

Univerzális programozás

Írd meg a saját programozás tankönyvedet!

Ed. BHAX, DEBRECEN,
2019. február 19, v. 0.0.4

Copyright © 2019 Dr. Bátfai Norbert

Copyright (C) 2019, Norbert Bátfai Ph.D., batfai.norbert@inf.unideb.hu, nbatfai@gmail.com,

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".

<https://www.gnu.org/licenses/fdl.html>

Engedélyt adunk Önnek a jelen dokumentum sokszorosítására, terjesztésére és/vagy módosítására a Free Software Foundation által kiadott GNU FDL 1.3-as, vagy bármely azt követő verziójának feltételei alapján. Nincs Nem Változtatható szakasz, nincs Címlapszöveg, nincs Hátlapszöveg.

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

COLLABORATORS

	<i>TITLE :</i> Univerzális programozás		
<i>ACTION</i>	<i>NAME</i>	<i>DATE</i>	<i>SIGNATURE</i>
WRITTEN BY	Bátfai, Norbert	2019. március 26.	

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME
0.0.1	2019-02-12	Az iniciális dokumentum szerkezetének kialakítása.	nbatfai
0.0.2	2019-02-14	Inciális feladatlisták összeállítása.	nbatfai
0.0.3	2019-02-16	Feladatlisták folytatása. Feltöltés a BHAX csatorna https://gitlab.com/nbatfai/bhax repójába.	nbatfai
0.0.4	2019-02-19	Aktualizálás, javítások.	nbatfai

Ajánlás

„To me, you understand something only if you can program it. (You, not someone else!) Otherwise you don't really understand it, you only think you understand it.”

—Gregory Chaitin, *META MATH! The Quest for Omega*, [METAMATH]

Tartalomjegyzék

I. Bevezetés	1
1. Vízió	2
1.1. Mi a programozás?	2
1.2. Milyen doksikat olvassak el?	2
1.3. Milyen filmeket nézzek meg?	2
II. Tematikus feladatok	3
2. Helló, Turing!	5
2.1. Végtelen ciklus	5
2.2. Lefagyott, nem fagyott, akkor most mi van?	6
2.3. Változók értékének felcserélése	7
2.4. Labdapattogás	8
2.5. Szóhossz és a Linus Torvalds féle BogomIPS	10
2.6. Helló, Google!	11
2.7. 100 éves a Brun tétel	11
2.8. A Monty Hall probléma	11
3. Helló, Chomsky!	12
3.1. Decimálisból unárisba átváltó Turing gép	12
3.2. Az $a^n b^n c^n$ nyelv nem környezetfüggetlen	12
3.3. Hivatkozási nyelv	13
3.4. Saját lexikális elemző	13
3.5. l33t.1	14
3.6. A források olvasása	14
3.7. Logikus	15
3.8. Deklaráció	16

4. Helló, Caesar!	18
4.1. int *** háromszögmátrix	18
4.2. C EXOR titkosító	19
4.3. Java EXOR titkosító	20
4.4. C EXOR törő	20
4.5. Neurális OR, AND és EXOR kapu	22
4.6. Hiba-visszaterjesztéssel perceptron	23
5. Helló, Mandelbrot!	24
5.1. A Mandelbrot halmaz	24
5.2. A Mandelbrot halmaz a std::complex osztállyal	27
5.3. Biomorfok	29
5.4. A Mandelbrot halmaz CUDA megvalósítása	32
5.5. Mandelbrot nagyító és utazó C++ nyelven	36
5.6. Mandelbrot nagyító és utazó Java nyelven	40
6. Helló, Welch!	41
6.1. Első osztályom	41
6.2. LZW	41
6.3. Fabejárás	41
6.4. Tag a gyökér	41
6.5. Mutató a gyökér	42
6.6. Mozgató szemantika	42
7. Helló, Conway!	43
7.1. Hangyaszimulációk	43
7.2. Java életjáték	43
7.3. Qt C++ életjáték	43
7.4. BrainB Benchmark	44
8. Helló, Schwarzenegger!	45
8.1. Szoftmax Py MNIST	45
8.2. Szoftmax R MNIST	45
8.3. Mély MNIST	45
8.4. Deep dream	45
8.5. Robotpszichológia	46

9. Helló, Chaitin!	47
9.1. Iteratív és rekurzív faktoriális Lisp-ben	47
9.2. Weizenbaum Eliza programja	47
9.3. Gimp Scheme Script-fu: króm effekt	47
9.4. Gimp Scheme Script-fu: név mandala	47
9.5. Lambda	48
9.6. Omega	48
 III. Második felvonás	 49
10. Helló, Arroway!	51
10.1. A BPP algoritmus Java megvalósítása	51
10.2. Java osztályok a Pi-ben	51
 11. Helló, Gutenberg!	 52
11.1. Juhász István: Magas szintű programozási nyelvek 1	52
11.2. Kernigan-Ritchie: A C programozási nyelv	54
11.3. Benedek-Levendovszky: Szoftverfejlesztés C++ nyelven	56
 IV. Irodalomjegyzék	 59
11.4. Általános	60
11.5. C	60
11.6. C++	60
11.7. Lisp	60

Előszó

Amikor programozónak terveztem állni, ellenezték a környezetemben, mondván, hogy kell szövegszerkesztő meg táblázatkezelő, de az már van... nem lesz programozói munka.

Tévedtek. Hogy egy generáció múlva kell-e még tömegesen hús-vér programozó vagy olcsóbb lesz allokálni igény szerint pár robot programozót a felhőből? A programozók dolgozók lesznek vagy papok? Ki tudhatná ma.

Mindenesetre a programozás a teoretikus kultúra csúcsa. A GNU mozgalomban látom annak garanciáját, hogy ebben a szellemi kalandban a gyerekeim is részt vehessenek majd. Ezért programozunk.

Hogyan forgasd

A könyv célja egy stabil programozási szemlélet kialakítása az olvasóban. Módszere, hogy hetekre bontva ad egy tematikus feladatcsokrot. Minden feladathoz megadja a megoldás forráskódját és forrásokat feldolgozó videókat. Az olvasó feladata, hogy ezek tanulmányozása után maga adja meg a feladat megoldásának lényegi magyarázatát, avagy írja meg a könyvet.

Miért univerzális? Mert az olvasótól (kvázi az írótól) függ, hogy kinek szól a könyv. Alapértelmezésben gyerekeknek, mert velük készítem az iniciális változatot. Ám tervezem felhasználását az egyetemi programozás oktatásban is. Ahogy szélesedni tudna a felhasználók köre, akkor lehetne kiadása különböző korosztályú gyerekeknek, családoknak, szakköröknek, programozás kurzusoknak, felnőtt és továbbképzési műhelyeknek és sorolhatnánk...

Milyen nyelven nyomjuk?

C (mutatók), C++ (másoló és mozgató szemantika) és Java (lebutított C++) nyelvekből kell egy jó alap, ezt kell kiegészíteni pár R (vektoros szemlélet), Python (gépi tanulás bevezető), Lisp és Prolog (hogyan lássuk mást is) példával.

Hogyan nyomjuk?

Ránts le a <https://gitlab.com/nbatfai/bhax> git repót, vagy méginkább forkolj belőle magadnak egy sajátot a GitLabon, ha már saját könyvön dolgozol!

Ha megvannak a könyv DocBook XML forrásai, akkor az alább látható **make** parancs ellenőrzi, hogy „jól formázottak” és „érvényesek-e” ezek az XML források, majd elkészíti a dlatex programmal a könyved pdf változatát, íme:

```
batfai@entropy:~$ cd glrepos/bhax/thematic_tutorials/bhax_textbook/
batfai@entropy:~/glrepos/bhax/thematic_tutorials/bhax_textbook$ make
rm -f bhax-textbook-fdl.pdf
xmllint --xinclude bhax-textbook-fdl.xml --output output.xml
xmllint --relaxng http://docbook.org/xml/5.0/rng/docbookxi.rng output.xml  ←
--noout
output.xml validates
rm -f output.xml
dlatex bhax-textbook-fdl.xml -p bhax-textbook.xls
Build the book set list...
Build the listings...
XSLT stylesheets DocBook - LaTeX 2e (0.3.10)
=====
Stripping NS from DocBook 5/NG document.
Processing stripped document.
Image 'dlatex' not found
Build bhax-textbook-fdl.pdf
'bhax-textbook-fdl.pdf' successfully built
```

Ha minden igaz, akkor most éppen ezt a legenerált `bhax-textbook-fdl.pdf` fájlt olvasod.



A DocBook XML 5.1 új neked?

Ez esetben forgasd a <https://tdg.docbook.org/tdg/5.1/> könyvet, a végén találsz az informatikai szövegek jelölésére használható gazdag „API” elemenkénti bemutatását.

I. rész

Bevezetés

1. fejezet

Vízió

1.1. Mi a programozás?

1.2. Milyen doksikat olvassak el?

- Olvasgasd a kézikönyv lapjait, kezd a **man man** parancs kiadásával. A C programozásban a 3-as szintű lapokat fogod nézegetni, például az első feladat kapcsán ezt a **man 3 sleep** lapot
- [[KERNIGHANRITCHIE](#)]
- [[BMECPP](#)]
- Az igazi kockák persze csemegéznek a C nyelvi szabvány [ISO/IEC 9899:2017](#) kódcsipeteiből is.

