# 3. gyakorlat

## Nem lesz pluszminusz

Mert elfelejtett nyomtatni.

## Fahrenheit Celsius konverzió

Standard outputra írjuk.

#include "stdio.h"

int main**(){**

 #define LOWER 0

 #define UPPER 200

 #define STEP 40

 int fahr**;**

 **for(**fahr**=**LOWER**;**fahr**<=**UPPER**;**fahr**+=**STEP**){**

 fprintf**(**stdout**,**"F = %d\tC=%f\n"**,**fahr**,**5.0**/**9**\*(**fahr**-**32**));**

 **}**

 **return** 0**;**

**}**

Fordítása: $gcc fahr.cpp -o fahr.exe$

## Kitérők a feladat során

**continue**, **break** nem jó ciklusszervezés. Ne használjuk.

Mekkora a mérete egy karakternek? Fordító implementációjától függ. **sizeof(**char**);** Nem mindenhol 1 byte. Biztosan lesz ilyen kérdés a beugróban.

5**/**9**==**0, mert egész osztást végez a fordító.

Explicit type casting: **(**float**)**5 Nem jó, mert elveszik a kódban.

Ennél jobb az 5.0 vagy 5.

Ha fprintf-nek %d-be adunk float-ot, nem fog helyesen működni. %f kell helyette.

Ciklus initialization részében nem lehet változót deklarálni:
**for(**int i**=**0**;**i**<**10**;**i**++){...}** Nem jó. (C++-ban jó.)

int i**;** így jó, két…

**for(**i**=**0**;**i**<**10**;**i**++){...}** sorban.

### Blokk

Kapcsos zárójelekkel tetszőleges helyen nyitható új blokk. Az **if** definíció szerint egy utasítást vár.

Különböző blokkok lokális változói átfedhetik egymást.

Blokkon belüli kitakart globális változó elérhető a scope operátorral.

int var**;** //Globális

**{**

 int var**;** // Lokális. De "bent" nem lehet sehogy elérni

 **{**

 //"bent"

 int var**;** //"Még lokálisabb"

 **::**var //A legkülső, globális var-t lehet csak elérni.

 **}**

 //Újra el lehet érni az első lokális változót.

**}**

### Élettartam

Mettől meddig él a változó a memóriában.

Globális: A program futásának kezdetétől végéig.

Lokális: A befoglaló blokk végéig. A hatókörrel összekapcsolódik, de a hatókör nem időtartami fogalom, ne keverjük össze.

### Hatókör

Globális minden azonosító (változó/függvény), ami függvényen, névtéren és osztályon kívül bevezetett.

Lokális minden azonosító, ami nem globális. A $fahr$ a main függvényen belül van bevezetve, ezért nem globális, hanem lokális.

Hatókör: globális deklarációtól a forrásfájl végéig. Lokális a deklarációtól a befoglaló blokk (függvény/névtér/osztály) végéig.

### Deklaráció

* Nem kötelező minősítő (extern)
* Alaptípus (int)
* Deklarátor rész (i)
* Nem kötelező kezdeti értékadó kifejezés (**=**0)

Deklarátor rész után már megkezdődik a változó hatóköre. Tehát az int x**=**x**;** lefordul, bár nincs értelme.

Függvény deklarálása: int getHeight**();**

### Definíció

Függvény definíciója:

int getHeight**(){**

 //Ha megírjuk a törzsét (Jelen törzs nem csinál semmit, de az is definíció.)

**}**

### Példák

int x**;** //Definíció

char z**;** //Definíció

char**\*** c**;** //Definíció (Pointert veszünk föl, ami egy karakter címét tárolja. Ugyan akkora helyet vesz fel, mint egy int.)

extern int x**;** //Deklaráció (Itt csak bevezetjük, nem definiáljuk.)

class Car**;** //Deklaráció (Csak a felhasználói típus neve van bevezetve.)

int f**();** //Deklaráció (Üres törzsű függvény.)

Referencia, álnév, konstans pointer különleges eset.

### Láthatóság

(Az elérhetőség más.)

Bevezetett nevek átfedhetik, eltakarhatják egymást.

Deklarálni akárhányszor lehet, de csak ugyan azt a nevet ugyan azzal a típussal. Definiálni csak egyszer lehet: int x**;** int x**;** Nem fordul le, mert ez már definíció. A fordító már tudja, mekkora helyet kell lefoglalnia neki.

$extern$: Kiterjesztés. "Keresd máshol!" Ha van két forrásfájlunk: $a.cpp$, $b.cpp$, ki lehet terjeszteni az int x**;** hatókörét $a$-ról $b$-re: extern int x**;** segítségével. int x**;** nem elég.

Beégetett értékekkel nem érdemes dolgozni. Inkább:
#define LOWER 0
#define UPPER 200
#define STEP 40
és utána **for(**fahr**=**LOWER**;**fahr**<=**UPPER**;**fahr**+=**STEP**)**

C gondolkodás: Fusson minél gyorsabban, foglaljon minél kevesebb memóriát. C++ gondolkodás: néhány pazarló változóval a kód minősége jelentősen javul.

Fordítás előtt a preprocesszor kicserél minden #define-t.

Zárójelet paraméterezést tudja, következő órán megnézzük.

### A $\#define$ hátrányai

* Semmi ellenőrzés nincs.
* Nem változó, ezért nincs hatóköre, élettartama.
* Tart onnan ahol definiáltuk, amíg nem találkozunk $\#undef$-fel.
* Ha makróval behelyettesítjük, mindent végig helyettesit.
* Nem lehet debuggolni sem, fordító nem is latja
* Elveszik a kódban, nem látszik, ezért is írjuk csupa nagybetűvel