A jegyzet szerkesztés alatt áll, fokozott figyelmet kérek a használatánál.

Bevezetés az informatikába

Gyakorlati segédlet gazdasági informatikusok részére

Készítette: Szeghalmy Szilvia
Módosítva: 2016.09.13.

A jegyzet szerkesztés alatt áll, fokozott figyelmet kérek a használatánál.

Tartalomjegyzék

1. BEVEZETÉS 3

1.1. MIÉRT PONT A C NYELV? 3
1.2. FEJLESZTŐI KÖRNYEZET LETÖLTÉSE, TELEPÍTÉSE, BEÁLLÍTÁSA 3

2. RÖVID ÖSSZEFOGLALÓ A C NYELV ELEMEIRŐL 6

2.1. TÍPUSOK 6
2.2. KONSTANS 6
2.3. NEVESÍTETT KONSTANS 6
2.4. VÁLTOZÓK 7
2.5. ÉRTÉKADÓ UTASÍTÁS 7
2.6. KIFEJEZÉS 7
2.7. ELÁGAZTATÓ UTASÍTÁS 8
2.8. CIKLUSSZERVEZŐ UTASÍTÁSOK 9
2.9. VEZÉRLŐ UTASÍTÁSOK 9
2.10. FÜGGVÉNY 9
2.11. BEOLVASÁS ÉS KIÍRATÁS 10

3. FELADATOK ÉS MEGOLDÁSAIK 12

3.1. INPUT ÉS OUTPUT, AVAGY „ÜZENETVÁLTÁSOK A FELHASZNÁLÓVAL” 12
3.2. ELÁGAZTATÓ UTASÍTÁSOK, AVAGY „MERRE TOVÁBB?” 15
3.3. CIKLUS, AVAGY „LUSTÁNAK LENNI JÓ” 17
3.4. FÜGGVÉNYEK, AVAGY „AZ ÚJRAHASZNOSÍTÁS MŰVÉSZETE” 26
3.5. TÖMBÖS FELADATOK TESZTELÉSÉHEZ 29

4. TARTSD TISZTÁN! 30
5. HIBAVADÁSZAT 33

A jegyzet szerkesztés alatt áll, fokozott figyelmet kérek a használatánál.

1. Bevezetés

A jegyzet nem lektorált, tehát hibákat tartalmazhat, és minden bizonnyal tartalmaz is. Használata csak kellő körültekintéssel javasolt! A hibákat a szeghalmy.szilvia@inf.unideb.hu címre küldve lehet jelezni.

A *Bevezetés az informatikába* tárgy programozás része a *Magasszintű programozási nyelvek 1.* tárgy könnyebb megértését hivatott elősegíteni.

A jegyzetben használt jelölések

Szintaktikai leírásban szereplő jelölések:

{a|b}	alternatíva (vagy *a* vagy *b* írandó a helyére)
[a]	a szögletes zárójelben lévő rész opcionális (*a*-t nem kötelező kiírni)
[a]...	a zárójelben lévő rész, *a* tetszőlegesen sokszor ismételhető
<leírás>	A megadott rész a leírás szerint helyettesítendő

A programozási eszközök ismertetésénél a felső sorban az általános eset szerepel, alatta pedig egy konkrét példa.

```
típus változónév;
int x;
```

1.1. Miért pont a C nyelv?

Számos programozási nyelv létezik és valószínűleg jobb nyelvet is találhatnánk ahhoz, hogy a teljesen kezdőkkel megszerettessük a programozást. Azonban a *Bev. info.* tárgy keretein belül a programozás célja deklaráltan a *Magasszintű programozási nyelvek 1.* tárgy előkészítése. Igazodunk hát az ott tanított ismeretekhez, így biztos lehet benne az Olvasó, hogy a kezdeti erőfeszítések már a következő félévben elkezdenek megtérülni, később pedig más programnyelvek tanulásánál is jó alap lehet.

1.2. Fejlesztői környezet letöltése, telepítése, beállítása

Mielőtt belekezdünk a programozásba és megírjuk az első programunkat, nem árt egy olyan környezetet beállítani, ami segíti a későbbi munkánkat. A gyakorlaton az ajánlott IDE (*integrált fejlesztői környezet: szövegszerkesztő, fordító, nyomkövetési eszközök, stb.*) a Code::Blocks. Igyenes és több platform alatt is használható. A telepítő az alábbi linken érhető el.

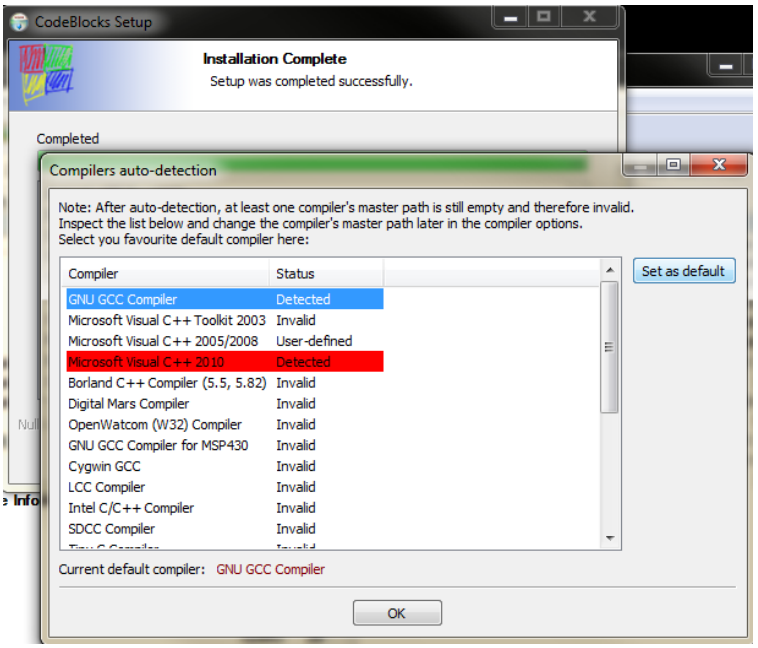
http://www.codeblocks.org/downloads/26

Windows felhasználóknak ajánlott a **codeblocks-16.01mingw-setup.exe** (illetve az aktuális verzió) letöltése. Ha az általunk használt gépen nem vagyunk rendszergazdák, akkor a *_user* végződéssel ellátott változat kell letölteni.

Megj.: A fejlesztőkörnyezet telepítését és beállítását mutató részeknél a különböző verziókban lehetnek eltérések, ezért nem kell kétségbe esni, ha a jegyzetben szereplő képernyőképek eltérnek az aktuálisan láttottól.

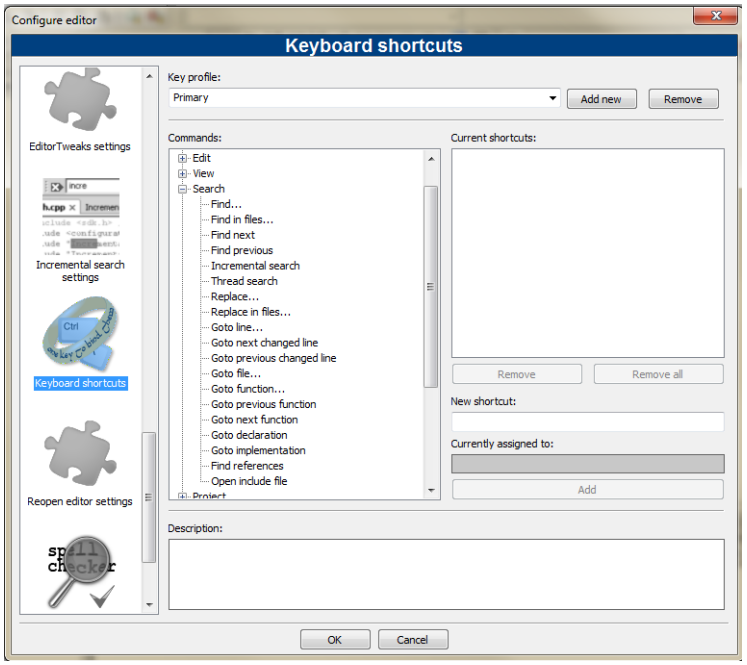
A telepítés futtatása után az első lépések teljesen egyértelműek, csak a *Next* gomb nyomkodására van

szükség. Amikor a telepítő rákérdez arra, hogy kívánjuk-e most futtatni a programot, válasszuk a *Yes* gombot. Amennyiben az alábbihoz hasonló ábra jelenik meg (1. ábra), válasszuk ki a listából a **GNU GCC Compiler**-t (ennek hatására a sor kék lesz), majd kattintsunk a *Set as default* gombra, végül az *OK*-ra. Még egy *Next* után elindul a *Code::Blocks*.



1. ábra: Code:Blocks telepítése. A megjelenő képernyő kinézete természetesen függ a letöltött verziótól és attól, hogy milyen fordítók találhatók a gépen, előfordulhat, hogy a lista egyetlen elemet tartalmaz.

Amennyiben magyar billentyű kiosztást használunk, a gyorsbillentyűk közt kénytelenek leszünk módosításokat tenni. Ehhez válasszuk a *Settings->Editor* menüpontot. Az új ablak bal oldali sávjában görgessünk lefele a *Keyboard shortcuts* nevű pontig és kattintsunk rá.



2. ábra: Code:Blocks telepítés.

A jegyzet szerkesztés alatt áll, fokozott figyelmet kérek a használatánál.

A *Commands* listában válasszuk ki az alábbi listapontokat és kattintsunk minden esetben a jobb oldalon lévő *Remove* gombra. (A jobb szélen lévő jelekkel jelzem, hogy melyik jelről vesszük le a gyorsbillentyűt.)

