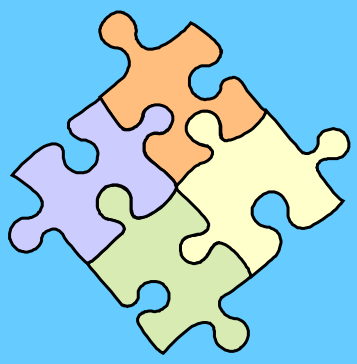
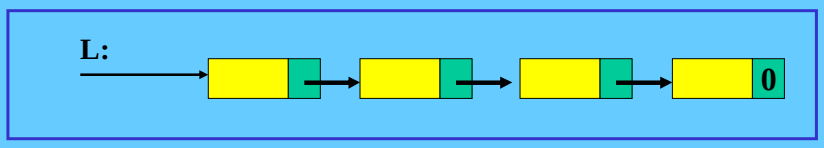
Elemi alkalmazások fejlesztése II.

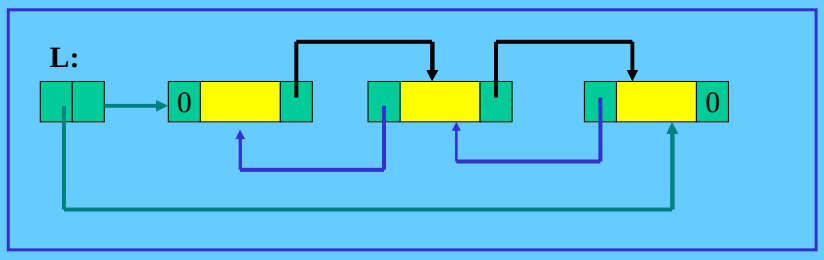


Láncolt adatszerkezetek

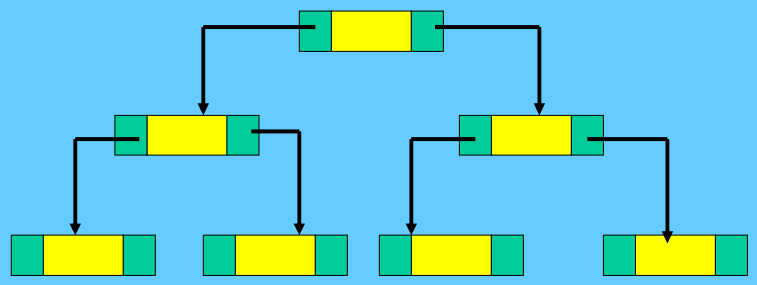
Egyirányú láncolt lista



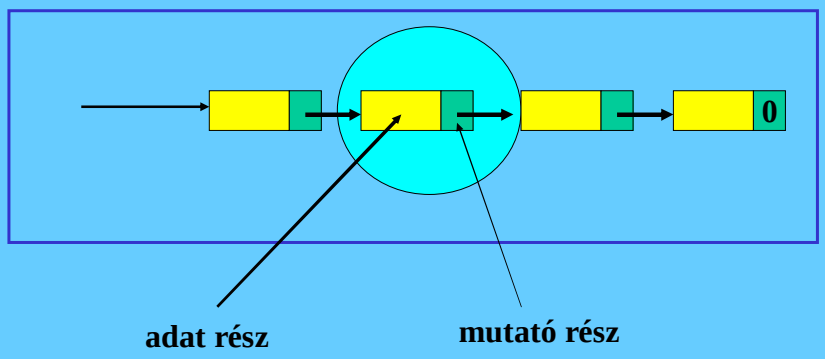
Kétirányú láncolt lista



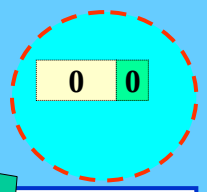
Bináris fa



A lista eleme



Listaelem deklarálása

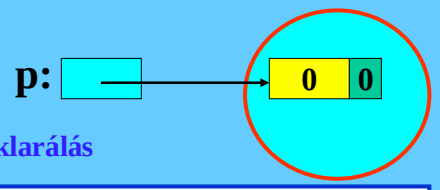


Node

1 Deklarálás

```
struct Node {
   int value;
   Node* next;
   Node(int i=0, Node *p=0)  {
      value=i;
      next=p;
   }
};
```

Listaelem létrehozása 1.



Node

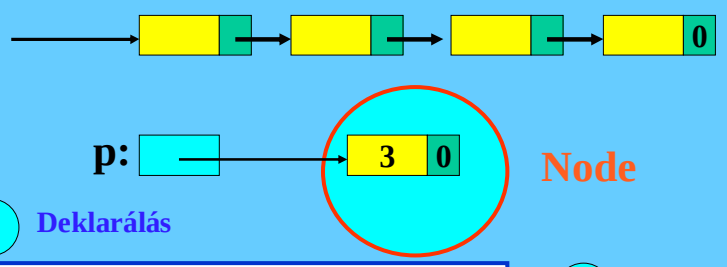
1 Deklarálás

```
struct Node {
   int value;
   Node* next;
   Node(int i=0, Node *p=0)  {
      value=i;
      next=p;
   }
};
```

2 Létrehozás

```
Node* p=new Node;
```

Listaelem létrehozása 2.

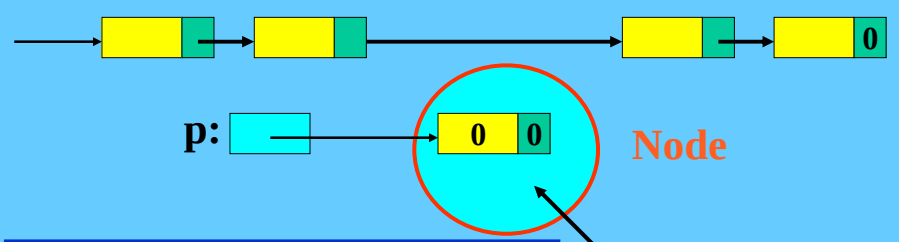


1 Deklarálás

```
struct Node {
   int value;
   Node* next;
   Node(int i=0, Node *p=0)  {
      value=i;
      next=p;
   }
};
```

2 Létrehozás

```
Node* p=new Node(3);
```




```
struct Node {
   int value;
   Node* next;
   Node(int i=0, Node *p=0)  {
      value=i;
      next=p;
   }
};
```

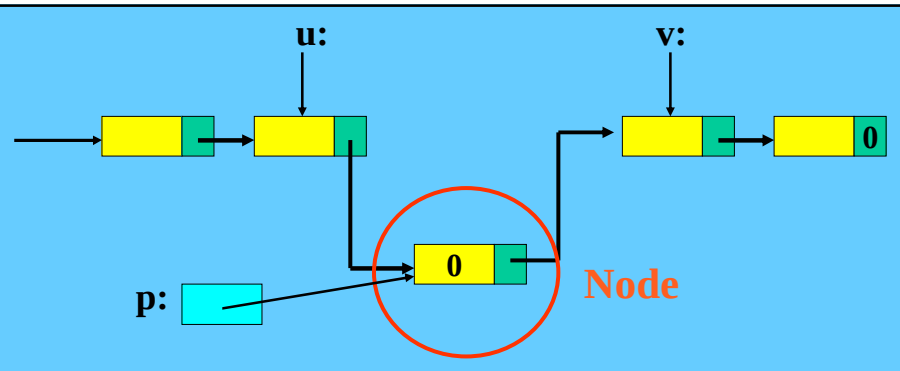
1

```
Node* p=new Node;
```

2

3

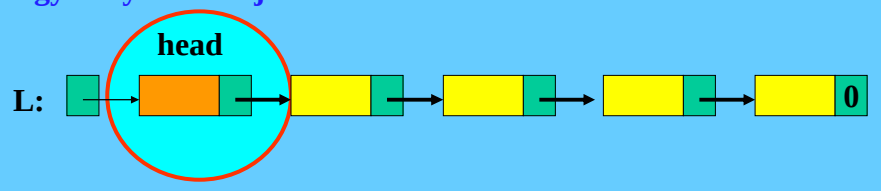
Az elemet nekünk kell „befűzni” a listába



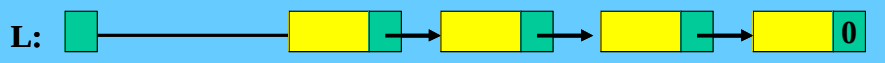

```
Node *u,*v,*p;
. . .

Node* p=new Node;
u->next=p;
p->next=v;
. . .
```

Az elemet nekünk kell „befűzni” a listába

Egyirányú lista fejelemmel



Egyirányú lista fejelem nélkül



Egy egyszerű rendező program

Modulszerkezet

SimpleList.cpp

Láncolás + Rendezés

new(p); p^adat:=x

L=NIL

p.^mut :=NIL

L :=p

u:=NIL; v:=L

v ≠ NIL ∧ x > v^.adat

u:=v; v:=v^.mut;

