Olvasólecke az **Informatika a fizikában 2.** kurzushoz	Szerző: **Majorosi Szilárd** (TTIK Elméleti Fizikai Tanszék)	Olvasási idő: **40-45 perc**.

ELEMI ADATTÍPUSOK ÉS VÁLTOZÓK

A C nyelv vezérlésének logikai építőkövei (az előfordítót nem számolva) a **változók** (adatok) és a **függvények** (operátorok). Saját magunk is definiálhatjuk őket.

- Változó például: `double x;`
- Függvény például: `int main(){}`

A változozók olyan dinamikusan írható-és-olvasható adatok a memóriában, amelyekre saját nevükkel hivatkozhatunk és műveleteket végezhetünk velük. A C nyelvben egy változóhoz szervesen hozzátartozik annak **típusa**, amely definiálja nemcsak annak memóriában elfoglalt méretét, hanem annak érték szerinti interpretációját és a rajta végezhető lehetséges műveleteket.

A függvények utasításokat tárolnak, amelyeket egymás után hajtanak végre; a függvények és utasításaik is a memóriában megtalálhatók, azonban C programozás során nem játszanak szerepet.[1]

1. Elemi adattípusok és ábrázolásuk

Az adatok tárolása a számítógép memóriájában kettes számrendszerben (bináris sorozatként) történik. *Ez hardvertől függő*, például memória fizikai felépítésétől (regiszterek mérete) és a processzor architektúrájától. Szerencsére napjainkban a számítógépek x86 vagy x64 architektúrát követik, így ezek viselkedése többé kevésbé standardizált (a lenntiekben ezekre utal majd az „általában” kifejezés). Ezenkívül a C fordító megválasztásától is függ a típusméret.

A C nyelvben *az egyes elemi típusokhoz speciális kulcsszavak tartoznak.* A következőekben sorra vesszük ezeket az elemi típusokat, és használatukat.

1.1. int - egész szám

A `int` kulcsszó a C nyelvben **egész számokhoz** tartozó típust jelöl. Erről fontos tudni, hogy a *memóriában történő ábrázolásuk egzakt*. Napjainkban használt számítógépek esetén általában 4 bájt méretűek (de minimum 2). Nézzünk példát, hogy ez mit is jelent:

- 4 bájt = 4×8 bit = 32 bit = 1 előjelbit + 31 számjegy bit.
  Például (2 bájt): 0 000000000100101 bináris kód: $1\cdot 2^5+1\cdot 2^2+1\cdot 2^0 = 32+4+1 = 37$.
- Lehetőségek száma: $2^{32} = 4294967296$, értékkészlet $-2147483648 \ldots 2147483647$.

[1] Rosszindulatú alkalmazások viszont igyekeznek ezt kihasználni: ha hozzáférnek a függvény memóriaterületéhez, akkor képesek oda nem illő utasításokat beírni.

A fenti példában szereplő konkrét bináris kód a C nyelv részét nem képezi, a bitben megadott méret a legnagyobb nagyságú ábrázolható egész szám értékét korlátozza.[2] Ha nem fér el egy egész szám az `int`-hez tartozó memóriába (fennt ez 4 bájt), akkor fellép az ún. *túlcsordulás* (overflow), ami hibás működést eredményez. Ez főleg egész számok szorzásánál okozhat gondot.

1.2. float és double - valós számok

Bár a nem nehéz elképzelni, hogy a fenti egész számokkal is lehetséges korlátozottan valós számokat is reprezentálni[3], azonban a tudományos közösség más megoldás mellett döntött, amelyeket elnevezték „lebegőpontos" valós számoknak és az IEEE 754 szabványban (→ link) rögzítettek.

A „lebegőpontos" valós szám a C nyelvben *az igazi* **valós számok** *közelítő reprezentációja*, méghozzá úgy, hogy a memória kódolás a *valós szám normálalakjában* történik: van egy véges pontosságú decimális szám (mantissza) szorozva 2 valamelyik hatványával (exponens) - amelynek eredményeképp a decimális rész tizedespontja több nagyságrendet „lebeghet". A C nyelvben két kulcsszó is jelöli őket, `float` és `double`. Napjainkban a következőt jelentik:

- A `float` (4 bájt) relatív 7 tizedesjegy pontos és az elérhető exponens $10^{\pm 38}$.
- A `double` (8 bájt) relatív 15 tizedesjegy pontos és az elérhető exponens $10^{\pm 308}$.
- Példák C-ben: `5., 0.00007452, -972624.131`. Ezenkívül megadhatók exponenciális jelöléssel is: `-1.142423e-28, 34.142423e+5`, (azaz $-1.142423 \cdot 10^{-28}$, $34.142423 \cdot 10^{5}$).

