

01: LearnC

A C programozás alapjai - ami a Cocoa-hoz szükséges

Learn C

Ez a cikk a [Cocoa Dev Central: Learn C](#) magyar nyelvű fordítása, az eredeti angol változat [Scott Stevenson](#) munkája.

A következő oldalakon a Cocoa programozáshoz szükséges alapvető C programozás ismereteket foglaljuk össze. Feltételezzük, hogy ismersz legalább egy olyan programozási nyelvet, amelyik tartalmaz függvényeket, változókat és ciklusokat. Ezen kívül szükséges a Mac OS X Terminal programjának alapszintű ismerete.

Egy kis felmérés

Győződj meg róla, hogy érted az alábbi program kódot mielőtt folytatnád az olvasást.

```
function display_area_code ($code)
{
print ("$code ");
}

$area_codes[0] = 408;
$area_codes[1] = 650;
$area_codes[2] = 510;

/* this is a comment */

$count = 3;

for ($i = 0; $i < $count; $i++)

display_area_code ($area_codes[$i]);
```

Ebben a példában található egy függvény, egy print utasítás, egy megjegyzés, egy tömb, változók és egy ciklus. Ha ez mind ismerős számodra, akkor teljesítetted a folytatáshoz szükséges peremfeltételeket.

Az alábbi példaprogramokat [innen](#) összecsomagolt formátumban letöltheted.

Ahhoz, hogy programot fejlessz Mac OS X-re, az Xcode-ot telepítened kell a gépedre, ugyanakkor az alábbi kódok valószínűleg bármilyen operációs rendszer alatt működni fognak.

Egy egyszerű C program

Legyen ez az első C programunk. Az alábbi kódot másold át egy szövegszerkesztőbe és mentsd el **test1.c** néven.

```
test1.c
```

```
#include

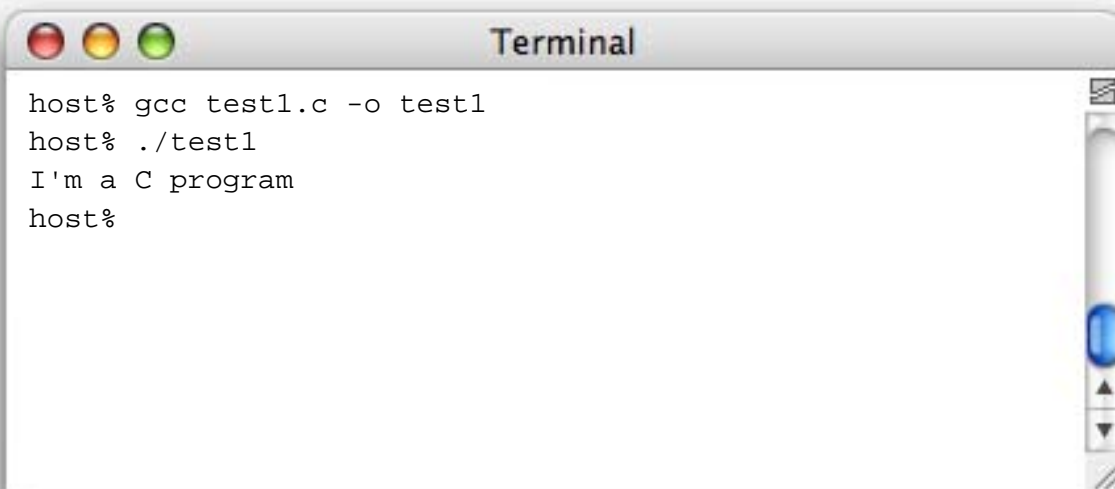
main ()
{
printf ("Ez egy C program\n");
}
```

Az **#include** utasítás beemeli az írás/olvasásért felelős C utasításokat. Most a **printf** függvényre lesz szükségünk. A következő sorban a **main** függvény található, amit minden C program tartalmaz. Végül a **printf** függvény segítségével kiiratunk egy szöveget. A **\n** egy új sort rak a szöveg után.

Használjuk a GCC fordítót

Nyisd meg a Terminal programot és lépj be abba a könyvtárba, ahol a test.c fájl található. A **gcc test1.c -o test1** parancs segítségével lefordítjuk, majd a **./test1** paranccsal futtathatjuk.

Ha hibajelzést kapnál, akkor ellenőrizd, hogy pontosan írtad-e be a program kódot.



```
Terminal

host% gcc test1.c -o test1
host% ./test1
I'm a C program
host%
```

A **-o** paraméter megmondja a gcc fordítónak, hogy mi legyen a program neve. Ha ezt elhagyjuk, akkor a program neve **"a.out"** lesz.

