

Programozás

(GKxB_INTM114)

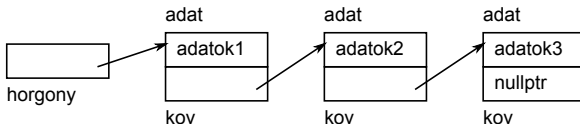
Dr. Hatwagner F. Miklós

Széchenyi István Egyetem, Győr

https://github.com/wajzy/GKxB_INTM114.git

2024. május 7.

Verem megvalósítható tömbbel, de annak 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;
};
```

```
#include <iostream>
#include "verem2.h"
using namespace std;

#define N 5

int main() {
    cout << N << " egész 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;
}
```

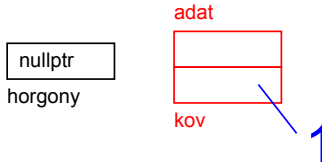
```
struct Lista1 { 4
    int adat;    5
    Lista1* kov; 6
};              7
```

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

horgony

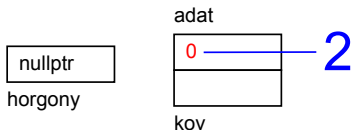
100

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



1000

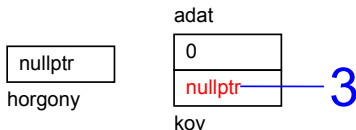
```
6  bool berak(int adat) {
7      Lista1* uj = new Lista1;
8      if(uj != nullptr) {
9          uj->adat = adat;
10         uj->kov = horgony;
11         horgony = uj;
12         return true;
13     } else {
14         return false;
15     }
16 }
```



```

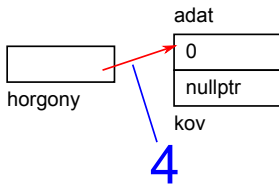
6  bool berak(int adat) {
7      Lista1* uj = new Lista1;
8      if(uj != nullptr) {
9          uj->adat = adat;
10         uj->kov = horgony;
11         horgony = uj;
12         return true;
13     } else {
14         return false;
15     }
16 }

```



100

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

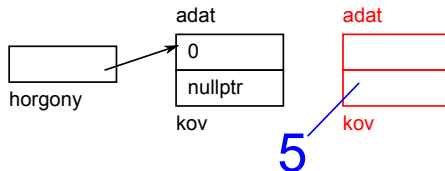


100

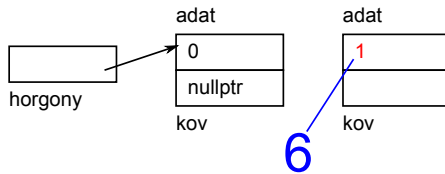
```

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

```



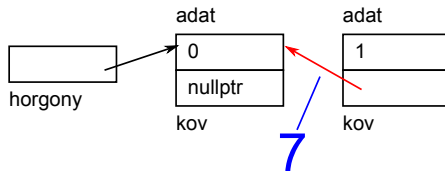
```
6  bool berak(int adat) {
7      Lista1* uj = new Lista1;
8      if(uj != nullptr) {
9          uj->adat = adat;
10         uj->kov = horgony;
11         horgony = uj;
12         return true;
13     } else {
14         return false;
15     }
16 }
```



