C programozás

4 óra

Operátorok és mutatók

1. Előző óra, rövid összefoglalás

értékadás:

objektum = érték; =értékadás operátora

(kifejezés, változó)

a bal oldal azt az objektumot jelöli ki (címzi meg) a memóriában, ahova a jobb oldalon megadott kifejezés értékét be kell tölteni

balérték (lvalue), illetve jobbérték (rvalue)

Operátorok

- Aritmetikai operátorok (+ - * / %)
- Relációs operátorok (< <= > >= == !=)
- Logikai operátorok (&& || !)
- Bitenkénti operátorok (& | ^ ~)
- Léptető operátorok (>> <<)
- Értékadó operátorok (= += *= . . .)
- Feltételes operátor (?:)

Aritmetikai operátorok

\+ - * / %

/ - egészek között: egészrész (5 / 2 ⇒ 2)

% - maradék képzés (5 % 2 ⇒ 1)

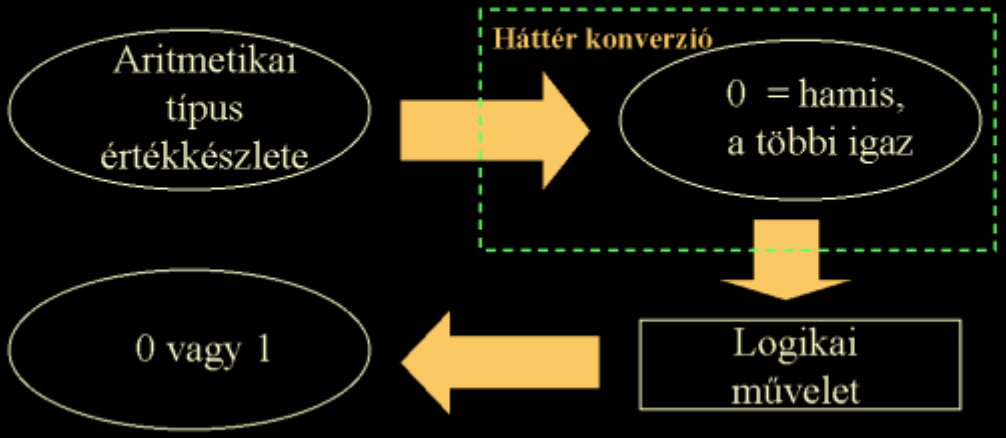
Relációs operátorok

<	kisebb
<=	kisebb vagy egyenlő
>	nagyobb
>=	nagyobb vagy egyenlő
==	egyenlő
!=	nem egyenlő

Eredményük 0 v. 1
hamis v. igaz

Logikai operátorok

Nincs logikai típus, de van logikai operátor.



Logikai operátorok (2)

Logikai vagy:

kifejezés1 **||** *kifejezés2*

Sorrend !!

Példák: a < 3 || a > 5

c > 3 || a > 5 **;** if (a < 3 || a > 5) c = 8 **;**

Logikai és:

kifejezés1 **&&** *kifejezés2*

Logikai tagadás:

! *kifejezés*

Sorrend !!

– logikai műveletek:

```
int a = 5, b = 2, c, d;
...
c = (a < b);   /* hamis, tehát 0 lesz */
d = !c && a;   /* 1, mert !c igaz és a is
                  igaz */
```

Nincs logikai típus, de vannak logikai műveletek!

Értékadó operátorok

- Az értékadás is kifejezés ⇒ többszörös értékadás. Pl: a = b = c = 12
- Minden kétoperandusú operátor kombinálható az értékadással.
- Rövidebb írásmód. Egyszerű optimalizálási lehetőség.

pl: a += 3 ⇒ a = a + 3
a *= c ⇒ a = a * c
x /= y ⇒ x = x / y

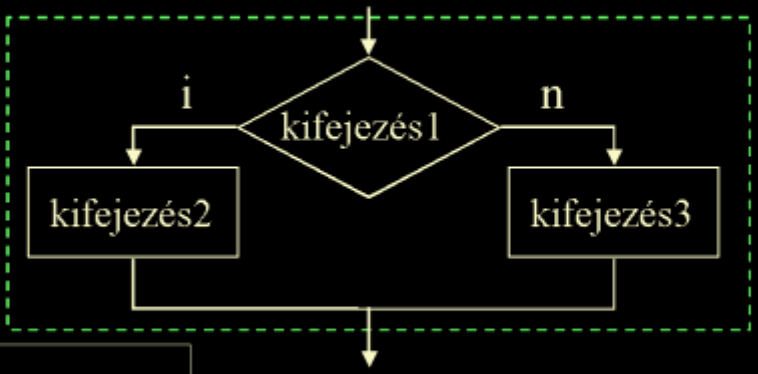
Balérték (lvalue)

- Olyan érték (kifejezés), ami értékadás bal oldalán szerepelhet.
- Az operátorok egy része balértéket igényel (pl. értékadó op.), de van olyan operátor is, ami balértéket állít elő.
- Legegyszerűbb balérték a változó neve.

Feltételes operátor (?:)

kifejezés1 **?** *kifejezés2* **:** *kifejezés3*

Sorrend !!



Példák:

a = a > b **?** a **:** b

y = x == 3 ? x - 2 : z - 8

++ és -- operátor

`++` inkrementáló (növelés eggyel)

`--` dekrementáló (csökkentés eggyel)

Prefix `++i` `--i` és postfix `i++` `i--` forma

kifejezés (i = 5)	kif. értéke	i értéke
`i++`	5	6
`i--`	5	4
`++i`	6	6
`--i`	4	4

A különbség oka: a ***fő hatás*** és a ***mellékhatás*** bekövetkezésének különböző sorrendje.

