Az első C kódom: Hello World!

Ez a bevezetés lépésről lépésre vezeti le, hogyan készítünk kezdő programot a C programozási nyelven. Minden lépés egyben egy feladat is a hallgató számára, melyet érdemes leírnia, fordítania és futtatnia a saját gépén. A lépések után pedig ismertetésre kerül, hogy mit miért csinálunk.

0. lépés: Nyissunk egy Linux terminalt.

1. lépés: Hozzunk létre a “touch” utasítással egy helloworld.c elnevezésű file-t.

Megjegyzés:

- Ez egy 0 byte méretű szöveges dokumentum, “ls -l” utasítással látható.
- A .c kiterjesztés a C programozási nyelv sajátja, a fordítók ezt várják mint bemeneti file.

2. lépés: Hívjuk meg a Linux C fordítóját a “gcc helloworld.c” utasítással.

Megjegyzés:

- A gcc a Linuxon általánosan elérhető alkalmazás, de nem az egyetlen fordító. Több fordító is létezik, Windows alatt az MSVC az alapértelmezett, Macintoshokon a clang az általános, de ezen népszerűbb platformokon bátran használhatjuk bármelyik másikat is (OK, MSVC nem ajánlott más platformon mint a Win).
- A gcc Az Etalon fordító, a nyelvi fejlesztéseket nagyon követő alkalmazásról van ugyanis szó, és nem is tér el tőle (MSVC-ről határozottan ugyanezek nem mondhatóak el).
- Számos kapcsolóval módosíthatjuk a futását (lásd “man gcc”), például a “-Wall” a legtöbb esetben javasolt (az ImProg kurzuson a gyakorlatvezetők általánosan használják is).

Hiba:

- A gcc a terminalra hibaüzenettel tér vissza! Ugyanis minden C programban (sőt, tetszőleges programozási nyelven írt alkalmazásban is) szükséges egy belépési pont, ami a hívónak a tudtára adja a program futásának kezdőpontját. A hívó többnyire az operációs rendszer vagy valamilyen virtuális gép (pl. Java esetében).
- Ez a belépési pont a C nyelvben a main() függvény.

3. lépés: Nyissuk meg a helloworld.c filet egy tetszőleges szövegszerkesztőben (mcedit, vim, stb.).
Írjuk a file-ba a következő sorokat:

```
main() {}
```

Mentsük a változásokat, lépjünk ki, majd ismét hívjuk meg a “gcc helloworld.c” utasítást.

Megjegyzés:

- A gcc sikeresen lefut a kódunkra, de figyelmeztetéssel tér vissza. Az “ls” utasítással megjelenik egy *a.out* futtatható állomány, melyet el is indíthatunk “./a.out” utasítással a terminalról. Ez valójában az első sikeres C programunk (mégha nem is csinál semmit)!
- A *main()* ugyanis egy függvény, aminek van visszatérési típusa (bővebben a függvények leírásánál a megfelelő gyakorlaton). A visszatérési típus lehet beépített típus, felhasználói típus (bővebben a típusokról a változókról tartott gyakorlaton), de akár visszatérési érték nélküli függvény is, ezt “void” kulcsszóval jelezzük a függvény előtt.

- A *{}* kapcsos zárójelek jelentik a függvényen belül végrehajtandó utasítások kezdeti- és végpontját. Ez minden függvényre ugyanúgy néz ki, nem csak a main() függvényre, azaz tetszőleges függvény definíciójának alakja:

  visszatérési típus függvénynév(bemeneti paraméterek) { utasítások }

- Mivel minden függvénynek kötelezően van visszatérési típusa, amennyiben nem adjuk meg azt, úgy a fordító az alapértelmezett *int* típust adja neki implicit. A függvények visszatérési típusának nem-megadása nem elfogadott programozói gyakorlat, mindig adjunk visszatérési típust.

  Általánosan elmondható szabály, hogy ne bízzuk magunkat a fordítóra, a programozó feladata, hogy minden folyamatot, utasítást tudatosan meghatározzon egy programon belül, pl. jelen esetben az alapértelmezett visszatérési típusnál.

