Liste simplu inlantuite: neordonate si ordonate

Aplicatie

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

a= adauga o persoana noua in evidenta
c= afiseaza varsta unei persoane care se citeste
m= modifica adresa unei persoane a carei nume se citeste
d= sterge o persoana din evidenta
l= listeaza toate persoanele
x= terminare program

In aplicatii ca aceasta, numarul de elemente din lista nu este cunoscut initial, si in plus variaza in timpul executiei programului. De aceea, utilizarea unui tablou de structuri (fie acesta alocat static sau chiar dinamic) nu este o solutie eficienta pentru ca spatial alocat la tablou este fix si trebuie alocat de o dimensiune acoperitoare cazului cel mai defavorabil (numarul maximde elemente ce pot exista la un moment dat).

Daca in plus lista trebuie pastrata ordonata dupa un anumit criteriu (de exemplu ordonata alfabetic dupa nume), si intervin operatii de adaugare si eliminare din lista, acestea sunt greu de realizat in cazul alegerii unei structuri de tip tablou pentru memorarea elementelor listei.

Pentru aplicatia enuntata, se va ilustra utilizarea *listelor simplu inlantuite*.

Se defineste tipul structurat `persoana`, avand campurile `nume`, `varsta`, `adresa`; Campul `urm` este un pointer catre elementul urmator in lista. In felul acesta, trebuie cunoscuta numai adresa primului element de tip persoana, deoarece in continuare fiecare element contine adresa elementului urmator prin intermediul campului urm. Ultimul element din lista trebuie sa aiba campul urm egal cu NULL. Adaugarea si eliminarea unui element din lista se realizeaza prin modificari ale legaturilor elementelor vecine si spatial de memorie poate fi alocat dinamic pentru fiecare element, individual.

```
struct persoana {      /* tipul unui nod din lista */
      char *nume;
      int varsta;
      char *adresa;
      struct persoana *urm;
      };

struct persoana * lista; /* indicator la prima persoana din lista */
```

Figura prezinta o lista simplu inlantuita neordonata, impreuna cu variabila pointer `lista` care indica primul nod, capul listei.

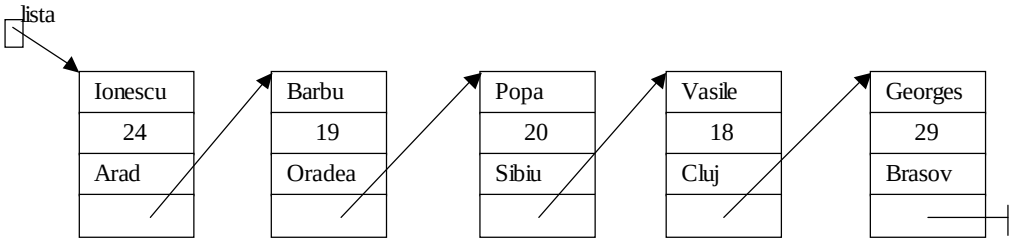

Figure 1. Exemplu de lista simplu inlantuita

S-a folosit si se va folosi in continuare aceasta notatie grafica simplificata pentru nodurile listei, reprezentand pentru simplitate si campurile care sunt pointeri la siruri in interiorul nodului. Crearea nodurilor se face prin alocare dinamica de memorie (cu malloc). Intai se aloca spatiu pentru noul nod, apoi se aloca spatiu pentru acele campuri ale nodului care sunt pointeri la siruri de caractere.

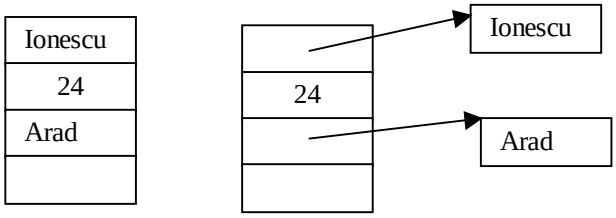

Figure 2. Tipul structurat persoana

Se vor discuta 2 variante de realizare a listei, intai ca si lista simplu inlantuita neordonata, apoi ca si lista ordonata dupa nume.

