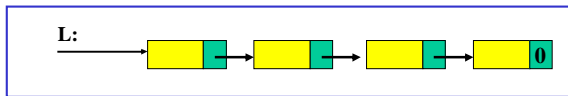


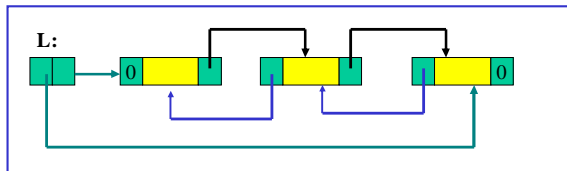
EGYSZERŰ LÁNCOLT LISTÁK.....	2
LISTAELEM.....	2
LISTAELEM BEFŰZÉSE.....	2
EGY EGYSZERŰ RENDEZETT LISTA.....	3
SIMPLESORT.CPP (RÉSZLET).....	3
EGYIRÁNYÚ LISTA, FEJ ELEM NÉLKÜL, MUTATÓT IS VISSZAADÓ METÓDUSOKKAL.....	5
SORTER.H.....	5
SIMPLELIST.H.....	5
EGYIRÁNYÚ LISTA FEJELEM NÉLKÜL, AKTUÁLIS MUTATÓVAL ÉS HIBAKÓDDAL	5
MODULSZERKEZET	5
LISTA TÍPUS: LIST.H.....	6
LISTA TÍPUS: LIST.CPP.....	7
EGYSZERŰ RENDEZÉS LISTA TÍPUS ALKALMAZÁSÁVAL - RENDEZO.CPP	9
RENDEZO.CPP	10
SIMPLESORT.CPP – TELJES PROGRAM.....	11
LIST.H – TELJES PROGRAM.....	12
LIST.CPP – TELJES PROGRAM	13
RENDEZO.CPP – TELJES PROGRAM	16

Egyszerű láncolt listák

Egyirányú láncolt lista

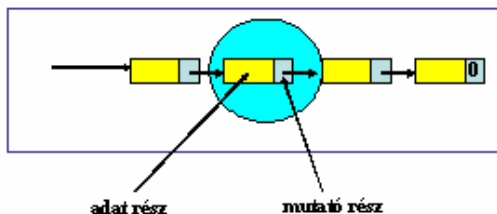


Kétirányú láncolt lista



 Saját jegyzetem

Listaelem



 Saját jegyzetem

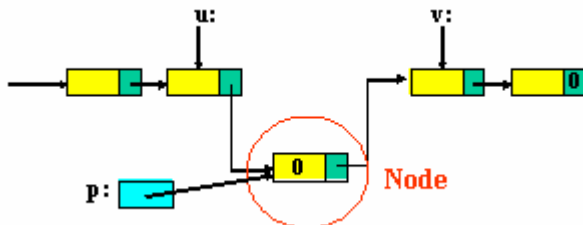
deklarálás

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

alkalmazás

```
Node* p=new Node;
Node* q=new Node(3);
Node* r=new Node(3,0);
```

Listaelem befűzése

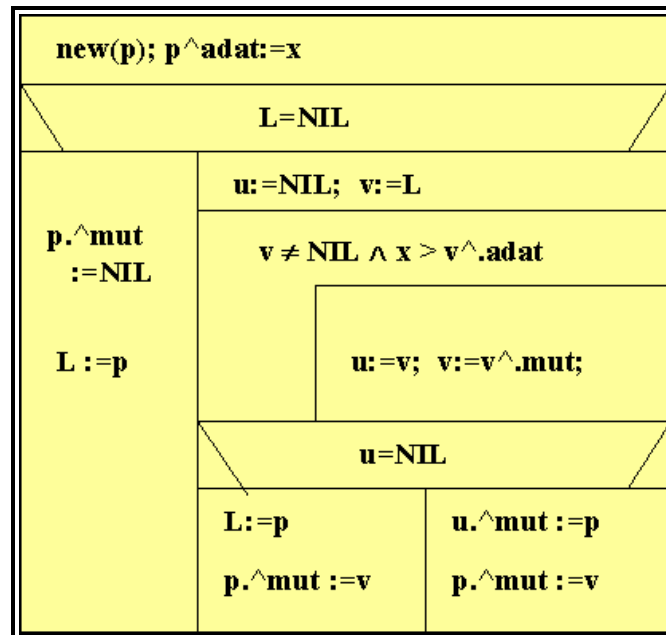


```
Node *u,*v,*p;
...
Node* p=new Node;
u->next=p;
p->next=v;
...
```

