We get the maximum of the programming

Programming and robotpsychology for children under 12

Ed. ESAMU Book U12, Entrópia Samu, 2016. nov. 6, v. book.u12.en.0.0.1

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 11, 2016	
English translation from the original Hungarian version book.u12.hu.0.0.7.2		November 11, 2016	
English translation from the original Hungarian version book.u12.hu.0.0.7.2		November 11, 2016	
English translation from the original Hungarian version book.u12.hu.0.0.7.2		November 11, 2016	
English translation from the original Hungarian version book.u12.hu.0.0.7.2		November 11, 2016	
English translation from the original Hungarian version book.u12.hu.0.0.7.2		November 11, 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

I Vocabulary **1**

II Programming of the present **4**

1 Introduction **6**

1.1 Good beginning . . . 6

1.1.1 The first day – `Hello, World!` . . . 6

1.1.1.1 Dad – mentor . . . 6

1.1.1.2 Gréta . . . 8

1.1.1.3 Nándi . . . 8

1.1.1.4 Matyi . . . 8

1.1.2 Second day – for loop . . . 8

1.1.2.1 Dad – mentor . . . 9

1.1.2.2 Gréta . . . 9

1.1.2.3 Nándi . . . 9

1.1.2.4 Matyi . . . 9

1.1.2.5 Task . . . 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 Third day – multiplication table . . . 13

1.1.3.1 Dad – mentor . . . 13

1.1.3.2 Gréta . . . 13

1.1.3.3 Nándi . . . 13

1.1.3.4 Matyi . . . 14

1.1.3.5 Task . . . 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 Fourth day – class . . . 16
1.1.4.1 Dad – mentor . . . 17
1.1.4.2 Gréta . . . 17
1.1.4.3 Nándi . . . 17
1.1.4.4 Matyi . . . 17
1.1.4.5 Tasks . . . 18
1.1.4.5.1 Tasks written by the kids . . . 23
1.1.5 Fifth day– on slippery ground . . . 23
1.1.5.1 Dad – mentor . . . 24
1.1.5.2 Gréta, Nándi, Matyi . . . 27
1.1.6 Sixth day – copy constructor . . . 27
1.1.6.1 Dad – mentor . . . 28
1.1.6.2 Gréta . . . 28
1.1.6.3 Nándi . . . 28
1.1.6.4 Matyi . . . 28

III The future's programming 29

2 Introduction 31
2.1 . . . 31

3 Index 32

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 complicated, 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

Good beginning

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

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;
}
```

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 parentheses: 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 shoulders 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 Finland. 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.
       ...
```

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.

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.

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.

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;
}
```

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.

Gréta

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

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.

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.

Task

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

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;
}
```

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;
}
```

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;
}
```

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
```

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;`.

Gréta

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
$
```

Matyi

After the undermentioned unsuccessful 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.

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
```

Gréta

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;
}
```

Matyi

The `i*i` is a great idea, but it was enough

```
#include <iostream>

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

because this yields a syntactic error.

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

There is an unnecessary character in the first part of the for loop declaration, furthermore the program cannot understand the `int i=1<i<11;` statement.

After this, he solved the task in a way mentioned earlier regarding Nándi, and he maxed out too this loop with pushing the upper limits higher. Of course, after some time this lead to overflow, which they could notice from the negative numbers in the multiplications.

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

Fourth day – class

```
#include <iostream>

class Soldier
{
public:
    std::string name;
    int HP;
    int damage;

    Soldier(std::string name, int HP, int damage) {

        this->name = name;
        this->HP = HP;
        this->damage = damage;
    }

    bool alive() {
```

```
        return HP > 0;
    }

    friend std::ostream &operator<< (std::ostream &stream, Soldier &soldier) {
        stream << soldier.name << " " << soldier.HP << " " << soldier.damage;
        return stream;
    }

};


