# 4. előadás

## Vizsga időpontok

Nem decimális számrendszerben lesz megadva. Segítség: Ugyan azon a napon van mind. (csütörtök)

014. 0x12. 14:00 a nyelvi laborban (Fordítás: 12. 18.)

01. 010. 14:00 a Lovardában (Fordítás: 01. 8.)

01. 0xF. 14:00 a Lovardában (Fordítás: 01. 15.)

01. 0x16. 14:00 a Lovardában (Fordítás: 01. 22.)

UV: 01. 035. a Lovardában, kezdés vagy 13. vagy 14. Majd kiderül. (Fordítás: 01. 29.)

### Számrendszerek

Teljesen meg van lepődve mindenki, hogy nem tanultuk eddig.

Hexadecimális: Tizenhatos számrendszer $166=0xA6$

Oktális: Nyolcas számrendszer $16=020$

## Lebegőpontos konstans literálok

1.4 típusa: double (dupla lebegőpontos számábrázolás)

Van float is, ha kisebbet szeretnénk.

**sizeof(**float**)<=sizeof(**double**)<=sizeof(**long double**)**

Egység: **sizeof(**char**)===**e

Ennél nagyobb vagy egyenlő az int.

long double lebegőpontos: 1.39L

1.1f float lebegőpontos konstans

43e-1 a tízes alapú exponenciális: $43\*10^{-1}=4.3$

## Karakter konstansok

'a'**,** 'T'**,** '\n'**:** char

Előjelessége nem definiált. Gépek egy része így, másik része úgy kezelte. Ezért nem ragaszkodtak egyikhez sem. Ha meg szeretnénk határozni:

signed char

unsigned char

Ismétlés: **sizeof(**T**)==sizeof(**unsigned T**)==sizeof(**signed T**)**

## Boolean

**sizeof(**bool**)<=sizeof(**long**)**

Oka: Későn vezették be a bool-t.

Ez valamennyire pazarló: Egy darab logikai értékért 32 vagy 64 bitet használunk fel.

**true==**1

**false==**0

//Minden más érték **true** (**!**0 **->** **true**)

# Operátorok és kifejezések

## Operátor

Pl.: **+,** **-,** **\***

Az értékadás (a**=**b**;**) is operátor. Értéke $b$. C volt az első nyelv, ami operátorként kezelte az értékadást. Ennek sok előnye van, de egy hátránya: **if(**a**=**b**){**...**}** vs **if(**a**==**b**){**...**}**

## Kifejezés

Pl.: 3 **+** x

Operátorok és operandusok (konstansok, változónevek vagy azonosítók).

Kifejezésnek is van típusa és értéke.

Az érték futás közben derül ki, míg konstansról fordítás alatt is tudjuk.

Típusa viszont fordítás során ismert, mert erősen típusos nyelv.

Pl.: A 3 **+** x kifejezés típusa x típusától függ. Ha $x$ int, akkor $3+x$ is int. Ha $x$ double, akkor a $3$ is double lesz, konverzió szabályok miatt. És lebegőpontos összeadás hajtódik végre.

Nincs külön egész és lebegőpontos osztás.
6**/**4**:** int
7.2**/**2**:** float

### Inkrementálás, ++

a**++** **!=** **++**a

a**++** eredménye a megnövelés előtti érték.

**++**a hatékonyabb, de valami miatt ritkán használjuk.

**for(**int i**=**0**;**i**<**N**;++**i**){**...**}** és **for(**int i**=**0**;**i**<**N**;**i**++){**...**}** között nincs különbség, csak annyi, hogy az i**++** lassabb. (Az i**++** fölöslegesen kiszámol, majd eldob egy értéket.)

v**[**i**++]=**0 és v**[++**i**]=**0 között már van különbség.

## Precedencia, kiértékelési sorrend

a **+** b **+** 5 **\*** x **+** 3

Legmagasabb szinten például a névfeloldási operátor ($ : :$ vagyis a scope operátor), majd a metódushívás (pont operátor, $.$)

Ha nem vagyunk benne biztosak, nyugodtan zárójelezzük be: a **+** b **+** **(**5 **\*** x**)** **+** 3

Legutolsó a vessző operátor.

## Kötés

Az a**=**b-nek van értéke. Ezt lehet bővíteni: s**=**x**=**a**=**b és bővíteni: s**=**x**=**a**=**b.

Az értékadás jobbról köt: s**=(**x**=(**a**=**b**))**.

A szorzás balról köt: a **\*** b **\*** c ⇔ **(**a **\*** b**)** **\*** c

## Szokásos aritmetikai konverziók (Usual arithmetic conversions)

a **\*** b

Ha $a$ és $b$ típusa különbözik…

Ha az egyik lebegőpontos (pl.: long double), akkor a másik is long double-lé konvertálódik, és az eredmény is long double lesz.

Ha nincs ilyen típusú operandus, de volt egy long double, akkor… Ezt inkább copy paste MSDN-ről:

1. If either operand is of type long double, the other operand is converted to type long double.
2. If the above condition is not met and either operand is of type double, the other operand is converted to type double.
3. If the above two conditions are not met and either operand is of type float, the other operand is converted to type float.
4. If the above three conditions are not met (none of the operands are of floating types), then integral conversions are performed on the operands as follows:
	1. If either operand is of type unsigned long, the other operand is converted to type unsigned long.
	2. If the above condition is not met and either operand is of type long and the other of type unsigned int, both operands are converted to type unsigned long.
	3. If the above two conditions are not met, and either operand is of type long, the other operand is converted to type long.
	4. If the above three conditions are not met, and either operand is of type unsigned int, the other operand is converted to type unsigned int.
	5. If none of the above conditions are met, both operands are converted to type int.

Bár van egy kis különbség a tábla és MSDN között… De ezt úgysem érdemes megtanulni. Én inkább mindig explicit meghatározom, mi konvertálódjon mivé. De ezt a C-s castolást a tanár nem szereti: **(**double**)(**4**)** Változó castolása: **static\_cast<**double**>(**var**)**

Ez nincs is mindig összhangban a matematikai szabályrendszereinkkel. Például int és unsigned int változók közötti összehasonlítás esetén sajnos az int konvertálódik unsigned int-té.

## Logikai operátorok

**&&,** **||**

Lusta (rövidzáras) kiértékelésűek: Csak akkor számolja ki, ha muszáj.

Ha x**<**4, akkor az **if(**x**<**4**||**y**>**6**)** kifejezés második része nem hajtódik végre. (Igaz vagy bármi = igaz)

Ha x**≥**4, akkor az **if(**x**<**4**&&**y**>**6**)** kifejezés második része nem hajtódik végre. (Hamis és bármi = hamis)

JAVA-ban ez az alapértelmezett viselkedés. De ott a | és & operátorok különbözőek. C-ben bitenkénti és/vagy, JAVA-ban (…).