 Saját jegyzetem

Egy egyszerű rendezett lista

A feladat elkészítéséhez az adatszerkezetek órán megismert struktogrammot használjuk.



SimpleSort.cpp (részlet)

```
int main(){
//Listaelem
struct Node {
    int value;
    Node* next;
};
```

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

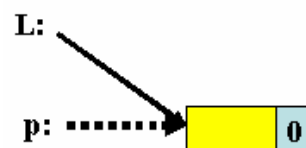
```
//Rendezett lista építés
Node* lista;
Node *p,*u,*v;
lista=0;
while(x!= -1) {
```



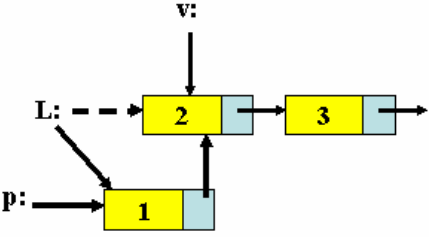

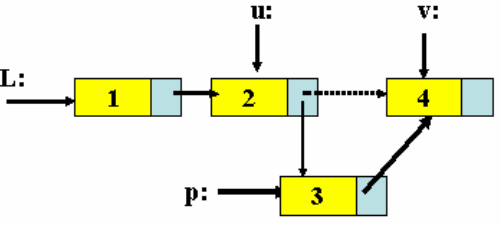
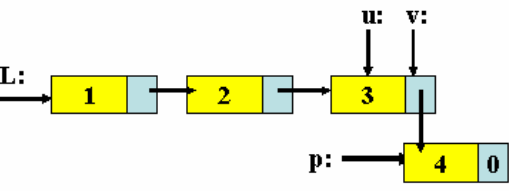

```
    p=new Node();
    p->value=x;
```

```
    if(lista==0) {
        p->next=0;
        lista=p;
    }
```


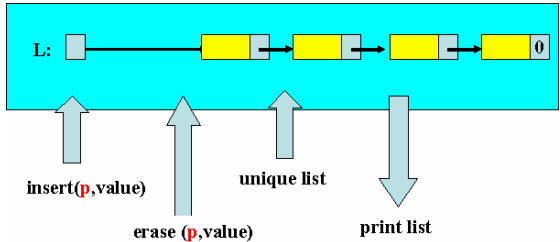



Beszúrás üres listába

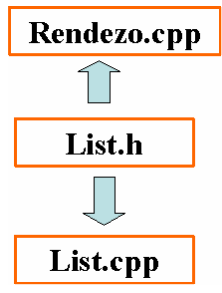


<pre> else { u=0; v=lista; while ((v!=0)&&(x>v->value)) { u=v; v=v->next; } </pre>	 <i>Beszúrás helyének megkeresése</i>
<pre> if (u==0) { lista=p; p->next=v; } </pre>	 <i>Beszúrás nem üres listába első elemként.</i> 
<pre> else { u->next=p; p->next=v; } </pre>	 <i>Beszúrás nem üres listába Belső elem</i>  <i>Utolsó elem</i> 
<pre> //A lista megjelenítése if(lista==0) { cout << "A lista üres!" << endl; } else { for(Node* q=lista; q->next!=0; q=q->next) { cout << q->value << "," ; } cout << q->value << endl ; } //Következő adat bekérése cout << "Adat: " ; cin >> x; cout << endl; } return 0; } </pre>	