u=NIL

L:=p

p.^mut :=v

u.^mut :=p

p.^mut :=v

Beszúrás üres listába

new(p); p^adat:=x

L=NIL

p.^mut :=NIL

L :=p

u=NIL; v:=L

v ≠ NIL ∧ x > v^.a

u:=v; v:=

u=NIL

L:=p

p.^mut :=v

u.^mu

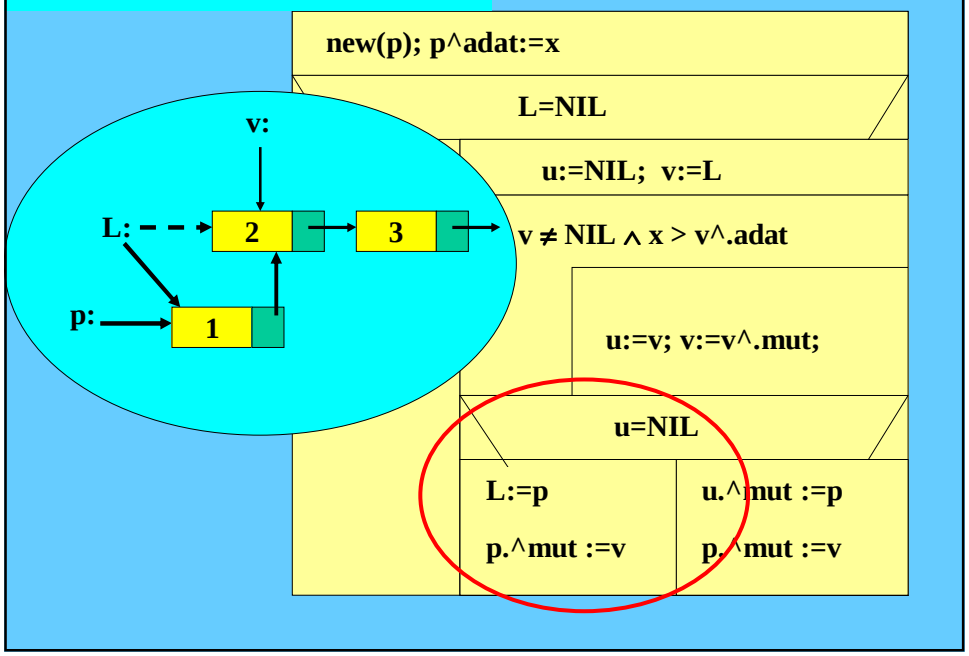
p.^mut :=v

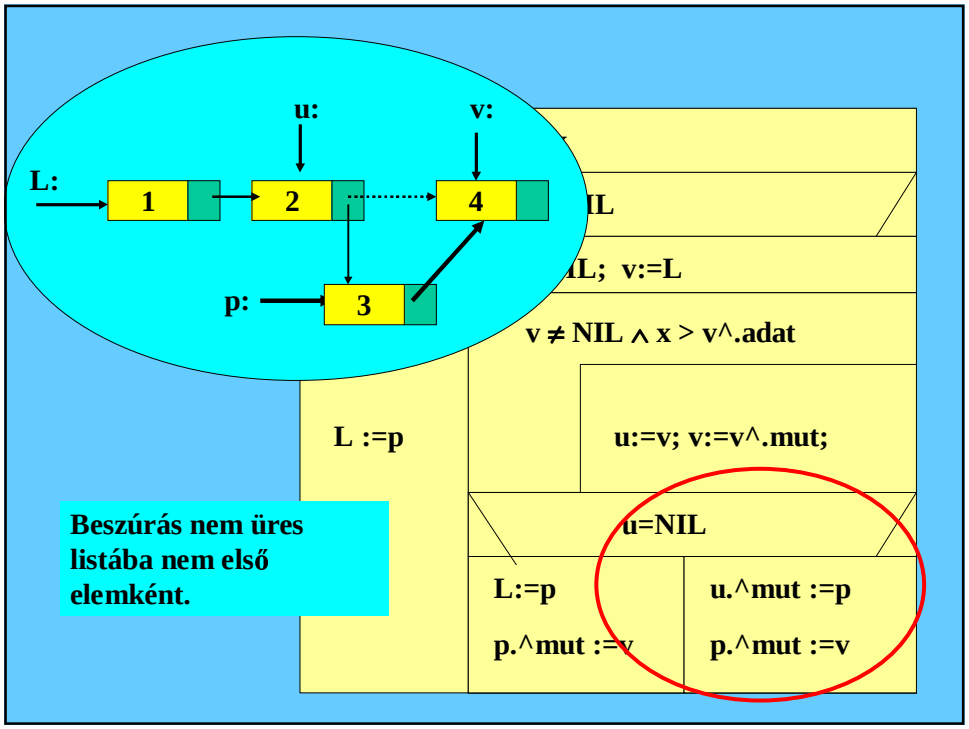
L:

p:

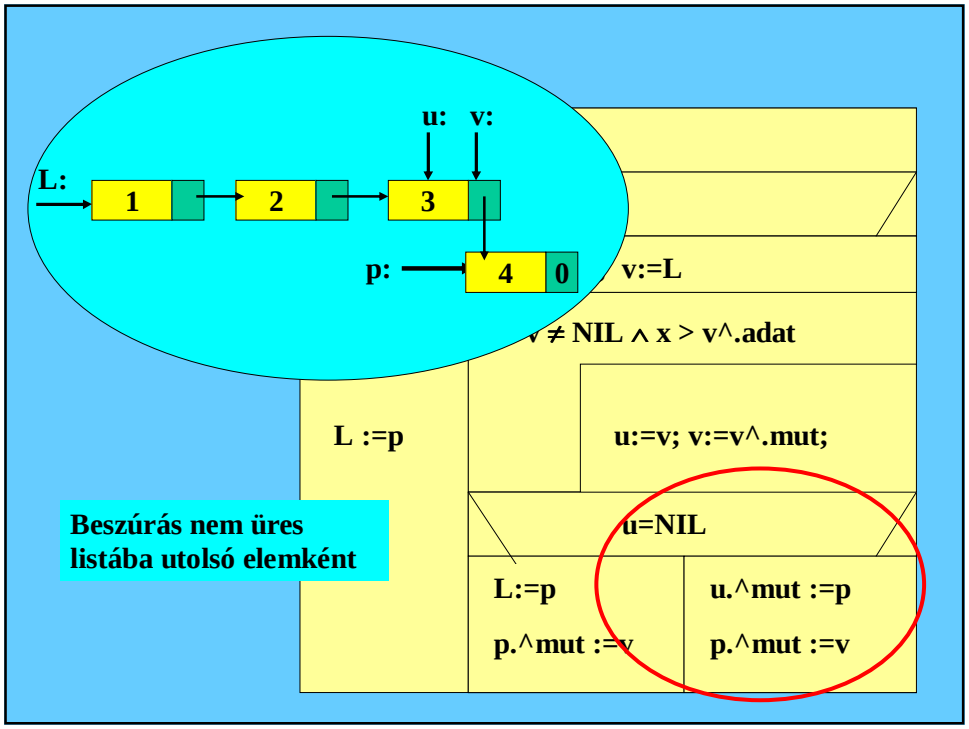
0

Beszúrás nem üres listába első elemként.





Beszúrás nem üres listába nem első elemként.



Kódolás

SimpleSort.cpp

```
int main()
{

//Listaelem

struct Node {
   int value;
   Node* next;
};
```

```
//Első adat bekérése
    int x;
    cout << "-1:  vege. " << endl;
    cout << "Adat: " ; cin >> x; cout << endl;
```

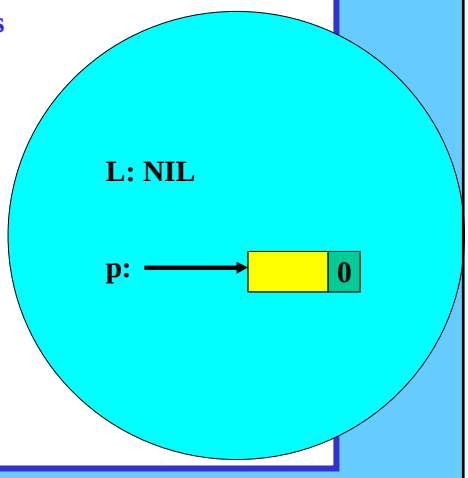
new(p); p^adat:=x

L=NIL

p.^mut :=NIL

L :=p

```
//Rendezett lista építés
    Node* lista;
    Node *p,*u,*v;
    lista=0;

    while(x!= -1) {
        p=new Node();
        p->value=x;
        if(lista==0) {
            p->next=0;
            lista=p;
        }
```



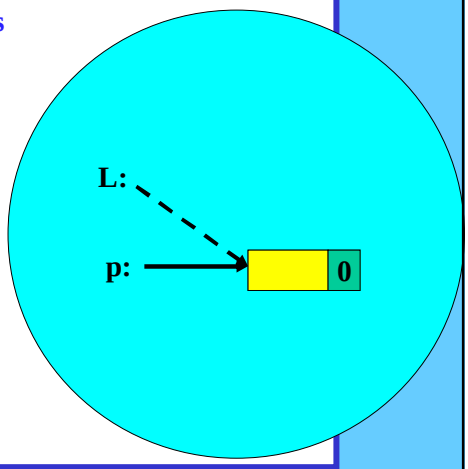
new(p); p^adat:=x

L=NIL

p.^mut :=NIL

L :=p

```
//Rendezett lista építés
    Node* lista;
    Node *p,*u,*v;
    lista=0;

    while(x!= -1) {
        p=new Node();
        p->value=x;
        if(lista==0) {
            p->next=0;
            lista=p;
        }
```

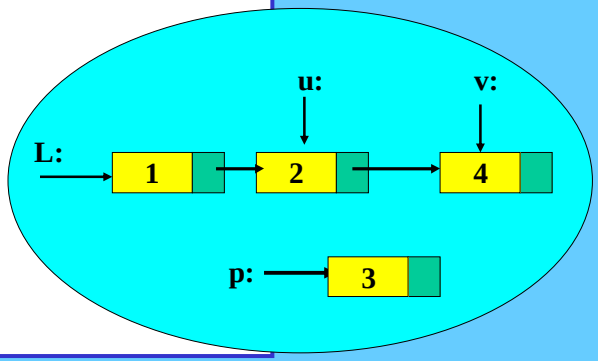



```
        else {
            u=0;
            v=lista;
            while ((v!=0)&&(x>v->value)) {
                u=v;
                v=v->next;
            }
            if (u==0) {
                lista=p;
                p->next=v;
            }
            else {
                u->next=p;
                p->next=v;
            }
        }
}
```

u:=NIL; v:=L

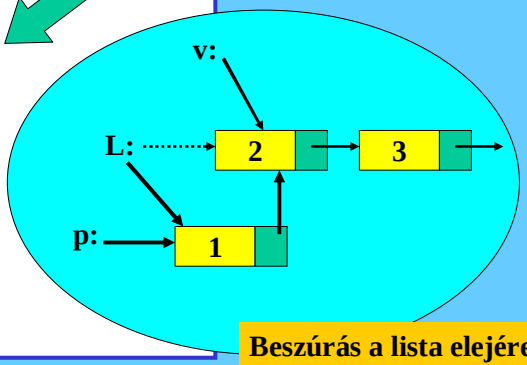
v ≠ NIL ∧ x > v^.adat

u:=v; v:=v^.mut;



```
else {
   u=0;
   v=lista;
   while ((v!=0)&&(x>v->value)) {
      u=v;
      v=v->next;
   }
   if (u==0) {
      lista=p;
      p->next=v;
   }
   else {
      u->next=p;
      p->next=v;
   }
}
}
```

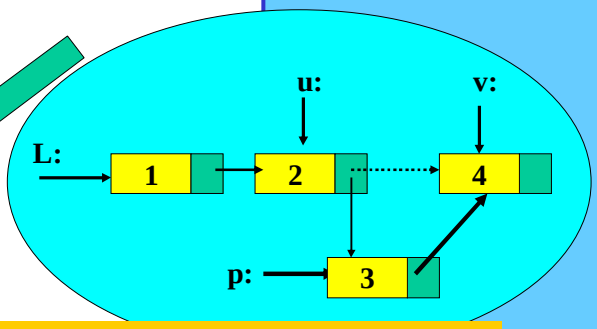
u=NIL	
L:=p p.^mut :=v	u.^mut :=p p.^mut :=v



Beszúrás a lista elejére

```
else {
   u=0;
   v=lista;
   while ((v!=0)&&(x>v->value)) {
      u=v;
      v=v->next;
   }
   if (u==0) {
      lista=p;
      p->next=v;
   }
   else {
      u->next=p;
      p->next=v;
   }
}
}
```

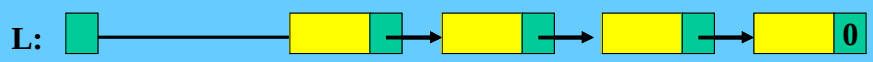
u=NIL	
L:=p p.^mut :=v	u.^mut :=p p.^mut :=v



Beszúrás a lista belsejébe vagy a végére.