Előnyük, hogy ilyen valós számok szorzása és osztása során sokkal nehezebb kilépni az értékkészletükből. Hátrányuk, hogy a tizedesjegyben *megadott pontosságnál mindig kerekíteni kell*. Manapság általában a `double`-t használják – ekkor a nagy precíziót szükségelő számításokat kivéve, úgy tekintjük őket mintha igazi valós számok lennének.

1.3. char - karakterek

A `char` típus *1 byte méretű előjeles egész számot* jelöl (értékei: -128...0...127). Az ilyen egész számok **karakterek** kódolására szolgálnak, az interpretációhoz ASCII kódrendszert (→ link) használva. Ez lefedi *az angol betűket, a számokat, az alap írásjeleket és matematikai műveleteket* (összesen 128 karakter, a nemnegatív értékekre). A karakterek sorrendje többnyire a logikai sorrendet követi.[4]

- Példák karakterekre C-ben: `'a', 'z', 'F', 'K' , '0', '7' , '*', '+', ' ', '/'`.
- Néhány **speciális karakter** (escape karakter): `'\n'` új sor, `'\t'` tabulátor, `'\\'` visszafele per jel (backslash), `'\"'` idézőjel...

[2] A C nyelvben a méretek megadásánál nincsenek bitek, csak bájtok. A memóriaterületet sem binárisan, hanem hexadecimális számrendszerben szokás kiírni. Ennek ellenére vannak a C nyelvben olyan műveletek, amelyek a memóriabeli ábrázolás bitjeire hatnak (bitwise operators).

[3] Ha az egész szám egyik fix tizedesjegyéhez tizedespontot rendelünk, akkor az már valós számot ír le közelítőleg. Más programnyelvekben van ilyenre példa, ezeket decimális valós számoknak hívják.

[4] A többi 128 karakter más kódtáblázatok szerint különböző karaktereket reprezentálhat. Ilyen kódtáblázatok, amelyek magyar karaktereket is tartalmaznak UTF-8 és Latin-2 (ANSI ISO/IEC 8859-2). Sajnos arra nem lehet számítani általánosan, hogy a C fordító is így interpretálja az ékezetes karaktereket. Itt megjegyzendő, hogy UTF-8 karakterkódolás esetén 1 karakter több bájt hosszú is lehet.

- 0 értékű **char** típushoz nem a '0' karakter tartozik, hanem egy speciális '\0' karakter.

A karakterek nem csak önmagukban fordulnak elő, C kódban az *idézőjelek között lévő kifejezések (karakterláncok) elemeit is ezek alkotják*. Ezenkívül, mivel ezek egész számok is, így azoknak megfelelő műveletek is végezhetők rajtuk, de ekkor nagyon kell figyelni túlcsordulás által okozott hibákra.

1.4. Az elemi adattípusok módosítása – néhány példa

A C nyelvben vannak további kulcsszavak, amelyek a fenti három alaptípust képesek módosítani, ha *egy elemi típus elé írjuk őket*. Ilyen az **unsigned** kulcsszó is: ez úgy módosíthatja az utána írt egész számokat jelölő típusokat, hogy teljes memóriája nemnegatív egész számokat reprezentáljon. A **long** kulcsszó azt jelzi a fordítónak, hogy az utána írt elemi típus memóriamérete legyen a szokásosnál hosszabb vagy azzal egyenlő. Akár kettő is lehet belőle. Példák:

- **unsigned int** – előjel nélküli egész (legalább 2 bájtos, általában 4)
  (az **int** nemnegatív értékei a memóriában ugyanazok mint az **unsigned int**-é)
- **long int** – hosszú egész (legalább 4 bájtos)
- **unsigned long int** – hosszú előjel nélküli egész (legalább 4 bájtos)
- **long long int** – nagyon hosszú egész (legalább 8 bájtos)

It található teljes lista C nyelv típusairól: → link.

Mivel a típusunk (vagy változónk) *memóriában lefoglalt méretében* általában nem lehetünk biztosak, ezért a C nyelvben ennek *lekérdezésére* rendelkezésre áll egy nyelvi elem, a **sizeof**() operátor:

```
sizeof(típus/változó)
```

Ez az operátor **int**-ként megadja a **sizeof()** argumentumában szereplő típus vagy változó méretét bájtban.