Search → Goto function	]
DoxyBlocks → Block comment	}
DoxyBlocks → Run CHM	&

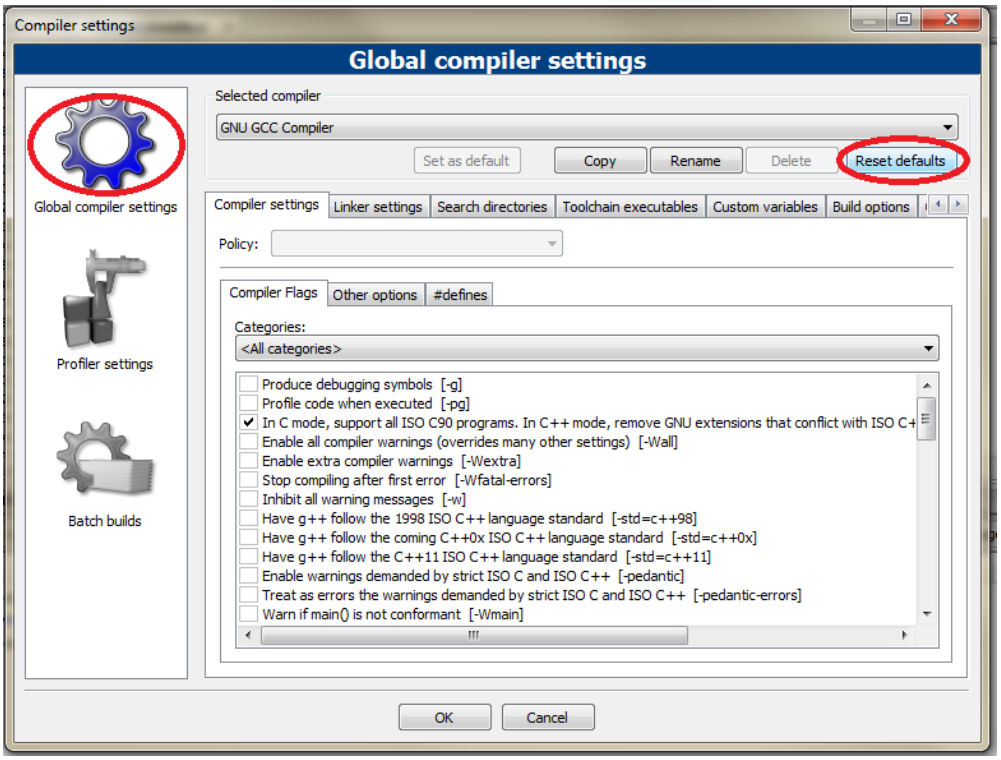
Ezek után tesztelhetjük egy nagyon egyszerű programmal a környezetet.

```
#include <stdio.h>

int main(){
   printf("Hello");
   return 0;
}
```

A fájlt mentsük el például hello.c néven. A fordítás és a futtatás hatására (F9 vagy csavar + zöld nyíl ikon) egy fekete konzolt kell kapnunk benne egy *Hello* üzenettel.

Ha létező *Code::Blocks* projekt fájlt szeretnénk megnyitni, akkor a kezdőképernyőt válasszuk az *Open an Existing Project* ikont, és keressük ki a megfelelő *.cbp* kiterjesztésű fájlt.

Amennyiben a fordítás hatására a program látszólag semmit sem csinál, akkor próbálkozzunk a *Settings → Compiler* menüpontban *a Global Compiler Settings* lapon *a Reset Defaults* gomb megnyomásával.



3. ábra: Code:Blocks fordító beállításainak alapértelmezettre állítása.

Figyeljünk rá, hogy a program futtatása után a konzolt mindig zárjuk le, különben nem fogjuk tudni újra futtatni a programot a Code::Blockson belülről.

A jegyzet szerkesztés alatt áll, fokozott figyelmet kérek a használatánál.

2. Rövid összefoglaló a C nyelv elemeiről

Ebben a fejezetben nagyon röviden, áttekintjük azokat a programozási eszközöket, melyekkel a gyakorlat során találkozni fogunk. A fejezet kétség kívül unalamas és tömény, ezért semmi esetre se zavarja az Olvasót, ha úgy érzi nem jegyzet meg sok mindent az olvasás alatt. A következő fejezetben lévő apró feladatok gép mellett történő megoldása után már egészen más lesz a helyzet.

2.1. Típusok

A típusokat tartományukkal, műveleteikkel, és reprezentációjukkal adjuk meg.
Az **órán gyakran használt** aritmetikai típusok:

Típus	Tartomány	Műveletek
int	egész szám	aritmetikai, logikai
float	valós számok	aritmetikai, logikai
double	dupla pontos valós számok	aritmetikai, logikai
char	1 bájtos egész	aritmetikai, logikai

Származtatott típusok: tömb, függvény, mutató, struktúra

Típusok közti átjárás
Az aritmetikai típusok közül vannak olyanok, melyeknél az egyik típus magában foglalja a másikat. Ilyen például a double - float, vagy a double - int, int - char, stb. párosítás. A bővebb tartomány irányában szabadon mozoghatunk, információ veszteség nélkül. Egy egész értéket információveszteség nélkül lehet egy valós típusú változóban tárolni. Fordított helyzetben viszont a valós szám egész része kerül csak tárolásra. A 3.8 egészként tárolva 3 lesz. Tehát nem kerekítés, hanem vágás történik.
Az aritmetikai műveleteknél, ha a két operandus típusa eltérő, a művelet elvégzése előtt, a bővebb tartomány irányába konvertálódik a másik operandus értéke.

2.2. Konstans

A konstansok érték és típus részből állnak. A konstansok értékét az alakjuk definiálja. Az adott típusú konstans, a típus tartományából veheti fel az értékét. Példák:

```
2                               //int típusú konstans
2.4                             //double típusú konstans
2.4f                            //float típusú konstans
'k'                             //karakter típusú konstans
"alma"                          //konstans karakterlánc
```

2.3. Nevesített konstans

Olyan konstans, amely névvel ellátott, és ezzel a névvel tudunk hivatkozni az értékére a program szövegében.

```
const típus változónév = kezdőérték;
const int N = 4;
```

C-ben a makró használata is gyakori konstans definiálására. Ezt a program elején a main függvényen kívül tehetjük meg.

```
#define MIT MIRE
#define N 4
```

Az előfordító a program szövegében előforduló *MIT* karaktersorozat minden előfordulását kicseréli a *MIT* után álló karaktersorozatra[1]. A példában *N* minden előfordulása *4*-re cserélődik.

Megj.: A makrók ennél jóval szélesebb körben használhatók, egész kifejezések helyettesítésére is lehetőségünk van, paraméteres formában is alkalmazható.

```
#define K (5*6)
#define MAX(a, b) ((a)>(b) ? (a) : (b))
```

*Az első sor eredményeként K minden előfordulása (5*6)-ra cserélődik. A második sorban egy olyan makrót láthatunk, mely két paramétert vár. Pl.: Ha a kódban szerpel MAX(1, 3), akkor a helyére ((1)>(3)?(1) :(2)), ha szerepel MAX(3-4, 5), akkor a helyett, a ((3-4)) > (5) ? (3-4) : (5)) kifejezés kerül.*

2.4. Változók

A változóknak van értéke, attribútumai (számunkra ez most a típus), címe és neve. A név betűvel kezdődik és betűvel vagy számjeggyel folytatódhat, vagyis egy *azonosító*. A C-ben betűnek számítanak az angol abc kis- és nagybetűi és a _ (aláhúzás), $ (dollár) jelek.[2] A változóra a nevével hivatkozhatunk a program szövegében. A cím az a memóriaterület, amelyen a változó értéke elhelyezkedik. Az érték pedig a változó típusához tartozó tartomány egy eleme.

```
típus változónév[=kezdőérték][, változónév2 [= kezdőérték2]]…
char c;                          //c nevű 1 bájtos egész
int x, y=6;                      //x és y egész típusú változók
int t[5];                        //t nevű 5 elemű egészekből álló tömb
létrehozása
```

2.5. Értékadó utasítás

```
Változó = kifejezés;
```

A bal oldalon mindig változó áll, a jobb oldalon tetszőleges *kifejezés* állhat. Az értékadás során a kifejezés kiértékelődik (értéke meghatározásra kerül), és ez az érték kerül a baloldali változóba.

```
int x, y=3, z, k;
int t[3];
x = 5;                           //x új értéke 5 lesz.
x = y+5*(2+5);                   //x új értéke 38 lesz
z = k = 5;                       //k értéke 5 lesz, z értéke szintén z =
(k=5)
t[0] = 2;                        //A t tömb legelső eleme kettő lesz.
```

2.6. Kifejezés

A kifejezések operandusokból, és operátorokból (műveleti jelek), valamint zárójelekből épülnek fel. Az operandus lehet konstans, nevesített konstans, változó, vagy függvényhívás.

---

1 A pontosvesszőnek itt nincs sorlezáró szerepe! A *#define N 4;* eredményeként az *N* előfordulásai a *"4;"* karaktersorozatra cserélődnek!

[2] Egyre több fordító támogatja a nemzeti karakterek szerepeltetését is, de ezzel inkább ne éljünk, mert más rendszeren problémát okozhat.

A jegyzet szerkesztés alatt áll, fokozott figyelmet kérek a használatánál.

A **gyakorlatokon gyakran használt** operátorok precedenciája a C nyelvben a következő:

( )	[]				→
!	++	--			←
*	/	%			→
+	-				→
<	>	<=	>=		→
==	!=				→
&&					→
||					→
=	+=	-=	*=	/=	%= ←

A táblázat felső sorában lévő műveleti jelek kötnek legerősebben (vagyis a kiértékelés során ezek hajtódnak végre először), és soronként gyengülnek. Az azonos sorban lévő operátorok kiértékelése a nyíl által mutatott sorrendben történik.

() Precedencia felülírása (matematikában megszokott zárójelezés) és függvényhívás jele.

[] Tömb operátor.

! Logikai operátor: tagadás.

++ -- Növeli, illetve csökkenti az egész értékű változó értékét.
Kifejezésben a prefix és postfix alak más-más eredményt adhat.

```
k = i++;                    //jelentése: k=i; i=i+1;
k = ++i;                    //jelentése: i=i+1; k=i;
```

* / + - Matematikában megszokott jelentéssel.
Az osztás egész számokon végrehajtva az eredmény egészrészét adja meg.

% Maradékos osztás. Negatív számokra ne használjuk.

< > <= >= == != Hasonlító operátorok. Jelentésük rendre: kisebb, nagyobb, kisebb egyenlő, nagyobb egyenlő, egyenlő, nem egyenlő. A hasonlító operátorok *a op b* értéke 1, ha *a op b* teljesül, különben 0. Ahol *op* valamelyik fenti jel.

&& || Logikai *és* illetve *vagy* operátorok. Rövidzár elven működnek, vagyis egy logikai kifejezés csak addig a pontig értékelődik ki, ameddig el nem dől az értéke.

= += -= *= /= %= Értékadó operátorok. Az értékadás bal oldalán változónak kell szerepelni! Jobb oldalon tetszőleges kifejezés állhat. Az *x op= y* művelet jelentése azonos az *x = x op (y)* kifejezéssel, ahol *op* az egyenlőségjel előtt álló jelek egyike.

C-ben, ha egy kifejezés értéke nulla, akkor logikai értéke hamis, különben logikai értéke igaz!

2.7. Elágaztató utasítás

```
if (kif)
   utasitas1;
[else
   utasitas2;]
```

Kiértékelődik a *kif* kifejezés, ha az értéke `igaz` (nullától eltérő), akkor lefut az *utasítás1*, különben, ha van *else* ág, akkor lefut az *utasítás2*.

2.8. Ciklusszervező utasítások

Elöltesztelő ciklus:

```
while(kif)
   utasitas;
```

Működése: kiértékelődik a *kif* kifejezés, ha igaz (nullától eltérő), akkor lefutnak a ciklus belsejében lévő *utasítások* (ciklusmag). Ez a két lépés addig ismétlődik, amíg a kifejezés értéke eltér nullától.

