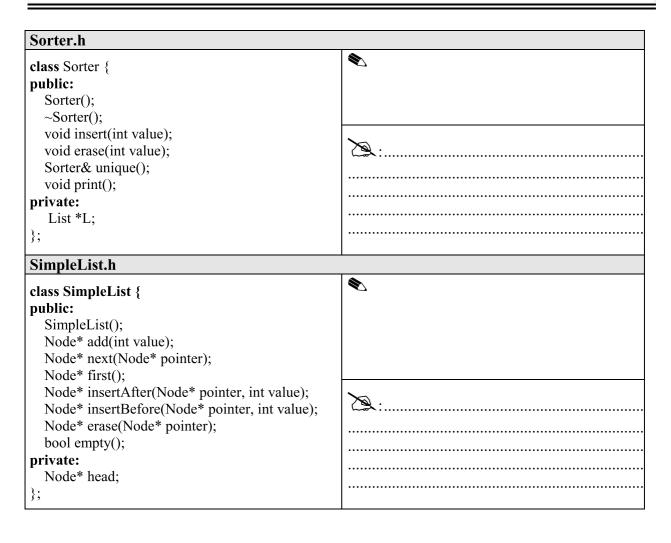
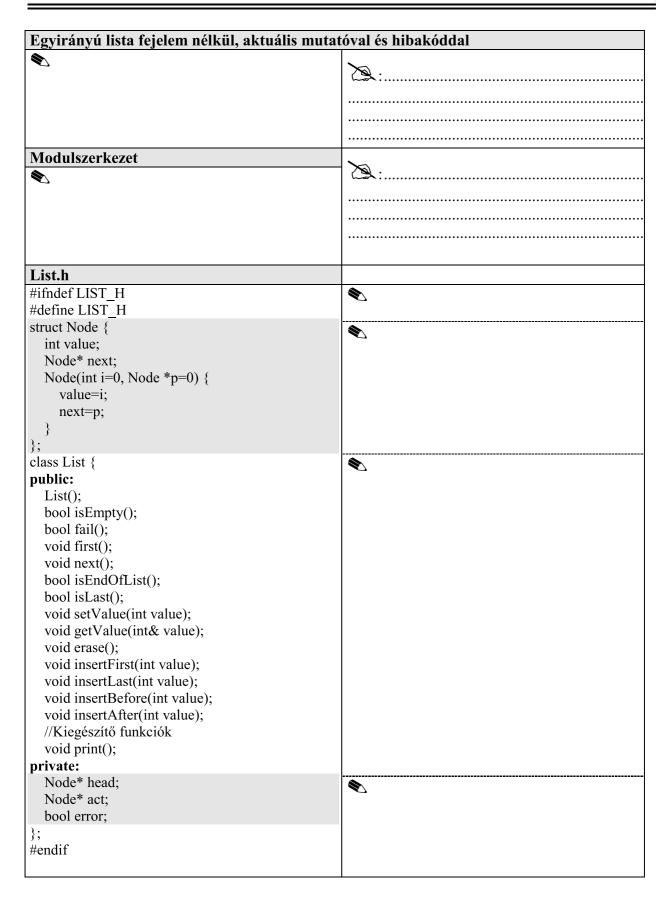
Egyszerű láncolt listák

Egyirányú lista		
&		
Listaelem		
₽		
deklarálás	alkalmazás	-
struct Node {	BERRYALIAN NO.	-
int value;	Node* p=new Node;	
Node* next;		
Node(int i=0, Node *p=0) {	Node* q=new Node(3);	
value=i;		
next=p;	Node* r=new Node(3,0);	
}		
};		
Listaelem befűzése		
		╡
Node *u,*v,*p;		
Node* p=new Node;		
u->next=p;		
p->next=v;		
	Saját jegyzeteim	
I		
I		
I		
I		
I		
I		
I		
i		

```
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) {
                                                    🌯 Beszúrás üres listába
      p - next = 0;
       lista=p;
    else {
                                                    Beszúrás helyének megkeresése
      u=0;
       v=lista;
       while ((v!=0)\&\&(x>v->value)) {
         u=v;
         v=v->next;
      if (u==0) {
                                                    Nem üres lista elsö eleme
         lista=p;
         p->next=v;
       else {
                                                    Beszúrás nem üres lista belsejébe
         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;
```

