Programozás
(GKxB_INTM021)

Dr. Hatwágner F. Miklós

Széchenyi István Egyetem, Győr

2018. július 30.

Egész szám átalakítása karakterlánccá

Feladat

Készítsünk függvényt, ami egész számból `stringet` hoz létre!

Ötlet

1. Osszuk a számot maradékosan 10-zel, a maradékból előáll az aktuálisan legkisebb helyiértéken lévő számjegy karakter
2. Egészosztással osszuk a számot 10-zel
3. Ugorjunk 1-re, ha $szám > 0$

Problémák

- Negatív számok $\rightarrow$ a szám -1-szeresével végezzük a műveletet, majd a `stringbe` betesszük az előjel karaktert
- A számjegyek fordított sorrendben keletkeznek $\rightarrow$ `string` fordító fv.-t kell írni

Egész szám átalakítása karakterlánccá

itos1.cpp

```
16  string itos(int szam) {
17    bool negativ = false;
18    if(szam < 0) {
19      szam *= -1;
20      negativ = true;
21    }
22    string s = "";
23    do {
24      s += char(szam%10 + '0');
25      szam /= 10;
26    } while(szam != 0);
27    if(negativ) s += '-';
28    szofordit(&s);
29    return s;
30  }
```

Egész szám átalakítása karakterlánccá

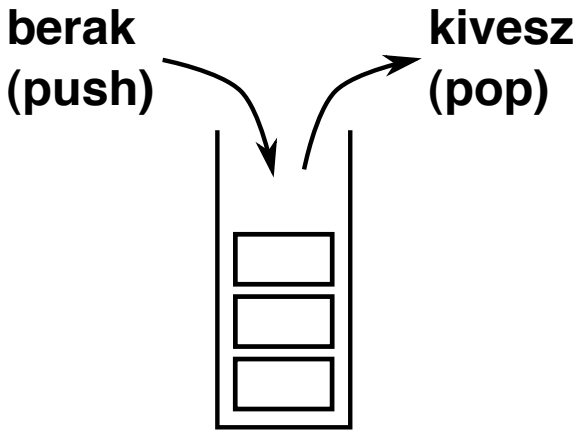
itos1.cpp

```
5  void szofordit(string* szo) {
6    int eleje, vege;
7    for(eleje=0, vege=(*szo).length()-1;
8        eleje<vege;
9        eleje++, vege--) {
10     char csere = (*szo)[eleje];
11     (*szo)[eleje] = (*szo)[vege];
12     (*szo)[vege] = csere;
13   }
14 }
```

Verem

Verem (Stack)

- LIFO (Last In, First Out) szervezésű tár: tárolás sorrendjével ellentétes sorrendben férünk hozzá az adatokhoz
- Megvalósítása: pl. tömbbel
- Műveletek: berak (push), kivesz (pop), kukucskál (peek), ürít (clear), ...



Egész szám átalakítása karakterlánccá

itos2.cpp

```
5  // verem muveletek
6  #define MAX 128
7  char verem[MAX];
8  int n = 0;
9
10 bool berak(char c) {
11   if(n < MAX) {
12     verem[n] = c;
13     n++;
14     return true;
15   } else {
16     return false;
17   }
18 }
```

Egész szám átalakítása karakterlánccá

itos2.cpp

```
20 char kivesz() {
21   if(n > 0) {
22     n--;
23     return verem[n];
24   } else {
25     return 0;
26   }
27 }
28
29 bool ures() {
30   return n==0;
31 }
```

Egész szám átalakítása karakterlánccá

itos2.cpp – karakterek sorrendjének megfordítása veremmel

```
33  // int --> string atalakitas
34  string itos(int szam) {
35    bool negativ = false;
36    if(szam < 0) {
37      szam *= -1;
38      negativ = true;
39    }
40    do {
41      berak(szam%10 + '0');
42      szam /= 10;
43    } while(szam != 0);
44    if(negativ) berak('-');
45    string s = "";
46    while(!ures()) {
47      s += kivesz();
48    }
49    return s;
50  }
```

Egész szám átalakítása karakterlánccá

Néha tényleg szükség van az adatok sorrendjének megfordítására, máskor elég ügyesen szervezni a programot ...

itos3.cpp

```
5  string itos(int szam) {
6    bool negativ = false;
7    if(szam < 0) {
8      szam *= -1;
9      negativ = true;
10   }
11   string s = "";
12   do {
13     s = char(szam%10 + '0') + s;
14     szam /= 10;
15   } while(szam != 0);
16   if(negativ) s = '-' + s;
17   return s;
18 }
```

Fordított lengyel logikájú (RPN) kalkulátor

$$(1 + 2) * (3 + 4) =$$
$$\downarrow$$
$$1\ 2 + 3\ 4 + * =$$

Algoritmus

```
while (a következő operátor vagy operandus nem az állomány vége)
    if (szám)
        told a verembe
    else if (operátor)
        léptesd ki az operandusokat
        végezd el a műveletet
        told a verembe az eredményt
    else
        hiba
```

Fordított lengyel logikájú (RPN) kalkulátor

rpn1.cpp

```
33  int main(void) {
34    string input;
35    do {
36      cout << "rpn: "; cin >> input;
37      if(input == "+") {
38        berak(kivesz() + kivesz());
39      } else if(input == "-") {
40        double op = kivesz();
41        berak(kivesz() - op);
42      } else if(input == "*") {
43        berak(kivesz() * kivesz());
44      } else if(input == "/") {
45        double op = kivesz();
46        berak(kivesz() / op);
47      } else if(input == "=") {
48        cout << kivesz() << endl;
49      } else { // szam
50        berak(stod(input)); // C++11
51      }
52    } while(input != "=");
53    return 0;
54  }
```

Kimenet

```
rpn: 1
rpn: 2
rpn: +
rpn: 3
rpn: 4
rpn: +
rpn: *
rpn: =
21
```

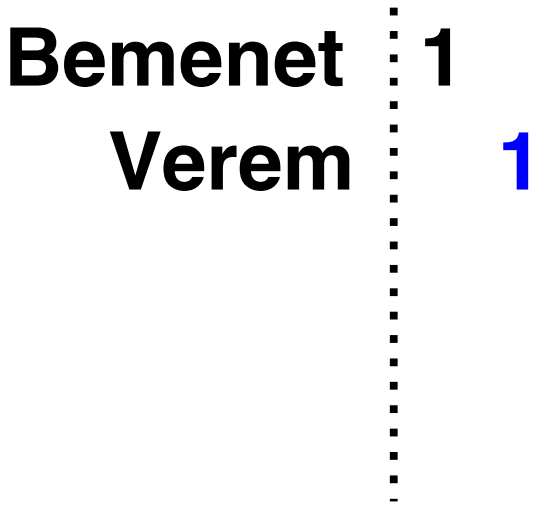
Fordított lengyel logikájú (RPN) kalkulátor

Veremállapot változása futás közben

Bemenet

Verem

Veremállapot változása futás közben

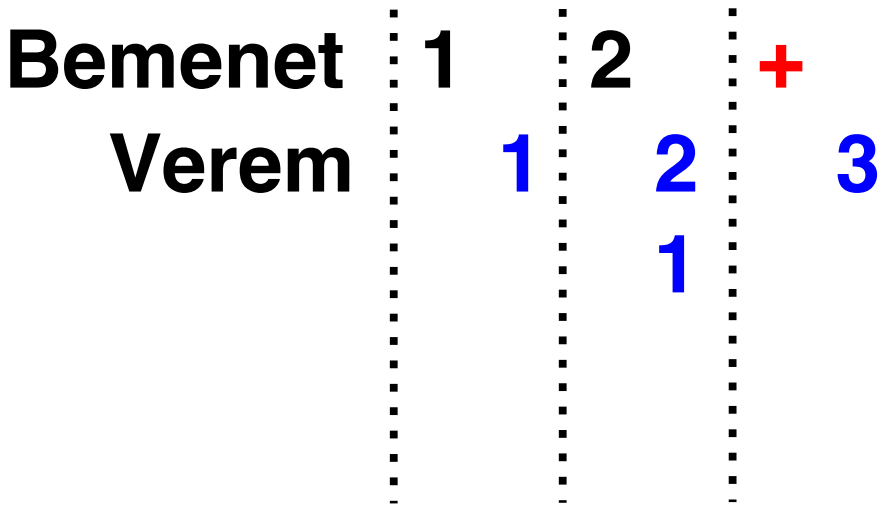


Fordított lengyel logikájú (RPN) kalkulátor

Veremállapot változása futás közben

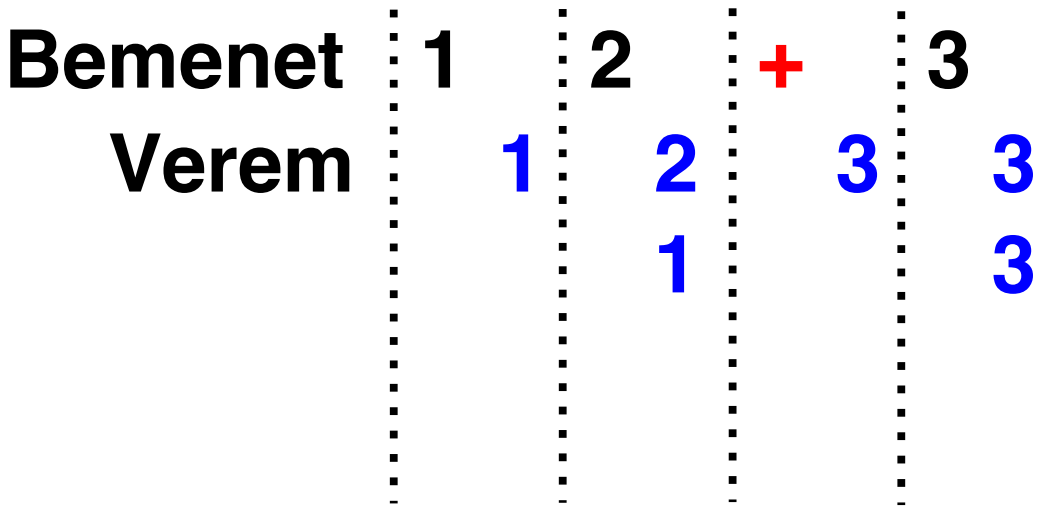
Fordított lengyel logikájú (RPN) kalkulátor

Veremállapot változása futás közben



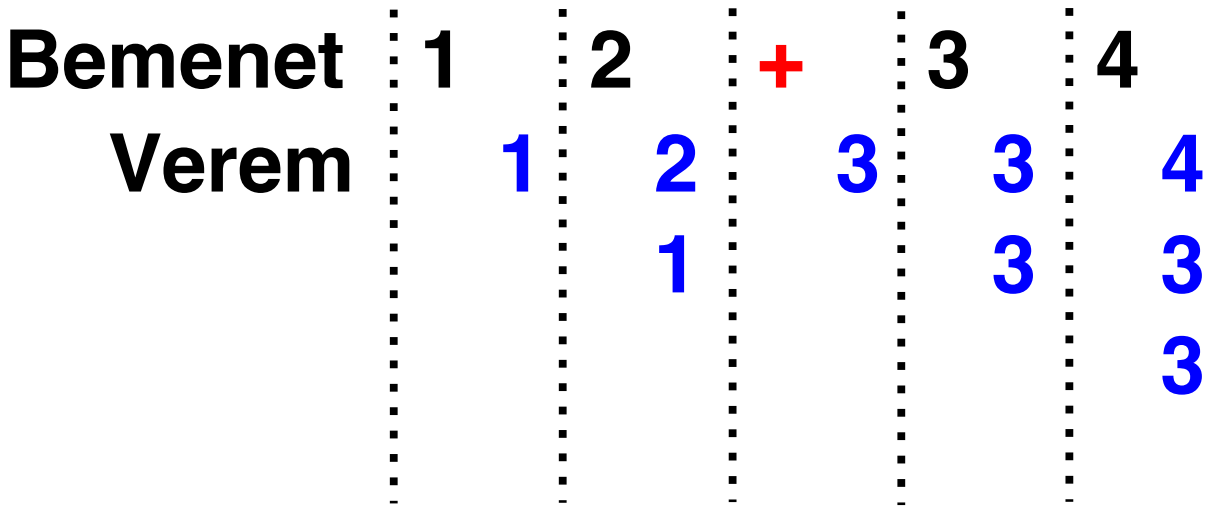
Fordított lengyel logikájú (RPN) kalkulátor

