

# **Kimaxoljuk a programozást**

---

## **Programozás és robotpszichológia 12 alatti gyerekeknek**

Ed. ESAMU Book U12, Entrópia Samu, 2016. okt. 29, v. book.u12.hu.0.0.7.1

Copyright © 2016 Dr. Bátfai Norbert

Entrópia Samu Book U12

Copyright (C) 2016, Norbert Bátfai. Ph.D., batfai.norbert@inf.unideb.hu, nbatfai@gmail.com, Margaréta Bátfai, batfai.greta@gmail.com, Nándor Bátfai, batfai.nandi@gmail.com, Mátyás Bátfai, batfai.matyi@gmail.com

This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program. If not, see <http://www.gnu.org/licenses/>.

Ez a program szabad szoftver; terjeszthető illetve módosítható a Free Software Foundation által kiadott GNU General Public License dokumentumában leírtak; akár a licenc 3-as, akár (tetszőleges) későbbi változata szerint.

Ez a program abban a reményben kerül közreadásra, hogy hasznos lesz, de minden egyéb GARANCIA NÉLKÜL, az ELADHATÓSÁGRA vagy VALAMELY CÉLRA VALÓ ALKALMAZHATÓSÁGRA való származtatott garanciát is beleértve. További részleteket a GNU General Public License tartalmaz.

A felhasználónak a programmal együtt meg kell kapnia a GNU General Public License egy példányát; ha mégsem kapta meg, akkor tekintse meg a <http://www.gnu.org/licenses/> oldalon.

<http://gnu.hu/gplv3.html>

**COLLABORATORS**

	<i>TITLE :</i> Kimaxoljuk a programozást		
<i>ACTION</i>	<i>NAME</i>	<i>DATE</i>	<i>SIGNATURE</i>
WRITTEN BY	Bátfai, Norbert, Bátfai, Margaréta, Bátfai, Nándor, és Bátfai, Mátyás	2016. november 1.	

**REVISION HISTORY**

NUMBER	DATE	DESCRIPTION	NAME
0.0.1	2016-09-19	Iniciális dokumentum. A fejlesztés az ESAMU központi tárolójában történik: <a href="https://github.com/nbatfai/SamuEntropy">https://github.com/nbatfai/SamuEntropy</a> , verziótörténet tekintetében lásd az ottani changelog-ot.	nbatfai

# Tartalomjegyzék

<b>I. Szójegyzék</b>	<b>1</b>
<b>II. A jelen programozása</b>	<b>4</b>
<b>1. Bevezetés</b>	<b>6</b>
1.1. Jó kezdet	6
1.1.1. Első nap – Helló, világ!	6
1.1.1.1. Apa – mentor	6
1.1.1.2. Gréta	8
1.1.1.3. Nándi	8
1.1.1.4. Matyi	8
1.1.2. Második nap – for ciklus	8
1.1.2.1. Apa – mentor	9
1.1.2.2. Gréta	9
1.1.2.3. Nándi	9
1.1.2.4. Matyi	9
1.1.2.5. Feladat	9
1.1.2.5.1. Nándi	10
1.1.2.5.2. Matyi	11
1.1.2.5.3. Gréta	12
1.1.3. Harmadik nap – szorzótábla	13
1.1.3.1. Apa – mentor	13
1.1.3.2. Gréta	13
1.1.3.3. Nándi	13
1.1.3.4. Matyi	14
1.1.3.5. Feladat	15
1.1.3.5.1. Gréta	15
1.1.3.5.2. Nándi	15
1.1.3.5.3. Matyi	16

---

1.1.4.	Negyedik nap – osztály	16
1.1.4.1.	Apa – mentor	17
1.1.4.2.	Gréta	17
1.1.4.3.	Nándi	17
1.1.4.4.	Matyi	17
1.1.4.5.	Feladatok	18
1.1.4.5.1.	A gyerekek által kiírt feladatok	22
1.1.5.	Ötödik nap – veszélyes vizeken	23
1.1.5.1.	Apa – mentor	24
1.1.5.2.	Gréta	26
1.1.5.3.	Nándi	26
1.1.5.4.	Matyi	26
1.1.6.	Hatodik nap – másoló konstruktor	26
1.1.6.1.	Apa – mentor	26
1.1.6.2.	Gréta	27
1.1.6.3.	Nándi	27
1.1.6.4.	Matyi	27
<b>III.</b>	<b>A jövő programozása</b>	<b>28</b>
<b>2.</b>	<b>Bevezetés</b>	<b>30</b>
2.1.		30
<b>3.</b>	<b>Tárgymutató</b>	<b>31</b>

# Ajánlás

Ezt a könyvet az olvasóinak, a jövő robotpszichológusainak ajánljuk.

# Előszó

Ha 12 alatti gyerek vagy, akkor ezt az unalmas előszót kihagyhatod, nem neked szántuk, hanem a mentorodnak. Persze abból nem lehet baj, ha elolvasod, csak ne aludj el közben, mert egy olyan távolabbi jövőről szól, ami Neked most még lehet, hogy teljesen érdektelen.

A könyv két részből áll, az első – kevesek élvezetére számítva – C++ programozás, a jelen programozása. A második a jövő egy lehetséges programozási gyakorlata, amikor már nem klasszikus értelemben programozunk, hanem csak játszunk. Játék (esport) lévén itt lehet tömeges olvasótáborra számítani.

Egy hacker ismerősöm fejtette ki nekem egyszer, hogy a programozás mint tevékenység nem algoritmizálható, ezért nem is tanítható. Csak egy lehetőség van: fanatizálni az érdeklődőket, hogy programozzanak és a csoda magától megtörténik: némelyekből programozó lesz!

Ezt a hitvallást mi is magunkénak érezzük. Ám ebben a doksiban mégis megpróbáljuk programozni tanítani a 12 év alatti gyerekeket... C++ nyelven. Ezzel foglalkozunk az első részben, de a csodát igazán nem is ettől várjuk, hanem a második résztől, ahol bevezetjük a jövő – víziónk szerinti – programozását.

Mi ez a vízió?

Amikor gimnazista koromban megismerkedtem a számítógéppel<sup>1</sup> két kapcsolatom volt vele: programoztam és játszottam rajta. Ennek a két tevékenységnek volt közös része, amikor például egy 8x8 bites karaktert átdefiniáltam egy emberkére és az egy saját „játékomban” kolbászolt a képernyőn... de a játék és a programozás azért két különböző dolog volt akkor és az ma is.

De különböző lesz a jövőben is? Víziónk szerint lesznek területek, ahol nem! Ilyenek lehetnek például a mesterséges intelligencia területéhez tartozó különböző kognitív jellegű programozási feladatok, mint például a gépi látás, amikor a gép nem csupán feldolgozza a képeket, hanem valamilyen szinten olyan kérdésekre is az emberhez hasonlóan tud válaszolni, hogy mit ábrázol a kép!

Az ilyen rendszereket nem „from scratch” programozzuk majd, hanem kiindulunk egy általános megoldásból és azt finomítjuk. Ha ez a megoldás mondjuk TensorFlow<sup>2</sup> alapú akkor például a TensorBoard<sup>3</sup> adatfolyam szerkesztőjével...

S mi is az Entrópia Samu víziója? Húzzunk még egy újabb interfészt ezekre az adatfolyam vizualizációs eszközökre, egy olyat amely képes entrópia-csökkentő játék élményt adni... egyszerűen tegyük játékká a programozást! Az asimovi robotpszichológia programozói értelmezésében<sup>4</sup> ez a tevékenység robotpszichológiának tekinthető (vagy legalábbis annak csírájának, hiszen intuitíven a Westworld című filmben látható robotpszichót tekinthetnénk az asimovi vízió tökéletes, de természetesen ma még csak elképzelt megvalósításának).

