Programozás
(GKxB_INTM021)

Dr. Hatwagner F. Miklós

Széchenyi István Egyetem, Győr

2021. április 13.

Verem

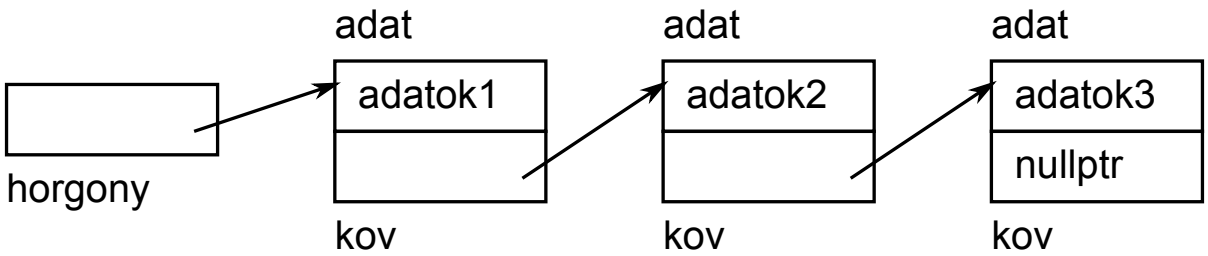
Verem megvalósítható egydimenziós tömbbel

Probléma:

a tömb mérete véges, fordítási időben adott

Lehetséges megoldás:

láncolt lista (Linked List, önhivatkozó adatszerkezet)



Egyszeresen láncolt lista (Singly Linked List)

Felhasználható struktúra

```
struct Lista1 {
    ADAT adat;
    Lista1 *kov;
};
```

Verem

veremTeszt3.cpp

```
#include <iostream>
#include "verem3.h"
using namespace std;

#define N 5
int main() {
  cout << N << " egesz verembe rakasa: ";
  for(int i=0; i<N; i++) {
    cout << i << '\t';
    berak(i);
  }
  cout << "\nVisszaolvasva:\t\t";
  while(not ures()) {
    cout << kivesz() << '\t';
  }
  cout << endl;
  return 0;
}
```

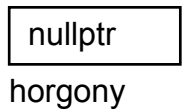
verem3.h

```
struct Lista1 {
  int adat;
  Lista1* kov;
};
```

Verem – berak()

verem3.cpp

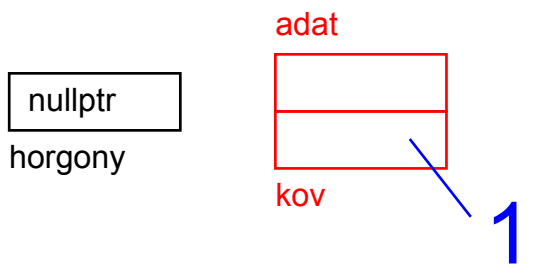
```
5  static Lista1* horgony = nullptr;
6
7  bool berak(int adat) {
8    Lista1* uj = new Lista1;
9    if(uj != nullptr) {
10     uj->adat = adat;
11     uj->kov = horgony;
12     horgony = uj;
13     return true;
14   } else {
15     return false;
16   }
17 }
```



Verem – berak()

verem3.cpp

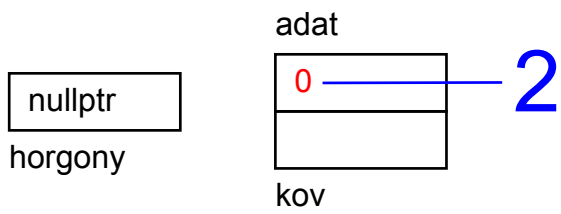
```
5  static Lista1* horgony = nullptr;
6
7  bool berak(int adat) {
8    Lista1* uj = new Lista1;
9    if(uj != nullptr) {
10     uj->adat = adat;
11     uj->kov = horgony;
12     horgony = uj;
13     return true;
14   } else {
15     return false;
16   }
17 }
```



Verem – berak()

verem3.cpp

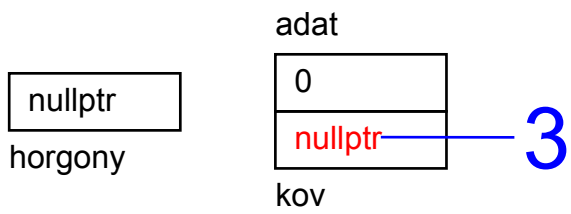
```
5  static Lista1* horgony = nullptr;
6
7  bool berak(int adat) {
8    Lista1* uj = new Lista1;
9    if(uj != nullptr) {
10     uj->adat = adat;
11     uj->kov = horgony;
12     horgony = uj;
13     return true;
14   } else {
15     return false;
16   }
17 }
```



Verem – berak()

verem3.cpp

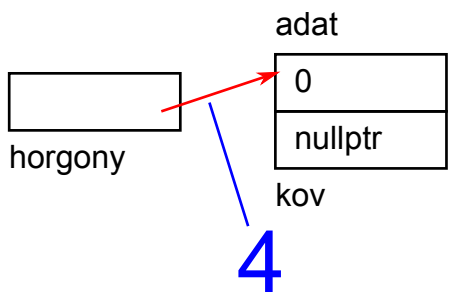
```
5  static Lista1* horgony = nullptr;
6
7  bool berak(int adat) {
8    Lista1* uj = new Lista1;
9    if(uj != nullptr) {
10     uj->adat = adat;
11     uj->kov = horgony;
12     horgony = uj;
13     return true;
14   } else {
15     return false;
16   }
17 }
```



Verem – berak()

verem3.cpp

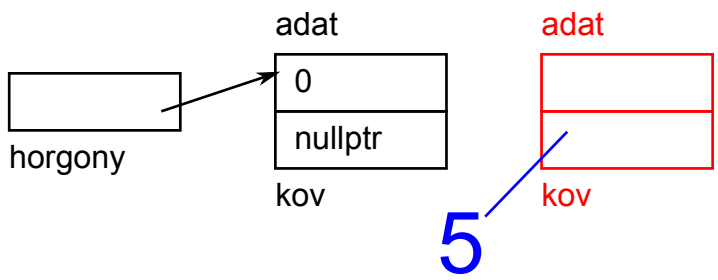
```
5  static Lista1* horgony = nullptr;
6
7  bool berak(int adat) {
8    Lista1* uj = new Lista1;
9    if(uj != nullptr) {
10     uj->adat = adat;
11     uj->kov = horgony;
12     horgony = uj;
13     return true;
14   } else {
15     return false;
16   }
17 }
```



Verem – berak()

verem3.cpp

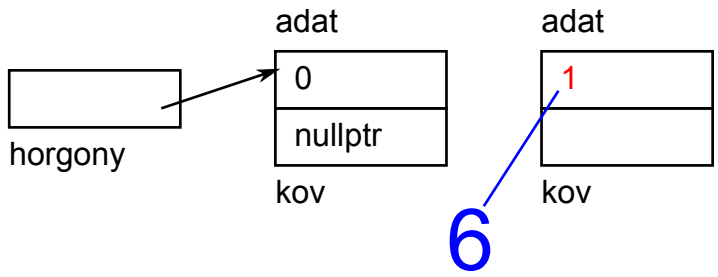
```
5  static Lista1* horgony = nullptr;
6
7  bool berak(int adat) {
8    Lista1* uj = new Lista1;
9    if(uj != nullptr) {
10     uj->adat = adat;
11     uj->kov = horgony;
12     horgony = uj;
13     return true;
14   } else {
15     return false;
16   }
17 }
```



Verem – berak()

verem3.cpp

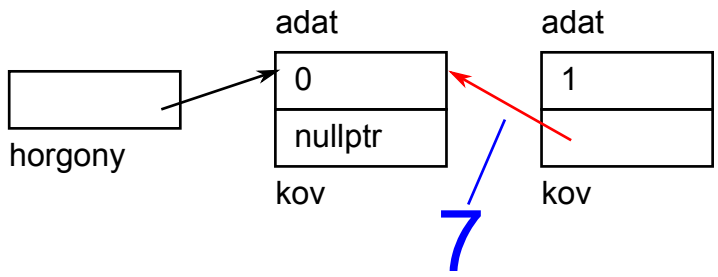
```
5  static Lista1* horgony = nullptr;
6
7  bool berak(int adat) {
8    Lista1* uj = new Lista1;
9    if(uj != nullptr) {
10     uj->adat = adat;
11     uj->kov = horgony;
12     horgony = uj;
13     return true;
14   } else {
15     return false;
16   }
17 }
```



Verem – berak()

verem3.cpp

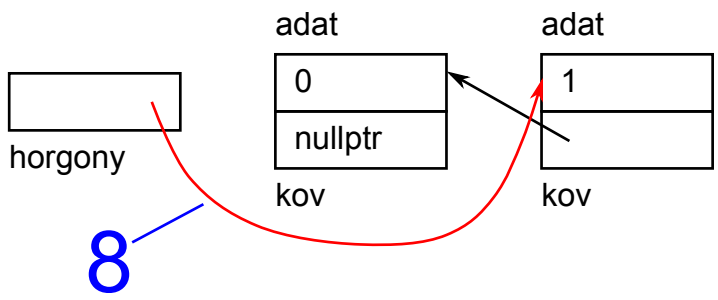
```
5  static Lista1* horgony = nullptr;
6
7  bool berak(int adat) {
8    Lista1* uj = new Lista1;
9    if(uj != nullptr) {
10     uj->adat = adat;
11     uj->kov = horgony;
12     horgony = uj;
13     return true;
14   } else {
15     return false;
16   }
17 }
```



Verem – berak()

verem3.cpp

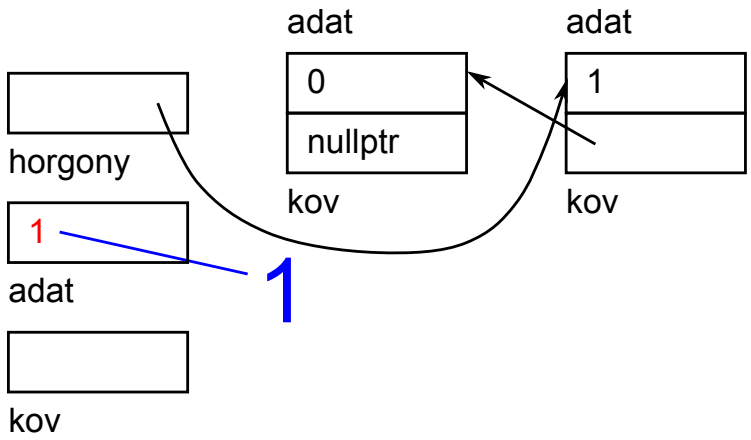
```
5  static Lista1* horgony = nullptr;
6
7  bool berak(int adat) {
8    Lista1* uj = new Lista1;
9    if(uj != nullptr) {
10     uj->adat = adat;
11     uj->kov = horgony;
12     horgony = uj;
13     return true;
14   } else {
15     return false;
16   }
17 }
```



