



A C++ program nyelv operátorai

1. értékadó operátor =

A programozásban az értékadó operátor az egyenlőség jel. Ennek az operátornak a segítségével lehet az adott értéket eltárolni egy változóban. Az operátor 2 operandusú. Az értékadás iránya csak jobbról balra lehetséges. Az operátor bal oldalán lévő változóban tároljuk a jobb oldalon lévő értéket.

- baloldalon csak változó állhat
- jobb oldalon lehet változó és konstans is.

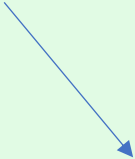
A matekban az $x = 2$ és az $2 = x$ ugyan azt jelenti, azonban a programozásban csak az $x = 2$ alak lehetséges. A változó kezdeti érték adását hívjuk inicializációnak.

2. logikai operátorok

és vagy operátorok működését leginkább **bool tábla** segítségével érthetjük meg.

és operátor jele: `&&`

vagy operátor jele: `||`



A	B	A && B	A B	!A
0	0	0	0	1
1	0	0	1	0
0	1	0	1	1
1	1	1	1	0

A fenti táblázat tartalmazza, hogy az A, B különböző értékeire az ÉS, VAGY operátorok milyen értéket adnak vissza.

pl.:

```
bool a = true;
```

```
bool b = false;
```

```
bool eredmény = a && b;
```

```
cout << "a és b logikai kifejezés kiértékelésének eredménye: " << eredmény << endl;
```

Az ÉS operátornál csak akkor kapunk igaz értéket ha mindkét operandus igaz. A fenti táblázat minden lehetséges esetet tartalmaz.

3. relációs operátor

A relációs operátorok (<,>) segítségével össze lehet hasonlítani 2 számot. A lent látható logikai kifejezést, ha kiértékeljük igaz vagy hamis logikai értéket kapunk. A kiértékelés eredményét eltároltam az eredmény változóban, majd kiírtam a konzol ablakra.

```
int a = 10;
int b = 20;
bool eredmeny = a < b;
cout << "a < b logikai kifejezés kiértékelésének eredménye: " << eredmeny << endl;
```

4. maradék képező operátor

A maradék képező operátor segítségével az osztás után keletkező maradékot eltárolhatjuk egy változóban.

```
int a = 4;
int b = 2;
int maradek = a % b;
cout << "az osztás után keletkező maradék: " << maradek << endl;
```

5. ternary operátor ? :

kifejezés_1 ? kifejezés_2 : kifejezés_3

A feltételes operátornál az 1. logikai kifejezés kiértékelése lehet igaz vagy hamis. Ettől függően végrehajtódik a 2. vagy a 3. kifejezés. Ha a 1. kifejezés igaz, végrehajtódik a 2., ha hamis végrehajtódik a 3. kifejezés helyén lévő programkód.

```
#include <iostream>
using namespace std;
int main()
{
    bool test = false;
    int eredmeny = test ? 10 : 20;
    cout << eredmeny << endl;
    system("pause");
    return 0;
}
```

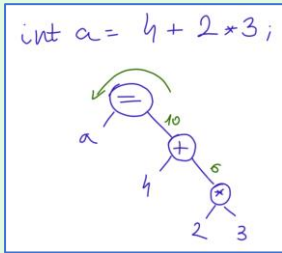
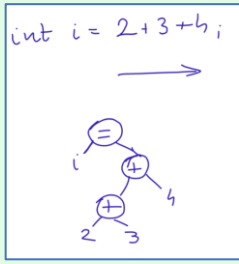
6. operátor precedencia

Az programkódban lévő operátorokat tartalmazó kifejezések kiértékelése csak a helyes sorrendben lehetséges, az operátorok precedenciájának megfelelően. Ezt hívjuk tömören operátor precedenciának. A kiértékelésnél két szempontot kell figyelembe venni: **erősség és kiértékelési irány**. A következő táblázatban az erősség fentről lefelé csökken. A kiértékelés iránya jobbról balra, vagy fordítva lehetséges.

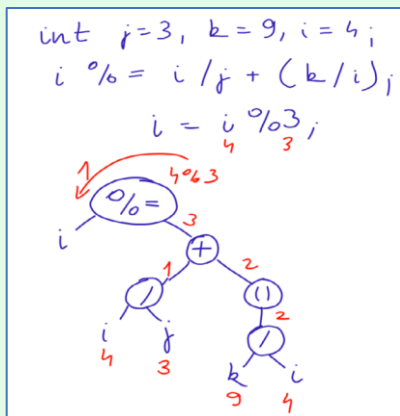
erősség:	név:	jel:
1.	postfix	() [] -> . ++ --
2.	unary prefix	! ~ & sizeof ++ --
3.	multiplicative	* / %
4.	additive	+ -
5.	shift	<< >>
6.	relational	< > <= >=
7.	equality	== !=
8.	and	&&
9.	or	
10.	conditional	? :
11.	assignment	= += -= *= /=
12.	comma	,

Az operátoroknál ++ jel állhat a változó előtt és után. Ha előtte áll pl.: ++i, akkor **prefix** operátornak hívjuk, ha utána áll, pl.: i++ akkor a **postfix** operátornak hívjuk. Az operátor lehet 2 vagy 1 operandusú. Egy operandusú pl.: !a, 2 operandusú pl.: az **a + b**, egyszerűen azért mert 2 változóra hat.