A fordító program

Szükséged van egy fordító programra annak érdekében, hogy a C forráskódból egy futtatható *bináris programot* készítsen. A lefordított program általában sokkal gyorsabban fut, mint egy szkript. Sok Mac OS X programot írnak C nyelven.

Az egyszerűség kedvéért most a gcc fordítót használjuk, de jó tudni, hogy van egy nagyon jó grafikus fordító program is ugyenerre a feladatra, amit Xcode-nak hívnak.

Hordozhatóság

Egy C program általában csak olyan típusú számítépen futtatható, ahol le lett fordítva. Azaz egy Mac OS X alatt fordított C programot nem tudunk futtatni például Linux környezetben. Ehhez az szükséges, hogy lefordítsuk ugyanazta forráskódot Linux alatt is.

Bonyolultabb C programok esetében szükség lehet arra is, hogy minden egyes platformra különböző kódot írjunk, figyelembe véve az ottani sajátosságokat. Ezt hívják a program *portolásának*.

C sajátosságok

Könnyebb tanulni a c programozást, ha egy már ismert programozási nyelvhez tudjuk hasonlítani. Röviden összefoglaltuk azokat az újdonságokat, amiket meg kell ismerned ha korábban már írtál PHP, vagy Perl programokat:

A C nyelv	
fordító	használható programot készít a C forráskódból
tipizált változók	a változók által tartalmazott adatok típusai
tipizált függvények	függvények visszatérési értékeinek típusai
header fájlok (.h)	függvények és változók definiálása egy külön fájlban
struktúrák	kapcsolódó értékek csoportjai
enum-ok	előre definiált értékek listája
mutatók	másik változóra mutató linkek

Ez egy meglehetősen leegyszerűsített séma. A C nyelv nem is annyira *komplikált*, bár kétségtelen, hogy alaposan el is lehet bonyolítani.

Ha a mutatóktól eltekintենk, akkor a *C szinte ugyanaz lenne, mint a PHP*. A mutatók (pointerek) meglehetősen trükkös lények, de a Cocoa megkímél attól, hogy a kezdő programozóknak el kelljen mélyülni ezek rejtelmeiben.

Ezért ebben a bevezető cikkben nem is fogunk részletesen foglalkozni a mutatókkal, arra koncentrálnunk, hogy minél gyorsabban elsajátítsuk az alapokat, és elegendő lesz majd később tanulmányozni a mutatókat részletesebben, amikor komolyabb programozási feladatokat kapunk.

02: LearnC

Változók, függvények

Tipizált változók

A szkript nyelvek esetében a változókat nagyon rugalmasan használhatjuk. Nyugodtan kicserélhetjük a változó egész szám tartalmát egy tizedes számra, vagy egy sztringre:

```
$variable = 2;  
$variable = 1.618;  
$variable = 'A';
```

A C szabályai ennél sokkal szigorúbbak. Előre *meg kell határozni a változó által tartalmazandó adat típusát* és a típus később már nem változtatható meg. Az iménti program kód C-beli változata így néz ki:

```
int variable1 = 2;  
float variable2 = 1.618;  
char variable3 = 'A';
```

Vegyük észre, hogy *három különböző változót* kellett deklarálni a C programbeli változatban. A típus deklarálás össze is vonható:

```
float var1 = 1.618;  
var2 = 3.921;  
var3 = 4.212;
```

Típusok

Jelenlegi céljainkhoz a következő beépített típusokra lesz szükség:

Típus	Leírás	Példák
int	egész szám (lehet negatív is)	0, 78, -1400
unsigned int	egész szám (nem negatív)	0, 46, 900
float	lebegőpontos valós szám	0.0, 1.618, -1.4
char	egy betű karakter, vagy szimbólum	'a', 'D', '?'

Bár csak ritkán van rájuk szükség, ne feledkezzünk meg a **double** típusról, amikor a lebegőpontos szám több tizedesjegyet tartalmazhat, mint a **float** és a **long** típusról, ami abszolút értékben nagyobb egész számokat tartalmazhat, mint az int.

A C egyik fontos tulajdonsága, hogy *megengedi a saját* változó típusok deklarálását.

