Lista liniară simplu înlănțuită alocată dinamic

```
#include <iostream>
using namespace std;

//definim structura unui nod
struct nod
{
   int info;      //informatia utila
   nod* urm;      //informatia de legatura: adresa urmatorului nod
};

//adaugare la inceputul listei
void adauga_inceput(nod* &prim, int x)
{
   nod* nou=new nod;    //este creat un nod nou cu operatorul C++ new
   nou->info=x;         //informatia utila a noului nod este x
   nou->urm=prim;       //nodul nou va indica spre elementul prim
   prim=nou;            //noul nod devine elementul prim/capul listei
}

//adaugare la sfarsitul listei
void adauga_sfarsit(nod* &prim, int x) {
   //este creat un nod nou
   nod* nou=new nod;
   nou->info=x;
   nou->urm=NULL;

   if (prim==NULL) //lista este vida
      prim=nou; //nodul nou devine primul si ultimul element al listei
   else //lista nu este vida
      {
      nod* temp=prim;
      while (temp->urm!=NULL)
      { //parcurgem lista pana la ultimul nod
         temp=temp->urm;
      }
      temp->urm=nou; //ultimul nod se leaga de nodul nou
   }
}
```

```
//adaugare in interior (dupa un nod cu o anumita valoare)
void adauga_interior(nod* prim, int val, int x)
{
   nod* temp=prim;
   //parcurg lista si caut nodul cu valoarea val
   while (temp!=NULL && temp->info!=val)
      temp = temp->urm;

   if (temp!=NULL)
      {
      //am gasit nodul cu valoarea cautata
      nod* nou=new nod;
      nou->info=x;
      nou->urm=temp->urm; //nodul nou indica spre nodul care urma dupa temp
      temp->urm=nou; //temp indica acum spre nodul nou
   }
}

//stergerea primului element
void sterge_inceput(nod* &prim)
{
   if (prim==NULL)
      return;
   nod* d=prim;
   prim=prim->urm;
   delete d; //stergerea se face cu operatorul C++ delete
   return;
}

//stergerea ultimului element
void sterge_sfarsit(nod* &prim)
{
   //lista este vida
   if (prim==NULL)
      return;

   //lista are doar un element
   if (prim->urm==NULL)
   {
      delete prim;
      prim=NULL;
      return;
   }
```

```
    //lista are cel putin doua elemente
    nod* temp=prim;
    while (temp->urm->urm!=NULL)
    { //parcurgem lista pana la ultimul nod
          temp=temp->urm;
    }
    delete temp->urm;
    temp->urm=NULL;
}

//sterge din interior (dupa un nod cu o anumita valoare)
void sterge_interior(nod* prim, int val)
{
    nod* temp=prim;
    //cautam nodul cu valoarea val, aflat in fata nodului pe care vrem sa-l stergem
    while (temp!=NULL && temp->info!=val)
       temp = temp->urm;

    if (temp!=NULL && temp->urm!=NULL)
    {
       nod* d=temp->urm;
       temp->urm=temp->urm->urm; //ocolim nodul de sters
       delete d;
    }
}

//parcurgerea si afisarea listei
void afisare(nod* prim)
{
    nod* temp=prim;
    while (temp!=NULL)
    {
       cout<<temp->info<<"->";
       temp=temp->urm;
    }
    cout<<"NULL"<<endl;
}
```

```
//sterge toata lista
void sterge_tot(nod* &prim)
{
   nod* temp;
   //cat timp mai avem elemente in lista
   while (prim!=NULL)
   {
      //salvam adresa nodului curent pentru a-l putea sterge
      temp=prim;
      //mutam capul listei la urmatorul element
      prim=prim->urm;
      //eliberam memoria pentru nodul salvat in temp
      delete temp;
   }
}

int main() {
   nod* prim=NULL; //la inceput lista este vida

   adauga_sfarsit(prim, 10);
   adauga_sfarsit(prim, 20);
   afisare(prim); //Lista este: 10->20->NULL
   adauga_inceput(prim, 5);
   afisare(prim); //Lista este: 5->10->20->NULL
   adauga_interior(prim, 10, 15); //adaugam 15 dupa 10
   afisare(prim); //Lista este: 5->10->15->20->NULL

   sterge_sfarsit(prim);
   afisare(prim); //Lista este: 5->10->15->NULL
   sterge_interior(prim, 10);
   afisare(prim); //Lista este: 5->10->NULL
   sterge_inceput(prim);
   afisare(prim); //Lista este: 10->NULL
   sterge_tot(prim);
   afisare(prim); //Lista este: NULL

   return 0;
}
```