1. Lista neordonata

Adaugarea in lista neordonata

Adaugarea unei noi persoane in lista presupune intai alocarea de memorie pentru un nou nod. Acest nou nod va fi apoi inlantuit la inceputul listei, devenind noul cap al listei. Insertia noului nod se face la inceputul listei, aceasta fiind varianta cea mai simpla de adaugare de noduri, in cazul listelor neordonate.

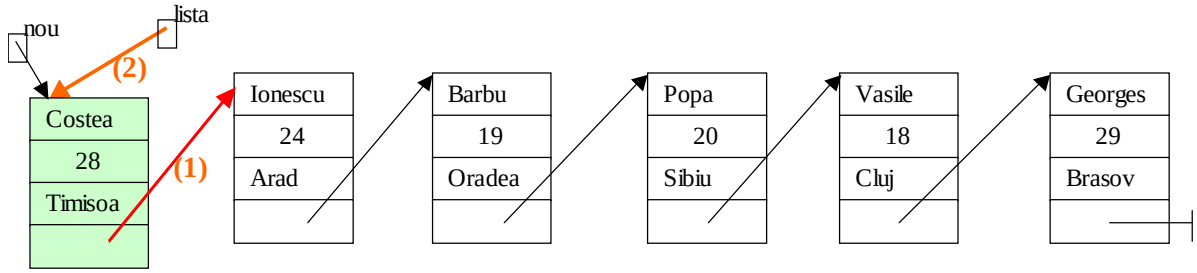

Figure 3. Adaugarea unui nod nou in capul listei

```
void adauga(void) {
/* se va  insera in lista la inceput */
char nume[100], adr[100];
```

```
int varsta;
struct persoana *nou;

printf("introd numele varsta adresa \n");
scanf("%s %d %s",&nume, &varsta, &adr);

/* aloca spatiu pentru noul nod */
if (((nou=(struct persoana *)malloc(sizeof(struct persoana)))==NULL) ||
    ((nou->nume=(char *)malloc((strlen(nume)+1)*sizeof(char)))==NULL)
||
    ((nou->adresa=(char *)malloc((strlen(adr)+1)*sizeof(char)))==NULL))
{
            printf("\n Eroare alocare memorie  \n");
            exit(1);
      }

/*copiaza informatiile in nod */
strcpy(nou->nume,nume);
strcpy(nou->adresa, adr);
nou->varsta=varsta;

/* introduce noul nod  in capul listei */
nou->urm=lista;
lista=nou;
}
```

Traversarea listei

Traversarea unei liste presupune executarea unei anumite operatii asupra tuturor nodurilor listei. Se utilozeaza un pointer curent p, care parcurge toate nodurile listei, de la primul nod pana la sfarsitul listei, indicat de inlantuirea nula.
Pentru afisarea tuturor persoanelor, se face o traversare a listei, cu tiparirea informatiilor din fiecare nod.

```
void list(void) {
struct persoana *p;
printf("Lista persoanelor este \n");
for (p=lista;p!=NULL ; p=p->urm)
  printf("%s %d \n",p->nume, p->varsta);
}
```