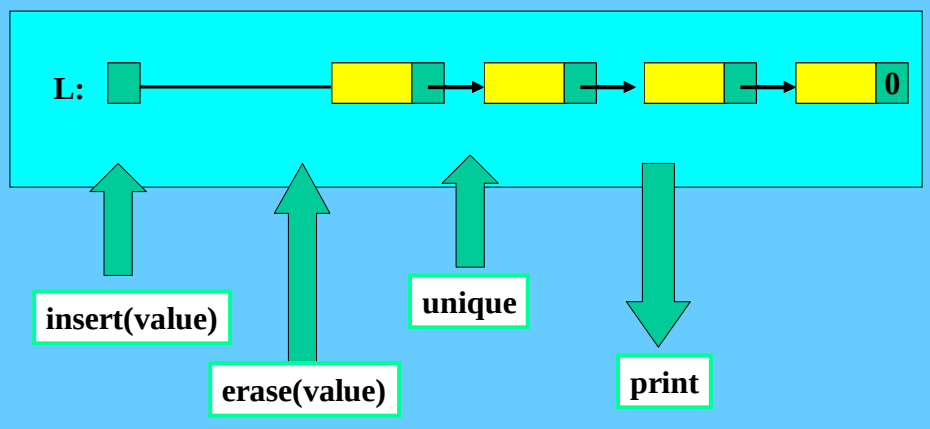
Lista típus (?)

Egyirányú lista fejelem nélkül



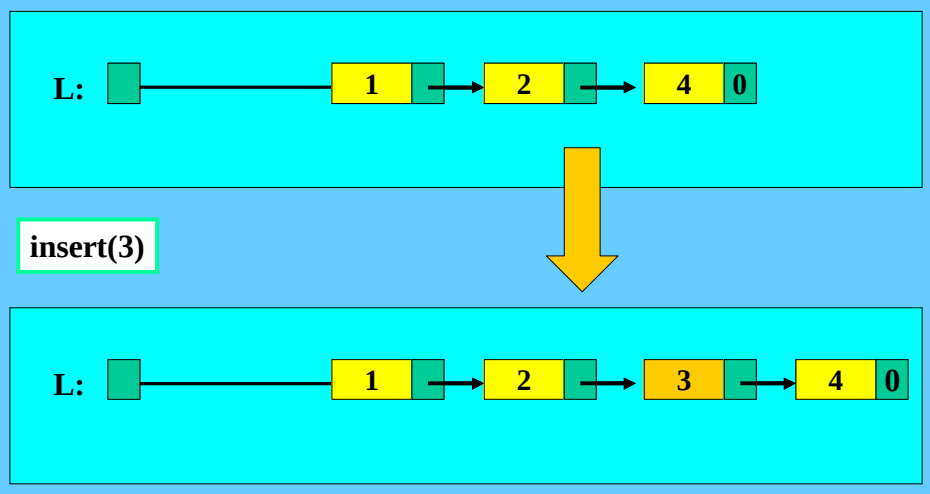

```
class List {

public:

private:
   Node* L;
};
```

Egyirányú lista fej elem nélkül



Pl. Lista egészek rendezett tárolására

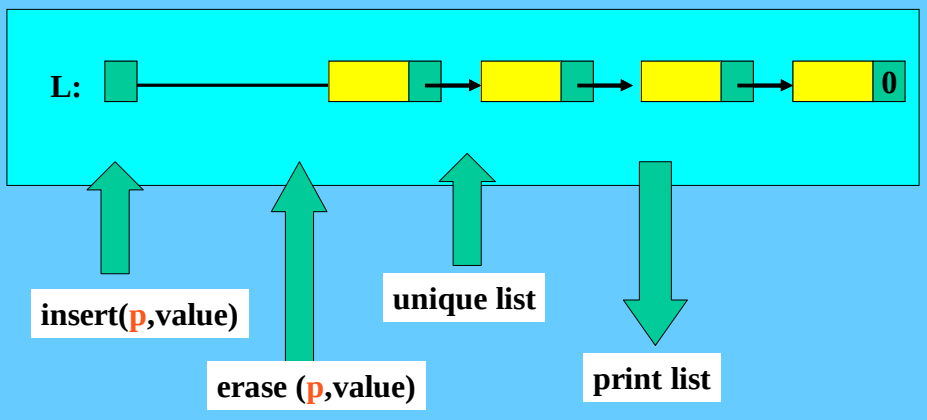
Egyirányú lista fej elem nélkül



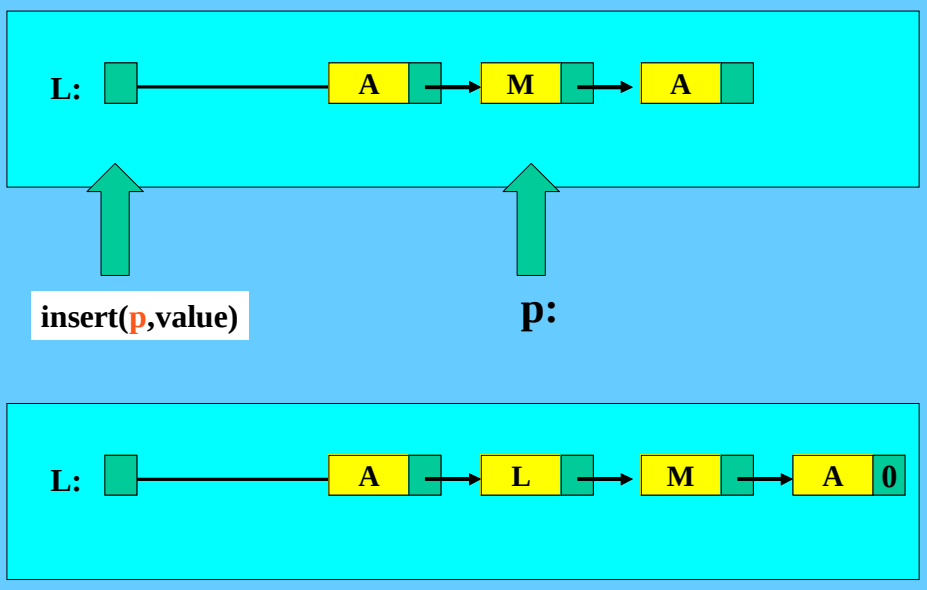
Sorter.h

```
class Sorter {

public:
    Sorter();
    ~Sorter();
    void insert(int value);
    void erase(int value);
    Sorter& unique();
    void print();

private:
     List *L;
};
```

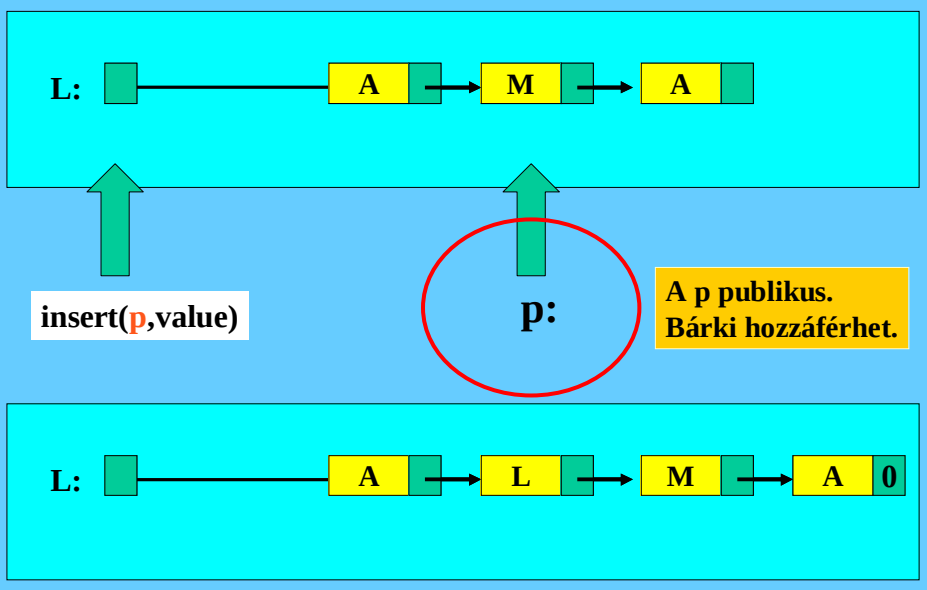
Egyirányú lista, fej elem nélkül, mutatót is visszaadó metódusokkal

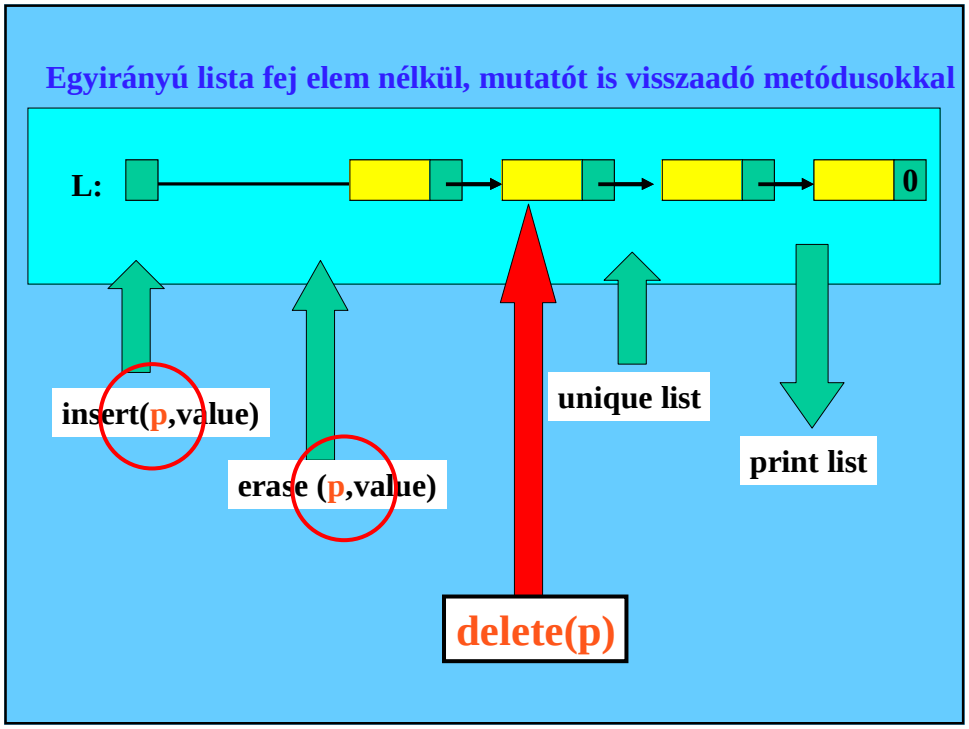


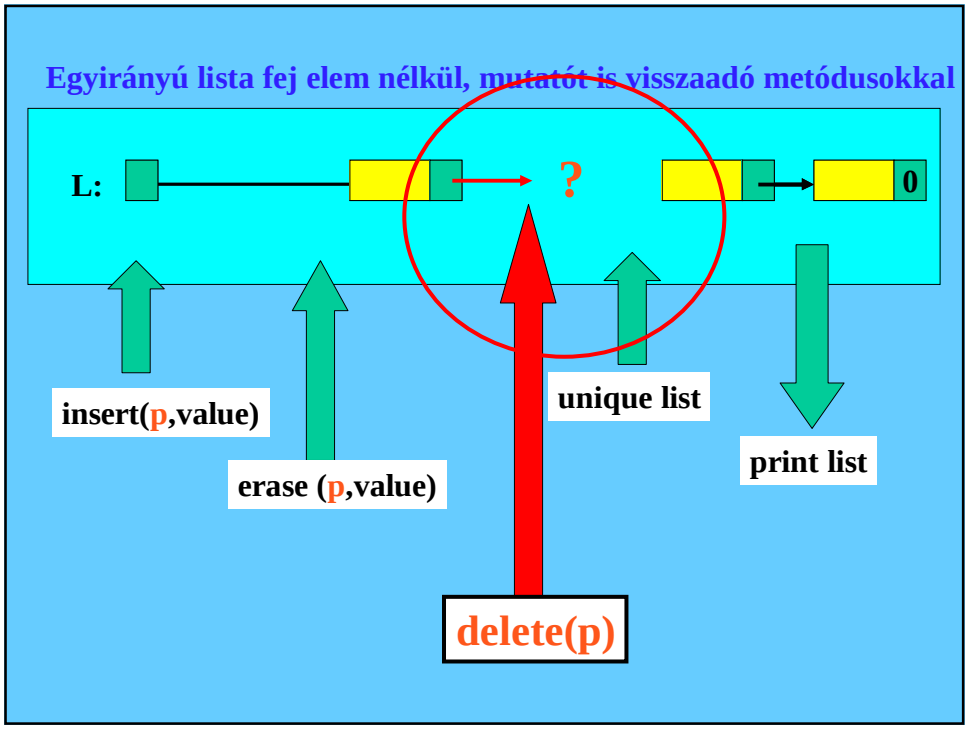
Egyirányú lista fej elem nélkül, mutatót is visszaadó metódusokkal