Hogy csináljuk? Az első részt könnyebb, ez máris elkezdhető: a gyerekek (Gréta 8, Nándi 8 és Matyi 10 évesek) hazaérnek a mindennapi úszóedzésről olyan 6 óra után, vacsora majd minden hétköznap megbeszélünk egy C++ forrást és ki is próbáljuk persze, másnapra ezt a programot mindhárman leírják a saját szavaikkal, nekem elküldik mailben, mire én beteszem ebbe a könyvbe – ez legalábbis az ideális terv íve :)

A második rész már komplikáltabb lesz, mert ezt – Gábor Dénes után szabadon – előbb fel kell találni. Ezzel próbálkozunk meg a „Entropy non-increasing games for improvement of dataflow programming” című publikáció készítésével.

Következik a szójegyzék, amely még mindig inkább a mentorra van hangolva, mint a gyermek olvasóra.

---

<sup>1</sup> Commodore 64, Balassi Bálint Gimnázium Balassagyarmat, hála Fűrész István számíttech. tanáromnak, a programozást is megszerettem.

<sup>2</sup> <https://www.tensorflow.org/>

<sup>3</sup> <https://github.com/tensorflow/tensorflow/tree/master/tensorflow/tensorboard>

<sup>4</sup> <https://github.com/nbatfai/Robopsychology>

## **I. rész**

# **Szójegyzék**



## D

### Debreceni Egyetem [ DE ]

A DE a Debreceni Egyetem rövidítése.

### Developmental Robotics 2 Robopsychology [ DevRob2Psy ]

A DevRob2Psy a Developmental Robotics 2 Robopsychology nevű Facebook csoport rövidítése. Ez egy informális csoport, amely a címbeli témák iránt érdeklődőket tömöríti, tagjai főleg a DE hallgatói és oktatói. Ebben a csoportban kommentelheti a kedves olvasó ezt a doksit is: <https://www.facebook.com/groups/devrob2psy/>.

lásd még "UDPROG".

## E

### Entrópia Samu [ ESAMU ]

Az esport és a mesterséges intelligencia között nyilvánvaló a szoros kapcsolat, melyet a jelen kutatásunkban még tovább akarunk fokozni azzal, hogy nem csupán a játékokbeli MI felhasználásra gondolunk, hanem magának az MI-nek a megteremtését várjuk az új, fejlesztendő esporttól. Az aktuális elképzelés szerint ez egy szoftveresen implementált, a családi PC-n futó, mobil eszköz érzékszervekkel ellátott fejlődésrobotikai és robotpszichológiai Samu alkalmazás lesz: az Entrópia Samu, <https://github.com/nbatfai/SamuEntropy>.

lásd még "Samu".

## F

### Fejlődésrobotika [ DevRob ]

Olyan tudományos irányvonal, amely a testtel rendelkező robotok tanulását az emberi tanulás szemszögéből vizsgálja. Alaptézise, hogy a robot testtel rendelkezik, ennek antézise lehet a csak szoftveres robot, a kettő szintézisét a PSAMU beküldött kéziratban adtuk meg.

lásd még "Robotpszichológia".

## I

### IEEE Recommended Practice for Software Requirements Specifications (SRS) [ IEEE Std 830-1998 ]

A szoftverkövetelmény dokumentum [IEEE szabványa](#). Entrópia Samu szoftverkövetelmény doksját ennek a szabványnak megfelelően készítjük majd el.

## J

### JIBO

Cynthia Breazeal társasági robotja, lásd <https://www.jibo.com/>

---

## R

### Robotpszichológia

Az [Asimovi robotpszichológia](#) programozói szemszögből értelmezett implementálása, lásd <https://github.com/nbatfai/-Robopsychology>.

## S

### Samu

Egyszer a fejlődésrobotikus családi csevegőrobottal, Bátfai Samuval, másrészt általában a mesterséges intelligencia fejlődésrobotikai megközelítésével kapcsolatos kutatásaink összefoglaló neve. Lásd még a <https://prezi.com/utlu1bevq9j2/egy-testetlen-fejlodesrobotikai-agens/> prezit és a <https://arxiv.org/abs/1511.02889> illetve a PSAMU beküldött kéziratot.  
lásd még "[ESAMU](#)".

## U

### The Yearbook of the Programmers of University of Debrecen [ UDPROG ]

A Debreceni Egyetem [reguláris programozás oktatásához](#) kapcsolódó, LinkedIn-en szervezett [fejlesztői közösség](#), melynek egyik gyűjtőpontjában [szoftverek](#), a másikban maga az [évkönyv](#) áll.

## **II. rész**

# **A jelen programozása**

A programozás és a Linux két jó barát, kezd azzal, hogy mentoroddal felrántasz a gépedre egy Linuxot!

# 1. fejezet

## Bevezetés

### 1.1. Jó kezdet

Ha már van Linuxod, olvashatod tovább, ha még nincs lapozz vissza!

#### 1.1.1. Első nap – Helló, világ!

Ezzel a C++ nyelvű forrásszöveggel kezdünk (amit például Gréta a `gretal.cpp` állományba begépelve kapott meg).

```
#include <iostream>

int main()
{
    std::cout << "Hello, Vilag!" << std::endl;
    return 0;
}
```

##### 1.1.1.1. Apa – mentor

A programozóra is igaz, hogy általában óriások vállán áll<sup>1</sup>, azaz felhasználja más programozók programjait. A `#include` utasítás éppen ezt csinálja, beilleszti a mások által megírt `iostream` programokat. Ebben az `iostream`-ben csatornákat megvalósító programok vannak. Például lesz a programodnak egy olyan csatornája<sup>2</sup>, amibe ha a programod benyomja a betűket, akkor a betűk megjelennek a képernyőn.

A C++ program alapvető építőeleme a függvény ennek van egy első sora, ami nekünk most a `int main()` sor. Ez három részből áll: megmondja mit fog kiszámolni a függvény: ez most egy számot, ezt jelenti az `int`. A második része a függvény neve, ez nálunk a `main`. A harmadik rész kerek zárójelek közé van zárva, ami itt van azt megkapja a függvény, ezekből kell számolnia. A mi esetünkben ez a rész üres, tehát semmit nem adunk meg neki bemenetként.

A függvény törzse a blokk, ami utasítások kapcsoszárójelek közé zárt listája. Minden utasítás egy pontosvesszővel van lezárva. Most két utasításunk van. Az első az `std::cout` nevű kimenő csatornára kinyomja a "Hello, Vilag!" szöveget. Ennek a csatornának a bemenő nyílása van a proginkban, a kimenője pedig a képernyőre nyitva. Aztán kinyomunk még egy speciális jelet, ami a egy új sorba ugratja a képernyőn a kurzort.

<sup>1</sup> A nagy tudós Isaac Newton így fejezte ki (Robert Hooknak írt levelében) hogy ő is tudott mások munkájára alapozni: "If I have seen further it is by standing on the shoulders of Giants" avagy körülbelüli fordításban: azért láthattam messzebbre, mert óriások vállán álltam.