Soldier &fight(Soldier &first, Soldier &second)
{
    for (; first.alive() && second.alive() ;) {
        first.HP = first.HP - second.damage;
        second.HP = second.HP - first.damage;
    }

    if (first.HP > second.HP)
        return first;
    else
        return second;
}


int main()
{
    Soldier me {"Nandi", 100, 10};
    Soldier enemy {"Barbarian", 80, 5};

    std::cout << fight(me, enemy) << std::endl;

    return 0;
}
```

Dad – mentor

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

Gréta

Nándi

Matyi

The class is a class. The class is a group of data and functions. What does a soldier have? HP, damage and name.

When does a soldier alive? When his/her HP is more than 0. The `alive` function gives information about this. The `alive` function tells whether the soldier alive or not to its caller.

There are more channels. The name of one of these is: iostream and an other one is the ostream.

How much HP and how much damage does a soldier have? Nándi's HP is 100 and his damage is 10. The Barbarian has 80 HP and 5 damage. We give these to the constructor. We talk about a constructor when the name of the function and the class is the same.

Finally, we print the name of the winner soldier to the channel. The return returns a number to the operating system.

Tasks

Here we gave more tasks and it happened in a way that the kids solved them together (the first two tasks, the three of them on one computer, but the second task mostly done only by Matyi)

- Matyi reckoned in his head before the execution that it will give 60. We modified the programme according to the `fight` that the two soldiers will be shown on the screen in every step of the battle so to get the following result:

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

  The solution was this addition to the function which defines the fight in the `fight`:

```
Soldier &fight(Soldier &first, Soldier &second)
{
    for (; first.alive() && second.alive() ;) {
        first.HP = first.HP - second.damage;
        second.HP = second.HP - first.damage;

        std::cout << first << std::endl;
        std::cout << second << std::endl;
    }

    if (first.HP > second.HP)
        return first;
    else
        return second;
}
```

- Modify the previous task's programme in a way of after the fight's winner let's create the third soldier object with this data `{"Giant", 180, 20}` and this new soldier fights against the winner, so we have to get the following result:

```
$ g++ nandi4.cpp -o nandi4 -std=c++11
$ ./nandi4
Nandi 95 10
Barbarian 70 5
Nandi 90 10
Barbarian 60 5
Nandi 85 10
Barbarian 50 5
Nandi 80 10
Barbarian 40 5
```

```
Nandi 75 10
Barbarian 30 5
Nandi 70 10
Barbarian 20 5
Nandi 65 10
Barbarian 10 5
Nandi 60 10
Barbarian 0 5
Nandi 60 10
Nandi 40 10
Giant 170 20
Nandi 20 10
Giant 160 20
Nandi 0 10
Giant 150 20
Giant 150 20
```

That was the first try on the computer for the solution, in the main function the following modification:

```
int main()
{
    Soldier me{"Nandi", 100, 10};
    Soldier against{"Barbarian", 80, 5};
    Soldier giant{"Giant",180,20};

    std::cout << fight(me, against ,me,giant) << std::endl;

    return 0;
}
```

which gave a really telling error, the compiler programme said correctly what is the problem with this one: too many arguments are given to the `fight` named function

```
$ g++ nandi4.cpp -o nandi4 -std=c++11
nandi4.cpp: In function ‘int main()’:
nandi4.cpp:57:44: error: too many arguments to function ‘Soldier& fight(Soldier&, Soldier ↩
    &)’
     std::cout << fight(me, against ,me,giant) << std::endl;
                                            ^
nandi4.cpp:32:9: note: declared here
 Soldier& fight(Soldier& first, Soldier& second)
         ^
```

and really, because now just check the `fight`'s head! `Soldier &fight(Soldier &first, Soldier &second)` This one waits for two soldiers and not four (and it gives back the winner one). According to this, there is the next try in the main function:

```
    Soldier me{"Nandi", 100, 10};
    Soldier against{"Barbarian", 80, 5};
    Soldier giant{"Giant",180,20};
    Soldier winner =fight(me , against);
    fight(winner,giant);

    std::cout << fight(me, against ) << std::endl;
```

That one finally compiles and runs:

```
$ ./nandi4
Nandi 95 10
Barbarian 70 5
Nandi 90 10
```

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

But in this output Matyi, the eagle-eyed found out that something was not right. Well here it is better for the Reader if we sign the fights with `----------` in it:

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

instead of 2 there were 3 battles... our solution for this was to delete the last one (to be right, we put it in a comment)

Copy constructor copy constructor

And something really interesting happened! Just take a look on the last output line: `Nandi 60 10`, so it could happen that the `Nandi` named Soldier's HP is 60 while the int the previous fight against the giant it already went down to zero...? Now it has to stay as a mystery but soon, on the 6th day we will give you the answer.

```
    Soldier me{"Nandi", 100, 10};
    Soldier against{"Barbarian", 80, 5};
    Soldier giant{"Giant",180,20};
    Soldier winner =fight(me , against);
    fight(winner,giant);

   //std::cout << fight(me, against ) << std::endl;
```

to receive the wanted solution from the task's proposal, in this they pushed out the 2nd fight call to the channel then they were ready:

```
    Soldier me{"Nandi", 100, 10};
    Soldier against{"Barbarian", 80, 5};
    Soldier giant{"Giant",180,20};
    Soldier winner =fight(me , against);
    std::cout << fight(winner,giant ) << std::endl;
```

which now gives correctly what I asked for.

- Try it out, then explain this source code:

```
int main()
{
    Soldier me {"Nandi", 100, 10};
    Soldier against {"Barbarian", 80, 5};
    Soldier third {"Giant", 180, 20};

    std::cout << fight(fight(me, against), third) << std::endl;

    return 0;
}
```

They tried to answer it on paper, what Matyi phrased in a great way: TODO: when he said it in his own words that this is the two battles' shortened description.

For a checkup task they have to take the current code part as a sample and make it happen on the computer that the four Soldier have to fight, the fourth one with the current winner! There is their first try:

```
int main()
{
    Soldier me{"Nandi", 100, 10};
    Soldier against{"Barbarian", 80, 5};
    Soldier giant{"Giant",180,20};
    Soldier fourth{"MATYI",500,300};

    std::cout << fight(fight(me,against, )third)fourth) << std::endl;
}
```

which is obviously not able to compile because they have put the commas in totally wrong places:

```
$ 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 << fight(fight(me,against, )third)fourth) << std::endl;
```

They were able to fix this error really fast

```
    Soldier me{"Nandi", 100, 10};
    Soldier against{"Barbarian", 80, 5};
    Soldier giant{"Giant",180,20};
    Soldier fourth{"MATYI",500,300};
```

```
    std::cout << fight(fight(me,against ),third),fourth) << std::endl;
```

with what they already got a telling error:

```
$ g++ nandi4.cpp -o nandi4 -std=c++11
nandi4.cpp: In function ‘int main()’:
nandi4.cpp:59:50: error: ‘third’ was not declared in this scope
     std::cout << fight(fight(fight(me,against ),third),fourth) << std::me
```

as they have started from their code from yesterday where wasn't any `third` named reference but `giant` name was used instead of that so it was changed according to that:

```
$ g++ nandi4.cpp -o nandi4 -std=c++11
nandi4.cpp: In function ‘int main()’:
nandi4.cpp:74:61: error: expected ‘;’ before ‘)’ token
     std::cout << fight(fight(me,against ),third),fourth) << std::endl;
```

but it did not bring any success yet

```
$ g++ nandi4.cpp -o nandi4 -std=c++11
nandi4.cpp: In function ‘int main()’:
nandi4.cpp:74:61: error: expected ‘;’ before ‘)’ token
     std::cout << fight(fight(me,against ),third),fourth) << std::endl;
```

because there are three closing brackets and only two opening ones! The „How many fights do you want?" helping question worked because they wanted three, but there are only two fight function call, after discussing it's place they put the third one in:

```
    std::cout << fight(fight(fight(me,against ),third),fourth) << std::endl;
```

which already shows a good operation:

```
$ ./nandi4
Nandi 95 10
Barbarian 70 5
Nandi 90 10
Barbarian 60 5
Nandi 85 10
Barbarian 50 5
Nandi 80 10
Barbarian 40 5
Nandi 75 10
Barbarian 30 5
Nandi 70 10
Barbarian 20 5
Nandi 65 10
Barbarian 10 5
Nandi 60 10
Barbarian 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
```

Tasks written by the kids

- Gréta: three soldier had a championship. One is Gréta, her HP is 400, damage is 300. Second one is Nándi, his HP is 100, damage is 10. Third one Matyi, his HP is 100, damage is 5. The task is to code this programme!

  Other task is to create three soldiers and give them data, for example HP and damage. The reader could write any number when they are made.

  Other task is when the soldiers are fighting, if one won then it will receive it's HP again.

- Nándi: the task is to create two soldiers then make a battle with them! Give proper data, it's HP, damage, name is up to you.

- Matyi: the task is to make a cup! In a way that Nándi and Barbarian, Giant and Matyi fight with each other. The winners will have two battles. The battle's winner vs. Exploder, the other's winner vs. Wizard. After this the two winner fights each other.

  - Nándi: {HP: 100, damage 10}
  - Matyi: {HP: 500, damage 200}
  - Giant: {HP: 180, damage 20}
  - Barbarian: {HP: 80, damage 5}
  - Exploder: {HP: 25, damage 25}
  - Wizard: {HP: 60, damage 40}

Fifth day– on slippery ground

Let's transform a bit the `nandi4.cpp` source: put in some tracking, say outputs which will help to see on the screen what is happening in the programme.

```
#include <iostream>

class Soldier
{
public:
    std::string name;
    int HP;
    int damage;

    Soldier(std::string name, int HP, int damage) {

        this->name = name;
        this->HP = HP;
        this->damage = damage;
    }

    bool alive() {
        return HP > 0;
    }

    friend std::ostream &operator<< (std::ostream &stream, Soldier &Soldier) {
        stream << Soldier.name << " " << Soldier.HP << " " << Soldier.damage;
        return stream;
    }

};


Soldier &fight(Soldier &first, Soldier &second)
{
    std::cout << "fight func: fight start, " << first << " vs. " << second << std::endl;

    for (; first.alive() && second.alive() ;) {
```

```
34
35          first.HP = first.HP - second.damage;
36          second.HP = second.HP - first.damage;
37
38          std::cout << "fight func: " << first << std::endl;
39          std::cout << "fight func: " << second << std::endl;
40
41      }
42
43      if (first.HP > second.HP)
44          return first;
45      else
46          return second;
47  }
48
49
50  int main()
51  {
52
53      Soldier me {"Nandi", 100, 10};
54      Soldier against {"Barbarian", 80, 5};
55      Soldier third {"Giant", 180, 20};
56
57      std::cout << "main func: " << fight(fight(me, against), fight(me, third)) << std::endl;
58
59      return 0;
60  }
```

Dad – mentor

What do we expect? The `me` Soldier object (`Nandi`) puts down the `against` Soldier object (the `Barbarian`). The `third` Soldier object (`Giant`) puts down the `me` object, then the two winner: `me` and `third` plays, where `third` wins.