Tipizált függvények

A C nyelvben a függvények visszatérési értékét is előre kell definiálni. A visszatérési érték típusa tetszőleges C változó típus lehet, és a függvény neve előtt kell megadni.

```
int numberOfPeople ()
{
return 3;
}
```

```
float dollarsAndCents ()
{
return 10.33;
}
```

```
char firstLetter ()
{
return 'A';
}
```

A visszatérési érték lehet **void** is, amiről egyelőre elegendő úgy értelmezni, hogy nincs visszatérési érték:

```
void printHello ()
{
printf ("Hello\n");
}
```

Paraméter típusok

A függvényeknek *átadott értékek* típusát is előre meg kell adni. A szkript nyelvekkel ellentétben itt nincs lehetőség default értékek beállítására.

```
int difference (int value1, int value2)
{
return value1 - value2;
}

float changeDue (float amountPaid, float costOfItem)
{
return amountPaid - costOfItem;
}
```

Függvények deklarálása

A C nyelvben egy függvényt még az előtt kell deklarálni mielőtt bármilyen kódrészletben hivatkoznánk rá. Mindegyik függvényt elhelyezhetjük a main() függvény előtt, de ez néha kényelmetlen lehet. lehetőség van azonban a *függvény prototípus* használatára. Ez olyan mint egy függvény definíció, csak nem tartalmaz kapcsos zárójeleket és pontosvesszővel kell lezárni:

```
int difference ( int value1, int value2 );
float changeDue ( float amountPaid, float costOfItem );
```

A függvény prototípusban megadjuk a függvény nevét, a visszatérési érték típusát és az átadott paraméterek típusát. A következő példa kódot illesszük a **test2.c** fájlba.

```
test2.c

#include

int sum ( int x, int y );
main ()
{
int theSum = sum (10, 11);
printf ( "Sum: %i\n", theSum );
}
int sum ( int x, int y )
{
return x + y;
}
```

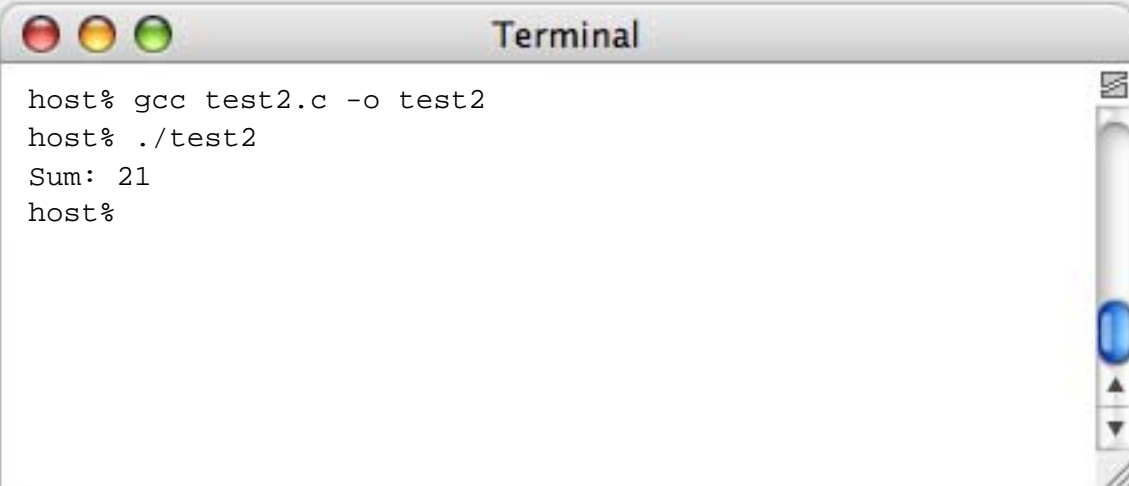
Az include sor biztosítja, hogy használhatjuk a printf függvényt. A következő sor definiálja a *sum* függvényt a protípus definíció segítségével. A main függvényen belül meghívjuk a sum függvényt és az eredményt a *theSum* nevű változóban tároljuk.

Mivel a sum függvény visszatérési értékének az int típust adtuk meg, ezért a theSum változó típusánk is ugyanennek kell lennie. *A változó típusának meg kell egyeznie a függvény visszatérési értékének típusával.*

A theSum változó tartalmát a *printf* utasítás segítségével tudjuk megjeleníteni. A végén található maga a sum függvény, amelyik összeadja a két bemeneti számot. Ezt hívjuk a függvény implementációjának.