Veremállapot változása futás közben



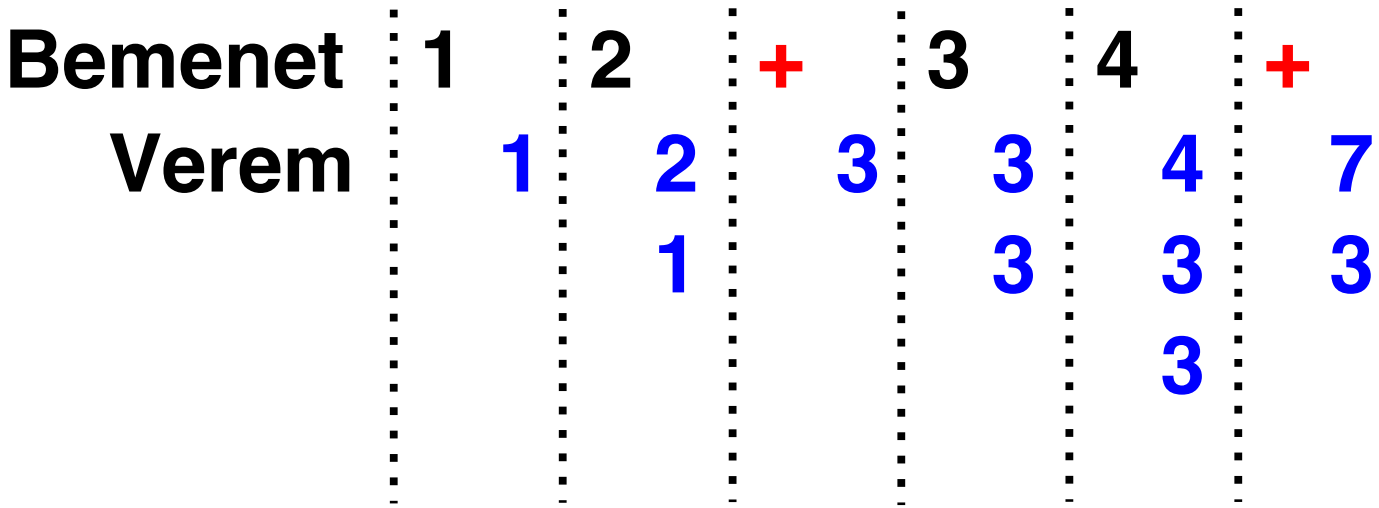
Fordított lengyel logikájú (RPN) kalkulátor

Veremállapot változása futás közben



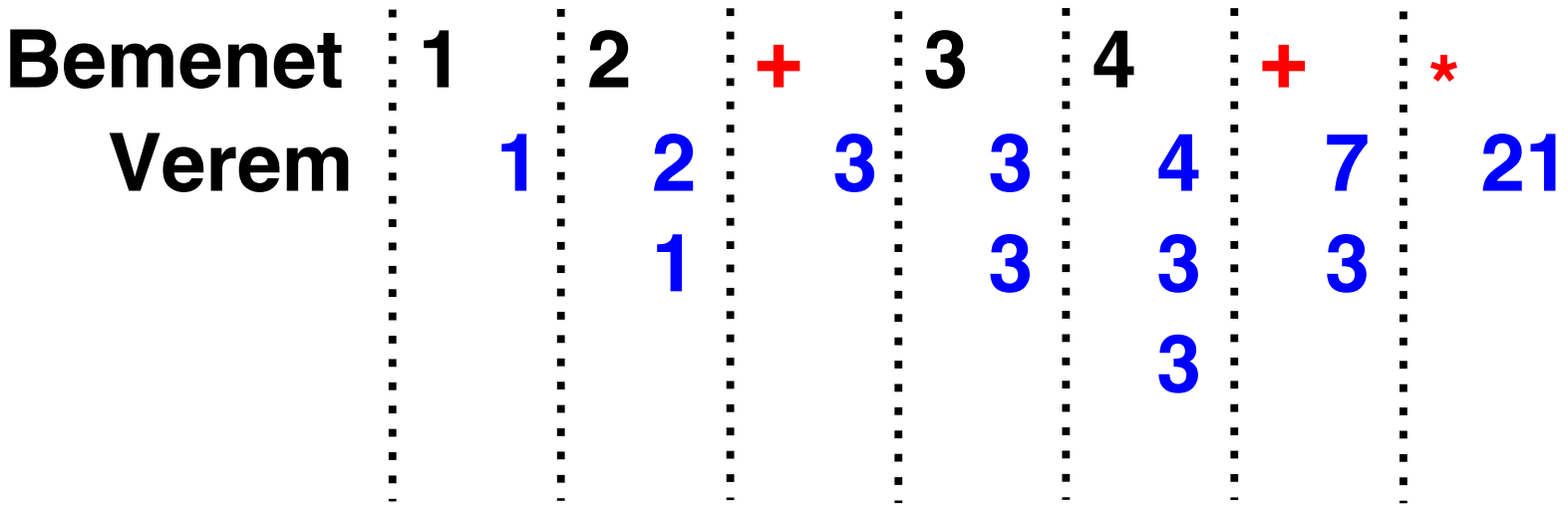
Fordított lengyel logikájú (RPN) kalkulátor

Veremállapot változása futás közben



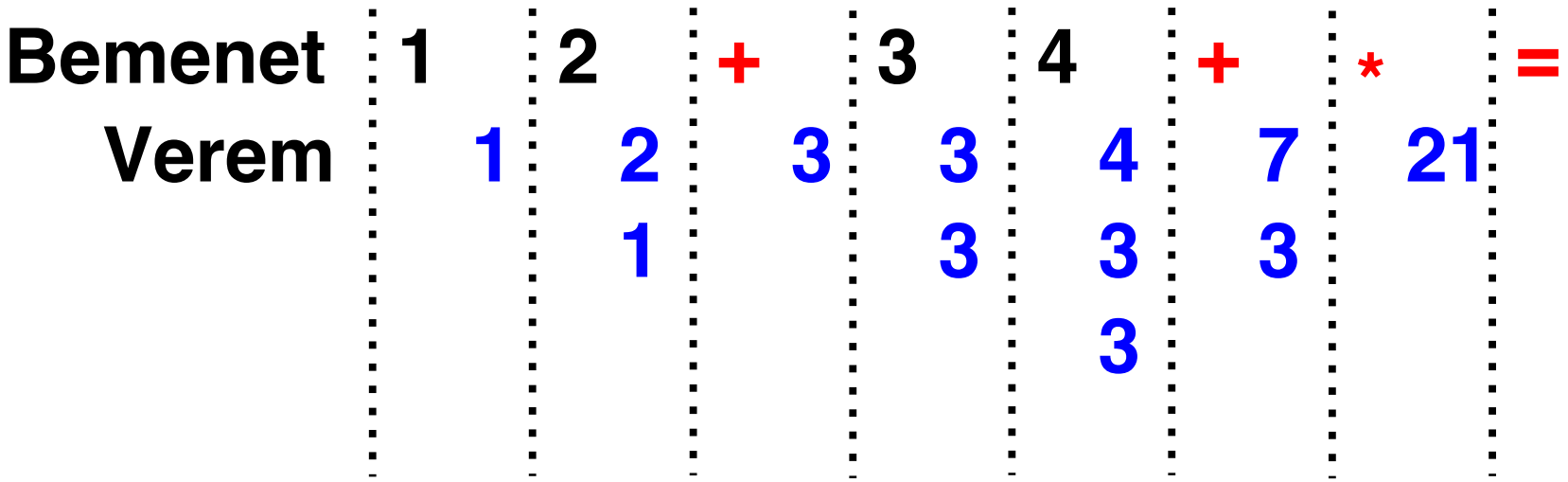
Fordított lengyel logikájú (RPN) kalkulátor

Veremállapot változása futás közben



Fordított lengyel logikájú (RPN) kalkulátor

Veremállapot változása futás közben



Kifejezések kiértékelése

Probléma

RPN alkalmazása nehézkes, szokatlan

Megoldás

1. Automatikus átalakítás infix alakról RPN-re, kiértékelés az ismert módszerrel
2. Infix alak megoldóját kell elkészíteni

Kifejezések kiértékelése

Szabályok legyenek a következők:

additiv
- multiplikativ
- multiplikativ + additiv
- multiplikativ − additiv

multiplikativ
- elsodleges
- elsodleges * multiplikativ
- elsodleges / multiplikativ

elsodleges
- (additiv)
- szam

Például az 1 + 2 így írható át:

additiv

↓

multiplikativ + additiv

↓

multiplikativ + multiplikativ

↓

elsodleges + elsodleges

↓

szam + szam

Kifejezések kiértékelése

Szabályok legyenek a következők:

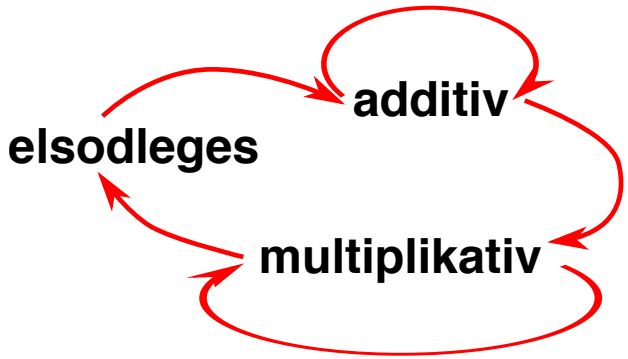
additiv
    multiplikativ
    multiplikativ + additiv
    multiplikativ − additiv

multiplikativ
    elsodleges
    elsodleges * multiplikativ
    elsodleges / multiplikativ

elsodleges
    (additiv)
    szam

Probléma

- `additiv`, `multiplikativ`, `elsodleges`: függvények
- kölcsönös rekurzió (mutual recursion) → nem lehet minden fv.-t a hívása előtt definiálni!



Deklaráció, definíció

deklaracio.cpp

```
4  // deklaracio
5  int osszead(); // csak visszateresi ertek es nev
6  int osszead();
7  int i;
8  /* valtozo ujradeklaralasa tiltott
9     error: redefinition of 'int i'
10 int i; */
11 struct s {
12   int tag1;
13   double tag2;
14 };
15 /* tipus (struktura) ujradeklaralasa titltott
16    error: redefinition of 'struct s'
17 struct s {
18   int tag1;
19   double tag2;
20 }; */
```

Deklaráció, definíció

deklaracio.cpp

```
22 // prototipus, azonositok nelkul
23 int osszead(int, int); // vissz. ertek, nev, parameterek
24                         // tipusa+sorrendje rogzitett
25 // prototipus, azonositokkal
26 int osszead(int x, int y); // azonositok hatokore csak
27 int osszead(int k, int l); // a prototipusra terjed ki
28
29 int main(void) {
30   cout << "2 + 3 = " << osszead(2, 3) << endl;
31   return 0;
32 }
33
34 // definicio: teljes formai informacio a fuggvenyrol
35 int osszead(int a, int b) {
36   return a+b;
37 }
38
39 /* ujradefinialas tiltott
40    error: redefinition of int osszead(int, int)
41 int osszead(int a, int b) {
42   return a+b;
43 } */
```

Kifejezések kiértékelése

kifejezes1.cpp

```
44  // kifejezes kiertekeles
45  double additiv();
46  double multiplikativ();
47  double elsodleges();
48
49  double additiv() {
50    double ertek = multiplikativ();
51    if(not ures()) {
52      string op = kivesz();
53      if(op == "+") {
54        return ertek + additiv();
55      } else if(op == "-") {
56        return ertek - additiv();
57      } else {
58        berak(op);
59      }
60    }
61    return ertek;
62  }
```

additiv
- multiplikativ
- multiplikativ + additiv
- multiplikativ − additiv

Kifejezések kiértékelése

kifejezes1.cpp

```
64  double multiplikativ() {
65    double ertek = elsodleges();
66    if(not ures()) {
67      string op = kivesz();
68      if(op == "*") {
69        return ertek * multiplikativ();
70      } else if(op == "/") {
71        return ertek / multiplikativ();
72      } else {
73        berak(op);
74      }
75    }
76    return ertek;
77  }
```

multiplikativ
- elsodleges
- elsodleges * multiplikativ
- elsodleges / multiplikativ

