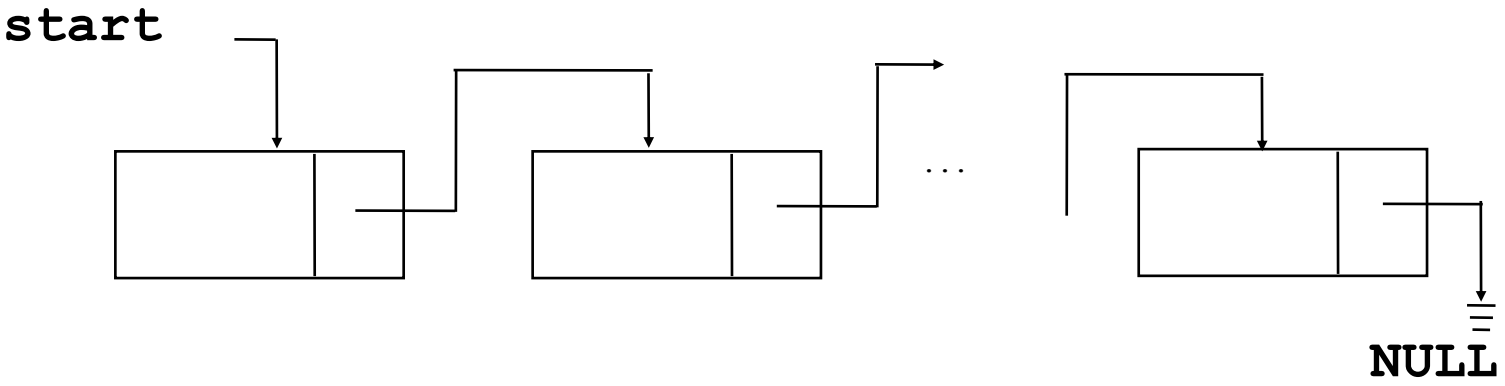
# Списък (List)

Списъкът е подредена последователност от еднотипни елементи без определена дисциплина за тяхно включване и изключване. Стекът, опашката и декът могат да се разглеждат като частен случай на структурата списък. Най-простият случай на структурата е този, когато всеки елемент на списъка съдържа указател към следващия елемент:



## Основните операции със структурата са:

- **Инициализиране** на списъка (`INIT`).
- **Включване** на елемент на списъка.

Поради липсата на дисциплина, операцията е възможна в няколко варианта:

2.1. включване в началото (`ADD_1`);

2.2. включване по средата (`ADD_2`);

2.3. включване в края (`ADD_3`)

- **Изключване/изтриване** на елемент (аналогично на предходната операция):

3.1. изтриване на първия елемент (`DEL_1`);

3.2. изтриване на междинен елемент (`DEL_2`);

3.3. изтриване на последния елемент (`DEL_3`);

Към изброените операции могат да се добавят и следните:

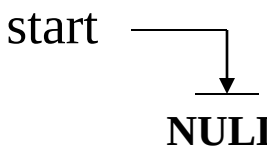
- **Проверка** за празен списък (`EMPTY`).
- **Търсене** на елемент (`SEARCH`);

Примерно описание на структурата списък:

```
struct elem
      {int key; elem *next;} *start;
```

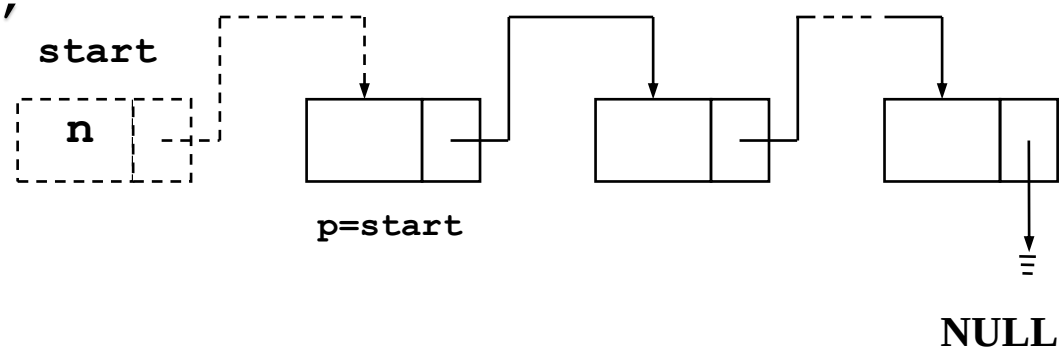
Инициализация на списък:

```
void init()
{ start=NULL;}
```

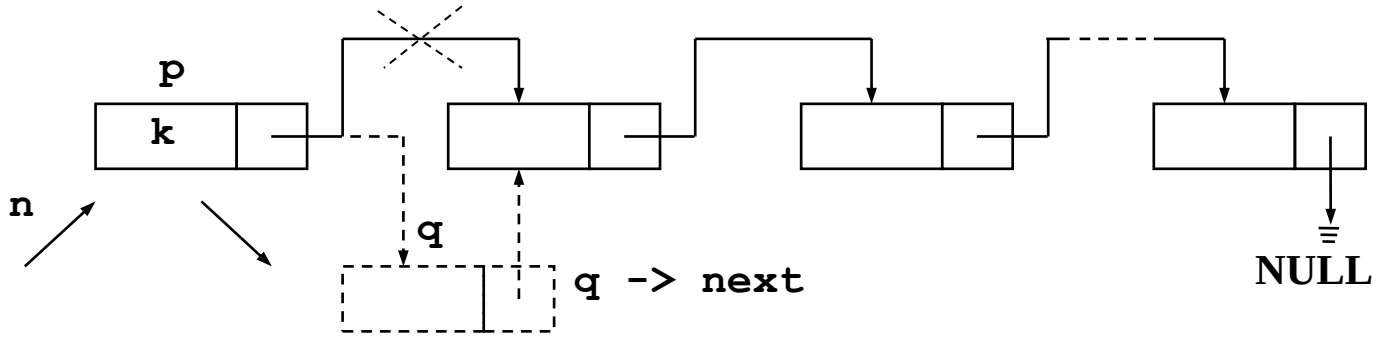



Включване на елемент в началото на списък:

```
void add1(int n)
{
  elem *p=start;
  start=new elem;
  start->key=n;
  start->next=p;
}
```