Egyirányú lista, fej elem nélkül, mutatót is visszaadó metódusokkal

<p>Sorter.h</p> <pre> class Sorter { public: Sorter(); ~Sorter(); void insert(int value); void erase(int value); Sorter& unique(); void print(); private: List *L; }; </pre>	<p> <i>A p publikus.. Bárki hozzáférhet.</i></p>  <p> VESZÉLYES</p>
<p>SimpleList.h</p> <pre> class SimpleList { 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(); private: Node* head; }; </pre>	

Egyirányú lista fejelem nélkül, aktuális mutatóval és hibakóddal

<p>Modulszerkezet</p> 	
--	--

Lista típus: List.h

List.h	
<pre>#ifndef LIST_H #define LIST_H struct Node { int value; Node* next; Node(int i=0, Node *p=0) { value=i; next=p; } }; class List { public: List(); bool isEmpty(); bool fail(); void first(); void next(); bool isEndOfList(); bool isLast(); void setValue(int value); void getValue(int& value); void erase(); void insertFirst(int value); void insertLast(int value); void insertBefore(int value); void insertAfter(int value); //Kiegészítő funkciók void print(); private: Node* head; Node* act; bool error; }; #endif</pre>	<div>✎</div> <div>✎ ✎ ✎ Lista elem deklarációja</div> <div>✎ ✎ Metódusok deklarációja Nincs mutató. A művelet helyét private adattagba tettük. ✎</div> <div>✎ ✎ ✎ ✎ Lista elem deklarációja</div>

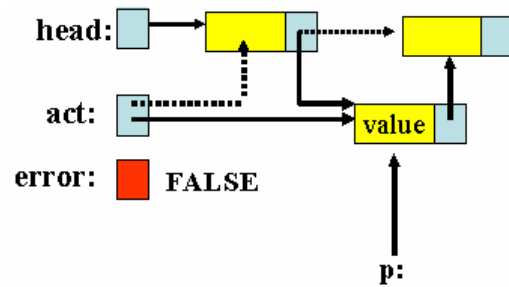
Lista típus: List.cpp

List.cpp	
<code>#include <iostream.h> using namespace std; #include "list.h"</code>	
<code>List::List() { head=0; act=0; error=false; }</code>	
<code>void List::insertFirst(int value) { Node* p=new Node; p->value=value; p->next=head; head=p; act=head; return; }</code>	
<code>void List::insertLast(int value) { Node* p=new Node; p->value=value; p->next=0; if (head == 0) { head=p; act=head; return; } else { Node* u=head; Node* v=head->next; while(v!=0) { u=v; v=v->next; } u->next=p; act=p; } return ; }</code>	<p>Üres a lista</p> <p>Nem üres a lista</p>
<code>bool List::isEmpty() { return(head==0); }</code>	

```

void List::insertAfter(int value) {
    if (act == 0) {
        error=true;
        return;
    }
    else {
        Node* p=new Node;
        p->value=value;
        p->next=act->next;
        act->next=p;
        error=false;
        return;
    }
}

```



```

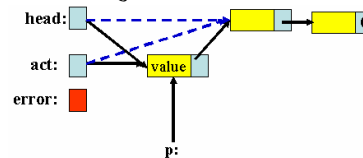
void List::insertBefore(int value) {
    if (act == 0) {
        error=true;
        return;
    }
    else if (act==head) {
        insertFirst(value);
    }

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

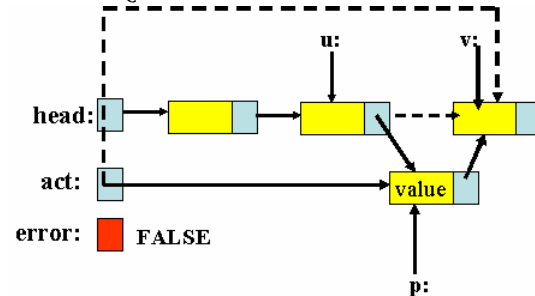
```