Kifejezések kiértékelése

kifejezes1.cpp

```
79  double elsodleges() {
80    if(not ures()) {
81      string kovetkezo = kivesz();
82      if(kovetkezo == "(") {
83        double ertek = additiv();
84        kivesz(); // ) eltavolitasa
85        return ertek;
86      } else {
87        return stod(kovetkezo); // C++11
88      }
89    } else {
90      return 0.;
91    }
92  }
```

elsodleges
    (additiv)
    szam

Kifejezések kiértékelése

kifejezes1.cpp

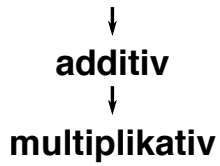
```
94  int main() {
95    string input;
96    while(cout<<"Kif.: ", cin>>input, input!="=") {
97      berak(input);
98    }
99    fordit();
100   cout << additiv();
101   return 0;
102 }

33  void fordit() {
34    int eleje, vege;
35    for(eleje=0, vege=n-1;
36        eleje<vege;
37        eleje++, vege--) {
38      string csere = verem[eleje];
39      verem[eleje] = verem[vege];
40      verem[vege] = csere;
41    }
42  }
```

Kifejezések kiértékelése: (1+2)*(3+4)=

Verem
(
1
+
2
)
*
(
3
+
4
)

```
49  double additiv() {
50    double ertek = multiplikativ();
51    if(not ures()) {
52      string op = kivesz();
53      if(op == "+") {
54        return ertek + additiv();
55      } else if(op == "-") {
56        return ertek - additiv();
57      } else {
58        berak(op);
59      }
60    }
61    return ertek;
62  }
```

Kifejezések kiértékelése: (1+2)*(3+4)=

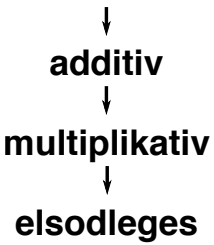


Verem

(
1
+
2
)
*
(
3
+
4
)

```
64  double multiplikativ() {
65    double ertek = elsodleges();
66    if(not ures()) {
67      string op = kivesz();
68      if(op == "*") {
69        return ertek * multiplikativ();
70      } else if(op == "/") {
71        return ertek / multiplikativ();
72      } else {
73        berak(op);
74      }
75    }
76    return ertek;
77  }
```

Kifejezések kiértékelése: (1+2)*(3+4)=

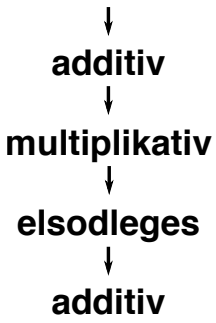


Verem

(
1
+
2
)
*
(
3
+
4
)

```
79  double elsodleges() {
80    if(not ures()) {
81      string kovetkezo = kivesz();
82      if(kovetkezo == "(") {
83        double ertek = additiv();
84        kivesz(); // ) eltavolitasa
85        return ertek;
86      } else {
87        return stod(kovetkezo); // C++11
88      }
89    } else {
90      return 0.;
91    }
92  }
```

Kifejezések kiértékelése: (1+2)*(3+4)=

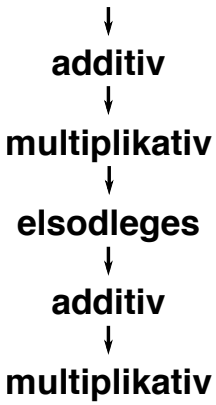


Verem

1
+
2
)
*
(
3
+
4
)

```
49  double additiv() {
50    double ertek = multiplikativ();
51    if(not ures()) {
52      string op = kivesz();
53      if(op == "+") {
54        return ertek + additiv();
55      } else if(op == "-") {
56        return ertek - additiv();
57      } else {
58        berak(op);
59      }
60    }
61    return ertek;
62  }
```

Kifejezések kiértékelése: (1+2)*(3+4)=

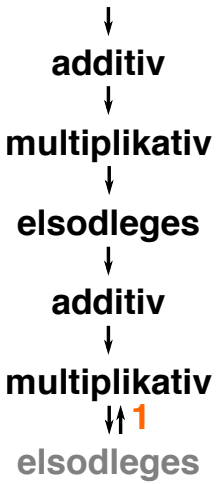


Verem

1
+
2
)
*
(
3
+
4
)

```
64  double multiplikativ() {
65    double ertek = elsodleges();
66    if(not ures()) {
67      string op = kivesz();
68      if(op == "*") {
69        return ertek * multiplikativ();
70      } else if(op == "/") {
71        return ertek / multiplikativ();
72      } else {
73        berak(op);
74      }
75    }
76    return ertek;
77  }
```

Kifejezések kiértékelése: (1+2)*(3+4)=

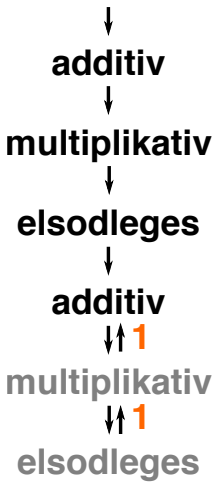


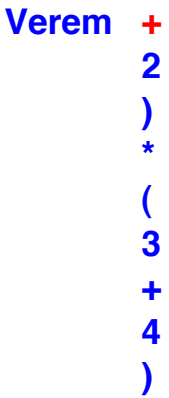
Verem

+
2
)
*
(
3
+
4
)

```
79  double elsodleges() {
80    if(not ures()) {
81      string kovetkezo = kivesz();
82      if(kovetkezo == "(") {
83        double ertek = additiv();
84        kivesz(); // ) eltavolitasa
85        return ertek;
86      } else {
87        return stod(kovetkezo); // C++11
88      }
89    } else {
90      return 0.;
91    }
92  }
```

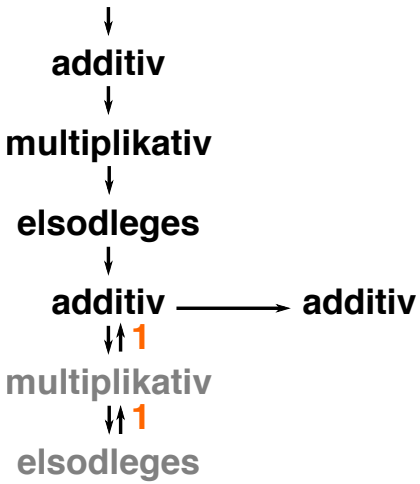
Kifejezések kiértékelése: (1+2)*(3+4)=






```
64  double multiplikativ() {
65    double ertek = elsodleges();
66    if(not ures()) {
67      string op = kivesz();
68      if(op == "*") {
69        return ertek * multiplikativ();
70      } else if(op == "/") {
71        return ertek / multiplikativ();
72      } else {
73        berak(op);
74      }
75    }
76    return ertek;
77  }
```

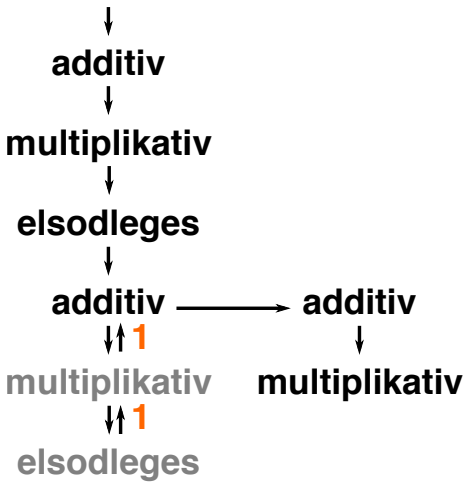
Kifejezések kiértékelése: (1+2)*(3+4)=



Verem 2
)
*
(
3
+
4
)

```
49  double additiv() {
50    double ertek = multiplikativ();
51    if(not ures()) {
52      string op = kivesz();
53      if(op == "+") {
54        return ertek + additiv();
55      } else if(op == "-") {
56        return ertek - additiv();
57      } else {
58        berak(op);
59      }
60    }
61    return ertek;
62  }
```

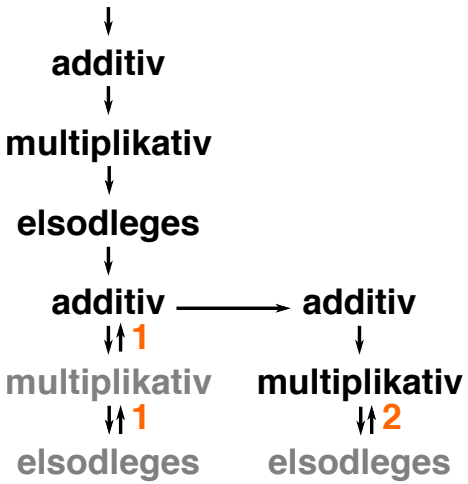
Kifejezések kiértékelése: (1+2)*(3+4)=



Verem 2
)
*
(
3
+
4
)

```
64  double multiplikativ() {
65    double ertek = elsodleges();
66    if(not ures()) {
67      string op = kivesz();
68      if(op == "*") {
69        return ertek * multiplikativ();
70      } else if(op == "/") {
71        return ertek / multiplikativ();
72      } else {
73        berak(op);
74      }
75    }
76    return ertek;
77  }
```

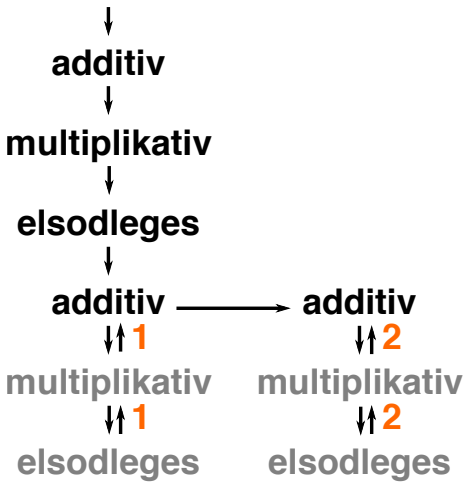
Kifejezések kiértékelése: (1+2)*(3+4)=



Verem
)
*
(
3
+
4
)

```
79  double elsodleges() {
80    if(not ures()) {
81      string kovetkezo = kivesz();
82      if(kovetkezo == "(") {
83        double ertek = additiv();
84        kivesz(); // ) eltavolitasa
85        return ertek;
86      } else {
87        return stod(kovetkezo); // C++11
88      }
89    } else {
90      return 0.;
91    }
92  }
```

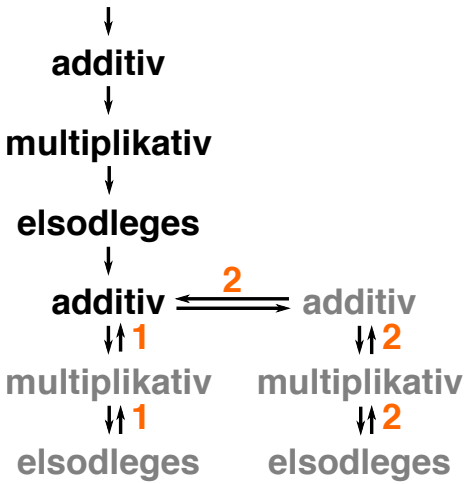
Kifejezések kiértékelése: (1+2)*(3+4)=



Verem
)
*
(
3
+
4
)

```
64  double multiplikativ() {
65    double ertek = elsodleges();
66    if(not ures()) {
67      string op = kivesz();
68      if(op == "*") {
69        return ertek * multiplikativ();
70      } else if(op == "/") {
71        return ertek / multiplikativ();
72      } else {
73        berak(op);
74      }
75    }
76    return ertek;
77  }
```