```

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

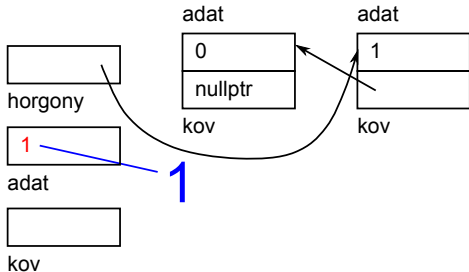
```



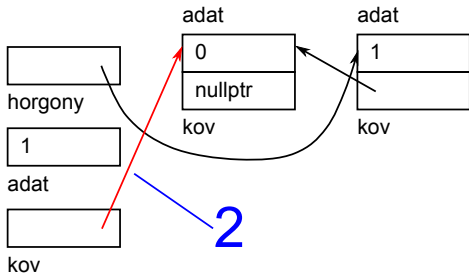
100

verem2.cpp

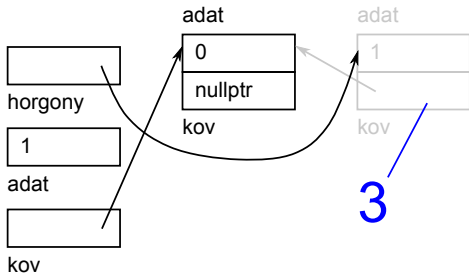
```
19 int kivesz() {
20     if(horgony == nullptr) {
21         std::cerr<<"A verem ures.\n";
22         return 0;
23     } else {
24         int adat = horgony->adat;
25         Lista1* kov = horgony->kov;
26         delete horgony;
27         horgony = kov;
28         return adat;
29     }
30 }
```



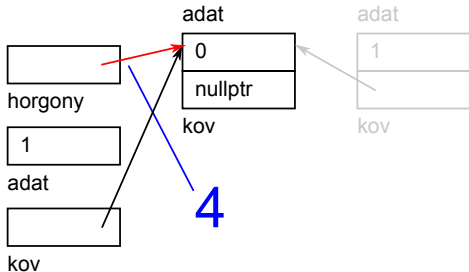
```
int kivesz() {
    if(horgony == nullptr) {
        std::cerr<<"A verem ures.\n";
        return 0;
    } else {
        int adat = horgony->adat;
        Lista1* kov = horgony->kov;
        delete horgony;
        horgony = kov;
        return adat;
    }
}
```



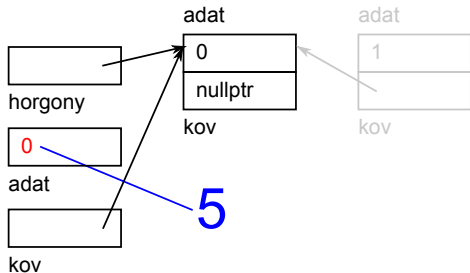
```
int kivesz() {
    if(horgony == nullptr) {
        std::cerr<<"A verem ures.\n";
        return 0;
    } else {
        int adat = horgony->adat;
        Lista1* kov = horgony->kov;
        delete horgony;
        horgony = kov;
        return adat;
    }
}
```



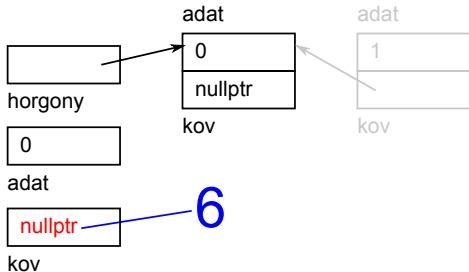
```
int kivesz() {
    if(horgony == nullptr) {
        std::cerr<<"A verem ures.\n";
        return 0;
    } else {
        int adat = horgony->adat;
        Lista1* kov = horgony->kov;
        delete horgony;
        horgony = kov;
        return adat;
    }
}
```



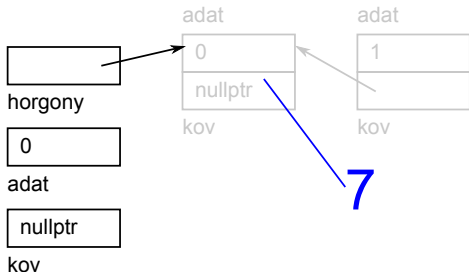