Cautarea in lista neordonata

Pentru afisarea varstei unei persoane sau modificarea adresei unei persoane, e nevoie ca intai sa se localizeze nodul respective printr-o operatie de cautare. Cautarea presupune gasirea unui nod care contine un anumit nume. Cautarea este o traversare mai speciala a listei: lista este parcursa pana cand se intalneste un nod avand campul nume egal cu cel cautat sau pana se ajunge la sfarsitul listei (la NULL), daca in lista nu e present un nod cu numele cautat.

```
struct persoana* cauta(char *nume) {
/* cauta daca este prezent in lista o persoana cu numele dat.
daca da, returneaza un pointer la nodul in care se gaseste;
daca nu, returneaza NULL */
struct persoana *p;
for (p=lista; p!=NULL; p=p->urm)
  if (strcmp(p->nume,nume)==0) return p;
return NULL;
}
```

```
 void varsta(void) {
 /* afiseaza varsta unei persoane */
 char nume[100];
 struct persoana *p;
 printf("Introd numele"); scanf("%s",&nume);
 p=cauta(nume);
 if (p!=NULL) {
    printf("Varsta lui %s este %d \n", p->nume, p->varsta);
    }
 else printf("%s nu exista \n",nume);
 }

void modifica(void) {
/* modifica adresa unui student */
char nume[100], adr[100];
struct persoana *p;
printf("Introd numele"); scanf("%s",&nume);
p=cauta(nume);
if (p!=NULL) {
   printf("Introd noua adresa");
   scanf("%s",&adr);
   free(p->adresa);
   /* aloca spatiu pentru adresa noua */
   if ((p->adresa=(char *)malloc((strlen(adr)+1)*sizeof(char)))==NULL) {
     printf("\n Eroare alocare memorie pentru adresa \n");
     exit(1);
   }
   strcpy(p->adresa, adr);
   }
else printf("%s nu exista \n",nume);
}
```

Stergerea unui nod din lista

Pentru eliminarea unui nod din lista, sunt folositi 2 pointeri care parcurg lista, q1 indica nodul current (cel care va trebui eliminat), si q2 nodul anterior lui. Este necesar ca parcurgerea listei sa fie facuta cu 2 pointeri pentru ca la eliminarea lui q1, inlantuirea urm a nodului anterior lui va fi modificata sa indice catre urmatorul nod dupa q1.

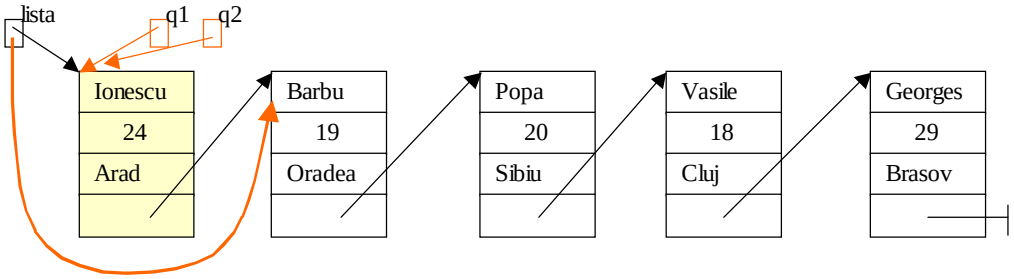

Figure 4. Stergerea unui nod din capul listei

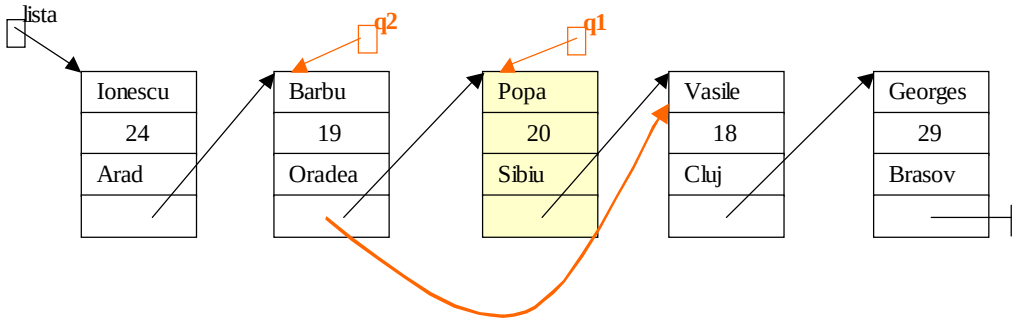

Figure 5. Stergerea unui nod din interiorul listei

```
void sterge(void) {
/* scoate din evidenta o persoana */
char nume[100];
struct persoana *q1,*q2;
printf("Introd numele "); scanf("%s",&nume);
for (q1=q2=lista; q1!=NULL; q2=q1, q1=q1->urm)
   if (strcmp(q1->nume,nume)==0) {
         if (q1==lista) { /* sterge capul listei */
               lista=q1->urm;
         }
         else { /* sterge un nod din interiorul listei */
               q2->urm=q1->urm;
         }

        free(q1->nume);
        free(q1->adresa);
      free(q1);
      printf("\n am sters %s",nume);
      return;
      }
  /* aici se ajunge daca numele nu a fost gasit */
printf("Stergere: %s nu a fost gasit in lista \n",nume);
}
```