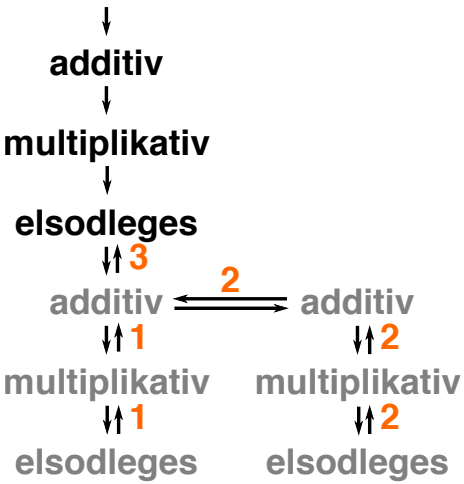
Kifejezések kiértékelése: (1+2)*(3+4)=

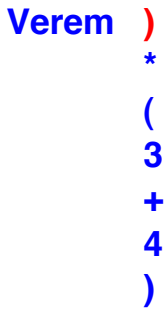


Verem
)
*
(
3
+
4
)

```
49  double additiv() {
50    double ertek = multiplikativ();
51    if(not ures()) {
52      string op = kivesz();
53      if(op == "+") {
54        return ertek + additiv();
55      } else if(op == "-") {
56        return ertek - additiv();
57      } else {
58        berak(op);
59      }
60    }
61    return ertek;
62  }
```

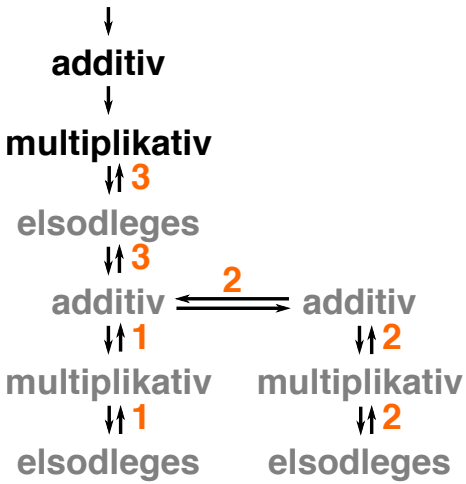
Kifejezések kiértékelése: (1+2)*(3+4)=






```
49  double additiv() {
50    double ertek = multiplikativ();
51    if(not ures()) {
52      string op = kivesz();
53      if(op == "+") {
54        return ertek + additiv();
55      } else if(op == "-") {
56        return ertek - additiv();
57      } else {
58        berak(op);
59      }
60    }
61    return ertek;
62  }
```

Kifejezések kiértékelése: (1+2)*(3+4)=



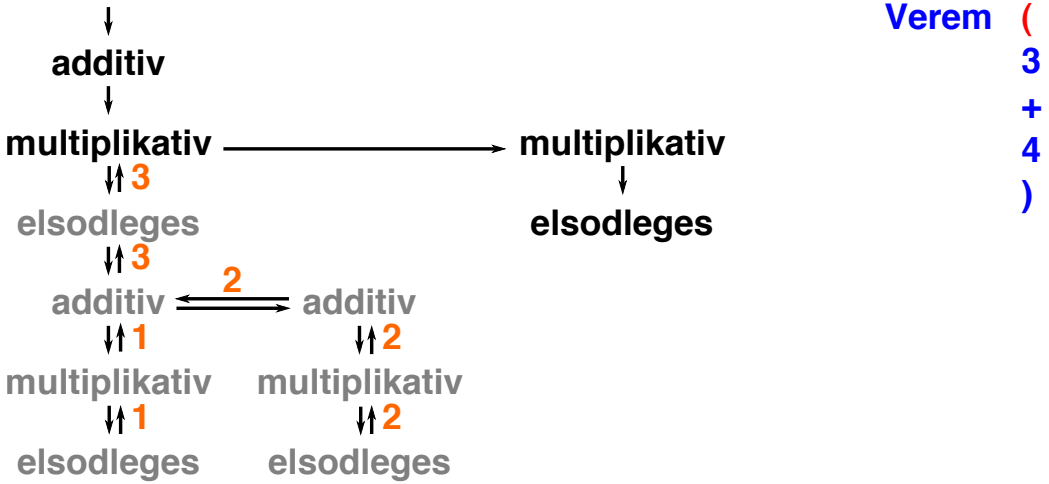
Verem *
(
3
+
4
)

```
79  double elsodleges() {
80    if(not ures()) {
81      string kovetkezo = kivesz();
82      if(kovetkezo == "(") {
83        double ertek = additiv();
84        kivesz(); // ) eltavolitasa
85        return ertek;
86      } else {
87        return stod(kovetkezo); // C++11
88      }
89    } else {
90      return 0.;
91    }
92  }
```

Kifejezések kiértékelése: (1+2)*(3+4)=

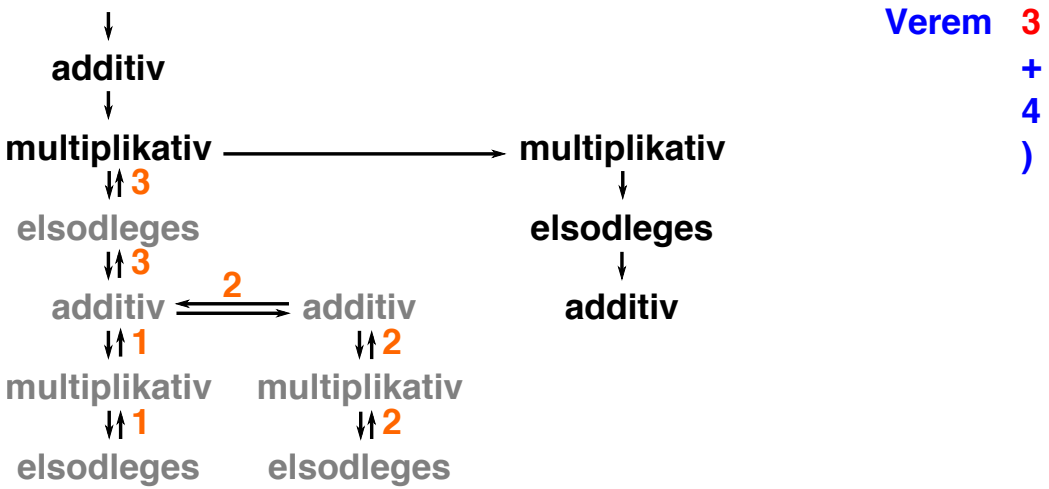

```
64  double multiplikativ() {
65    double ertek = elsodleges();
66    if(not ures()) {
67      string op = kivesz();
68      if(op == "*") {
69        return ertek * multiplikativ();
70      } else if(op == "/") {
71        return ertek / multiplikativ();
72      } else {
73        berak(op);
74      }
75    }
76    return ertek;
77  }
```

Kifejezések kiértékelése: (1+2)*(3+4)=



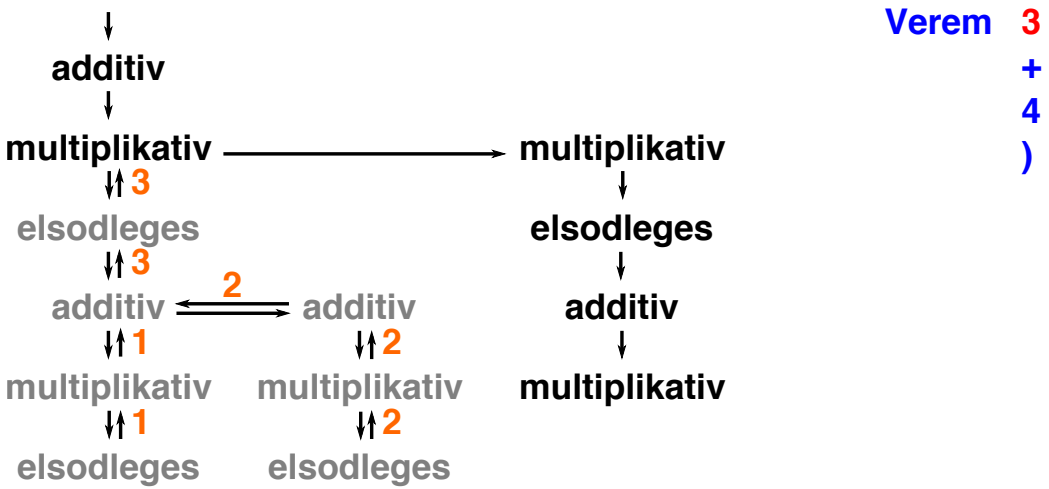

```
79  double elsodleges() {
80    if(not ures()) {
81      string kovetkezo = kivesz();
82      if(kovetkezo == "(") {
83        double ertek = additiv();
84        kivesz(); // ) eltavolitasa
85        return ertek;
86      } else {
87        return stod(kovetkezo); // C++11
88      }
89    } else {
90      return 0.;
91    }
92  }
```

Kifejezések kiértékelése: (1+2)*(3+4)=



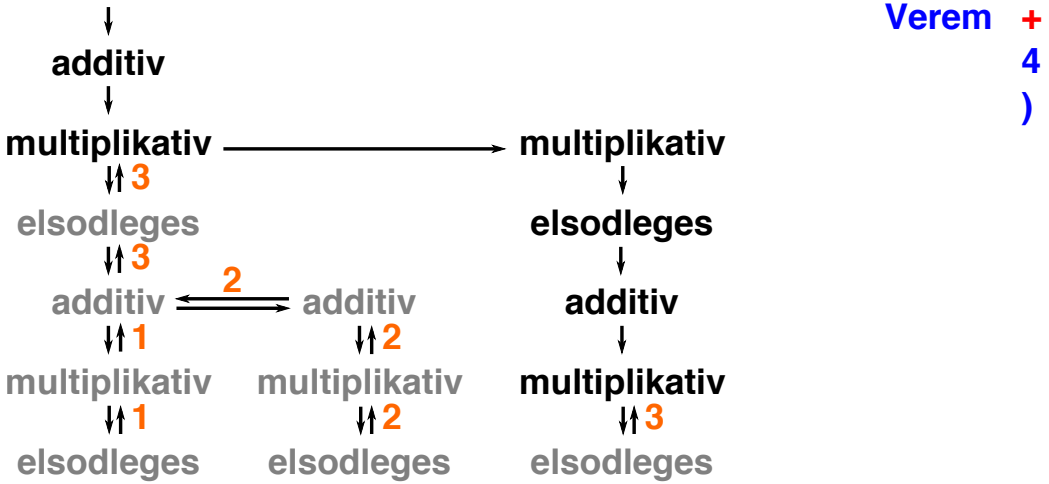

```
49  double additiv() {
50    double ertek = multiplikativ();
51    if(not ures()) {
52      string op = kivesz();
53      if(op == "+") {
54        return ertek + additiv();
55      } else if(op == "-") {
56        return ertek - additiv();
57      } else {
58        berak(op);
59      }
60    }
61    return ertek;
62  }
```

Kifejezések kiértékelése: (1+2)*(3+4)=



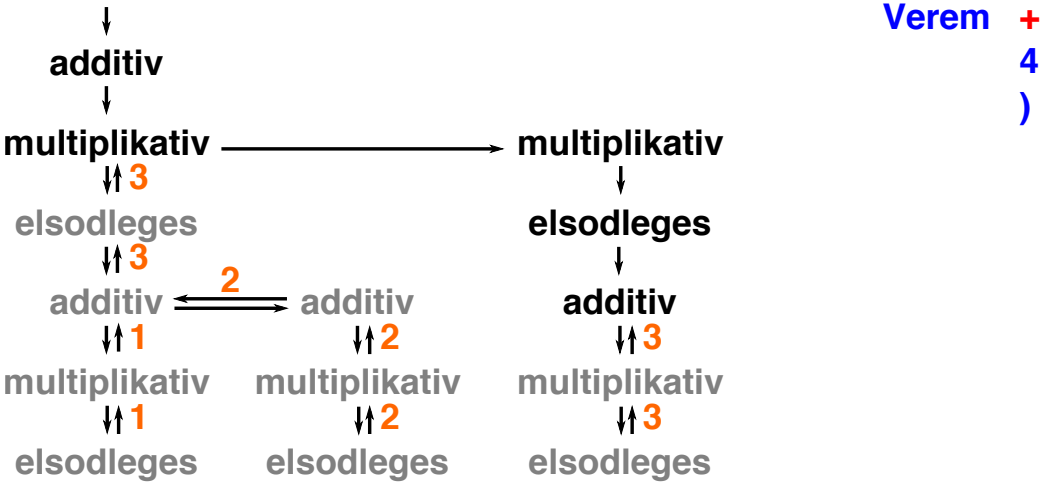

```
64  double multiplikativ() {
65    double ertek = elsodleges();
66    if(not ures()) {
67      string op = kivesz();
68      if(op == "*") {
69        return ertek * multiplikativ();
70      } else if(op == "/") {
71        return ertek / multiplikativ();
72      } else {
73        berak(op);
74      }
75    }
76    return ertek;
77  }
```