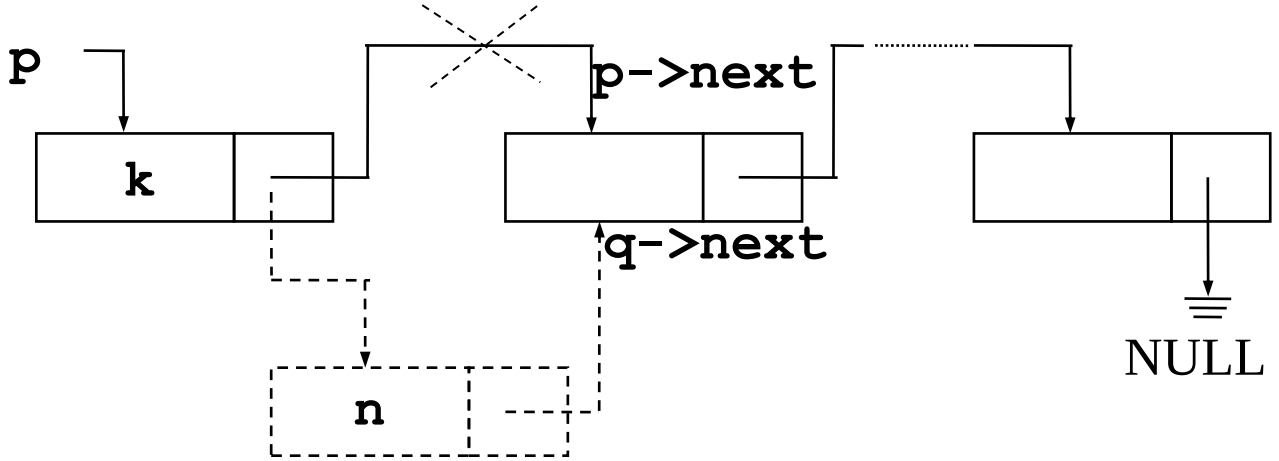


Включване на елемент в средата на списък преди друг, чиято стойност е **k**. Тук се предполага, че елементът **k** съществува:



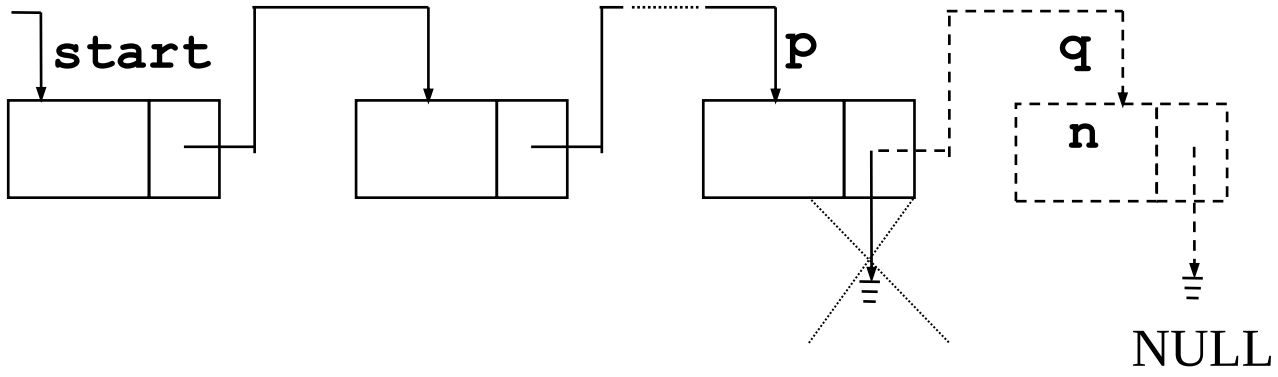

```
void add2(int n, int k)     //n – е добавяната стойност
{
  elem *q;
  elem *p=start;
  while (p->key!=k)          //търсене на елемент със
       p=p->next;            //стойност k
  q=new (elem);              //създаване на нов елемент
  q->next=p->next;
  q->key=p->key;             //прехвърляне на ключовата
  p->next=q;                 //стойност в *q
  p->key=n;                  //запис на n в p
}
```

Включване на елемент в средата на списъка след друг, чиято стойност е **k** (предполага се, че елементът **k** съществува):



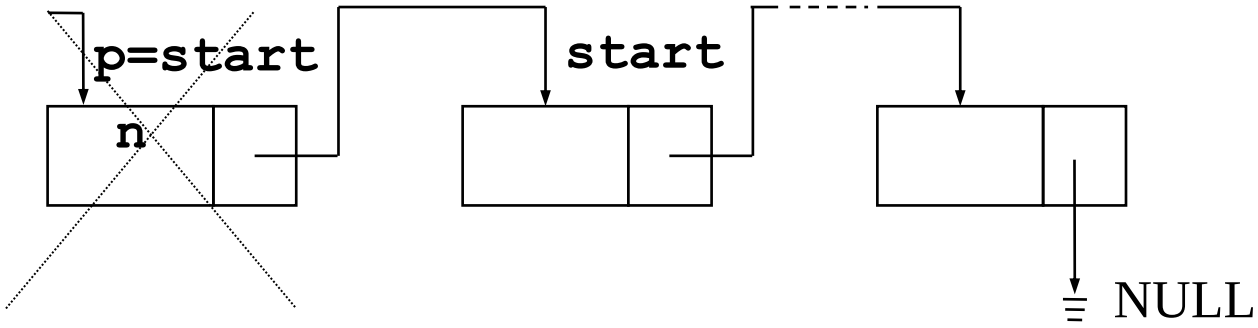

```
void add3(int n, int k) //n – е добавяната стойност
{
  elem *q;
  elem *p=start;
  while (p->key!=k) //търсене на елемент със
       p=p->next;                    //стойност k
  q=new elem;          //създаване на нов елемент
  q->key=n;            //задаване на ключова стойност
  q->next=p->next;     //вмъкване на
  p->next=q;           //новосъздадения елемент
}
```

