitoare la o intreprindere s-ar repeta in toate nodurile corespunzatoare persoanelor care sunt salariatii acestei intreprinderi. In cazul in care informatia privitoare la o intreprindere este replicata in mai multe noduri, o comanda de tipul “modifica profilul de activitate al unei intreprinderi” ar trebui sa actualizeze o multime de noduri, in timp ce pe structura multilista aceasta se face simplu, modificand un singur nod. De asemenea, structura multilista are si alte avantaje in aceasta problema, permitand generarea simpla de statistici/listari ale datelor pe fiecare intreprindere.

Operatiile pe lista de intreprinderi sunt operatii obisnuite pe lista simplu inlantuita.

Functia `adauga_intr` adauga in lista principala un nou nod, corespunzator unei intreprinderi cu denumirea si profilul dat.

```
void adauga_intr( char *nume, char *profil) {
    intreprindere *q1,*q2, *nou;

    for(q1=q2=lista; q1!=NULL && strcmp(q1->nume, nume)<0; q2=q1,
                                                  q1=q1->urm);

    if(( nou = (intreprindere *)malloc(sizeof(intreprindere)))
      == NULL ||
      ( nou->nume = (char*)malloc(strlen(nume)+1)) == NULL  ||
      ( nou->profil = (char*)malloc(strlen(profil)+1)) == NULL ){
                  printf("Eroare : memorie insuficienta\n");
                  exit( 1 );
             }
             else {
                  strcpy( nou->nume,nume);
                  strcpy(nou->profil,profil);
                  nou->salariati=NULL;
             }
    nou->urm = q1;
    if(q1 == q2 ) {
            lista=nou;
      }
    else {
      q2->urm = nou;
    }
 }
```

Functia de cautare a unei intreprinderi cu nume dat, returneaza un pointer la nodul din lista de intreprinderi unde s-a gasit sau NULL daca o intreprindere cu acest nume nu exista.

```
intreprindere * cauta_intr(char * n) {
      intreprindere *intr;

      for (intr=lista; intr!=NULL && (strcmp(intr->nume, n)<0);
      intr=intr->urm);
      if (intr !=NULL && strcmp(intr->nume, n)==0)
```

```
                return intr;
      return NULL;
}
```

Adaugarea unei persoane cu numele n in lista de salariati a intreprinderii cu denumirea i este o operatie mai complexa, care presupune urmatoarele etape:

- cauta in lista de intreprinderi nodul corespunzator intreprinderii i
- daca gaseste un asemenea nod intr, creaza un nou nod de tip persoana incare copiaza datele persoanei si pe care il insereaza in lista intr->salariati

Inserarea nodului nou in lista de salariati este facuta de o functie `adauga`.

```
void adauga_persoana(char *i, char *n, int v) {
      persoana *t; intreprindere *intr;

      intr=cauta_intr(i);

      if (intr==NULL) {
            printf("Eroare: intreprinderea %s nu exista \n", i);
            return;
      }
      else
      if (( t =(persoana*)malloc(sizeof(persoana))) == NULL ||
          ( t->nume = (char*)malloc(strlen(n)+1)) == NULL )  {
                printf("Eroare : memorie insuficienta\n");
                exit( 1 );
            }
            else {
                strcpy( t->nume,n);
                t->varsta = v;
                intr->salariati =adauga(intr->salariati,t);
            }
}
```

Functia `adauga`, de adaugare a unui nod nou intr-o lista de persoane are 2 parametrii: un pointer la inceputul listei in care se adauga si nodul nou care trebuie inserat. Functia returneaza un pointer la inceputul listei actualizate(dupa insertie). Este necesar ca functia adauga sa returneze lista ca si rezultat, pentru ca este posibil ca aceasta a se modifice in cadrul functiei (in urma insertiei unui nou nod chiar la inceput, se actualizeaza capul listei).

```
      persoana *adauga( persoana *lista_pers, persoana *nou) {
         persoana *q1,*q2;

         for(q1=q2=lista_pers; q1!=NULL && strcmp(q1->nume,
                                    nou->nume)<0; q2=q1, q1=q1->urm);

         nou->urm = q1;
         if(q1 == q2 ) return nou;
         else {
            q2->urm = nou;
            return lista_pers;
          }
  }
```

Pentru listarea datelor din evidenta, exista 2 functii: `listeaza_salariati` realizeaza afisarea unei liste de persoane care e indicata ca si parametru al acestei functii, si

functia `listeaza`, care realizeaza afisarea informatiilor despre toate intreprinderi, pentru fiecare intreprindere listand salariatii acesteia (se apeleaza functia listeaza_salariati pentru fiecare lista de salariati).

```
      void listeaza_salariati(persoana * l) {
            persoana * p;
            for (p=l; p!=NULL; p=p->urm)
                  printf("  %s    %2d \n", p->nume, p->varsta);

       }

       void listeaza(void) {
            intreprindere * intr;
            for (intr=lista; intr!=NULL; intr=intr->urm) {
                  printf("Intreprinderea %s cu profil de %s \n",
                        intr->nume, intr->profil);
                  printf("Lista salariatilor: \n ");
                  listeaza_salariati(intr->salariati);
            }
       }
```

Functiile prezentate pot fi apelate dintr-un program principal care functioneaza pe baza unui meniu din care utilizatiorul poate alege comenzile in mod interactive. Un exemplu este prezentat in final.

```
void afis_meniu(void){
   printf(" i = adauga o noua intreprindere \n");
   printf(" a = adauga un salariat la o intreprindere\n");
   printf(" l = listeaza tot \n");
   printf(" s = listeaza toti salariatii unei intreprinderi\n");
   printf(" x =  Terminarea programului \n\n");
 }


void main(void) {
 char cmd;
 int terminat=0;
 char s1[30], s2[30]; int v;
 intreprindere * intr;

 do {
   afis_meniu();
   cmd=getch();
   switch (cmd) {
   case 'i':  printf("Introduceti numele si profilul intreprinderii
:");
         scanf("%s %s", &s1, &s2);
         adauga_intr(s1,s2);
         break;
   case 'a':
         printf("Introduceti numele intreprinderii, numele si varsta
persoanei \n");
         scanf("%s %s %d", &s1, &s2,&v);
         adauga_persoana(s1, s2, v);
         break;
   case 'l': listeaza();
```

```
            break;
      case 's': printf("Introduceti numele intreprinderii \n");
            scanf("%s", &s1);
            intr=cauta_intr(s1);
            if (!intr) {
                  printf("Eroare: intreprinderea cu nume %s nu exista
\n", s1);
            }
            else {
                  listeaza_salariati(intr->salariati);
            }
            break;
      case 'x': terminat=1;
            break;
      default: printf("comanda gresita\n");
      }
   } while (!terminat);
}
```