<sup>2</sup> Honnan van ez a csatorna? A gépeden (de például a telefonodon vagy a tableteden is) van egy különleges program, amelynek a neve operációs rendszer, a Linux kernel, amit egyébként még egyetemi diák korában kezdett el írni Finnországban Linus Torvalds. Az operációs rendszer a programozó jóbarátja, ez a program biztosítja például a szóban forgó csatornákat is.

A második utasítás a `return` ami visszaadja a függvény meghívójának a függvény által kiszámolt értéket. Most igazából nem számoltunk semmit, csak kiírtunk a képernyőre, ezért visszaadunk egy nullát. Ez a hagyomány szerint azt jelenti, hogy minden okés volt a programmal, mert a `main` függvényed hívója maga az operációs rendszer program volt, a Linux kernele, neki jelzett a progí, hogy nem volt gond a viselkedésével. Ezzel a `main` függvény végetért és ezzel a progink is befejeződött.

Most próbáljuk ki a programot!

Először lefordítjuk a forráskódot a saját gépünk procijának saját nyelvére. A proci számára érthető fordítást a `gretal` nevű állományba mentetjük el a `g++` fordítóprogival. Majd ennek a nevét a pont per után leírva futtatjuk a lefordított progit. A futtatás az a folyamat, amikor a proci olvassa el a lefordított programot.

Entert nyomva a pont per után írt állománynév után mi is történt? Végrehajtódott a `main` függvény, ami a kimenő csatornára kinyomta, hogy Helló, Világ! ami ennek megfelelően megjelent a képernyőn.

```
$ g++ gretal.cpp -o gretal
$ ./gretal
Hello, Vilag!
$
```

### A processzor nyelve

A C++ nyelvű forrásszövegeket a gépek nem értik, hanem a C++ programozók értik! A forráskódot le kell fordítani a processzorok saját nyelvére. Ezt a fordítást végzi el a `g++` fordító program<sup>a</sup>. Nem kell aggódnod, a `g++` program megvan a gépedre, a kezdetek óta segíti a Linuxot!

A mentorod megmutathatja, hogyan fest a nyelv, amit a processzor ért: számok. Íme például az első forrásprogidból fordított futtatható kód:



```
$ hexdump -v -e '/1 "%02X "' gretal | more
7F 45 4C 46 02 01 01 00 00 00 00 00 00 00 00 02 00 3E 00 01 00 00 00 50 07 40 00  ←
    00 00 00 00 40 00 00 00 00 00 00 00 00 40 1C 00 00 00 00 00 00 00 00 00 40 00 38
00 09 00 40 00 1F 00 1C 00 06 00 00 00 05 00 00 00 40 00 00 00 00 00 00 00 40 00 40  ←
    00 00 00 00 00 40 00 40 00 00 00 00 00 F8 01 00 00 00 00 00 00 F8 01 00 00 00 00
00 00 08 00 00 00 00 00 00 00 03 00 00 00 04 00 00 00 38 02 00 00 00 00 00 00 38...
```

... mert ez így megy még több képernyőn! Láthatod, hogy a proci nyelvét meg a C++ programozók nem értik.


<sup>a</sup> A `g++` fordítót olyan híres programozó írta akinek jól ismert hacker neve is van: RMS, azaz Richard Matthew Stallman. Ha álmodban a Gyűrűk urában találánád magad mint Frodó, akkor biztos, hogy ő lesz Gandalf :)

### A Linux programozói kézikönyve

Az imént a gretal állomány tartalmát néztük meg a **hexdump** paranccsal: `$ hexdump -v -e '/1 "%02X"' gretal | more` de ne gondold ám, hogy ilysmi ákombákomot fejből le tudok írni! Egyszerűen megnéztem a manuálban.

A C programozásban méginkább az a napi gyakorlat, hogy ha kell valamit használnod ott van a manuálban, meg kell nézni, hogyan teheted meg! Ez egy írott szakma, ami nem Istentől való, hanem mérnökök írták, ez az egyszerű titok: csak el kell olvasni. Nézd meg például, hogy mit ír a kézikönyv az `std::cout` objektumról:

```
std::cout(3)                                C++ Programmer's Manual                                std::cout ↔
(3)
```

 **NAME**

`std::cout` - Standard output stream

**TYPE**

object

**SYNOPSIS**

```
#include <iostream>

extern ostream cout;
```

**DESCRIPTION**

Object of class ostream that represents the standard output stream oriented to narrow characters (of type char). It corresponds to the C stream stdout.

...

#### 1.1.1.2. Gréta

A processzor számokat számol és az `std::cout`-ba írjuk a betűket. A g++ fordítja le a programot. Az `iostream` egy csatorna. A `#include` átilleszti a másik programját a sajátodba. A 0 azt jelenti, hogy jól vége lett a programnak, azaz jól ment a program le.

#### 1.1.1.3. Nándi

A g++ a processzor nyelvére fordít. A `#include` egy parancs, amely azt adja ki, hogy másold be. A `cout` valójában egy csatorna, amelyen betűk mennek át. A `<<` azt jelenti, hogy arra told a betűket. A 0 azt jelenti, hogy jól sikerült a main függvény.

#### 1.1.1.4. Matyi

A `#include` az első parancs. A `#include`-al beillesztünk más programokat. Az `iostream` egy csatorna. Az `int main` az egy fő függvény. A main függvényt az operációs rendszer hívja. A függvény azt jelenti, hogy a program építőeleme. `std::cout << Hello Vilag << std::endl` ez azt jelenti, hogy a betűket belenyomom a csatornába. Az `std::endl` azt jelenti, hogy a kurzor a következő sorba megy. A `return` visszaad a függvény hívójának egy számot. A `return 0;` ez azt jelenti, hogy nincs baj a programmal.

### 1.1.2. Második nap – for ciklus

Picit fejlesztünk az előző 1-es programunkon, így kapjuk majd ezt a 2-est:

```
#include <iostream>

int main()
{
    std::cout << "Hello Vilag!" << std::endl;
    for(int i=0; i<3; ++i)
        std::cout << i << std::endl;
    return 0;
}
```

#### 1.1.2.1. Apa – mentor

A `for` utasítással tehetünk ismétléseket, más névvel iterációkat a program működésébe. Az iteráció azt jelenti, hogy valamit (amit iterálunk) többször (akár nullászor, azaz egyszer sem vagy végtelenszer) végrehajtunk egymás után.

Általánosan így fest egy `for` `for(honnan indul;meddig megy;hogyan megy) {azt iteráljuk, ami itt van}`.

A 2-es forrásban egy egész számot `int i` léptetünk `i=0` értéktől indítva (inicializálva) és addig megyünk, amíg igaz, hogy `i<3`, azaz a `i=0, 1, 2` kis egész számokra fut le az `i` változó, sorrendben a 0, 1, 2 egymás alatti kiíratása.

#### 1.1.2.2. Gréta

A `for` egy ciklus. A `for` az `i=0`-tól indul és `i<3`-ig tart.

#### 1.1.2.3. Nándi

A `for` ismétli a ciklust. A `for` egy ciklus. Mit is jelent a ciklus? Nyomj egy fekvőtámaszt, utána 10 fekvőtámaszt. A 10 fekvőtámasz egy ciklus 1-től 10-ig.

#### 1.1.2.4. Matyi

A `for` egy ciklus. Azért ciklus, mert sokszor végrehajtja a programot. A `++i` azt jelenti, hogy eggyel növekszik. Például ha `i` egy akkor

1. `++i=2`
2. `++i=3`
3. `++i=4`.

Ha kapcsolószerű lenne a `for`-nál akkor több parancs lenne. Az `i=0` az, hogy honnan indul, az `i<3`, hogy meddig megy, a `++i`, hogy mennyivel növekszik.