```
int kivesz() {
    if (horgony == nullptr) {
        std::cerr << "A verem ures.\n";
        return 0;
    } else {
        int adat = horgony->adat;
        Lista1* kov = horgony->kov;
        delete horgony;
        horgony = kov;
        return adat;
    }
}
```



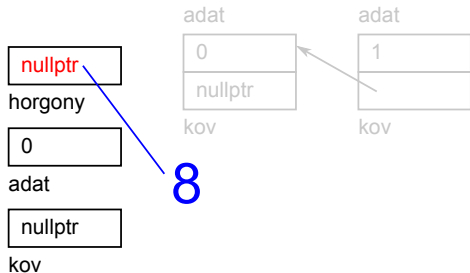
```
int kivesz() {
    if(horgony == nullptr) {
        std::cerr<<"A verem ures.\n";
        return 0;
    } else {
        int adat = horgony->adat;
        Lista1* kov = horgony->kov;
        delete horgony;
        horgony = kov;
        return adat;
    }
}
```



```
int kivesz() {
    if(horgony == nullptr) {
        std::cerr<<"A verem ures.\n";
        return 0;
    } else {
        int adat = horgony->adat;
        Lista1* kov = horgony->kov;
        delete horgony;
        horgony = kov;
        return adat;
    }
}
```



```
int kivesz() {
    if (horgony == nullptr) {
        std::cerr << "A verem ures.\n";
        return 0;
    } else {
        int adat = horgony->adat;
        Lista1* kov = horgony->kov;
        delete horgony;
        horgony = kov;
        return adat;
    }
}
```



```

1  #include <iostream>
2  #include <cstdlib> // nullptr mutato
3  #include "Lista1.h"
4  using namespace std;
5
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 }

```



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 }

```



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 }

```



Lista1.cpp

```

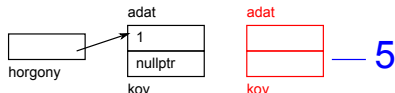
5 // 'elozo' utan beszur egy uj elemet
6 Lista1 *beszur1(int adat, Lista1 *elozo) {
7     Lista1* uj = new Lista1;
8     if (uj) { // if (uj != nullptr) { //...
9         uj->adat = adat;
10        if (elozo) {
11            uj->kov = elozo->kov;
12            elozo->kov = uj;
13        } else {
14 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

```




© 2006 The Authors
Journal compilation © 2006 Blackwell Publishing Ltd

J

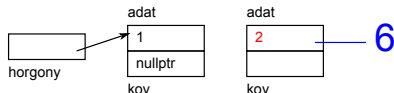


Lista1.cpp

```

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 }

```

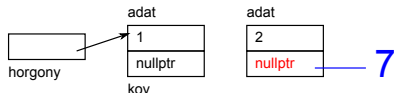


Lista1.cpp

```

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 }

```

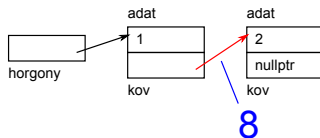


Lista1.cpp

```

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 }

```



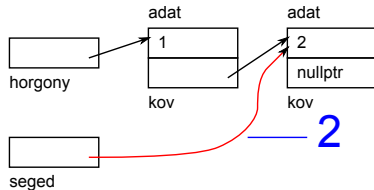
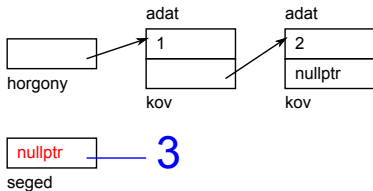
8

© 2006 The Authors
Journal compilation © 2006 Blackwell Publishing Ltd

```

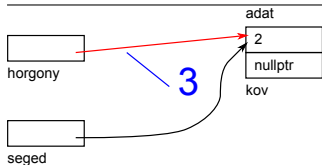
6  int main() {
7      cout << "Adjon meg egeszeket, leallas negativ szamra!\n";
8      Lista1 *horgony=nullptr, *seged=nullptr;
9      int szam;
10     while(cin>>szam, szam>=0) {
11         seged = beszur1(szam, seged);
12         if(horgony == nullptr) horgony = seged;
13     }
14     cout << "Ezeket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

```

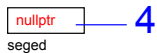