A kiértékelésnél segítséget nyújthat a precedencia fa használata.

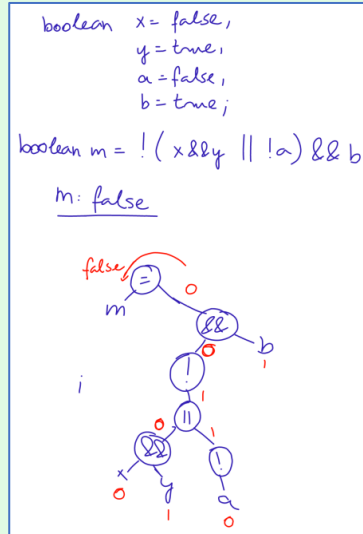
<p>1. feladat</p>  <p>operátor precedencia erősségének sorrendje: * + = A legerősebb precedenciával * rendelkezik, így vele kezdődik a precedencia fa alul.</p>	<p>2. feladat</p> <pre>int i = 3; int j, k; j = k = i;</pre> <p>Érték adó operátor kiértékelési iránya:</p>	<p>3. feladat</p>  <p>Az összeadó operátor kiértékelési iránya: balról jobbra</p>
--	---	--

4. feladat



Alulra, a kezdéshez kerül a legerősebb precedencia.

5. feladat: logikai operátor



előforduló operátorok:

() zárójel

&& és

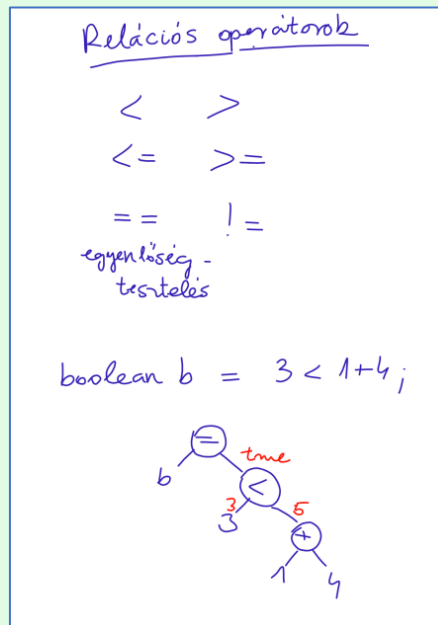
|| vagy

! tagadás

operátorok erőssége: () ! és vagy

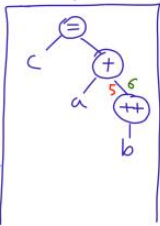
- tagadással kezdünk

6. feladat

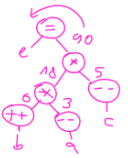


operátorok erőssége: +, <, =

7. feladat: növelő, csökkenő operátor

<p><u>Inkrementálás / dekrementálás</u></p> <p>$i = i + 1;$ $i--;$ $i += 1;$ $i++;$ // post increment $++i;$ // preincrement</p>	<p>$int\ a = 2, b = 5;$ $int\ c = a + b++;$ $int\ c = a + ++b;$</p> <p>1. kiértékelés 2. növelés</p> <p>$c : 7$ $b : 6$</p> <hr/> <p>$c = a + b$ $b = b + 1$</p>  <p>1. növelés 2. kiértékelés</p> <p>$b : 6$ $c : 8$</p> <hr/> <p>$b = b + 1$ $c = a + b$</p>
	pre
	post

8. feladat: növelő, csökkenő operátor

<p>$int\ a = 3, b = 5, c = 6;$ $int\ d = ++a + ++b / --c;$ $int\ e = b++ * --a * c--;$</p> <p>$a = ?$ 3 $b = ?$ 7 $c = ?$ 5 $d = ?$ 5 $e = ?$ 30</p> 	<p>$int\ d = ++a + ++b / --c$ $d = 5$ $int\ d = ++a + ++b / c --$ $d = 5$ (c--) függvényként működve visszatér a kifejezés eredeti értékével, az pedig 6</p> <p>$int / int = int$ (implicit módon) $int\ d = ++a + ++b / --c$ $d = 5$ $int\ d = a++ + ++b / --c$ (itt 3-mat ad hozzá) $d = 4$</p>
--	---


9. Feladat

```
int a=4, b=0;
int c1=-a + b;
cout<<"c1: "<<c1<<endl;
int c2=a-- + b;
cout<<"c2: "<<c2<<endl;
```

//itt nincs hatása post és pre esetnek

```
int eredmény1 = f1();
cout << "e1: " << eredmény1 << endl;
int eredmény2 = f2();
cout << "e2: " << eredmény2 << endl;
//így meg van!!
```

10. feladat

<p>Háromlábú (ternary) op. ? :</p> <pre>int a=2, b=3; int c = ++a == b ? a+b : a-b;</pre> <p>a: 2 3 b: 3 c: <u>6</u></p> 	<p>++a == b true sorrendje: ++, ==</p> <p>++a pre fix a=3 //a először magnövelődik, ezután van érték vizsgálat</p> <p>c = 6;</p>
--	--