Programul principal

Programul principal va consta dintr-un ciclu in care accepta comenzi si executa operatiile corespunzatoare.

```
void afis_meniu(void){
printf("\n\nAlegeti optiunea: \n");
printf("m= modifica adresa unei pers\n");
printf("v= afla varsta unei persoane \n");
printf("a= adauga o noua persoana\n");
printf("d= sterge o persoana din evidenta\n");
printf("l= listeaza toate persoanele \n");
printf("x= terminare program\n");
}

void main(void) {
 char cmd;
 int terminat=0;

 do {
```

```
 afis_meniu();
 cmd=getch();
 switch (cmd) {
   case 'm': modifica(); break;
case 'v': varsta(); break;
   case 'a': adauga_ordonat(); break;
   case 'd': sterge(); break;
   case 'l': list(); break;
   case 'x': terminat=1; break;
   default: printf("comanda gresita\n");
   }
} while (!terminat);
}
```

2. Lista ordonata

Daca afisarile listei de persoane trebuie facute in ordinea alfabetica a numelui, cel mai bine este ca lista sa fie pastrata in permanenta ordonata alfabetic dupa nume. In plus, faptul ca lista este ordonata creste performanta operatiilor de cautare dupa nume care se fac in lista. Campul care serveste la identificarea nodurilor si asupra caruia opereaza criteriul de ordonare este numit campul cheie (cum este campul nume in aplicatia in discutie).

Cautarea unui element intr-o lista ordonata

Cautarea unui element cu o cheie data (nume) intr-o lista ordonata se face traversand lista atata timp cat nu s-a ajuns la sfarsitul listei (`p!=NULL`) si cheile din nodurile curente sunt mai mici decat cheia cautata (`strcmp(p->nume, nume)<0`). In cazul in care nu exista in lista un nod cu cheia cautata, de cele mai multe cazuri nu se mai ajunge cu parcurgerea listei pana la sfarsitul ei, ci cautarea se opreste in momentul intalnirii primului nod care are cheia mai mare decat cheia cautata.

```
struct persoana* cauta_ordonat(char *nume) {
/* cauta daca este prezent in lista o persoana cu numele dat.
daca da, returneaza un pointer la nodul in care se gaseste;
daca nu, returneaza NULL */
struct persoana *p;
for (p=lista; p!=NULL && strcmp(p->nume, nume)<0; p=p->urm);
if (p!=NULL && strcmp(p->nume,nume)==0) return p;
else return NULL;
}
```

Inserarea unui element intr-o lista ordonata

Adaugarea unui nou element la lista aflata instare ordonata se face astfel incat lista sa ramana ordonata si dupa adaugarea noului element. Pentru aceasta se face o cautare a pozitiei de inserare in lista: noul nod va fi inserat inaintea primului nod care are cheia mai mare decat cheia lui. Se utilizeaza un pointer curent q1 care avanseaza in lista, atata timp cat nodurile curente au chei mai mici decat noua cheie de inserat si nu s-a ajuns la sfarsitul listei. Daca q1 s–a oprit pe primul nod cu cheie mai mare decat cheia de inserat, noul nod trebuie inserat inaintea nodului q1. Este necesar deci accesul si la nodul anterior lui q1, pentru a putea reface legaturile. Se utilizeaza pentru aceasta tehnica parcurgerii listei cu 2 pointeri, q1 si q2, q2 fiind tot timpul cu un pas in urma lui q1. Daca nu s-a gasit nici un nod cu cheia mai mare sau egala cu cea noua, q1 ajunge pana la NULL si q2 la ultimul nod al listei, insertia se face la coada listei, urmand aceeasi secventa de refacere de legaturi. Un caz particular este inserarea in capul listei, cand noul nod are

cheia mai mica decat toate cheile existente in lista. Inserarea in capul listei se face la fel ca la listele neordonate.