A példa program fordítása

Nyissuk meg a Terminal programot és lépünk be a *test2.c*-t tartalmazó könyvtárba, majd adjuk ki a "*gcc test2.c -o test2*" parancsot annak érdekében, hogy lefordítsuk a programot, amit ezután a "*./test2*" parancssal tudunk futtatni.



```
Terminal
host% gcc test2.c -o test2
host% ./test2
Sum: 21
host%
```

03: LearnC

Stringek

String formázás

Jogosan merül fel a kérdés, hogy mit jelent az iménti kódban a `%i`? A PHP-ban és más szkript nyelvekben egyszerűen beilleszthetjük a változó nevét a dupla idézőjelek közé, és annak értéke automatikusan beillesztődik:

```
$var1 = 3;
$var2 = 8;
print ("Első érték: $var1 második érték: $var2");
```

A C-ben erre nincs lehetőség! Egy *formázó stringet* kell használni és egy a változó típusának megfelelő jelölőt:

```
int var1 = 3;
int var2 = 8;
printf ("Első érték: %i második érték: %i", var1, var2);
```

Jelölő formázás	
int	%i / %d
unsigned int	%u
float	%f
char	%c

A formázó string az idézőjelek közötti rész. A `%` jelölőt oda rakjuk, ahová a változó értékét szeretnénk beilleszteni.

A százalék jel mögé a kiírandó változó típusának megfelelő karaktert kell rakni. Most a `%i`-t használtuk, mert egy *int* típusú változó értékét írtuk ki.

A formázó stringet egy vessző követi és a jelőknek megfelelő változónevek következnek, szintén vesszővel elválasztva. Tehát a lényeg az, hogy a változónevek a záró idézőjel és a vessző után következnek.

A Cocoa programok az *NSLog* utasítást használják a *printf* helyett, de a formázási szabályok nagyjából ugyanazok.

Típus konverzió (casting)

Néha szükségünk lehet egy *változó típusának egy másik típusra való konverziójára*. Például van egy float típusú változónk, amit egy int típusú függvényben szeretnénk használni.

Néha előfordul, hogy nyugodtan használhatsz egy int típust float helyett és a rendszer automatikusan átkonvertálja azt hibaüzenet nélkül, de az a biztos, ha mi magunk végezzük el a konverziót. Ezt nevezzük *casting*-nak.

Íme egy példa:

```
int multiply (int x, int y)
{
return x * y;
}

int trips = 6;
float distance = 4.874;
int approxDistance = (int)distance;

int total = multiply ( trips, approxDistance );
```

A casting itt lényegében azért valósul meg, hogy a kívánt új típust zárójelek között az érték elé írjuk.

A casting eredménye változatos lehet attól függően, hogy mit és hova konvertálunk. Ha egy float típust konvertálunk int típusba, akkor a tizedes pont utáni részt a casting egyszerűen levágja. *A casting nem kerekíti a számokat.*

A casting *csak ideiglenes konvertálást végez*, a változó értékét nem változtatja meg! A fenti példában a *distance* továbbra is egy *float változó marad* a konverzió után is, tehát a minden egyes alkalommal el kell végezni a konverziót, ha ennek az egész részére van szükségünk.

További példák a casting-ra

Egy kicsit leegyszerűsíthetjük a fenti példát, ha kihagyjuk az approxDistance változót és a castingot a függvényhíváson belül végezzük el:

```
int result = multiply (trips, (int)distance);
```

Néha szükség lehet arra is, hogy egy függvény visszatérési értékét kell más típusra konvertálni. Ezt is hasonlóan oldhatjuk meg:

```
int multiply (int x, int y)
{
return x * y;
}

float eredmény;
eredmény = (float) multiply (3, 6);
```

A függvény definíciójából azt látjuk, hogy a *multiply* visszatérési értéke int.

Viszont az eredményt valós típusként szeretnénk tárolni, ezért amikor meghívjuk a függvényt a *(float)* konverziót használjuk a függvény neve előtt. Ezért az eredmény változó a következő számot fogja tartalmazni: **18.0**.

04: LearnC

Header fájlok

Header fájlok

Már láttuk, hogy a C nyelvben minden függvény még az előtt kell deklarálni mielőtt hivatkozunk rá a programban, ezért nagy segítséget jelentene, ha *ezeket a függvény definíciókat egy csoportban láthatnánk* és egy helyen tudnánk menedzselni őket. Éppen erre szolgál a header (fejléc) fájl.

A header fájlok különösen hasznosak nagy projektek esetében, mivel egy könnyen átláthatóvá teszik a program szerkezetét anélkül, hogy minden egyes soron át kellene rágni magunkat.