```
$ g++ nandi5.cpp -o nandi5 -std=c++11
$ ./nandi5
fight func: fight start, Nandi 100 10 vs. Giant 180 20
fight func: Nandi 80 10
fight func: Giant 170 20
fight func: Nandi 60 10
fight func: Giant 160 20
fight func: Nandi 40 10
fight func: Giant 150 20
fight func: Nandi 20 10
fight func: Giant 140 20
fight func: Nandi 0 10
fight func: Giant 130 20
fight func: fight start, Nandi 0 10 vs. Barbarian 80 5
fight func: fight start, Barbarian 80 5 vs. Giant 130 20
fight func: Barbarian 60 5
fight func: Giant 125 20
fight func: Barbarian 40 5
fight func: Giant 120 20
fight func: Barbarian 20 5
fight func: Giant 115 20
fight func: Barbarian 0 5
fight func: Giant 110 20
main func: Giant 110 20
```

On the other hand what are the tracking messages showing? What happened? The `third` Soldier object (`Giant`) put down the `me` (`Nandi`) object. Then the already weakened `me` Soldier object (`Nandi`) put down the `against` object the (`Barbarian`). Then the two winners: `against` and `third` plays, where `third` wins. Well, we didn't really count on this, did we?

The explanation: We thought when we played it through in our heads that first the `fight(me, against)` happens, then the `fight(me, third)` and in the end the two winners will play. There is a chance that a computer exists on which it will happen like this, but mine is not that computer! That's because the C++ language does not determine the order of the function call's ingoing parameters. In our case that the external fight function could start with either the right or the left side's fight call's execution. Because it is not determined that it will happen like this on the first computer and like that on the other one. That is the reason why we are on slipper ground. What can the programmer do? What does the programmer have to do? He has to avoid to write sources in which situations like this could appear. For this there are many opportunities, it's only questionable in style that which one will the programmer use. Right now it's enough to know that the C++ is not a playground (for example the Java or C# is) in the meaning of the programmer shouldn't only know properly what will the effect be of his programme on the objects in the memory will be, he HAS to know it perfectly!

The code will be much more readable if we remove the function calls from the function call, so we write this:

```
int main()
{

    Soldier me {"Nandi", 100, 10};
    Soldier against {"Barbarian", 80, 5};
    Soldier third {"Giant", 180, 20};

    Soldier firstWinner = fight(me, against);
    Soldier secondWinner = fight(me, third);

    std::cout << "main func: " << fight(firstWinner, secondWinner) << std::endl;

    return 0;
}
```

Here it is already obvious in C++ language what does the programmer want, but sadly that's still not what is happening yet:

```
fight func: fight start, Nandi 100 10 vs. Barbarian 80 5
fight func: Nandi 95 10
fight func: Barbarian 70 5
fight func: Nandi 90 10
fight func: Barbarian 60 5
fight func: Nandi 85 10
fight func: Barbarian 50 5
fight func: Nandi 80 10
fight func: Barbarian 40 5
fight func: Nandi 75 10
fight func: Barbarian 30 5
fight func: Nandi 70 10
fight func: Barbarian 20 5
fight func: Nandi 65 10
fight func: Barbarian 10 5
fight func: Nandi 60 10
fight func: Barbarian 0 5
fight func: fight start, Nandi 60 10 vs. Giant 180 20
fight func: Nandi 40 10
fight func: Giant 170 20
fight func: Nandi 20 10
fight func: Giant 160 20
fight func: Nandi 0 10
fight func: Giant 150 20
fight func: fight start, Nandi 60 10 vs. Giant 150 20
fight func: Nandi 40 10
fight func: Giant 140 20
fight func: Nandi 20 10
```

```
fight func: Giant 130 20
fight func: Nandi 0 10
fight func: Giant 120 20
main func: Giant 120 20
```

Because it could happen that in the third battle the `Nandi`'s HP is 60 again when it went down to zero before! Let's see a great solution finally, just put one 'and' sign after the Soldier class:

```
    Soldier& firstWinner = fight(me, against);
    Soldier& secondWinner = fight(me, third);
```

and let's see a miracle:

```
fight func: fight start, Nandi 100 10 vs. Barbarian 80 5
fight func: Nandi 95 10
fight func: Barbarian 70 5
fight func: Nandi 90 10
fight func: Barbarian 60 5
fight func: Nandi 85 10
fight func: Barbarian 50 5
fight func: Nandi 80 10
fight func: Barbarian 40 5
fight func: Nandi 75 10
fight func: Barbarian 30 5
fight func: Nandi 70 10
fight func: Barbarian 20 5
fight func: Nandi 65 10
fight func: Barbarian 10 5
fight func: Nandi 60 10
fight func: Barbarian 0 5
fight func: fight start, Nandi 60 10 vs. Giant 180 20
fight func: Nandi 40 10
fight func: Giant 170 20
fight func: Nandi 20 10
fight func: Giant 160 20
fight func: Nandi 0 10
fight func: Giant 150 20
fight func: fight start, Nandi 0 10 vs. Giant 150 20
main func: Giant 150 20
```

finally that is happening what the programmer imagined!

With this version the kids started to play in freestyle (spontaneously):

```
    Soldier me {"Matyi", 1000, 100};
    Soldier against {"Greta", 800, 50};
    Soldier third {"Nandi", 800, 50};

    Soldier &firstWinner = fight(me, against);
    Soldier &secondWinner = fight(me, third);

    std::cout << "main func: " << fight(firstWinner, secondWinner) << std::endl;
```

```
fight func: fight start, Matyi 1000 100 vs. Greta 800 50
fight func: Matyi 950 100
fight func: Greta 700 50
fight func: Matyi 900 100
fight func: Greta 600 50
fight func: Matyi 850 100
fight func: Greta 500 50
fight func: Matyi 800 100
```

```
fight func: Greta 400 50
fight func: Matyi 750 100
fight func: Greta 300 50
fight func: Matyi 700 100
fight func: Greta 200 50
fight func: Matyi 650 100
fight func: Greta 100 50
fight func: Matyi 600 100
fight func: Greta 0 50
fight func: fight start, Matyi 600 100 vs. Nandi 800 50
fight func: Matyi 550 100
fight func: Nandi 700 50
fight func: Matyi 500 100
fight func: Nandi 600 50
fight func: Matyi 450 100
fight func: Nandi 500 50
fight func: Matyi 400 100
fight func: Nandi 400 50
fight func: Matyi 350 100
fight func: Nandi 300 50
fight func: Matyi 300 100
fight func: Nandi 200 50
fight func: Matyi 250 100
fight func: Nandi 100 50
fight func: Matyi 200 100
fight func: Nandi 0 50
fight func: fight start, Matyi 200 100 vs. Matyi 200 100
fight func: Matyi 0 100
fight func: Matyi 0 100
main func: Matyi 0 100
```

But Matyi didn't want in the end that the `Matyi` name-attribute object to play on it's own... their next tries were about that the `Matyi` object should win in the end, but not to fight with itself. Attempt after attempt, but much to the delight of them, once the `Gréta`, once the `Nándi` object won, and of course few times the `Matyi`'s fight with itself came back till he realized that there are tasks which are impossible to solve within given borders. (Because if Matyi loses the first match then he won't go alive to the second one so he won't get into the finals. If this wouldn't happen, so he wins the first but fails the second then he would get to the finals but well... not alive.)

Let's brush up in the next point the first task that Gréta threw in!

Gréta, Nándi, Matyi

> „Other task is when the soldiers are fighting, if one won then it will receive it's HP again. "
>
> —Gréta

Sixth day – copy constructor

1

Dad – mentor

Gréta

Nándi

Matyi

Part III

The future's programming

From Erwin Schrödinger's researches connected with life[5] we know that the secret of life is the ability 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' uncontrolledness 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 also 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

Chapter 3

Index

D
DeepMind
    TensorFlow, *see* Google
DevRob, 2
DevRob2Psy, 2

E
ESAMU, 2

J
JIBO, 2

R
robotpsihology, 2
robotpszichológia, viii

S
Samu, 3
SRS, 2

T
TensorFlow
    TensorBoard, *see* DeepMind

U
UDPROG, 3