Kifejezések kiértékelése: (1+2)*(3+4)=



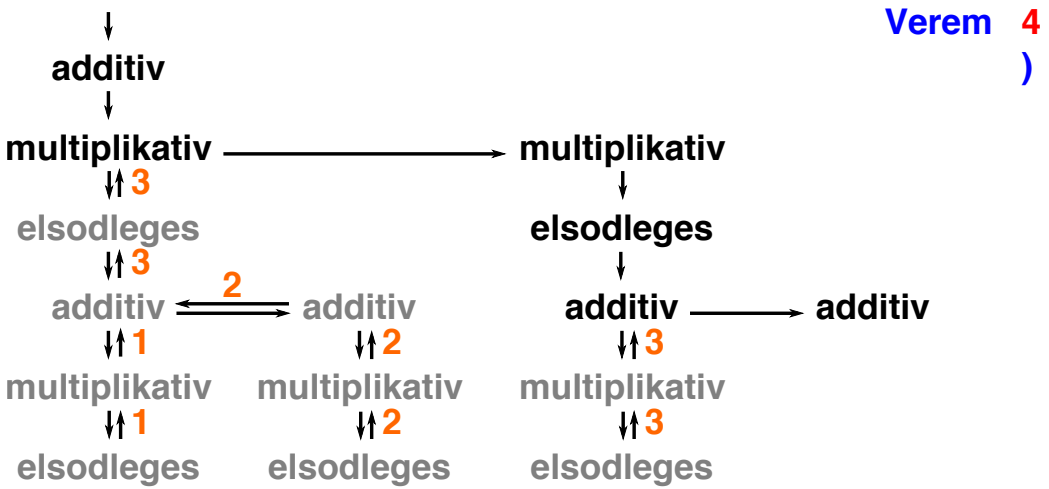

```
79  double elsodleges() {
80    if(not ures()) {
81      string kovetkezo = kivesz();
82      if(kovetkezo == "(") {
83        double ertek = additiv();
84        kivesz(); // ) eltavolitasa
85        return ertek;
86      } else {
87        return stod(kovetkezo); // C++11
88      }
89    } else {
90      return 0.;
91    }
92  }
```

Kifejezések kiértékelése: (1+2)*(3+4)=



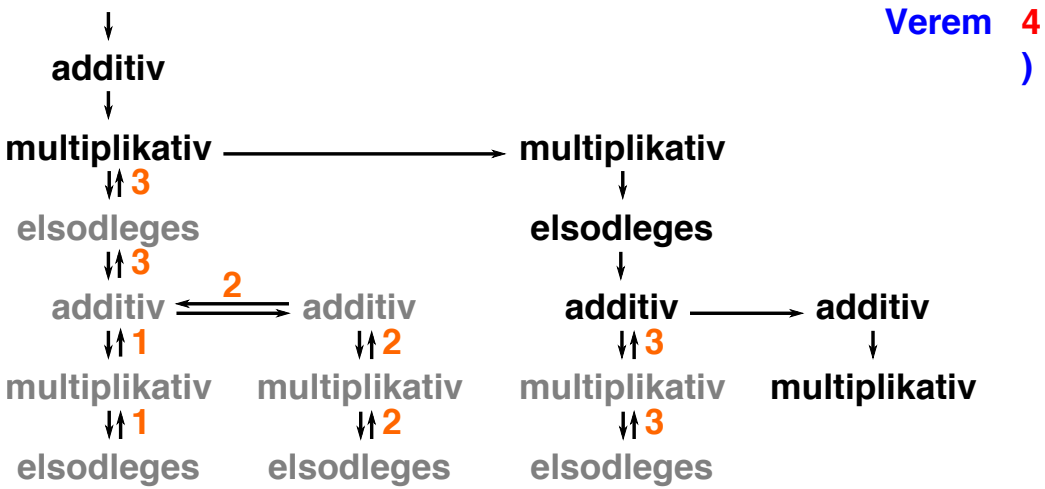

```
64  double multiplikativ() {
65    double ertek = elsodleges();
66    if(not ures()) {
67      string op = kivesz();
68      if(op == "*") {
69        return ertek * multiplikativ();
70      } else if(op == "/") {
71        return ertek / multiplikativ();
72      } else {
73        berak(op);
74      }
75    }
76    return ertek;
77  }
```

Kifejezések kiértékelése: (1+2)*(3+4)=



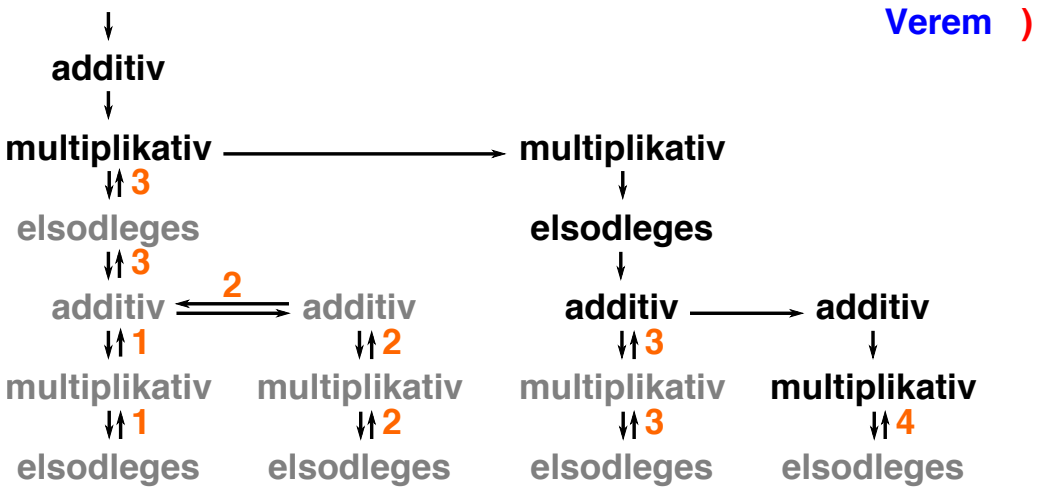

```
49  double additiv() {
50    double ertek = multiplikativ();
51    if(not ures()) {
52      string op = kivesz();
53      if(op == "+") {
54        return ertek + additiv();
55      } else if(op == "-") {
56        return ertek - additiv();
57      } else {
58        berak(op);
59      }
60    }
61    return ertek;
62  }
```

Kifejezések kiértékelése: (1+2)*(3+4)=



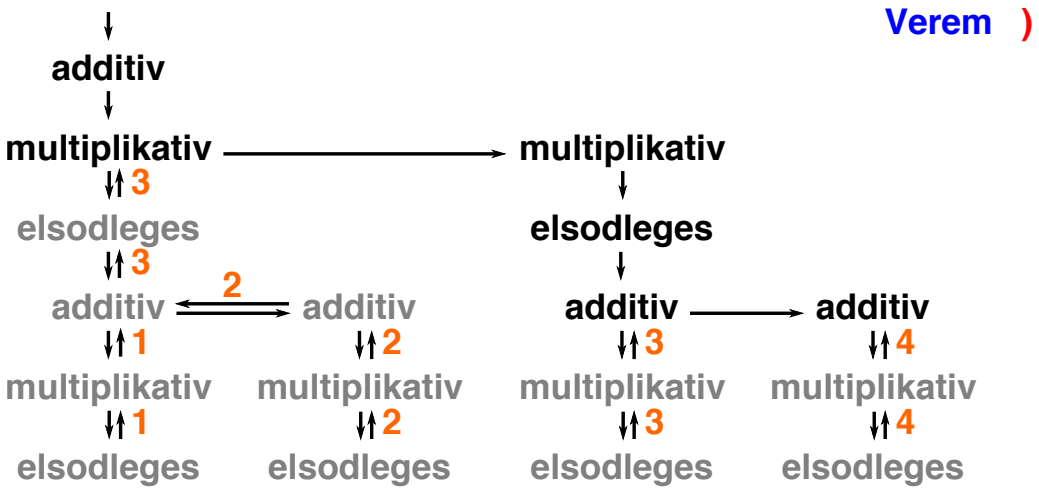

```
64  double multiplikativ() {
65    double ertek = elsodleges();
66    if(not ures()) {
67      string op = kivesz();
68      if(op == "*") {
69        return ertek * multiplikativ();
70      } else if(op == "/") {
71        return ertek / multiplikativ();
72      } else {
73        berak(op);
74      }
75    }
76    return ertek;
77  }
```

Kifejezések kiértékelése: (1+2)*(3+4)=



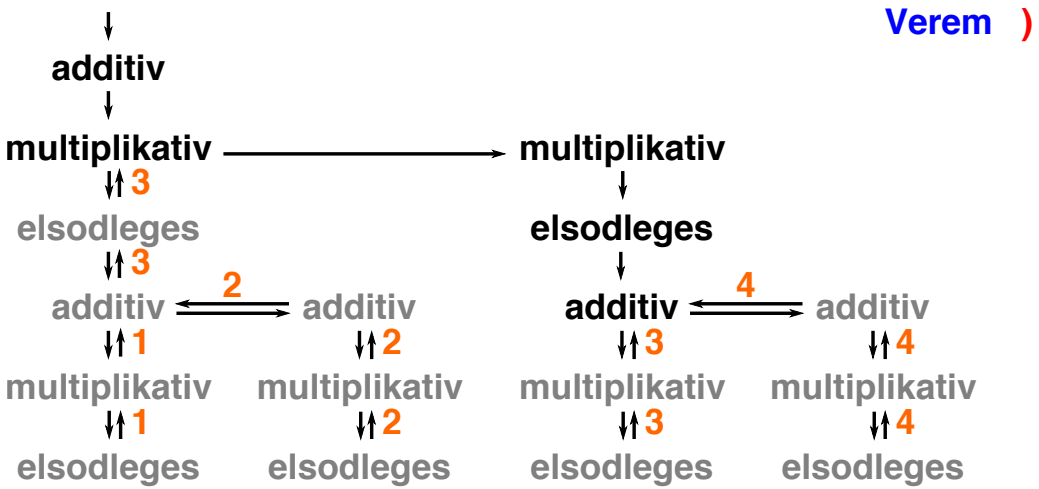

```
79  double elsodleges() {
80    if(not ures()) {
81      string kovetkezo = kivesz();
82      if(kovetkezo == "(") {
83        double ertek = additiv();
84        kivesz(); // ) eltavolitasa
85        return ertek;
86      } else {
87        return stod(kovetkezo); // C++11
88      }
89    } else {
90      return 0.;
91    }
92  }
```