## Включване на елемент в края на списък (ако не е известна ключовата стойност на последния елемент в списъка):



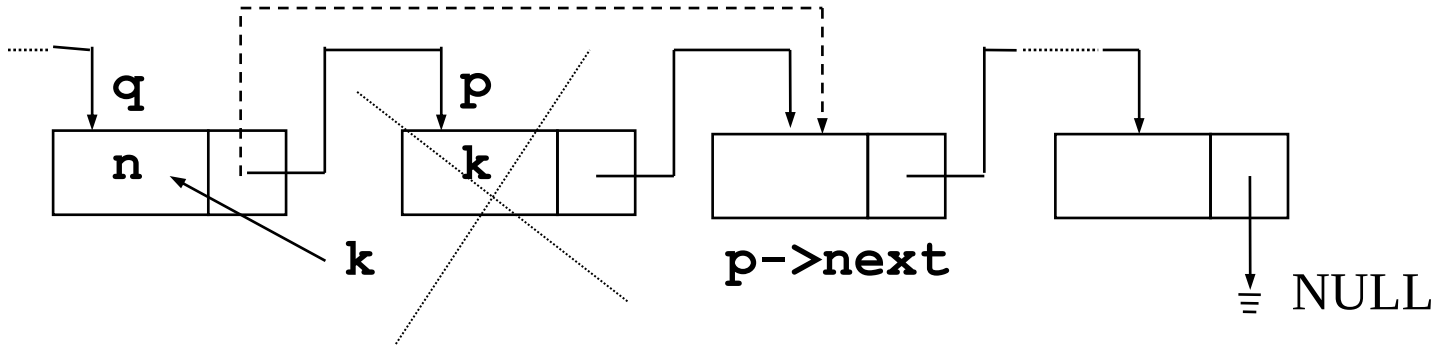

```
void add4(int n)
{
  elem *p=start, *q;
  q=new elem;          //създаване на новия елемент
  q->key=n;            //задаване на ключовата му стойност
  q->next=NULL;       //елементът ще е последен в списъка
  while (p->next)     //търсене на края на списъка
    p=p->next;
  p->next=q;      //свързване на елемента *q със списъка
}
```

## Изключване на първи елемент от списък:



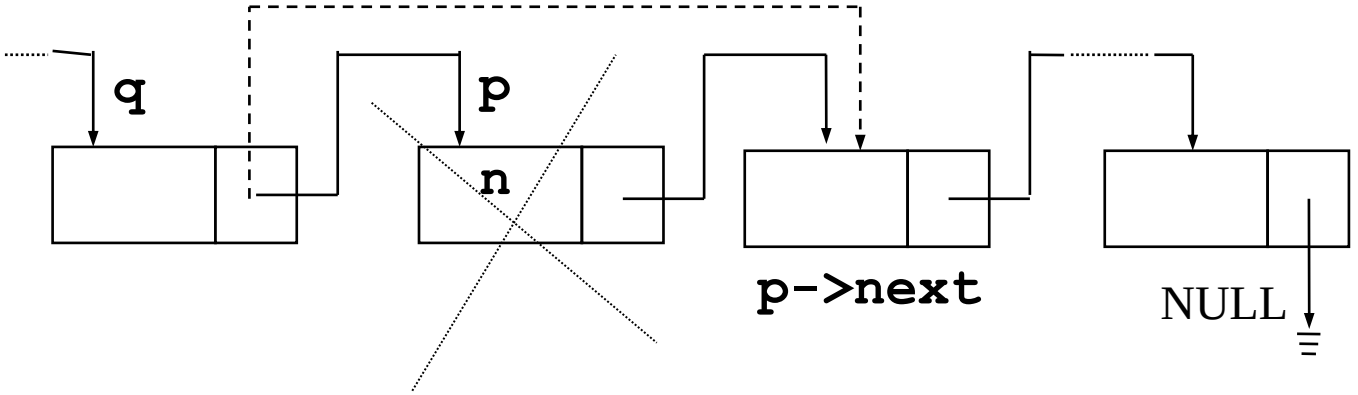

```
int del1(int &n)     //Ключовата стойност на изтрития
{                    //елемент се връща в главната функция
  elem *p=start;
  if (start)        //проверка, дали списъкът не е празен
    {
      n=start->key;
      start=start->next;//задаване на нов начален адрес
      delete p;             //изтриване
      return 1;
    }
      else
        return 0;           //списъкът е празен
}
```

Изключване на елемент в средата на списък преди друг, чиято ключова стойност е **k** (известно е, че елементът съществува):



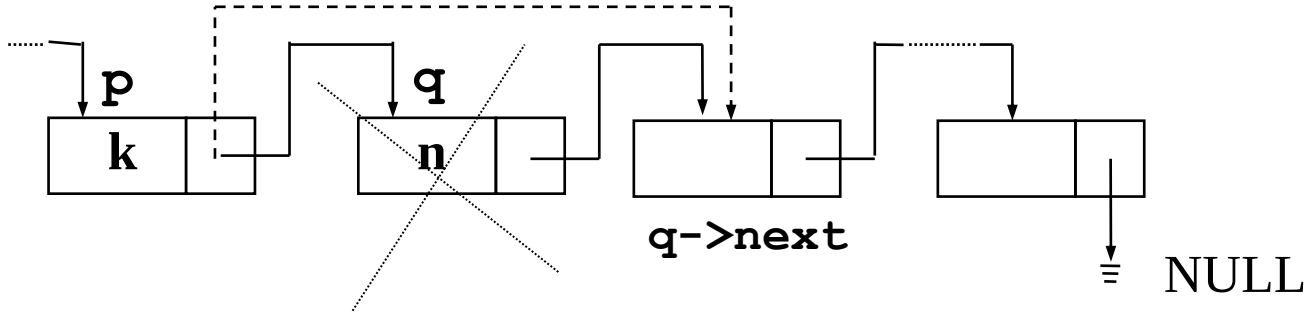

```
void del2(int &n, int k)     //Ключовата стойност на изтрития елемент
{                            //се връща в главната функция
  elem *q;
  elem *p=start;
  while (p->key!=k)          //търсене на елемента със  стойност k
      {
       q=p;
       p=p->next;
      }
    n=q->key;               //на практика се изтрива елемента *p
    q->next=p->next;        //но ключовата му стойност k
    q->key=p->key;          //се прехвърля в предходния елемент, като
                            //старата му стойност се изтрива от списъка
    delete p;
}
```