#### 1.1.2.5. Feladat

Módosítsuk úgy a 2-es progit, hogy egymás alá négyszer írja ki a `Hello, világ!`-ot! A gyerekek időrendben a következő megoldásokat adták.



### 1.1.2.5.1. Nándi

A 2-es program helyett gyakorlatilag az 1-esből indult ki, mert a 2-esből a `for` ciklust törölte, majd betabulálva betett egy új `Hello, világ!`-ot az alábbiak szerint.

```
#include <iostream>

int main()
{
    std::cout << "Hello Vilag!" << std::endl;
    Hello Vilag!
    return 0;
}
```

amivel az alábbi hibát kapta:

```
$ g++ nandi2.cpp -o nandi2
nandi2.cpp: In function 'int main()':
nandi2.cpp:6:19: error: 'Hello' was not declared in this scope
    Hello Vilag!
```

Mivel a kiírandó szöveg betűi nem voltak idézőjelek között, ezért a fordító megpróbálhatta felismerni mint a program részét, de nem járt sikerrel. A következő próbálkozása már ugyan hiba nélkül lefordult

```
#include <iostream>

int main()
{
    std::cout << "Hello Vilag!" << std::endl;
    "Hello Vilag!";
    return 0;
}
```

de a progí így nem csinált semmit. Ezt a következő módosítás követte

```
#include <iostream>

int main()
{
    std::cout << "Hello Vilag!" << std::endl;
    std::cout << "Hello Vilag!";
    return 0;
}
```

majd ez, hogy legyen 4 `Hello, világ!`

```
#include <iostream>

int main()
{
    std::cout << "Hello Vilag!" << std::endl;
    std::cout << "Hello Vilag!";
    std::cout << "Hello Vilag!";
    std::cout << "Hello Vilag!";
    return 0;
}
```

végül hogy legyen soremelés is következett egy jó megoldás

```
#include <iostream>

int main()
{
    std::cout << "Hello Vilag!" << std::endl;
    std::cout << "Hello Vilag!" << std::endl;
    std::cout << "Hello Vilag!" << std::endl;
    std::cout << "Hello Vilag!" << std::endl;
    return 0;
}
```

#### 1.1.2.5.2. Matyi

A 2-es programból indult ki, először a ciklus határát módosította 3-ról 4-re és az `i` helyett beírta a `Hello, világ!`-ot.

```
int main()
{
    std::cout << "Hello Vilag!" << std::endl;
    for(int i=0; i<4; ++i)
        std::cout << Hello Vilag! << std::endl;;
    return 0;
}
```

amire ezt a (korábban Nándinál már látott) hibát kapta

```
$ g++ matyi2.cpp -o matyi2
matyi2.cpp: In function 'int main()':
matyi2.cpp:7:22: error: 'Hello' was not declared in this scope
    std::cout << Hello Vilag! << std::endl;
```

erre visszaírta az `i`-t a kiíratásba és a ciklusváltozó növelését kivette:

```
#include <iostream>

int main()
{
    std::cout << "Hello Vilag!" << std::endl;
    for(int i=0; i<3; )
        std::cout << i << std::endl;
    return 0;
}
```

amivel végtelen ciklusba futott a programja a futtatáskor... csak szaladtak a `Hello, Világ!`-ok a terminál programban<sup>3</sup>, a végén már csak az ablak becsukása segített!



#### Matyi beszámolásában

Ha 1-től 4-ig `++i` akkor négyszer kiírja, hogy: `Hello Vilag`. Ha nem tesszük oda az `++i`-t akkor az lesz mint velem, a `Hello Világ`ot végtelen ciklusban végrehajtja.

A ciklusváltozó inkrementálását (növelését) visszarakva megkapott aztán már egy jó megoldást:

<sup>3</sup> A ciklusmag iterálása után az `i` növelését kivéve az mindig 0 marad, soha nem éri el a 3-at, azaz soha nem lesz vége a ciklusmag iterálásának. Ez bizony egy végtelen ciklus!

```
#include <iostream>

int main()
{
    std::cout << "Hello Vilag!" << std::endl;
    for(int i=0; i<3; ++i)
        std::cout << "Hello Vilag!" << std::endl;
    return 0;
}
```

#### 1.1.2.5.3. Gréta

Először Ő is a 2-es programmal indított és 1-re növelte a ciklusváltozó kezdőértékét

```
#include <iostream>

int main()
{
    std::cout << "Hello Vilag!" << std::endl;
    for(int i=1; i<3; ++i)
        std::cout << i << std::endl;;
    return 0;
}
```

majd fordítva és futtatva tanácstalanságában lényegében visszatért az 1-es progija hekkeléséhez, mivel a ciklust törölte, majd a Helló Világ után írt egy 4-est:

```
#include <iostream>

int main()
{
    std::cout << "Hello Vilag!"4 << std::endl;

    return 0;
}
```

amivel ezt a hibát kapta<sup>4</sup>

```
$ g++ greta2.cpp -o greta2
greta2.cpp: In function 'int main()':
greta2.cpp:5:32: error: expected ';' before numeric constant
    std::cout << "Hello Vilag!"4 << std::endl;
```

ezt javítandó ezzel próbálkozott

```
#include <iostream>

int main()
{
    std::cout << "Hello Vilag!" << std::endl;
    std::cout << std::endl;
    return 0;
}
```

ami már fordult, futott, igaz még hibásan:

---

<sup>4</sup> A fordítási hibaüzenetek szövege (az error: után) sokszor segít a szintaktikai hibák megtalálásában, máskor meg nem... Most ez utóbbi a szitu. Idővel kialakul majd a rutinod és rögtön fogod látni, hogy ott a 4-esnek nincs semmi keresnivalója.

```
$ g++ greta2.cpp -o greta2
$ ./greta2
Hello Vilag!

nbatfai@robopsy:~$
```

s ezt már egy jó megoldás követett:

```
#include <iostream>

int main()
{
    std::cout << "Hello Vilag!" << std::endl;
    std::cout << "Hello Vilag!" << std::endl;
    std::cout << "Hello Vilag!" << std::endl;
    std::cout << "Hello Vilag!" << std::endl;
    return 0;
}
```

### 1.1.3. Harmadik nap – szorzótábla

Feladatot oldottak meg a gyerekek: módosítsuk úgy a 2-es progit, hogy kiírja a hetes szorzótáblát, azaz ezt a kimenetet érjük el:

```
1 * 7 = 7
2 * 7 = 14
3 * 7 = 21
.
.
.
10 * 7 = 70
```

#### 1.1.3.1. Apa – mentor

Papíron röviden megbeszéltük hogy az első szám pont úgy változik mint a for ciklus *i* változója, a másik szám pedig mindig a 7-es. Utóbbit például így definiálhatjuk `int m =7;`. A kihatást külön megbeszéltük, hogy az első szám az *tk. az i*, a második az *m*, a harmadik a kettő összeszorozása, azaz ilyesmit írhatunk majd a gépnél: `cout << i << "*" << m << i*m << endl;`.