Hátultesztelő ciklus:

```
do
   utasitas;
while(kif);
```

Hasonló az előzőhöz, viszont itt a ciklusmag első körben mindig lefut, és csak utána jön a kifejezés kiértékelése.

Összetett ciklus:

```
for([kif1]; [kif2]; [kif3])
   utasitas;
```

Működése:
1. lépés: A *kif1*, ha meg van adva, egyetlen egyszer értékelődik ki, amikor program végrehajtása a ciklusfejet eléri. Általában a ciklusváltozó kezdőértékének beállítására használatos.
2. lépés: A *kif2*, az ismétlődésre vonatkozó feltétel. Ez fut le másodszor. Ha a *kif2* értéke igaz, akkor lefut a ciklusmag, és utána lefut a *kif3*, ami általában egy léptető utasítás. A második lépés ismétlődik, amíg a feltétel nullától eltérő.

2.9. Vezérlő utasítások

```
continue;
```

A ciklus belsejében kiadva átugorja a ciklusmagban lévő további sorokat, és a feltétel vizsgálattól (ill. for ciklus esetén a [kif3]-tól folytatódik a program.

```
break;
```

Ciklus belsejében kiadva azonnal kilép a ciklusból.

```
return [érték];
```

Függvényből való kilépésre szolgál. Az érték visszaadódik a hívás helyére (ha meg van adva).

2.10. Függvény

A függvények segítségével kiemelhetünk olyan programrészeket, melyek többször is szerepelnek a programban. A függvénynek lehetnek formális paraméterei. Ez teszi lehetővé, hogy a függvény más-más hívásnál eltérő értékekkel dolgozhasson. Ha a függvény típusa nem *void,* akkor a függvény eredménye egy konkrét érték lesz, melyet visszatérési értéknek nevezünk. (Bővebben: Prog. 1. tárgyon.)

```
TIPUS név([formális paraméter lista]){
```

```
    ...
}
```

A formális paraméterlista felépítése:

```
TIPUS név1[, TIPUS név2]…
```

Példa:

```
float oszt(int a, int b){
   return a / (float) b;          //a  (float)  rész  nélkül  egész  osztást
végezne
}
```

A fenti függvény két egész típusú formális paraméterrel rendelkezik. A visszatérési érték egy valós szám lesz, az első és második paraméter hányadosa.

Nézzünk két példát a függvény hívására:

```
int main(){
   float z = oszt(10, 5);         //A z változó értéke 2 lesz
   int x = 5;
   z = oszt(x, x-3 );             //A z változó értéke 5/2, vagyis 2.5 lesz
}
```

2.11. Beolvasás és kiíratás

A IO műveletek nem képezik közvetlenül a C nyelv részét, de a C szabvány standard könyvárakban rögzít bizonyos IO függvényeket, melyek minden C implementációban elérhetőnek kell lenniük (lásd. *stdio.h*). Órán a *scanf*, és *printf* függvények kerülnek elő.

```
int scanf(formátum_sztring [,valtozo_cim]…);
int printf(formátum_sztring [,kifejezes]…);
```

Nézzük először a *printf* függvényt. A formátumsztring egyrészt tartalmazat „sima” szöveget, mely egyszerűen a standard kimenetre másolódik, másrészt tartalmazhat formázó karaktereket, melyek megadják, hogy a paraméterlistában szereplő megfelelő kifejezés milyen módon kerüljön kiírásra.

```
%c (a paraméter kiírása karakterként)
%d (a paraméter kiírása egész számként)
%f (a paraméter kiírása valós számként)
%lf (a paraméter kiírása valós számként, de csak C99-től a szabványos)
```

A formátumsztring tartalmazhat olyan speciális jeleket is melyeket a \ (fordított per jel) vezet be, például:

```
\n (sortörés)
\r (kocsi vissza)
\t (tabulátor)
\f (új lap)
```

A *printf* függvény visszatérési értékét ritkán használjuk fel. Azt adja meg, hogy hány karakter került kiírásra.

A *scanf* függvény esetében is hasonló lehetőségeink vannak, azonban a formátumsztringben megadott karakterek nem kiírásra kerülnek, hanem lehetséges input formátumát írják le. Ha a leírásban „sima" karakterlánc szerepel, akkor pontosan az ott megadott karakterek érkezését várja el a függvény, míg ha whitespace karakter szerepel, akkor arra tetszőleges whitespace karaktersorozat illeszkedik a beolvasás során (pl.: egyetlen szóközre illeszkedik nulla szóköz, egy szóköz, két szóköz, tetszőlegesen sok szóköz vagy tabulátorjel, tehát akármilyen whitespace karakterből felépülő sorozat). A függvény a megfelelő formázó karakterekkel jelölt inputokat a paraméterlistában felsorolt változókba olvassa be.

A *scanf* visszatérési értéke azt adja meg, hogy hány érték beolvasása történt meg sikeresen, melyet felhasználhatunk pl. az input helyességének ellenőrzésére.

Lássunk néhány példát:

```
int x = 4, z;              //létrehozunk pár változót a példa kedvéjért
float y = 1.4f;
double g = 2.0;
char nev[41];
char karakter;

printf("ertek: %5d", x);  //x értékének kiíratása 5 helyen egész számként
printf("%d %f", x, y);    //egy valós és egy egész szám kiíratása szóközzel
printf("%f", y);          //valós szám kiíratása
printf("%f", g);          //duplapontos valóst is így írhatunk ki
printf("%lf", g);         //illetve C99-es szabványtól, már így is

printf("%c", karakter);   //karakter kiíratása
printf("%d", karakter);   //kiírás egész számként (vö. char előjeles egész)
printf("%s", nev);        //karaktersosorzat kiiratása


scanf("%d", &x);          //1 egész szám beolvasása x változóba
scanf("%d%d", &x, &z);    //1-1 egész szám beolvasása x-be és z-be
scanf("%f", &y);          //valós szám beolvasása y-ba
scanf("%lf", &g);         //dupla pontosságú valós szám beolvasása g-be
scanf("%c", &karakter);   //1 karakter beolvasása a karakter változóba
scanf("%s", nev);         //white spac karakterig olvas (pl: szóköz)
scanf("%d,%d", x, z);     //két vesszővel elválasztott egész beolvasása

if( scanf("%d%d", &x, &z) == 2 )
   printf("Szorzatuk: %d", x*y);
```

A `scanf("%s", nev);` sor kivételével minden esetben írtunk egy & jelet a változónév elé. Erre azért van szükségünk, mert a `scanf` egy memóriacímet vár, ahová beírhatja a beolvasott értékeket, a & jel, pedig egy változó neve elé írva pont azt adja meg. Egy tömb neve viszont eleve egy memóriacímet takar (az első elem memóriabeli helye), ezért ott nincs rá szükség.

A jegyzet szerkesztés alatt áll, fokozott figyelmet kérek a használatánál.

3. Feladatok és megoldásaik

Ez a fejezet nagyon rövid, kezdők számára is könnyen érthető programokat tartalmaz, melyeket különböző feladatok követnek. A feladatok megoldásához mindig csak kis mértékben kell módosítani az előtte álló programot, amit éppen ezért alapprogramnak fogunk hívni a fejezetben.

Sok feladat egyetlen célja, hogy egy adott programozási eszközt gyakoroltassa. Elsősorban tehát a szintaktika megjegyzésében segítenek a feladatok, emellett egy-két olyan „sablont" lehet megtanulni vele, amiből kiindulva később már értelmesebb feladatokat is meg lehet oldani.

A feladatok alatt a teljes megoldás, vagy a javítandó részlet van megadva.

3.1. Input és output, avagy „üzenetváltások a felhasználóval"

A „Hello Word" programmal már találkoztunk a környezet beállításánál, most az első interaktív programunk következik, mely alatt azt értjük, hogy a felhasználótól is várunk közreműködést.

Az alábbi példában egyetlen egész számot akarunk beolvasni a billentyűzetről és visszaírni azt a képernyőre[3].

```
#include <stdio.h>

int main(){
   int a;
   scanf("%d", &a );    //Ügyelj a & jelre, a scanf a változó címét várja
   printf("%d", a );
   /*ide szúrhatók a várakoztató részek, pl.: system("pause");*/
   return 0;
}
```

1. Példaprogram

Valljuk be, a fenti program nem túl felhasználó barát. Tegyük fel, hogy egy barátunkat kérjük meg a tesztelésre. Egy fekete konzolt fog látni és foglma sem lesz arról, hogy mit kell csinálnia. Amennyiben a programunkat humán felhasználók számára készítjük, fontos, hogy megfelelő üzenetekkel egyértelművé tegyük milyen inputot várunk, illetve tudassuk a felhasználóval, hogy a képernyőn mit lát.

```
#include <stdio.h>

int main(){
   int a;
   printf("Adj meg egy egész számot!\n"); //most már tudja, mit várunk
   scanf("%d", &a );
   printf("Ezt a számot adtad meg: %d", a ); //most már tudja, mit lát
   return 0;
}
```

[3] Precízebben fogalmazva a program a standard inputot (stdin) és a standard outputot (stdout) használja majd, melyek a program és külvilág közti kapcsolattartásra szolgáló adatfolyamok. Alapértelmezett módon az előbbi a billentyűzettel, az utóbbi a képernyővel van összekapcsolva, de mindkét adatfolyam átirányítható.

A jegyzet szerkesztés alatt áll, fokozott figyelmet kérek a használatánál.

1. Feladat: Alakítsd át úgy a példaprogramot, hogy valós értékre működjön!

```
#include <stdio.h>

int main(){
   float a;
   printf("Addj meg egy valós számot!\n");
   scanf("%f", &a );
   printf("Ezt a számot adtad meg: %f", a );

   return 0;
}
```

vagy

```
#include <stdio.h>

int main(){
   double a;
   printf("Addj meg egy valós számot!\n");
   scanf("%lf", &a );
   printf("Ezt a számot adtad meg: %f", a );

   return 0;
}
```

2. Feladat: Alakítsd át úgy az programot, hogy az két valós értéket kérjen be, és írja ki azok összegét.

```
#include <stdio.h>

int main(){
   float a, b;    //Természetesen most is használhatnák a double típust is
   printf("Adj meg egy valós számot!\n");
   scanf("%f", &a );    //ha a double típusú, akkor %lf
   printf("Adj meg még egy valós számot!\n");
   scanf("%f", &b );    //ha b double típusú, akkor %lf
   printf("Összeg: %f", a+b );

   return 0;
}
```

2. Példaprogram

Az alábbi program a felhasználó nevét kéri be, majd a nevet felhasználva köszön neki. A név eltárolásához egy 41 karakteres tömböt használunk. Mivel a karaktersorozatok C-ben ‘\0’-vál zárulnak, ezért maximum 40 hosszúságú név esetén működik helyesen a program. Látható, hogy a *scanf*-ben és *printf*-ben "%s"-t használunk a karaktersorozat (**s**tring) beolvasására, illetve kiírására. A kiírásban szereplő "\n" jelek sortörést eredményeznek.