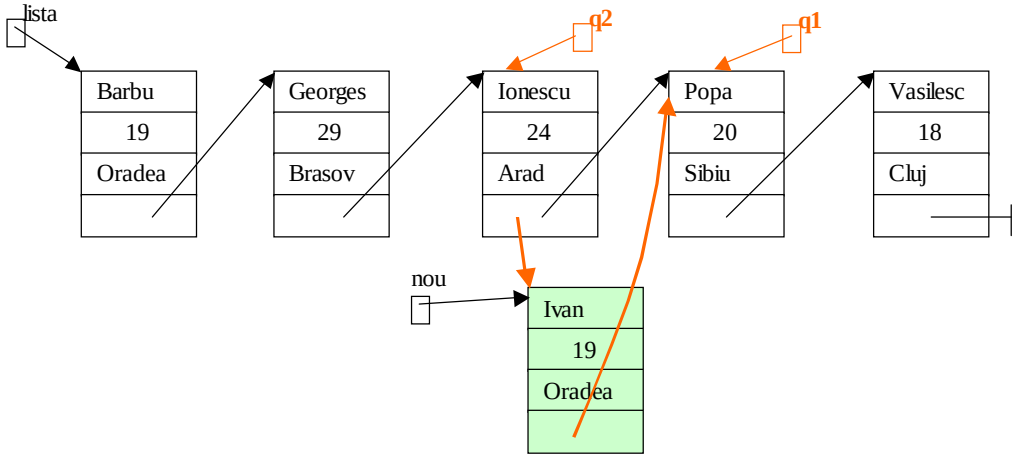

Figure 6. Inserarea unui nod in lista astfel incat aceasta sa ramana ordonata dupa nume