```
// kiirja a lista osszes elemet
void kiir1(Lista1 *horgony) {
    Lista1 *seged;
    for(seged=horgony; seged;
        seged=seged->kov) {
        std::cout << seged->adat
                    << '\t';
    }
}
```

20
21
22
23
24
25
26
27



```
// torli a teljes listát
void torolMindet1(Lista1 *horgony) {
    while(horgony) {
        Lista1 *seged = horgony->kov;
        delete horgony;
        horgony = seged;
    }
}
```

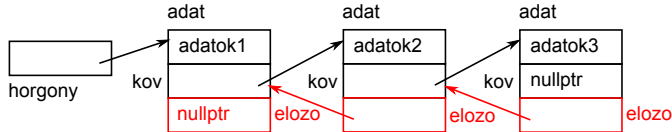


```
// torli a teljes listát
void torolMindet1(Lista1 *horgony) {
    while(horgony) {
        Lista1 *seged = horgony->kov;
        delete horgony;
        horgony = seged;
    }
}
```

45

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;
};
```

```
1 #include <iostream>
2 #include "sor2.h"
3 using namespace std;
4
5 int main() {
6     berak(1); berak(2); berak(3); berak(4);
7     cout << kivesz() << '\n';
8     cout << kivesz() << '\n';
9     berak(6);
10    cout << kivesz() << '\n';
11    cout << kivesz() << '\n';
12    cout << kivesz() << '\n';
13    // nincs mit kivenni
14    cout << kivesz() << '\n';
15    return 0;
16 }
```

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

sor2.cpp

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

nullptr

eleje

nullptr

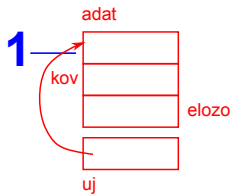
vege

```
static Lista2* eleje = nullptr;
static Lista2* vege = nullptr;

bool berak(int adat) {
    Lista2* uj = new Lista2;
    if (uj != nullptr) {
        uj->adat = adat;
        uj->elozo = nullptr;
        uj->kov = eleje;
        if (eleje != nullptr) {
            eleje->elozo = uj;
        }
        eleje = uj;
        if (vege == nullptr) {
            vege = uj;
        }
        return true;
    } else {
        return false;
    }
}
```

eleje

vege



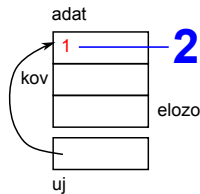
Új elemek sorba helyezése (enqueue)

sor2.cpp

```

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

```

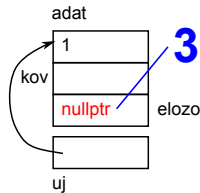


sor2.cpp

```

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

```



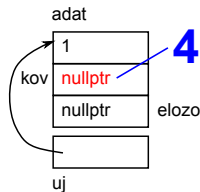
Új elemek sorba helyezése (enqueue)

sor2.cpp

```

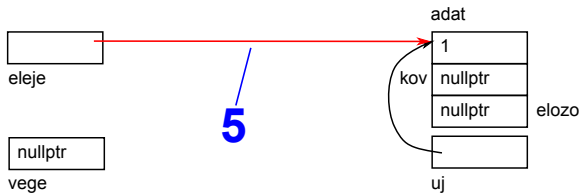
4 static Lista2* eleje = nullptr;
5 static Lista2* vege = nullptr;
6
7 bool berak(int adat) {
8     Lista2* uj = new Lista2;
9     if (uj != nullptr) {
10         uj->adat = adat;
11         uj->elozo = nullptr;
12         uj->kov = eleje;
13         if (eleje != nullptr) {
14             eleje->elozo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

