Laborator 5 - Lista simplu, dublu inlantuit

Liste : simplu inlantuit; dublu inlantuit; circulare

Laborator 5-1 : Liste simplu inlantuit

Lista simplu inlantuit este o structura de date dinamice. Spre deosebire de vectori, lista nu este alocata ca bloc omogen de memorie, ci ca elemente separate de memorie. Fiecare nod al listei contine:informatia utila si adresa urmatorului element din lista. Aceasta organizare permite acces secvential la elementele listei. Pentru accesarea listei trebuie cunoscuta adresa primului element; elementele urmatoare sunt accesate parcurgand lista, dupa adresa elementului urmator, pana la final cand adresa ultimului element este *null*.

Structura unui nod are urmatoarea configuratie: informatii (date) si adresa catre elementul urmator, ca in figura 5.1

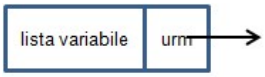

Figura 5.1 Structura unui nod al listei

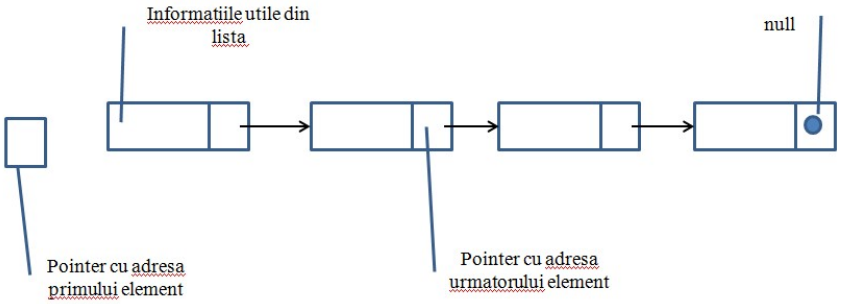

Figura 5.2 Lista simplu inlantuit

5.1 Structura listei

Pentru memorarea listei se foloseste o structura autoreferita. Acesta structura va avea forma:

```
 // Structura unui element  dintr-o lista simplu inlantuit

struct Nod {

   // datele efective memorate

    char nume[100];

    int an;

    // legatura catre nodul urmator

    Nod * urm;

 };
```

In cazul in care elemenul este ultimul din lista, pointerul urmator va avea valoarea NULL. Declararea listei se face sub forma:

```
   Nod * prim = NULL;
```

5.2 Operatii cu liste

5.2.1. *Parcurgere si afisare lista*

Lista este parcursa pornind de la pointerul spre primul element si avansand folosind pointerii din structura pana cand pointerul continut in variabila *urm* este NULL.

```
 // Parcurgere si afisare lista simplu inlantuit

begin
if list is not empty then
   while Nod not null do
         write nume, an
        next element
   endwhile
endif
end


void Afisare(Nod *el) {
```

```
 // cat timp mai avem elemente  in lista
     while (el != NULL) {
          // afiseaza elementul curent
           cout << el->nume << "  " << el->an << endl;
          // avanseaza la elementul urmator
          el = el->urm;
        }
   }
```

5.2.2 Inserare / Adaugare element

Inserarea unui element se poate face la inceputul sau la sfarsitul listei.

a) Inserare la inceput

Acesta este cazul cel mai simplu: trebuie doar alocat elementul, legat de primul element din lista si repozitionarea capului listei:

```
// Inserare element la inceputul unei  liste simplu inlantuit

begin
   new Nod el
   el->urm  <-  prim
   prim <- el
end

void adaugareInceput(Nod * & prim, char * nume, int an)

{

  // Alocare nod si initializare valoare

   Nod *el = new Nod;

   strcpy(el->nume , nume);

   el->an=an;

   // introducere nod in lista


   el->urm = prim;

   // mutarea capului listei

   prim = el;

}
```

b) Inserare / Adaugare la finalul listei

Pentru aceasta situatie trebuie parcursa lista de la primul pana la ultimul element din lista, si dupa aceea se adauga elementul nou si se realizeaza legatura de lista. De asemenea, trebuie avut in vedere cazul in care lista este vida.

```
// Inserare element la sfarsitul unei  liste simplu inlantuit

begin
  new Nod el
  while Nod not null do
        next element
   endwhile
   element->urm  <-  el
end

void adugareFinal( Nod * &prim, char *nume, int an)

{

// Alocare si initializare nod

   Nod *el = new Nod;

   strcpy(el->nume,nume);

   el->an=an;

   el->urm = NULL;

// verifica  daca  lista este vida

   if (prim == NULL)

        prim = el;

    else

        {

            // se parcurge lista pana la final
```

```
            Nod *list = prim;
            while (list->urm != NULL)
                    list = list->urm;
            // adaugam elementul nou in lista
            list->urm = el;
           }
 }
```

5.2.3. Regasire element (cautare)

Regasirea unui element dintr-o lista presupune parcurgerea listei pentru identificarea nodului in functie de criteriul de cautat. Cele mai uzuale criterii sunt cele legate de pozitia in cadrul listei si de informatiile utile continute de nodul acesteia. Rezultatul operatiei este reprezentat de adresa primului element gasit sau NULL in caz contrar.