Изключване на елемент с ключова стойност **n** в средата на списък (известно е, че елементът съществува):



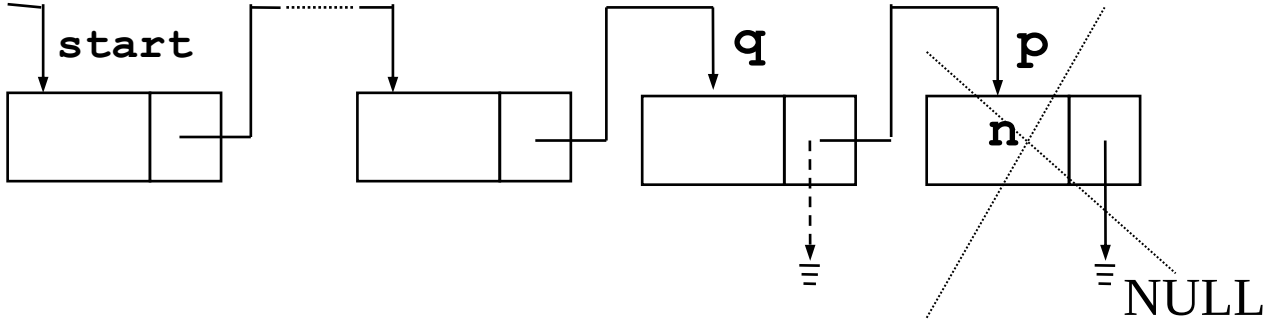

```
void del3(int &n)        //Ключовата стойност на изтрития елемент
 {                       //се връща в главната функция
  elem *q;
  elem *p=start;
  while (p->key!=n)      //търсене на елемента със стойност n
       {
        q=p;
        p=p->next;
       }
  n=p->key;
  q->next=p->next;
  delete p;              //изтриване на елемента *p
}
```

Изключване на елемент в средата на списък след друг, чиято ключова стойност е **k** (известно е, че елементът съществува):



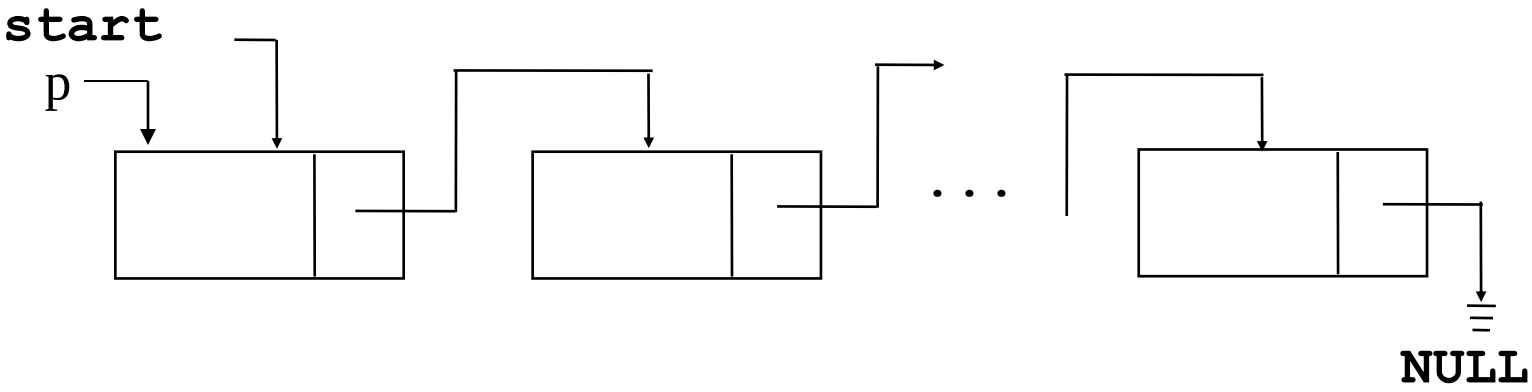

```
void del4(int &n, int k)    //Ключовата стойност на
{                           //изтрития елемент се
                            //връща в главната функция
  elem *q;
  elem *p=start;
  while (p->key!=k)         //търсене на елемент
      p=p->next;            //със стойност n
  q=p->next;
  n=q->key;
  p->next=q->next;          //пренасочване на връзката
  delete q;                 //изтриване на елемента
}
```

## Изключване на последен елемент в списък, който не е празен:




```
int del5(int &n)                    //Ключовата стойност на
                                    //изтрития елемент
{                                   //се връща в главната функция
  elem *p=start, *q;
  while (p->next)                   //търсене на последния
                                    //елемент
     {
      q=p;
      p=p->next;
     }
  n=p->key;
  q->next=NULL;                     //създаване на новия край
  delete p;                         //изтриване на елемента
}
```