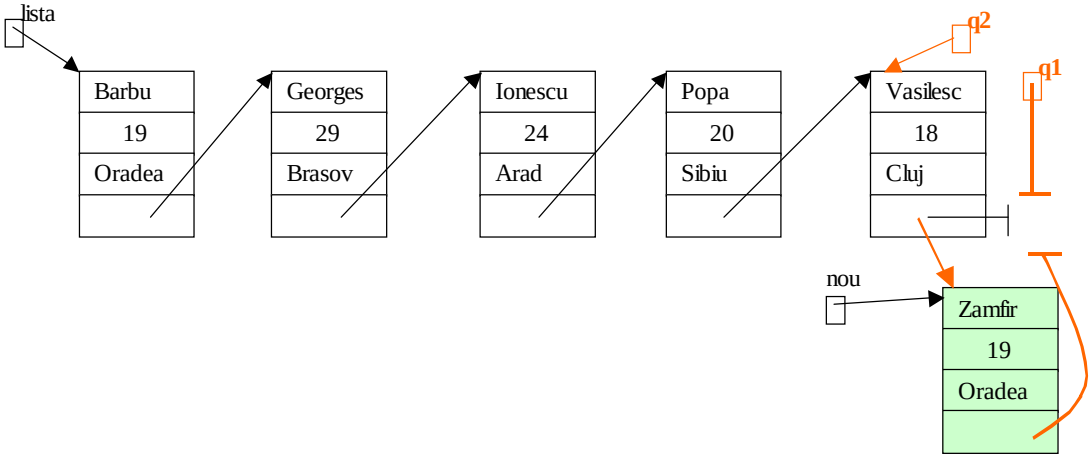

Figure 7. Insertia la sfarsit intr-o lista ordonata

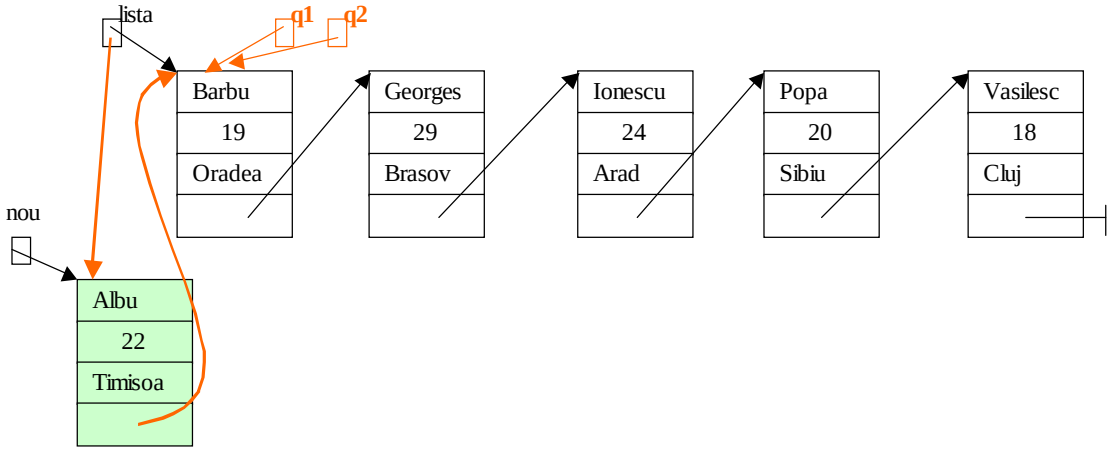

Figure 8,Insertia la inceput intr-o lista ordonata

```
void adauga_ordonat(void) {
/* se va  insera in lista in ordinea alfabetica a numelui */
char nume[100], adr[100];
```

```
int varsta;
struct persoana *nou, *q1, *q2;

printf("introd numele varsta adresa \n");
scanf("%s %d %s",&nume,  &varsta, &adr);

/* aloca spatiu pentru noul nod */
if (((nou=(struct persoana *)malloc(sizeof(struct persoana)))==NULL) ||
    ((nou->nume=(char *)malloc((strlen(nume)+1)*sizeof(char)))==NULL)
||
    ((nou->adresa=(char *)malloc((strlen(adr)+1)*sizeof(char)))==NULL))
{
            printf("\n Eroare alocare memorie  \n");
            exit(1);
      }

/*copiaza informatiile in nod */
strcpy(nou->nume,nume);
strcpy(nou->adresa, adr);
nou->varsta=varsta;

/* introduce noul nod  in lista,cauta locul de insertie */
for (q1=q2=lista; q1!=NULL && strcmp(q1->nume, nume)<0; q2=q1, q1=q1-
>urm);

if (q1!=NULL && strcmp(q1->nume, nume)==0) {
      printf("Eroare: %s apare deja in lista\n", nume);
      return;
}
else
if (q1!=q2) { /* inserarea nu se face la inceput */
      q2->urm=nou;
      nou->urm=q1;
}
else { /* inserarea se face la inceputul listei */
      nou->urm=lista;
      lista=nou;
}
}
```

Eliminarea unui element dintr-o lista ordonata

Eliminarea unui nod din lista ordonata seface la fel ca si in cazul listei neordonate, difera doar modul de loxcalizare a nodului cetrebuie sters, in cauare se tine cont defaptul ca lista e ordonata si cautarea se face doar atata timp cat noduri;ecurente au cheia mai mica decat cheia cautata.

```
void sterge_ordonat(void) {
/* scoate din evidenta o persoana */
char nume[100];
struct persoana *q1,*q2;
printf("Introd numele "); scanf("%s",&nume);

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

if (q1!=NULL && strcmp(q1->nume,nume)==0) {
   if (q1==lista) { /* sterge capul listei */
       lista=q1->urm;
   }
   else { /* sterge un nod din interiorul listei */
```

```
        q2->urm=q1->urm;
    }

  free(q1->nume);
  free(q1->adresa);
  free(q1);
  printf("\n am sters %s",nume);
  return;
}

/* aici se ajunge daca numele nu a fost gasit */
printf("Stergere: %s nu a fost gasit in lista \n",nume);
}
```