1.3. Milyen filmeket nézzek meg?

- 21 - Las Vegas ostroma, <https://www.imdb.com/title/tt0478087/>, benne a **Monty Hall probléma** bemutatása.

II. rész

Tematikus feladatok

**Bátf41 Haxor Stream**

A feladatokkal kapcsolatos élő adásokat sugároz a <https://www.twitch.tv/nbatfai> csatorna, melynek permanens archívuma a <https://www.youtube.com/c/nbatfai> csatornán található.

DRAFT

2. fejezet

Helló, Turing!

2.1. Végtelen ciklus

Írj olyan C végtelen ciklusokat, amelyek 0 illetve 100 százalékban dolgoztatnak egy magot és egy olyat, amely 100 százalékban minden magot!

Megoldás forrása ha a magot 100%-on akarjuk dolgoztatni: <https://sourceforge.net/p/udprog/code/ci/master/tree/source/kezdo/elsoc/vegtelen/v.c>

```
int main ()
{
    for (;;)

    return 0;
}
```

Megoldás forrása ha a magot 0%-on akarjuk dolgoztatni: <https://sourceforge.net/p/udprog/code/ci/master/tree/source/kezdo/elsoc/vegtelen/vs.c>

```
#include <unistd.h>
int main ()
{
    for (;;)
        sleep (1);
    return 0;
}
```

Megoldás ha minden magot 100%-on akarunk dolgoztatni:

```
#include <omp.h>
int main()
{
    #pragma omp parallel
    for(;;) {}
}
```

A sleep függvény alvó/várakozó állapotba küldi a magot ezért a terhelés 0%-os lesz. A sleep elhagyása nélkül a program egy magot 100%-on dolgoztat. Ha az első példát paralelel minden magon futtatjuk akkor minden mag 100%-osan le lesz terhelve.

2.2. Lefagyott, nem fagyott, akkor most mi van?

Mutasd meg, hogy nem lehet olyan programot írni, amely bármely más programról eldönti, hogy le fog-e fagyni vagy sem!

Megoldás forrása: tegyük fel, hogy akkora haxorok vagyunk, hogy meg tudjuk írni a Lefagy függvényt, amely tetszőleges programról el tudja dönteni, hogy van-e benne végtelen ciklus:

```
Program T100
{
    boolean Lefagy(Program P)
    {
        if(P-ben van végtelen ciklus)
            return true;
        else
            return false;
    }

    main(Input Q)
    {
        Lefagy(Q)
    }
}
```

A program futtatása, például akár az előző v. c ilyen pszeudókódjára:

```
T100(t.c.pseudo)
true
```

akár önmagára

```
T100(T100)
false
```

ezt a kimenetet adja.

A T100-as programot felhasználva készítsük most el az alábbi T1000-set, amelyben a Lefagy-ra építő Lefagy2 már nem tartalmaz feltételezett, csak konkrét kódot:

```
Program T1000
{
    boolean Lefagy(Program P)
    {
        if(P-ben van végtelen ciklus)
            return true;
    }
}
```

```
    else
        return false;
}

boolean Lefagy2(Program P)
{
    if(Lefagy(P))
        return true;
    else
        for(;;);
}

main(Input Q)
{
    Lefagy2(Q)
}

}
```

Mit for kiírni erre a T1000 (T1000) futtatásra?

- Ha T1000 lefagyó, akkor nem fog lefagyni, kiírja, hogy true
- Ha T1000 nem fagyó, akkor pedig le fog fagyni...

akkor most hogy fog működni? Sehogy, mert ilyen Lefagy függvényt, azaz a T100 program nem is létezik.

Tanulságok, tapasztalatok, magyarázat...

2.3. Változók értékének felcserélése

Írj olyan C programot, amely felcseréli két változó értékét, bármiféle logikai utasítás vagy kifejezés használata nélkül!

Megoldás videó: https://bhaxor.blog.hu/2018/08/28/10_begin_goto_20_avagy_elindulunk

Megoldás forrása segédváltozó nélkül: <https://sourceforge.net/p/udprog/code/ci/master/tree/source/kezdol-elsoc/valtozocsere/csere.c>

```
#include<stdio.h>

int main()
{
    int valtozo_1 = 2, valtozo_2 = 4;

    printf("valtozo_1=%d valtozo_2=%d\n", valtozo_1, valtozo_2);
```



```
    valtozo_1 = ( valtozo_1 - valtozo_2 );
    valtozo_2 = ( valtozo_1 + valtozo_2 );
    valtozo_1 = ( valtozo_2 - valtozo_1 );

    printf("valtozo_1=%d valtozo_2=%d\n",valtozo_1, valtozo_2);

    return 0;
}
```

Megoldás forrása segédváltozóval:

```
int main()
{
    int v1=1, v2=2, v3;
    v3=v1;
    v1=v2;
    v2=v3;
    return 0;
}
```

A csere segédváltozóval bevezet egy új ideiglenes változót, amiben az első változó értékét tároljuk amíg abba belereakjuk a 2. változó értékét mert ekkor a v1 értéke elvész. Ezután a második változóba belerakjuk a segédváltozóban eltárolt első változó értékét.

Segédváltozó nélküli csere csak egyszerű kivonás meg összeadás.

2.4. Labdapattogás

Először if-ekkel, majd bármiféle logikai utasítás vagy kifejezés nasználata nélkül írj egy olyan programot, ami egy labdát pattogtat a karakteres konzolon! (Hogy mit értek pattogtatás alatt, alább láthatod a videókon.)

Megoldás videó: <https://bhaxor.blog.hu/2018/08/28/labdapattogas>

Megoldás forrása: <https://sourceforge.net/p/udprog/code/ci/master/tree/source/kezdo/elsoc/pattog/pattog.c>

```
/* Készítette: 'Kojak' felhasználó
Dátum: 2014-02-24

A megoldás hibái javítva Józan Csaba splint ellenőrzése alapján. (Sipos
Ádám)
Dátum: 2015-02-08
*/

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

```
static void gotoxy(int x, int y)                                //kurzor pozicionálása
{
    int i;
    for(i=0; i<y; i++) printf("\n");                          //lefelé tolás
    for(i=0; i<x; i++) printf(" ");                          //jobbra tolás
    printf("o\n");      //labda ikonja
}

void usleep(int);
int main(void)
{
    int egyx=1;
    int egyy=-1;
    int i;
    int x=10;    //a labda kezdeti pozíciója
    int y=20;
    int ty[23]; //magasság      // a pálya mérete
    int tx[80]; //szélesség

    //pálya széleinek meghatározás

    for(i=0; i<23; i++)
        ty[i]=1;

    ty[1]=-1;
    ty[22]=-1;

    for(i=0; i<79; i++)
        tx[i]=1;

    tx[1]=-1;
    tx[79]=-1;

    for(;;)
    {
        //címsor és pozíció kijelzése
        for(i=0; i<36; i++)
            printf("_");

        printf("x=%2d", x);
        printf("y=%2d", y);

        for(i=0; i<=35; i++)
            printf("_");
    }
}
```

```
(void) gotoxy(x,y);  
//printf("o\n"); Áthelyezve a gotoxy függvényre  
  
x+=egy x;  
y+=egy y;  
  
egy x*=tx[x];  
egy y*=ty[y];  
  
usleep (200000);  
(void) system("clear");  
}  
  
}
```

Tanulságok, tapasztalatok, magyarázat...

2.5. Szóhossz és a Linus Torvalds féle BogoMIPS

Írj egy programot, ami megnézi, hogy hány bites a szó a gépeden, azaz mekkora az int mérete. Használd ugyanazt a while ciklus fejet, amit Linus Torvalds a BogoMIPS rutinjában!

Megoldás forrása: <https://sourceforge.net/p/udprog/code/ci/master/tree/source/kezdo/elsoc/szohossz/szohossz.c#>

```
#include <stdio.h>  
int  
main (void)  
{  
    int h = 0;  
    int n = 0x01;  
    do  
        ++h;  
    while (n <= 1);  
    printf ("A szóhossz ezen a gépen: %d bites\n", h);  
    return 0;  
}
```

A változó értékét egyre állítjuk, létrehozunk egy számlálót, és a változót 1-el balra shifteljük, miközben a számlálót is növeljük ciklusonként. A ciklus addig megy amíg 2 számrendszerbeli számban van egyes. (1 bájt = 00000000, az 1 egy bájtban ábrázolva 00000001 ezt az egyest shifteljük balra eggyel 00000010. Ha kiesik az egyes akkor a while-ban a logikai érték hamis lesz mert az n=0. Ezért a számláló megmutatja hogy hány bites egy int.)

2.6. Helló, Google!

Írj olyan C programot, amely egy 4 honlapból álló hálózatra kiszámolja a négy lap Page-Rank értékét!

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

2.7. 100 éves a Brun tétel

Írj R szimulációt a Brun tétel demonstrálására!

Megoldás videó: <https://youtu.be/xbYhp9G6VqQ>

Megoldás forrása: https://gitlab.com/nbatfai/bhax/blob/master/attention_raising/Primek_R

2.8. A Monty Hall probléma

Írj R szimulációt a Monty Hall problémára!

Megoldás videó: https://bhaxor.blog.hu/2019/01/03/erdos_pal_mit_keresett_a_nagykonyvben_a_monty_hall-paradoxon_kapcsan

Megoldás forrása: https://gitlab.com/nbatfai/bhax/tree/master/attention_raising/MontyHall_R

Tanulságok, tapasztalatok, magyarázat...

3. fejezet

Helló, Chomsky!

3.1. Decimálisból unárisba átváltó Turing gép

Állapotátmenet gráfiával megadva írd meg ezt a gépet!

Megoldás forrása:

```
#include <stdio.h>
int main()
{
    int x;
    printf("Tizes szamrendszerbeli szam:");
    scanf("%d", &x);
    printf("\nUnaris alakja:");
    for(int i = 0; i < x; i++)
    {
        printf("/");
    }
    printf("\n");
    return 0;
}
```

Decimálisból unáris (egyes számrendszer) számrendszerbe való átváltás során csak a 10-es számrendszerben megadott számú tetszőleges karaktert kell leírni. A fenti példában ez egy for ciklussal van megoldva de bármilyen más megoldás is jó ha az megfelelő számú tetszőleges karakter ad vissza eredményül.

3.2. Az $a^n b^n c^n$ nyelv nem környezetfüggetlen

Mutass be legalább két környezetfüggő generatív grammatikát, amely ezt a nyelvet generálja!

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

3.3. Hivatkozási nyelv

A [KERNIGHANRITCHIE] könyv C referencia-kézikönyv/Utasítások melléklete alapján definiáld BNF-ben a C utasítás fogalmát! Majd mutass be olyan kódcsipeteket, amelyek adott szabvánnyal nem fordulnak (például C89), mással (például C99) igen.

```
<nem terminális> ::= konkatenációja terminálisoknak, nem terminálisoknak,  
illetve {iteráció}, [opcionális], alter|natíva  
<egész szám> ::= <előjel><szám>  
<előjel> ::= [-|+]  
<szám> ::= <számjegy>{<számjegy>}  
<számjegy> ::= 0|1|2|3|4|5|6|7|8|9
```

```
<címkézett_utasítás> ::= <azonosító> | case | default  
<kifejezésutasítás> ::= <kifejezés>  
<deklarációs_lista> ::= <deklaráció>  
<utasítás_lista> ::= <utasítás>  
<összetett_utasítás> ::= <deklarációs_lista> | <utasítás_lista>  
<kiválasztó_utasítás> ::= if | if else | switch  
<iterációs_utasítás> ::= while | do while | for  
<vezérlésátadó_utasítás> ::= goto | continue | break | return
```

Példák a szabványok közötti különbségekre:

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

A fenti példa C89-es szabvánnyal nem fordul le mí C99-es szabvánnyal le fordul mert abban már megengedett a // -el való kommentelés, míg a C89-es szabványban csak a /* és a */ voltak kommentet jelző szimbólumok. Szóval a fenti példa C89-es szabványban így nézne ki:

```
int main(){  
  /*int a;*/  
  return 0;  
}
```

Ez mind a 2 szabványban le fog fordulni.

3.4. Saját lexikális elemző

Írj olyan programot, ami számolja a bemenetén megjelenő valós számokat! Nem elfogadható olyan megoldás, amely maga olvassa betűnként a bemenetet, a feladat lényege, hogy lexert használjunk, azaz óriások vállán álljunk és ne kispályázzunk!

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

3.5. l33t.l

Lexelj össze egy l33t ciphert!

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

3.6. A források olvasása

Hogyan olvasod, hogyan értelmezed természetes nyelven az alábbi kódcsipeteket? Például

```
if(signal(SIGINT, jelkezelő)==SIG_IGN)
    signal(SIGINT, SIG_IGN);
```

Ha a SIGINT jel kezelése figyelmen kívül volt hagyva, akkor ezen túl is legyen figyelmen kívül hagyva, ha nem volt figyelmen kívül hagyva, akkor a jelkezelő függvény kezelje. (Miután a **man 7 signal** lapon megismertem a SIGINT jelet, a **man 2 signal** lapon pedig a használt rendszerhívást.)



Bugok

Vigyázz, sok csipet kerülendő, mert bugokat visz a kódba! Melyek ezek és miért? Ha nem megyránzésre, elkapja valamelyiket esetleg a splint vagy a frama?

i.

```
if(signal(SIGINT, SIG_IGN)!=SIG_IGN)
    signal(SIGINT, jelkezelő);
```

Ha a SIGINT nem volt figyelmen kívül hagyva akkor a jelkezelő kezelje. Ha figyelmen kívül volt hagyva továbbra is maradjon úgy.

ii.

```
for(i=0; i<5; ++i)
```

A függvény 5x fog lefutni, viszont figyelni kell arra hogy az i változót még a for ciklus előtt deklarálni kell, és ha az i-t ++i-vel növeltük akkor lehetőleg továbbra is úgy használjuk hogy könnyebben érthető maradjon a kód.

iii.

```
for(i=0; i<5; i++)
```

Itt is ugyanaz igaz mint az előzőnél. A ciklus 5x le fog futni ha a ciklusváltozót deklarátuk a ciklus előtt.

iv.

```
for(i=0; i<5; tomb[i] = i++)
```

Nyelvtani hiba nincs ha már létrehoztuk a tomb nevű tömböt és az i-t. Le is fog fordulni, viszont az eredmény bugos lesz.

v.

```
for(i=0; i<n && (*d++ = *s++); ++i)
```

A kód hibamentes ha már létrehoztuk a látható változókat és mutatókat.

vi.

```
printf("%d %d", f(a, ++a), f(++a, a));
```

A printf függvény ki fog írni 2 decimális számot ha már megvan az f függvény, az a változó, és ha az a változó megfelelő típusú az f függvényhez. Arra kell figyelni hogy ha az f függvény visszatérési értéke nem int akkor a kiírt értékek nem biztos hogy pontosak lesznek.

vii.

```
printf("%d %d", f(a), a);
```

A printf ki fogja írni az f függvény visszatérési értékét a-ra decimális alakban, és a értékét decimális alakban. Itt is ugyanarra kell figyelni mint az előbb. Nyelvtani hiba nincs a kódrészletben.

viii.

```
printf("%d %d", f(&a), a);
```

A kiírás megtörténik viszont az f függvény most az a változó memória címével fog dolgozni nem az a értékével ha ezt akarjuk akkor nincs semmi gond.

Tanulságok, tapasztalatok, magyarázat...: Figyeljünk hogy hogyan használjuk az operátorokat, figyeljünk a kiértékelés és az értékadás sorrendjére (i++, ++i), és ismerjük a függvényeket amiket használunk.

3.7. Logikus

Hogyan olvasod természetes nyelven az alábbi Ar nyelvű formulákat?

```
$(\text{forall } x \text{ } \text{exists } y ((x < y) \wedge (y \text{ prime})))$
```

Végtelen sok prím van.

```
$(\text{forall } x \text{ } \text{exists } y ((x < y) \wedge (y \text{ prime})) \wedge (\text{SSy } \text{prime})) \leftarrow$
```

Végtelen sok ikerprím van.

```
$(\text{exists } y \text{ } \text{forall } x (x \text{ prime}) \supset (x < y))$
```

Véges sok prím van.

```
$(\text{exists } y \text{ } \text{forall } x (y < x) \supset \neg (x \text{ prime}))$
```

Véges sok prím van.

Megoldás forrása: https://gitlab.com/nbatfai/bhax/blob/master/attention_raising/MatLog_LaTeX

Megoldás videó: <https://youtu.be/ZexiPy3ZxsA>, https://youtu.be/AJSXOQFF_wk

3.8. Deklaráció

Vezesd be egy programba (forduljon le) a következőket:

- egész

```
int a;
```

- egészre mutató mutató

```
int *b;
```

- egész referenciája

```
int &r;
```

- egészek tömbje

```
int t[5];
```

- egészek tömbjének referenciája (nem az első elemé)

```
int (&tr)[5] = t;
```

- egészre mutató mutatók tömbje

```
int *d[5];
```

- egészre mutató mutatót visszaadó függvény

```
int *h();
```

- egészre mutató mutatót visszaadó függvényre mutató mutató

```
int *(*h)();
```

- egészet visszaadó és két egészet kapó függvényre mutató mutatót visszaadó, egészet kapó függvény

```
int (*v(int c))(int a, int b);
```

- függvénymutató egy egészet visszaadó és két egészet kapó függvényre mutató mutatót visszaadó, egészet kapó függvényre

```
int (*( *z)(int))(int, int);
```

Mit vezetnek be a programba a következő nevek?

- ```
int a;
```

Egy egész típusú változót.

- ```
int *b = &a;
```

Egy egész típusú mutatót ami a-ra mutat.

- ```
int &r = a;
```

a változónak a referenciája.

- ```
int c[5];
```

Egy 5 elemű egész típusú tömböt.

- ```
int (&tr)[5] = c;
```

Egészek tömbjének referenciáját.

- ```
int *d[5];
```

5 elemű int-re mutató mutatók tömbjét.

- ```
int *h ();
```

Egy függvényt ami int-re mutató mutatót ad vissza.

- ```
int *(*l) ();
```

Egy int-re mutató mutatót visszaadó függvényre mutató mutatót.(pl. az előző függvényre)

- ```
int (*v (int c)) (int a, int b)
```

int-et visszaadó, két intet kapó függvényre mutató mutatót visszaadó egészet kapó függvényt.