#### 1.1.3.2. Gréta

#### 1.1.3.3. Nándi

A tegnapi feladat Matyi féle for-os megoldásából indult ki, gyorsan ment

```
include <iostream>

int main()
{
    int m =7;
    std::cout << "Nandi szorzotabla" << std::endl;
    for(long int i=1; i<4; ++i)
        std::cout <<i <<" * "<<m <<" = " <<i*m<< std::endl;

    return 0;
}
```

csak a ciklus felső határát kellett módosítani egy jó megoldáshoz.

```
include <iostream>

int main()
{
    int m =7;
    std::cout << "Nandi szorzotabla" << std::endl;
    for(long int i=1; i<11; ++i)
        std::cout <<i <<" * " <<m <<" = " <<i*m<< std::endl;

    return 0;
}
```

```
$ ./nandi2
Nandi szorzotabla
1 * 7 = 7
2 * 7 = 14
3 * 7 = 21
4 * 7 = 28
5 * 7 = 35
6 * 7 = 42
7 * 7 = 49
8 * 7 = 56
9 * 7 = 63
10 * 7 = 70
```

Nándi ráérzett a dolog ízére, kimaxolta a feladatot: kinyomtatta 10000-ig a szorzótáblát majd meg akarta bolondítani a for ciklust azzal, hogy

```
#include <iostream>

int main()
{
    int m =7;
    std::cout << "Nandi szorzotabla" << std::endl;
    for(long int i=1; i<0; ++i)
        std::cout <<i <<" * " <<m <<" = " <<i*m<< std::endl;

    return 0;
}
```

```
$ g++ nandi2.cpp -o nandi2
$ ./nandi2
Nandi szorzotabla
$
```

#### 1.1.3.4. Matyi

Az alábbi sikertelen próbálkozás

```
#include <iostream>

int main()
{
    std::cout << "Hello Vilag!" << std::endl;
    for(int i=1;m=7 i<4; ++i)
        std::cout << i <<"*" <<m<<"=" <<i*m<< std::endl;
```

```
    return 0;
}
```

után Nánditól függetlenül (külön szobában programoztak éppen) ugyanazon a megoldási úton érte el a jó eredményt.

#### 1.1.3.5. Feladat

Módosítsuk úgy a programot, hogy ne a szorzótáblát, hanem az első 10 négyzetszámot produkálja! Azaz az alábbi kimenetet adja:

```
1 * 1 = 1
2 * 2 = 4
3 * 3 = 9
.
.
.
10 * 10 = 100
```

##### 1.1.3.5.1. Gréta

##### 1.1.3.5.2. Nándi

Nagyon jól kezdett:

```
#include <iostream>

int main()
{
    long int m = i;
    std::cout << "Nandi szorzotabla" << std::endl;
    for(long int i=1; i<11; ++i)
        std::cout << i << "*" << m << " = " << i*m << std::endl;
    return 0;
}
```

```
nandi2.cpp:5:17: error: 'i' was not declared in this scope
    long int m = i;
```

majd miután figyelmeztettem, hogy ott ahová az `long int m = i;` utasítást írta, ott az `i` még nem ismert, hiszen csak lejjebb mondtuk meg, hogy az egy kis szám (`int`) ami a `for` ciklus szerint változik, aztán előállt egy működő megoldással

```
#include <iostream>

int main()
{
    std::cout << "Nandi szorzotabla" << std::endl;
    for(long int i=1; i<11; ++i)
    {
        long int m = i;
        std::cout << i << " * " << m << " = " << i*m << std::endl;
    }
    return 0;
}
```

### 1.1.3.5.3. Matyi

Az `i*i` remek 5let, de nem kellett volna semmi más ezen túl

```
#include <iostream>

int main()
{
    int i=1<i<11;
    std::cout << "MATYI SZORZOTABLA" << std::endl;
    for(int i=1<; i<11 ; ++i)
        std::cout << i << " * " << i << "=" << i*i << std::endl;
    return 0;
}
```

mert ez így szintaktikailag hibás

```
matyi2.cpp:6:17: error: expected primary-expression before ';' token
    for(int i=1<; i<11 ; ++i)
```

a for i ciklusváltozójába becsúszott egy felesleges karakter, illetve a `int i=1<i<11;` értelmezhetetlen a program szempontjából.

Eztán a Nándinál már ismertetett irányba oldotta meg a feladatot, s Ő is kimaxolta ezt is a ciklus felső határának feltolásával. Ez egy idő után persze túlcserélődéshez vezetett, amit abból éreztek gyanúsak, hogy negatív számok jelentek meg a szorzatban.

```
.
.
.
46338 * 46338=2147210244
46339 * 46339=2147302921
46340 * 46340=2147395600
46341 * 46341=-2147479015
46342 * 46342=-2147386332
46343 * 46343=-2147293647
.
.
.
```

### 1.1.4. Negyedik nap – osztály

```
1  #include <iostream>
2
3  class Katona
4  {
5  public:
6      std::string nev;
7      int elet;
8      int sebzes;
9
10     Katona(std::string nev, int elet, int sebzes) {
11
12         this->nev = nev;
13         this->elet = elet;
14         this->sebzes = sebzes;
15     }
16
17     bool el() {
18         return elet > 0;
```

```
19     }
20
21     friend std::ostream &operator<< (std::ostream &stream, Katona &katona) {
22         stream << katona.nev << " " << katona.elet << " " << katona.sebzes;
23         return stream;
24     }
25
26 };
27
28
29 Katona &harcol(Katona &egyik, Katona &masik)
30 {
31     for (; egyik.el() && masik.el() ;) {
32         egyik.elet = egyik.elet - masik.sebzes;
33         masik.elet = masik.elet - egyik.sebzes;
34     }
35
36     if (egyik.elet > masik.elet)
37         return egyik;
38     else
39         return masik;
40 }
41
42
43 int main()
44 {
45     Katona en {"Nandi", 100, 10};
46     Katona ellen {"Barbar", 80, 5};
47
48     std::cout << harcol(en, ellen) << std::endl;
49
50     return 0;
51 }
```

#### 1.1.4.1. Apa – mentor

```
$ g++ nandi4.cpp -o nandi4 -std=c++11
$ ./nandi4
Nandi 60 10
```

#### 1.1.4.2. Gréta

#### 1.1.4.3. Nándi

#### 1.1.4.4. Matyi

A class az egy osztály. Az osztály az adatok és függvények csoportja. Milye van egy katonának? Élete, sebzése és neve.

Mikor él egy katona? Amikor az élete több mint 0. Ezt az `el` függvény mondja el. Az `el` függvény visszaadja a hívójának, hogy igaz-e, hogy él-e a katona.

Van több csatorna. Az egyik csatorna neve: `iostream` a másíknak `ostream`.

Mennyi az élete és a sebzése egy katonának? Nándinak az élete 100 és a sebzése 10. A Barbárnak az élete 80 és a sebzése 5. Ezeket a konstruktornak adjuk meg. A konstruktorról akkor beszélünk ha az osztály és a függvény neve megegyezik.

A végén a győztes katona nevét kiírjuk a csatornára. A `return` visszaad egy számot az operációs rendszernek.



#### 1.1.4.5. Feladatok

Itt már több feladatot adunk ki, úgy alakult, hogy ezeket közösen oldották meg a gyerekek (az első kettőt hárman egy gép előtt, de a másodikat már leginkább csak Matyi).