Cautarea dupa valoare, spre exemplu dupa variabila *nume*.

Se parcurge lista pana la finalul acesteia sau pana la regasirea elementului:

// Cautare element dupa *nume*

```
begin Caut( nume)
   search elem_nume
   while nod is not null and el->nume is not elem_nume do
                next element
   endwhile
   write nod
end
```

```
Nod* cautNume(Nod * el, char *sir)
{
   while ( el  != NULL  &&  strcmp(el->nume,  sir) != 0)
            el = el ->urm;
return el;
}
```

apelul functiei trebuie sa transmita primul element al listei.

5.2.4. Stergere element

a) Stergerea unui element din lista

In acest caz avem nevoie de adresa elementului de dinaintea celui de sters. Stergerea elementului in fapt se realizeaza prin refacerea legaturilor fara elementul in cauza, dupa care se elibereaza memoria corespunzatoare elementului sters:

// sterge un element al listei primind ca parametru adresa sa

```
begin
  // elem <- caut (nume)
  while elem not equal NULL and elem->nume not equal nume then
       elem <- elem -> urm
   endwhile
   if elem not equal NULL then
       elem -> urm <- elem->urm->urm
       delete elem->urm
 endif
end
```

```
void stergereElement( Nod *  elem)
{
    // salvam referinta la elementul de sters
    Nod * elSterg = elem->urm;
    // refacem legaturile peste elementul sters
    elem->urm = elem->urm->urm;
     // si eliberam memoria
    delete elSterg;
}
```

b) stergerea unui element din lista dupa o valoare, spre exemplu *an*

Se cauta precedentul elementului si se foloseste functia de stergere element, anterioara:

```
void stergElemVal(Nod * &el, int nr)
{
```

```
  // daca este primul element, atunci
  // il stergem si mutam primul element
  if (el->an == nr)
   {
     Nod * elSterg = el;
     el = el->urm;
     delete elSterg;
   }
   // cautam anteriorul
   Nod * elem = el;
   while (elem->urm != NULL && elem->urm->an != nr)
       elem = elem->urm;
      // daca a fost gasit, atunci il stergem
     if (elem->urm != NULL)
       stergereElement(elem);
}
```

5.Discutii

Enumerati avantaje si dezavantaje ale listelor simplu inlantuit fata de masive (vectori) alocate static/dinamic.

Dati exemple de cazuri in care este eficenta flosirea listelor si cazuri in care este eficienta folosirea vectorilor.

5. Exercitii

5.1. Scrieti secventele pseudocod specifice pentru o lista simplu inlantuit:

- adaugare la inceput 2 elemente
- adaugare la final 2 elemente
- afisare continut lista rezultata
- stergere primul element
- cautare un nod dupa o valoare

5.2. Scrieti secventele pseudocod specifice pentru o lista dublu inlantuit:

- adaugare la inceput 2 elemente
- adaugare la final 2 elemente
- afisare continut lista rezultata
- stergere primul element
- cautare un nod dupa o valoare

5.3.* Sa se realizeze functiile pentru urmatoarele actiuni asupra listelor simplu inlantuit, descarcand codul sursa de aici:

1. Creati o functie de stergere element dupa valoarea variabilei *nume*, similar functiei: void stergElemVal(Nod * &el, int nr)
2. Concatenarea a doua liste simplu inlantuit.
3. Stergerea unei liste simplu inlantuit.
4. Copierea unei liste simplu inlantuit intr-o alta lista, element cu element.
5. Sortarea unei liste simplu inlantuit, utilizand criteriul *an.*
6. Gasirea elementului aflat pe pozitia i de la sfarsitul listei.
7. Inversarea unei liste prin modificarea legaturilor.
8. Stabilirea simetriei unei liste.
9. Interschimbarea a doua elemente prin modificarea legaturilor.
10. Se considera o lista ce contine elemente care au ca informatie utila: numr, prenume, specializare, an, media. Scrieti si apelati functia care afiseaza studentii din anul 1 din specializarea informatica.

Last modified: Wednesday, 2 November 2022, 1:41 PM

« PREVIOUS ACTIVITY
C5 - Liste particulare

NEXT ACTIVITY
Tema - Laborator 5 »