# Търсене на елемент - итеративен вариант:



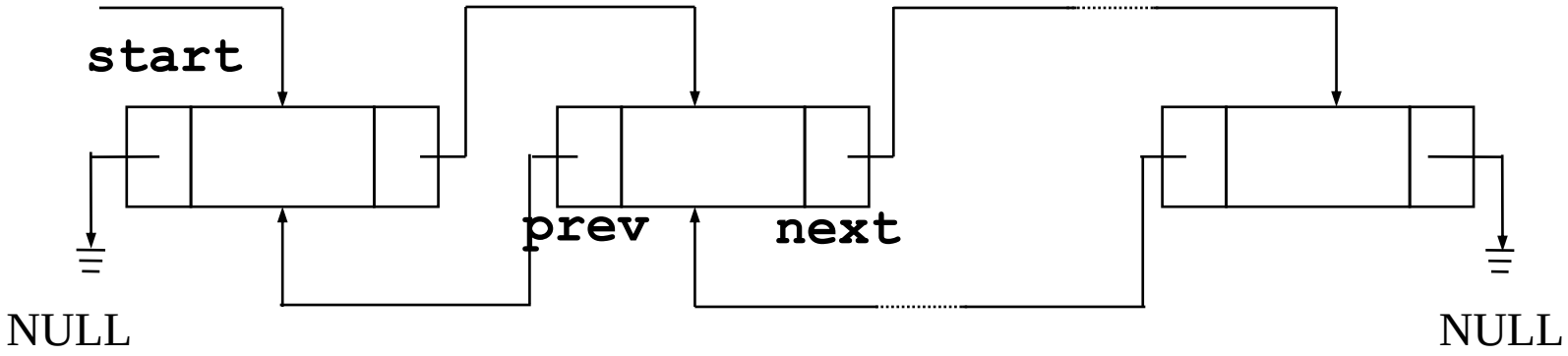

```
int search_iter(int n)
{
  elem *p=start;
  if (start)                    //проверка за непразен списък
    {
      while ((p->key!=n)&&(p->next))
         p=p->next;
      if (p->key==n)
         return 1;              //търсеният елемент е намерен
          else
         return 0;              //елементът не е намерен
    }
    return 0;                   //списъкът е празен
}
```

# Търсене на елемент в подреден списък - рекурсивен вариант

```
int search_recurs(struct elem *p, int n)
{                                       //търсене и включване на елемент,
 elem *q;                               //ако той не е в списъка
  if (start==NULL)                      //проверка за празен списък
     {                                  //добавяне на първия елемент
       start=new elem;       start->key=n;
       start->next=NULL;  return 0;
     }
       else
     { if ((p->key<n)&&(p->next==NULL))
        {                                       //добавяне в края на списъка
          q=new elem;   q->key=n;
          q->next=NULL; p->next=q; return 0;
        }
         else
          { if (p->key==n) return 1;            //елементът е открит
              else
                if (p->key<n)
                   search_recurs(p->next, n);
                  else                          //открито е мястото за
                  {                             //добавяне на елемента
                   q=new elem; q->next=p->next;
                   q->key=p->key; p->key=n;
                   p->next=q; return 0;
                  }
           }
     }
}
```

# Двойно-свързан или двусвързан списък

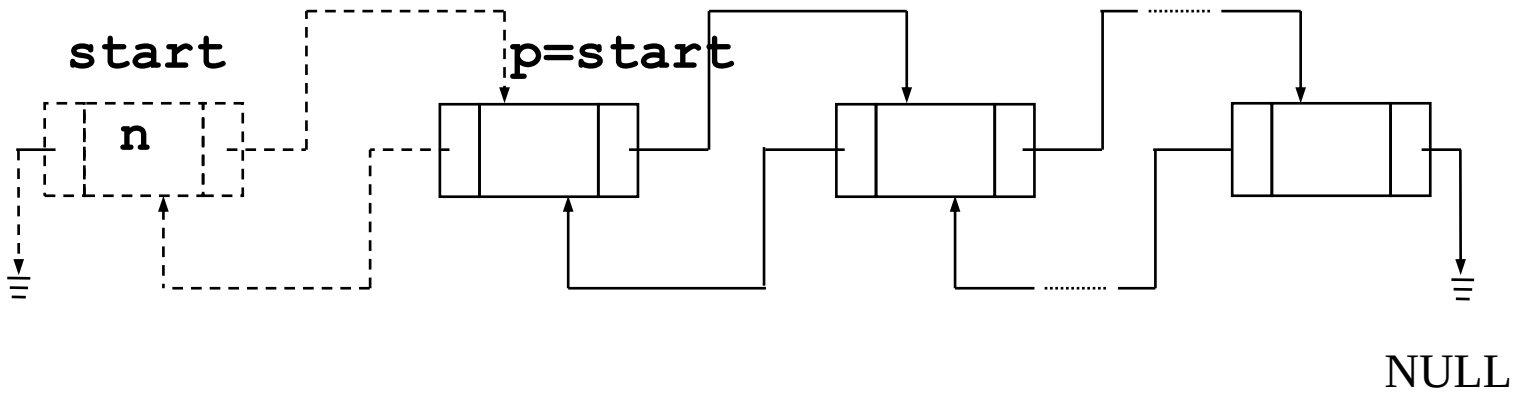
Всеки елемент от тази структура съдържа по два указателя - към предходния и към следващия елемент:




Примерно описание на структурата:

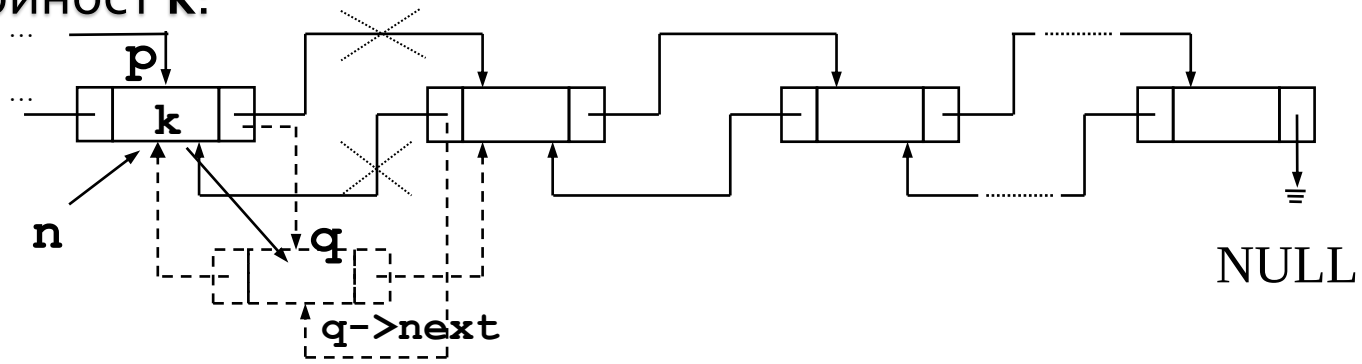
```
struct elem
   { int key; elem *prev; elem *next;} *start;
```