Verem – kivesz()

verem3.cpp

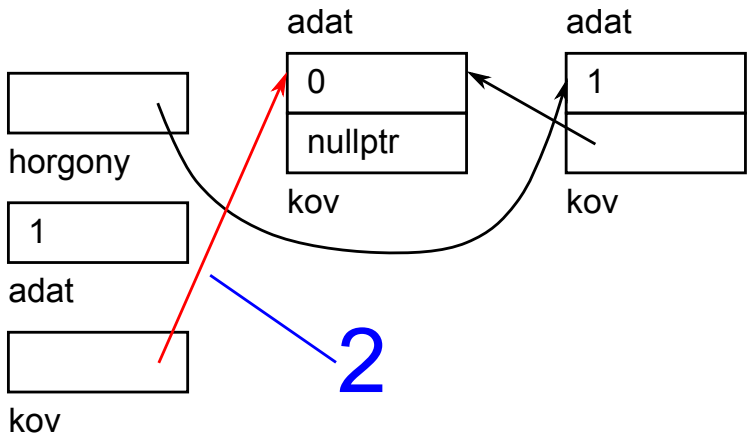
```
19  int kivesz() {
20    if(horgony == nullptr) {
21      std::cerr<<"A verem ures.\n";
22      return 0;
23    } else {
24      int adat = horgony->adat;
25      Lista1* kov = horgony->kov;
26      delete horgony;
27      horgony = kov;
28      return adat;
29    }
30  }
```



Verem – kivesz()

verem3.cpp

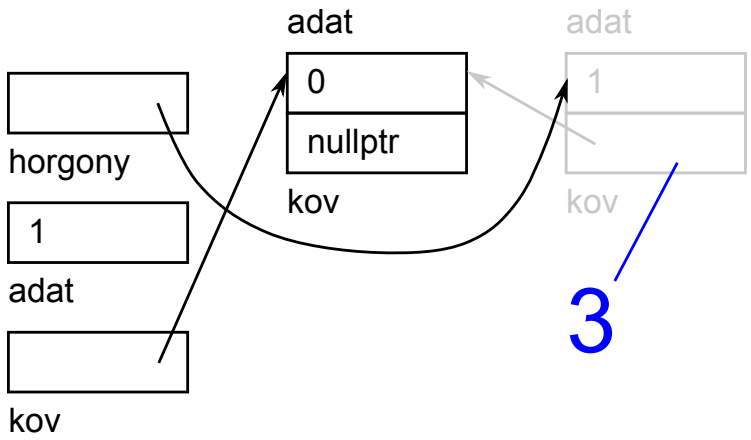
```
19  int kivesz() {
20    if(horgony == nullptr) {
21      std::cerr<<"A verem ures.\n";
22      return 0;
23    } else {
24      int adat = horgony->adat;
25      Lista1* kov = horgony->kov;
26      delete horgony;
27      horgony = kov;
28      return adat;
29    }
30  }
```



Verem – kivesz()

verem3.cpp

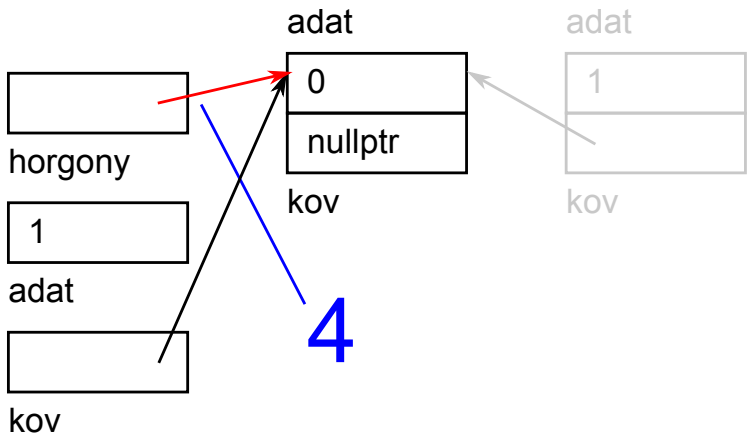
```
19  int kivesz() {
20    if(horgony == nullptr) {
21      std::cerr<<"A verem ures.\n";
22      return 0;
23    } else {
24      int adat = horgony->adat;
25      Lista1* kov = horgony->kov;
26      delete horgony;
27      horgony = kov;
28      return adat;
29    }
30  }
```



Verem – kivesz()

verem3.cpp

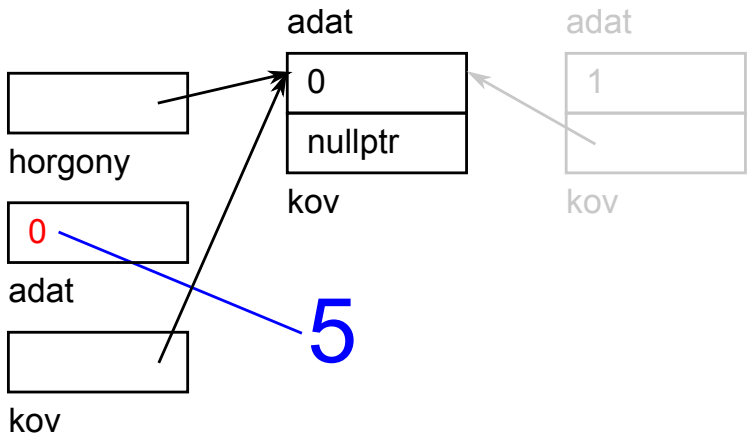
```
19 int kivesz() {
20   if(horgony == nullptr) {
21     std::cerr<<"A verem ures.\n";
22     return 0;
23   } else {
24     int adat = horgony->adat;
25     Lista1* kov = horgony->kov;
26     delete horgony;
27     horgony = kov;
28     return adat;
29   }
30 }
```



Verem – kivesz()

verem3.cpp

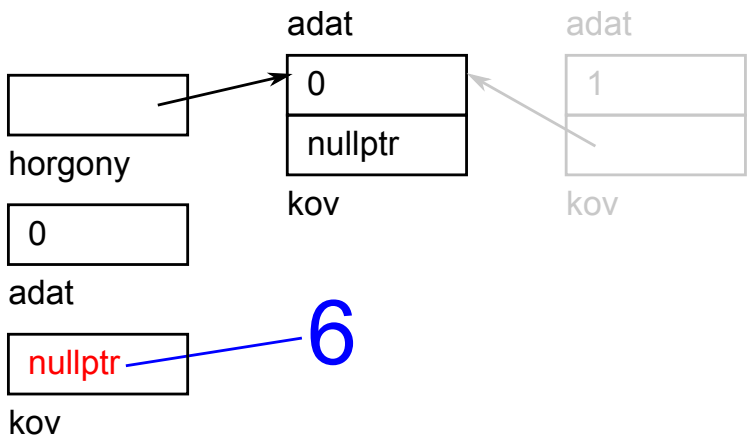
```
19 int kivesz() {
20   if(horgony == nullptr) {
21     std::cerr<<"A verem ures.\n";
22     return 0;
23   } else {
24     int adat = horgony->adat;
25     Lista1* kov = horgony->kov;
26     delete horgony;
27     horgony = kov;
28     return adat;
29   }
30 }
```



Verem – kivesz()

verem3.cpp

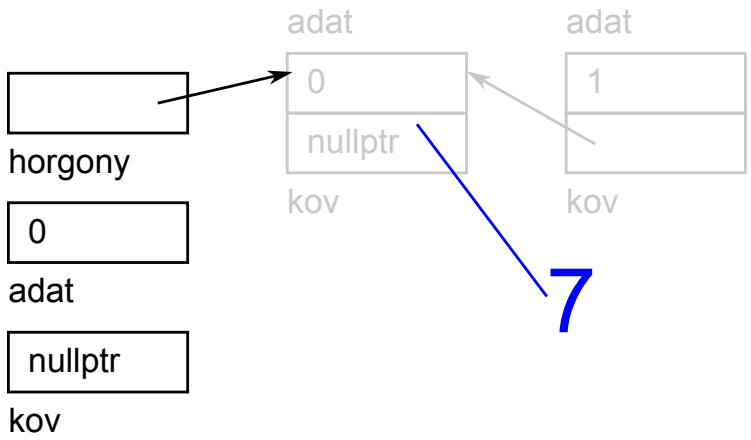
```
19 int kivesz() {
20   if(horgony == nullptr) {
21     std::cerr<<"A verem ures.\n";
22     return 0;
23   } else {
24     int adat = horgony->adat;
25     Lista1* kov = horgony->kov;
26     delete horgony;
27     horgony = kov;
28     return adat;
29   }
30 }
```



Verem – kivesz()

verem3.cpp

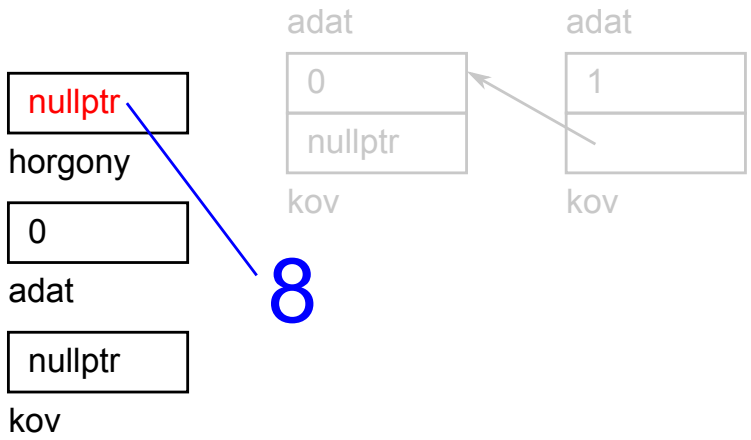
```
19 int kivesz() {
20   if(horgony == nullptr) {
21     std::cerr<<"A verem ures.\n";
22     return 0;
23   } else {
24     int adat = horgony->adat;
25     Lista1* kov = horgony->kov;
26     delete horgony;
27     horgony = kov;
28     return adat;
29   }
30 }
```



Verem – kivesz()

verem3.cpp

```
19  int kivesz() {
20    if(horgony == nullptr) {
21      std::cerr<<"A verem ures.\n";
22      return 0;
23    } else {
24      int adat = horgony->adat;
25      Lista1* kov = horgony->kov;
26      delete horgony;
27      horgony = kov;
28      return adat;
29    }
30  }
```



Egyszeresen láncolt lista

Készítsünk általánosan használható függvényeket egyszeresen láncolt lista manipulálásához!

listaTeszt1.cpp (Lista1.cpp, Lista1.h)

```
#include <iostream>
#include <cstddef> // nullptr mutato
#include "Lista1.h"
using namespace std;

int main() {
  cout << "Adjon meg egeszeket, leallas negativ szamra!\n";
  Lista1 *horgony=nullptr, *seged=nullptr;
  int szam;
  while(cin>>szam, szam>=0) {
    seged = beszur1(szam, seged);
    if(horgony == nullptr) horgony = seged;
  }
  cout << "Ezeket adta meg:\n";
  kiir1(horgony);
  torolMindet1(horgony);
  return 0;
}
```

Egyszeresen láncolt lista – beszúrás

Lista1.cpp

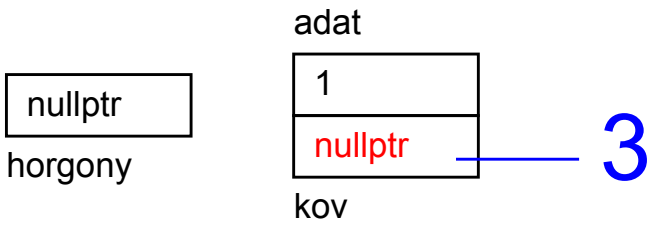
```
5  // 'elozo' utan beszur egy uj elemet
6  Lista1 *beszur1(int adat, Lista1 *elozo) {
7    Lista1* uj = new Lista1;
8    if(uj) { // if(uj != nullptr) { //...
9      uj->adat = adat;
10     if(elozo) {
11       uj->kov = elozo->kov;
12       elozo->kov = uj;
13     } else {
14       uj->kov = nullptr;
15     }
16   }
17   return uj;
18 }
```

