We get the maximum of the programming

Programming and robotpsychology for children under 12

Ed. ESAMU Book U12, Entrópia Samu, 2016. nov. 3, v. book.u12.hu.0.0.7.2

COLLABORATORS

	TITLE : We get the maximum of the programming	

ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY	Dr.. Norbert Bátfai, Margaréta Niobé Bátfai, Nándor Benjámin Bátfai, Mátyás Bendegúz Bátfai, Artúr Sáfrány, Gergő Bogacsovics, Kinga Enyedi, Kristóf Kovács, and Sándor Kovács	November 5, 2016	
...		November 5, 2016	
...		November 5, 2016	
...		November 5, 2016	
...		November 5, 2016	
...		November 5, 2016	

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME
0.0.1	2016-09-19	Initial document. The developement happens at the central storage of ESAMU: https://github.com/nbatfai/SamuEntropy, for version history see the changelog there.	nbatfai

Contents

Dedication

We recommend this book for its readers, for the robotpsychologists of the future.

Foreword

If you are a child under age 12, you can skip this boring foreword, we didn't devote it to you but your mentor. Of course it's not problem if you read over, just don't fall asleep meanwhile, because it's about a further future that is still completely unconcerned for you.

The book consists of two parts, the first one – anticipating for a few's pleasure – C++ programming, the programming of the present. The second one is the possible programming exercise of the future, when we already don't program in the classic manner, but only play. Since it's a game (esport), here massive amount of readers can be counted on.

Once my hacker friend expounded to me that the programming as an activity can't be algorithmic, therefore it's not teachable. There is only one possibility: encourage the ones that are interested in programming to program and the wonder happens by itself: some becomes programmer!

We feel this creed as ours, too. However, in this document we still try to teach programming the children under age 12 in language... `C++`. We deal with this in the first part, but we don't wait for the wonder from this but from the second part, where we introduce – according to our vision – the programming of the future.

What is this vision?

When I was in highschool, I started to use a computer [1] and I had two relationship with it: I programmed and I played games on it. These two activities had one same part, for example when I defined over a 8x8 bytes character to a person and in my own „game," it moved on a screen... but the game and the programming was two different thing and still is.

But would it different in the future? In our vision will be places where not! For example different programming tasks with a cognitive character belonging to the area of artificial intelligence eg. machine vision when the machine is not only processes the images, but at similar level also can answer questions like people, what is depicted in the picture!

Such system we are programming not like „from scratch", but we set out from a general solution and we refine it. If this solution based on for example TensorFlow [2], we will work with TensorBoard's [3] stream editor.

And what is Samu Entropy's vision? Let's pull a newer interface over these data flow onto visualisation devices which can give you entropy-decrease gaming experience... in one word: let us make the programming to a game. The asimovian robot programming psychology interpretation of this activity [4] is considered as a robot psychology (or at least as it's germ, since intuitively in the Westworld titled movie appearing robotpsychology could be considered as the the perfect but of course today still imagined realization of the asimovian vision).

How are we doing? The first part is easier: the children (Gréta 8, Nándi 8 and Matyi 10 years old) they arrive home from the swimming train after 6 pm, they have dinner, then everyday we discuss one C++ source and of course we try it. They describe in their own words for the next day and they send me them in e-mail, and I get into this book – at least this is the ideal plan. :)

The second part would be more complecated, because that – after Dénes Gábor – you have to invent. We trying to make a publication named „Entropy non-increasing games for improvement of dataflow programming".

Next up is the glossary, which is still more tuned to the mentor as a child reader.

[1] Commodore 64, Balassi Bálint High School Balassagyarmat, thanks to István Fűrész informatics teacher, I started to like programming, too.
[2] https://www.tensorflow.org/
[3] https://github.com/tensorflow/tensorflow/tree/master/tensorflow/tensorboard
[4] https://github.com/nbatfai/Robopsychology

Part I

Vocabulary

D

Developmental Robotics 2 Robopsychology [DevRob2Psy]

The DevRob2Psy is the abbreviation of the Developmental Robotics 2 Robopsychology facebook group. This is an informal group wich collects the students and professors who are interested in this topic. In this group you can write comments with your own ideas: https://www.facebook.com/groups/devrob2psy/.

See Also "UDPROG".

Developmental robotics [DevRob]

It is a scientific field which aims at studying the developmental mechanisms, architectures and constraints that allow lifelong and open-ended learning of new skills and new knowledge in embodied machines. Its basic thesis is that the robot has a body, the anti thesis of this could be a robot that only has software, the synthesis of the two was given in the PSAMU manuscript.