- ```
int ((*z) (int)) (int, int);
```

int-et visszaadó, két intet kapó függvényre mutató mutatót visszaadó egészet kapó függvényre mutató mutatót.

4. fejezet

Helló, Caesar!

4.1. int *** háromszögmátrix

Megoldás forrása:

```
#include <stdio.h>
#include <stdlib.h>
int main ()
{
    int nr = 3;
    double **tm;

    printf("%p\n", &tm);
    if ((tm = (double **) malloc (nr * sizeof (double *))) == NULL)
    {
        return -1;
    }
    printf("%p\n", tm);
    for (int i = 0; i < nr; ++i)
    {
        if ((tm[i] = (double *) malloc ((i + 1) * sizeof (double))) == NULL)
        {
            return -1;
        }
    }
    printf("%p\n", tm[0]);

    for (int i = 0; i < nr; ++i)
    for (int j = 0; j < i + 1; ++j)
        tm[i][j] = i * (i + 1) / 2 + j;

    for (int i = 0; i < nr; ++i)
    {
        for (int j = 0; j < i + 1; ++j)
            printf ("%f, ", tm[i][j]);
```

```
printf ("\n");
}
for (int i = 0; i < nr; ++i)
{
for (int j = 0; j < i + 1; ++j)
printf ("%f, ", tm[i][j]);
printf ("\n");
}
for (int i = 0; i < nr; ++i)
free (tm[i]);
free (tm);
return 0;
}
```

Ha a négyzetes mátrix főátlója alatt vagy felett minden elem 0, a mátrix háromszögmátrix. Az alsó háromszögmátrix elemeit sorfolytonosan bejárva el tudjuk helyezni egy tömbben, az így kapott elemek száma $n(n+1)/2$ lesz. A `malloc` függvény képes lefoglalni a dinamikus területen egy, a paramétereként kapott méretű területet. A `free` függvény felszabadítja a lefoglalt területet.

4.2. C EXOR titkosító

Írj egy EXOR titkosítót C-ben!

Megoldás forrás:

```
#include <stdio.h>
#include <unistd.h>
#include <string.h>
#define MAX_KULCS 100
#define BUFFER_MERET 256
int main (int argc, char **argv)
{
char kulcs[MAX_KULCS];
char buffer[BUFFER_MERET];
int kulcs_index = 0;
int olvasott_bajtok = 0;
int kulcs_meret = strlen (argv[1]);
strncpy (kulcs, argv[1], MAX_KULCS);
while ((olvasott_bajtok = read (0, (void *) buffer, BUFFER_MERET))
{
for (int i = 0; i < olvasott_bajtok; ++i)
{
buffer[i] = buffer[i] ^ kulcs[kulcs_index];
kulcs_index = (kulcs_index + 1) % kulcs_meret;
}
write (1, buffer, olvasott_bajtok);
}
}
```

Ez a titkosítási módszer a xor művelet azonosságain alapul: $A \text{ XOR } B = C$ és $C \text{ XOR } B = A$ (A=tiszta bemeneti adat, B=titkosító kulcs, C=kimenő titkosított adat)

4.3. Java EXOR titkosító

Írj egy EXOR titkosítót Java-ban!

Megoldás videó:

```
public class Titkosito {
    public Titkosito(String kulcsSzöveg,
        java.io.InputStream bejövőCsatorna,
        java.io.OutputStream kimenőCsatorna)
        throws java.io.IOException {
        byte [] kulcs = kulcsSzöveg.getBytes();
        byte [] buffer = new byte[256];
        int kulcsIndex = 0;
        int olvasottBájtok = 0;
        while((olvasottBájtok =
            bejövőCsatorna.read(buffer)) != -1) {
            for(int i=0; i<olvasottBájtok; ++i) {
                buffer[i] = (byte)(buffer[i] ^ kulcs[kulcsIndex]);
                kulcsIndex = (kulcsIndex+1) % kulcs.length;
            }
            kimenőCsatorna.write(buffer, 0, olvasottBájtok);
        }
    }
    public static void main(String[] args) {
        try {
            new Titkosito(args[0], System.in, System.out);
        } catch(java.io.IOException e) {
            e.printStackTrace();
        }
    }
}
```

Itt csak nyelvi különbségek vannak a feladatok között.

4.4. C EXOR törő

Írj egy olyan C programot, amely megtöri az első feladatban előállított titkos szövegeket!

```
#define MAX_TITKOS 4096
#define OLVASAS_BUFFER 256
#define KULCS_MERET 8
#define _GNU_SOURCE
#include <stdio.h>
```

```
#include <unistd.h>
#include <string.h>

double
atlagos_szohossz (const char *titkos, int titkos_meret)
{
    int sz = 0;
    for (int i = 0; i < titkos_meret; ++i)
        if (titkos[i] == ' ')
            ++sz;

    return (double) titkos_meret / sz;
}

int tiszta_lehet (const char *titkos, int titkos_meret)
{
    double szohossz = atlagos_szohossz (titkos, titkos_meret);
    return szohossz > 6.0 && szohossz < 9.0
        && strcasestr (titkos, "hogy") && strcasestr (titkos, "nem")
        && strcasestr (titkos, "az") && strcasestr (titkos, "ha");
}

void exor (const char kulcs[], int kulcs_meret, char titkos[], int ←
    titkos_meret)
{
    int kulcs_index = 0;
    for (int i = 0; i < titkos_meret; ++i)
    {
        titkos[i] = titkos[i] ^ kulcs[kulcs_index];
        kulcs_index = (kulcs_index + 1) % kulcs_meret;
    }
}

int exor_tores (const char kulcs[], int kulcs_meret, char titkos[],
    int titkos_meret)
{
    exor (kulcs, kulcs_meret, titkos, titkos_meret);
    return tiszta_lehet (titkos, titkos_meret);
}

int
main (void)
{
    char kulcs[KULCS_MERET];
    char titkos[MAX_TITKOS];
    char *p = titkos;
    int olvasott_bajtok;
    // titkos fajt berantasa
    while ((olvasott_bajtok =
        read (0, (void *) p,
            (p - titkos + OLVASAS_BUFFER <
                MAX_TITKOS) ? OLVASAS_BUFFER : titkos + MAX_TITKOS - p)))
        p += olvasott_bajtok;
```

```
// maradék hely nullazasa a titkos bufferben
for (int i = 0; i < MAX_TITKOS - (p - titkos); ++i)
    titkos[p - titkos + i] = '\0';
// osszes kulcs eloallitasa
for (int ii = '0'; ii <= '9'; ++ii)
    for (int ji = '0'; ji <= '9'; ++ji)
        for (int ki = '0'; ki <= '9'; ++ki)
            for (int li = '0'; li <= '9'; ++li)
                for (int mi = '0'; mi <= '9'; ++mi)
                    for (int ni = '0'; ni <= '9'; ++ni)
                        for (int oi = '0'; oi <= '9'; ++oi)
                            for (int pi = '0'; pi <= '9'; ++pi)
                                {
                                    kulcs[0] = ii;
                                    kulcs[1] = ji;
                                    kulcs[2] = ki;
                                    kulcs[3] = li;
                                    kulcs[4] = mi;
                                    kulcs[5] = ni;
                                    kulcs[6] = oi;
                                    kulcs[7] = pi;

                                    if (exor_tores (kulcs, KULCS_MERET, titkos, p - titkos))
                                        printf
                                        ("Kulcs: [%c%c%c%c%c%c%c%c]\nTiszta szoveg: [%s]\n",
                                         ii, ji, ki, li, mi, ni, oi, pi, titkos);

                                    // ujra EXOR-ozunk, igy nem kell egy masodik buffer
                                    exor (kulcs, KULCS_MERET, titkos, p - titkos);
                                }

return 0;
}
```

Ez az exor törő az előző feladattal működik, az ha a bemeneti szöveg magyar nyelvű és a titkosító kulcs 8 számjegyből áll. Nagyjából ez csak egy sima brute force algoritmus ami minden lehetséges kulccsal elvégzi a törést és ha az eredmény a feltételeknek (atlagos_szohossz és tiszta_lehet függvények) megfelel akkor azt adja vissza eredményül.

4.5. Neurális OR, AND és EXOR kapu

Megoldás videó: <https://youtu.be/Koyw6IH5ScQ>

Megoldás forrása: https://gitlab.com/nbatfai/bhax/blob/master/attention_raising/NN_R/mn.r

Neurális hálónak nevezzük azt a párhuzamos működésre képes információfeldolgozó eszközt, amely nagyszámú, hasonló típusú elem összekapcsolt rendszeréből áll Ezenfelül egyik legfontosabb jellemzője az hogy rendelkezik tanulási algoritmussal és képes előhívni a megtanult információt.

4.6. Hiba-visszaterjesztéses perceptron

C++

Megoldás forrása: <https://github.com/nbatfai/nahshon/blob/master/ql.hpp#L64>

Tanulságok, tapasztalatok, magyarázat...

DRAFT

5. fejezet

Helló, Mandelbrot!

5.1. A Mandelbrot halmaz

Mandelbrot-halmaz c komplex számokból áll melyekre az alábbi rekurzív sorozat nem tart a végtelenbe. $x_1 := c$ és $x_{n+1} := x_n * x_n + c$. Benoit Mandelbrot fedezte fel 1980-ban.

Megoldás videó: <https://youtu.be/gvaqijHlRU8> Megoldás Forrása: https://gitlab.com/nbatfai/bhax/blob/master/attention_raising/CUDA/mandelpngt.c++