Ha üres lista

Első elemként



Belső elemként



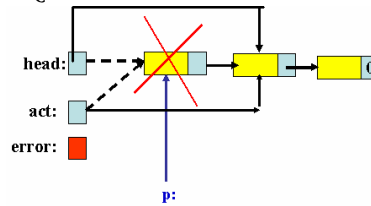

```

void List::erase() {
    if(act==0) {
        error=true;
        return;
    }
    else if (act=head) {
        Node* p=head;
        head=head->next;
        act=head;
        delete(p);
        error=false;
        return;
    }
    else {
        Node* p=head;
        while(p->next!=act) {
            p=p->next;
        }
        p->next=act->next;
        delete(act);
        act=p->next;
        error=false;
        return;
    }
}

```



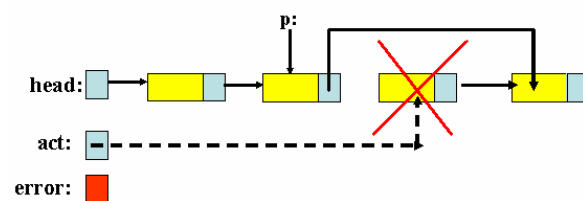
Első elemet kell törölni.



Hol van a megelőző elem?



Belső elemet kell törölni



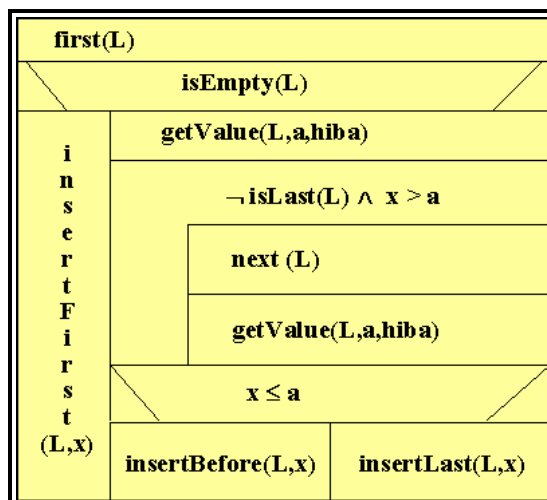
```

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

```

A value kimenő paraméter!!

Egyszerű rendezés lista típus alkalmazásával - Rendezo.cpp



Rendezo.cpp	
<pre> #include <iostream> using namespace std; #include "list.h" int main() { 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); } if (x<=a) { l.insertBefore(x); } else { l.insertAfter(x); } } l.print(); cout << "Adat: " ; cin >> x; cout << endl; } return 0; } </pre>	<div data-bbox="758 302 790 347"></div> <div data-bbox="758 347 790 392"></div> <p data-bbox="758 683 957 716"><i>Ha üres,, akkor ...</i></p> <p data-bbox="758 784 989 817"><i>Ha nem üres, akkor ...</i></p> <p data-bbox="758 1019 965 1052"><i>Hova kell beszúrni?</i></p> <div data-bbox="758 1052 790 1097"></div> <div data-bbox="758 1097 790 1142"></div>

SimpleSort.cpp – teljes program

```
//Fordítási direktívák
#include <iostream> using namespace std;
int main() {
//Listaelem
struct Node {
    int value;
    Node* next;
};
//Első adat bekérése
int x;
cout << "-1: vege. " << endl;
cout << "Adat: " ; cin >> x; cout << endl;
//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;
        }
    }
}
//A lista megjelenítése
if(lista==0) {
    cout << "A lista üres!" << endl;
}
else {
    for(Node* q=lista; q->next!=0; q=q->next) {
        cout << q->value << "," ;
    }
    cout << q->value << endl ;
}
//Következő adat bekérése
cout << "Adat: " ; cin >> x; cout << endl;
}
return 0;
}
```