2. Változók a C nyelvben

A továbbiakban tekintsük át, hogy a C nyelvben miként hozhatunk létre és használhatunk változókat.

2.1. Változók deklarációja

A C nyelvben minden változó rendelkezik *egy típussal, és egy (általunk választott) névvel*. Az alábbi utasítással létrehozhatunk egy **típus** típusú és nev nevű változót :

```
típus nev;
```

A fenti utasítás a nev nevű és **típus** típusú változó deklarációja: ekkor *kijelentjük, hogy* nev *létezik és memóriát rendelünk hozzá*. Ezután ezt a változót pusztán nev nevére való hivatkozással

használhatjuk. Ez a kijelentés nem társít határozott kezdőértéket az adott változóhoz, azaz annak kezdőértéke *definiálatlan*.[5]

2.2. Változók inicializációja

Amikor *a változóhoz először értéket rendelünk*, azt inicializációnak nevezzük. C-ben a változóknak megfelelő érték adható a = operátorral, ekkor érték típusának egyeznie kell a `nev` típusával, bár elemi típusoknál erre nem mindig kell ügyelni.[6] *Az inicializációt célszerű a deklarációval együtt megtenni*, például a következő utasítással:

```
típus nev = érték;
```

Ekkor a `nev` változónak a kezdőértéke határozottan `érték` lesz.

Amennyiben *egy utasítással több azonos típusú változó deklarációját és inicializációját akarjuk elvégezni*, azt általános esetben így írhatjuk:

```
típus nev1 = érték1, nev2 = érték2, ...;
```

A felsorolást a vessző (,) szintaktikai elem *jelzi*.

Ha olyan változó(ka)t szeretnénk, amelynek a kezdőértéke nem változtatható meg, akkor ezek deklarációjakor a típusa elé írjuk oda a **`const`** kulcsszót (**`const típus`**). Ez gyakorlatilag konstansá teszi a kérdéses változó(ka)t.[7]

2.3. Változók neve – megkötések

A változók elnevezésére érdemes beszédes, nem túl hosszú nevet választani. Ezenkívül vonatkozik rájuk pár szabály:

- Az első karakter betű legyen. Betű alatt az angol abc egy karaktere értendő.
- A többi karakter: betű (kis és nagy számít), szám, _
- Nem lehet benne szóköz, speciális karater: `#, &, %`, ...
- Nem lehetnek a C nyelv kulcsszavai: **`int, double, if, for`** ...

Az adott kódblokkban minden változónak egyedi névvel kell rendelkeznie.

[5] Ha a programozásunk során valamely művelet vagy érték *definiálatlan* az eggyel rosszabb minősítés annál, hogy a művelet eredménye vagy egy érték *helytelen*. Ez úgy lehetséges, hogy a programunk néha helyesen viselkedik és néha nem, azaz a program működése nem determinisztikus. Ez általában akkor okoz súlyos fejtörést, amikor bonyolult programot írunk, és nem értjük, hogy miért ad a program helytelen eredményt.

[6] Az ebben az olvasóleckében tárgyalt típusoknál a fordító az adott számértéket típuskényszerítéssel az elvárt típussá alakítja, kérdezés nélkül. Csak akkor ad hibaüzenetet ha nincs szabványos eljárás az egyik típusról a másikra való konvertálására.

[7] Az egyetemes konstansok beírására C-nyelvben azonban nem ez a bevett eljárás: azokat **`#define`**-al szokták definiálni a fejlécfájlokban. A típusuk az értékükből következik.

1. Forráskód. Példák változók deklarációjára és inicializációjára. Figyeljük meg a szintaxist.

```
1  //Main függvény a belépési pont, röviden írva
2  int main(){
3      //Valós számok deklarációja, aztán inicializáció
4      double var_x, var_y;
5      var_x = -1.5; //tizedespont megléte a számértékben double típusra utal
6      var_y = 2.e+4; //az érték 20000
7
8       //Egész számok többes deklarációja és inicializációja
9      int var_i = 3, var_j = 20, var_k = -300;
10
11     //Konstans karakterek
12     const char a = 'a'; const char b = 'b';
13
14     //Más számtípusok (értékekhez extra betűt fűzhetünk pontosítás céljából)
15     float float_x = -1.5f;  //f az jelzi, hogy az érték típusa float
16     float float_i  = var_i;  //egész szám float-ra konvertálódik
17     unsigned long int u_long_l = 3287483647UL; //UL: a számérték unsigned long
18     long long int longlong_l   = 3287483647LL; //LL: a számérték long long
19
20     //Méret lekérdezése
21     int meret_char = sizeof(char); //típus mérete
22     int meret_var_x =    sizeof(var_x); //változó mérete
23 }
```