# Включване на елемент в началото на двусвързан списък:



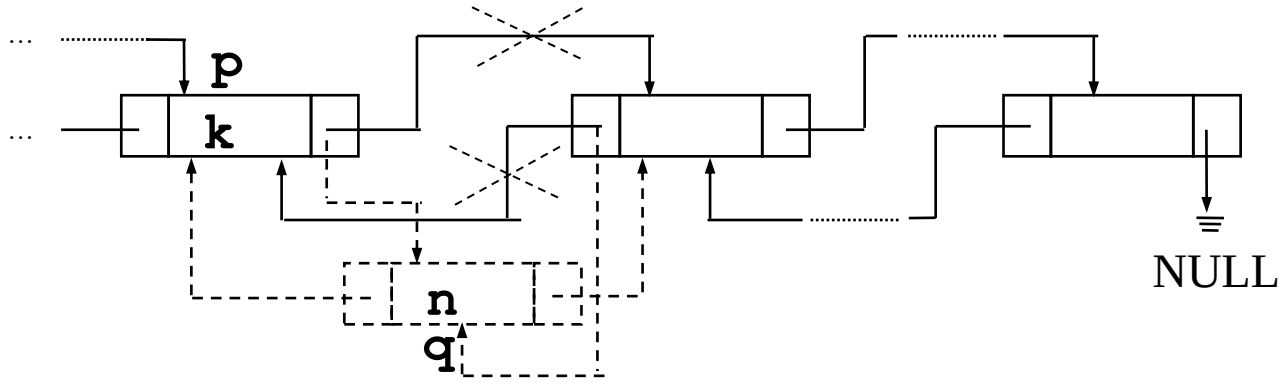

```
void add1(int n)                    //добавяне в началото
{
  elem *p=start;
  start=new elem;                   //създаване на новия елемент
  start->key=n;
  start->next=p;
  start->prev=NULL;                 //пренасочване на указателите
  p->prev=start;
}
```

Включване на междинен елемент със стойност **n** в двусвързания списък преди друг елемент с ключова стойност **k**:



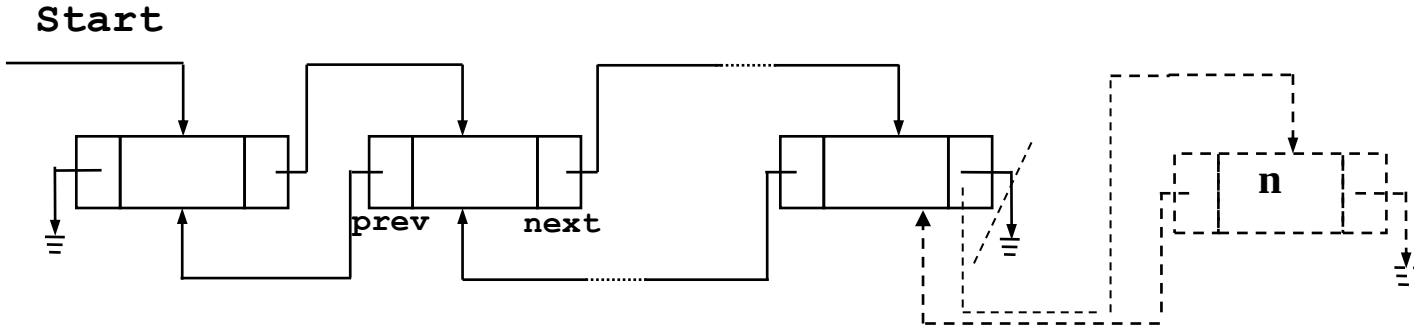

```
void add2(int n, int k)
{
  elem *q;
  elem *p=start;
  while (p->key!=k)          //търсене на k
       p=p->next;
  q=new (elem);              //създаване на нов елемент
  q->key=p->key;
  q->next=p->next;           //пренасочване на връзките
  q->prev=p; p->next->prev=q;
  p->next=q;
  p->key=n;
}
```

Включване на междинен елемент със стойност **n** в двусвързания списък след друг елемент с ключова стойност **k**:



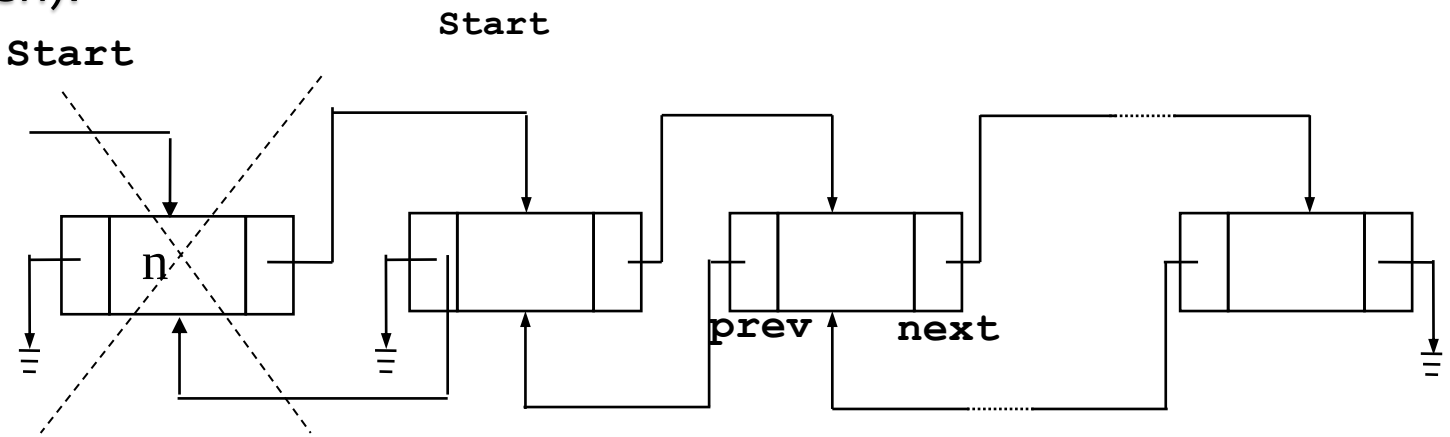

```
void add3(int n, int k)
{
  elem *q;
  elem *p=start;
  while (p->key!=k)
       p=p->next;
    q=new elem;              //създаване на нов елемент
    q->key=n;
    q->next=p->next;
    p->next->prev=q;         //пренасочване
    p->next=q;               //на връзките
    q->prev=p;
}
```