```



```
static Lista2* eleje = nullptr;
static Lista2* vege = nullptr;

bool berak(int adat) {
    Lista2* uj = new Lista2;
    if (uj != nullptr) {
        uj->adat = adat;
        uj->elozo = nullptr;
        uj->kov = eleje;
        if (eleje != nullptr) {
            eleje->elozo = uj;
        }
        eleje = uj;
        if (vege == nullptr) {
            vege = uj;
        }
        return true;
    } else {
        return false;
    }
}
```

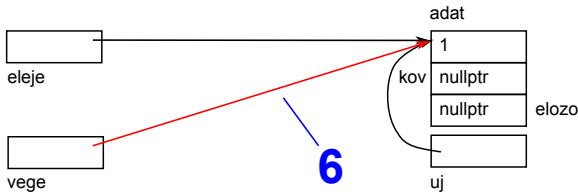


sor2.cpp

```

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

```



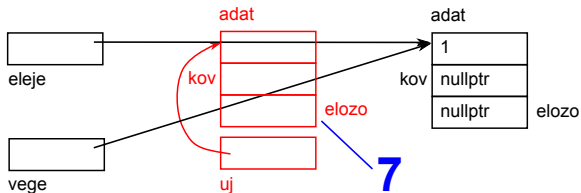
Új elemek sorba helyezése (enqueue)

sor2.cpp

```

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

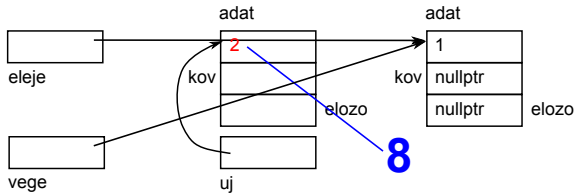
```



Új elemek sorba helyezése (enqueue)

sor2.cpp

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



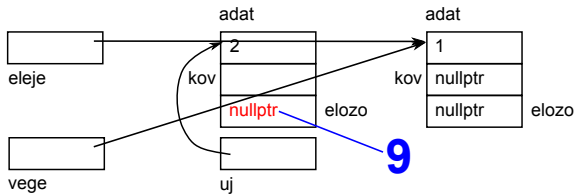
Új elemek sorba helyezése (enqueue)

sor2.cpp

```

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

```

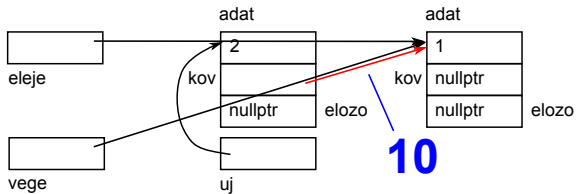


100

```

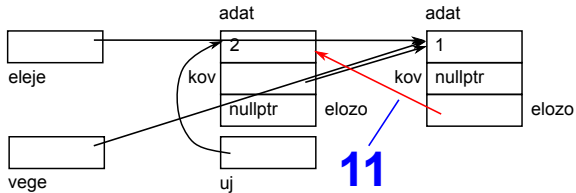
4 static Lista2* eleje = nullptr;
5 static Lista2* vege = nullptr;
6
7 bool berak(int adat) {
8     Lista2* uj = new Lista2;
9     if (uj != nullptr) {
10         uj->adat = adat;
11         uj->elozo = nullptr;
12         uj->kov = eleje;
13         if (eleje != nullptr) {
14             eleje->elozo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

```



```
static Lista2* eleje = nullptr;
static Lista2* vege = nullptr;

bool berak(int adat) {
    Lista2* uj = new Lista2;
    if (uj != nullptr) {
        uj->adat = adat;
        uj->elozo = nullptr;
        uj->kov = eleje;
        if (eleje != nullptr) {
            eleje->elozo = uj;
        }
        eleje = uj;
        if (vege == nullptr) {
            vege = uj;
        }
        return true;
    } else {
        return false;
    }
}
```