4. lépés: Nyissuk meg a file-t és egészítsük ki a *main()* függvényt az *int* visszatérési típusával.
A kódunk most:

```
int main() {}
```

Mentsünk, lépjünk ki, fordítsuk le megint a kódot a gcc-vel.
Megjegyzés:

- A gcc most hibajelzés nélkül lefordul.
- A *main()* függvény visszatérési értéke a hívó fél számára elérhető és felhasználható. Mivel nem adjuk meg a visszatérési értéket, ezért az alapértelmezett 0-val fog visszatérni.

Haladó:

- Módosítsuk a *main()* függvényt úgy, hogy ne térjen vissza értékkel, azaz legyen *void* a *main()* függvény előtt az *int* helyett. Fordítsuk le “gcc -Wall” utasítással. Ekkor szintén kapunk egy figyelmeztetést, mivel a main függvény javasolt visszatérési típusa *int*, hogy a hívó azt fel tudja használni. Ettől függetlenül a futtatható *a.out* file létrejön és elindítható.

4. lépés: Adjunk meg visszatérési értéket a függvénynek, ezt a r*eturn* kulcsszóval tudjuk megtenni.
A kódunk most:

```
int main()
{
        return 0;
}
```

Mentsünk, lépjünk ki, fordítsuk le megint, indítsuk el a futtatható állományt.
Megjegyzés:

- Minden nem-void visszatérési típusú függvénynek van *return* utasítása, amivel a függvény visszatér a hívóhoz a *return* utasítás mögött megadott értékkel. Ha ezt nem adjuk meg, akkor az alapértelmezett visszatérési érték 0 lesz a visszatérési típusnak megfelelően (bővebben a típusoknál). Ez a már említett programozói gyakorlattal szembemegy, vagyis a *return* nem-megadása a nem-void függvényekben valójában egy rejtett utasítás lesz, amit a fordító oda fog tenni. Azaz, mindig legyen *return* utasítás, ha van visszatérési típusa a függvénynek, a megfelelő értékkel.
- A *main()* függvény 0 visszatérési értéke jelzi a hívó félnek (operációs rendszernek) a futás sikerességét, ha ettől eltér, akkor általában valamilyen hibás futást jelzünk a hívó felé.

- A ; karakter kötelező utasításlezáró karakter, gondoljunk rá úgy, mint a mondat végi írásjelre. Hiánya a fordítóknál hibát okoz, amit jelezni is fog és nem lesz sikeres a fordítási művelet.
- A kapcsos zárójelek áthelyezése a kód olvashatósága miatt van, a fordító számára mindegy, hogy ömlesztve írjuk a kódot, vagy tördeljük.

5. lépés: Írjuk a kódunk elejére a következő sort:

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

Mentsünk, lépjünk ki, fordítsuk le megint, indítsuk el a futtatható állományt.

Megjegyzés:

- A *#include* egy speciális utasítás, úgynevezett makró, mely a fordító előfeldolgozó rendszerének szól. A fordítás folyamata valójában összetett, három lépésből áll: előfeldolgozás (preprocessing), fordítás (compiling), összefűzés (linking).
- A *#include* utasítás az előfeldolgozót arra utasítja, hogy az *stdio.h* fejlécállomány teljes tartalmát másolja a helloworld.c forráskódunk elejére.
- A fejlécállomány, azaz header file-k (*.h* kiterjesztésükkel jelzik header mivoltukat) a C forráskód szétbontásában játszanak szerepet, a modularizálás egy fontos eszköze, mely nagyobb projektek esetén az átláthatóságot könnyíti (bővebben a modularizálás tárgyalásakor).
- Az *stdio.h* a standard könyvtár (library) egy része, ezek előre megírt függvényeket, konstans változókat, makrókat tartalmaznak, melyeket akár mi is megírhatunk.
- Több makró is létezik, bővebben a makrók szekcióban foglalkozunk velük.
- A makrók meghívása nem csak a kód elején lehetséges, de tetszőleges helyen is. Az *#include <stdio.h>* is áthelyezhető és nem okoz (a jelenlegi kóddal legalábbis) hibát (nem az include makró áthelyezése fogja okozni a gondot, hanem a függvény deklarációjának hiánya, amikor is meghívnánk a függvényt).
  De átláthatóság, karbantarthatóság miatt mégis minden include makrót a kód elejére írjunk!
- Egy header file include-ja tulajdonképp másolást jelent, azaz ez a forráskód méretének növekedését eredményezi, mielőtt a tulajdonképpeni fordítás megkezdődne, valamint a másolás természetesen teljes fordítási idő növekedését is maga után vonja. Tehát kifejezetten rossz programozói hozzáállás, ha minden elérhető fejlécet include-olunk, amire nincs szükségünk!