Включване на последен елемент със стойност **n** в двусвързания списък с начален указател **start**:



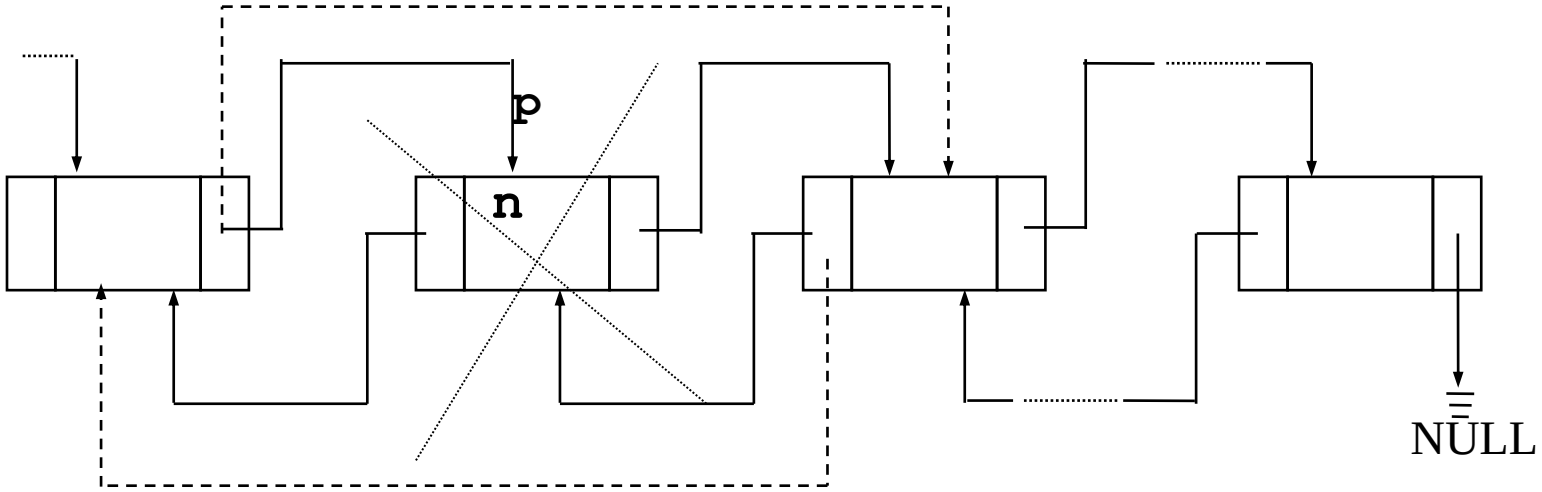

```
void add4(int n)
{
  elem *p=start, *q;
  q=new elem;                    //създаване на нов елемент
  q->key=n;
  q->next=NULL;
  if (start)                     //проверка за непразен списък
   {while (p->next)
      p=p->next;
    p->next=q;
   }
     else
       start=q;                  //елементът е единствен
  q->prev=p;
}
```

Изключване на първия елемент от двусвързан списък (предполага се, че елементът е в списъка и списъкът не е празен):



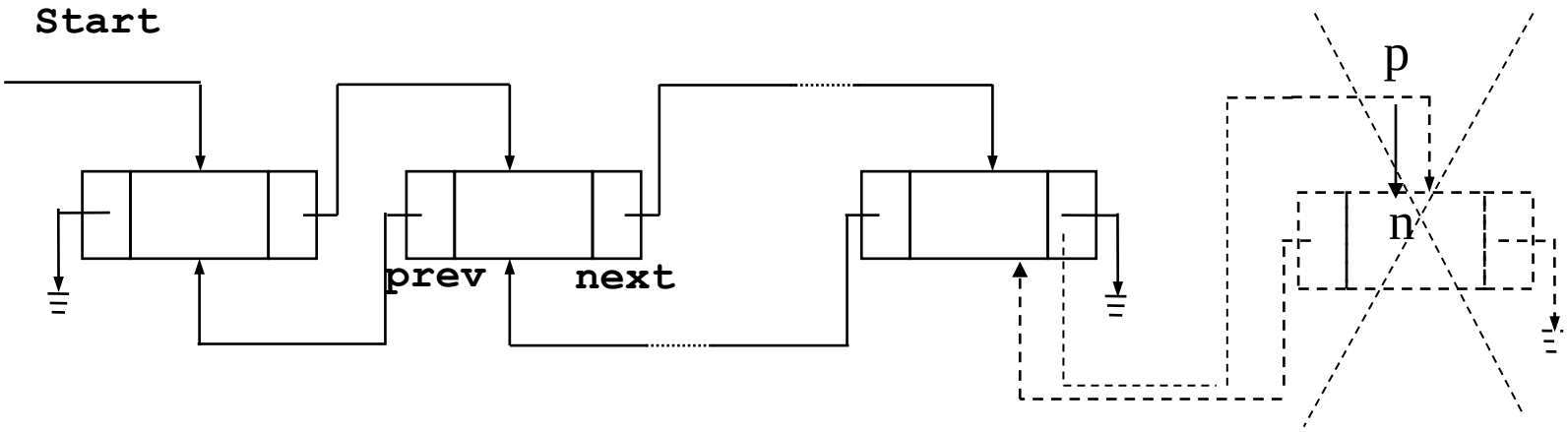

```
void del1(int &n)          //стойността на изтрития елемент
{                          //е достъпна в main()
  elem *p=start;
  n=start->key;
  start=start->next;       //пренасочване
  start->prev=NULL;        //на връзките
  delete p;                //изключване
}
```