See Also "Robotpsychology".

E

Samu Entropy [ESAMU]

It is quite obvious that there is a close relation between esport and artificial intelligence. This relation is further enhanced by our research which tries to focus not only on using the artificial intelligence but on creating it in a new way. Our idea is to develop a new developmental robotics and robopsychology based game called Samu Entropy as a social robot that will be implemented by software on the family's home desktop and mobile devices. https://github.com/nbatfai/SamuEntropy.

See Also "Samu".

I

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

The Software Requirements Specifications IEEE Standard. We are going to make the documentation of the Samu Entropy with this standard.

J

JIBO

The social robot of Cynthia Breazeal, see also https://www.jibo.com/

R

Robotpsyhology

The Asimov robotpsyhology in programing view https://github.com/nbatfai/Robopsychology.

S

Samu

Once the devedevelopment robotics family chatrobot, Samu Bátfai, on the other hand the collective name of our development robotics research. Also https://prezi.com/utlu1bevq9j2/egy-testetlen-fejlodesrobotikai-agens/ the presentation https://arxiv.org/abs/1511.02889 and the PSAMU manual.

See Also "ESAMU".

U

University of Debrecen [UD]

The UD is the abbreviation of the University of Debrecen

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

The University of Debrecen regular programing education and, our group in Linkedin `https://www.linkedin.com/groups/Yearbook-Programmers-University-Debrecen-7446358?homeNewMember=&gid=7446358fromEmail=&ut=2mVSHNO5Dwwm41&trk=eml-grp-sub`, which have two basis once the softwares, and the other is the yearbook.

Part II

Programming of the present

Programming and Linux are two good friends, begin with installing Linux on your PC, with the help of your mentor!

Chapter 1

Introduction

1.1 Good beginning

If you alreay have Linux on your PC then you can read further, if you don't then scroll back!

1.1.1 The first day – `Hello, World!`

We begin with this `C++` source code (which was for example given to Gréta in the `greta1.cpp` file).

```
#include <iostream>

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

1.1.1.1 Dad – mentor

It is true for the programmer, too, that generally he/she stands on the shoulders of giants [1], in other words, he or she reuses other programmers' programs. The `#include` command does exactly this, it pastes the `iostream` programs written by others. In this `iostream` there are programs that materialize the channels. For example your program will have a channel[2], whereto your program sends the letters, and these letters will be displayed on the screen.

The basic foundation of a `C++` program is the function, this has a first line, which is now the `int main()` line. This is made of three parts: it tells what the function will calculate: this is now a number, the int keyword means this. The second part is the name of the function, this in our case is `main`. The third part is written between parantheses: the function receives what is here, it has to calculate from these . In our case this part is empty, in other words we didn't give the function anything as an input.

The body of the function is the block, which is a list of commands written between curly braces. Every command ends with a semicolon. We have now two commands. The first one is the one called `std::cout` which prints the `"Hello, Vilag!"` text to output channel. The input part of this channel is in our program and the output part is opened to the screen. Then we print a special signal, which makes the cursor jump to a new line on the screen.

[1] The great scientist, Isaac Newton wrote this (in his letter written to Robert Hook) to express that he could also build upon the work of others: "If I have seen further it is by standing on the sholders of Giants".

[2] Where does this channel come from? On your machine (or for example on your phone or even on your tablet) there is a special program, called the operating system, the Linux kernel, which Linus Torvalds started to write when he was on a university in Finnland. The operating system is a good friend of the programmer, this program guarantees fro example these channels, too.

The second command is the `return` which gives back the value calculated by the function to the caller of the function. We didn't calculate anything now, only printed to the screen, so we returned 0. This means by convention that everything was okay with the program, because the caller of your `main` function was the operating system itself, the kernel of the Linux, the program indicates that it didn't have any problems. With this, the `main` function ended and with that our program finished.

Now let us try out the program!

First, we translate the source code so that the processor of our machine understands it. We save this translation in the greta1 file with the g++ compiler. Then by writing its name after the dot slash we run the compiled program. Running a program is the process when the processor reads the compiled program.

What happened after pressing enter after typing dot slash and the filename? The main function was executed, which printed the Hello, World! text to the output channel, which was therefore displayed on the screen.