2.4. printf() függvény – formázott kiíratás

Azért, hogy elkezdhessünk C-ben változókat használva programokat írni, szükség lesz arra, hogy az egyes utasítások eredményét megjelenítsük. Ennek a legalapvetőbb módja a változók értékeinek konzolra történő kiíratása. Ehhez szükségünk lesz az **<stdio.h>**-ban lévő `printf()` függvényre. A kiíratáshoz a következő általános alakú utasítást írjuk:

```
printf("karakter sorozat", ...);
```

Az idézőjelek között lévő karakter sorozatot kiírja a konzolra (standard kimenetre). Ez az utasítás egy olyan függvényhívás, amelynek *változó a lehetséges argumentumainak a száma.* Ez azt jelenti, hogy a ... helyére további változók neveit (vagy számértékeket) írhatunk ,-vel elválasztva. Ha a ... helyére nem írunk semmit, akkor az előtte lévő vesszőt is töröljük.

Ezzel az utasítással *változók formázott kiíratása* is megvalósítható, aminek a használata sajátos szabályokat követ:

- Ha **%** jel van az első argumentum karakter sorozatában, *amelyet megfelelő helyettesítő kifejezés követ*, akkor sorban a többi argumentumhoz tartozó memóriaterületről kiolvassa az adott típust, és a **%**-os kifejezés helyére formázva kiírja azt a konzolra.
- Az argumokként megadott változók helyességét a `printf()` *nem ellenőrzi*: ha az első argumentumban lévő **%**-os kifejezések elvárt típusa és az argumentumokként beírt további változók típusa és sorrendje nem egyezik meg, akkor a kiíratás eredménye *definiálatlan* lehet.

Több beolvasásra és kiíratásra szolgáló függvény C-ben ilyen vagy ehhez hasonló szabályokat követ.

printf() és társai – helyettesítő kifejezések

A fennt említett **%**-os kifejezéseket helyettesítő kifejezéseknek hívjuk. Ezek egy rövid listája azzal, hogy milyen típusú változók kiíratására vonatkoznak:

- **%c** – `char` (karakter) – kiíratásnál int is lehet
- **%s** – `const char*` (ez egy másik ”karakter sorozat”)
- **%d, %i** – `int` (egész szám) – kiíratásnál char is lehet
- **%ld, %li** – `long int` (hosszú egész szám)
- **%f, %e, %g** – `float` (valós szám) – kiíratásnál double is lehet –
  decimális formában, normálalakban, rövid formában
- **%lf, %le, %lg** – `double` (hosszú valós szám)
- **%%** – %, stb.

Ezek még egyéb szintaxis alapján pontosíthatók, hogy formázott kiíratás valósuljon meg, például: `"%+7.3lf"` – az előjelet mindig kiírva (+), legalább 7 karakter széles mezőben (7), 3 tizedesjegy pontossággal (.3), hosszú (l), valós számot decimálisan (f).

A teljes lista megtalálható itt: → link.

2. Forráskód. Példák változók konzolra történő kiírására.

```c
#include <stdio.h>  //Kiíratáshoz kell
int main(){
    //Kiíratott változók
    double var_x = -1.5, var_y = 2.e+4;
    int var_i = 3, var_j = 20, var_k = -300;
    const char a = 'a', b = 'b';
    float float_x = -1.5f;
    unsigned long int ulong_l   = 3287483647UL;

   //Egesz számok decimálisan
   printf(" i = %d; j = %d; k = %d; ", var_i, var_j, var_k);
   //Karakter kiírása
   printf(" %c; %c;\n", a, b);
   //Valós számok - decimálisan
   printf(" x = %lf; y = %lf  (decimalis);\n", var_x, var_y);
   //Valós számok - formázott exponenciális
   printf(" x = %10.6le; y = %10.6le  (exponencialis);\n", var_x, var_y);
   //Egyeb számok
   printf(" float_x = %f; ulong_l = %lu;", float_x, u_long_l);
}
```

```
Kimenet:  i = 3; j = 20; k = -300; a; b;
Kimenet:  x = -1.500000; y = 20000.000000 (decimalis);
Kimenet:  x = -1.500000e+000; y = 2.000000e+004 (exponencialis);
Kimenet:  float_x = -1.500000; ulong_l = 3287483647;
```

3. Algebrai operátorok (műveletek)

A következőekben tekintsük át az alapvető matematikai és logikai számolások elvégzésését szolgáló műveleteket, és a hozzájuk kapcsolódó szintaktikai elemeket (operátorokat).