- Matyi fejben kiszámolta, még a futtatás előtt, hogy 60-at fog kiírni. Módosítsuk úgy a programot a `harcol` függvényben, hogy a harc minden lépésében nyomja ki a két katonát a képernyőre, azaz az alábbi futási eredmény legyen:

```
$ g++ nandi4.cpp -o nandi4 -std=c++11
$ ./nandi4
Nandi 95 10
Barbar 70 5
Nandi 90 10
Barbar 60 5
Nandi 85 10
Barbar 50 5
Nandi 80 10
Barbar 40 5
Nandi 75 10
Barbar 30 5
Nandi 70 10
Barbar 20 5
Nandi 65 10
Barbar 10 5
Nandi 60 10
Barbar 0 5
Nandi 60 10
```

A megoldás ez a kiegészítés volt a `harcol` definiáló `harcol` nevű függvényben:

```
Katona &harcol(Katona &egyik, Katona &masik)
{
    for (; egyik.el() && masik.el(); ) {
        egyik.elet = egyik.elet - masik.sebzes;
        masik.elet = masik.elet - egyik.sebzes;

        std::cout << egyik << std::endl;
        std::cout << masik << std::endl;
    }

    if (egyik.elet > masik.elet)
        return egyik;
    else
        return masik;
}
```

- Módosítsuk úgy az előző feladat megoldásának programját úgy, hogy a harc győztese után konstruáljunk meg egy harmadik katona objektumot ezekkel az adatokkal: {"Orias", 180, 20} és ez az új katona a győztesrel harcoljon, azaz az alábbi futási eredmény legyen:

```
$ g++ nandi4.cpp -o nandi4 -std=c++11
$ ./nandi4
Nandi 95 10
Barbar 70 5
Nandi 90 10
Barbar 60 5
Nandi 85 10
Barbar 50 5
Nandi 80 10
```

```

Barbar 40 5
Nandi 75 10
Barbar 30 5
Nandi 70 10
Barbar 20 5
Nandi 65 10
Barbar 10 5
Nandi 60 10
Barbar 0 5
Nandi 60 10
Nandi 40 10
Orias 170 20
Nandi 20 10
Orias 160 20
Nandi 0 10
Orias 150 20
Orias 150 20

```

Ez volt a gép melletti első próbálkozás a megoldásra, a main függvénybeli alábbi módosítás:

```

int main()
{
    Katona en{"Nandi", 100, 10};
    Katona ellen{"Barbar", 80, 5};
    Katona giant{"Giant", 180, 20};

    std::cout << harcol(en, ellen, en, giant) << std::endl;

    return 0;
}

```

ami egy jól is olvasható hibát adott, a fordító program pontosan megmondta mi ezzel a megoldási próbálkozással a gond: túl sok bemenő paramétert (too many arguments) adtunk meg a harcol nevű függvénynek

```

$ g++ nandi4.cpp -o nandi4 -std=c++11
nandi4.cpp: In function 'int main()':
nandi4.cpp:57:44: error: too many arguments to function 'Katona& harcol(Katona&, Katona&)'
    std::cout << harcol(en, ellen, en, giant) << std::endl;
                                   ^
nandi4.cpp:32:9: note: declared here
    Katona& harcol(Katona& egyik, Katona& masik)
    ^

```

s valóban, hiszen a harcol fejét nézzük csak meg! Katona &harcol(Katona &egyik, Katona &masik) Két bemenő katonát vár és nem négyet (és visszaadja a harcban győző katonát). Ennek megfelelően ez volt a következő próbálkozás a main függvényben:

```

    Katona en{"Nandi", 100, 10};
    Katona ellen{"Barbar", 80, 5};
    Katona giant{"Giant", 180, 20};
    Katona gyoztas =harcol(en, ellen);
    harcol(gyoztas, giant);

    std::cout << harcol(en, ellen) << std::endl;

```

Ez már lefordul, majd lefut:

```

$ ./nandi4
Nandi 95 10
Barbar 70 5
Nandi 90 10

```

```
Barbar 60 5
Nandi 85 10
Barbar 50 5
Nandi 80 10
Barbar 40 5
Nandi 75 10
Barbar 30 5
Nandi 70 10
Barbar 20 5
Nandi 65 10
Barbar 10 5
Nandi 60 10
Barbar 0 5
Nandi 40 10
Giant 170 20
Nandi 20 10
Giant 160 20
Nandi 0 10
Giant 150 20
Nandi 60 10
```

Ám ebben a kimenetben Matyi, az élesszemű észrevette, hogy valami nem stimmel. Jobban láthatja a kedves olvasó, ha -----el jelöljük benne a csatákat:

```
$ ./nandi4
Nandi 95 10
Barbar 70 5
Nandi 90 10
Barbar 60 5
Nandi 85 10
Barbar 50 5
Nandi 80 10
Barbar 40 5
Nandi 75 10
Barbar 30 5
Nandi 70 10
Barbar 20 5
Nandi 65 10
Barbar 10 5
Nandi 60 10
Barbar 0 5
-----
Nandi 40 10
Giant 170 20
Nandi 20 10
Giant 160 20
Nandi 0 10
Giant 150 20
-----
Nandi 60 10
```

2 csata helyett 3 volt... erre a megoldásuk az utolsó kitörlése (pontosabban kommentbe tétele) volt



#### Másoló konstruktor másoló konstruktor

S történt itt még valami nagyon érdekes! Nézd meg az utolsó kimeneti sort: Nandi 60 10, hogy lehet, hogy a Nandi elnevezésű katona élete 60, amikor előtte az óriással való harcban már lement nullára...? Most ez misztikus kérdés kell legyen, de választ kapsz majd rá az 6. napon.

```
Katona en{"Nandi", 100, 10};
Katona ellen{"Barbar", 80, 5};
Katona giant{"Giant",180,20};
Katona gyoztas =harcol(en , ellen);
harcol(gyoztas,giant);

//std::cout << harcol(en, ellen ) << std::endl;
```

hogy a feladat kiírásában megkívánt megoldást kapják, ebben a 2. harcol hívást kinyomták a csatornára és elkészültek:

```
Katona en{"Nandi", 100, 10};
Katona ellen{"Barbar", 80, 5};
Katona giant{"Giant",180,20};
Katona gyoztas =harcol(en , ellen);
std::cout << harcol(gyoztas,giant ) << std::endl;
```

ami már pont ugyanazt adja, amit kértem.

- Próbáld ki, majd magyarázd meg ezt a forráskódot:

```
int main()
{
    Katona en {"Nandi", 100, 10};
    Katona ellen {"Barbar", 80, 5};
    Katona harmadik {"Orias", 180, 20};

    std::cout << harcol(harcol(en, ellen), harmadik) << std::endl;

    return 0;
}
```

Papíron próbáltak meg választ adni, amit Matyi ügyesen így fogalmazott meg: TODO: saját szavaival amikor azt mondta, hogy a két csata rövidített leírása.