Egyirányú lista fej elem nélkül, mutatót is visszaadó metódusokkal




Egyirányú lista fej elem nélkül, mutatót is visszaadó metódusokkal
L:
0
insert(p,value)
erase (p,value)
unique list
print list
delete(p)

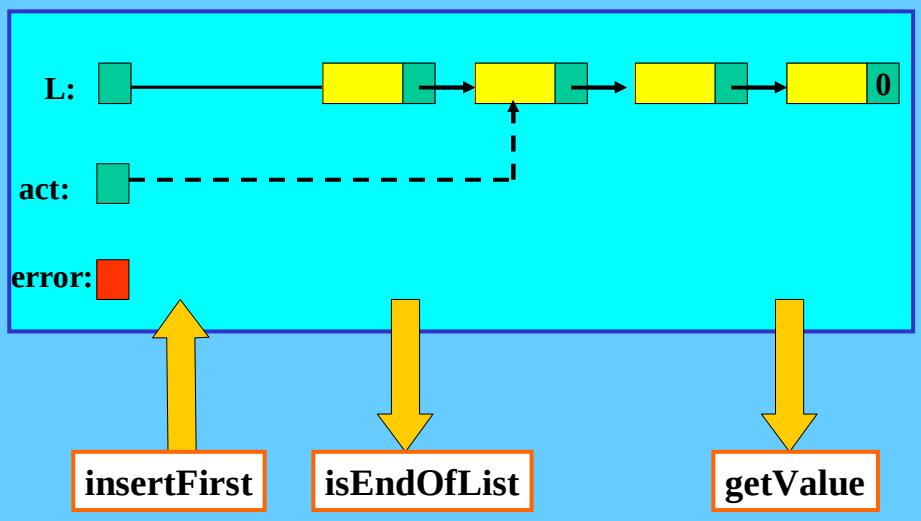

Egyirányú lista fej elem nélkül, mutatót is visszaadó metódusokkal
L:
?
0
insert(p,value)
erase (p,value)
unique list
print list
delete(p)

SimpleList.h

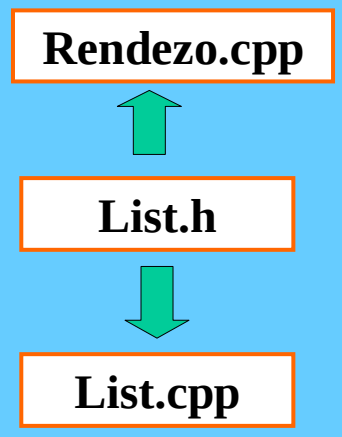
```
public:
    SimpleList();
    Node* add(int value);
    Node* next(Node* pointer);
    Node* first();
    Node* insertAfter(Node* pointer, int value);
    Node* insertBefore(Node* pointer, int value);
    Node* erase(Node* pointer);
    bool empty();
```

VESZÉLYES!!

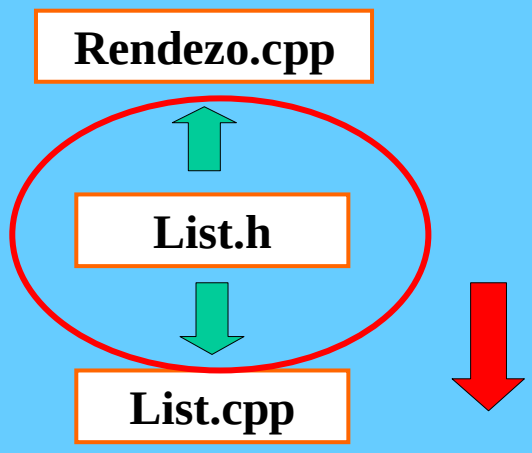
Egyirányú lista fejelem nélkül, aktuális mutatóval és hibakóddal



Modulszerkezet



Modulszerkezet



List.h

```
#ifndef LIST_H
#define LIST_H

struct Node {
   int value;
   Node* next;
   Node(int i=0, Node *p=0) {
      value=i;
      next=p;
   }
};
```

Lista elem deklarációja

List.h

```
class List {
public:
   List();
   bool isEmpty();
   bool fail();
   void first();
  . . .
   void erase();
   void insertFirst(int value);
   void insertLast(int value);
   void insertBefore(int value);
   void insertAfter(int value);
   //Kiegészítő funkciók
   void print();
```

metódusok deklarációja

List.h

```
class List {
public:
    List();
    bool fail();
    void first();
    void next();
    bool isEndOfList();
    . . .
    void erase();
    void insertFirst(int value);
    void insertLast(int value);
    void insertBefore(int value);
    void insertAfter(int value);
    //Kiegészítő funkciók
    void print();
```

void

Nincs mutató.
A művelet helyét private adattag tartalmazza.

List.h

```
class List {
public:
    List();
    bool fail();
    void first();
    void next();
    bool isEndOfList();
    . . .
    void erase();
    void insertFirst(int value);
    void insertLast(int value);
    void insertBefore(int value);
    void insertAfter(int value);
    //Kiegészítő funkciók
    void print();
```

A listán csak metódusok segítségével tudunk „közlekedni”.

List.h

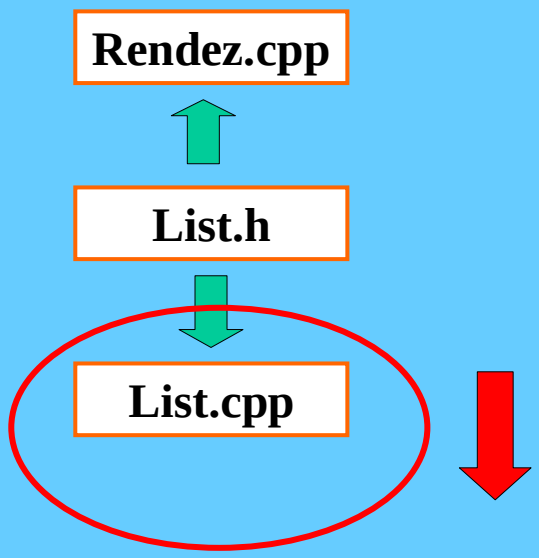
```
private:
    Node* head;
    Node* act;
    bool error;

};
#endif
```

head:

act:

error:

Modulszerkezet



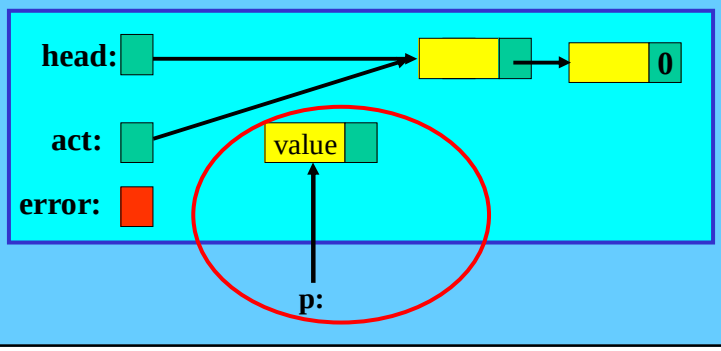
```
//Létrehozás
List::List() {
    head=0;
    act=0;
    error=false;
}
```

List.cpp:
konstruktor


```
void List::insertFirst(int value) {
  Node* p=new Node;
  p->value=value;
  p->next=head;
  head=p;
  act=head;
  return;
}
```

List.cpp:
insertFirst

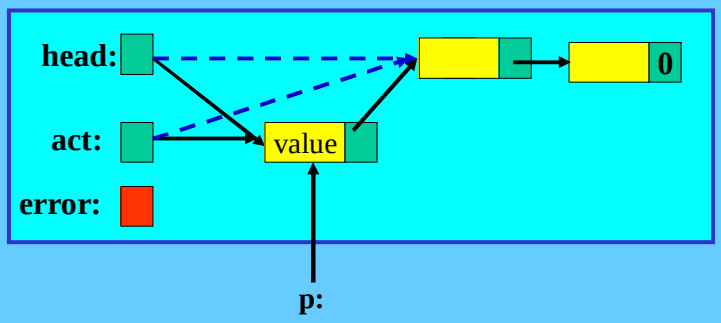
1



```
void List::insertFirst(int value) {
  Node* p=new Node;
  p->value=value;
  p->next=head;
  head=p;
  act=head;
  return;
}
```

List.cpp:
insertFirst

2



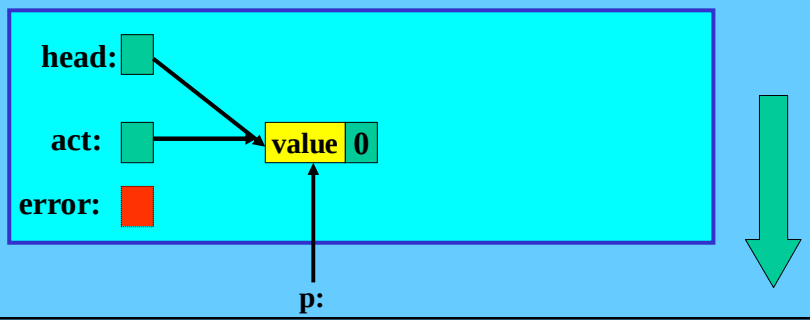
insertLast(L,e)

new(p); p^.adat:=e; p^.mut:=NIL	
L = NIL	
L:=p act:=L	u:=L; v:=L^.mut
	v ≠ NIL
	u:=v; v:=v^.mut
	u^.mut:=p; act:=p

```
void List::insertLast(int value) {
    Node* p=new Node;
    p->value=value;
    p->next=0;
    if (head == 0) {
        head=p;
        act=head;
        return;
    }
```