```
$ g++ greta1.cpp -o greta1
$ ./greta1
Hello, World!
$
```

The language of the processor

The machines do not understand the various `C++` source codes, but the `C++` programmers do! The source code must be compiled to the processor's own language. This is executed by the g++compiler program [a]. You don't have to worry, the g++ program is already on your computer, it has been helping the Linux since a very long time!

Your mentor can show you what the language that your processor understands actually looks like: it is made of numbers. Here is for example the runnable code compiled from you first program:

```
$ hexdump -v -e '/1 "%02X "'  greta1 | more
7F 45 4C 46 02 01 01 00 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 40 1C 00 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...
```

... because this continues on more displays! You can see that the language of your processor is not understandable for the `C++` programmers.

[a] The g++ compiler was written by a famous programmer who has a well-known hacker name: RMS, in other words Richard Matthew Stallman. If in your dreams you found yourself in the world of The Lord of The Rings as Frodo, then he will be Gandalf. :)

The programmer manual of Linux
Previously we saw the contents of the `greta1` file with the **hexdump** command: `$ hexdump -v -e '/1 "%02X "' greta1 | more` but don't think that I can write these things immediately! Simply I just check them in the manual. In the world of C programming, it is even more of a daily routine to check what you have to use in the manual and see how you can make it work! This is a written profession, it was not given to us by God, it was invented by engineers, this is the simple truth: you only have to read it. For example take a look at what the manual says about the `std::cout` object:

```
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

The processor counts numbers and prints the letters to the `std::cout` . The g++ compiles the program. The iostream is a channel. The `#include` pastes others' code into yours. The 0 means that your program executed successfully.

1.1.1.3 Nándi

The g++ translates to the processor's language. The `#include` is a command, which tells the computer to copy other's code into yours. The `cout` is a channel for letters to cross it. The << means that push the letters in this direction. The 0 means that the main function was executed properly.

1.1.1.4 Matyi

The `#include` is the first command. With the `#include` we paste into other programs. The `iostream` is a channel. The `int main` is the main function. The main function is called by the operating system. The function means that it is a building block of the program. `std::cout << Hello World << std::endl` This means that we print the letters to the channel. The `std::endl` means that the cursor goes to the next line. The `return` returns a number to the caller of the function. The `return 0;` means that the program finished successfully.

1.1.2 Second day – for loop

We improve our first program, so that we can get this second one:

```
#include <iostream>

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

1.1.2.1 Dad – mentor

With the `for` statement we can repeat things, in other words we can make iterations in our program. The iteration means that we execute something (the thing that we iterate) over an over again, multiple times (it can even be 0: 0 or infinite).

Generally, a for loop looks like this `for(starting point;how many times;ending point) {we iterate what is here}`.

In the second source we iterate a whole number `int i` starting from the value of `i=0` (initialising) and we continue until it is true that `i<3`, so the i variable takes on the values 0, 1 and 2, printing them in this order.

1.1.2.2 Gréta

The for is a loop. The for starts from i=0 and iterates while `i<3` is true.

1.1.2.3 Nándi

The for repeats the loop. The for is a loop. What does a loop mean? Do one push-up then 10 push-ups. The 10 push-ups are a loop from 1 to 10.

1.1.2.4 Matyi

The for is a loop. It is a loop because it executes the program multiple times. The ++i means that we increase the value of i by 1. For example if i equals 1 then

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

If there was a curly brace next to the for statement then there would be more commands. The i=0 is where the iteration starts, the `i<3` shows how long it will iterate and the ++i shows by how many it increases on every iteration.

1.1.2.5 Task

Modify the second program to write the `Hello, World!` text to the screen 4 times! The chilren gave the following solutions in the following order.

1.1.2.5.1 Nándi

He took the first program as a starting point generally instead of the second one because he deleted the for loop from the second, then he place a `Hello, World!` text, indenting it with tabulators.

```
#include <iostream>

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

With this, he got the following error.

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

The compiler tried to identify the text as a part of the code because it was not between apostrophes, but it did not succeed. Although on his next try his program compiled successfully,

```
#include <iostream>

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

his program didn't do anything. The next change was this:

```
#include <iostream>

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

Then this, to print 4 `Hello World!`

```
#include <iostream>

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

and finally to start printing in new lines:

```
#include <iostream>

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

1.1.2.5.2 Matyi

He started from program 2, first he modified the bound of the loop from 3 to 4 and he wrote `i` instead of `Hello, World!`.