Egyszeresen láncolt lista – beszúrás

Lista1.cpp

```
5  // 'elozo' utan beszur egy uj elemet
6  Lista1 *beszur1(int adat, Lista1 *elozo) {
7    Lista1* uj = new Lista1;
8    if(uj) { // if(uj != nullptr) { //...
9      uj->adat = adat;
10     if(elozo) {
11       uj->kov = elozo->kov;
12       elozo->kov = uj;
13     } else {
14       uj->kov = nullptr;
15     }
16   }
17   return uj;
18 }
```

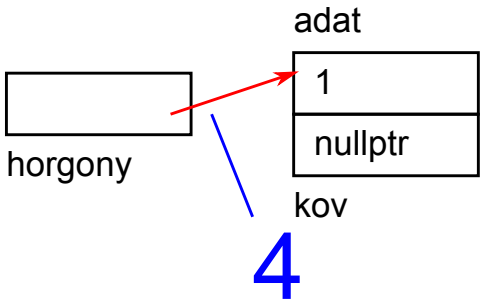
Egyszeresen láncolt lista – beszúrás



Lista1.cpp

```
5  // 'elozo' utan beszur egy uj elemet
6  Lista1 *beszur1(int adat, Lista1 *elozo) {
7    Lista1* uj = new Lista1;
8    if(uj) { // if(uj != nullptr) { //...
9      uj->adat = adat;
10     if(elozo) {
11       uj->kov = elozo->kov;
12       elozo->kov = uj;
13     } else {
14       uj->kov = nullptr;
15     }
16   }
17   return uj;
18 }
```

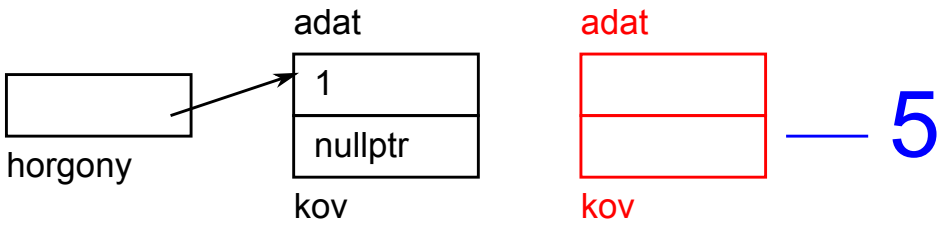
Egyszeresen láncolt lista – beszúrás



listaTeszt1.cpp

```
6   int main() {
7     cout << "Adjon meg egeszeket, leallas negativ szamra!\n";
8     Lista1 *horgony=nullptr, *seged=nullptr;
9     int szam;
10    while(cin>>szam, szam>=0) {
11      seged = beszur1(szam, seged);
12      if(horgony == nullptr) horgony = seged;
13    }
14    cout << "Ezeket adta meg:\n";
15    kiir1(horgony);
16    torolMindet1(horgony);
17    return 0;
18  }
```

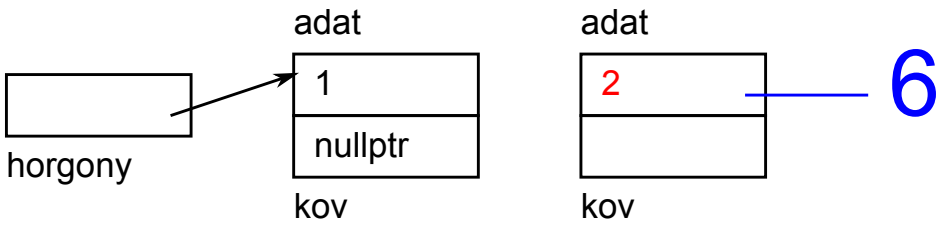
Egyszeresen láncolt lista – beszúrás



Lista1.cpp

```
5  // 'elozo' utan beszur egy uj elemet
6  Lista1 *beszur1(int adat, Lista1 *elozo) {
7    Lista1* uj = new Lista1;
8    if(uj) { // if(uj != nullptr) { //...
9      uj->adat = adat;
10     if(elozo) {
11       uj->kov = elozo->kov;
12       elozo->kov = uj;
13     } else {
14       uj->kov = nullptr;
15     }
16   }
17   return uj;
18 }
```

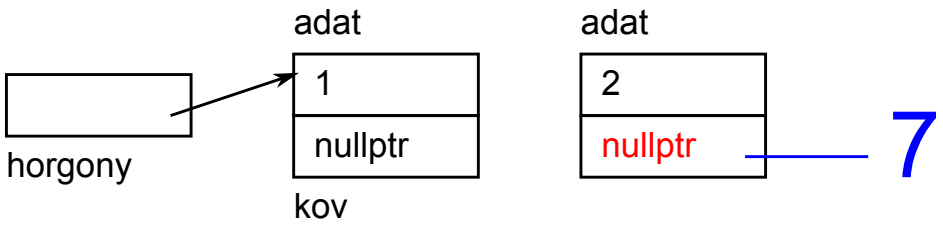
Egyszeresen láncolt lista – beszúrás



Lista1.cpp

```
5  // 'elozo' utan beszur egy uj elemet
6  Lista1 *beszur1(int adat, Lista1 *elozo) {
7    Lista1* uj = new Lista1;
8    if(uj) { // if(uj != nullptr) { //...
9      uj->adat = adat;
10     if(elozo) {
11       uj->kov = elozo->kov;
12       elozo->kov = uj;
13     } else {
14       uj->kov = nullptr;
15     }
16   }
17   return uj;
18 }
```

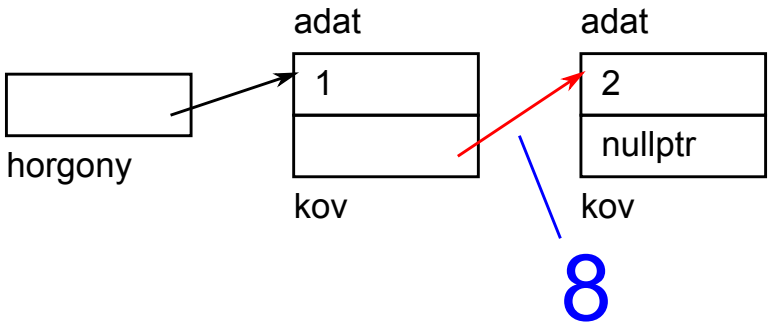
Egyszeresen láncolt lista – beszúrás



Lista1.cpp

```
5  // 'elozo' utan beszur egy uj elemet
6  Lista1 *beszur1(int adat, Lista1 *elozo) {
7    Lista1* uj = new Lista1;
8    if(uj) { // if(uj != nullptr) { //...
9      uj->adat = adat;
10     if(elozo) {
11       uj->kov = elozo->kov;
12       elozo->kov = uj;
13     } else {
14       uj->kov = nullptr;
15     }
16   }
17   return uj;
18 }
```

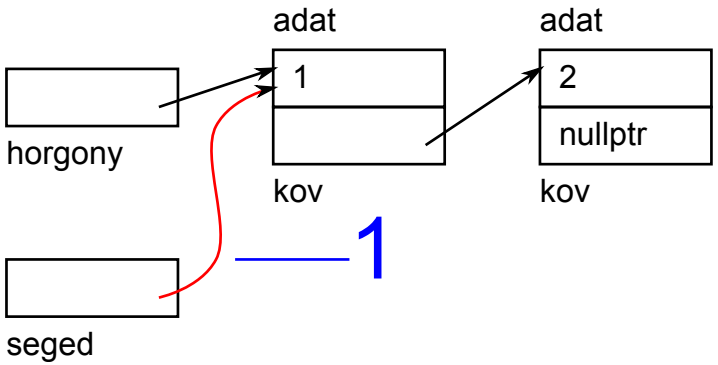
Egyszeresen láncolt lista – beszúrás

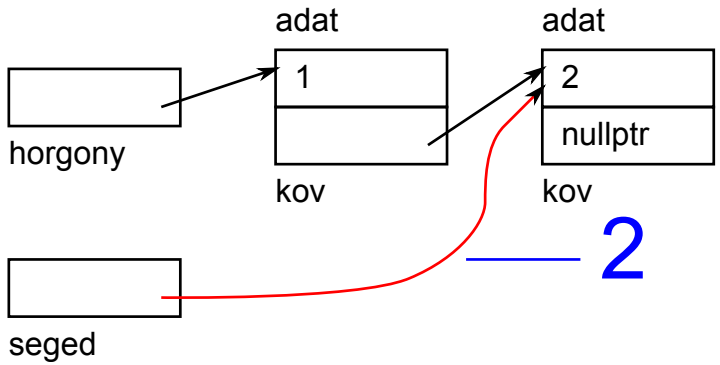


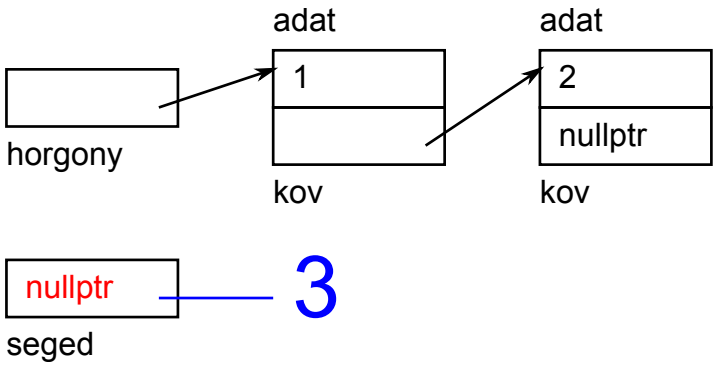
listaTeszt1.cpp

```
6  int main() {
7    cout << "Adjon meg egeszeket, leallas negativ szamra!\n";
8    Lista1 *horgony=nullptr, *seged=nullptr;
9    int szam;
10   while(cin>>szam, szam>=0) {
11     seged = beszur1(szam, seged);
12     if(horgony == nullptr) horgony = seged;
13   }
14   cout << "Ezeket adta meg:\n";
15   kiir1(horgony);
16   torolMindet1(horgony);
17   return 0;
18 }
```