Amennyiben olyan környezetünk van, ahol a konzol automatikusan bezárodik a futtatás után, ezért nem látjuk a végeredményt,

A jegyzet szerkesztés alatt áll, fokozott figyelmet kérek a használatánál.

a return 0; sor előtti system("pause"); megoldja, hogy a konzol csak akkor záródjon be, ha a felhasználó leüt egy billetnyűt.

```
#include <stdio.h>

int main(){
   char knev[41];
   printf("Mi a keresztneved?\n>");
   scanf("%s", knev );
   printf("Szervusz %s!\n", knev);

   return 0;
}
```

3. Feladat: Módosítsd a programot úgy, hogy az a felhasznló vezetéknevét kérje be.

Az alábbi programban a változások pirossal kiemelve láthatók. A feladat megodlásához igazság szerint a *printf*-ben szereplő keresztnév vezetéknévre való átírása is elegendő, azonban érdemes olyan változóneveket használni, mely utal arra, hogy mire szolgál az adott változó. Megtévesztő változóneveket semmi esetre se használjunk.

```
#include <stdio.h>

int main(){
   char vnev[41];
   printf("Mi a vezetekneved?\n>");
   scanf("%s", vnev );
   printf("Szervusz %s!\n", vnev);

   /*ide szúrhatók a várakoztató részek, pl.: system("pause");*/
   return 0;
}
```

4. Módosítsd a programot úgy, hogy a felhasználó korát is kérje be, majd a kor begépelése köszönjön a a következő módon: „Szervusz <kor> éves <nev>”. (Például: Szervusz 12 éves Olga) A kor tárolásához hozz létre egy `int` típusú „kor” nevű változót. A beolvasáshoz használd a `scanf(”%d”, &kor);` utasáítást.

```
#include <stdio.h>

int main(){
   char knev[41];
   printf("Mi a keresztneved?\n>");
   scanf("%s", knev );

   int kor;
   printf("Hany eves vagy?\n>");
   scanf("%d", &kor );
   printf("Szervusz %d eves %s.\n", kor, nev);

   return 0;
}
```

A jegyzet szerkesztés alatt áll, fokozott figyelmet kérek a használatánál.

3.2. Elágaztató utasítások, avagy „merre tovább?”

A szorgalmas hallgató reggel felkel, ha órája van (legalábbis, ha gyakorlati órája van). Ha éhes, megreggelizik. Majd elkóválygyog az egyetemig. Ha még mindig nem tudja kinyitni a szemeit, akkor vesz egy kávét a büfében. Már, ha nyitva van a büfé. Ha nincs nyitva, akkor megteszi az autómatából árult lötty is.

Képzeletbeli hallgatónk tehát más-más tevékenységeket hajt végre, attól függően, hogy bizonyos feltételek teljesülnek-e vagy sem. A jó hír, hogy a programunk is képes lehet erre, például az **if** szerkezet használatával. Nézzünk most erre néhány példát.

3. Példaprogram

Az előző „köszönő” programunk kissé udvariatlan. Még a 99 éves nagyit is letegezi és vele is bolondozik a korával. Alakítsuk át úgy, hogy a felnőtteket „Jó napot kivanok <nev>!”-vel köszöntse.
Mivel a felhasználó által megadott életkor alapján dönthető el, hogy felnőtt-e az illető, az if-be a korra vonatkozó feltétel kerül. Azt is könnyű kitalálni, hogy az if-nek a kor beolvasása után kell állnia, hiszen előtte még nem tudjuk az illető korát. A kódunk tehát valami ilyesmi:

```
#include <stdio.h>

int main(){
   char knev[41];
   printf("Mi a keresztneved?\n>");
   scanf("%s", knev );

   int kor;
   printf("Hany eves vagy?\n>");
   scanf("%d", &kor );

   if( kor >= 18 )
      printf("Jo napot kivanok %s.\n", kor, nev);
   else
      printf("Szervusz %d eves %s.\n", kor, nev);

   return 0;
}
```

5. Kérd be két felhasználó keresztnevét és korát, majd írd ki melyikük az idősebb, vagy egyidősek-e.

```
#include <stdio.h>

int main(){
   char knev[41];
   printf("Mi a keresztneved?\n>");
   scanf("%s", knev );

   int kor;
   printf("Hany eves vagy?\n>");
```

```
    scanf("%d", &kor );

    char knev2[41];
    printf("Es neked mi a keresztneved?\n>");
    scanf("%s", knev2 );  //mar az uj valtozot hasznaljuk

    int kor2;
    printf("Hany eves vagy?\n>");
    scanf("%d", &kor2 );  //Ne felejtsuk el itt is atirni a valtozot az ujra

    if( kor > kor2 )      //osszehasonlitjuk a ket valtozoban tarolt erteket
       printf("%s az idosebb.\n", knev);
    else if( kor < kor2)
       printf("%s az idosebb.\n", knev2);
    else
       printf("Egyidosek vagytok.\n");

    return 0;
}
```

4. Példaprogram

Lássunk egy másik példát, ahol a felhasználó kora alapján kiírja a program, hogy a triatlonon serdülő kategóriába esik-e vagy sem.

```
#include <stdio.h>

int main(){
   int kor;
   printf("Hany eves vagy?\n>");
   scanf("%d", &kor );

   if( 14 <= kor && kor <= 15)  //vagy if(kor == 14 || kor == 15)
      printf("serdülő");
   else
      printf("nem serdülő");

   return 0;
}
```

6. Egészítsd ki a programot úgy, hogy a táblázat alapján adja meg a korosztály besorolást. Ha a kort egyik korosztály sem tartalmazza, akkor az „Egyéb” felirat jelenjen meg. A szintaktika ehhez hasonló lesz, persze a közéső else if-es rész többször fog ismétlődni:

```
if( feltétel)
   utasítás;
else if( feltétel)              //ez a rész ismétlődni fog eltérő
feltételekkel
   utasítás;                    //és eltérő szöveggel
else
   utasítás;
```

Életkor	Korosztály
11 évesig	Újonc
12–13	Gyermek
14–15	Serdülő
16–17	Ifjúsági
18–19	Junior

```
#include <stdio.h>

int main(){
   int kor;
   printf("Hany eves vagy?\n>");
   scanf("%d", &kor );

   printf("A triatlon korosztályod: ");
   if(  kor <= 11)
      printf("Újonc");
   else if( kor <= 13)
      printf("Gyermek");
   else if( kor <= 15)
      printf("Serdülő");
   else if( kor <= 17)
      printf("Ifjúsági");
   else if( kor <= 19)
      printf("Junior");
   else
      printf("Egyéb");

   return 0;
}
```

3.3. Ciklus, avagy „lustának lenni jó”

Írjunk ki a képernyőre 1-től 10000-ig minden egész számot. Lássunk is neki!

```
#include <stdio.h>

int main(){
   printf("1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
```

Na ne…, így soha nem végzünk! Ha valamit sokszor kell megismételni, arra ott van a ciklus.

```
#include <stdio.h>

int main(){
   int i;
   for(i = 1; i<=10000; ++i)
      printf("%d ", i);
```

```
    return 0;
}
```

Hát nem kényelmesebb így? A ciklus fejlécében megadtuk, hogy az *i* változó honnan induljon (*i*=1). Megadjuk azt is hogyan változzon (*++i*, vagyis minden lépésnél 1-gyel növeljük az értékét), és azt is, hogy meddig lépjünk be újra a ciklusba (*i*<=10000, vagyis ha *i* értéke eléri a 10000-t akkor még belépünk és kiírjuk az 10000-et, viszont, ha az *i* már 10001, akkor nem lépünk be a ciklusba.

Az alábbi feladatokban nem lesz ilyen nagy a tartomány, de maradj a ciklusnál a megoldások során. Csak az ellenőrzés megkönnyítésére lett szűkítve a számok száma.

5. Példaprogram

Az alábbi program a számokat jeleníti meg 1-től 10-ig, a határokat is beleértve egy-egy szóközzel elválasztva.

```
#include <stdio.h>

int main(){
   int i;
   for(i = 1; i<=10; ++i)
      printf("%d ", i);

   return 0;
}
```

7. Kezdjük valami nagyon egyszerűvel: Módosítsd úgy a programot, hogy az a 0 és 15 közötti számokat jelenítse meg (a határokat is beleértve).

```
#include <stdio.h>

int main(){
   int i;
   for(i = 0; i<=15; ++i)
      printf("%d ", i);

   return 0;
}
```

8. Akkor sem kell félni, ha negatív számokkal kell dolgozni: Módosítsd úgy a programot, hogy az a -10 és 10 közötti számokat jelenítse meg (határokat is beleértve).

```
#include <stdio.h>

int main(){
   int i;
   for(i = -10; i<=10; ++i)
      printf("%d ", i);

   return 0;
}
```

9. Emeljük a tétet. Mi a helyzet, ha nem egyesével akarunk lépkedni? Módosítsd úgy programot, hogy az a 10 és 100 közötti páros számokat jelenítse meg (határokat is beleértve).

```
#include <stdio.h>

int main(){
   int i;
   for(i = 10; i<=100; i+=2)  //i+=2 vagy i=i+2
      printf("%d ",i);

   return 0;
}
```

Jó, de kevésbé hatékony megoldás az alábbi is. (Persze, ilyen kevés érték esetében ebből semmit sem fogunk észlelni.)

```
#include <stdio.h>

int main(){
   int i;
   for(i = 10; i<=100; ++i)
      if(i%2 == 0)    //ha i értéke 2-vel osztva 0 maradékot ad
         printf("%d ",i);

   return 0;
}
```

10. A ciklusváltozó értékét nyugodtan lehet csökkenteni is, csak arra kell figyelnünk, hogy a kezdőértéket és a feltételt is ennek megfelelően adjuk meg. Módosítsd úgy a programot, hogy az a 100 és 10 közötti számokat jelenítse meg csökkenő sorrendben, a **határok nélkül**.

```
#include <stdio.h>

int main(){
   int i;
   for(i = 99; i>10; --i)
      printf("%d ",i);

   return 0;
}
```

11. Módosítsd úgy a programot, hogy az kérje be az alsóhatárt, felső határt (vesszővel elválasztva) és íjra ki a szigorúan a határok között lévő értékeket.

```
  int main(){
    int i, ah, fh;
     printf("Add meg az also es a felso hatart (pl: 2,10):");
     scanf("%d,%d", &ah,&fh);  //az elvárt elválasztó karaktert is odaírjuk

     for(i = ah+1; i<fh; ++i)
       printf("%d ", i);
```

```
  return 0;
}
```