3.1. Aritmetikai műveletek

Aritmetikai műveletek alatt a **4 alapműveletet és az osztás maradékát** értjük (jele %, csak egész számokra működik). Hozzájuk tartozó operátorok (szimbólumok):

$+, \quad -, \quad *, \quad /$ és $\%$

Ezek a műveletek 2 argumentumot igényelnek (változó nevét, vagy értéket), egyet az operátor bal, a másikat a jobb oldalára írva. A program futásakor az adott operátor által definiált művelet eredményét fogja visszahelyettesíteni az adott helyre – ezt a szokásos matematekai szabályokat követve történik. A műveletek elvégzésének a sorrendje és a (kerek) zárójelezés a szokott matematikai logikát követi. *A helyzetet viszont bonyolítja, hogy ezen műveletek eredménye függ az argumentumok típusától.*

Vigyázat: Két egész szám hányadosa mindig egész szám lesz!

Például: $5/2 \rightarrow 2$ és nem 2.5. Viszont $5\%2 \rightarrow 1$.

3.2. Típus kényszerítés (type casting)

A C nyelvben egy rendkívül fontos művelet az úgynevezett **típus kényszerítés** (type casting). Ezen művelethez tartozó operátor szintaxisa a következő:

`(újtípus)` változó/érték;

Ez a művelet egy adott változó vagy számértéknek `újtípus`-ba konvertálását végzi el. *Algebrai szempontból valós számok és egész számok között konvertálhatunk vele*: ilyenkor általában lekerekítés történik. A matematikai zárójelezéssel is keverhető, ekkor a kettőt csak a szintaxisa különbözteti meg. Használata nem elemi típusokra is működik, de körültekintően szükséges ekkor eljárni. [8]

Típuskonverzió C programozás során kétféle módon szerepelhet. Amennyiben a fenti utasítás a kódban is közvetlenül ki van írva, akkor a típus kényszerítés láthatósága *explicit*. Azonban a fordító a C nyelvi standardban rögzített szabályok alapján kérdezés nélkül végezhet típuskonverziót, ilyenkor annak módja *implicit*. Ez utóbbi viselkedés azért hasznos mert egyszerűbbé teszi a forráskódunkat. Például: valós számok és egész számok közötti műveletek automatikusan valós számot adnak, különböző memóriaméretű egész számok közöttiek automatikusan a legnagyobb egész számot. [9]

Vigyázat: Sokszor akkor is megtörténik típuskonverzió, amikor nem számítunk rá!

Például: pl. implicit $5/2. \rightarrow 2.5$, explicit $((\text{double})5)/2. \rightarrow 2.5$

3.3. Érték adó operátorok

Az **értékadó operátorok** *adott változók értékeinek megváltoztatására szolgálnak*. Ezek alapja az *egyenlőségjel* (=) operátor:

[8] A típuskonverziós operátor általában azt eredményezi amit józan ésszel elvárunk tőle. C nyelvben típusok közötti általános konverzióra is alkalmas, de természetesen vannak olyan C adattípusok amelyek nem konvertálhatók egymásba.

[9] Összességében vegyes típusokra alkalmazott műveletek elvégzéséséhez logikusan a jobb típust választja a C fordító.

$=$ és $+=, \quad -=, \quad *=, \quad /=, \quad \%=,$ satöbbi.

Az egyenlőségjel (=) *a bal oldalára írt változónak, a jobb oldalára írt változó értékét adja*, implicit típus kényszerítés után.[10] Az egyes matematikai operátorok és az egyenlőségjel egybeírásából származott értékadó operátorok a matematikai műveletet hattatják az operátor bal oldalára, és oda eltárolják az eredményt.

- Például: $a-=2$ ugyanaz mint $a=a-2$.