Egyszeresen láncolt lista – bejárás



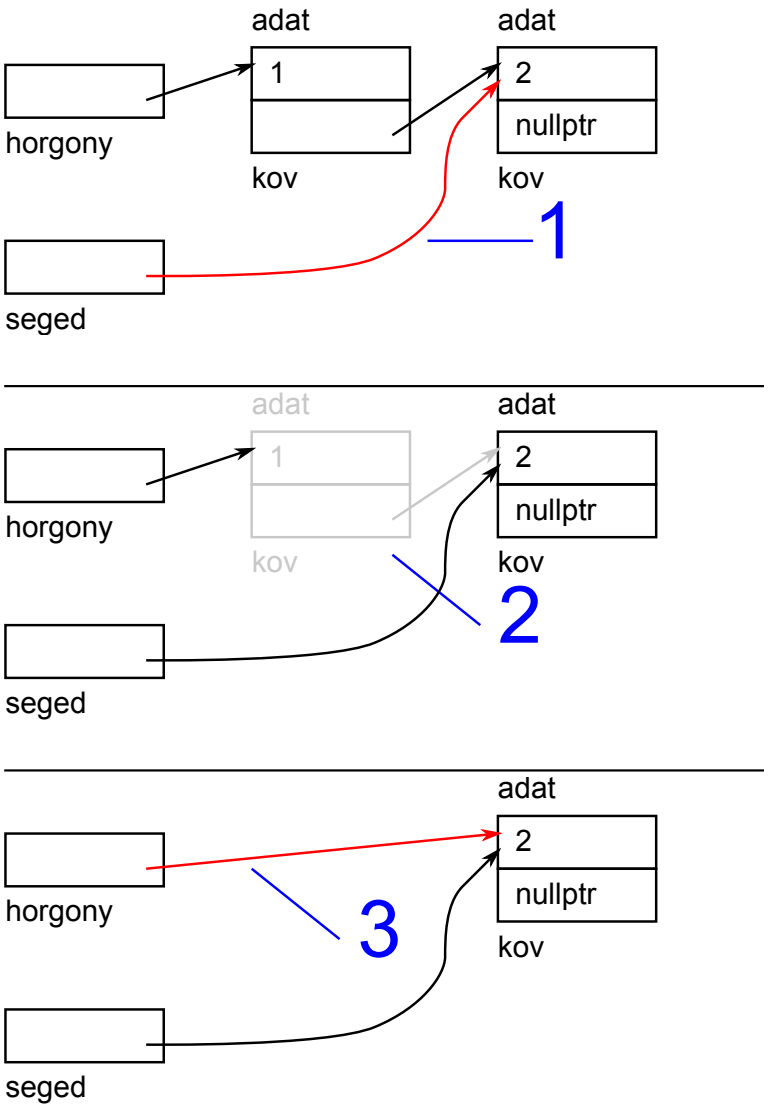




Lista1.cpp

```
// kiirja a lista osszes elemet                    20
void kiir1(Lista1 *horgony) {                      21
  Lista1 *seged;                                   22
  for(seged=horgony; seged;                        23
      seged=seged->kov) {                          24
    std::cout << seged->adat                       25
              << '\t';                             26
  }                                                27
```

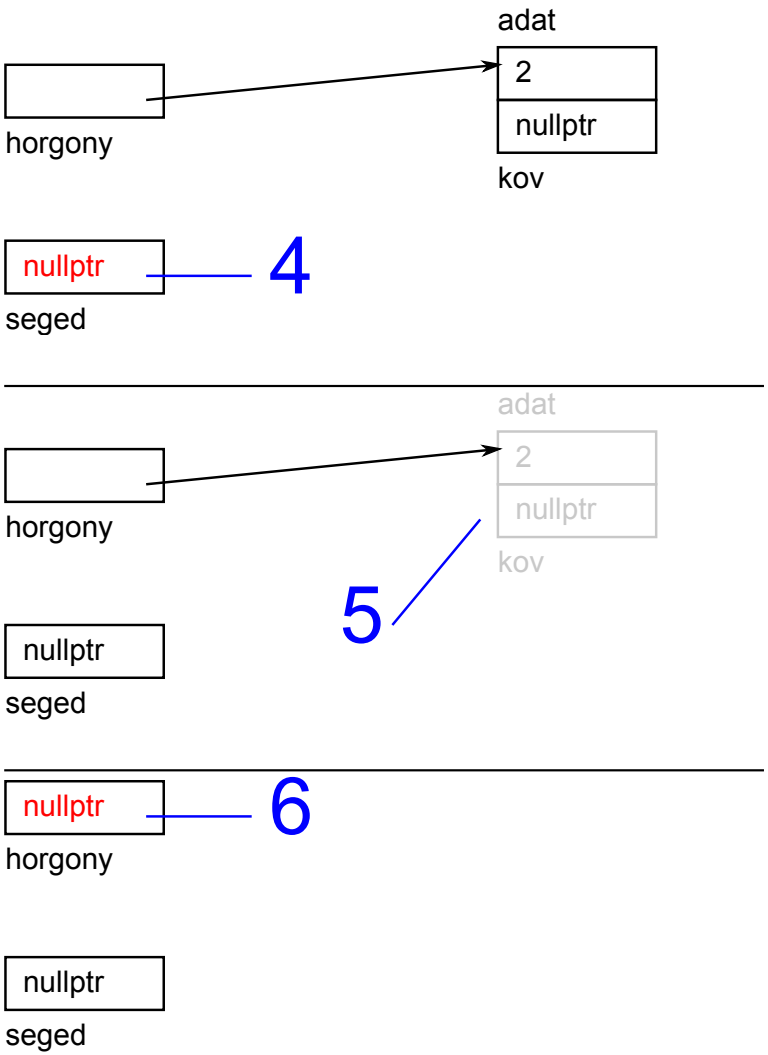
Egyszeresen láncolt lista – törlés



Lista1.cpp

```
// torli a teljes listat                    38
void torolMindet1(Lista1 *horgony) {        39
  while(horgony) {                          40
    Lista1 *seged = horgony->kov;           41
    delete horgony;                         42
    horgony = seged;                        43
  }                                         44
}                                           45
```

Egyszeresen láncolt lista – törlés



Lista1.cpp

```
// torli a teljes listat                    38
void torolMindet1(Lista1 *horgony) {        39
  while(horgony) {                          40
    Lista1 *seged = horgony->kov;           41
    delete horgony;                         42
    horgony = seged;                        43
  }                                         44
}                                           45
```

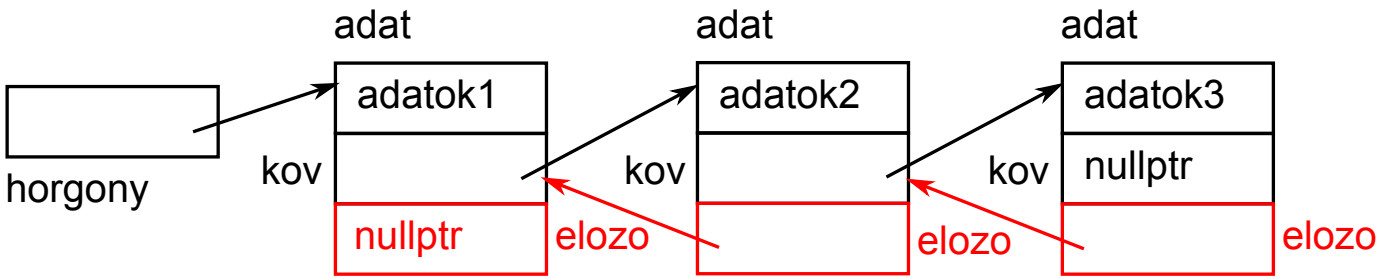
Sor

Készítsük el a verem mintájára a sor megvalósítását is láncolt listával!
Probléma:

A lista utolsó elemének eltávolítása nehézkes

Megoldás:

Kétszeresen láncolt lista



Kétszeresen láncolt lista (Doubly Linked List)

Felhasználható struktúra

```
struct Lista2 {
    ADAT adat;
    Lista2 *elozo, *kov;
};
```

sorTeszt2.cpp

```cpp
#include <iostream>
#include "sor2.h"
using namespace std;

int main() {
  berak(1); berak(2); berak(3); berak(4);
  cout << kivesz() << '\n';
  cout << kivesz() << '\n';
  berak(6);
  cout << kivesz() << '\n';
  cout << kivesz() << '\n';
  cout << kivesz() << '\n';
  // nincs mit kivenni
  cout << kivesz() << '\n';
  return 0;
}
```

sor2.h

```cpp
struct Lista2 {
  int adat;
  Lista2 *elozo, *kov;
};
```

Sor – berak()

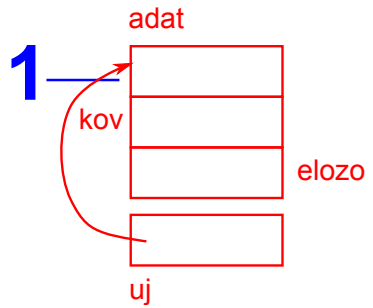
sor2.cpp

```
5  static Lista2* eleje = nullptr;
6  static Lista2* vege
   = nullptr;
7
8  bool berak(int adat) {
9    Lista2* uj = new Lista2;
10   if(uj != nullptr) {
11     uj->adat = adat;
12     uj->elozo = nullptr;
13     uj->kov = eleje;
14     if(eleje != nullptr) {
15       eleje->elozo = uj;
16     }
17     eleje = uj;
18     if(vege == nullptr) {
19       vege = uj;
20     }
21     return true;
22   } else {
23     return false;
24   }
25 }
```

Sor – berak()

sor2.cpp

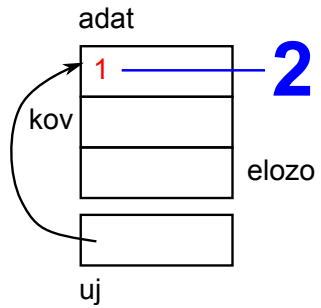
```
5  static Lista2* eleje = nullptr;
6  static Lista2* vege
   = nullptr;
7
8  bool berak(int adat) {
9    Lista2* uj = new Lista2;
10   if(uj != nullptr) {
11     uj->adat = adat;
12     uj->elozo = nullptr;
13     uj->kov = eleje;
14     if(eleje != nullptr) {
15       eleje->elozo = uj;
16     }
17     eleje = uj;
18     if(vege == nullptr) {
19       vege = uj;
20     }
21     return true;
22   } else {
23     return false;
24   }
25 }
```



Sor – berak()

sor2.cpp

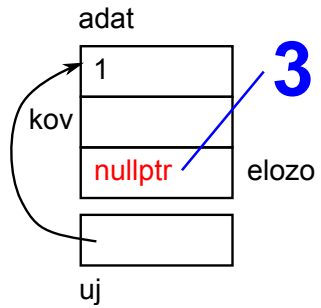
```
5  static Lista2* eleje = nullptr;
6  static Lista2* vege
   = nullptr;
7
8  bool berak(int adat) {
9    Lista2* uj = new Lista2;
10   if(uj != nullptr) {
11     uj->adat = adat;
12     uj->elozo = nullptr;
13     uj->kov = eleje;
14     if(eleje != nullptr) {
15       eleje->elozo = uj;
16     }
17     eleje = uj;
18     if(vege == nullptr) {
19       vege = uj;
20     }
21     return true;
22   } else {
23     return false;
24   }
25 }
```



Sor – berak()

sor2.cpp

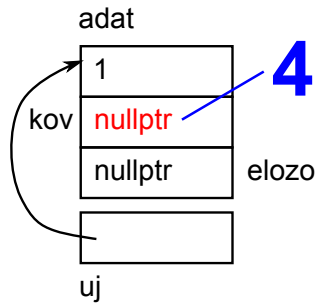
```
5  static Lista2* eleje = nullptr;
6  static Lista2* vege
   = nullptr;
7
8  bool berak(int adat) {
9    Lista2* uj = new Lista2;
10   if(uj != nullptr) {
11     uj->adat = adat;
12     uj->elozo = nullptr;
13     uj->kov = eleje;
14     if(eleje != nullptr) {
15       eleje->elozo = uj;
16     }
17     eleje = uj;
18     if(vege == nullptr) {
19       vege = uj;
20     }
21     return true;
22   } else {
23     return false;
24   }
25 }
```