List.cpp:
insertLast

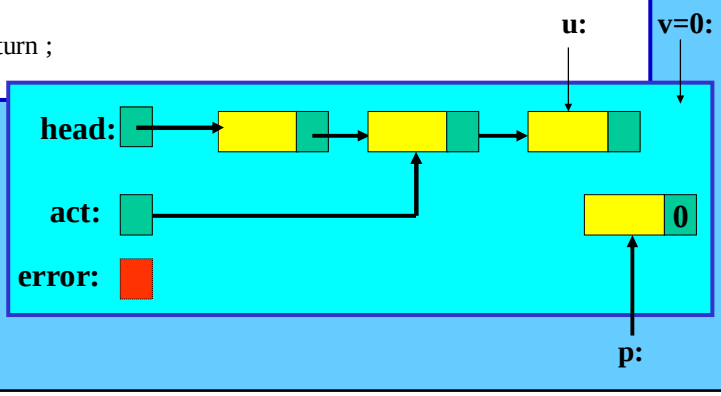
1




```
else {
    Node* u=head;
    Node* v=head->next;
    while(v!=0) {
        u=v;
        v=v->next;
    }
  u->next=p;
  act=p;
  }
  return ;
}
```

List.cpp:
insertLast

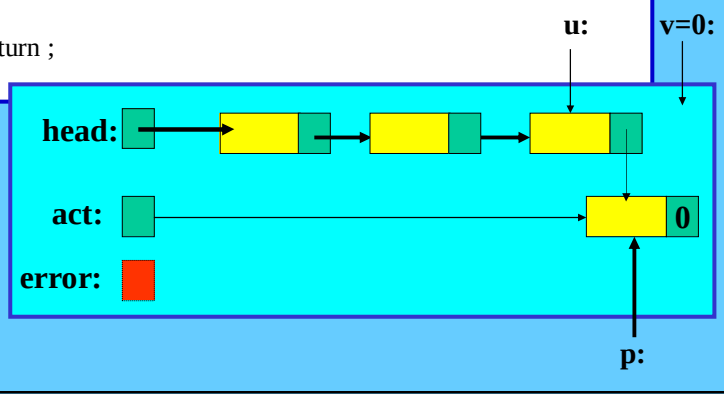
2



```
else {
    Node* u=head;
    Node* v=head->next;
    while(v!=0) {
        u=v;
        v=v->next;
    }
  u->next=p;
  act=p;
  }
  return ;
}
```

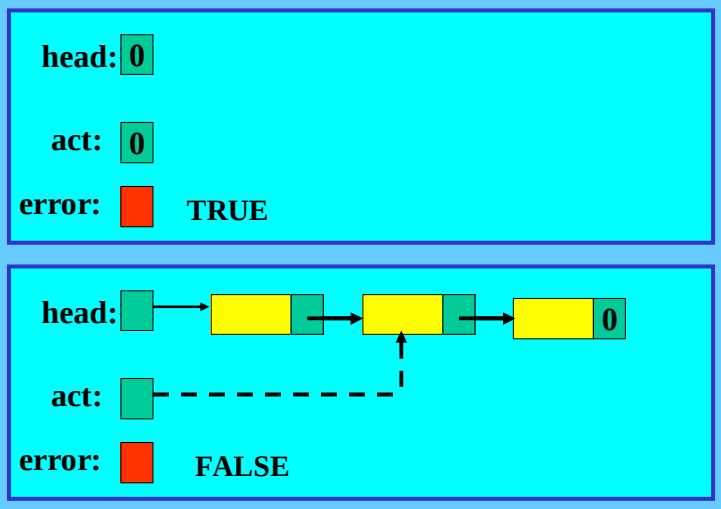
List.cpp:
insertLast

3




```
bool List::isEmpty() {
    return(head==0);
}
```

List.cpp:
isEmpty



insertAfter(L,e,error)

act=NIL

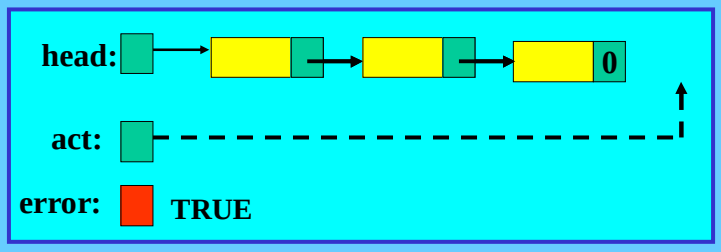
e r r o r : = ↑

new(p); p^.adat:=e
p^.mut:=act^.mut
act^.mut:=p
error:=↓

```
void List::insertAfter(int value) {
   if (act == 0) {
      error=true;
      return;
   }
```

List.cpp:
insertAfter

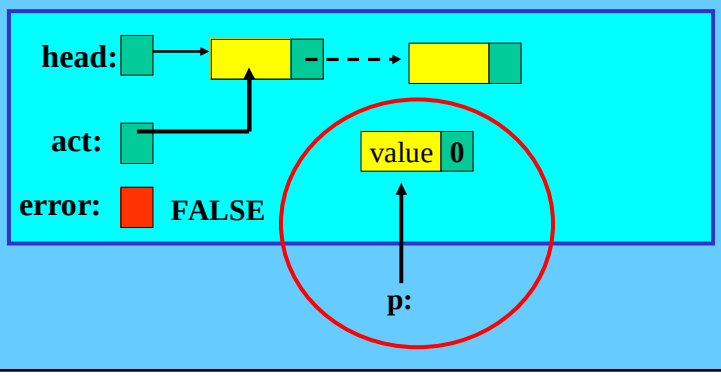
1



```
else {
    Node* p=new Node;
    p->value=value;
    p->next=act->next;
    act->next=p;
    error=false;
    return;
  }
}
```

List.cpp:
insertAfter

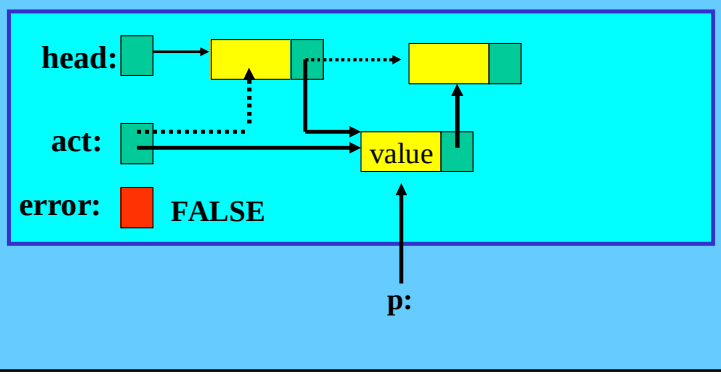
2




```
else {
    Node* p=new Node;
    p->value=value;
    p->next=act->next;
    act->next=p;
    error=false;
    return;
  }
}
```

List.cpp:
insertAfter

3



insertBefore(L,e,error)

act=NIL

error := ↑

new(p); p^.adat:=e

u=:L; v:=L^.mut

v ≠ act

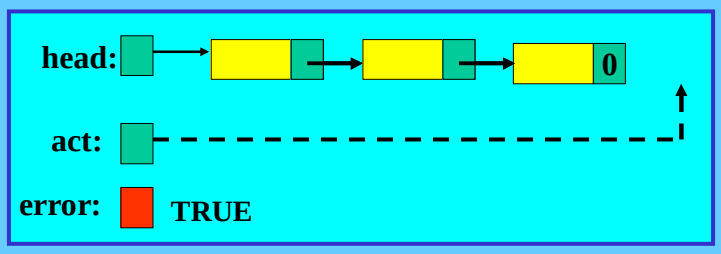
u:=v; v:=v^.mut

u^.mut:=p
p^.mut:=act
act:=p
error:=↓

```
void List::insertBefore(int value) {
    if (act == 0) {
        error=true;
        return;
    }
```

List.cpp:
insertBefore

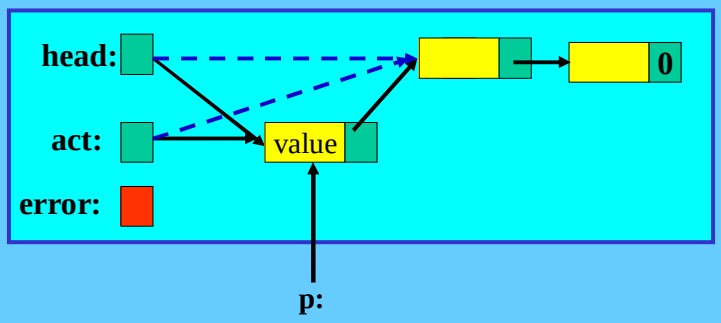
1



```
else if (act==head) {
   insertFirst(value);
}
```

List.cpp:
insertBefore

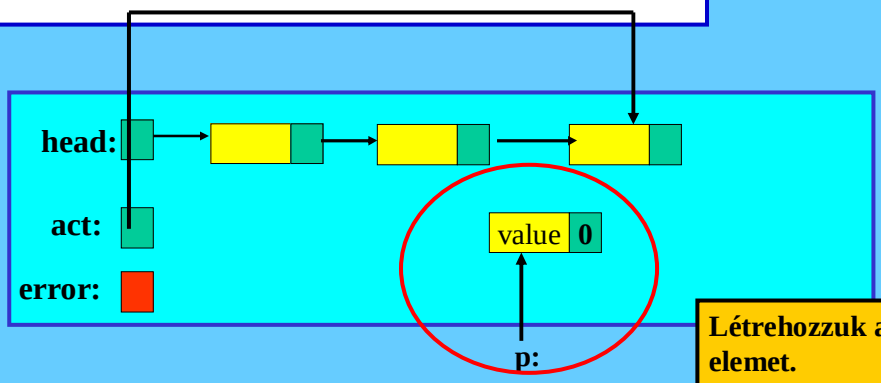
2




```
else {
  Node* p=new Node;
  p->value=value;
  Node* u=head;
  Node* v=head->next;
  while(v!=act) {
     u=v;
     v=v->next;
  }
```

List.cpp:
insertBefore

3

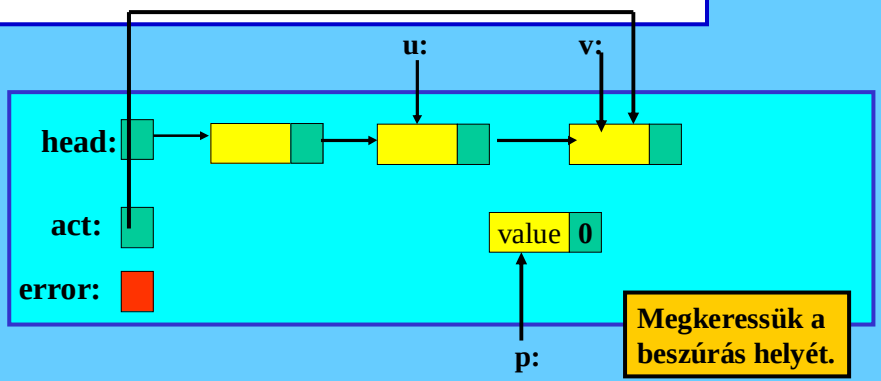


Létrehozzuk az elemet.

```
else {
  Node* p=new Node;
  p->value=value;
  Node* u=head;
  Node* v=head->next;
  while(v!=act) {
    u=v;
    v=v->next;
  }
```