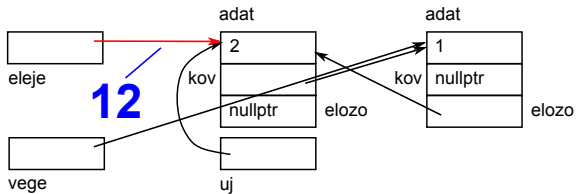


sor2.cpp

```

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

```

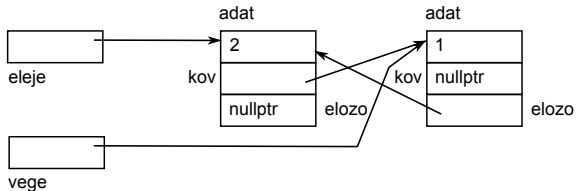


sor2.cpp

```

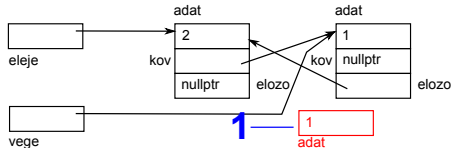
4 static Lista2* eleje = nullptr;
5 static Lista2* vege = nullptr;
6
7 bool berak(int adat) {
8     Lista2* uj = new Lista2;
9     if (uj != nullptr) {
10         uj->adat = adat;
11         uj->elozo = nullptr;
12         uj->kov = eleje;
13         if (eleje != nullptr) {
14             eleje->elozo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

```



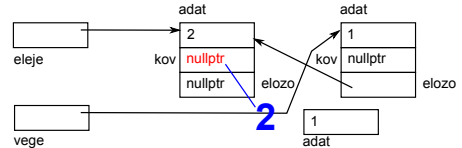
sor2.cpp

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



sor2.cpp

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



The diagram shows a linked list with two nodes. The first node has value 2, and the second node has value 1. The 'vege' pointer (head) is being updated to point to the second node, bypassing the first node. The first node is marked for deletion with a red 'X' and the label 'ujVege'.

```

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

```

The diagram illustrates a memory layout with two memory blocks. The first block, labeled 'adat', contains the value '2' and two 'nullptr' entries. The second block, labeled 'uVege', contains the value '1' and two 'nullptr' entries. A variable 'eleje' points to the first block, and a variable 'vege' points to the second block. A blue arrow labeled '4' points from 'vege' to 'adat', indicating a pointer update or a specific memory address.

```

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

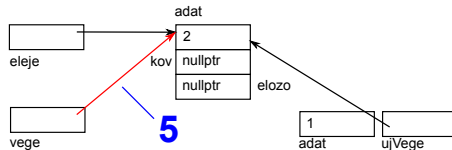
```

sor2.cpp

```

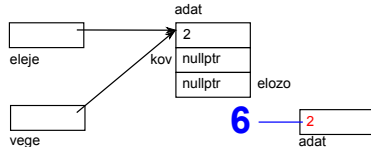
26 int kivesz() {
27     if(vege == nullptr) {
28         std::cerr << "A sor ures.\n";
29         return 0;
30     } else {
31         int adat = vege->adat;
32         if(vege->elozo != nullptr) {
33             vege->elozo->kov = nullptr;
34         }
35         Lista2* ujVege = vege->elozo;
36  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

```



sor2.cpp

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

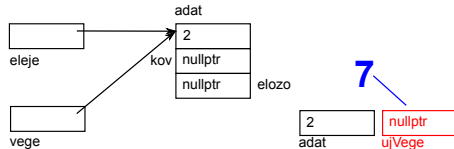


sor2.cpp

```

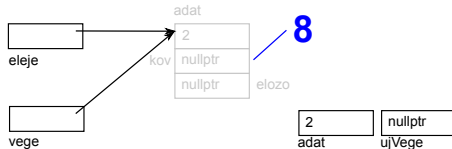
26 int kivesz() {
27     if(vege == nullptr) {
28         std::cerr << "A sor ures.\n";
29         return 0;
30     } else {
31         int adat = vege->adat;
32         if(vege->elozo != nullptr) {
33             vege->elozo->kov = nullptr;
34         }
35         Lista2* ujVege = vege->elozo;
36  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

```



sor2.cpp

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



sor2.cpp

```

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

```



sor2.cpp

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

nullptr — 10
eleje

nullptr
vege

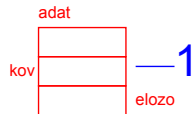
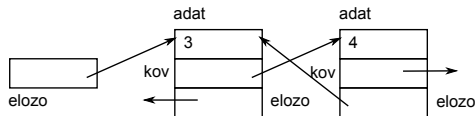
2 nullptr
adat ujVege

Készítsünk ismét általános célú függvényeket!