```
// mandelpngt.c++
// Copyright (C) 2019
// Norbert Bátfai, batfai.norbert@inf.unideb.hu
//
// This program is free software: you can redistribute it and/or modify
// it under the terms of the GNU General Public License as published by
// the Free Software Foundation, either version 3 of the License, or
// (at your option) any later version.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
// GNU General Public License for more details.
//
// You should have received a copy of the GNU General Public License
// along with this program. If not, see <https://www.gnu.org/licenses/>.
//
// Version history
//
// Mandelbrot png
// Programozó Páternosztter/PARP
// https://www.tankonyvtar.hu/hu/tartalom/tamop412A/2011-0063 ↩
// _01_parhuzamos_prog_linux
//
// https://youtu.be/gvaqijHlRU8
//
```

```
#include <iostream>
#include "png++/png.hpp"
#include <sys/times.h>

#define MERET 600
#define ITER_HAT 32000

void
mandel (int kepadat[MERET][MERET]) {

    // MÉRÜNK IDŐT (PP 64)
    clock_t delta = clock ();
    // MÉRÜNK IDŐT (PP 66)
    struct tms tmsbuf1, tmsbuf2;
    times (&tmsbuf1);

    // számítás adatai
    float a = -2.0, b = .7, c = -1.35, d = 1.35;
    int szelesseg = MERET, magassag = MERET, iteraciosHatar = ITER_HAT;

    // a számítás
    float dx = (b - a) / szelesseg;
    float dy = (d - c) / magassag;
    float reC, imC, reZ, imZ, ujreZ, ujimZ;
    // Hány iterációt csináltunk?
    int iteracio = 0;
    // Végigzongorázzuk a szélesség x magasság rácsot:
    for (int j = 0; j < magassag; ++j)
    {
        //sor = j;
        for (int k = 0; k < szelesseg; ++k)
        {
            // c = (reC, imC) a rács csomópontjainak
            // megfelelő komplex szám
            reC = a + k * dx;
            imC = d - j * dy;
            // z_0 = 0 = (reZ, imZ)
            reZ = 0;
            imZ = 0;
            iteracio = 0;
            // z_{n+1} = z_n * z_n + c iterációk
            // számítása, amíg |z_n| < 2 vagy még
            // nem értük el a 255 iterációt, ha
            // viszont elértük, akkor úgy vesszük,
            // hogy a kiindulási c komplex számra
            // az iteráció konvergens, azaz a c a
            // Mandelbrot halmaz eleme
            while (reZ * reZ + imZ * imZ < 4 && iteracio < iteraciosHatar)
            {
                // z_{n+1} = z_n * z_n + c
```

```
        ujureZ = reZ * reZ - imZ * imZ + reC;
        ujimZ = 2 * reZ * imZ + imC;
        reZ = ujureZ;
        imZ = ujimZ;

        ++iteracio;

    }

    kepadat[j][k] = iteracio;
}

times (&tmsbuf2);
std::cout << tmsbuf2.tms_utime - tmsbuf1.tms_utime
            + tmsbuf2.tms_stime - tmsbuf1.tms_stime << std::endl;

delta = clock () - delta;
std::cout << (float) delta / CLOCKS_PER_SEC << " sec" << std::endl;
}

int
main (int argc, char *argv[])
{

    if (argc != 2)
    {
        std::cout << "Hasznalat: ./mandelpng fajlnev";
        return -1;
    }

    int kepadat[MERET][MERET];

    mandel(kepadat);

    png::image < png::rgb_pixel > kep (MERET, MERET);

    for (int j = 0; j < MERET; ++j)
    {
        //sor = j;
        for (int k = 0; k < MERET; ++k)
        {
            kep.set_pixel (k, j,
                           png::rgb_pixel (255 -
                                              (255 * kepadat[j][k]) / ITER_HAT ←
                                              ,
                                              255 -
                                              (255 * kepadat[j][k]) / ITER_HAT ←
                                              ,
```

```
                255 -  
                (255 * kepadat[j][k]) / ITER_HAT ←  
                ));  
        }  
    }  
  
    kep.write (argv[1]);  
    std::cout << argv[1] << " mentve" << std::endl;  
}
```

5.2. A Mandelbrot halmaz a `std::complex` osztállyal

Megoldás Forrása: https://gitlab.com/nbatfai/bhax/blob/master/attention_raising/Mandelbrot/3.1.2.cpp Megoldás videó: <https://youtu.be/gvaqijHIRUs>

```
// Verzio: 3.1.2.cpp  
// Forditas:  
// g++ 3.1.2.cpp -lpng -O3 -o 3.1.2  
// Futtatas:  
// ./3.1.2 mandel.png 1920 1080 2040 ←  
-0.01947381057309366392260585598705802112818 ←  
-0.0194738105725413418456426484226540196687 ←  
0.7985057569338268601555341774655971676111 ←  
0.798505756934379196110285192844457924366  
// ./3.1.2 mandel.png 1920 1080 1020 ←  
0.4127655418209589255340574709407519549131 ←  
0.4127655418245818053080142817634623497725 ←  
0.2135387051768746491386963270997512154281 ←  
0.2135387051804975289126531379224616102874  
// Nyomtatas:  
// a2ps 3.1.2.cpp -o 3.1.2.cpp.pdf -l --line-numbers=1 --left-footer=" ←  
BATF41 HAXOR STR34M" --right-footer="https://bhaxor.blog.hu/" --pro= ←  
color  
// ps2pdf 3.1.2.cpp.pdf 3.1.2.cpp.pdf.pdf  
//  
//  
// Copyright (C) 2019  
// Norbert Batfai, batfai.norbert@inf.unideb.hu  
//  
// This program is free software: you can redistribute it and/or modify  
// it under the terms of the GNU General Public License as published by  
// the Free Software Foundation, either version 3 of the License, or  
// (at your option) any later version.  
//  
// This program is distributed in the hope that it will be useful,
```

```
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
// GNU General Public License for more details.
//
// You should have received a copy of the GNU General Public License
// along with this program. If not, see <https://www.gnu.org/licenses/>.

#include <iostream>
#include "png++/png.hpp"
#include <complex>

int
main ( int argc, char *argv[] )
{

    int szelesseg = 1920;
    int magassag = 1080;
    int iteraciosHatar = 255;
    double a = -1.9;
    double b = 0.7;
    double c = -1.3;
    double d = 1.3;

    if ( argc == 9 )
    {
        szelesseg = atoi ( argv[2] );
        magassag = atoi ( argv[3] );
        iteraciosHatar = atoi ( argv[4] );
        a = atof ( argv[5] );
        b = atof ( argv[6] );
        c = atof ( argv[7] );
        d = atof ( argv[8] );
    }
    else
    {
        std::cout << "Hasznalat: ./3.1.2 fajlnev szelesseg magassag n a b c d ↵"
                  << std::endl;
        return -1;
    }

    png::image < png::rgb_pixel > kep ( szelesseg, magassag );

    double dx = ( b - a ) / szelesseg;
    double dy = ( d - c ) / magassag;
    double reC, imC, reZ, imZ;
    int iteracio = 0;

    std::cout << "Szamitas\n";
```

```
// j megy a sorokon
for ( int j = 0; j < magassag; ++j )
{
    // k megy az oszlopokon

    for ( int k = 0; k < szelesseg; ++k )
    {

        // c = (reC, imC) a halo racspontjainak
        // megfelelo komplex szam

        reC = a + k * dx;
        imC = d - j * dy;
        std::complex<double> c ( reC, imC );

        std::complex<double> z_n ( 0, 0 );
        iteracio = 0;

        while ( std::abs ( z_n ) < 4 && iteracio < iteraciosHatar )
        {
            z_n = z_n * z_n + c;

            ++iteracio;
        }

        kep.set_pixel ( k, j,
                        png::rgb_pixel ( iteracio%255, (iteracio*iteracio <=
                        )%255, 0 ) );
    }

    int szazalek = ( double ) j / ( double ) magassag * 100.0;
    std::cout << "\r" << szazalek << "%" << std::flush;
}

kep.write ( argv[1] );
std::cout << "\r" << argv[1] << " mentve." << std::endl;
}
```

Az `std::complex` osztály miatt a komplex számokat egy változóban is tudjuk tárolni.

5.3. Biomorfok

Megoldás videó: <https://youtu.be/IJMbgRzY76E>

Megoldás forrása: https://gitlab.com/nbatfai/bhax/tree/master/attention_raising/Biomorf

Clifford Pickover talált rá a biomorfokra egy Julia halmazokat rajzoló bug-os programmal. A Mandelbrot

és a Julia halmaz között annyi a különbség hogy a Julia halmazban a c egy konstans míg a Mandelbrot halmazban egy változó, szóval most az előző program egy módosított változatát használjuk.