List.cpp:
insertBefore

4

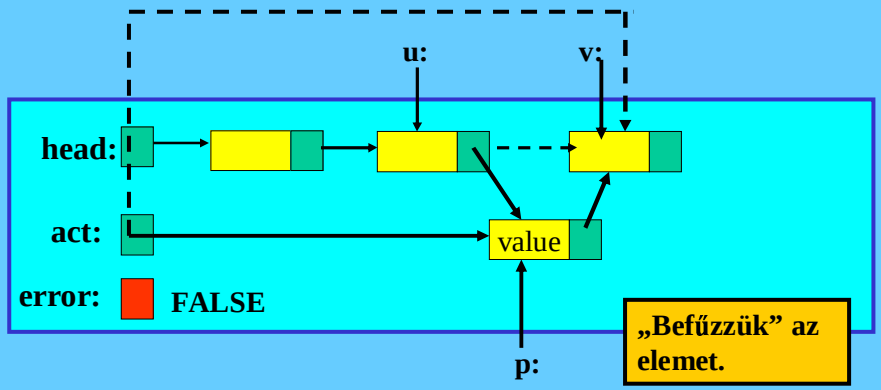


Megkeressük a beszúrás helyét.

```
    u->next=p;
    p->next=act;
    act=p;
    error=false;
    return;
  }
}
```

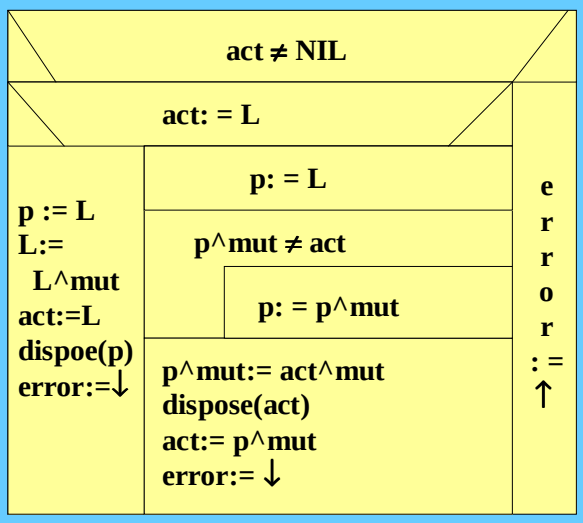
List.cpp:
insertBefore

5



„Befűzzük” az elemet.

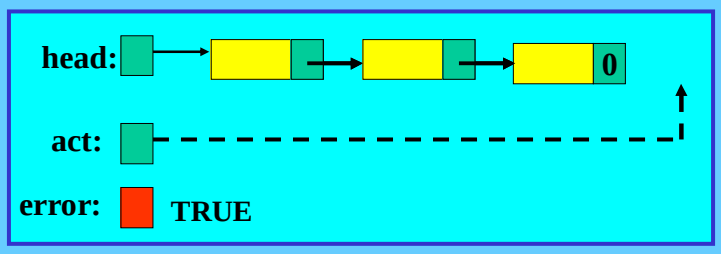
erase(L,error)




```
void List::erase() {
    if(act==0) {
        error=true;
        return;
    }
```

List.cpp:
erase

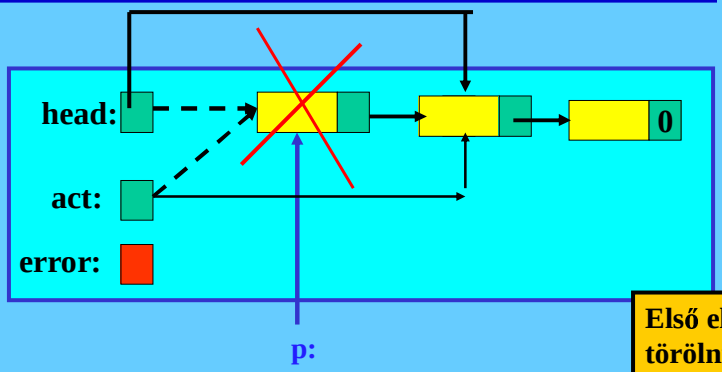
1



```
else if (act=head) {
    Node* p=head;
    head=head->next;
    act=head;
    delete(p);
    error=false;
    return;
}
```

List.cpp:
erase

2

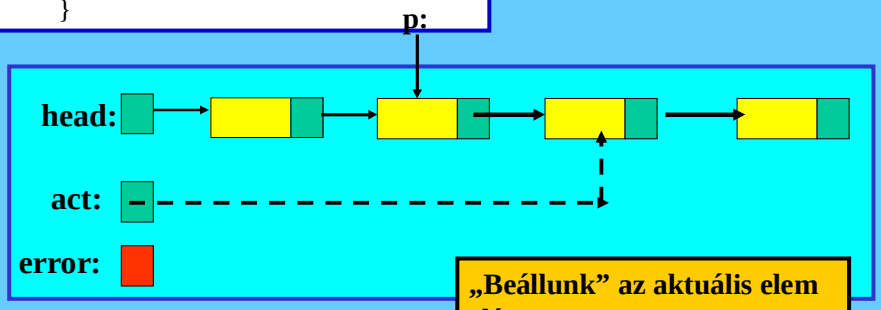


Első elemet kell törölni.

```
else {
    Node* p=head;
    while(p->next!=act) {
        p=p->next;
    }
    p->next=act->next;
    delete(act);
    act=p->next;
    error=false;
    return;
}
```

List.cpp:
erase

3

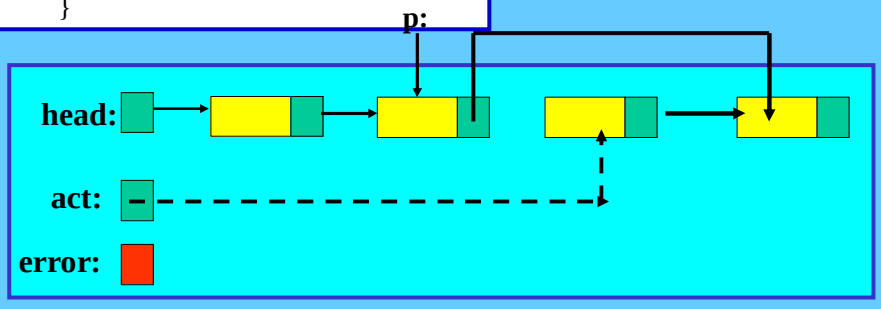


„Beállunk” az aktuális elem elé.

```
else {
   Node* p=head;
   while(p->next!=act) {
      p=p->next;
   }
   p->next=act->next;
   delete(act);
   act=p->next;
   error=false;
   return;
}
```

List.cpp:
erase

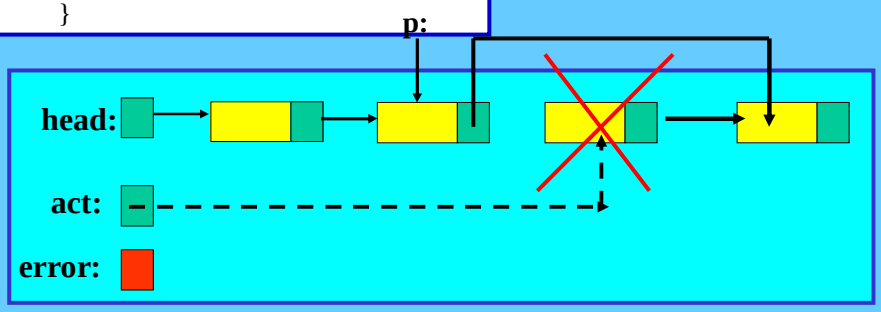
4




```
else {
   Node* p=head;
   while(p->next!=act) {
      p=p->next;
   }
   p->next=act->next;
   delete(act);
   act=p->next;
   error=false;
   return;
}
```

List.cpp:
erase

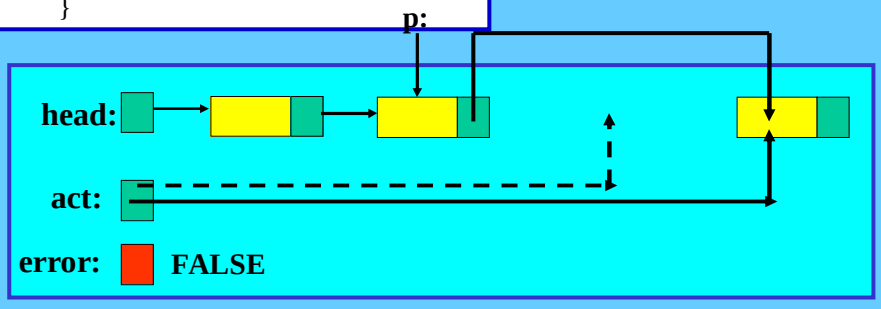
5



```
else {
   Node* p=head;
   while(p->next!=act) {
      p=p->next;
   }
   p->next=act->next;
   delete(act);
   act=p->next;
   error=false;
   return;
}
```

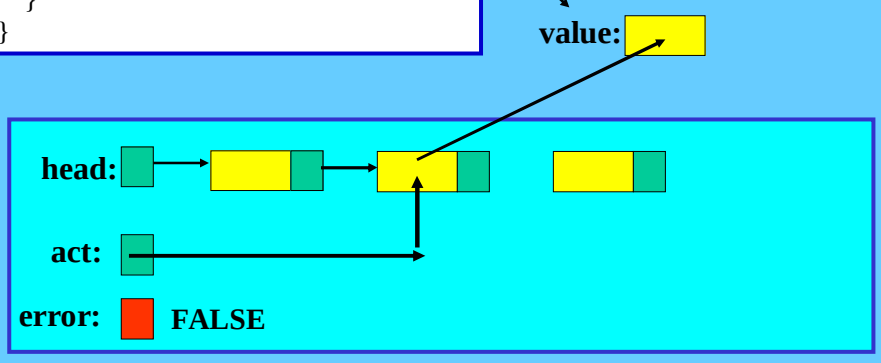
List.cpp:
erase

6




```
void List::getValue(int& value){
   if(act!=0){
      value=act->value;
      error=false;
   }
   else {
      error=true;
   }
}
```