Sor – berak()

sor2.cpp

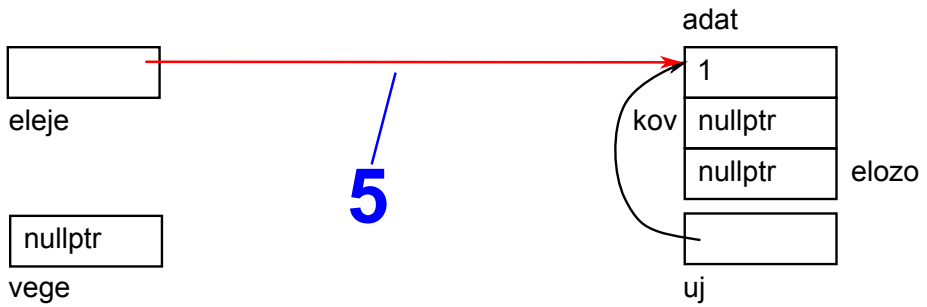
```
5  static Lista2* eleje = nullptr;
6  static Lista2* vege
   = nullptr;
7
8  bool berak(int adat) {
9    Lista2* uj = new Lista2;
10   if(uj != nullptr) {
11     uj->adat = adat;
12     uj->elozo = nullptr;
13     uj->kov = eleje;
14     if(eleje != nullptr) {
15       eleje->elozo = uj;
16     }
17     eleje = uj;
18     if(vege == nullptr) {
19       vege = uj;
20     }
21     return true;
22   } else {
23     return false;
24   }
25 }
```



Sor – berak()

sor2.cpp

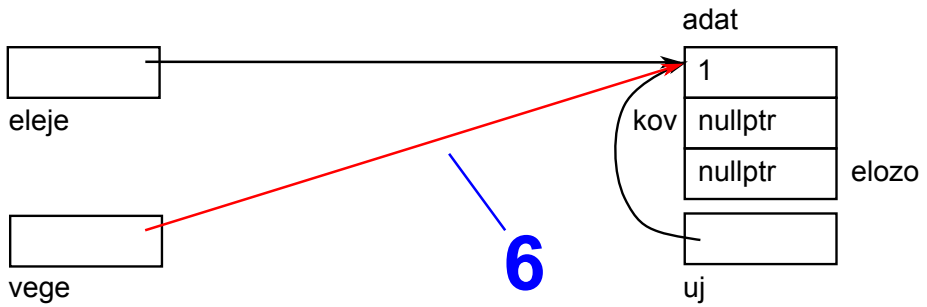
```
5  static Lista2* eleje = nullptr;
6  static Lista2* vege
   = nullptr;
7
8  bool berak(int adat) {
9    Lista2* uj = new Lista2;
10   if(uj != nullptr) {
11     uj->adat = adat;
12     uj->elozo = nullptr;
13     uj->kov = eleje;
14     if(eleje != nullptr) {
15       eleje->elozo = uj;
16     }
17     eleje = uj;
18     if(vege == nullptr) {
19       vege = uj;
20     }
21     return true;
22   } else {
23     return false;
24   }
25 }
```



Sor – berak()

sor2.cpp

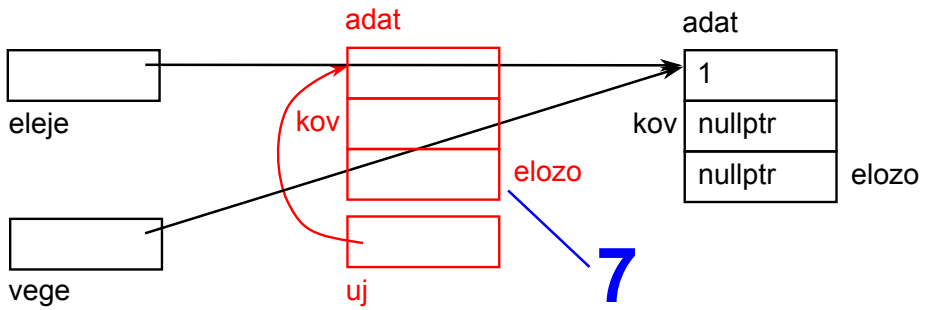
```
5  static Lista2* eleje = nullptr;
6  static Lista2* vege
   = nullptr;
7
8  bool berak(int adat) {
9    Lista2* uj = new Lista2;
10   if(uj != nullptr) {
11     uj->adat = adat;
12     uj->elozo = nullptr;
13     uj->kov = eleje;
14     if(eleje != nullptr) {
15       eleje->elozo = uj;
16     }
17     eleje = uj;
18     if(vege == nullptr) {
19       vege = uj;
20     }
21     return true;
22   } else {
23     return false;
24   }
25 }
```



Sor – berak()

sor2.cpp

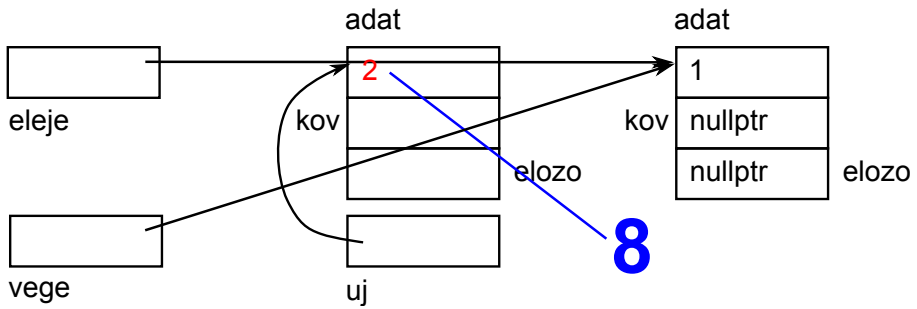
```
5  static Lista2* eleje = nullptr;
6  static Lista2* vege
   = nullptr;
7
8  bool berak(int adat) {
9    Lista2* uj = new Lista2;
10   if(uj != nullptr) {
11     uj->adat = adat;
12     uj->elozo = nullptr;
13     uj->kov = eleje;
14     if(eleje != nullptr) {
15       eleje->elozo = uj;
16     }
17     eleje = uj;
18     if(vege == nullptr) {
19       vege = uj;
20     }
21     return true;
22   } else {
23     return false;
24   }
25 }
```



Sor – berak()

sor2.cpp

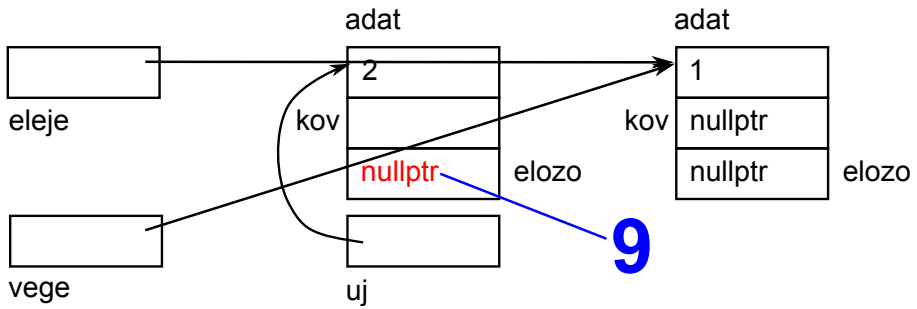
```
5   static Lista2* eleje = nullptr;
6   static Lista2* vege
    = nullptr;
7
8   bool berak(int adat) {
9     Lista2* uj = new Lista2;
10    if(uj != nullptr) {
11      uj->adat = adat;
12      uj->elozo = nullptr;
13      uj->kov = eleje;
14      if(eleje != nullptr) {
15        eleje->elozo = uj;
16      }
17      eleje = uj;
18      if(vege == nullptr) {
19        vege = uj;
20      }
21      return true;
22    } else {
23      return false;
24    }
25  }
```



Sor – berak()

sor2.cpp

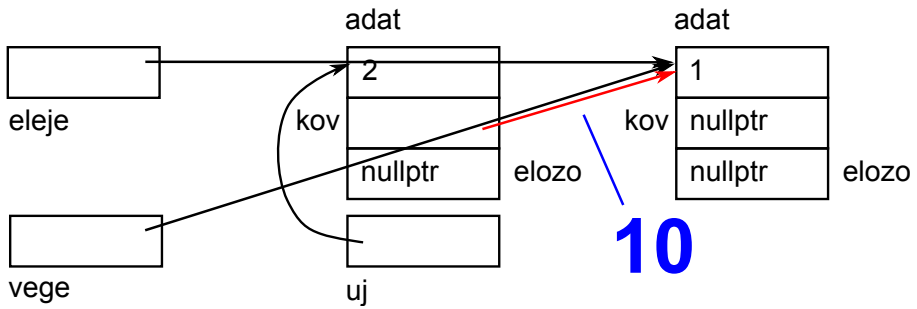
```
5   static Lista2* eleje = nullptr;
6   static Lista2* vege
    = nullptr;
7
8   bool berak(int adat) {
9     Lista2* uj = new Lista2;
10    if(uj != nullptr) {
11      uj->adat = adat;
12      uj->elozo = nullptr;
13      uj->kov = eleje;
14      if(eleje != nullptr) {
15        eleje->elozo = uj;
16      }
17      eleje = uj;
18      if(vege == nullptr) {
19        vege = uj;
20      }
21      return true;
22    } else {
23      return false;
24    }
25  }
```



Sor – berak()

sor2.cpp

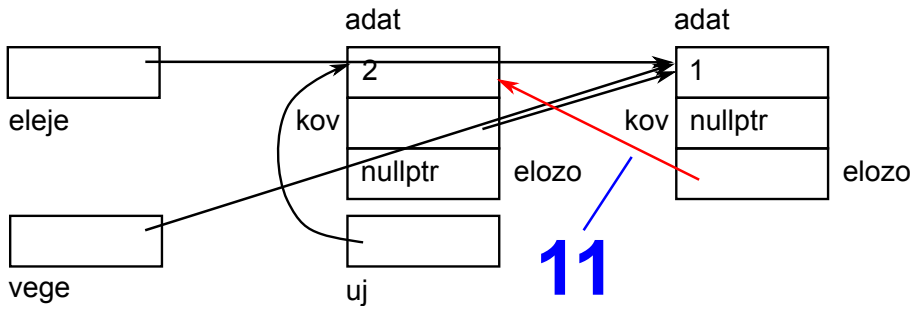
```
5  static Lista2* eleje = nullptr;
6  static Lista2* vege
   = nullptr;
7
8  bool berak(int adat) {
9    Lista2* uj = new Lista2;
10   if(uj != nullptr) {
11     uj->adat = adat;
12     uj->elozo = nullptr;
13     uj->kov = eleje;
14     if(eleje != nullptr) {
15       eleje->elozo = uj;
16     }
17     eleje = uj;
18     if(vege == nullptr) {
19       vege = uj;
20     }
21     return true;
22   } else {
23     return false;
24   }
25 }
```



Sor – berak()

sor2.cpp

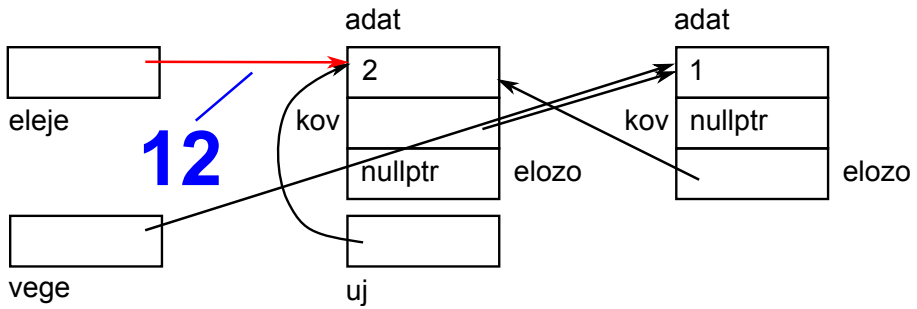
```
5  static Lista2* eleje = nullptr;
6  static Lista2* vege
   = nullptr;
7
8  bool berak(int adat) {
9    Lista2* uj = new Lista2;
10   if(uj != nullptr) {
11     uj->adat = adat;
12     uj->elozo = nullptr;
13     uj->kov = eleje;
14     if(eleje != nullptr) {
15       eleje->elozo = uj;
16     }
17     eleje = uj;
18     if(vege == nullptr) {
19       vege = uj;
20     }
21     return true;
22   } else {
23     return false;
24   }
25 }
```