Egy header fájl felépítése

Bemutatok egy header fájlt. Másold be ennek a tartalmát a következő fájlba: *math_functions.h*

```
math_functions.h
```

```
int sum (int x, int y);  
float average (float x, float y, float z);
```

A *math_functions.c* nevű fájlba rakjuk a függvény implementációkat.

```
math_functions.c
```

```
int sum (int x, int y)  
{  
    return (x + y);  
}  
  
float average (float x, float y, float z)  
{  
    return (x + y + z) / 3;  
}
```

Az *average* függvényben a három összeadandó értéket zárójelbe tettük. Ez fontos, mert ellenkező esetben csak az utolsó lett volna 3-mal osztva. Mindig hasznos, ha zárójelek segítségével csoportosítjuk az elvégzendő műveleteket.

A header fájlok használata

Most felépítünk egy programot, amelyik használja a header fájlban definiált függvényeket. Hozzuk létre a *test3.c* fájlt:

```
test3.c

#include
#include "math_functions.h"

main ()
{
int theSum = sum (8, 12);
float theAverage = average (16.9, 7.86, 3.4);

printf ("the sum is: %i ", theSum);
printf ("and the average is: %f \n", theAverage);
printf ("average casted to an int is: %i \n", (int)theAverage);
}
```

A már megszokott include utasítás (stdio.h) mellett, beemeljük a *math_functions.h* -t is.

Ez biztosítja a program számára, hogy használhassa a *sum* és az *average* függvényeket.

A main függvény belsejében meghívjuk a sum függvényt és az eredményt egy int változóban tároljuk. Ezután meghívjuk az average függvényt és az eredményt egy float változóban tároljuk.

A program a háromszor is meghívja a printf utasítást. Egyszer a *%i* szimbólumot használja a *theSum* int változó értékének a kiiratásához, aztán a *%f* szimbólumot a *theAverage* float változó értékének a kiiratásához, és végül ismét a *%i* szimbólumot annak érdekében, hogy a *theAverage* float változót konvertálja int-be.

A stdio.h kisebb-nagyobb zárójelk között van, mivel ez a C könyvtár része (erről majd később még bővebben szólnunk). A math_functions.h fájl viszont csak ennek a programnak a része, ezért ennek a nevét idézőjelek közé tettük.

Fordítsuk le a példa programot

Három fájlunk van, ezeknek mind ugyanabban a könyvtárban kell elhelyezkedniük:

math_functions.h - a matematikai függvények deklarációi

math_functions.c - a matematikai függvények implementációi

test3.c - az aktuális program kód

Vegyük elő ismét a Terminál programot, lépünk be a fenti három fájlt tartalmazó könyvtárba és fordítsuk le a programot a következő utasítással: `gcc test3.c math_functions.c -o test3`.

A programot a `./test3` paranccsal tudjuk futtatni.

```
Terminal
host% gcc test3.c
math_functions.c -o test3
host% ./test3
the sum is: 20 and the
average is: 9.386666
average casted to an int is:
9
host%
```

Ez alkalommal a gcc programnak két input fájlt adunk meg: *test3.c* és *math_functions.c*. A gcc program összefűzi e két fájl .c tartalmát egyetlen programba. Érdekes megfigyelni, hogy a *math_functions.h* header fájlt nem kellett külön megadni a fenti gcc parancsban, az include parancsban felhívtuk rá a gcc program figyelmét.

05: LearnC Struktúrák

Struktúrák

A struktúrák *változók strukturált csoportjai*. Mutatok egy példát egy hangfájlokkal kapcsolatos információk tárolására szolgáló struktúrára.

```
typedef struct {
int lengthInSeconds;
int yearRecorded;
} Song;
```

```
typedef struct {
int lengthInSeconds;
int yearRecorded;
} Song;
```

```
song1
lengthInSeconds: 213
yearRecorded: 1994
```

```
song2
lengthInSeconds: 248
yearRecorded: 1988
```

Úgy látszik mintha két int változót deklarálnánk itt, de valójában egy új típusú változót definiálunk. A *typedef* utasítás hozzárendel egy nevet a struktúrához, jelen példában ez a: *Song*.

Mindegyik *Song* típusú változó, amelyiket a későbbiekben definiálni fogunk két értéket fog tartalmazni: *lengthInSeconds* és *yearRecorded* (tehát a dal hosszát másodpercekben, illetve rögzítés évét). Jelen példában mindkét mező int értéket tartalmaz, de egy struktúrában bármelyik mező tetszőleges típusú lehet, sőt akár egy másik struktúra is.