Kifejezések kiértékelése: (1+2)*(3+4)=



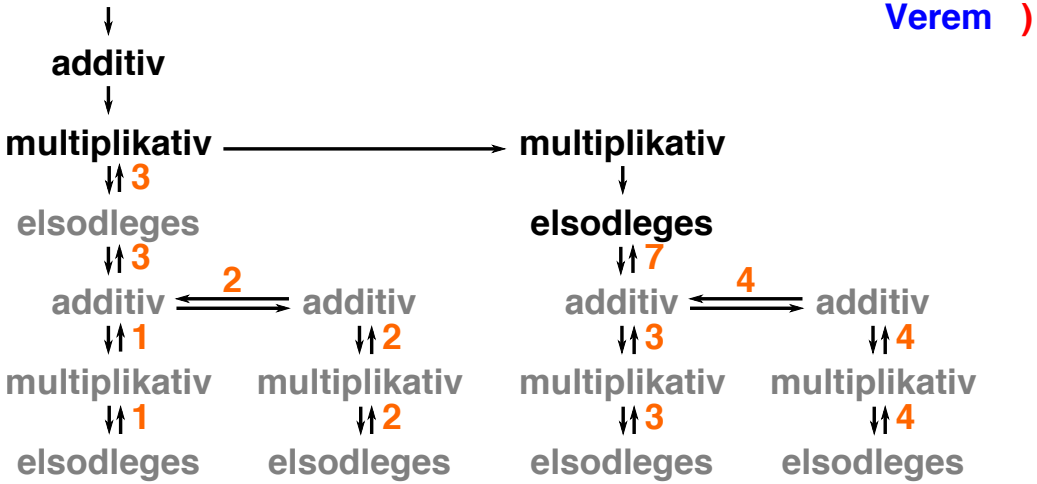

```
64  double multiplikativ() {
65    double ertek = elsodleges();
66    if(not ures()) {
67      string op = kivesz();
68      if(op == "*") {
69        return ertek * multiplikativ();
70      } else if(op == "/") {
71        return ertek / multiplikativ();
72      } else {
73        berak(op);
74      }
75    }
76    return ertek;
77  }
```

Kifejezések kiértékelése: (1+2)*(3+4)=



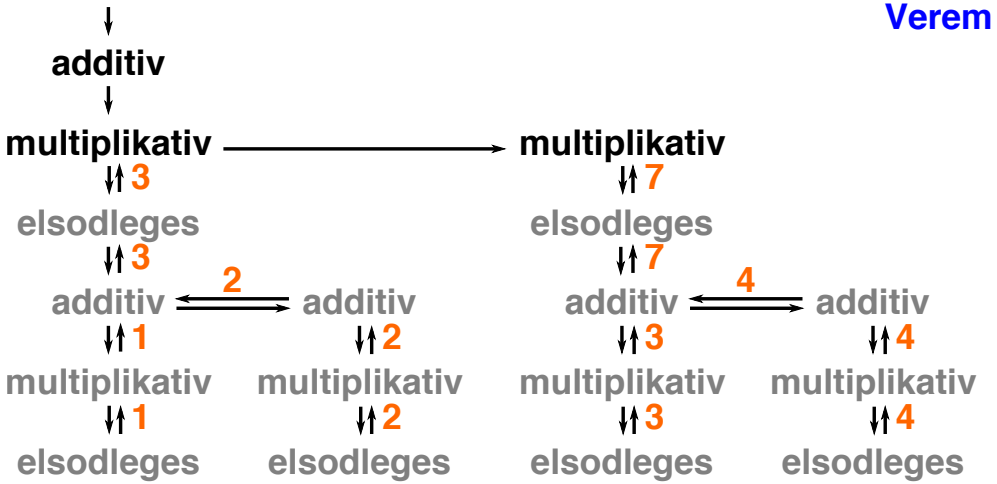

```
49  double additiv() {
50    double ertek = multiplikativ();
51    if(not ures()) {
52      string op = kivesz();
53      if(op == "+") {
54        return ertek + additiv();
55      } else if(op == "-") {
56        return ertek - additiv();
57      } else {
58        berak(op);
59      }
60    }
61    return ertek;
62  }
```

Kifejezések kiértékelése: (1+2)*(3+4)=



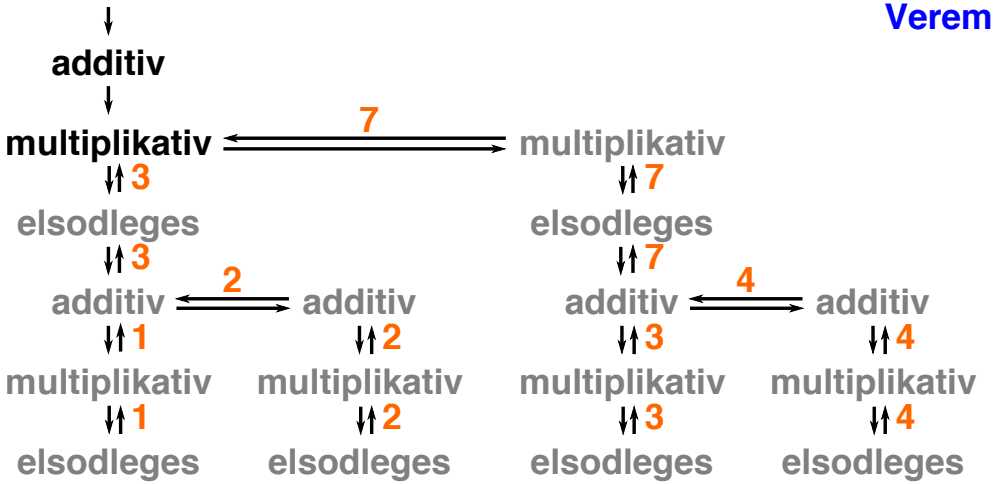

```
79  double elsodleges() {
80    if(not ures()) {
81      string kovetkezo = kivesz();
82      if(kovetkezo == "(") {
83        double ertek = additiv();
84        kivesz(); // ) eltavolitasa
85        return ertek;
86      } else {
87        return stod(kovetkezo); // C++11
88      }
89    } else {
90      return 0.;
91    }
92  }
```

Kifejezések kiértékelése: (1+2)*(3+4)=



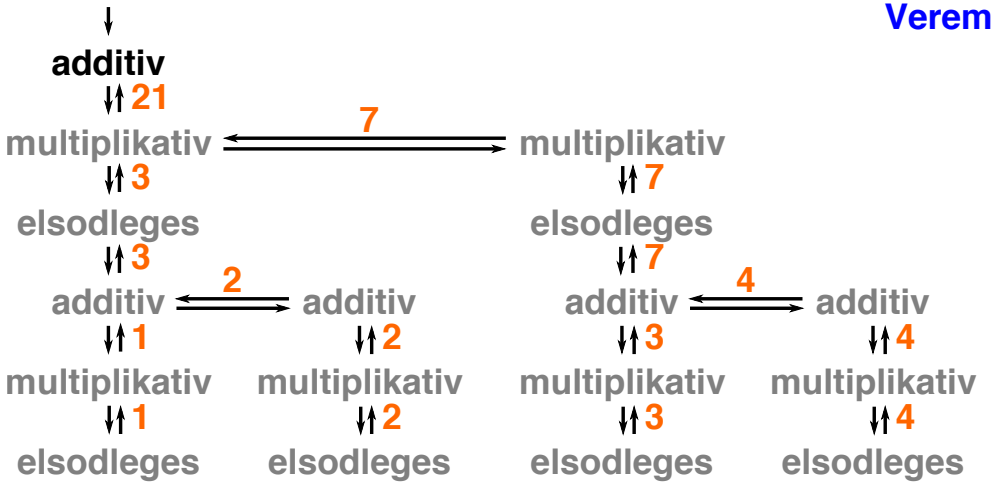

```
64  double multiplikativ() {
65    double ertek = elsodleges();
66    if(not ures()) {
67      string op = kivesz();
68      if(op == "*") {
69        return ertek * multiplikativ();
70      } else if(op == "/") {
71        return ertek / multiplikativ();
72      } else {
73        berak(op);
74      }
75    }
76    return ertek;
77  }
```

Kifejezések kiértékelése: (1+2)*(3+4)=



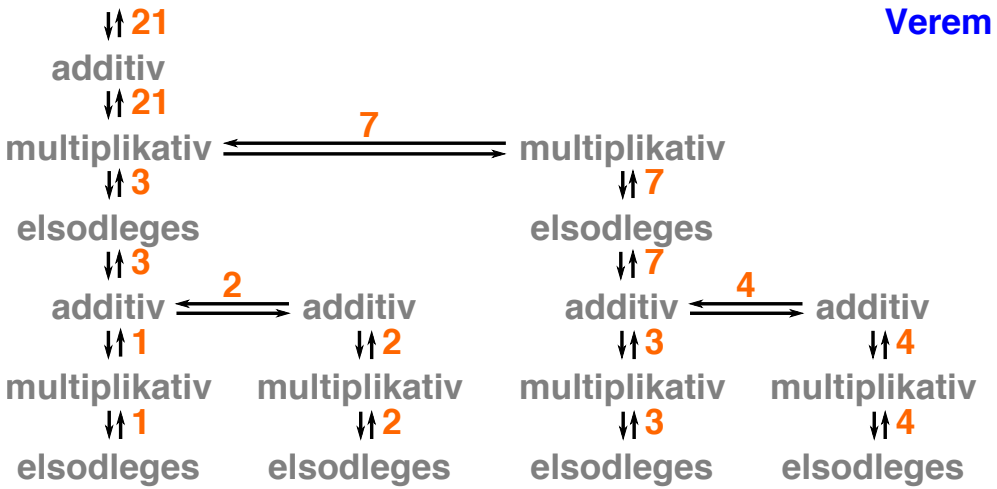

```
64  double multiplikativ() {
65    double ertek = elsodleges();
66    if(not ures()) {
67      string op = kivesz();
68      if(op == "*") {
69        return ertek * multiplikativ();
70      } else if(op == "/") {
71        return ertek / multiplikativ();
72      } else {
73        berak(op);
74      }
75    }
76    return ertek;
77  }
```

Kifejezések kiértékelése: (1+2)*(3+4)=




```
49  double additiv() {
50    double ertek = multiplikativ();
51    if(not ures()) {
52      string op = kivesz();
53      if(op == "+") {
54        return ertek + additiv();
55      } else if(op == "-") {
56        return ertek - additiv();
57      } else {
58        berak(op);
59      }
60    }
61    return ertek;
62  }
```

Kifejezések kiértékelése: (1+2)*(3+4)=




```
49  double additiv() {
50    double ertek = multiplikativ();
51    if(not ures()) {
52      string op = kivesz();
53      if(op == "+") {
54        return ertek + additiv();
55      } else if(op == "-") {
56        return ertek - additiv();
57      } else {
58        berak(op);
59      }
60    }
61    return ertek;
62  }
```

Projekt

Eddigi programjaink egyetlen forrásfájlból álltak: pl. `kifejezes1.cpp`
Problémák:

- Áttekinthetetlenül nagyra nőnek a forrásszövegek
- Több programozó együttes munkája nehézkes

Megoldás:

- több forrásfájl $\rightarrow$ egy program
- forrásfájlok összekapcsolása: projekt segítségével (pl. `makefile`)

Feladat:

- Alakítsuk át a kifejezéseket kiértékelő programot!
- A verem kezelése eléggé általános, sok helyen felhasználásra kerülő, logikailag összetartozó függvényeket igényel $\rightarrow$ helyezzük ezeket külön forrásfájlba!