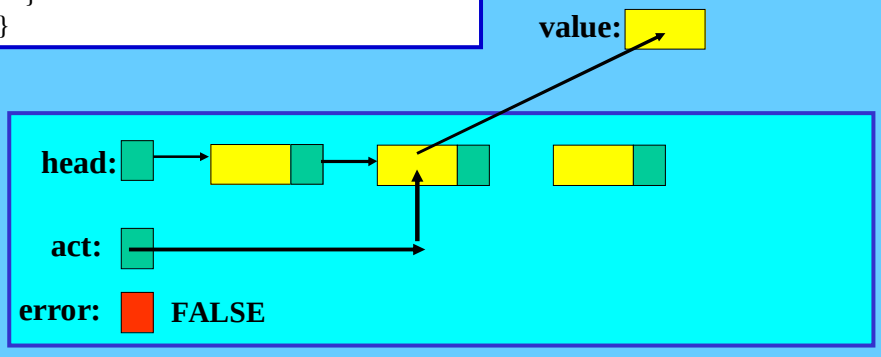
List.cpp:
getValue



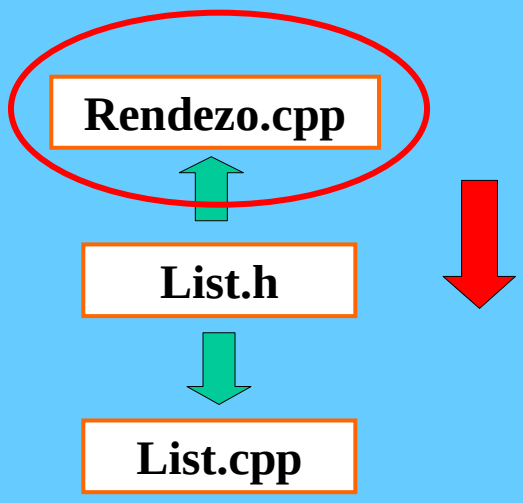
```
void List::getValue(int& value){
    if(act!=0){
        value=act->value;
        error=false;
    }
    else {
        error=true;
    }
}
```

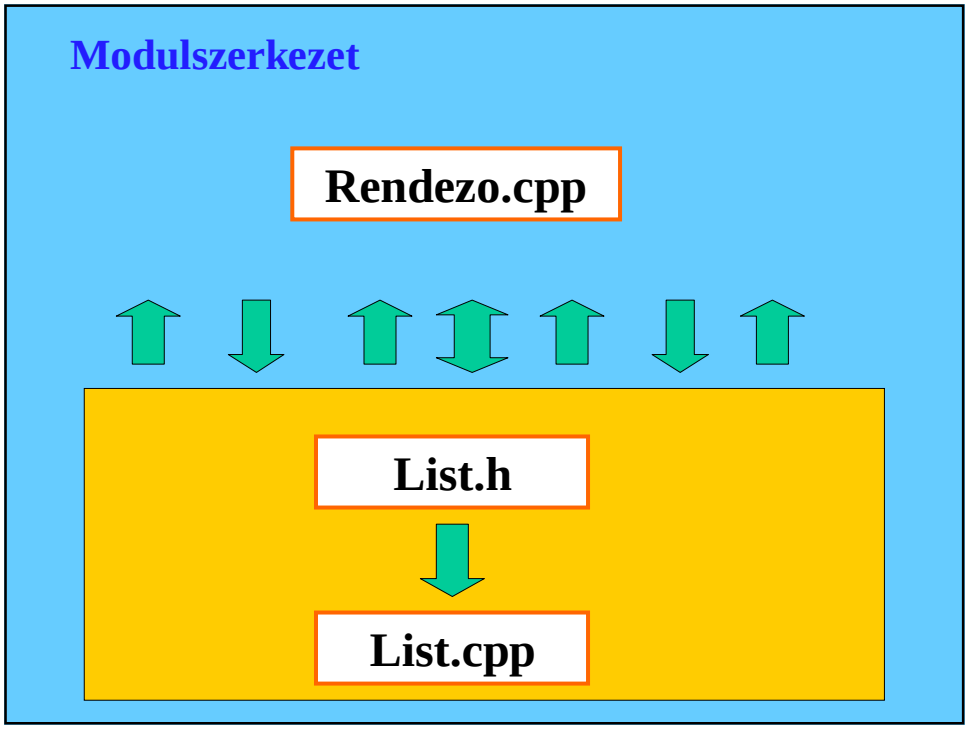
List.cpp:
getValue

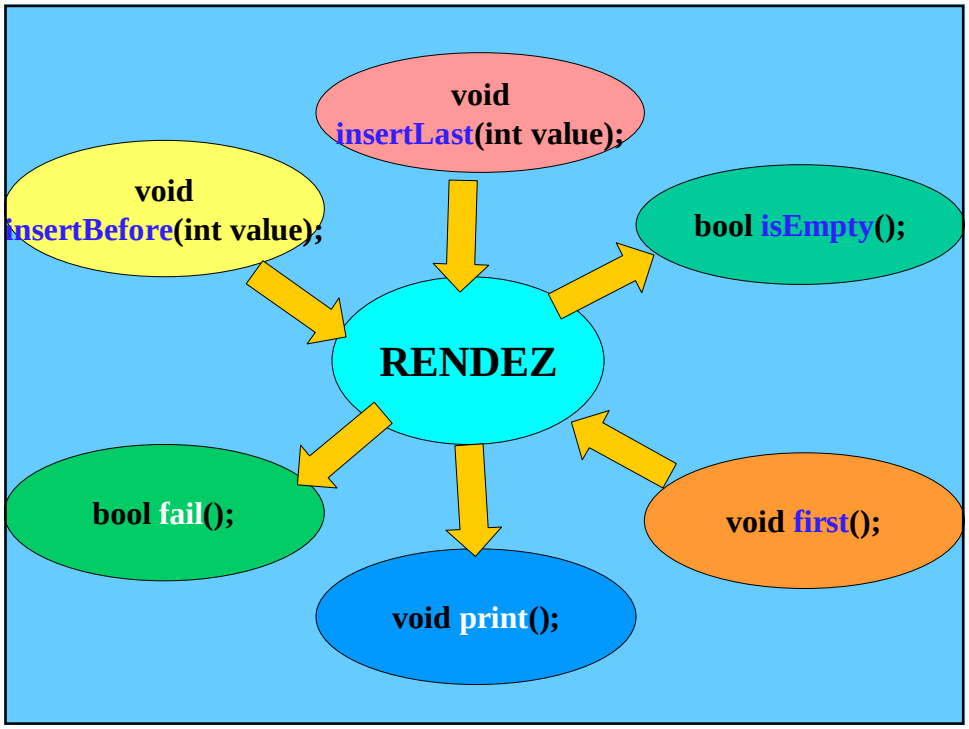
Kimenő paraméter!



Modulszerkezet



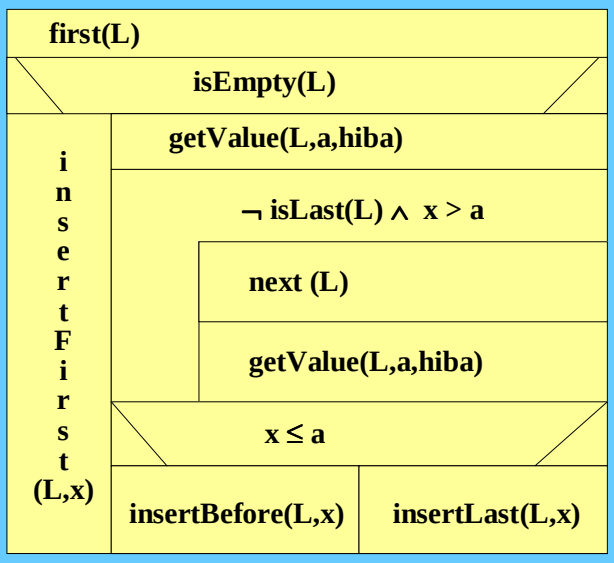

Modulszerkezet
Rendezo.cpp
List.h
List.cpp

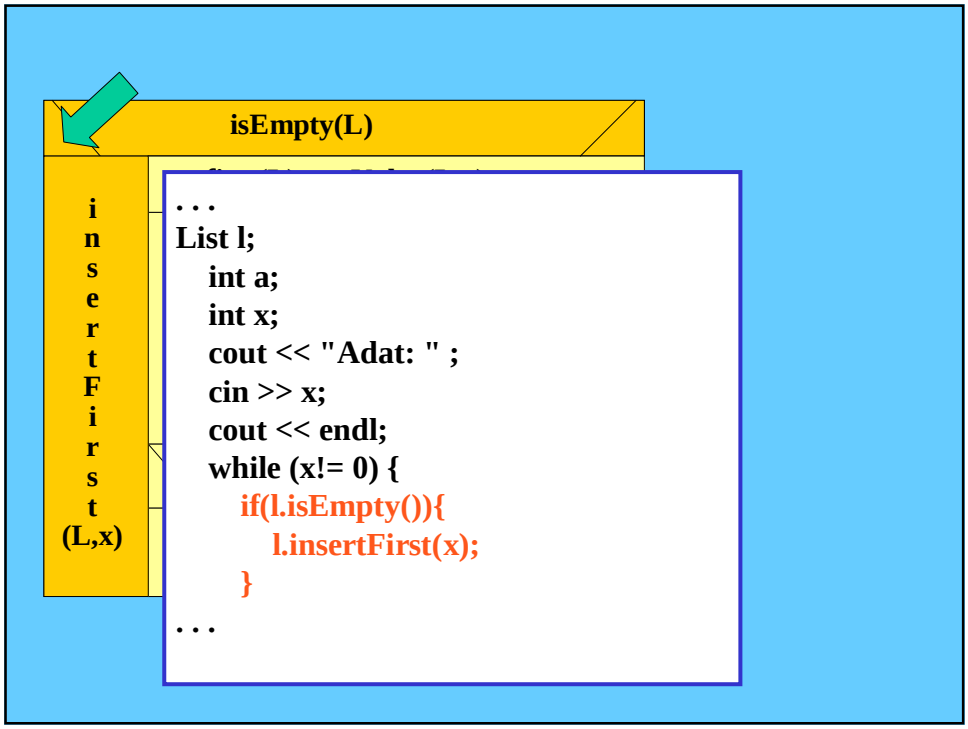

void insertLast(int value);
void insertBefore(int value);
bool isEmpty();
RENDEZ
bool fail();
void first();
void print();

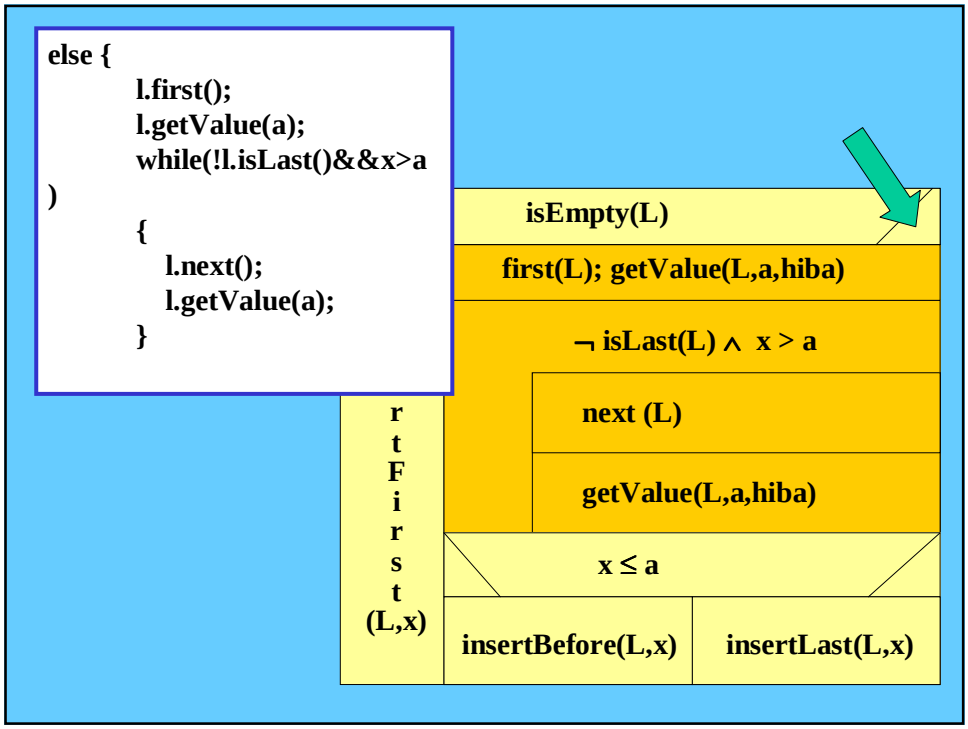
List.h

```
class List {
public:
   List();
   bool isEmpty();
   bool fail();
   void first();
  . . .
   void erase();
   void insertFirst(int value);
   void insertLast(int value);
   void insertBefore(int value);
   void insertAfter(int value);
   //Kiegészítő funkciók
   void print(); ...
```