6. Példaprogram

Az eddigi ciklusos példákban mindig egész szám töltötte be a ciklusváltozó szerepét, de ennek egyáltalán nem kell így lennie. Az alábbi program a [0; 5] tartományba eső valós számokat írja ki, 0-tól indulva, 0,25-ös lépésközzel.

```
#include <stdio.h>

int main(){
   double i;  //a változót valósnak vesszük fel

   for(i = 0; i<=5; i += 0.25) //a léptetésnél 0.25 (ponttal!!!)
     printf("%f ", i); //és figyeljünk arra, hogy valósként jelenjen meg

  return 0;
}
```

Mielőtt továbblépünk erről a részről, egy kicsit játszunk a valós számokkal. Próbáljuk ki float típussal is a fenti kódot.

```
   float i;
   for(i = 0; i<=5; i += 0.25f)
     printf("%f ", i);
```

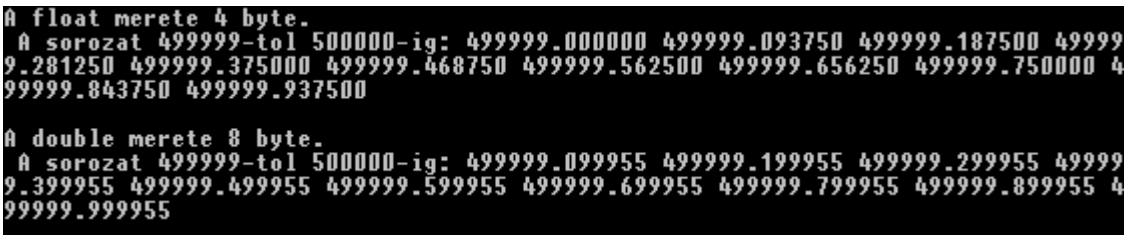
Majd próbáljuk meg 0.1-es lépésközzel is:

```
   double i;
   for(i = 0; i<=5; i += 0.1)
     printf("%f ", i);

   float i;
   for(i = 0; i<=5; i += 0.1f)
     printf("%f ", i);
```

```
d:\>jajj.exe
0.000000 0.100000 0.200000 0.300000 0.400000 0.500000 0.600000 0.700000 0.800000
 0.900000 1.000000 1.100000 1.200000 1.300000 1.400000 1.500000 1.600000 1.70000
0 1.800000 1.900000 2.000000 2.100000 2.200000 2.300000 2.400000 2.500000 2.6000
00 2.700000 2.799999 2.899999 2.999999 3.099999 3.199999 3.299999 3.399999 3.499
999 3.599999 3.699999 3.799999 3.899998 3.999998 4.099998 4.199998 4.299998 4.39
9998 4.499998 4.599998 4.699998 4.799998 4.899998 4.999998
```

4. ábra: Ha lebegőpontos számokkal dolgozunk, számítanunk kell a hiba halmozódására.

Ajjaj, mi történik itt? Aggódalomra semmi ok, gondoljunk a számábrázolásnál tanultakra. A 0.1 kettes számrendszerben 0.00011 0011 0011… Ezt az értéket nem tudta a gép pontosan eltárolni sem a *float*, sem a *double* típust használva, ezért a számítás során egyre nő a hiba (5. ábra).



5. ábra: A változó értékét 0-ról indulva 500 000-es értékig növeltük, minden körben 0,1-gyet hozzáadva. Csak a float és a double tárolásához használt bájtszám és a sorozat utolsó néhány eleme van megjelenítve. Látható, hogy double típusnál kisebb a hiba, de ott is megjelenik. (A teljes sorozatot ne próbáld meg egyesével kiírni a képernyőre, többmillió elemnél az már egy kicsit lassú művelet.)

12. Lássuk mi a helyzet a karekterekkel. Kérjünk be két karaktert a felhasználótól, és jelenítsük meg a közt lévő karaktereket.

```
#include <stdio.h>

int main(){
   char i, ah, fh;
   printf("Add meg az also es a felso hatart (pl: a,h):");
   scanf("%c,%c", &ah,&fh);

   for(i = ah; i<=fh; ++i)
     printf("%c ", i);

  return 0;
}
```

Ha teszteljük a programot az *'a' és 'z'* karaktereket megadva, az angol ábécé összes karaktere megjelenik szép sorban. Azért működik, mert a karakterek az ASCII kódjukkal vannak reprezentálva, és az ASCII táblában az angol ábécé betűi egymás után helyezkednek el (külön a nagyok, külön a kicsik). Egy kis változtatással a kódban már láthatjuk is, hogy csak rajtunk múlik miként kívánjuk értelmezni a háttétben álló sorozatot.

#include <stdio.h>

```
int main(){
   char i, ah, fh;
   printf("Add meg az also es a felso hatart (pl: a,h):");
   scanf("%c,%c", &ah,&fh);
   for(i = ah; i<=fh; ++i)
     printf("Karakter: %c, ASCII kod: %d\n", i, i);

  return 0;
}
```

Próbáljuk most az 'a' és az 'é' karakterre. Ez már kevésbé sikeres. Az 'é' ASCII kódja a 130-as, ennek a bináris reprezentációja pedig 1000 0010. A *char* típus viszont egy bájtos előjeles egész, tehát ha a legfelső bit 1, akkor azt a gép kettes komplemensben lévő számként kezeli. Visszaszámolja az eredeti értéket, ami a -126-os értéket takarja, tehát a for ciklusban megadott felső határ kisebb, mint az alsó. Javítsunk rajta! Válasszunk a *char* helyett olyan típust, ami képes [0, 255] tartományban tárolni a

számokat, pl. unsigned char-t. Ha előjeles típusnál akarunk maradni, akkor egy bővebb tartománnyal rendelkező típust kell választani, például *short*ot[4] vagy *int*et, ezek ugyanis tartalmázzák a [0, 255] egész számokból álló tartományt is.

```
int main(){
   unsigned char i, ah, fh;
   printf("Add meg az also es a felso hatart (pl: a,h):");
   scanf("%c,%c", &ah,&fh);
   for(i = ah; i<=fh; ++i)
     printf("Karakter: %c, ASCII kod: %u\n", i, i);

  return 0;
}
```

7. Példaprogram

Ciklust nem csak akkor használhatunk, ha sokszor kell megismételni valamit. Akkor is érdemes ehhez a szerkezethez fordulni, ha nem tudjuk előre, hogy hányszor kell végrehajtanunk valamit.

Az alábbi program számokat olvas be a konzolról, egészen addig, ameddig a nulla értéket nem üti be a felhasználó. És mivel a for ciklus már minden bizonnyal az Olvasó könyönkén jön ki, ideje egy másikat is megismerni.

```
#include <stdio.h>

int main(){
   int x;
   do{
      printf("Szam>");
      scanf("%d", &x);
   }while(x!=0); // while(x) formában is írhatnánk (0-hamis, minden más igaz)

   return 0;
}
```

Persze ez is megoldható for ciklussal is:

```
#include <stdio.h>

int main(){
   int x;
   for(x = 1 ; x ; ){//ha x logiaki értéke igaz (x != 0 is állhatna ott)
      printf("Szam>");
      scanf("%d", &x);
   };
   return 0;
}
```

Az x=1 biztosítja, hogy a for ciklust elérve belépjünk a ciklusba, hiszen így igaz lesz a ciklusfeltétel.

[4] Előjeles egész, általában 2 bájt a mai rendszereken

Belépve a ciklusba x-be beolvassuk egy számot. Majd… hoppá, hol van a for ciklus 3. kifejezése? A 3. kifejezés általában a ciklusváltozó módosítására szolgál. Ezt viszont most a cikluson belül tettük meg a *scanf*-fel. Mert az alábbi mégis csak ronda lenne így elsőre[5]:

```
for(x = 1 ; x ; printf("Szam>"),scanf("%d", &x))
      ;
```

Szóval visszatérve a folyamatra: a szám beolvasása után újra kiértékelődik a feltétel és annak értéke dönt róla, hogy belépünk-e a ciklusba vagy sem.

13.Módosítsd úgy a programot, hogy az visszaírja a pozitív számokat a kimenetre.

```
#include <stdio.h>

int main(){

   int x;
   do{
      printf("Szam>");
      scanf("%d", &x);
      if( x > 0)
         printf("%d\n", x);
   }while(x!=0);

   return 0;
}
```

14.Módosítsd úgy a programot, hogy az visszaírja a pozitív, páros számokat a kimenetre.

```
#include <stdio.h>

int main(){

   int x;
   do{
      printf("Szam>");
      scanf("%d", &x);
      if( x > 0 && x % 2 == 0 )  //vagy: if( x>0 && !(x%2) )
         printf("%d\n", x);
   }while(x!=0);

   return 0;
}
```

8. Példaprogram

Az alábbi program az előzőhez hasonlóan működik. Nulláig olvas számokat és vissza is írja a kimenetre a beovlasott értékeket. Annak érdekében, hogy a nulla „végjel” már ne kerüljön kiírásra egy kicsit módosult a ciklus. A while feltétele egy mindig igaz értéket tartalmaz. Ilyen esetben a ciklus belsejében kell elhelyezni a ciklus befejeződését biztosító részt (pirossal jelölve).

---

[5] Ez bizony lefordulna! A C nyelv sok mindent megenged, de ez nem jelenti azt, hogy élni is kell vele. Igyekezzünk tiszta, világos kódokat írni, az ilyen lehetőségeket pedig hagyjuk meg szórakozásnak.

```
#include <stdio.h>

int main(){
   int x;
   while(1){
      printf("Szam>");
      scanf("%d", &x);
      if(x == 0)  //gyakran a !x alakban fordul elő
         break;
      printf("%d\n", x);
   }
   return 0;
}
```

15. Módosítsd úgy a programot, hogy minden beolvasott értékre írja ki, hogy egyjegyű-e a szám vagy sem (a nulla „végjelre” ne írjunk ki semmit).

```
#include <stdio.h>

int main(){
   int x;
   do{
      printf("Szam>");
      scanf("%d", &x);
      if(x == 0)
         break;
      if( -10 < x && x < 10)
        printf("Egyjegyű\n");
      else
        printf("Nem egyjegyű\n");
   }while(x!=0);

   return 0;
}
```

9. Példaprogram

Az alábbi program az előzőhez hasonlóan működik, azonban a cikluson kívül egy másik változót is felveszünk, melyet az inputértékek alapján változtatni fogunk. A konrkét pédában a beolvasott számok összegét határozzuk meg. A számsor megadása most is a nulla hatására fejeződik be. Figyeljük meg, hogy a kiiratás a cikluson kívülre került, annak érdekében, hogy csak a végeredményt jelenítsük meg.

```
#include <stdio.h>

int main(){
   int x;
   int s = 0;
   while(1){
      printf("Szam>");
      scanf("%d", &x);
      if(x == 0)  //gyakran a !x alakban fordul elő
         break;
      s += x;
   }
```

```
   printf("Az osszeg: %d", s);

   return 0;
}
```