Projekt

Régi módszer:

kifejezes1.cpp

```
bool berak(string s) { /* ... */ }
string kivesz() { /* ... */ }
bool ures() { /* ... */ }
void fordit() { /* ... */ }

double additiv() { /* ... */ }
double multiplikativ() { /* ... */ }
double elsodleges() { /* ... */ }
int main() { /* ... */ }
```

Új módszer:

kifejezes2.cpp

```
double additiv() { /* ... */ }
double multiplikativ() { /* ... */ }
double elsodleges() { /* ... */ }
int main() { /* ... */ }
```

↓ `#include "verem2.h"`

verem2.h

```
// veremfv.-ek prototípusai
```

↑ `#include "verem2.h"`

verem2.cpp

```
bool berak(string s) { /* ... */ }
string kivesz() { /* ... */ }
bool ures() { /* ... */ }
void fordit() { /* ... */ }
```

Régi módszer:

Fordítás

```
g++ -std=c++11 -Wall -c kifejezes1.cpp
```

Kapcsoló-szerkesztés

```
g++ -o kifejezes1 kifejezes1.o
```

Összeállítás

```
g++ -std=c++11 -Wall -o kifejezes1 kifejezes1.cpp
```

Új módszer:

Fordítás

```
g++ -Wall -c verem2.cpp
g++ -std=c++11 -Wall -c kifejezes2.cpp
```

Kapcsoló-szerkesztés

```
g++ -o kifejezes2 kifejezes2.o verem2.o
```

Összeállítás

```
g++ -std=c++11 -Wall -o kifejezes2 verem2.cpp kifejezes2.cpp
```

Projekt

verem2.cpp

```
1  #include <string>
2  #include "verem2.h"
3  using namespace std;
4
5  #define MAX 128
6  // hatokor korlatozas a verem2.cpp-re
7  static string verem[MAX];
8  static int n = 0;
9
10 bool berak(string s) {
11   if(n < MAX) {
12     verem[n] = s;
13     n++;
14     return true;
15   } else {
16     return false;
17   }
18 }
```

Projekt

verem2.h

```
bool berak(std::string s);
std::string kivesz();
bool ures();
void fordit();
```

kifejezes2.cpp

```
1  #include <iostream>
2  #include <string> // stod()
3  #include "verem2.h"
4  using namespace std;
5
6  // kifejezes kiertekeles
7  double additiv();
8  double multiplikativ();
9  double elsodleges();
10
11 double additiv() {
```

Projekt

Készítsünk programot, mely kezeli

- gépkocsik adatait
  - nyilvántartja: gyártás dátumát, utolsó műszaki vizsga dátumát, mv. darabszámát
  - kiszámítja: meddig érvényes a mv.? (az első 4 évre szól, minden későbbi 2-re)
- emberek adatait
  - nyilvántartja: nevet, születési dátumot
  - kiszámítja: elmúlt-e már 17 éves (kaphat-e „B" kat. jogosítványt)
- majd ezek felhasználásával kiszámítja
  - van jogosítvány + érv. műszaki v. = mehetünk autózni
  - van jogosítvány + lejárt műszaki v. = le kell műszakiztatni az autót
  - nincs jogosítvány = kicsit várunk még

Projekt

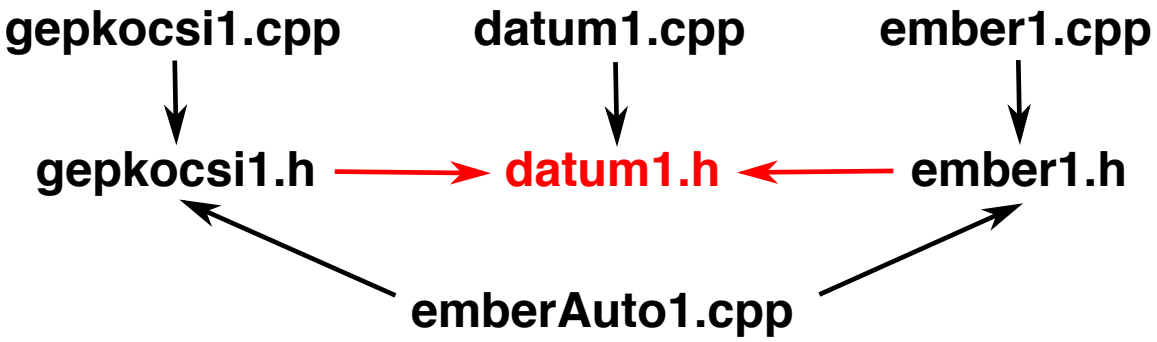
ember1.h

```
#include "datum1.h"

struct ember {
  std::string nev;
  datum szuletes;
};

bool elmult17(const ember* e, const datum* ma);
```

gepkocsi1.h

```
#include "datum1.h"

struct gepkocsi {
  datum gyartas;
  datum utolsoMuszaki;
  int muszakiDb;
};

datum muszakiErvenyesseg(const gepkocsi* gk);
```

Projekt



Probléma: újradefiniált típus

In file included from ember1.h:1:0, from emberAuto1.cpp:3:
datum1.h:3:8: error: redefinition of 'struct datum'

In file included from gepkocsi1.h:1:0, from emberAuto1.cpp:2:
datum1.h:3:8: error: previous definition of 'struct datum'

Források: gepkocsi1.cpp, datum1.cpp, ember1.cpp, emberAuto1.cpp
Fejfájlok: gepkocsi1.h, datum1.h, ember1.h

Előfeldolgozó

Feltételes fordítás: feltételektől függően bizonyos programrészek megőrzése/kihagyása

Feltétles fordítás

```
#if konstans-kifejezés1 <szekció1>
<#elif konstans-kifejezés2 <szekció2>>
/* ... */
<#elif konstans-kifejezésN <szekcióN>>
<#else <végső-szekció>>
#endif
```

- A `konstans-kifejezések` típusa logikainak tekintendő.
- Csak karakter és egész állandókat, `defined` operátort tartalmazhat

Előfeldolgozó

A `defined` operátor

- célja: makrók definiáltságát ellenőrzi
- alakjai:
  `defined(`*azonosító*`)`
  `defined` *azonosító*
- válasz: logikai, és logikai operátorokkal együtt használható
- alkalmazása: pl. biztosítja, hogy egy fejfájl egyszer kerülhessen csak beépítésre (include/header guard), platform-specifikus megoldások közül egynek a használata

fej.h

```
#if !defined(FEJ)
  #define FEJ
  /* érdemi tartalom */
#endif
```

Az `#ifdef`, `#ifndef` direktívák

- céljuk: makró definiáltságát/definiálatlanságát ellenőrzik
- `#if defined(`*azonosító*`)` ≡ `#ifdef` *azonosító*
- `#if !defined(`*azonosító*`)` ≡ `#ifndef` *azonosító*

fej.h

```
#ifndef FEJ
  #define FEJ
  /* érdemi tartalom */
#endif
```

Projekt

datum2.h

```
1  #ifndef DATUM
2  #define DATUM
3
4  #include <string>
5
6  struct datum {
7    int ev, ho, nap;
8  };
9
10 bool szoko(int ev); // szokoev megallapitas

23 // nap even beluli szamabol ho es nap szamolasa
24 datum hoEsNap(int ev, int evNapja);
25 #endif
```

datum2.cpp, ember2.cpp, ember2.h, gepkocsi2.cpp, gepkocsi2.h, emberAuto2.cpp

Sor

Sor (Queue)

- First In First Out (FIFO) adatszerkezet
- Alkalmazása: pl. eltérő késleltetésű, de hasonló adatátviteli sebességű hw. eszközök közötti kommunikációnál, pufferezés
- Megvalósítható pl. egydimenziós tömbbel

sor1.h

```
#ifndef SOR1
#define SOR1

#define MERET 4
bool berak(int adat);
int kivesz();

#endif
```

Sor

sor1.cpp

```
1  #include <iostream>
2  #include "sor1.h"
3
4  static int sor[MERET];
5  static int eleje=0, vege=0, db=0;
6
7  bool berak(int adat) {
8    if(db < MERET) {
9      sor[vege] = adat;
10     db++;
11     vege++;
12     if(vege == MERET) {
13       vege = 0;
14     }
15     return true;
16   } else {
17     std::cerr << "A sor megtelt.\n";
18     return false;
19   }
20 }
```

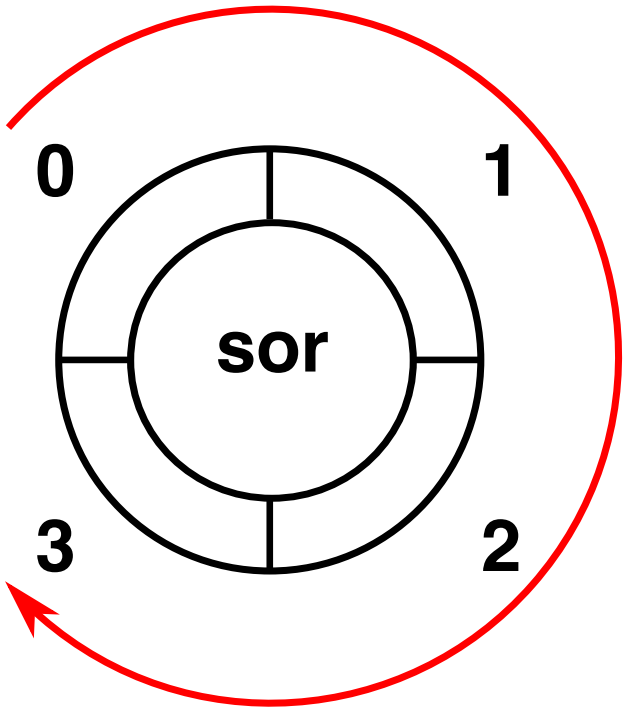
sor1.cpp

```
22  int kivesz() {
23    if(db > 0) {
24      int adat = sor[eleje];
25      db--;
26      eleje++;
27      if(eleje == MERET) {
28        eleje = 0;
29      }
30      return adat;
31    } else {
32      std::cerr << "A sor ures.\n";
33      return 0;
34    }
35  }
```

sorTeszt1.cpp

```
#include <iostream>
#include "sor1.h"
using namespace std;

int main() {
  berak(1); berak(2); berak(3); berak(4);
  berak(5); // nem fer bele
  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;
}
```