List.h – teljes program

```
#ifndef LIST_H
#define LIST_H

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

class List
{
public:
    List();
    bool isEmpty();
    bool fail();
    void first();
    void next();
    bool isEndOfList();
    bool isLast();
    void setValue(int value);
    void getValue(int& value);
    void erase();
    void insertFirst(int value);
    void insertLast(int value);
    void insertBefore(int value);
    void insertAfter(int value);
    //Kiegészítő funkciók
    void print();

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

#endif
```

List.cpp – teljes program

```
#include <iostream> using namespace std;
#include "list.h"
List::List() {
    head=0;
    act=0;
    error=false;
}
//A visszatérési érték igaz, ha a lista üres
bool List::isEmpty() {
    return(head==0);
}
//Hiba lekérdezése. Lekérdezés után a hibaállapot törlődik
bool List::fail() {
    if (error){
        error=false;
        return true;
    }
    else
        return false;
}
//Első elemre állítás. Üres lista esetén hiba.
void List::first() {
    if (head!=0)
        act=head;
    else
        error=true;
}
//Rááll a következő elemre
void List::next() {
    if (act!=0)
        act=act->next;
}
//Lista vége lekérdezés
bool List::isEndOfList(){
    return (act==0);
}
//Utolsó elem lekérdezés
bool List::isLast() {
    return (act!=0 && act->next==0);
}
//Aktális elem értéke
void List::getValue(int& value){
    if(act!=0){
        value=act->value;
        error=false;
    }
    else {
        error=true;
    }
}
//Aktális elem beállítása (módosítása)
void List::setValue(int value){
    if(act!=0){
        act->value=value;
        error=false;
    }
    else {
        error=true;
    }
}
```

```
//Aktuális elem törlése. Ha nincs aktuális elem, akkor hiba.
```

```
//Törlés esetén az aktuális elem a törölt utáni elem lesz.
```

```
void List::erase() {
```

```
    if(act==0) {  
        error=true;  
        return;  
    }
```

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

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

```
}
```

```
//Beszúrás első elemként. Üres és nem üres listára egyaránt működik.
```

```
void List::insertFirst(int value) {
```

```
    Node* p=new Node;  
    p->value=value;  
    p->next=head;  
    head=p;  
    act=head;  
    return;  
}
```

```
//Beszúrás utolsó elemként. Üres és nem üres listára egyaránt működik.
```

```
void List::insertLast(int value) {
```

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

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

```
    u->next=p;  
    act=p;  
}
```

```
    return ;
```

```
}
```

```
//Beszúrás az aktuális elem mögé. Ha nincs aktuális elem vagy a lista üres, akkor hiba.
```

```
void List::insertAfter(int value) {
```

```
    if (act == 0) {
        error=true;
        return;
    }
    else {
        Node* p=new Node;
        p->value=value;
        p->next=act->next;
        act->next=p;
        error=false;
        return;
    }
}
```

```
//Beszúrás az aktuális elem elé. Ha nincs aktuális elem vagy a lista üres, akkor hiba.
```

```
void List::insertBefore(int value) {
```

```
    if (act == 0) {
        error=true;
        return;
    }
    else if (act==head) {
        insertFirst(value);
    }
    else {
        Node* p=new Node;
        p->value=value;
        Node* u=head;
        Node* v=head->next;
        while(v!=act) {
            u=v;
            v=v->next;
        }
        u->next=p;
        p->next=act;
        act=p;
        error=false;
        return;
    }
}
```

```
//A szabványos kimeneten megjeleníti a lista elemeit
```

```
void List::print() {
```

```
    Node* p=head;
    if (p != 0) {
        cout << "A lista: " ;
        while(p != 0) {
            cout << p->value << " ";
            p=p->next;
        }
        cout << endl;
    }
    else cout << "Üres lista!" << endl;
}
```

Rendezo.cpp – teljes program

```
#include <iostream> using namespace std;
#include "list.h"

int main()
{
    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);
            }
            if (x<=a) {
                l.insertBefore(x);
            }
            else {
                l.insertLast(x);
            }
        }
        l.print();
        cout << "Adat: " ;
        cin >> x;
        cout << endl;
    }
    return 0;
}
```