Изключване на междинен елемент със стойност **n** от двусвързан списък (предполага се, че елементът е в списъка):




```
void del2(int n)
{
  elem *p=start;
  while (p->key!=n)          //търсене на елемента n
    p=p->next;
  p->prev->next=p->next;     //пренасочване на
  p->next->prev=p->prev;     //връзките
  delete p;                          //изтриване
}
```

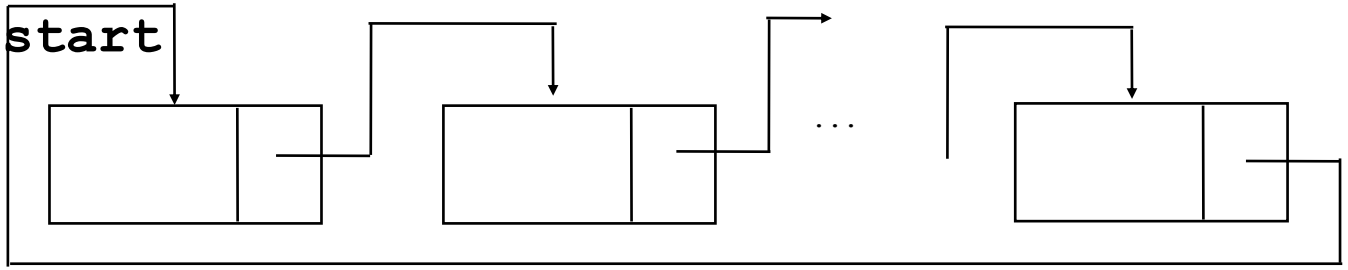
Изключване на последен елемент от двусвързан списък (предполага се, че елементът е в списъка):



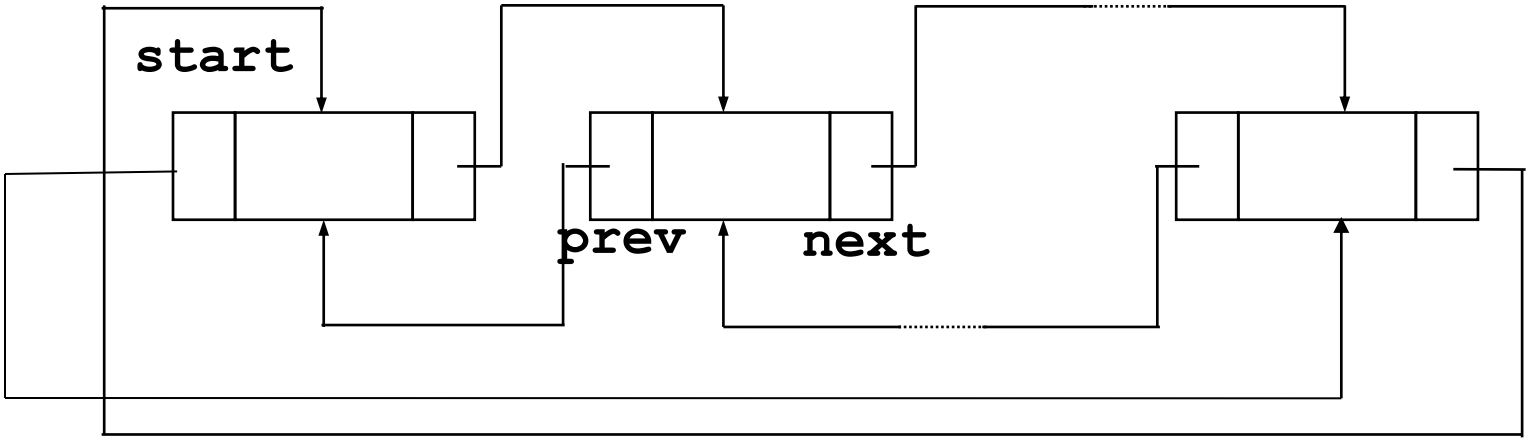

```
void del3(int &n)
{
  elem *p=start;
  while (p->next)                    //търсене на последния елемент
      p=p->next;
  n=p->key;
  p->prev->next=NULL;                //пренасочване на връзката
  delete p;                          //изтриване
}
```

# Кръгови (циклични) списъци

- единично свързани




- двойно-свързани



## Пример за работа с единично-свързан кръгов списък:

Моделиране на играта "Броеница": N деца застават в кръг и получават номера от 1 до N. Като се започне от дете 1, по часовниковата стрелка се отброяват M деца. Дете с номер M излиза от кръга, след което започва ново броене от следващото дете. Продължава, докато остане само едно дете, чийто номер трябва да се определи.

```
#include <iostream>
using namespace std;
struct elem
   { int key; elem *next;} *start=NULL;
void add (int n)
{
  elem *p=start;
  start=new elem;
  start->key=n;
  start->next=p;
}
```

```
void main()
{
  int i,n,m;
  cout<<”Въведете броя на децата n: ";
  cin>>n;
  for (i=n; i>0; i--)
     add(i);
  elem *p=start;
  while (p->next!=NULL)
      p=p->next;
  p->next=start;
  cout<<"\n Въведете m: ";
  cin>>m;
  p=start;
  elem *q;
```

```
do
  {
    for (i=1; i<m; i++)
      {
          q=p;
          p=p->next;
      }
    q->next=p->next;
    cout<<"\n номер за изтриване  "<<p->key;
    delete(p);
    p=q->next;
  }
  while (p!=p->next);
  cout<<"\nОстава дете с номер "<<p->key;
}
```