Miután definiáltunk egy struktúrát, ugyanúgy használhatjuk azt, mint az alapvető int, float, vagy char típusokat. Akármennyi *Song* változót létrehozhatunk, mindegyik tartalmazni fogja a megfelelő hossz értéket és évszámot. A struktúra valamelyik mezőjéhez a pont szintaktika segítségével tudunk értéket rendelni:

```
Song song1;
```

```
song1.lengthInSeconds = 213;  
song1.yearRecorded = 1994;
```

```
Song song2;
```

```
song2.lengthInSeconds = 248;  
song2.yearRecorded = 1998;
```

Láthatjuk, hogy létrehoztunk egy *Song* típusú *song1* nevű változót és a pont szintaktika segítségével beállítottuk a hosszúságot és az évszámot. A *song2* változó is *Song* típusú, de a megfelelő mezők más értékeket tartalmaznak.

Struktúrák a függvények belsejében

A függvények kimenő, vagy bemenő struktúrákat is képesek meghatározni. Ezeket a függvény deklarációkat és magukata struktúrákat is egy header fájlba írhatjuk.

Másoljuk be a következő kódot a *song.h* nevű fájlba:

```
song.h
```

```
typedef struct {  
int lengthInSeconds;  
int yearRecorded;  
} Song;
```

```
Song make_song (int seconds, int year);  
void display_song (Song theSong);
```

Ebben a header fájlban a *Song* struktúrát és a következő két függvényt deklaráltuk: *make_song* és *display_song*. Még a metódus implementációkat kell elkészíteni.

Másoljuk be a következő kódot a *song.c* nevű fájlba:

song.c

```
#include
#include "song.h"

Song make_song (int seconds, int year)
{
    Song newSong;

    newSong.lengthInSeconds = seconds;
    newSong.yearRecorded = year;
    display_song (newSong);

    return newSong;
}

void display_song (Song theSong)
{
    printf ("the song is %i seconds long ", theSong.lengthInSeconds);
    printf ("and was made in %i\n", theSong.yearRecorded);
}
```

A *make_song* függvény két int bemeneti értéket vár és egy *Song* struktúrát ad vissza. A *display_song* függvény egy *Song* struktúrát vár bemenetként és megjeleníti annak az értékeit. Figyeljük meg, hogy a *make_song* függvény meghívja a *display_song* függvényt amikor egy új *Song*-ot készít.

A *song.h*-t be kell illeszteni a *song.c*-be mivel a függvények használják a *Song* típust. Fontos: a *song.h* header fájlt minden olyan fájlba be kell illeszteni, amelyik valamilyen módon használja a *Song* struktúrát.

Struktúrák használat közben

Most készítünk egy programot, amelyik használja a *song.h* és a *song.c* fájlokat.

Másoljuk be a következő kódot a *test4.c* nevű fájlba:

test4.c

```
#include
#include "song.h"

main ()
{
    Song firstSong = make_song (210, 2004);
    Song secondSong = make_song (256, 1992);

    Song thirdSong = { 223, 1997 };
    display_song ( thirdSong );
}
```

```
Song fourthSong = { 199, 2003 };  
}
```

Ebben a programban két Song-ot készítünk a `make_song` függvény segítségével.

A harmadik Song már nem a `make_song` segítségével készül, hanem egy másik módszerrel, ahol a vesszővel elválasztott értékek kapcsos zárójelek közé kerültek. Mivel nem hívtuk meg a `make_song` függvényt, ezért a `display_song` függvény nem lesz automatikusan meghívva, nekünk kell azt külön meghívni.

A *negyedik Song* a harmadikhoz hasonlóan készül, de mivel itt egyáltalán nem hívjuk meg a `display_song` függvényt *az adatai egyáltalán nem fognak megjelenni a Terminalban*.

Látható, hogy jobb ha használjuk a függvényt amikor egy új struktúra példányt hozunk létre, mivel akkor jobban kézben tudjuk tartani a folyamatot. A fenti példában ebben az esetben automatikusan megjelennek az új Song adatai.

Fordítsuk le a példa programot

Fordítsuk le a `song.h`, `song.c` és a `test4.c` fájlokat és figyeljük meg az eredményt.

A Terminalban lépünk be abba a könyvtárba, ahol a fenti három fájl található és adjuk ki a "`gcc test4.c song.c -o test4`" parancsot.