16. Módosítsd úgy a programot, hogy megszámolja hány értéket adott meg a felhasználó. (a nulla „végjel” nem számít bele)

```
#include <stdio.h>

int main(){
   int x;
   int s = 0;
   while(1){
      printf("Szam>");
      scanf("%d", &x);
      if(x == 0)  //gyakran a !x alakban fordul elő
         break;
      ++s; //vagy pl.: s = s + 1
   }
   printf("Darabszám: %d", s);

   return 0;
}
```

17. Módosítsd úgy a alprogramot, hogy megadja a beolvasott elemek szorzatát. (a nulla „végjel” nem számít bele) Feltételezheted, hogy ad meg a felhasználó legalább egy értéket.

```
#include <stdio.h>

int main(){
   int x;
   double s = 1.0; //szorzatnál gyakran szükséges a tágabb tartomány is
   while(1){
      printf("Szam>");
      scanf("%d", &x);
      if(x == 0)  //gyakran a !x alakban fordul elő
         break;
      s *= x; //vagy pl.: s = s * x
   }
   printf("A szorzat: %f", s);  //valós kiíratás

   return 0;
}
```

18. Módosítsd úgy a programot, hogy megadja a beolvasott elemek szorzatát (a nulla „végjel” nem számít bele). Ha a felhasználó nem adott meg egyetlen elemet se (egyetlen nulla értéket adott meg), akkor írd ki, hogy „érvénytelen”. Tipp: a beolvasott elemek száma fentebb már szerepelt.

```
#include <stdio.h>

int main(){
   int x;
   double s = 1.0;
```

```
    int db = 0;
    while(1){
       printf("Szam>");
       scanf("%d", &x);
       if(x == 0)  //gyakran a !x alakban fordul elő
          break;
       s *= x; //vagy pl.: s = s * 1
       ++db;
    }
    if(db > 0)  // if(db) is elegendő, mert minden 0-tól eltérő érték „igaz”
       printf("A szorzat: %f", s);
    else
       printf("érvénytelen");

    return 0;
}
```

3.4. Függvények, avagy „az újrahasznosítás művészete”

10. Példaprogram

Az alábbi program egy olyan függvényt tartalmaz, mely egy téglalap két szomszédos oldalának hosszát várja paraméterként és megadja annak méretét. A programhoz tartozó `main` függvény megmutatja, hogyan tárolhatjuk el a függvény által adott értéket a későbbi felhasználáshoz. (A felhasználás ezúttal csak egy sima kiiratás lesz, ezért akár az `t` helyére a `teglalapTerulete(oldal1, oldal2)`-t is írhattuk volna.

```
#include <stdio.h>

float teglalapTerulete(float a, float b){
   return a*b;
}

int main(){
   float oldal1, oldal2;
   printf("Add meg téglalap két szomszédos oldalának hosszát vesszővel
elválasztva. \n>");
   scanf("%f,%f", &oldal1, &oldal2);

   float t = teglalapTerulete(oldal1, oldal2);
   printf("A téglalap területe: %f", t);

   return 0;
}
```

19. Egészítsd ki a programot egy téglalap kerületének kiszámítására szolgáló függvénnyel. A *main*-en belül hívd meg a függvényt és jelenítsd meg a kerületet is.

```
#include <stdio.h>

float teglalapTerulete(float a, float b){
```

```
    return a*b;
}

float teglalapKerulete(float a, float b){
    return 2*(a+b);
}

int main(){
    float oldal1, oldal2;
    printf("Add meg téglalap két szomszédos oldalának hosszát vesszővel
elválasztva. \n>");
    scanf("%f,%f", &oldal1, &oldal2);

    float t = teglalapTerulete(oldal1, oldal2);
    printf("A téglalap területe: %f", t);

    float k = teglalapKerulete(oldal1, oldal2);
    printf("A téglalap kerülete: %f", k);

    return 0;
}
```

20. Írj programot, mely egy sugarad olvas be. Jelenítsd meg az adott sugarú kör kerületét, területét, és az adott sugarú gömb felszínét és térfogatát. A kerület, terület, felszín és térfogat számítását is egy-egy függvény végezze.

21. Írj programot, mely egy hasáb 3 szomszédos élének hosszát olvassa be és az alapján kiírja a hasáb felszínét és térfogatát. A felszín és a térfogad számítása egy-egy függvénnyel történjen.

11. Példaprogram

Az alábbi program egy olyan függvényt tartalmaz, mely képes egy egészekből álló tömb összegét meghatározni. A programhoz tartozó main függvényben két tömböt hozunk létre, és a függvény felhasználásával kiszámítjuk a tömbbeli számok összegét.

```
#include <stdio.h>

int osszeg(int t[], int n){
    int i, ossz = 0;
    for(i = 0; i<n; ++i)
        ossz += t[i];
    return ossz;
}

int main(){
    int t[] = {0, 3, 5, 11, 4};
    int ossz = osszeg(t, sizeof(t)/sizeof(t[0]));

    int t2[] = {6, 7, 2, 3, 5, 11, 4};
    int ossz2 = osszeg(t2, sizeof(t2)/sizeof(t2[0]));

    printf("Osszeg: %d\n", ossz);
```

```
   printf("Osszeg2: %d\n", ossz2);

   return 0;
}
```

1. Írj függvényt, mely visszadja egy egészekből álló tömb elemeinek átlagát. A tömböt és méretét paraméterben kapja! Figyelj rá, hogy valós típusú értéket adjon vissza a függvényed.
2. Írj függvényt, mely visszadja hány pozitív elem van egy egészekből álló tömbben. A tömböt és méretét paraméterben kapja!
3. Írj függvényt, mely kiszámolja 2 N dimenziós vektor belső szorzatát. (Mindkét vektor egy N elemű tömbbel van reprezentálva.)
4. Írj függvényt, mely meghatározza egy egészekből álló tömb legnagyobb elemének tömbbeli pozícióját. A tömböt és méretét paraméterben kapja!

Megoldások

A megoldások során most csak a függvényeket adjuk meg.

1.

```
double atlag(int t[], int n){
   int i, sum = 0;
   for(i = 0; i<n; ++i)
      sum += t[i];
   return sum/(double)n; //egészosztás: 1/2 = 0, de 1/(double)2 = 0.5
}
```

2.

```
int pozitiv(double t[N]){
   int i, count = 0;
   for(i = 0; i<N; ++i)
      if(t[i]>0) ++count;
   return count;
}
```

3.

```
double szorzat(double v[N], double v2[N]
   int i;
   double s = 0;
   for(i = 0; i<N; ++i)
      s += v[i]*v2[i];
   return s;
}
```

4.

```
int maximumhely(int t[N]){
   int i, maxh = 0;
   for(i = 1; i<N; ++i)
   if(t[maxh] < t[i])
      maxh = i;
   return maxh;
```

```
}
```

3.5. Tömbös feladatok teszteléséhez

A teszteléshez érdemes olyan méretű tömböt, és olyan elemeket választani, hogy a megoldást le is tudd ellenőrizni. Az alábbi kód használható keretnek a tömb feldolgozására irányuló feladatoknál.

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#define N 12 // ha más elemszámot akarsz, ezt változtasd

//min max közti értékekkel lesz tele a tömb
int feltolt(int t[], int n, int min, int max){
   int i, m = max - min + 1;
   for(i = 0; i<n; ++i)
      t[i] = rand() % m + min;
   }

void kiir(int a[], int meret){
   int i;
   for(i = 0; i<meret; ++i){
      printf("%d ", a[i]);
   }
}


int main(){
   //véletlenszám generátor inicializálása
   srand(time(NULL));
   int tomb[N];

   // így tudod feltölteni például -20 és 20 közti értékekkel
   feltolt(tomb, N, -20, 20);

   //írhatod ide a kódod, vagy készíthetsz függvényt.

   kiir(tomb, N);

     return 0;
}
```

4. Tartsd tisztán!

Talán nem haragszik meg érte egy korábbi (ma már sokkal jobban programozó) hallgatóm, hogy egy korai próbálkozásából merítek ihletet arra vonatkozóan, hogy hogyan nem kellene kinéznie a kódunknak. A hallgató célja az az volt, hogy különböző méretű tömböket véletlen, nem nulla értékekkel töltsön fel, kiírassa azokat, és különböző értékeket számoljon a tömb elemeire (ami esetünkben már nem lényeges, a kód így is elég hosszú lesz):

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int main(){
   time_t t;
   srand((unsigned int)&t); //a véletlenszámgenerátor inicializálása

   int sorozat1[1000];
   int sorozat2 [100];
   int sorozat3 [900];

   int i, szam;
   for(i = 0; i<1000; ++i){
     do{
       szam = rand()%100 - 100;
     }while(szam == 0);
     sorozat1 [i] = szam;
   }

   for(i = 0; i<100; ++i){
     do{
       szam = rand()%100 - 100;
     }while(szam == 0);
     sorozat2 [i] = szam;
   }

    for(i = 0; i<900; ++i){
     do{
       szam = rand()%100 - 100;
     }while(szam == 0);
     sorozat3 [i] = szam;
   }

   for(i = 0; i<1000; ++i)
     printf("%d ", sorozat1[i]);
   puts("");  //sztring kiírásra szolgál (formázás nélkül, sortöréssel)

   for(i = 0; i<100; ++i)
     printf("%d ", sorozat2[i]);
   puts("");

   for(i = 0; i<900; ++i)
     printf("%d ", sorozat3[i]);
   puts("");
```

```
    //A folytatást most mellőzzük…

    return 0;
}
```

Ha ebben a szellemben dolgozunk tovább, akkor borzasztó hosszú lesz a kód, ami nem csak a megírás idejét, de a hibázási lehetőség esélyét is növeli. Nehezebb lesz áttekinteni és karbanatartni a kódot. A beégetett számok tovább rontják a helyzetet, mert könnyű eltéveszteni, melyik konstans melyik tömbhöz tartozik.