Ezenkívül van két értékadó operátor amely kakukktolyásnak számít. Ezek egy változó jobb és bal oldalára is írhatók:

$++, \qquad --.$

- Például: $++a$ ugyanaz mint $a=a+1;$ $\qquad --a$ ugyanaz mint $a=a-1;$

Az értékadó operátorokról azt is érdemes megjegyezni, hogy nem csak értéket adnak, hanem *a kifejezésnek eredménye is lesz, meghozzá az a változó (vagy annak értéke) amelybe beírtuk az eredményt.* Ennek további felhasználását nem javasoljuk, mert átláthatatlanabbá teheti a forráskódunkat. [11]

[10] Amennyiben az = jel bal oldalán szereplő kifejezés típusa nem egyezik meg a jobb oldal típusával, akkor a fordító implicit típus kényszerítést végez rajta, hogy megfeleljen a bal oldal típusnak. Amennyiben a típuskényszerítés nem lehetséges, akkor a fordítás során hibaüzenetet kapunk.

[11] Remek példával szolgál erre $++$ (és $--$) operátor. Ha a változót a $++$ jobb oldalára írjuk, akkor a operátor ezen változót a megnövelt értékével adja vissza, ha a bal oldalára, akkor a változó megnövelése előtti értékét. Innen látható, hogy az előbbi viselkedés egyezik meg az egyenlőségjeles értékadó operátok viselkedésével.

3. Forráskód. Példák változókkal elvégzett aritmetikai műveletekre, és típus kényszerítésre.

```c
#include <stdio.h>
int main(){
    double var_x = -1.5, var_y = 2.e+4;
    int var_i = 3, var_j = 20, var_k = -300;

   //Alapműveletek egész számokkal (eredmény közvetlenül kiírva)
   printf("i+j = %i; i-j = %i; ", var_i+var_j, var_i-var_j);
   printf("j*i = %i; j/i = %i; j%%i = %i;\n", var_j*var_i, var_j/var_i,
           var_j%var_i); //sortörés nem számít

   //Ténylegesen változó változó
   double var_z = 20*(var_x+var_y/2)-11;  printf("z = %g; ", var_z);
   var_z *= 2; printf("z *= 2; z = %g;\n", var_z);

   //Vegyes
  printf("j/i = %g (valos); x/y = %i (egesz);\n",
                  ((double) var_j)/var_i, (int) (var_x/var_y) );
}
```

```
Kimenet:  i+j = 23; i-j = -17; j*i = 60; j/i = 6; j%i = 2;
Kimenet:  z = 199959; z *= 2; z = 399918;
Kimenet:  j/i = 6.66667 (valos); x/y = 0 (egesz);
```

3.4. Relációs operátorok

A relációs operátorok olyan műveletet végeznek, amely számértékek (vagy változók) *összehasonlítására* szolgál. Ezek az operátorok sorrendben: **egyenlő-e, nemegyenlő** (!=)**, kisebb, nagyobb, kisebb egyenlő, nagyobb egyenlő operátora**:

$$==, \quad !=, \quad >, \quad <, \quad >=, \quad <=$$

Minden relációs operátor két argumentumot vár el, az egyiket tőle jobbra, a másikat tőle balra írva. *Ha a relációs operátorral képzett állítás igaz, akkor az eredmény 1; hamis esetén a kifejezés eredménye 0 lesz.* Az eredmény típusa C-nyelvben `int`.

- Amennyiben egy változót több értékkel akarunk összehasonlítani, akkor már használnunk kell a következő pontban ismertetett logikai operátorokat.
- Ha a jobb- és baloldal típusa nem azonos, akkor a fordító implicit típuskényszerítést végezhet.

Vigyázat: Amennyiben összecseréljük = és a == operátorokat, akkor a gyakran előfordul, hogy a fordító sem jelez hibát!

Például: Legyen `int` $a = 1$. Ekkor $2 == a$; $a == 2$; és $a = 2$; utasítás is helyes, és eredményük a következő `int`: 0; 0; 2. Viszont $2 = a$; utasítás fordításkor már hibát ad!

3.5. Logikai operátorok

A logikai operátorok (a relációs operátorokkal karöltve) általában a C programunk vezérlésében játszanak fontos szerepet. *Ez gyakorlatban a Boole-algebrának megfelelő műveleteket jelenti*: 1 vagy 0 (igaz/hamis) értékekből állítanak elő újabb 1 vagy 0 (igaz/hamis) értéket. A következő operátorok jobb- és baloldali argumentummal rendelkeznek:

- `&&` – ÉS (AND) – ha mindkét érték nem 0, akkor igaz.
- `||` – VAGY (OR) – ha legalább az egyik érték nem 0, akkor igaz.

