






































SimpleSort.cpp (részlet)	
<pre>int main(){ //Listaelem struct Node {     int value;     Node* next; };</pre>	
<pre>//Első adat bekérése int x; cout &lt;&lt; "-1: vege. " &lt;&lt; endl; cout &lt;&lt; "Adat: " ; cin &gt;&gt; x; cout &lt;&lt; endl;</pre>	
<pre>//Rendezett lista építés Node* lista; Node *p,*u,*v; lista=0; while(x!= -1) {</pre>	
<pre>    p=new Node();     p-&gt;value=x;</pre>	
<pre>    if(lista==0) {         p-&gt;next=0;         lista=p;     }</pre>	 <i>Beszúrás üres listába</i>
<pre>    else {         u=0;         v=lista;         while ((v!=0)&amp;&amp;(x&gt;v-&gt;value)) {             u=v;             v=v-&gt;next;         }</pre>	 <i>Beszúrás helyének megkeresése</i>
<pre>    if (u==0) {         lista=p;         p-&gt;next=v;     }</pre>	 <i>Nem üres lista első eleme</i>
<pre>    else {         u-&gt;next=p;         p-&gt;next=v;     } }</pre>	 <i>Beszúrás nem üres lista belsejébe</i>
<pre>//A lista megjelenítése if(lista==0) {     cout &lt;&lt; "A lista üres!" &lt;&lt; endl; } else {     for(Node* q=lista; q-&gt;next!=0; q=q-&gt;next) {         cout &lt;&lt; q-&gt;value &lt;&lt; "," ;     }     cout &lt;&lt; q-&gt;value &lt;&lt; endl ; } //Következő adat bekérése cout &lt;&lt; "Adat: " ; cin &gt;&gt; x; cout &lt;&lt; endl; } return 0; }</pre>	










Sorter.h	
<pre>class Sorter { public:     Sorter();     ~Sorter();     void insert(int value);     void erase(int value);     Sorter&amp; unique();     void print(); private:     List *L; };</pre>	<div></div> <div>:..... ..... ..... ..... .....</div>
SimpleList.h	
<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>	<div></div> <div>:..... ..... ..... ..... .....</div>






Saját jegyzeteim







Egyirányú lista fejelem nélkül, aktuális mutatóval és hibakóddal	
	 : ..... ..... ..... .....
<b>Modulszerkezet</b>	
	 : ..... ..... ..... .....
<b>List.h</b>	
<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&amp; 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>	   

<b>List.cpp</b>	
#include <iostream.h> using namespace std; #include "list.h"	
List::List() { head=0; act=0; error=false; }	
void List::insertFirst(int value) { Node* p=new Node; p->value=value; p->next=head; head=p; act=head; return; }	  : ..... .....
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 ; }	  <i>Üres lista?</i>  <i>Hol a vége?</i> 
bool List::isEmpty() { return(head==0); }	
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; } }	 

<pre>void List::insertBefore(int value) {     if (act == 0) {         error=true;         return;     }</pre>	 Ha üres lista
<pre>    else if (act==head) {         insertFirst(value);     }</pre>	 Aktuális mutató a lista végén van
<pre>    else {         Node* p=new Node;         p-&gt;value=value;         Node* u=head;         Node* v=head-&gt;next;         while(v!=act) {             u=v;             v=v-&gt;next;         }</pre>	 Hová kell beszúrní?
<pre>        u-&gt;next=p;         p-&gt;next=act;         act=p;         error=false;         return;     } }</pre>	
<pre>void List::erase() {     if(act==0) {         error=true;         return;     }</pre>	
<pre>    else if (act=head) {         Node* p=head;         head=head-&gt;next;         act=head;         delete(p);         error=false;         return;     }</pre>	
<pre>    else {         Node* p=head;         while(p-&gt;next!=act) {             p=p-&gt;next;         }</pre>	 Hol van a megelőző elem?
<pre>        p-&gt;next=act-&gt;next;         delete(act);         act=p-&gt;next;         error=false;         return;     } }</pre>	
<pre>void List::getValue(int&amp; value){     if(act!=0){         value=act-&gt;value;         error=false;     }     else {         error=true;     } }</pre>	 A value kimenő paraméter!!

Rendezo.cpp	
<pre> #include &lt;iostream&gt; using namespace std; #include "list.h"  int main() {     List l;     int a;     int x;     cout &lt;&lt; "Adat: "; cin &gt;&gt; x;     cout &lt;&lt; endl;     while (x!= 0) {         if(l.isEmpty()){             l.insertFirst(x);         }         else {             l.first();             l.getValue(a);             while(!l.isLast()&amp;&amp;x&gt;a ) {                 l.next();                 l.getValue(a);             }             if (x&lt;=a) {                 l.insertBefore(x);             }             else {                 l.insertAfter(x);             }         }         l.print();         cout &lt;&lt; "Adat: " ; cin &gt;&gt; x;         cout &lt;&lt; endl;     }     return 0; } </pre>	
	 Ha üres,, akkor
	 Hova kell beszúrni?
	
	

	Saját jegyzeteim	
		





**SimpleSort.cpp**

```
//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**

```
#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**

```
#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 beállítás (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**

```
#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;
}
```