Lássunk egy tisztább megoldást:

1. A kódban lévő „bűvös számokat” (legalább azokat, amik több helyen is előfordulnak a kódban) nevezzük el, hogy tudjuk, melyik érték mit jelöl.
2. Azokat a részeket, amit többször is el kell végezni, csak más-más értékekre emeljük ki függvényekbe:
    a. tömb feltöltése véletlen értékekkel,
    b. tömb értékeinek megjelenítése.
3. Lássuk el a programot kommentekkel, hogy ezzel is segítsük a későbbi karbantartást. A függvényeknél érdemes megadni azok célját, a paraméterek leírását és (ha nem void), akkor azt is, hogy (milyen esetben) mit ad vissza a függvény.

```
/**
* A fájl elei kommentben gyakran szerepel a program leírása
* A készítő neve, ideje
* Komolyabb programoknál a licenc
*/

#include <stdio.h>
#include <stdlib.h>
#include <time.h>

///
/// A tömbök mérete
///
#define SOROZAT1_MERET 1000
#define SOROZAT2_MERET 100
#define SOROZAT3_MERET 900


/**
* A függvény feltölti a tömböt véletlenszerű, de nem nulla
*  egész értékekkel a [-100, 99] tartományban.
* @params tomb   A feltöltendő tömb.
* @params meret  A tömb mérete.
*/
void feltolt(int tomb[], int meret){ //tombot és annak méretét várjuk
   int i;  //Ez az i, csak ezen a függvényen belül „él”.
```

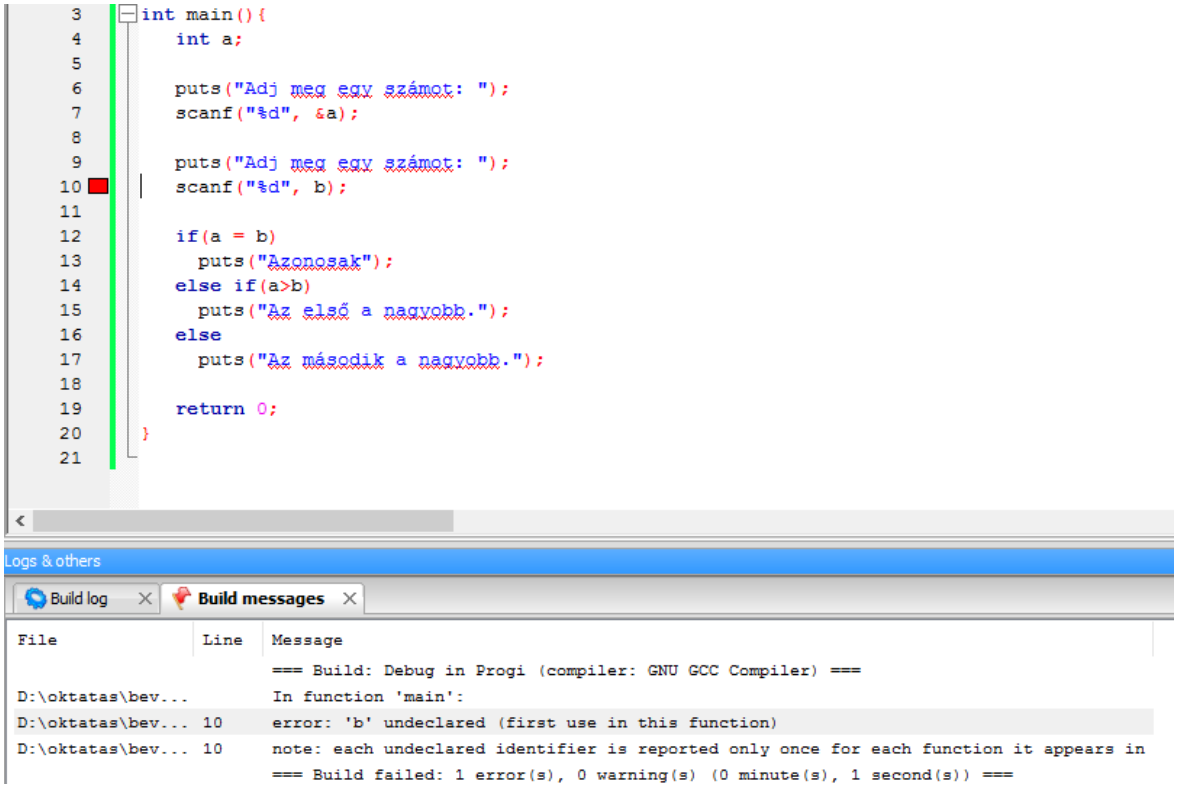
```
    for(i = 0; i<meret; ++i)
      do{
        szam = rand()%100 - 100;
      }while(szam == 0);
      tomb[i] = szam;
    }
}


/**
* A függvény szóközzel elválasztva megjelenítni a tömb elemeit.
* @params tomb   A kiírandó tömb.
* @params meret  A tömb mérete.
*/
void kiir(int tomb[], int meret){
    int i; //Ez az i, nem azonos a feltoltben szereplővel (csak druszánok)
    for(i = 0; i<meret; ++i)
      printf("%d ", tomb[i]);
    puts("");
}

int main(){
    time_t t;
    srand((unsigned int)&t); //a véletlenszámgenerátor inicializálása

    int sorozat1[SOROZAT1_MERET];
    int sorozat2[SOROZAT2_MERET];
    int sorozat3[SOROZAT3_MERET];

    feltolt(sorozat1, SOROZAT1_MERET);
    feltolt(sorozat2, SOROZAT2_MERET);
    feltolt(sorozat3, SOROZAT3_MERET);

    kiir(sorozat1, SOROZAT1_MERET);
    kiir(sorozat2, SOROZAT2_MERET);
    kiir(sorozat3, SOROZAT3_MERET);



    return 0;
}
```

A fenti kód most már könnyebben áttekinthető és karbantartható. A függvények újrahasznosíthatóságának mértékét még lehetne fokozni, például azzal, ha a véletlenszám generátornál a határokat is átadnánk, akinek van kedve nyugodtan megoldhatja azt is.

A jegyzet szerkesztés alatt áll, fokozott figyelmet kérek a használatánál.

5. Hibavadászat

A hibák lehetnek *szintaktikai* jellegűek, ami alatt azt kell értenünk, hogy nem tartottuk be a nyelvtani szabályokat. Pl.: elmaradt a pontosvessző, egy-egy zárójel, elégpeltük a függvény nevét, stb. Az ilyen hibák megtalálása érdekében böngésszük át a fordítás után kapott üzeneteket. A Code::Blocks-ban *Logs&others* ablak *Build messages* fülén tehetjük ezt meg (5. ábra). Az adott sorra duplán kattintva odaugorhatunk a javítandó sorra.

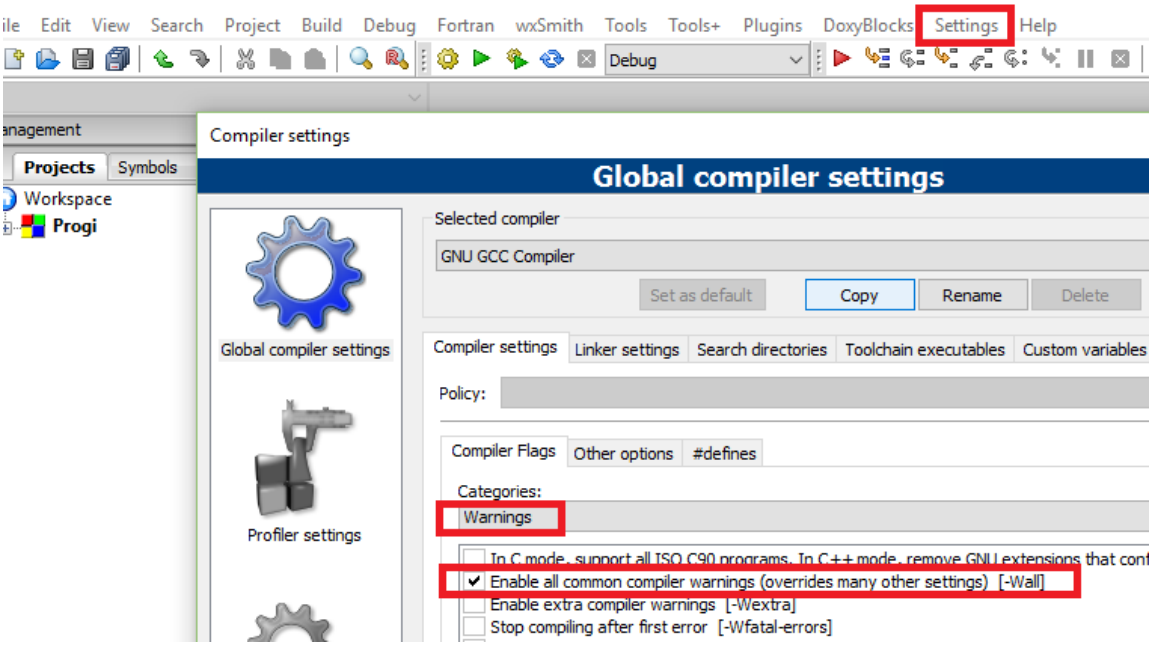


5. ábra: Fordítási hiba: A kiírás szerint a main függvény 10. sorában szereplő *b* deklarálása elmaradt. Feltételezve, hogy a *b*-t *int b;*-vel kívántuk deklarálni, az adott sorban egy másik hiba is van, az elmaradt & jel, de azt nem fogja hibának jelezni a fordító, mert szintaktikailag helyes. Szintén van egy hiba a 12. sorban, ugyanis egyenlőségvizsgálat helyett (==) értékadás (=) történt.

Figyelmeztetések (warningok):

A fordítás során érdemes olyan beállítást használni, hogy a rendszer az 5. ábra aláírásában ecsetelt hibákra is figyelmeztessen bennünket. Ennek módját a 6. ábra mutatja.

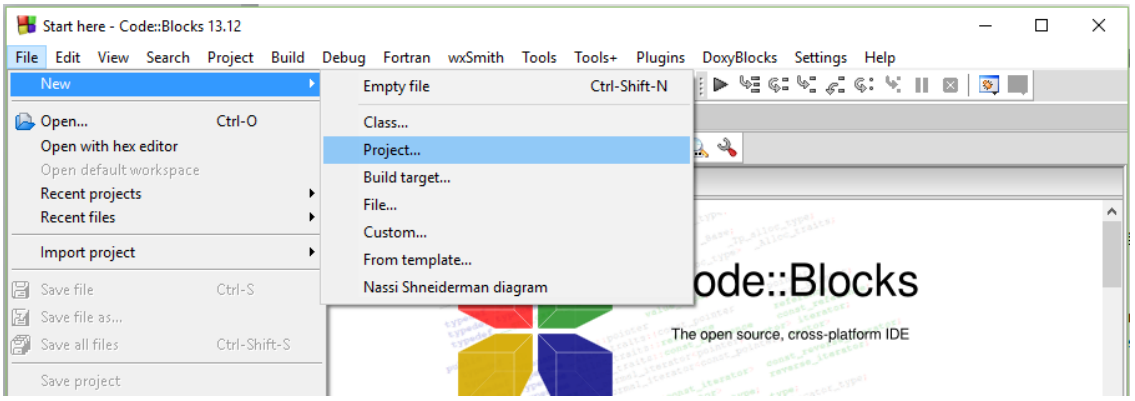
A jegyzet szerkesztés alatt áll, fokozott figyelmet kérek a használatánál.



6. ábra: A warningok beállításához a Settings menü, Compiler almenüjét választva feljön a képen látható ablak. Válasszuk a „Warnings” kategóriát és jelöljük be az „Enable all common compiler warnings”-t.