Sor – berak()

sor2.cpp

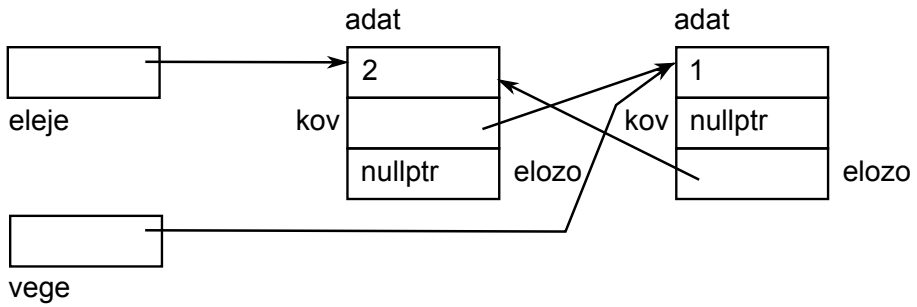
```
5  static Lista2* eleje = nullptr;
6  static Lista2* vege
   = nullptr;
7
8  bool berak(int adat) {
9    Lista2* uj = new Lista2;
10   if(uj != nullptr) {
11     uj->adat = adat;
12     uj->elozo = nullptr;
13     uj->kov = eleje;
14     if(eleje != nullptr) {
15       eleje->elozo = uj;
16     }
17     eleje = uj;
18     if(vege == nullptr) {
19       vege = uj;
20     }
21     return true;
22   } else {
23     return false;
24   }
25 }
```



Sor – berak()

sor2.cpp

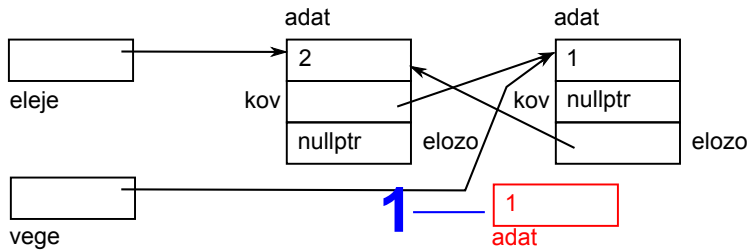
```
5  static Lista2* eleje = nullptr;
6  static Lista2* vege
   = nullptr;
7
8  bool berak(int adat) {
9    Lista2* uj = new Lista2;
10   if(uj != nullptr) {
11     uj->adat = adat;
12     uj->elozo = nullptr;
13     uj->kov = eleje;
14     if(eleje != nullptr) {
15       eleje->elozo = uj;
16     }
17     eleje = uj;
18     if(vege == nullptr) {
19       vege = uj;
20     }
21     return true;
22   } else {
23     return false;
24   }
25 }
```



Sor – kivesz()

sor2.cpp

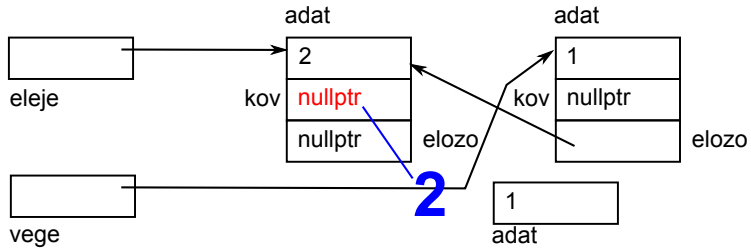
```
27  int kivesz() {
28    if(vege == nullptr) {
29      std::cerr << "A sor ures.\n";
30      return 0;
31    } else {
32      int adat = vege->adat;
33      if(vege->elozo != nullptr) {
34        vege->elozo->kov = nullptr;
35      }
36      Lista2* ujVege = vege->elozo;
37      delete vege;
38      vege = ujVege;
39      if(vege == nullptr) eleje = nullptr;
40      return adat;
41    }
42  }
```



Sor – kivesz()

sor2.cpp

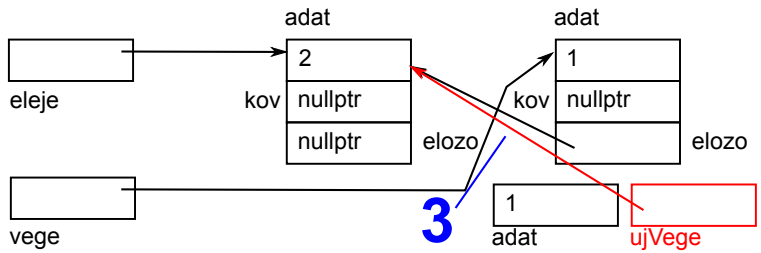
```
27  int kivesz() {
28    if(vege == nullptr) {
29      std::cerr << "A sor ures.\n";
30      return 0;
31    } else {
32      int adat = vege->adat;
33      if(vege->elozo != nullptr) {
34        vege->elozo->kov = nullptr;
35      }
36      Lista2* ujVege = vege->elozo;
37      delete vege;
38      vege = ujVege;
39      if(vege == nullptr) eleje = nullptr;
40      return adat;
41    }
42  }
```



Sor – kivesz()

sor2.cpp

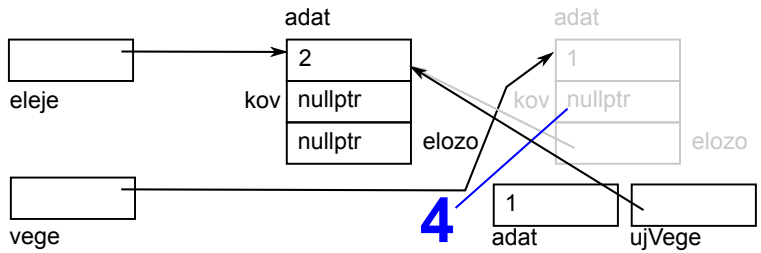
```
27  int kivesz() {
28    if(vege == nullptr) {
29      std::cerr << "A sor ures.\n";
30      return 0;
31    } else {
32      int adat = vege->adat;
33      if(vege->elozo != nullptr) {
34        vege->elozo->kov = nullptr;
35      }
36      Lista2* ujVege = vege->elozo;
37      delete vege;
38      vege = ujVege;
39      if(vege == nullptr) eleje = nullptr;
40      return adat;
41    }
42  }
```



Sor – kivesz()

sor2.cpp

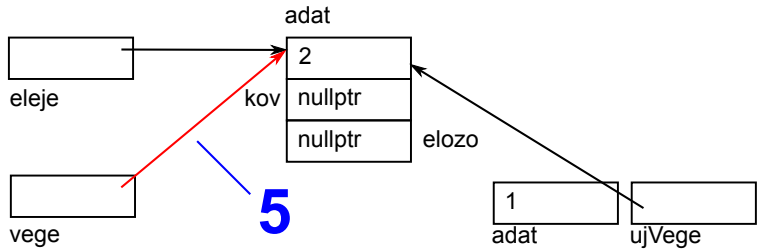
```
27  int kivesz() {
28    if(vege == nullptr) {
29      std::cerr << "A sor ures.\n";
30      return 0;
31    } else {
32      int adat = vege->adat;
33      if(vege->elozo != nullptr) {
34        vege->elozo->kov = nullptr;
35      }
36      Lista2* ujVege = vege->elozo;
37      delete vege;
38      vege = ujVege;
39      if(vege == nullptr) eleje = nullptr;
40      return adat;
41    }
42  }
```



Sor – kivesz()

sor2.cpp

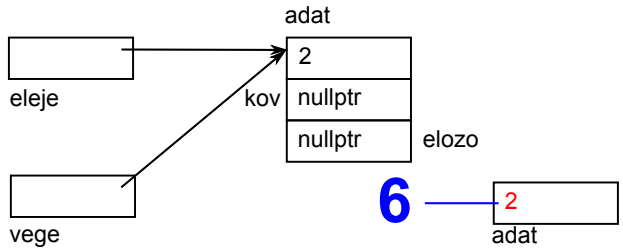
```
27  int kivesz() {
28    if(vege == nullptr) {
29      std::cerr << "A sor ures.\n";
30      return 0;
31    } else {
32      int adat = vege->adat;
33      if(vege->elozo != nullptr) {
34        vege->elozo->kov = nullptr;
35      }
36      Lista2* ujVege = vege->elozo;
37      delete vege;
38      vege = ujVege;
39      if(vege == nullptr) eleje = nullptr;
40      return adat;
41    }
42  }
```



Sor – kivesz()

sor2.cpp

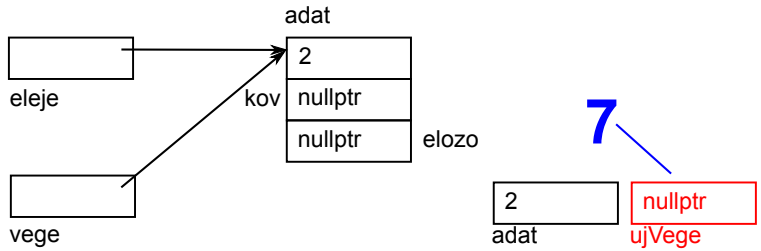
```
27  int kivesz() {
28    if(vege == nullptr) {
29      std::cerr << "A sor ures.\n";
30      return 0;
31    } else {
32      int adat = vege->adat;
33      if(vege->elozo != nullptr) {
34        vege->elozo->kov = nullptr;
35      }
36      Lista2* ujVege = vege->elozo;
37      delete vege;
38      vege = ujVege;
39      if(vege == nullptr) eleje = nullptr;
40      return adat;
41    }
42  }
```



Sor – kivesz()

sor2.cpp

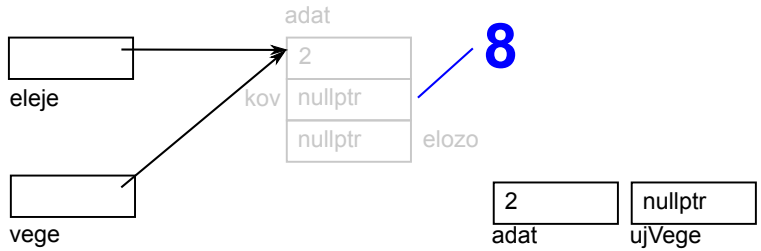
```
27  int kivesz() {
28    if(vege == nullptr) {
29      std::cerr << "A sor ures.\n";
30      return 0;
31    } else {
32      int adat = vege->adat;
33      if(vege->elozo != nullptr) {
34        vege->elozo->kov = nullptr;
35      }
36      Lista2* ujVege = vege->elozo;
37      delete vege;
38      vege = ujVege;
39      if(vege == nullptr) eleje = nullptr;
40      return adat;
41    }
42  }
```