6. lépés: Hívjuk meg a *printf(...)* függvényt az alábbi módon a *main()* függvényen belül, a *return* utasítás előtt. A teljes kódunk most ebben a formában van:

```
#include <stdio.h>

int main()
{
   printf("Hello World!\n");
   return 0;
}
```

Mentsünk, lépjünk ki, fordítsuk le megint, indítsuk el a futtatható állományt. A terminalon megjelenik a *"Hello World!"* üzenet.

Megjegyzés:

- A *printf* f az *stdio.h*-ban található, már implementált függvény. Az *stdio.h* headerben találhatóak a ki- és bemeneti (input-output, röviden io) műveletek.

- Teljes leírása: https://en.cppreference.com/w/c/io/fprintf
- A pontos szintaktikája: *int printf (const char *, ...);*
  Azaz *int* típusú a visszatérési értéke, printf a függvény neve, a bemeneti argumentumja egy konstans karakterláncra mutató változó, ami a mi esetünkben a *"Hello World!"*, valamint a … részen további argumentumok találhatóak (valójában a *printf* a függvények bemutatására nem a legjobb választás, ugyanis a függvények egy speciális csoportjába, a *variadic* függvények közé tartozik, ahol tetszőleges sok argumentum megadható a hívó fél részéről és az implementációnak ezt kezelnie is kell tudni, lásd https://en.cppreference.com/w/c/variadic).
  Későbbi anyagban részletesen foglalkozunk a *printf* függvény használatával, mivel az egyik leggyakrabban használt standard könyvtári függvény.

Ezzel sikeresen megírtuk első C programunkat!

+1. lépés: Másoljuk le a *printf* utasítást ugyanabba a sorba (a fordító számára mindegy, hogyan tördelünk). Ahogy említve volt, egy utasítás nézhető egy mondatnak is, és mint minden mondatnak, ennek a végén is van a mondatot lezáró írásjel, a pontosvessző karakter. A két *printf* utasítás között is szükséges ilyen karakter, azaz a kód helyesen így néz ki:

```
printf("Hello World!\n"); printf("Hello World!\n");
```

Fordítsuk le ezt a kódot és futtassuk. Természetesen kétszer lesz kiírva a *"Hello World!"*.
Most cseréljük ki a két *printf* közötti pontosvesszőt egyszerű vesszőre:

```
printf("Hello World!\n"), printf("Hello World!\n");
```

Fordítsuk le ezt a kódot és futtassuk. A kód ugyanúgy sikeresen fordul és ugyanazt írja ki!
Megjegyzés:

- Ahogy a pontosvesszőt a mondat végét lezáró karakternek is nézhetjük, úgy a vessző a két vagy több utasításból álló sorozat elválasztó karaktere. Pontosan ugyanúgy, mint az összetett mondatokban a tagmondatokat elválasztó vessző.
- A *return* utasítás esetében a következő kód fordítási hibát ad:

```
printf("Hello World!\n"), return 0;
```

A *printf* (vagy bármilyen más utasítás) a *return* után irandó és helyesen így néz ki:

```
return printf("Hello World!\n"), 0;
```

Próbáljuk meg mind a két esettel fordítani a kódot!