++ és -- operátor (2)

Prefix forma:

```
int j, i = 2;
j = ++i;
```

1. ***Mellékhatás:*** `i` értékét 1-gyel megnöveljük
2. ***Fő hatás:*** kiértékeljük az operandust: `i==3`, ez lesz a `++i` kifejezés értéke
3. `j`-be a kiértékelt jobbérték kifejezés értékét, a 3-at másoljuk (értékadó operátor mellékhatása)

++ és -- operátor (3)

Postfix forma:

```
int j, i = 2;
j = i++;
```

1. ***Fő hatás:*** kiértékeljük az operandust: **`i==2`**, ez lesz az **`i++`** kifejezés értéke
2. ***Mellékhatás:*** **`i`** értékét 1-gyel megnöveljük
3. **`j`**-be a kiértékelt jobbérték kifejezés értékét, a 2-t másoljuk (értékadó operátor mellékhatása)

++ és -- operátor (4)

Mikor melyik formát használjuk?

- A leggyakrabban csak az inkrementálás vagy dekrementálás miatt használjuk, a kiértékelés eredményét el is dobjuk - lásd a példát:

```
db++;
```

Itt tehát írhattuk volna azt is, hogy

```
++db;
```

- Ha összetett kifejezés része, nem mindegy! Ez egyébként illetlenség, mert nem átlátható a programkód működése.

Típuskonverziók

- egészek konverziója

  `int = char` vagy

  `long = int` értékadás - mindig OK

  `char = int short = int  int = long` - OK, ha belefér

  `signed = unsigned` vagy **`unsigned = signed`**

  VIGYÁZNI ! - előjel bit

- egész ⇔ lebegőpontos konverziók

  `int = double` - OK, ha belefér, csonkolás

  `double = int` - mindig megy

Típuskonverziók (2)

- lebegőpontos konverzió
  `double = float` - mindig OK
  `float = double` - OK, ha belefér
- aritmetikai konverziók
  - egészeknél: a legnagyobb ábrázolású egészre kiterjesztjük az operandusokat, az eredmény a legnagyobb értelmezésű operandus típusával fog megegyezni
  - valósaknál: a műveleteket mindig `double`-ban végezzük

Típuskonverziók (3)

- Használjuk az explicit típusmódosító operátort; ez a C++ esetén majd igen hasznos lesz. Pl.:

```
int a = 1, b = 2;
double x;
...
x = (double)(a);   /* 1.0 lesz */
x = (double)(a)/(double)(b)  /* 0.5 */
```

- Általában:

```
(új_típus)(régi_típusú_kifejezés)
```

<u>Egy-operandusú konverziók</u> (unary conversions):

Pl. értékadásnál: **ebbe = ezt;**

aktuális paraméter-átadásnál:

```
/* Függvény definíciója: */
        valami Fuggv (xxxx)
        {          ....
        }
        ...
        Fuggv (ezt);
```

Miből	Mibe	Eredmény
char, short	**int**	mindig, minden művelet előtt
rövidebb **int** pl. **short** **int**	hosszabb **int** **int** **long**	O.K.
rövidebb **unsigned** pl. **unsigned short** **unsigned**	hosszabb **unsigned** **unsigned** **unsigned long**	O.K.
hosszabb **int**	rövidebb **int**	túlcsordulhat (felső bitek elvesznek)
hosszabb **unsigned**	rövidebb **unsigned**	túlcsordulhat (felső bitek elvesznek)

rövidebb **+ int**	**unsigned**	O.K.
- int	**unsigned**	modulo 2^n, azaz **ebbe = ezt + 2^n** (**> ~_MAX** !!!)
unsigned	ugyanakkora **int**	túlcsordulhat
unsigned	hosszabb **int**	O.K.

Lebegőpontosból lebegőpontosba: van túlcsordulás-jelzés

Miből	**Mibe**	**Eredmény**
rövidebb lebegőpontos pl.: **float** **double**	hosszabb lebegőpontos **double** **long double**	O.K.
hosszabb	rövidebb	túlcsordulhat, pontosság csökkenhet

Egész - lebegőpontos között: van túlcsordulás-jelzés

Miből	Mibe	Eredmény
egész	lebegőpontos	ha lehet, a pontos érték megtartásával, ha nem: a legközelebbi két érték egyikére, ha nem lehet: túlcsordul
lebegőpontos	egész	törtrész elhagyásával, túlcsordulhat

<u>Két-operandusú konverziók</u> (binary conversions)

Két-operandusú operátoroknál, pl.

```
int      x;
unsigned u;
long     L;
x = ...;  u = ...;   L = x+u;
```

Itt az összeadáshoz két-operandusú, az értékadáshoz egy-op. konverzió kell.

A konverzók sorrendje rendre:

0. Ha egyik tömb, vagy függvény, akkor mutatóvá konvertálódik, és nincs további konverzió
1. (Csak aritmetikai típusúak lehetnek)
   Egy-operandusú konverzió a hossz növelésére
2. A két operandus azonos típussá alakítása a következő sorrend szerint:

Egyik operandus	**Másik operandus**	**Közös, új típus**
long double	bármi	**long double**
double	bármi	**double**
float	bármi	**float**
unsigned long	bármi	**unsigned long**
long	bármi (**int, unsigned**)	**long**
unsigned	bármi (**int**)	**unsigned**
int	bármi (**int**)	**int**

(Mint már tudjuk, short ill. char típusú értéket előbb mindig int típusúvá alakítja, azzal számol.)