A másik hibafajta a *szemantikai* hiba. Ekkor a kód lefordul, de nem (minden esetben) hozza az elvárt működést, netán ki is akad. Az ilyen hibák megkeresése során (a warningok átnézése mellett) használhatjuk a „fapados” megoldást, amikoris kiírásokkal tűzdeljük tele a kódot, megjelenítjük bizonyos változók értékét. Például, ha a fenti kódban visszaírjuk a beolvasott értékeket, máris világossá válik, hogy gond van a *b* változó beolvasásánál.

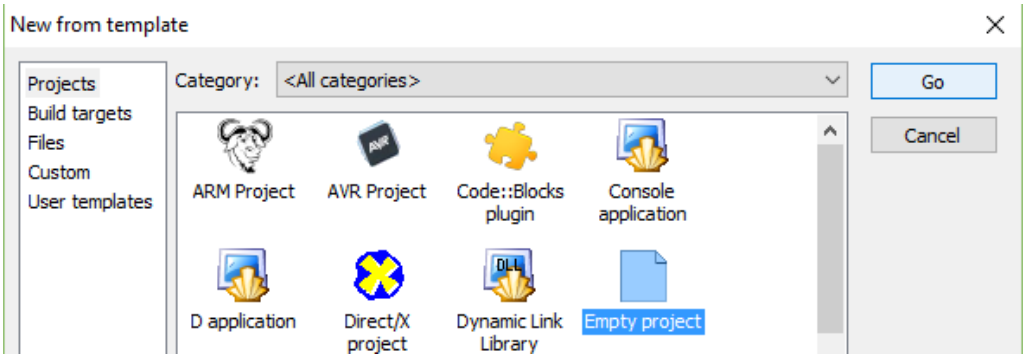
A másik megoldás, ha a fejlesztőkörnyezet segítségét vesszük igénybe. A fejlesztőkörnyezetekben *Debug* néven általában találhatunk egy előre definiált fordítási konfigurációt, ami a *nyomkövetést* támogatja. A Code::Blocks-ban akkor válik aktívvá, ha egy projektet hozunk létre (7. ábra).



7. ábra: Projekt létrehozása

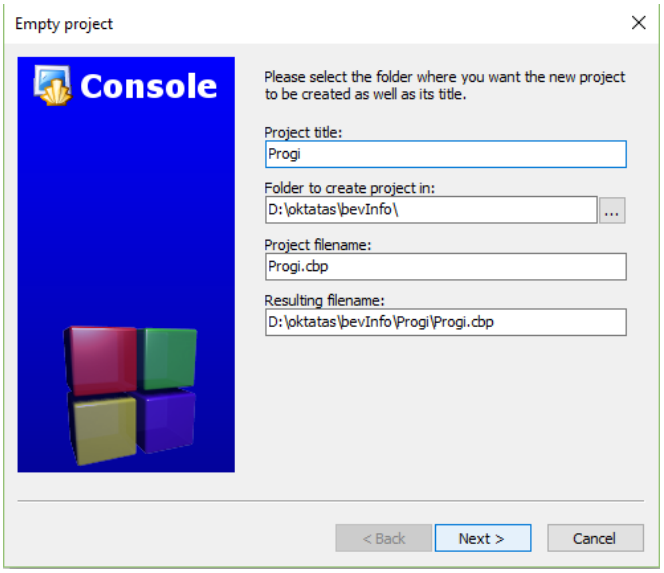
A feljövő ablakban válaszzuk ki az *Empty project*et (8. ábra) és nyomunka Go-ra.

A jegyzet szerkesztés alatt áll, fokozott figyelmet kérek a használatánál.



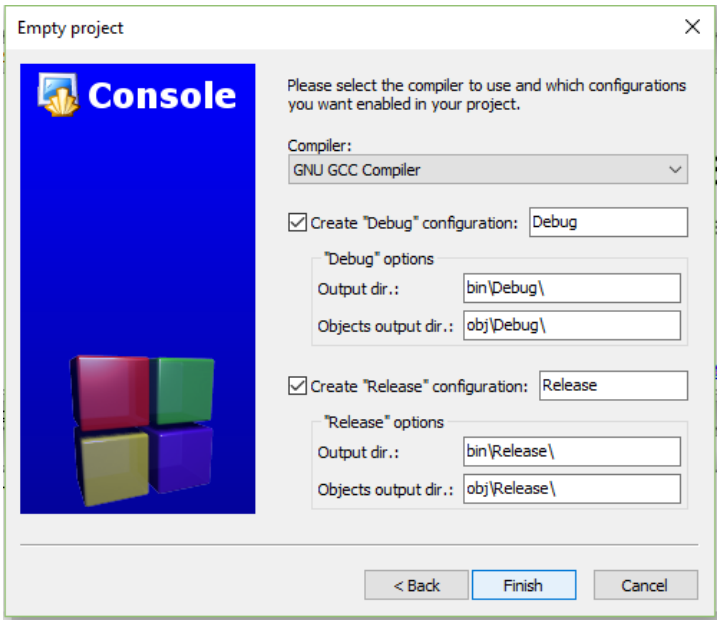
8. ábra: Projekt létrehozása

Majd töltsük ki a feljövő ablakban a megfelelő információkat (Projekt neve, mentés helye) és kattintsunak a *Next* gombra (9. ábra).



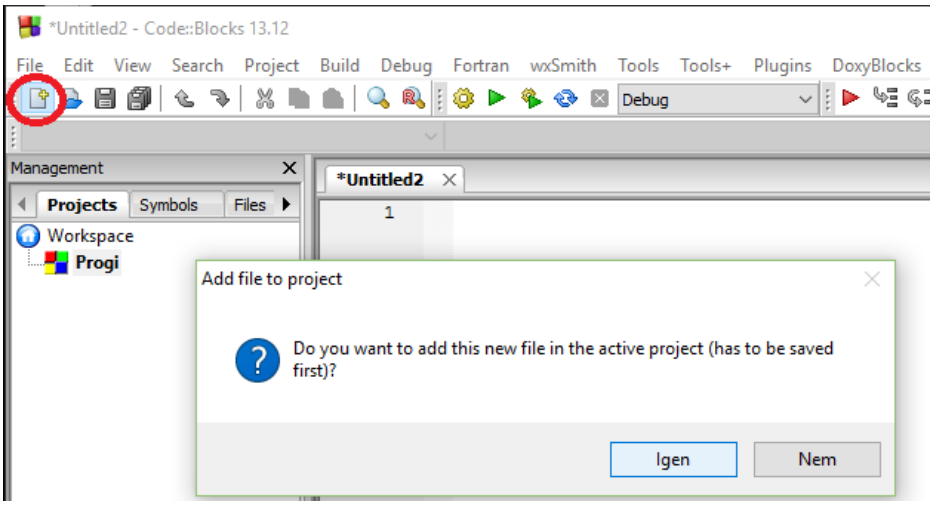
9. ábra: Projekt létrehozása

Ha ezzel készen vagyunk, akkor feljön egy új ablak (10. ábra), ahol látható, hogy külön könyvtárba fognak kerülni a *Debug* és a *Release* módban fordított programhoz tartozó fájlok.



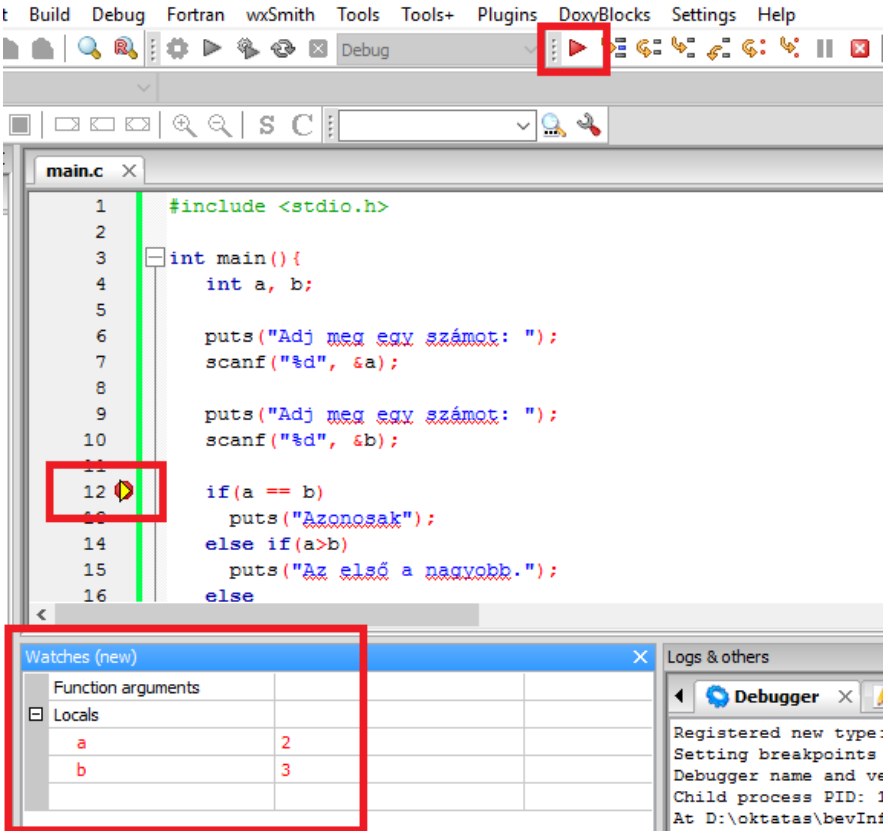
10. ábra: Projekt létrehozása

A projekt létrehozása után a *Management* fülön látható *Projects* ablakban kezelhetők a projektek (akár egyszerre több is). Ha üres projektet hoztunk létre, akkor a következő lépés új fájl hozzáadása, vagy egy meglévő hozzáadása. Előbbi esetén a Code::Blocks rákérdez, hogy hozzá akarjuk-e adni az aktív projekthez, és azt is, hogy mely konfigurációkhoz.



11. ábra: Új fájl hozzáadása az aktív projekthez. Az aktív projekt félkövérrel kiemelve a Management ablak Projects fülénél látható. Láthatjuk azt is, hogy a menüsor alatti eszköztárban most már a Debug mód is elérhető.

A nyomkövetés során lehetőségünk van arra, hogy bizonyos soron megállítsuk a kód futását, akár lépésenként is haladhatunk a kódban és közben monitorozhatjuk a változók értékét. Lásd. *Watches* ablak (12. ábra).



12. ábra: Futtatás Debug módban. A 12. sorban a szám jobb oldalára kattintva elhelyeztünk egy „break point”-ot, ami a futást megakasztja. A Watches ablakban láthatjuk a változók pillanatnyi értékét. A kódban való lépkedéshez a menüsor alatti piros nyíl mellett lévő ikonokat használhatjuk a nyomkövetés során.

A jegyzet szerkesztés alatt áll, fokozott figyelmet kérek a használatánál.

Irodalomjegyzék

[1.] N1570 Committee Draft (April 12, 2011) ISO/IEC 9899:201x
http://www.open-std.org/jtc1/sc22/WG14/www/docs/n1570.pdf