```
// Verzio: 3.1.3.cpp
// Forditas:
// g++ 3.1.3.cpp -lpng -O3 -o 3.1.3
// Futtatas:
// ./3.1.3 bmorf.png 800 800 10 -2 2 -2 2 .285 0 10
// Nyomtatas:
// a2ps 3.1.3.cpp -o 3.1.3.cpp.pdf -1 --line-numbers=1 --left-footer=" ←
// BATF41 HAXOR STR34M" --right-footer="https://bhaxor.blog.hu/" --pro= ←
// color
// ps2pdf 3.1.3.cpp.pdf 3.1.3.cpp.pdf.pdf
//
// BHAX Biomorphs
// Copyright (C) 2019
// Norbert Batfai, batfai.norbert@inf.unideb.hu
//
// This program is free software: you can redistribute it and/or modify
// it under the terms of the GNU General Public License as published by
// the Free Software Foundation, either version 3 of the License, or
// (at your option) any later version.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
// GNU General Public License for more details.
//
// You should have received a copy of the GNU General Public License
// along with this program. If not, see <https://www.gnu.org/licenses/>.
//
// Version history
//
// https://youtu.be/IJMbgRzY76E
// See also https://www.emis.de/journals/TJNSA/includes/files/articles/ ←
// Vol9_Iss5_2305--2315_Biomorphs_via_modified_iterations.pdf
//

#include <iostream>
#include "png++/png.hpp"
#include <complex>

int
main ( int argc, char *argv[] )
{
    int szelesseg = 1920;
    int magassag = 1080;
    int iteraciosHatar = 255;
```

```
double xmin = -1.9;
double xmax = 0.7;
double ymin = -1.3;
double ymax = 1.3;
double reC = .285, imC = 0;
double R = 10.0;

if ( argc == 12 )
{
    szelesseg = atoi ( argv[2] );
    magassag =  atoi ( argv[3] );
    iteraciosHatar =  atoi ( argv[4] );
    xmin = atof ( argv[5] );
    xmax = atof ( argv[6] );
    ymin = atof ( argv[7] );
    ymax = atof ( argv[8] );
    reC = atof ( argv[9] );
    imC = atof ( argv[10] );
    R = atof ( argv[11] );
}
else
{
    std::cout << "Hasznalat: ./3.1.2 fajlnev szelesseg magassag n a b c ↔  
d reC imC R" << std::endl;
    return -1;
}

png::image < png::rgb_pixel > kep ( szelesseg, magassag );

double dx = ( xmax - xmin ) / szelesseg;
double dy = ( ymax - ymin ) / magassag;

std::complex<double> cc ( reC, imC );

std::cout << "Szamitas\n";

// j megy a sorokon
for ( int y = 0; y < magassag; ++y )
{
    // k megy az oszlopokon

    for ( int x = 0; x < szelesseg; ++x )
    {

        double reZ = xmin + x * dx;
        double imZ = ymax - y * dy;
        std::complex<double> z_n ( reZ, imZ );

        int iteracio = 0;
```



```
for (int i=0; i < iteraciosHatar; ++i)
{
    z_n = std::pow(z_n, 3) + cc;
    //z_n = std::pow(z_n, 2) + std::sin(z_n) + cc;
    if(std::real ( z_n ) > R || std::imag ( z_n ) > R)
    {
        iteracio = i;
        break;
    }
}

kep.set_pixel ( x, y,
                png::rgb_pixel ( (iteracio*20)%255, (iteracio ←
                                *40)%255, (iteracio*60)%255 ));
}

int szazalek = ( double ) y / ( double ) magassag * 100.0;
std::cout << "\r" << szazalek << "%" << std::flush;
}

kep.write ( argv[1] );
std::cout << "\r" << argv[1] << " mentve." << std::endl;
}
```

5.4. A Mandelbrot halmaz CUDA megvalósítása

Megoldás videó: <https://youtu.be/gvaqijHIRUs> Megoldás forrása: https://gitlab.com/nbatfai/bhax/blob/-/master/attention_raising/CUDA/mandelpngc_60x60_100.cu

A CUDA egy NVIDIA által kifejlesztett SDK, melyet a General Purpose GPU-k kihasználására hoztak létre. GPU-t használjuk az összetett grafikai műveletek számítására, miközben CPU más műveleteket végez.

```
// mandelpngc_60x60_100.cu
// Copyright (C) 2019
// Norbert Bátfai, batfai.norbert@inf.unideb.hu
//
// This program is free software: you can redistribute it and/or modify
// it under the terms of the GNU General Public License as published by
// the Free Software Foundation, either version 3 of the License, or
// (at your option) any later version.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
```

```
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
// GNU General Public License for more details.
//
// You should have received a copy of the GNU General Public License
// along with this program. If not, see <https://www.gnu.org/licenses/>.
//
// Version history
//
// Mandelbrot png
// Programozó Páternosztter/PARP
// https://www.tankonyvtar.hu/hu/tartalom/tamop412A/2011-0063 ←
// _01_parhuzamos_prog_linux
//
// https://youtu.be/gvaqijHlRUs
//

#include <png++/image.hpp>
#include <png++/rgb_pixel.hpp>

#include <sys/times.h>
#include <iostream>

#define MERET 600
#define ITER_HAT 32000

__device__ int
mandel (int k, int j)
{
    // Végigzongorázza a CUDA a szélesség x magasság rácsot:
    // most éppen a j. sor k. oszlopában vagyunk

    // számítás adatai
    float a = -2.0, b = .7, c = -1.35, d = 1.35;
    int szelesseg = MERET, magassag = MERET, iteraciosHatar = ITER_HAT;

    // a számítás
    float dx = (b - a) / szelesseg;
    float dy = (d - c) / magassag;
    float reC, imC, reZ, imZ, ujreZ, ujimZ;
    // Hány iterációt csináltunk?
    int iteracio = 0;

    // c = (reC, imC) a rács csomópontjainak
    // megfelelő komplex szám
    reC = a + k * dx;
    imC = d - j * dy;
    // z_0 = 0 = (reZ, imZ)
    reZ = 0.0;
    imZ = 0.0;
```

```
iteracio = 0;
// z_{n+1} = z_n * z_n + c iterációk
// számítása, amíg |z_n| < 2 vagy még
// nem értük el a 255 iterációt, ha
// viszont elértük, akkor úgy vesszük,
// hogy a kiindulási c komplex számra
// az iteráció konvergens, azaz a c a
// Mandelbrot halmaz eleme
while (reZ * reZ + imZ * imZ < 4 && iteracio < iteraciosHatar)
{
    // z_{n+1} = z_n * z_n + c
    ujreZ = reZ * reZ - imZ * imZ + reC;
    ujimZ = 2 * reZ * imZ + imC;
    reZ = ujreZ;
    imZ = ujimZ;

    ++iteracio;
}
return iteracio;
}

/*
__global__ void
mandelkernel (int *kepadat)
{
    int j = blockIdx.x;
    int k = blockIdx.y;

    kepadat[j + k * MERET] = mandel (j, k);
}
*/

__global__ void
mandelkernel (int *kepadat)
{
    int tj = threadIdx.x;
    int tk = threadIdx.y;

    int j = blockIdx.x * 10 + tj;
    int k = blockIdx.y * 10 + tk;

    kepadat[j + k * MERET] = mandel (j, k);
}
```

[illegible]

```
        255 -
        (255 * kepadat[j][k]) / ITER_HAT,
        255 -
        (255 * kepadat[j][k]) / ITER_HAT));
    }
}
kep.write (argv[1]);

std::cout << argv[1] << " mentve" << std::endl;

times (&tmsbuf2);
std::cout << tmsbuf2.tms_utime - tmsbuf1.tms_utime
    + tmsbuf2.tms_stime - tmsbuf1.tms_stime << std::endl;

delta = clock () - delta;
std::cout << (float) delta / CLOCKS_PER_SEC << " sec" << std::endl;
}
```

5.5. Mandelbrot nagyító és utazó C++ nyelven

Építs GUI-t a Mandelbrot algoritmusra, lehessen egérrel nagyítani egy területet, illetve egy pontot egérrel kiválasztva vizualizálja onnan a komplex iteráció bejárta z_n komplex számokat!

Megoldás videó: https://bhaxor.blog.hu/2018/09/02/ismerkedes_a_mandelbrot_halmazzal

```
#include "frakablak.h"
FrakAblak::FrakAblak(double a, double b, double c, double d,
                    int szelesseg, int iteraciosHatar, QWidget *parent)
    : QMainWindow(parent)
{
    setWindowTitle("Mandelbrot halmaz");
    szamitasFut = true;
    x = y = mx = my = 0;
    this->a = a;
    this->b = b;
    this->c = c;
    this->d = d;
    this->szelesseg = szelesseg;
    this->iteraciosHatar = iteraciosHatar;
    magassag = (int)(szelesseg * ((d-c)/(b-a)));
    setFixedSize(QSize(szelesseg, magassag));
    fraktal= new QImage(szelesseg, magassag, QImage::Format_RGB32);
    mandelbrot = new FrakSzal(a, b, c, d, szelesseg, magassag, ←
        iteraciosHatar, this);
    mandelbrot->start();
}
```