Sor

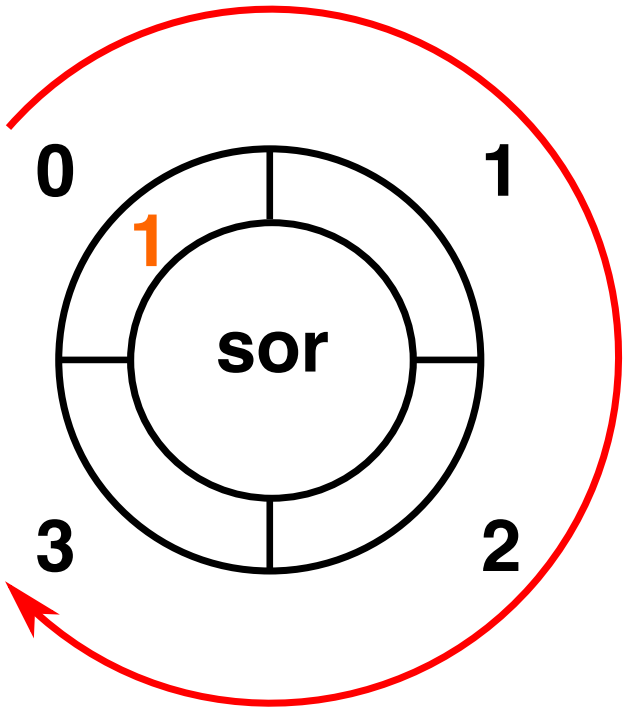


eleje == 0

db == 0

vege == 0

Sor

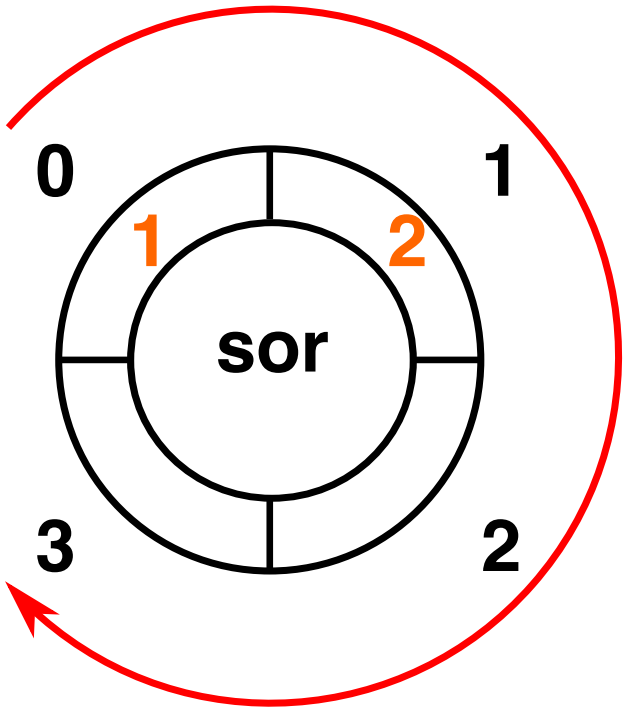


eleje == 0

db == 1

vege == 1

Sor

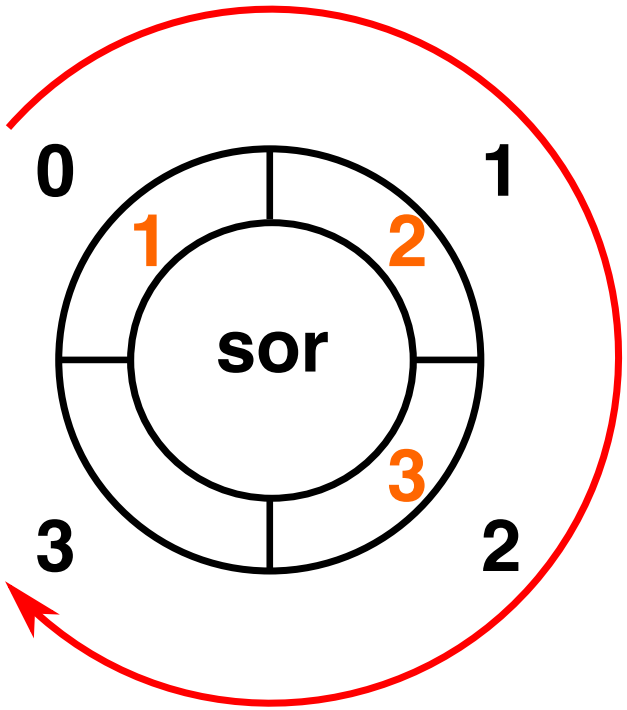


eleje == 0

db == 2

vege == 2

Sor

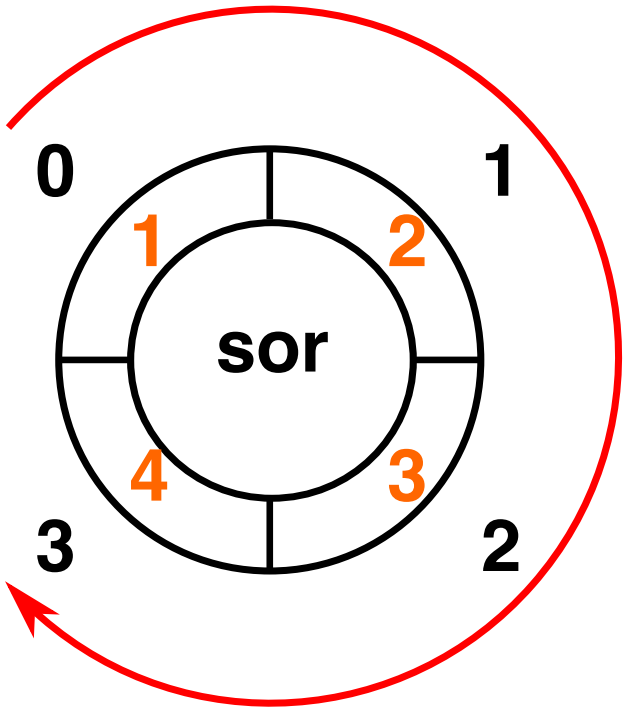


eleje == 0

db == 3

vege == 3

Sor

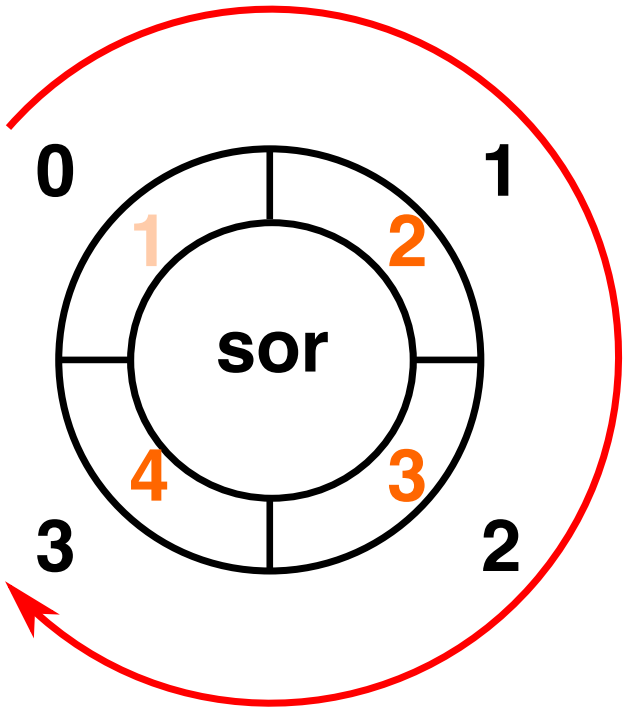


eleje == 0

db == 4

vege == 0

Sor

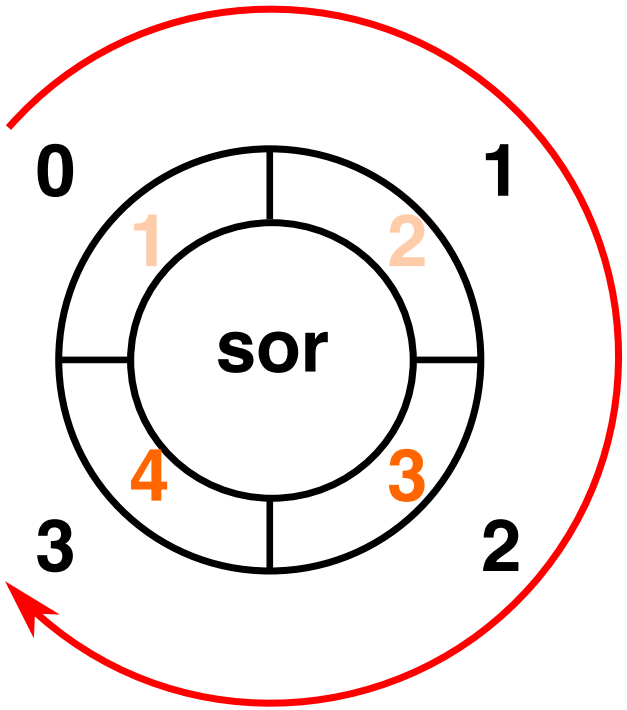


eleje == 1

db == 3

vege == 0

Sor

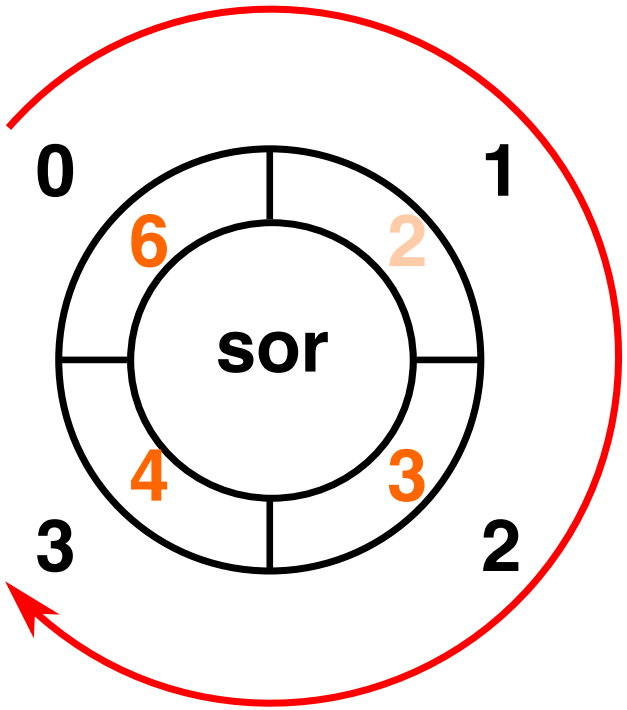


eleje == 2

db == 2

vege == 0

Sor

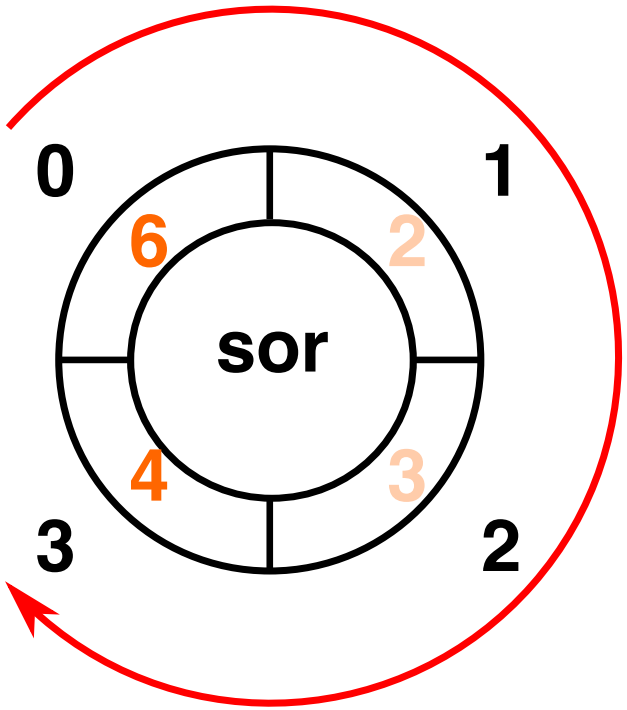


eleje == 2

db == 3

vege == 1

Sor

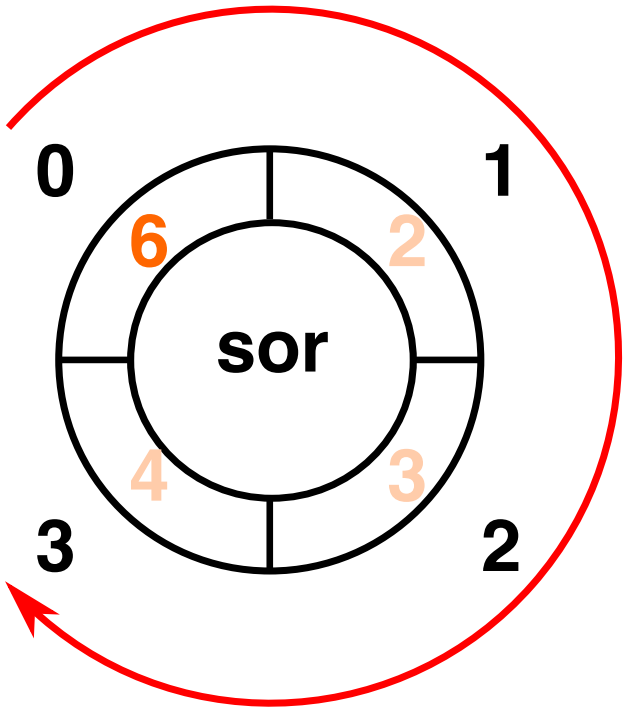


eleje == 3

db == 2

vege == 1

Sor

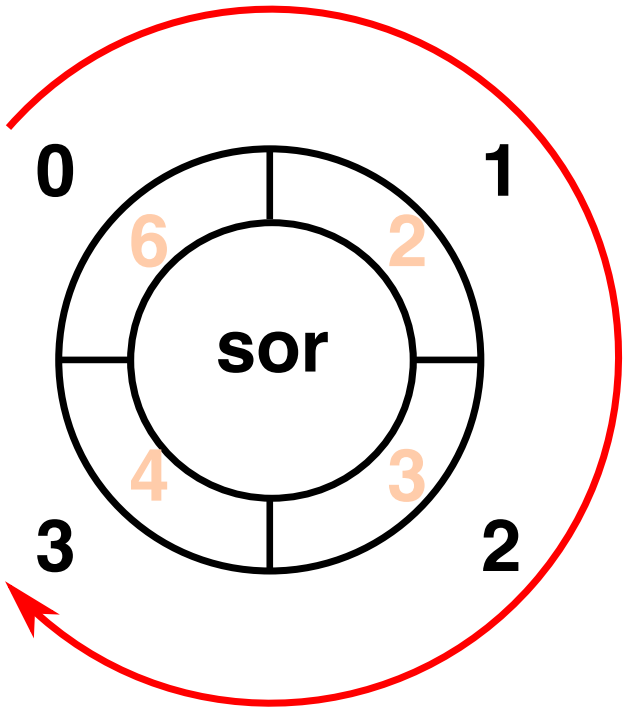


eleje == 0

db == 1

vege == 1

Sor



eleje == 1

db == 0

vege == 1