Ez pedig csak eggyel, amit mögé írunk:

- `!` – NEM (NOT) – az érték komplementerét adja.

A fordító az bemeneti értékekeket `int` típusra konvertálhatja, az eredményük is `int`.[12]

4. Forráskód. Példák logikai és relációs operátorok használatára.

```
1  #include <stdio.h>
2  int main(){
3     int a = 7, b = 2; //Relációs operátorokhoz
4     printf("| a == b | a >= b | a > b | a < b | a <= b |  relacio;\n");
5     printf("|    %i   |    %i   |   %i   |   %i   |    %i   |   eredmeny;\n",
6             a == b, a >= b, a > b, a < b, a <= b);
7     printf(" 2 < b < a | %i\n", (2 < b) && (b < a) ); // ÉS logikai operátor
8     //Boole algebra igazságtáblázatai
9     printf("NOT | %i |  %i  | %i \n", !0, !1, !-23); //NEM operátor
10    printf(" OR | %i |  %i  | %i \n", 0||0, 0||1, 0||-23); //VAGY operátor
11    printf("    | %i |  %i  | %i \n", 1||0, 1||1, 1||-23);
12 }
```

```
Kimenet:  | a == b | a >= b | a > b | a < b | a <= b | relacio;
Kimenet:  |    0   |    1   |    1  |    0  |    0   | eredmeny;
Kimenet:  2 < b < a | 0
Kimenet:  NOT | 1 | 0 | 0 |
Kimenet:   OR | 0 | 1 | 1 |
Kimenet:      | 1 | 1 | 1 |
```

[12] A C nyelvben Boole-algebrában használt igaz/hamis értékeket `int` típussal reprezentálják. Hamis az érték, ha az 0, igaz az érték, ha az nem 0.

Ellenőrző kérdések

1. Mit jelent az `int` kulcsszó?
2. Mi a különbség az igazi és a C nyelvben használt „lebegőpontos” valós számok között?
3. Milyen típus reprezentálja az egyszerűbb karaktereket a C nyelvben?
4. Kulcsszavakkal módosíthatók-e az elemi típusok?
5. Mit értünk a C nyelvben egy változó deklarációján?
6. Mit értünk a C nyelvben egy változó inicializációján?
7. Hogyan tudunk a C nyelvben szöveget kiírni a konzolra?
8. Hogyan működnek `printf()` függvényben a helyettesítő kifejezések?
9. Hogyan működik egész számokra az / és % aritmetikai operátor?
10. Mire jó a típus kényszerítés?
11. Sorolja fel a C nyelvben található relációs operátorokat!
12. Mi a jelentése az AND, OR, NOT Boole műveleteknek?

Gyakorló feladatok

1. Írjunk programot, amely a km/h-ban megadott sebesség értékeket átváltja m/s-ba. Váltsuk át a 90 km/h-t és az eredményt írassuk ki a képernyőre.
2. Írjunk egy programot, amely kiírja az első 10 természetes számot és azok négyzetét egymás mellé egy szépen rendezett táblázatba!
3. Írjunk egy programot, amely kiírja az AND operátor igazságtáblázatát!

Kiegészítések a kiválósághoz

1. Nézzünk utána `printf()` összes lehetséges helyettesítő kifejezésének!
2. C++ nyelvben a relációs operátorok eredménye `bool`, amely ott szintén egy elemi típus. Ez egy 1 bájt méretű egész szám, amely értéke csak `true` (= 1) és `false` (= 0) lehet. Ez a típus a C nyelv C99-es szabványában is létezik, amely elérhető a `<stdbool.h>`-ban.
3. C nyelv C99-es szabványában léteznek beépített típusok komplex számok reprezentálására is, például `double complex`, ehhez azonban szükséges `<complex.h>`. Ezekkel a komplex számokkal ezután szokásos módon végezhetők aritmetikai műveletek. Megjegyezzük, hogy a komplex számokat C++ nyelvben nem így valósították meg.

JELEN TANANYAG
A SZEGEDI TUDOMÁNYEGYETEMEN KÉSZÜLT,
AZ EURÓPAI ÚNIÓ TÁMOGATÁSÁVAL.
PROJEKTAZONOSÍTÓ: EFOP-3.4.3-16-2016-00014.