A "`./test4`" parancs segítségével futassuk a programunkat.



```
Terminal  
host% gcc test4.c song.c -o  
test4  
host% ./test4  
the song is 210 seconds long  
and was made in 2004  
the song is 256 seconds long  
and was made in 1992  
the song is 223 seconds long  
and was made in 1997  
host%
```

06: LearnC

Konstansok, enumok, könyvtárak

Konstansok

Egy változó értéke változhat a program futása közben. Ezzel ellentétben *egy konstans egyszer kap értéket* a deklaráció során, és ezt az értéket nem lehet megváltoztatni a program újraindításáig.

```
const float goldenRatio = 1.618;  
const int daysInWeek = 7;
```

Bármelyik alapvető C változó típusnak lehet konstans változata. Csak arra kell figyelni, hogy ilyen esetben a változó deklarációja során értéket kell adni neki.

Enum

Sok időt eltölthetnénk az enum-ok tanulmányozásával, de most csak a Cocoa szellemiségére koncentrálna a lehető legegyszerűbb módon tesszük ezt.

A következő példa egy a Cocoa NSString osztályából származtatott enum-ot mutat be, amelyik keresési opciókat definiál:

```
enum {  
    NSCaseInsensitiveSearch = 1,  
    NSLiteralSearch = 2,  
    NSBackwardsSearch = 4,  
    NSAnchoredSearch = 8,  
    NSNumericSearch = 64  
};
```

Az Apple a Cocoa-ban az enum-okat *egymással kapcsolatban levő konstansok csoportosítására használja*. Ezt gyakran használják bizonyos függvény módusok beállítására. A konstansok értéke egyáltalán nem fontos, csak a nevük érdekes.

Nézzünk egy példát az enum használatára. Nem érdekes ha nem érted, hogy ez a kód mit csinál, csak azt szemlélteti, hogyan lehet használni az enumot:

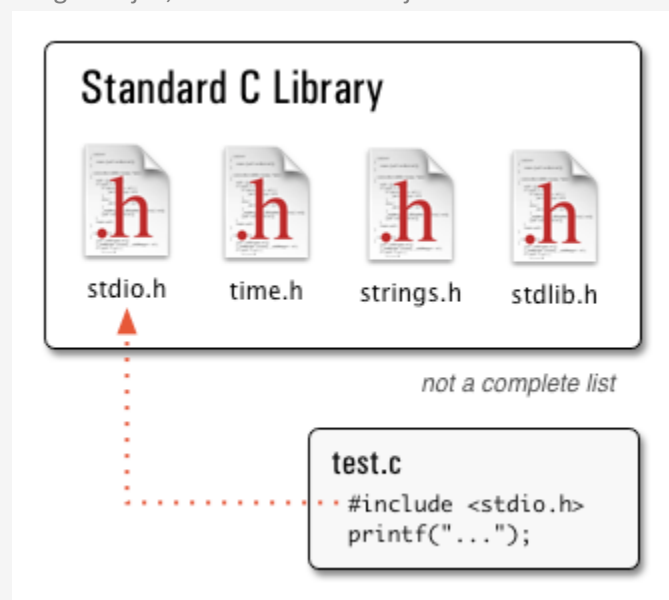
```
[string compare:anotherString options:NSCaseInsensitiveSearch];  
[string compare:anotherString options:NSLiteralSearch];
```

Könyvtárak

Mindegyik péda programba beemeltük a `stdio.h` fájlt, ami egy header fájl, a standard C könyvtárnak a része. A könyvtár *újrahasznosítható program kódok gyűjteménye*.

Ezek lehetnek függvények, struktúrák, konstansok, enumok.

Több ezer harmadik fél által készített C könyvtár létezik. Ezek között vannak ingyenesek és a hozzá tartozó `.c` fájlok is elérhetők mindenki számára. Vannak olyanok is amelyekért fizetni kell és nem a forráskódot szolgáltatják, hanem a header fájlokat és a dokumentációt.



Ha egy harmadik fél által készített könyvtárt szeretnénk használni, akkor le kell fordítanunk és be kell linkelni azt. Például ha egy olyan C programot szeretnénk írni, amelyik egy MySQL adatbázissal fog kommunikálni, akkor belinkelhetjük a `libmysql` könyvtárt és beemelhetjük a megfelelő header fájlokat a programunkba.

Ha mások által készített program kódot használasz, fontos figyelembe venni a felhasználási feltételeket. Néhány licenz például tartalmazza, hogy oszd meg a programod forráskódját.

Ha szereted a kalandozásokat, nézz körül az `/usr/include` könyvtárban, ahol a standard C header fájlok találhatóak.