```
}
FrakAblak::~FrakAblak()
{
    delete fraktal;
    delete mandelbrot;
}
void FrakAblak::paintEvent(QPaintEvent*) {
    QPainter qpainter(this);
    qpainter.drawImage(0, 0, *fraktal);
    if(!szamitasFut) {
        qpainter.setPen(QPen(Qt::white, 1));
        qpainter.drawRect(x, y, mx, my);
    }
    qpainter.end();
}
void FrakAblak::mousePressEvent(QMouseEvent* event) {
    x = event->x();
    y = event->y();
    mx = 0;
    my = 0;
    update();
}

void FrakAblak::mouseMoveEvent(QMouseEvent* event) {
    mx = event->x() - x;
    my = mx; // négyzet alakú
    update();
}

void FrakAblak::mouseReleaseEvent(QMouseEvent* event) {
    if(szamitasFut)
        return;
    szamitasFut = true;
    double dx = (b-a)/szelesseg;
    double dy = (d-c)/magassag;
    double a = this->a+x*dx;
    double b = this->a+x*dx+mx*dx;
    double c = this->d-y*dy-my*dy;
    double d = this->d-y*dy;
    this->a = a;
    this->b = b;
    this->c = c;
    this->d = d;
    delete mandelbrot;
    mandelbrot = new FrakSzal(a, b, c, d, szelesseg, magassag, ↵
        iteraciosHatar, this);
    mandelbrot->start();
    update();
}
void FrakAblak::keyPressEvent(QKeyEvent *event)
```

```
{
    if(szamitasFut)
        return;
    if (event->key() == Qt::Key_N)
        iteraciosHatar *= 2;
    szamitasFut = true;
    delete mandelbrot;
    mandelbrot = new FrakSzal(a, b, c, d, szelesseg, magassag, ←
        iteraciosHatar, this);
    mandelbrot->start();
}

void FrakAblak::vissza(int magassag, int *sor, int meret)
{
    for(int i=0; i<meret; ++i) {
        QRgb szin = qRgb(0, 255-sor[i], 0);
        fraktal->setPixel(i, magassag, szin);
    }
    update();
}

void FrakAblak::vissza(void)
{
    szamitasFut = false;
    x = y = mx = my = 0;
}
}
```

```
#include "frakszal.h"
```

```
FrakSzal::FrakSzal(double a, double b, double c, double d,
                  int szelesseg, int magassag, int iteraciosHatar, ←
                  FrakAblak *frakAblak)
{
    this->a = a;
    this->b = b;
    this->c = c;
    this->d = d;
    this->szelesseg = szelesseg;
    this->iteraciosHatar = iteraciosHatar;
    this->frakAblak = frakAblak;
    this->magassag = magassag;

    egySor = new int[szelesseg];
}

FrakSzal::~FrakSzal()
{
    delete[] egySor;
}

void FrakSzal::run()
{
    double dx = (b-a)/szelesseg;
```

```
double dy = (d-c)/magassag;
double reC, imC, reZ, imZ, ujureZ, ujimZ;
int iteracio = 0;
for(int j=0; j<magassag; ++j) {
    //sor = j;
    for(int k=0; k<szelesseg; ++k) {
        reC = a+k*dx;
        imC = d-j*dy;
        reZ = 0;
        imZ = 0;
        iteracio = 0;
        while(reZ*reZ + imZ*imZ < 4 && iteracio < iteraciosHatar) {
            ujureZ = reZ*reZ - imZ*imZ + reC;
            ujimZ = 2*reZ*imZ + imC;
            reZ = ujureZ;
            imZ = ujimZ;
            ++iteracio;
        }
        iteracio %= 256;
        egySor[k] = iteracio;
    }
    frakAblak->vissza(j, egySor, szelesseg);
}
frakAblak->vissza();
}
```

```
// main.cpp
#include <QApplication>
#include "frakablak.h"

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    FrakAblak w1;
    w1.show();
    /*
    FrakAblak w1,
    w2(-.08292191725019529, -.082921917244591272,
        -.9662079988595939, -.9662079988551173, 600, 3000),
    w3(-.08292191724880625, -.0829219172470933,
        -.9662079988581493, -.9662079988563615, 600, 4000),
    w4(.14388310361318304, .14388310362702217,
        .6523089200729396, .6523089200854384, 600, 38655);
    w1.show();
    w2.show();
    w3.show();
    w4.show();
    */
    return a.exec();
}
```



```
}  
}
```

Ebben a cikkben <https://www.tankonyvtar.hu/hu/tartalom/tkt/javat-tanitok-javat/apbs02.html#id570518> részletesen le van írva minden lépés magyarázattal csak a program Java nyelvben íródott.

5.6. Mandelbrot nagyító és utazó Java nyelven

Megoldás videó: <https://youtu.be/Ui3B6IJnssY> Megoldás forrása: <https://www.tankonyvtar.hu/hu/tartalom/tkt/javat-tanitok-javat/apbs02.html#id570518>

6. fejezet

Helló, Welch!

6.1. Első osztályom

Valósítsd meg C++-ban és Java-ban az módosított polártranszformációs algoritmust! A matek háttér teljesen irreleváns, csak annyiban érdekes, hogy az algoritmus egy számítása során két normálist számol ki, az egyiket elspájzold és egy további logikai taggal az osztályban jelzed, hogy van vagy nincs eltérve kiszámolt szám.

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat... térj ki arra is, hogy a JDK forrásaiban a Sun programozói pont úgy csinálták meg ahogyan te is, azaz az OO nemhogy nem nehéz, hanem éppen természetes neked!

6.2. LZW

Valósítsd meg C-ben az LZW algoritmus fa-építését!

Megoldás videó:

Megoldás forrása:

6.3. Fabejárás

Járd be az előző (inorder bejárású) fát pre- és posztorder is!

Megoldás videó:

Megoldás forrása:

6.4. Tag a gyökér

Az LZW algoritmust ültesd át egy C++ osztályba, legyen egy Tree és egy beágyazott Node osztálya. A gyökér csomópont legyen kompozícióban a fával!

Megoldás videó:

Megoldás forrása:

6.5. Mutató a gyökér

Írd át az előző forrást, hogy a gyökér csomópont ne kompozícióban, csak aggregációban legyen a fával!

Megoldás videó:

Megoldás forrása:

6.6. Mozgató szemantika

Írj az előző programhoz mozgató konstruktort és értékadást, a mozgató konstruktor legyen a mozgató értékadásra alapozva!

Megoldás videó:

Megoldás forrása:

7. fejezet

Helló, Conway!

7.1. Hangyaszimulációk

Írj Qt C++-ban egy hangyaszimulációs programot, a forrásaidról utólag reverse engineering jelleggel készíts UML osztálydiagramot is!

Megoldás videó: <https://bhaxor.blog.hu/2018/10/10/myrmecologist>

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

7.2. Java életjáték

Írd meg Java-ban a John Horton Conway-féle életjátékot, valósítsa meg a sikló-kilövőt!

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

7.3. Qt C++ életjáték

Most Qt C++-ban!

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

7.4. BrainB Benchmark

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

DRAFT

8. fejezet

Helló, Schwarzenegger!

8.1. Szoftmax Py MNIST

aa Python

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

8.2. Szoftmax R MNIST

R

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

8.3. Mély MNIST

Python

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

8.4. Deep dream

Keras

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

8.5. Robotpszichológia

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

DRAFT

9. fejezet

Helló, Chaitin!

9.1. Iteratív és rekurzív faktoriális Lisp-ben

Megoldás videó:

Megoldás forrása:

9.2. Weizenbaum Eliza programja

Éleszd fel Weizenbaum Eliza programját!

Megoldás videó:

Megoldás forrása:

9.3. Gimp Scheme Script-fu: króm effekt

Írj olyan script-fu kiterjesztést a GIMP programhoz, amely megvalósítja a króm effektet egy bemenő szövegre!

Megoldás videó: https://youtu.be/OKdAkI_c7Sc

Megoldás forrása: https://gitlab.com/nbatfai/bhax/tree/master/attention_raising/GIMP_Lisp/Chrome

Tanulságok, tapasztalatok, magyarázat...

9.4. Gimp Scheme Script-fu: név mandala

Írj olyan script-fu kiterjesztést a GIMP programhoz, amely név-mandalát készít a bemenő szövegből!

Megoldás videó: https://bhaxor.blog.hu/2019/01/10/a_gimp_lisp_hackelese_a_scheme_programozasi_nyelv

Megoldás forrása: https://gitlab.com/nbatfai/bhax/tree/master/attention_raising/GIMP_Lisp/Mandala

Tanulságok, tapasztalatok, magyarázat...

9.5. Lambda

Hasonlítsd össze a következő programokat!

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

9.6. Omega

Megoldás videó:

Megoldás forrása:

DRAFT

III. rész

Második felvonás

DRAFT

**Bátf41 Haxor Stream**

A feladatokkal kapcsolatos élő adásokat sugároz a <https://www.twitch.tv/nbatfai> csatorna, melynek permanens archívuma a <https://www.youtube.com/c/nbatfai> csatornán található.

DRAFT

10. fejezet

Helló, Arroway!

10.1. A BPP algoritmus Java megvalósítása

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

10.2. Java osztályok a Pi-ben

Az előző feladat kódját fejleszd tovább: vizsgáld, hogy Vannak-e Java osztályok a Pi hexadecimális kifejtésében!

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

11. fejezet

Helló, Gutenberg!

11.1. Juhász István: Magas szintű programozási nyelvek 1

Alapfogalmak: A magas szintű nyelv a számítógépek programozására kialakult nyelvek egyik szintje. Forrásszövegnek nevezzük a magas szintű programozási nyelven írt programot. Fő fajtái a programkészítésnek (a forrásszövegből) a fordítóprogramos (pl C) és az interpreteres módszer (pl JavaScript). A fordítóprogram a következő lépéseket hajtja végre: lexikális elemzés (darabolás egységekre), szintaktikai elemzés (teljesülnek-e a nyelv szabályai), szemantikai elemzés és kódgenerálás. A fordítóprogram mint "közvetítő réteg" rengeteg tesztet, elemzést hajt végre a programon a kódgeneráláson kívül (pl lexikális, szintaktikai, szemantikai elemzés). Míg a fordítóprogramos technikánál készül tárgyprogram, addig az interpreteres technikánál nem. A programozási nyelvek saját szintaktikai és szemantikai szabályait hivatkozási nyelvnek nevezzük. A programnyelvek osztályozása: vannak imperatív, deklaratív és egyéb nyelvek. Imperatív nyelvek: a program utasítások sorozata és a legfőbb programozói eszköz a változó. Az algoritmus működteti a processzort. Deklaratív nyelvek: nem algoritmikus nyelvek, a programozónak nincs lehetősége memóriaműveletekre (vagy csak korlátozott módon).

Adattípusok: Az adattípus konkrét programozási eszközök komponenseként jelenik meg. Legfontosabb tényezői a tartomány, a műveletek és reprezentáció. Tartomány: Olyan elemeket tartalmaz, amiket felvehet értékként egy programozási eszköz. (Műveletek: elemeken végrehajtható műveletek) Reprezentáció: belső ábrázolási mód. (gyakran előre meghatározott) Saját reprezentáció igen ritka (pl. Ada). Az adattípusok csoportjai: Egyszerű: egész, valós, karakteres és a logikai típusok. Összetett: Tömb (több fajta dimenziós változatok). Az egyszerű típusok közül az egész, valós, karakteres és a logikai típus már a programozás-tanulás korai fázisában előjön. Ugyanez elmondható a tömbről is, mely ugyan az összetett típusokhoz tartozik, de egy- és kétdimenziós változatait gyakran használjuk, szinte a kezdetektől fogva.

A nevesített konstans: Három komponense van: név, típus és érték. A nevesített konstans amit mi használunk általában a `#define`, amely minden előfordulást kicserél.

Változó: Az imperatív programozási nyelvek fő eszköze, négy komponense van: név, attribútumok, cím, érték. A változó alapja a név a többi "attribútumot" pedig hozzárendeljük. Az attribútumok a változó futás közbeni viselkedését határozzák meg, a cím pedig a változó elhelyezkedését. Ha két különböző névvel rendelkező változónak a futási idő egy adott pillanatában azonos a címkomponense és az értékkomponense is, akkor beszélhetünk többszörös tárhivatkozásról. Az eljárásorientált nyelvek leggyakoribb utasítása az értékadó utasítás.

Kifejezések: Szintaktikai eszközök, két komponensük az érték és a típus. Összetevői az operandusok, az

operátorok és a kerek zárójelek. Létezik egyoperandusú (unáris), kétoperandusú (bináris) és háromoperandusú (ternáris) operátor is. A kifejezések három alakja a Pre, in és Postfix mely értelemszerűen a operátor helyét mutatják az operandusokhoz képest. A kifejezés kiértékelése: a kifejezés értékének és típusának meghatározódása. A kiértékelés szempontjából a logikai operátorokkal rendelkező kifejezések speciálisak, ugyanis előfordulhat, hogy nem végezzük el az összes műveletet. (Logikus pl VAGY-nál igaz érték találása esetén többszámolni). Két programozási eszköz típusa azonos deklaráció-, név- és struktúra egyenértékűség esetén. A C egy kifejezésorientált nyelv. A mutató típus tartományának elemeivel összeadás és kivonás végezhető. A tömb típusú eszköz neve mutató típusú.