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

He got this error (earlier seen at Nándi already)

```
$ 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 World! << std::endl;
```

He wrote back i into the printing and deleted the incrementation of the loop variable:

```
#include <iostream>

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

So his program ran into an infinite loop at runtime... `Hello, World!` lines were just running in the terminal window[3], in the end only closing the window helped!

In the report of Matyi
If from 1 to 4 ++i, then writes four times: Hello World. If we don't put `++i` there, things will happen just like with me, executing Hello World in an infinite loop.

After putting incrementation of loop variable back, he got a good solution:

[3] After iteration of the core of the loop, deleting the incrementation of i, i will stay 0 and will never reach 3, so the iteration of the core of the loop will never end. This is certainly an infinite loop!

```
#include <iostream>

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

1.1.2.5.3 Gréta

First she started with program 2 too and incremented to 1 the beginning value of the loop variable

```
#include <iostream>

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

then compiling and running essentially she returned back to hacking her program 1 in her puzzlement, since she deleted the loop and after the words Hello World she wrote a number 4:

```
#include <iostream>

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

    return 0;
}
```

that she got this error to:[4]

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

correcting this, she was trying with this

```
#include <iostream>

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

that was already compiling and running, still wrong though:

[4] The text of compilation error (after the "error:") often helps at finding the syntax error, however sometimes this is not the case... Now the letter one is the case. With time your routine will be formed and you will immediately see that that 4 has nothing to do there.

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

nbatfai@robopsy:~$
```

and that was followed by a good solution:

```
#include <iostream>

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

1.1.3 Third day – multiplication table

The children solved a task: let's modify the program 2 so that it prints the 7 times table, i.e. we must reach this output:

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

1.1.3.1 Dad – mentor

On paper we discussed briefly that the first number changes just like the for loop variable i, the other number is always 7, in turn. We can define the latter one for example as `int m =7;`. We talked over the printing separately, that the first number is always i, the second one is m, the third one is the multiplication of the two, i.e. later at the computer we can write something like this: `cout << i << "*" << m << i*m << endl;`.

1.1.3.2 Gréta

1.1.3.3 Nándi

The task from yesterday started from Matyi's for-like solution, it was gone fast

```
include <iostream>

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

    return 0;
}
```

it was only needed to modify the upper bound of the loop to the right solution.

```
include <iostream>

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

    return 0;
}
```

```
$ ./nandi2
Nandi multiplication table
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 got an inkling of how stuff is going, he made the maximum of the task: he printed the times table to 10000, then he wanted to tweak the for loop as:

```
#include <iostream>

int main()
{
    int m =7;
    std::cout << "Nandi multiplication table" << 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 multiplication table
$
```

1.1.3.4 Matyi

After the undermentioned unsuccessul attempt

```
#include <iostream>

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

```
    return 0;
}
```

independently from Nándi (they were just programming in different rooms) he reached the right result through the same way.

1.1.3.5 Task

Let's modify the program so that instead of the multiplication table it produces the first 10 square numbers! I.e. it gives the undermentioned output:

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

1.1.3.5.1 Gréta

1.1.3.5.2 Nándi

He started very well:

```
#include <iostream>

int main()
{
    long int m =i;
    std::cout << "Nandi multiplication table" << 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;
```

after I warned him that at the place where he had written the statement `long int m =i;`, i is still not known because we only told later that it's a small number (int) which changes by the for loop, he came up with a solution.

```
#include <iostream>

int main()
{

    std::cout << "Nandi multiplication table" << 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 túlcsorduláshoz vezetett, amit abból éreztek gyanúsnak, 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

```
#include <iostream>

class Katona
{
public:
    std::string nev;
    int elet;
    int sebzes;

    Katona(std::string nev, int elet, int sebzes) {

        this->nev = nev;
        this->elet = elet;
        this->sebzes = sebzes;
    }

    bool el() {
        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 másiknak 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 mag. A konstruktorról akkor beszélünk ha az osztály és a függvény neve meggegyezik.

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 harcot 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 győztessel 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 győztessel! 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 függényhívás 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: tegyünk bele némi nyomkövetést, azaz olyan kiírásokat a képernyőre, amik segítenek látni, mi történik a programban.

```
#include <iostream>

class Katona
{
public:
    std::string nev;
    int elet;
    int sebzes;

    Katona(std::string nev, int elet, int sebzes) {

        this->nev = nev;
        this->elet = elet;
        this->sebzes = sebzes;
    }

    bool el() {
        return elet > 0;
    }

    friend std::ostream &operator<< (std::ostream &stream, Katona &katona) {
        stream << katona.nev << " " << katona.elet << " " << katona.sebzes;
        return stream;
    }

};


Katona &harcol(Katona &egyik, Katona &masik)
{
    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.sebzes;
36          masik.elet = masik.elet - egyik.sebzes;
37
38          std::cout << "harcol fgv: " << egyik << std::endl;
39          std::cout << "harcol fgv: " << masik << std::endl;
40
41      }
42
43      if (egyik.elet > masik.elet)
44          return egyik;
45      else
46          return masik;
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 föggvény 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!

Ezzel a változattal a gyerekek szabad stílusban (spontán) kezdtek el játszani:

```
    Katona en {"Matyi", 1000, 100};
    Katona ellen {"Greta", 800, 50};
    Katona harmadik {"Nandi", 800, 50};

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

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

```
harcol fgv: harc indul, Matyi 1000 100 vs. Greta 800 50
harcol fgv: Matyi 950 100
harcol fgv: Greta 700 50
harcol fgv: Matyi 900 100
harcol fgv: Greta 600 50
harcol fgv: Matyi 850 100
harcol fgv: Greta 500 50
```

```
harcol fgv: Matyi 800 100
harcol fgv: Greta 400 50
harcol fgv: Matyi 750 100
harcol fgv: Greta 300 50
harcol fgv: Matyi 700 100
harcol fgv: Greta 200 50
harcol fgv: Matyi 650 100
harcol fgv: Greta 100 50
harcol fgv: Matyi 600 100
harcol fgv: Greta 0 50
harcol fgv: harc indul, Matyi 600 100 vs. Nandi 800 50
harcol fgv: Matyi 550 100
harcol fgv: Nandi 700 50
harcol fgv: Matyi 500 100
harcol fgv: Nandi 600 50
harcol fgv: Matyi 450 100
harcol fgv: Nandi 500 50
harcol fgv: Matyi 400 100
harcol fgv: Nandi 400 50
harcol fgv: Matyi 350 100
harcol fgv: Nandi 300 50
harcol fgv: Matyi 300 100
harcol fgv: Nandi 200 50
harcol fgv: Matyi 250 100
harcol fgv: Nandi 100 50
harcol fgv: Matyi 200 100
harcol fgv: Nandi 0 50
harcol fgv: harc indul, Matyi 200 100 vs. Matyi 200 100
harcol fgv: Matyi 0 100
harcol fgv: Matyi 0 100
main fgv: Matyi 0 100
```

De Matyi nem akarta, hogy a végén a `Matyi` név-tulajdonságú objektum önmagával harcoljon... a következő próbálkozásaik arról szóltak, hogy a végén a `Matyi` objektum győzzön, de ne magával harcoljon. Próbálkozást próbálkozás követett, de a többiek örömére egyszer a `Gréta`, máskor a `Nándi` objektum győzött, s persze párszor visszaköszönt a `Matyi` önmagával való harca is, mire belátta, hogy vannak olyan feladatok, amelyek megadott keretek között megoldhatatlanok. (Mert ha az első meccset Matyi elveszti, akkor a másodikba már nem élve megy, így nem kerül a döntőbe. Ha ez nem teljesül, azaz az elsőt nyeri, de a másodikat bukja, akkor a döntőbe kerül már eleve nem élve.)

Elevenítsük fel a következő pontban az egyik Gréta által bedobott feladatot!

1.1.5.2 Gréta, Nándi, Matyi

> „Másik feladat, hogy amikor a katonák harcolnak és ha valamelyik győz, akkor újra megkapja az életét."
>
> —Gréta

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

Part III

The future's programming

From Erwin Schrödinger's life connected researches[5] we know that the secret of life is the talent to keep the entropy low. In light of that yet not only – as everyone – i like to eat but now I know this on a higher level of understanding too that I eat in order to support my life (and I breath). In many cases the original obvious feelings' uncontrolness is behind of the overweight.

We feel a similar mood for the activities as we find our happiness in it, could happen that it has an entropy based explanation? For example already in this article[6] the activity lifts the temperature but in the meantime it gives high level entropy heat, respectively with the faster breathing more high entropy carbon dioxide.

The entropy's non-increasing games conception is introduced in the ESAMU programmer handbook. Our current goal is it's elevation so in parallel with the „Entropy non-increasing games for improvement of dataflow programming" titled manuscript's creation, we also try to transplant the manuscript's results in practice through writing a concrete game, which will be also a neural based task-solving optimalization.

[5] See Erwin Schrődinger, 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.

[6] http://www.bmj.com/content/bmj/349/bmj.g7257.full.pdf

Chapter 2

Introduction

2.1

Chapter 3

Index