Még egy utolsó példa

Végezetül készítünk egy olyan példa programot, amelyik a jelen cikkben bemutatott legfontosabb elemeket tartalmazza. Szükségünk lesz a következő korábbi fájlokra:

math_functions.h - matematikai függvény deklarációk

math_functions.c - matematikai függvény implementációk

song.h - a Song struktúra és a kapcsolódó függvények deklarációja

song.c - a Song függvények implementációja

Másoljuk példa programunk kódját a *final.c* fájlba:

`final.c`

```
#include
#include "math_functions.h"
#include "song.h"

main ()
{
    const int numberOfSongs = 3;
    printf ("total number of songs will be: %i\n", numberOfSongs);

    int i;
    for (i = 0; i < numberOfSongs; i++) {
        printf ("loop trip %i ", i);
    }

    printf ("\n");

    Song song1 = make_song (223, 1998);
    Song song2 = make_song (303, 2004);
    Song song3 = { 315, 1992 };

    display_song (song3);

    int combinedLength = sum (song1.lengthInSeconds, song2.lengthInSeconds);
    printf ("combined length of song1 and song2 is %i\n", combinedLength);

    float x = (float) song1.lengthInSeconds;
    float y = (float) song2.lengthInSeconds;
    float z = (float) song3.lengthInSeconds;

    float averageLength;
    averageLength = average (x, y, z);

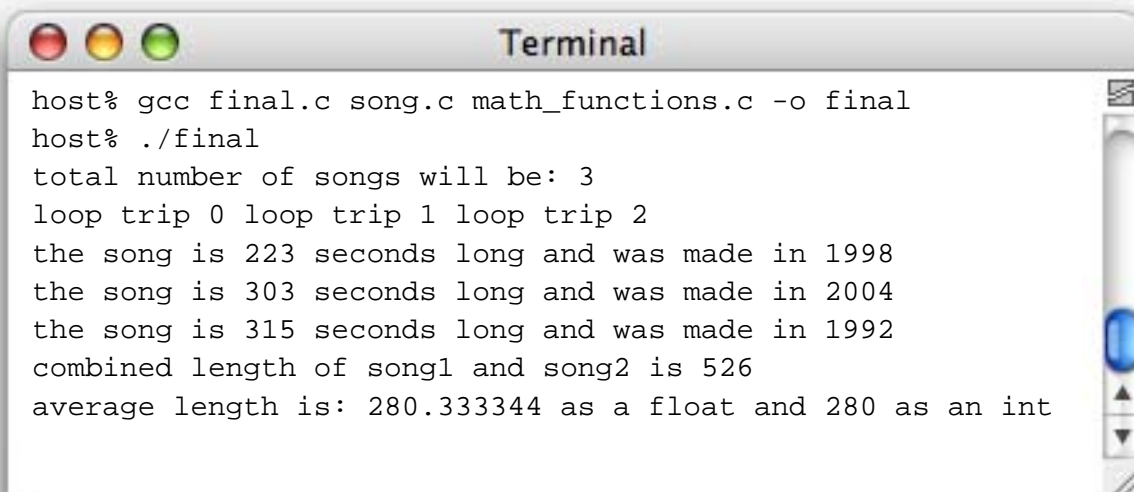
    printf ("average length is: %f as a float ", averageLength);
    printf ("and %i as an int\n", (int) averageLength);
}
```

Ez a program első pillantásra kicsit bonyolultnak tűnhet, de valójában csak ugyanazon a koncepciók ismétlése. Ha ez a példa nem teljesen érthető, kérlek olvasd át figyelmesen az elmondottakat.

Fordítsuk le a példa programot

Győződj meg róla, hogy imént említett négy fájl, valamint a *final.c* mindegyike ugyanabban a folderben van és a Terminal programban lépünk be ide. Adjuk ki a következő parancsot: `gcc final.c song.c math_functions.c -o final`.

A `./final` paranccsal futtassuk a programot!



```
host% gcc final.c song.c math_functions.c -o final
host% ./final
total number of songs will be: 3
loop trip 0 loop trip 1 loop trip 2
the song is 223 seconds long and was made in 1998
the song is 303 seconds long and was made in 2004
the song is 315 seconds long and was made in 1992
combined length of song1 and song2 is 526
average length is: 280.333344 as a float and 280 as an int
```

Utószó

Elég nagy területet sikerült összefoglalni viszonylag röviden. Ha pontosan érted az utolsó példát, akkor nyugodtan továbbléphetesz az Objective-C és a Cocoa tanulmányozásához.

Fordítás: Major Zoltán

Pdf: <http://alkalmazasfejlesztés.blogspot.com/>

2011-05-04