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

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

```
// 'elozo' utan beszur egy uj elemet
Lista2 *beszur2(int adat,
                 Lista2 *elozo) {
    Lista2 *uj = new Lista2;
    if (uj) {
        uj->adat = adat;
        if (elozo) {
            uj->elozo = elozo;
            uj->kov = elozo->kov;
            elozo->kov = uj;
            if (uj->kov) uj->kov->elozo = uj;
        } else {
            uj->elozo = uj->kov = nullptr;
        }
    }
    return uj;
}
```

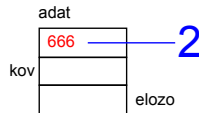
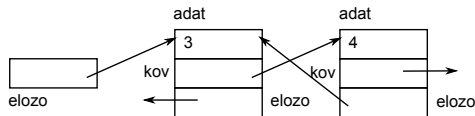


List a2.cpp

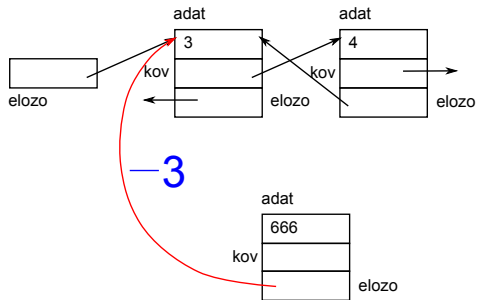
```

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

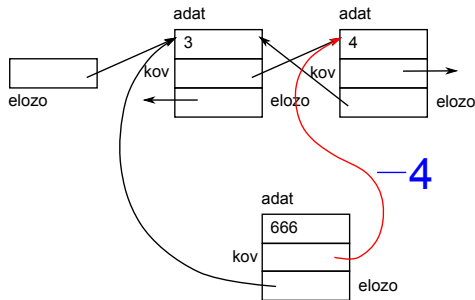
```



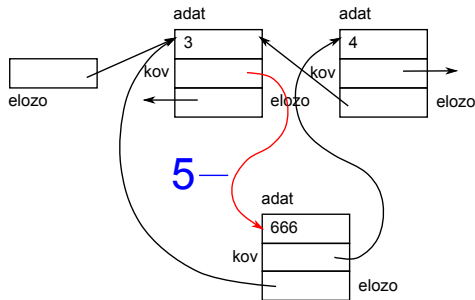
```
// 'elozo' utan beszur egy uj elemet
Lista2 *beszur2(int adat,
                Lista2 *elozo) {
    Lista2 *uj = new Lista2;
    if(uj) {
        uj->adat = adat;
        if(elozo) {
            uj->elozo = elozo;
            uj->kov = elozo->kov;
            elozo->kov = uj;
            if(uj->kov) uj->kov->elozo = uj;
        } else {
            uj->elozo = uj->kov = nullptr;
        }
    }
    return uj;
}
```