Sor – kivesz()

sor2.cpp

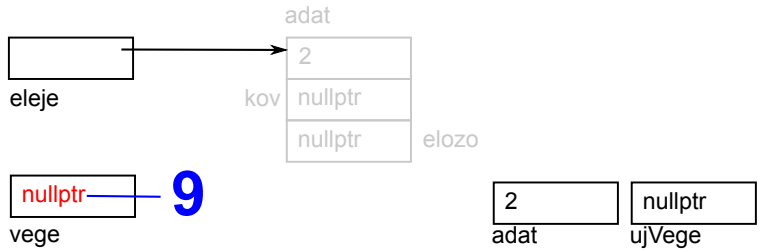
```
27  int kivesz() {
28    if(vege == nullptr) {
29      std::cerr << "A sor ures.\n";
30      return 0;
31    } else {
32      int adat = vege->adat;
33      if(vege->elozo != nullptr) {
34        vege->elozo->kov = nullptr;
35      }
36      Lista2* ujVege = vege->elozo;
37      delete vege;
38      vege = ujVege;
39      if(vege == nullptr) eleje = nullptr;
40      return adat;
41    }
42  }
```



Sor – kivesz()

sor2.cpp

```
27  int kivesz() {
28    if(vege == nullptr) {
29      std::cerr << "A sor ures.\n";
30      return 0;
31    } else {
32      int adat = vege->adat;
33      if(vege->elozo != nullptr) {
34        vege->elozo->kov = nullptr;
35      }
36      Lista2* ujVege = vege->elozo;
37      delete vege;
38      vege = ujVege;
39      if(vege == nullptr) eleje = nullptr;
40      return adat;
41    }
42  }
```



Sor – kivesz()

sor2.cpp

```
27  int kivesz() {
28    if(vege == nullptr) {
29      std::cerr << "A sor ures.\n";
30      return 0;
31    } else {
32      int adat = vege->adat;
33      if(vege->elozo != nullptr) {
34        vege->elozo->kov = nullptr;
35      }
36      Lista2* ujVege = vege->elozo;
37      delete vege;
38      vege = ujVege;
39      if(vege == nullptr) eleje = nullptr;
40      return adat;
41    }
42  }
```

Kétszeresen láncolt lista

Készítsünk ismét általános célú függvényeket!

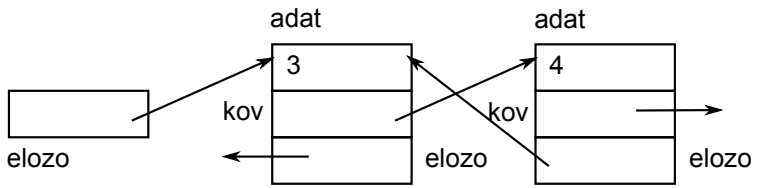
listaTeszt2.cpp (Lista2.cpp, Lista2.h)

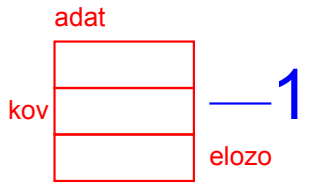
```
6  int main() {
7    Lista2 *horgony=nullptr, *seged=nullptr, *kozepe;
8    for(int i=0; i<7; i++) {
9      seged = beszur2(i, seged);
10     if(horgony == nullptr) {
11       horgony = seged;
12     }
13     if(i == 3) {
14       kozepe = seged;
15     }
16   }
17   kiir2(horgony);
18   kozepe = beszur2(666, kozepe);
```

Kétszeresen láncolt lista – beszúrás

Lista2.cpp

```
5  // 'elozo' utan beszur egy uj elemet
6  Lista2 *beszur2(int adat,
7                  Lista2 *elozo) {
8    Lista2 *uj = new Lista2;
9    if(uj) {
10     uj->adat = adat;
11     if(elozo) {
12       uj->elozo = elozo;
13       uj->kov = elozo->kov;
14       elozo->kov = uj;
15       if(uj->kov) uj->kov->elozo = uj;
16     } else {
17       uj->elozo = uj->kov = nullptr;
18     }
19   }
20   return uj;
21 }
```

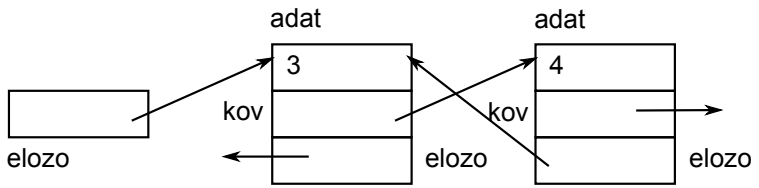


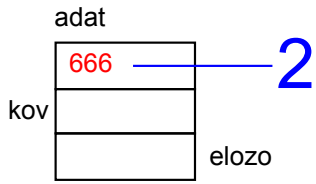


Kétszeresen láncolt lista – beszúrás

Lista2.cpp

```
5  // 'elozo' utan beszur egy uj elemet
6  Lista2 *beszur2(int adat,
7                  Lista2 *elozo) {
8    Lista2 *uj = new Lista2;
9    if(uj) {
10     uj->adat = adat;
11     if(elozo) {
12       uj->elozo = elozo;
13       uj->kov = elozo->kov;
14       elozo->kov = uj;
15       if(uj->kov) uj->kov->elozo = uj;
16     } else {
17       uj->elozo = uj->kov = nullptr;
18     }
19   }
20   return uj;
21 }
```

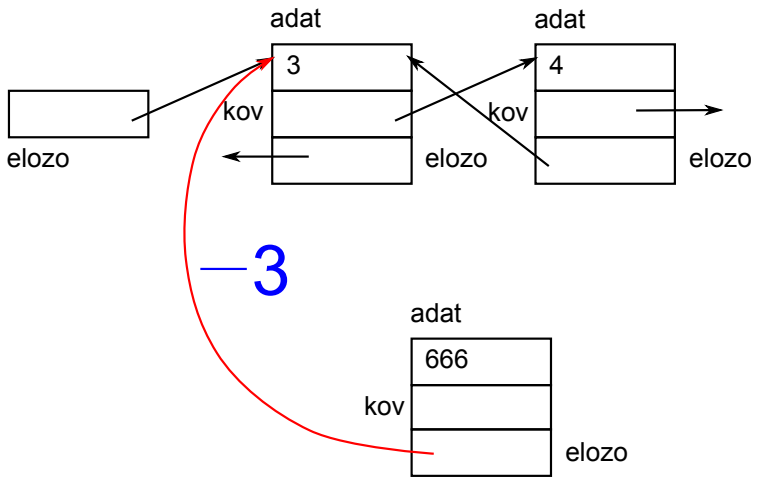




Kétszeresen láncolt lista – beszúrás

Lista2.cpp

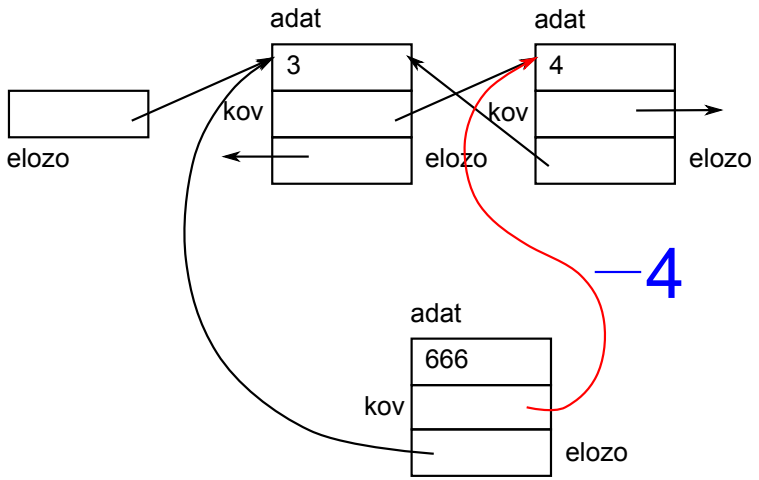
```
5  // 'elozo' utan beszur egy uj elemet
6  Lista2 *beszur2(int adat,
7                  Lista2 *elozo) {
8    Lista2 *uj = new Lista2;
9    if(uj) {
10     uj->adat = adat;
11     if(elozo) {
12       uj->elozo = elozo;
13       uj->kov = elozo->kov;
14       elozo->kov = uj;
15       if(uj->kov) uj->kov->elozo = uj;
16     } else {
17       uj->elozo = uj->kov = nullptr;
18     }
19   }
20   return uj;
21 }
```



Kétszeresen láncolt lista – beszúrás

Lista2.cpp

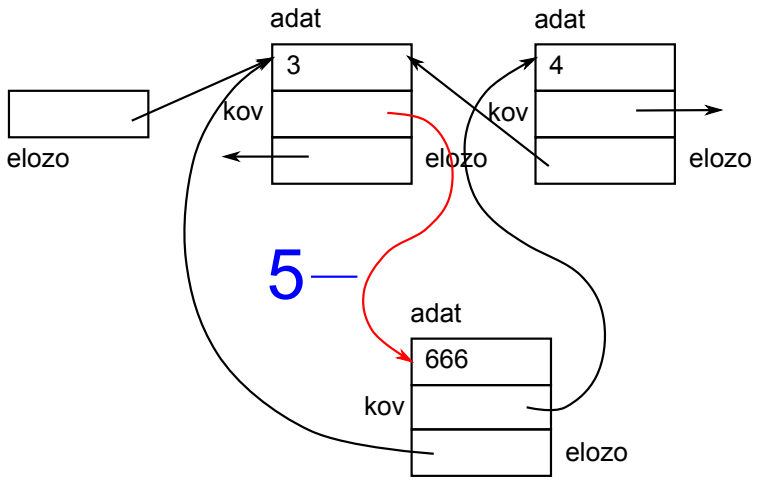
```
5  // 'elozo' utan beszur egy uj elemet
6  Lista2 *beszur2(int adat,
7                  Lista2 *elozo) {
8    Lista2 *uj = new Lista2;
9    if(uj) {
10     uj->adat = adat;
11     if(elozo) {
12       uj->elozo = elozo;
13       uj->kov = elozo->kov;
14       elozo->kov = uj;
15       if(uj->kov) uj->kov->elozo = uj;
16     } else {
17       uj->elozo = uj->kov = nullptr;
18     }
19   }
20   return uj;
21 }
```



Kétszeresen láncolt lista – beszúrás

Lista2.cpp

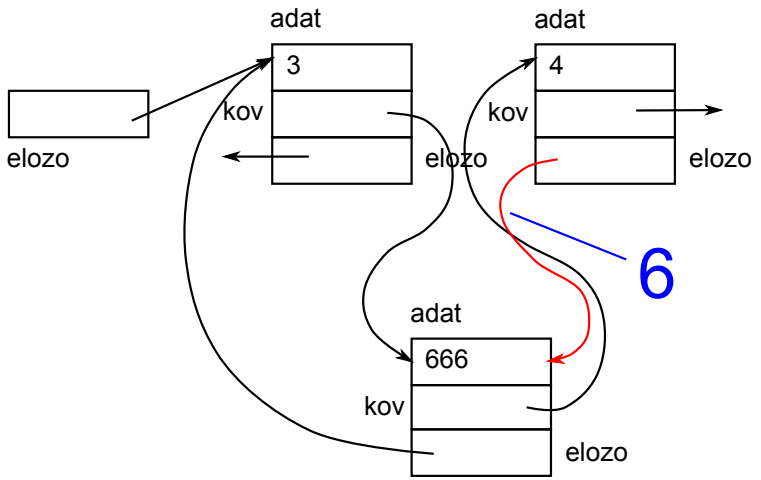
```
5  // 'elozo' utan beszur egy uj elemet
6  Lista2 *beszur2(int adat,
7                  Lista2 *elozo) {
8    Lista2 *uj = new Lista2;
9    if(uj) {
10     uj->adat = adat;
11     if(elozo) {
12       uj->elozo = elozo;
13       uj->kov = elozo->kov;
14       elozo->kov = uj;
15       if(uj->kov) uj->kov->elozo = uj;
16     } else {
17       uj->elozo = uj->kov = nullptr;
18     }
19   }
20   return uj;
21 }
```