A megértést ellenőrző feladatként azt kapták, hogy az aktuális kódcsipet mintájára valósítsák meg azt gép mellett, hogy négy katona harcoljon, a negyedik az addigi győztes! Íme az első próbálkozásuk

```
int main()
{
    Katona en{"Nandi", 100, 10};
    Katona ellen{"Barbar", 80, 5};
    Katona giant{"Giant",180,20};
    Katona negyedik{"MATYI",500,300};

    std::cout << harcol(harcol(en,ellen, )harmadik)negyedik) << std::endl;
}
```

ami nyilván nem fordul le, hiszen teljesen rossz helyre tették a vesszőket:

```
$ g++ nandi4.cpp -o nandi4 -std=c++11
^[[Anandi4.cpp: In function 'int main()':
nandi4.cpp:59:42: error: expected primary-expression before ')' token
    std::cout << harcol(harcol(en,ellen, )harmadik)negyedik) << std::endl;
```

Ezt a hibát gyorsan ki tudták javítani

```
Katona en{"Nandi", 100, 10};
Katona ellen{"Barbar", 80, 5};
Katona giant{"Giant",180,20};
Katona negyedik{"MATYI",500,300};

std::cout << harcol(harcol(en,ellen ),harmadik),negyedik) << std::endl;
```

amivel már egy beszédes fordítási hibát kaptak:

```
$ g++ nandi4.cpp -o nandi4 -std=c++11
nandi4.cpp: In function 'int main()':
nandi4.cpp:59:50: error: 'harmadik' was not declared in this scope
    std::cout << harcol(harcol(harcol(en,ellen ),harmadik),negyedik) << std::en
```

mivel ők a tegnapi kódjukból indultak ki, ahol nem volt harmadik nevű referencia, hanem a giant nevet használták helyette, ennek megfelelően módosítottak:

```
$ g++ nandi4.cpp -o nandi4 -std=c++11
nandi4.cpp: In function 'int main()':
nandi4.cpp:74:61: error: expected ';' before ')' token
    std::cout << harcol(harcol(en,ellen ),harmadik),negyedik) << std::endl;
```

de még ez sem hozott sikert

```
$ g++ nandi4.cpp -o nandi4 -std=c++11
nandi4.cpp: In function 'int main()':
nandi4.cpp:74:61: error: expected ';' before ')' token
    std::cout << harcol(harcol(en,ellen ),harmadik),negyedik) << std::endl;
```

hiszen három bezáró kerekzárójel van és csak kettő nyitó! A hány harcot akartok? segítő kérdés segített, mert hármat akarnak, de csak kér harcol függvényhívás van, a helyét megtanácskozva betették a harmadikat:

```
std::cout << harcol(harcol(harcol(en,ellen ),harmadik),negyedik) << std::endl;
```

ami már egy jó működést produkált:

```
$ ./nandi4
Nandi 95 10
Barbar 70 5
Nandi 90 10
Barbar 60 5
Nandi 85 10
Barbar 50 5
Nandi 80 10
Barbar 40 5
Nandi 75 10
Barbar 30 5
Nandi 70 10
Barbar 20 5
Nandi 65 10
Barbar 10 5
Nandi 60 10
Barbar 0 5
Nandi 40 10
Giant 170 20
Nandi 20 10
Giant 160 20
Nandi 0 10
Giant 150 20
Giant -150 20
MATYI 480 300
MATYI 480 300
```

#### 1.1.4.5.1. A gyerekek által kiírt feladatok

- Gréta: három katona bajnokságot csapott. Egyik Gréta, élete 400, sebzése 300. Másik Nándi, élete 100, sebzése 10. Harmadik Matyi, élete 100, sebzése 5. A feladat, hogy ezt programozd be!

Másik feladat, hogy csinálj három katonát és adj nekik adatokat, például élet, sebzés. Az olvasó akármilyen számot írhat be, ha csinálja.

Másik feladat, hogy amikor a katonák harcolnak és ha valamelyik győz, akkor újra megkapja az életét.

- Nándi: a feladat az, hogy csinálj két katonát és utána csinálj velük csatát! Csinálj pontos adatokat, az életét, a sebzését, a nevét te találhatod ki.
- Matyi: a feladat az, hogy csináljunk egy kupát! Úgy, hogy Nándi meg Barbár, Giant és Matyi harcoljon egymással. A győztesek két csatát játszanak. Egyil csata győztese vs. Robbantó, a másik győztes vs. Varázsló. Utána a két győztes harcol egymással.
  - Nándi: {élete: 100, sebzése 10}
  - Matyi: {élete: 500, sebzése 200}
  - Giant: {élete: 180, sebzése 20}
  - Barbár: {élete: 80, sebzése 5}
  - Robbantó: {élete: 25, sebzése 25}
  - Varázsló: {élete: 60, sebzése 40}

### 1.1.5. Ötödik nap – veszélyes vizeken

Alakítsuk át kicsit a `nandi4.cpp` forrást: tegyük bele némi nyomkövetést, azaz olyan kiírásokat a képernyőre, amik segítenek látni, mi történik a programban.

```
1 #include <iostream>
2
3 class Katona
4 {
5 public:
6     std::string nev;
7     int elet;
8     int sebzés;
9
10    Katona(std::string nev, int elet, int sebzés) {
11
12        this->nev = nev;
13        this->elet = elet;
14        this->sebzés = sebzés;
15    }
16
17    bool el() {
18        return elet > 0;
19    }
20
21    friend std::ostream &operator<< (std::ostream &stream, Katona &katona) {
22        stream << katona.nev << " " << katona.elet << " " << katona.sebzés;
23        return stream;
24    }
25
26 };
27
28 Katona &harcol(Katona &egyik, Katona &masik)
29 {
30     {
31         std::cout << "harcol fgv: harc indul, " << egyik << " vs. " << masik << std::endl;
32
33         for (; egyik.el() && masik.el(); ) {
34
35             egyik.elet = egyik.elet - masik.sebzés;
```

```

36         másik.elet = másik.elet - egyik.sebzes;
37
38         std::cout << "harcol fgv: " << egyik << std::endl;
39         std::cout << "harcol fgv: " << másik << std::endl;
40
41     }
42
43     if (egyik.elet > másik.elet)
44         return egyik;
45     else
46         return másik;
47 }
48
49
50 int main()
51 {
52
53     Katona en {"Nandi", 100, 10};
54     Katona ellen {"Barbar", 80, 5};
55     Katona harmadik {"Giant", 180, 20};
56
57     std::cout << "main fgv: " << harcol(harcol(en, ellen), harcol(en, harmadik)) << std::endl;
58
59     return 0;
60 }

```

#### 1.1.5.1. Apa – mentor

Mire számítunk? Az en katona objektum (Nandi) lenyomja az ellen katona objektumot (a Barbar-t). A harmadik katona objektum (Giant) lenyomja az en objektumot, majd a két győztes: en és harmadik játszik, ahol harmadik nyer.

```

$ g++ nandi5.cpp -o nandi5 -std=c++11
$ ./nandi5
harcol fgv: harc indul, Nandi 100 10 vs. Giant 180 20
harcol fgv: Nandi 80 10
harcol fgv: Giant 170 20
harcol fgv: Nandi 60 10
harcol fgv: Giant 160 20
harcol fgv: Nandi 40 10
harcol fgv: Giant 150 20
harcol fgv: Nandi 20 10
harcol fgv: Giant 140 20
harcol fgv: Nandi 0 10
harcol fgv: Giant 130 20
harcol fgv: harc indul, Nandi 0 10 vs. Barbar 80 5
harcol fgv: harc indul, Barbar 80 5 vs. Giant 130 20
harcol fgv: Barbar 60 5
harcol fgv: Giant 125 20
harcol fgv: Barbar 40 5
harcol fgv: Giant 120 20
harcol fgv: Barbar 20 5
harcol fgv: Giant 115 20
harcol fgv: Barbar 0 5
harcol fgv: Giant 110 20
main fgv: Giant 110 20

```

Ezzel szemben mit mutatnak a nyomkövető üzenetek, mi is történt? A harmadik katona objektum (Giant) lenyomja az en (Nandi) objektumot. Aztán a már meggyengült en katona objektumot (Nandi-t) lenyomja az ellen objektum (a Barbar).

Majd a két győztes: ellen és harmadik játszik, ahol harmadik nyer. Hát nem pontosan erre számítottunk, ugye?