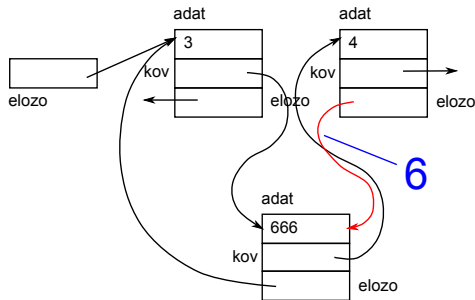
```
// 'elozo' utan beszur egy uj elemet
Lista2 *beszur2(int adat,
                Lista2 *elozo) {
    Lista2 *uj = new Lista2;
    if (uj) {
        uj->adat = adat;
        if (elozo) {
            uj->elozo = elozo;
            uj->kov = elozo->kov;
            elozo->kov = uj;
            if (uj->kov) uj->kov->elozo = uj;
        } else {
            uj->elozo = uj->kov = nullptr;
        }
    }
    return uj;
}
```



```
// 'elozo' utan beszur egy uj elemet
Lista2 *beszur2(int adat,
                Lista2 *elozo) {
    Lista2 *uj = new Lista2;
    if(uj) {
        uj->adat = adat;
        if(elozo) {
            uj->elozo = elozo;
            uj->kov = elozo->kov;
            elozo->kov = uj;
            if(uj->kov) uj->kov->elozo = uj;
        } else {
            uj->elozo = uj->kov = nullptr;
        }
    }
    return uj;
}
```




```
// 'elozo' utan beszur egy uj elemet
Lista2 *beszur2(int adat,
                Lista2 *elozo) {
    Lista2 *uj = new Lista2;
    if(uj) {
        uj->adat = adat;
        if(elozo) {
            uj->elozo = elozo;
            uj->kov = elozo->kov;
            elozo->kov = uj;
            if(uj->kov) uj->kov->elozo = uj;
        } else {
            uj->elozo = uj->kov = nullptr;
        }
    }
    return uj;
}
```



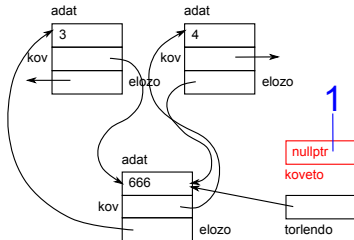
listaTeszt2.cpp (Lista2.cpp, Lista2.h)

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

Kimenet

| | | | | | | | |
|---|---|---|---|-----|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | |
| 0 | 1 | 2 | 3 | 666 | 4 | 5 | 6 |
| 1 | 2 | 3 | 4 | 5 | 6 | | |

Elem törlése kétszeresen láncolt listából



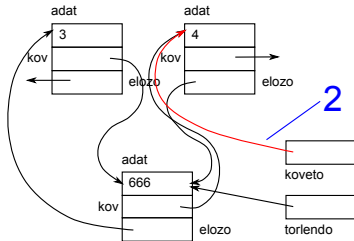
Lista2.cpp

```

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

```

Elem törlése kétszeresen láncolt listából



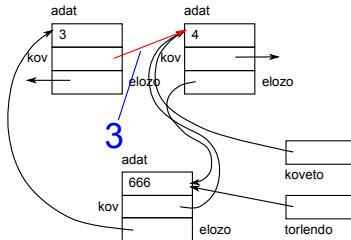
Lista2.cpp

```

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

```

Elem törlése kétszeresen láncolt listából



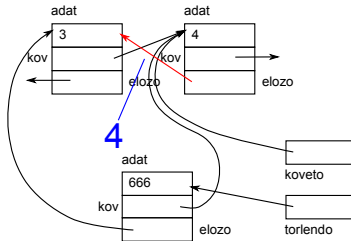
Lista2.cpp

```

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

```

Elem törlése kétszeresen láncolt listából



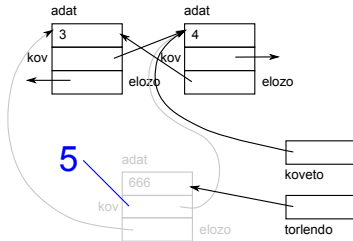
Lista2.cpp

```

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

```

Elem törlése kétszeresen láncolt listából

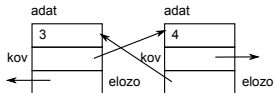


Lista2.cpp

```

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

```



100

```

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

```


Lista2.cpp

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

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