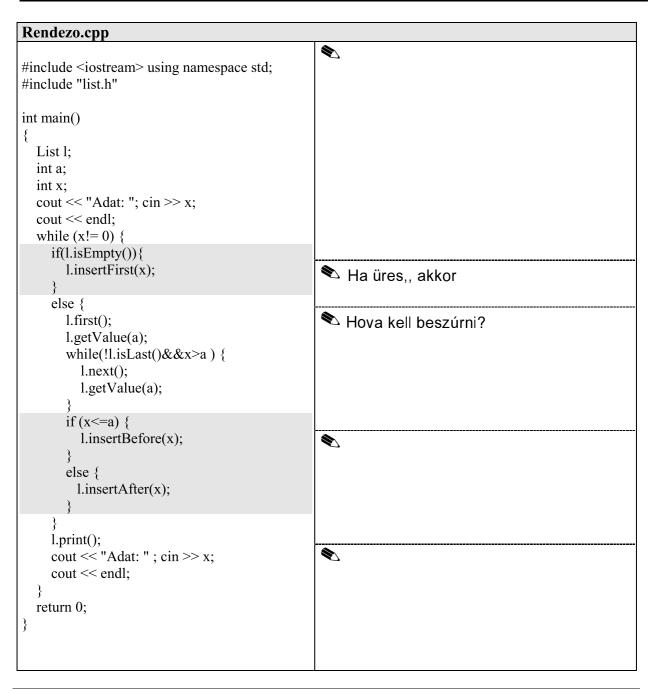






List.cpp	
#include <iostream.h> using namespace std;</iostream.h>	
#include "list.h"	
List::List() {	<u> </u>
head=0;	4 7
act=0;	
error=false;	
}	
void List::insertFirst(int value) {	
Node* p=new Node;	
p->value=value;	
p->next=head;	
head=p;	
act=head;	Q .
return;	
}	
void List::insertLast(int value) {	
Node* p=new Node;	
p->value=value;	
p->next=0;	
if (head == 0) {	🖎 Üres lista?
head=p;	
act=head;	
return;	
}	
else {	Nol a vége?
Node* u=head;	- Hora vege:
Node* v=head->next;	
while(v!=0) {	
u=v;	
v=v->next;	
}	
u->next=p;	
act=p;	
}	
return;	
}	
hool Listuis Empty (
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;	
}	
\ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \	

```
❤ Ha üres lista
void List::insertBefore(int value) {
  if (act == 0) {
    error=true;
    return;
  else if (act==head) {
                                                  🖎 Aktuális mutató a lista végén van
    insertFirst(value);
  else {
    Node* p=new Node;
                                                  Nová kell beszúrni?
    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;
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;
                                                  Nol van a megelözö elem?
       while(p->next!=act) {
         p=p->next;
      p->next=act->next;
      delete(act);
      act=p->next;
                                                  error=false;
      return;
void List::getValue(int& value){
                                                  A value kimenö paraméter!!
  if(act!=0){
    value=act->value;
    error=false;
  else {
    error=true;
  }
```





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;
        for(Node* q=lista; q->next!=0; q=q->next) {
          cout << q->value << ",";
       cout << q->value << endl;</pre>
     //Következő adat bekérése
     cout << "Adat: "; cin >> x; cout << endl;</pre>
  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 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\text{->}next\text{=}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\text{-}>value << " ";
       p=p->next;
     cout << endl;
  else cout << "Ures lista!" << endl;
```

Rendezo.cpp

```
#include <iostream> using namespace std;
#include "list.h"
int main()
  List 1;
  int a;
  int x;
  cout << "Adat: ";
  cin >> x;
  cout << endl;
  while (x!=0) {
     if(l.isEmpty()){
       1.insertFirst(x);
     else {
       1 first();
       1.getValue(a);
       while(!l.isLast() && x>a) {
          1.next();
          1.getValue(a);
        if (x \le a) {
          1.insertBefore(x);
       else {
         l.insertLast(x);
     1.print();
  cout << "Adat: ";
  cin >> x;
  cout << endl;
  return 0;
```