Kétszeresen láncolt lista – beszúrás

Lista2.cpp

```
5  // 'elozo' utan beszur egy uj elemet
6  Lista2 *beszur2(int adat,
7                  Lista2 *elozo) {
8    Lista2 *uj = new Lista2;
9    if(uj) {
10     uj->adat = adat;
11     if(elozo) {
12       uj->elozo = elozo;
13       uj->kov = elozo->kov;
14       elozo->kov = uj;
15       if(uj->kov) uj->kov->elozo = uj;
16     } else {
17       uj->elozo = uj->kov = nullptr;
18     }
19   }
20   return uj;
21 }
```



Kétszeresen láncolt lista

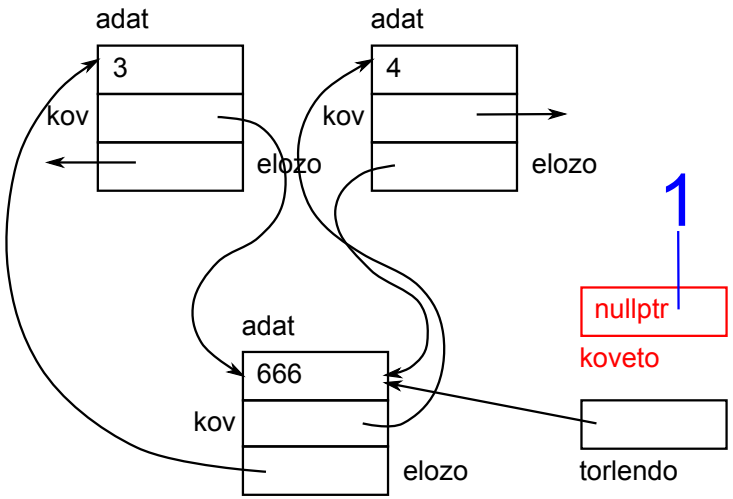
listaTeszt2.cpp (Lista2.cpp, Lista2.h)

```
19      kiir2(horgony);
20      torol2(kozepe);
21      horgony = torol2(horgony);
22      kiir2(horgony);
23      torolMindet2(horgony);
24      return 0;
25  }
```

Kimenet

```
0       1       2       3       4       5       6
0       1       2       3       666     4       5       6
1       2       3       4       5       6
```

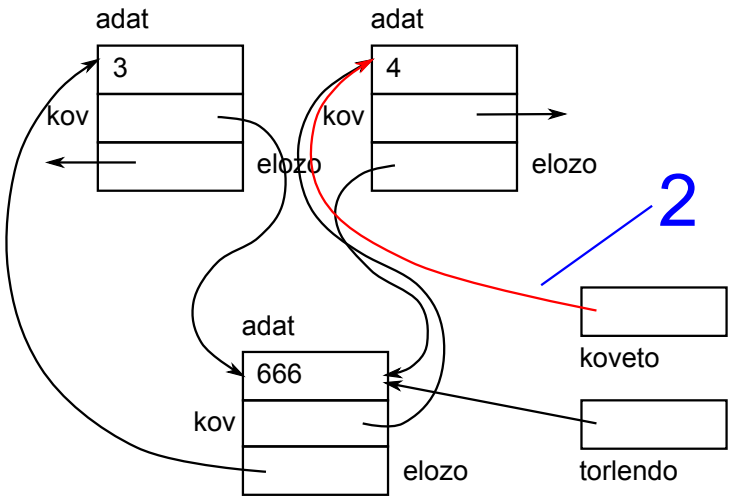
Kétszeresen láncolt lista – törlés



Lista2.cpp

```
31  // torli 'torlendo'-t, vissza: kov. elem
32  Lista2 *torol2(Lista2 *torlendo) {
33    Lista2 *koveto = nullptr;
34    if(torlendo) {
35      koveto = torlendo->kov;
36      if(torlendo->elozo) torlendo->elozo->kov = torlendo->kov;
37      if(torlendo->kov) torlendo->kov->elozo = torlendo->elozo;
38      delete torlendo;
39    }
40    return koveto;
41  }
```

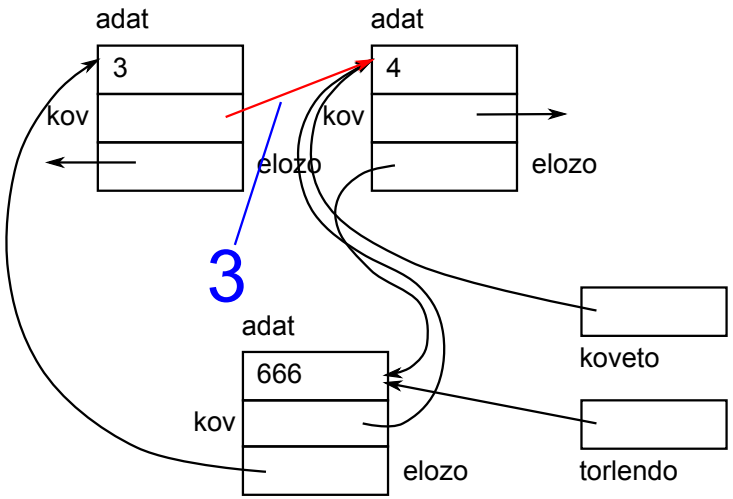
Kétszeresen láncolt lista – törlés



Lista2.cpp

```
31  // torli 'torlendo'-t, vissza: kov. elem
32  Lista2 *torol2(Lista2 *torlendo) {
33    Lista2 *koveto = nullptr;
34    if(torlendo) {
35      koveto = torlendo->kov;
36      if(torlendo->elozo) torlendo->elozo->kov = torlendo->kov;
37      if(torlendo->kov) torlendo->kov->elozo = torlendo->elozo;
38      delete torlendo;
39    }
40    return koveto;
41  }
```

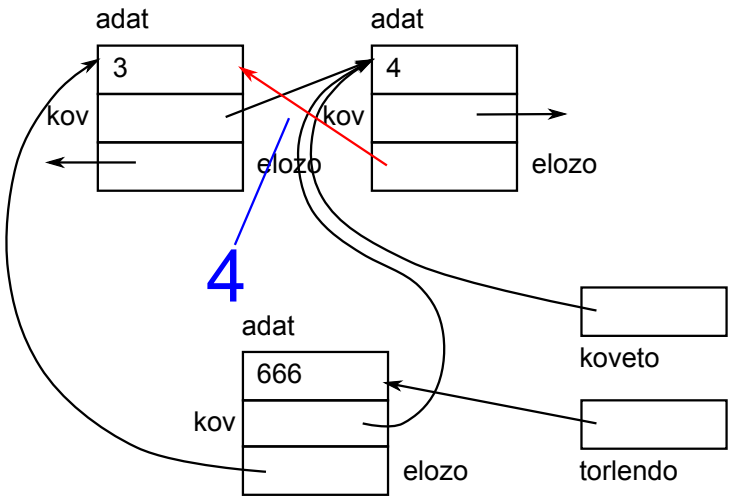
Kétszeresen láncolt lista – törlés



Lista2.cpp

```
31  // torli 'torlendo'-t, vissza: kov. elem
32  Lista2 *torol2(Lista2 *torlendo) {
33    Lista2 *koveto = nullptr;
34    if(torlendo) {
35      koveto = torlendo->kov;
36      if(torlendo->elozo) torlendo->elozo->kov = torlendo->kov;
37      if(torlendo->kov) torlendo->kov->elozo = torlendo->elozo;
38      delete torlendo;
39    }
40    return koveto;
41  }
```

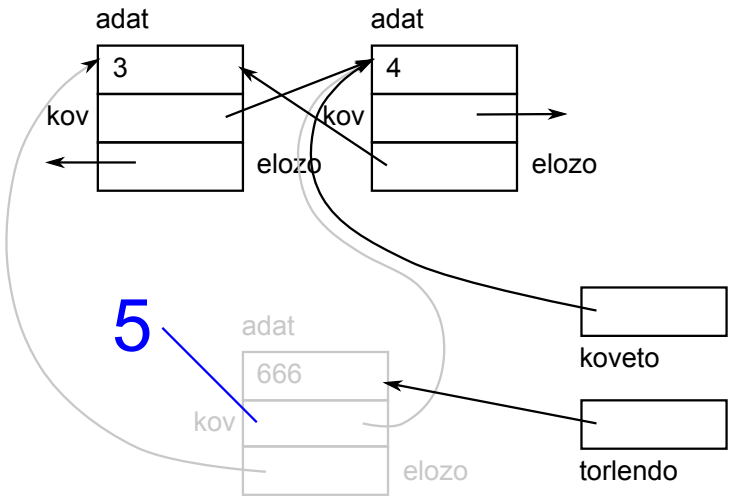
Kétszeresen láncolt lista – törlés



Lista2.cpp

```
31  // torli 'torlendo'-t, vissza: kov. elem
32  Lista2 *torol2(Lista2 *torlendo) {
33    Lista2 *koveto = nullptr;
34    if(torlendo) {
35      koveto = torlendo->kov;
36      if(torlendo->elozo) torlendo->elozo->kov = torlendo->kov;
37      if(torlendo->kov) torlendo->kov->elozo = torlendo->elozo;
38      delete torlendo;
39    }
40    return koveto;
41  }
```

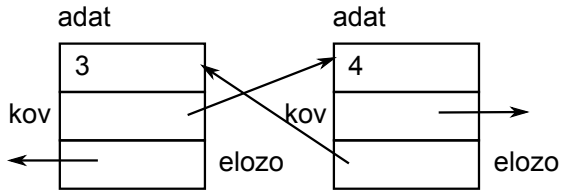
Kétszeresen láncolt lista – törlés



Lista2.cpp

```
31  // torli 'torlendo'-t, vissza: kov. elem
32  Lista2 *torol2(Lista2 *torlendo) {
33    Lista2 *koveto = nullptr;
34    if(torlendo) {
35      koveto = torlendo->kov;
36      if(torlendo->elozo) torlendo->elozo->kov = torlendo->kov;
37      if(torlendo->kov) torlendo->kov->elozo = torlendo->elozo;
38      delete torlendo;
39    }
40    return koveto;
41  }
```

Kétszeresen láncolt lista – törlés



Lista2.cpp

```
31  // torli 'torlendo'-t, vissza: kov. elem
32  Lista2 *torol2(Lista2 *torlendo) {
33    Lista2 *koveto = nullptr;
34    if(torlendo) {
35      koveto = torlendo->kov;
36      if(torlendo->elozo) torlendo->elozo->kov = torlendo->kov;
37      if(torlendo->kov) torlendo->kov->elozo = torlendo->elozo;
38      delete torlendo;
39    }
40    return koveto;
41  }
```

Kétszeresen láncolt lista

Lista2.cpp

```
43  // torli a teljes listat
44  void torolMindet2(Lista2 *horgony) {
45    Lista2 *seged = horgony;
46    while(seged) {
47      seged = torol2(seged);
48    }
49  }

23  // kiirja a lista osszes elemet
24  void kiir2(Lista2 *horgony) {
25    for(Lista2* seged=horgony; seged; seged=seged->kov) {
26      std::cout << seged->adat << '\t';
27    }
28    std::cout << std::endl;
29  }
```