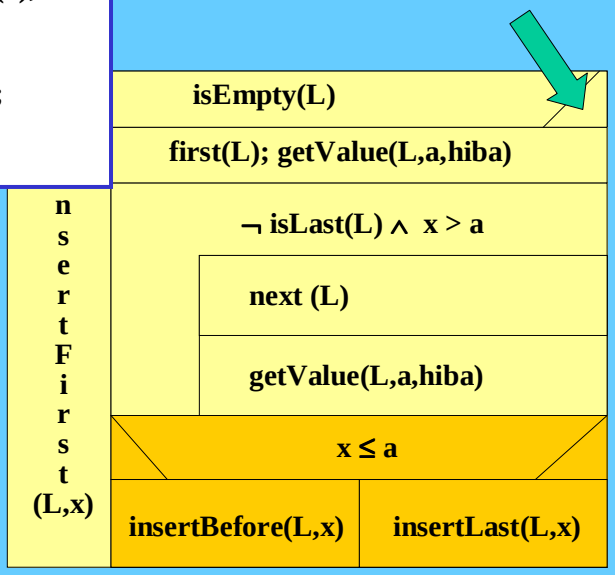
Egyszerű rendezés lista típus alkalmazásával



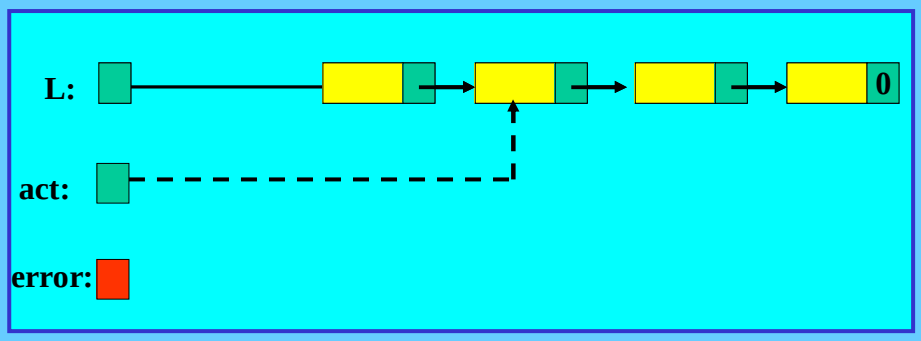

isEmpty(L)
insertFirst(L,x)
. . .
List l;
int a;
int x;
cout << "Adat: " ;
cin >> x;
cout << endl;
while (x!= 0) {
if(l.isEmpty()){
l.insertFirst(x);
}
. . .


else {
l.first();
l.getValue(a);
while(!l.isLast()&&x>a
)
{
l.next();
l.getValue(a);
}
isEmpty(L)
first(L); getValue(L,a,hiba)
¬ isLast(L) ∧ x > a
next (L)
getValue(L,a,hiba)
x ≤ a
insertBefore(L,x)
insertLast(L,x)
rtFirst(L,x)

```
if (x <= a) {
    l.insertBefore(x);
}
else {
    l.insertLast(x);
}
}
```

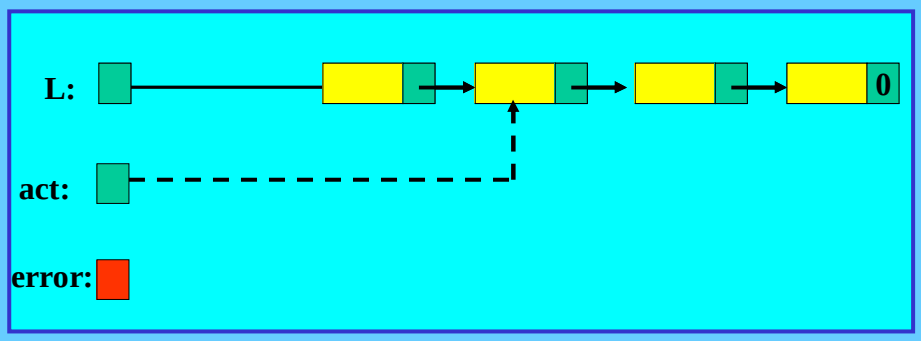


Egyirányú lista fejelem nélkül, aktuális mutatóval és hibakóddal



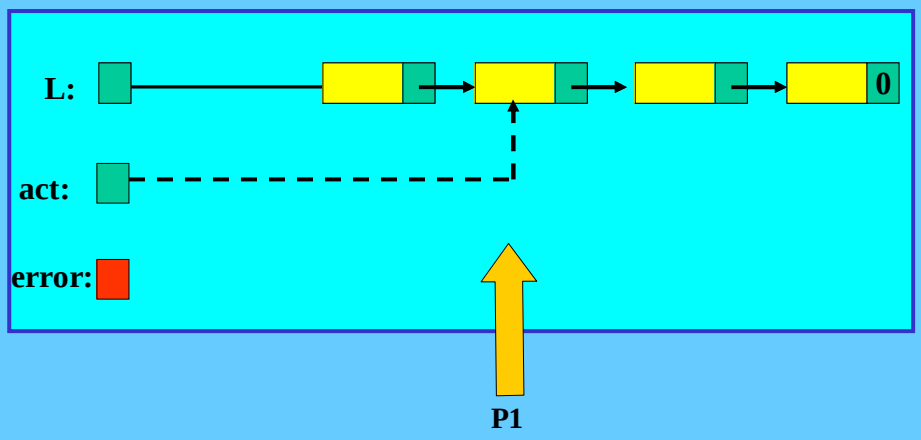
Kérdés:
Megoldhatunk-e ezzel a típussal olyan feladatokat, ahol egyszerre két mutatóval kell végigmenni a listán?

Egyirányú lista fejelem nélkül, aktuális mutatóval és hibakóddal



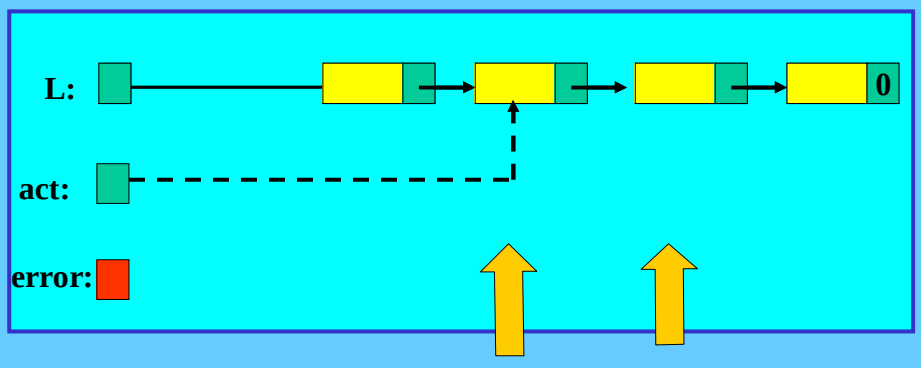
Például:
Legyenek a lista elemi szavak.
Keressük meg azt a szót, amely először fordul elő másodszor.

Egyirányú lista fejelem nélkül, aktuális mutatóval és hibakóddal



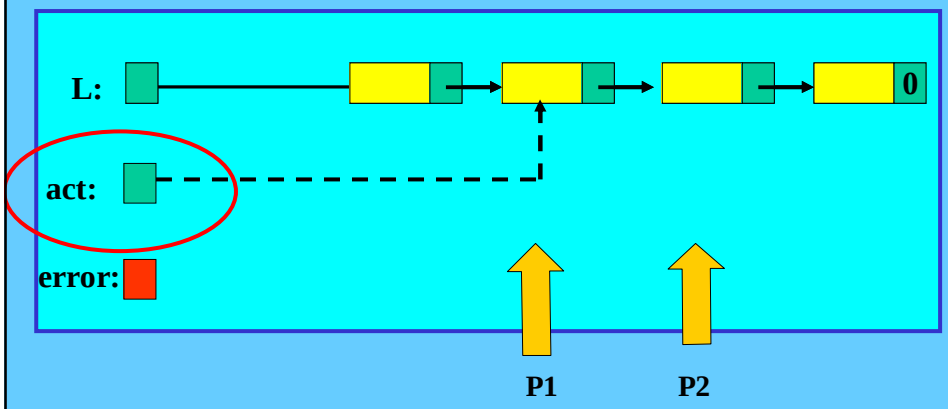
P1 mutasson a vizsgált szóra.

Egyirányú lista fejelem nélkül, aktuális mutatóval és hibakóddal



P2-vel keressük a szó párját.

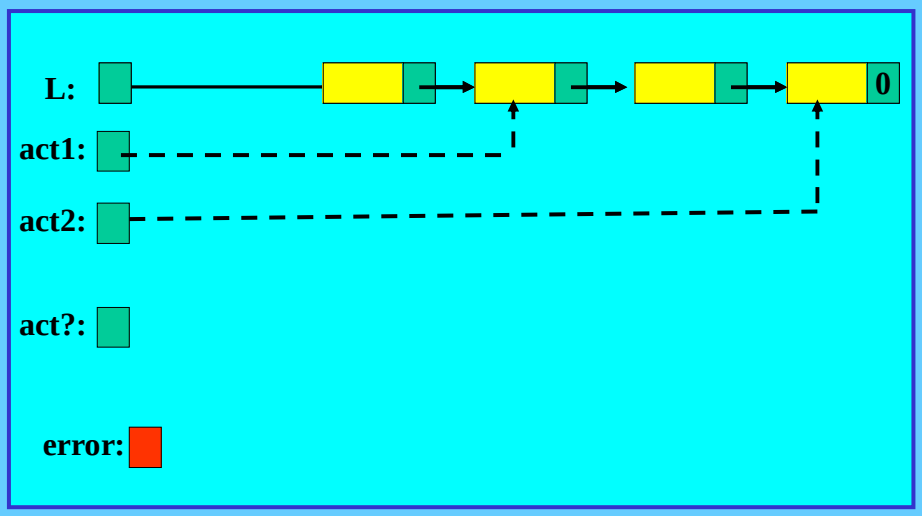
Egyirányú lista fejelem nélkül, aktuális mutatóval és hibakóddal



Mivel a típusban egyetlen aktuális mutatónk van, így ezt a feladatot ezzel a típussal nem tudjuk megoldani.

Megoldás?

Megoldás?



Megoldás?