A magyarázat: mi feltettük amikor fejben játszottuk le, hogy először a harcol(en, ellen) játszódik le, majd a harcol(en, harmadik) és végül a két győztes játszik. Lehet van olyan gép, ahol ez pont így lenne, viszont az enyémen pont nem így lett! Ez azért van, mert a C++ nyelv nem határozza meg a függvényhívás bemenő paramétereinek kiértékelési sorrendjét. Esetünkben, hogy a külső harcol függvény a jobb oldali vagy a bal oldali harcol hívás végrehajtásával kezdődik-e. Mivel nincs meghatározva az egyik gépen lehet így lesz, a másikon meg máshogy. Ezért hajózunk máris veszélyes vizeken. Mit tehet a programozó? Mit kell tennie a programozónak? El kell kerülnie az olyan források írását, amelyekben ilyen szituk előfordulnak. Erre több lehetőség is adódik, hogy a programozó melyikkel él, az majd stílus kérdése lesz. Most elég annyit tudni, hogy a C++ nem a homokozó (mert például a Java vagy a C sharp az) abban az értelemben, hogy a programozó nemcsak pontosan tudhatja, mi történik programja hatására a memóriában a program objektumaival, hanem pontosan tudnia is kell!

Olvashatóbb is lesz a kód, ha a függvényhívásokat kivesszük a függvényhívásból, azaz ezt írjuk:

```
int main()
{

    Katona en {"Nandi", 100, 10};
    Katona ellen {"Barbar", 80, 5};
    Katona harmadik {"Giant", 180, 20};

    Katona egyikGyoztes = harcol(en, ellen);
    Katona masikGyoztes = harcol(en, harmadik);

    std::cout << "main fgv: " << harcol(egyikGyoztes, masikGyoztes) << std::endl;

    return 0;
}
```

Itt már egyértelműen van leírva C++ nyelven, hogy mit szeretne a programozó, de sajnos még mindig nem az történik:

```
harcol fgv: harc indul, Nandi 100 10 vs. Barbar 80 5
harcol fgv: Nandi 95 10
harcol fgv: Barbar 70 5
harcol fgv: Nandi 90 10
harcol fgv: Barbar 60 5
harcol fgv: Nandi 85 10
harcol fgv: Barbar 50 5
harcol fgv: Nandi 80 10
harcol fgv: Barbar 40 5
harcol fgv: Nandi 75 10
harcol fgv: Barbar 30 5
harcol fgv: Nandi 70 10
harcol fgv: Barbar 20 5
harcol fgv: Nandi 65 10
harcol fgv: Barbar 10 5
harcol fgv: Nandi 60 10
harcol fgv: Barbar 0 5
harcol fgv: harc indul, Nandi 60 10 vs. Giant 180 20
harcol fgv: Nandi 40 10
harcol fgv: Giant 170 20
harcol fgv: Nandi 20 10
harcol fgv: Giant 160 20
harcol fgv: Nandi 0 10
harcol fgv: Giant 150 20
harcol fgv: harc indul, Nandi 60 10 vs. Giant 150 20
harcol fgv: Nandi 40 10
harcol fgv: Giant 140 20
harcol fgv: Nandi 20 10
harcol fgv: Giant 130 20
harcol fgv: Nandi 0 10
harcol fgv: Giant 120 20
```



```
main fgv: Giant 120 20
```

Mert hogy lehet, hogy a harmadik csatában a Nandi élete megint 60 mikor előtte már lement nullára! Lássunk hát már egy jó megoldást, egyetlen és jelet tegyünk a Katona osztály után:

```
Katona& egyikGyoztes = harcol(en, ellen);  
Katona& masikGyoztes = harcol(en, harmadik);
```

és lássunk csodát:

```
harcol fgv: harc indul, Nandi 100 10 vs. Barbar 80 5  
harcol fgv: Nandi 95 10  
harcol fgv: Barbar 70 5  
harcol fgv: Nandi 90 10  
harcol fgv: Barbar 60 5  
harcol fgv: Nandi 85 10  
harcol fgv: Barbar 50 5  
harcol fgv: Nandi 80 10  
harcol fgv: Barbar 40 5  
harcol fgv: Nandi 75 10  
harcol fgv: Barbar 30 5  
harcol fgv: Nandi 70 10  
harcol fgv: Barbar 20 5  
harcol fgv: Nandi 65 10  
harcol fgv: Barbar 10 5  
harcol fgv: Nandi 60 10  
harcol fgv: Barbar 0 5  
harcol fgv: harc indul, Nandi 60 10 vs. Giant 180 20  
harcol fgv: Nandi 40 10  
harcol fgv: Giant 170 20  
harcol fgv: Nandi 20 10  
harcol fgv: Giant 160 20  
harcol fgv: Nandi 0 10  
harcol fgv: Giant 150 20  
harcol fgv: harc indul, Nandi 0 10 vs. Giant 150 20  
main fgv: Giant 150 20
```

végre az történik amit a programozó elképzelt!

#### 1.1.5.2. Gréta

#### 1.1.5.3. Nándi

#### 1.1.5.4. Matyi

### 1.1.6. Hatodik nap – másoló konstruktor

1

#### 1.1.6.1. Apa – mentor

**1.1.6.2. Gréta**

**1.1.6.3. Nándi**

**1.1.6.4. Matyi**

## **III. rész**

# **A jövő programozása**

Erwin Schrödingernek az élettel kapcsolatos kutatásaiból<sup>5</sup> tudjuk, hogy az élet titka az entrópia alacsonyan tartásának képessége. Ennek fényében már nem csupán – mint mindenki – szeretek enni, hanem immár a megértés egy magasabb szintjén tudom is, hogy életem fenntartása érdekében eszek (és lélegzek). Sok esetben az eredeti nyilvánvaló érzésnek a kontrollálatlansága állhat az elhízás mögött.

A mozgáshoz hasonló kedvet érzünk, ugyanúgy örömmünket leljük benne, meglehet, hogy ennek is van entrópia alapú magyarázata? Például éppen akár a cikk<sup>6</sup> kapcsán is, hogy ugyan a mozgás emeli a hőmérsékletet de ezzel egyben a legmagasabb entrópiájú hőt ad le illetve a szaporább légzéssel több magas entrópiájú szén dioxidot.

Az entrópia nem növelő játékok koncepcióját az ESAMU programozói kézikönyvben vezettük be. Aktuális célunk ennek megkoronázása, azaz az „Entropy non-increasing games for improvement of dataflow programming” című kézirat készítésével párhuzamosan itt a kézirat eredményeit próbáljuk majd átültetni a gyakorlatba egy konkrét játék megírásával, ami játék egyben egy neurális alapú feladatmegoldó optimalizálását is jelenti majd.

---

<sup>5</sup> Lásd Erwin Schrödinger, What is life? : the physical aspect of the living cell, Cambridge University Press, 1944 illetve Roger Penrose, A császár új elméje Számítógépek, gondolkodás és a fizika törvényei, Akadémiai, Budapest, 1993.

<sup>6</sup> <http://www.bmj.com/content/bmj/349/bmj.g7257.full.pdf>

---

## **2. fejezet**

# **Bevezetés**

### **2.1.**

## 3. fejezet

# Tárgymutató

### D

DeepMind

TensorFlow, *lásd* Google

DevRob, [2](#)

DevRob2Psy, [2](#)

### E

ESAMU, [2](#)

### J

JIBO, [2](#)

### R

robotpszichológia, [vii](#), [3](#)

### S

Samu, [3](#)

SRS, [2](#)

### T

TensorFlow

TensorBoard, *lásd* DeepMind

### U

UDPROG, [3](#)