```
//Operátorok:  
() - függvényoperátor;  
[] - tömboperátor;  
. - minősítő operátor;  
-> - mutatóval történő minősítés operátora;  
* - indirekciós operátor;  
! - egyoperandusú operátor;  
~ - egyes komplement operátora;  
* - szorzás operátora;  
/ - osztás operátora;  
% - maradékképzés operátora;  
+ - összeadás operátora;  
- - kivonás operátora;  
<< >> - léptető operátorok;  
< > <= >= != - hasonlító operátorok;  
& ^ | - nem rövidzár logikai operátorok;  
&& || - rövidzár logikai operátorok;  
? : - háromoperandusú operátor.
```

Utasítások: Olyan egységek, amelyekkel megadjuk az algoritmusok lépéseit és amely segítségével a fordítóprogram tárgyprogramot generálhat. Típusai: Deklarációs utasítások: Nincs tárgykód a utasítás mögött. A tárgykód a végrehajtható utasításokból generálódik, a fordító segítségével és a végrehajtható utasítások. A deklarációs utasítások mögött nem áll tárgykód. A tárgykód a végrehajtható utasításokból generálódik, fordítóprogram segítségével. Az értékadó utasítás feladata beállítani vagy módosítani a változók értékkomponensét. Az üres utasítások -> üres gépi utasítást hajt végre. Az ugró utasítás (GOTO címke)-> a program egy adott pontjáról egy adott címkével ellátott végrehajtható utasításra adja át a vezérlést. A kétirányú elágaztató utasítás (IF feltétel THEN tevékenység [ELSE tevékenység]) arra szolgál, hogy a program egy adott pontján két tevékenység közül válasszunk, illetve egy adott tevékenységet végrehajtsunk vagy sem. ELSE-ág hiánya esetén rövid kétirányú elágaztató utasításról beszélünk. A többirányú elágaztató utasítás arra szolgál, hogy a program egy adott pontján egymást kölcsönösen kizáró tevékenységek közül egyet végrehajtsunk.

```
C-ben:  
SWITCH (kifejezés) {  
CASE egész_konstans_kifejezés: [tevékenység]  
[CASE egész_konstans_kifejezés: [tevékenység]]...  
[DEFAULT: tevékenység]  
};
```

A ciklusszervező utasítások lehetővé teszik, hogy a program egy adott pontján egy bizonyos tevékenységet akárhányszor megismételjünk. Felépítése: fej, mag, vég. A feltételes ciklusnál az ismétlődést az igaz vagy

hamis érték határozza meg. A feltétel a fejben(While) vagy a végben(For) szerepel, a mag az ismételendő végrehajtható utasításokat tartalmazza. Az előírt lépésszámú ciklusnál a ciklusparaméterek a fejben vannak. A változó által felvett értékekre fut le a ciklusmag. A ciklusváltozó a tartománynak vagy minden elemét fölveheti, vagy csak a tartományban szabályosan elhelyezkedő bizonyos értékeket. A felsorolós ciklusnak van ciklusváltozója, és minden felvett érték mellett lefut a mag. A végtelen ciklusnál sem a fejben, sem a végben nincs információ az ismétlődésre vonatkozóan. Az összetett ciklus pedig az előző ciklusfajták kombinációja.

C-ben:

```
//Kezdőfeltételes ciklus:
WHILE(feltétel) végrehajtható_utasítás
//Végfeltételes ciklus:
DO végrehajtható_utasítás WHILE(feltétel);
//FOR-ciklus:
FOR([kifejezés1]; [kifejezés2]; [kifejezés3]) végrehajtható_utasítás
//Vezérlő utasítások:
CONTINUE;
BREAK;
RETURN[kifejezés];
```

11.2. Kernigan-Ritchie: A C programozási nyelv

Vezérlési szerkezetek: Egy nyelv vezérlésátadó utasításai az egyes műveletek végrehajtási sorrendjét határozzák meg. Egy olyan kifejezés, mint $x = 0$, $i++$ vagy `printf(...)` utasítássá válik, ha egy pontosvesszőt írunk utána. A C nyelvben ugyanis a pontosvessző az utasításlezáró jel (terminátor). A `{ }` kapcsos zárójelekkel deklarációk és utasítások csoportját tudjuk összefogni egyetlen összetett utasításba vagy blokkba. Pl. `if`, `else`, `while`, `for` utasítások utáni utasításokat összefogó zárójelpár. Az `if-else` utasítást döntés kifejezésére használjuk. Az utasítás először kiértékeli a kifejezést, és ha ennek értéke igaz, akkor az 1. utasítást hajtja végre. Ha hamis és van `else` rész, akkor a 2. utasítás hajtódik végre.

```
if (kifejezés)
    1. utasítás
else
    2. utasítás
//az else rész opcionális
```

Az `else-if` utasítás adja a többszörös döntések programozásának egyik legáltalánosabb lehetőségét. A gép sorra kiértékeli a kifejezéseket és ha bármelyik ezek közül igaz, akkor végrehajtja a megfelelő utasítást, majd befejezi a láncot.

```
if (kifejezés)
    utasítás
else if (kifejezés)
    utasítás
.
.
.
else
```

utasítás

A switch utasítás úgy működik, hogy összehasonlítja egy kifejezés értékét több egész értékű állandó kifejezés értékével, és az ennek megfelelő utasítást hajtja végre. A case ágakban egy egész állandó vagy állandó egész értékű kifejezés található, és ha ennek értéke megegyezik a switch utáni kifejezés értékével, akkor végrehajtnak az utasítások. A default ág opcionális, akkor hajtódik végre, ha egyetlen case ághoz tartozó feltétel sem teljesült.

```
switch (kifejezés) {  
    case állandó kifejezés: utasítások  
    case állandó kifejezés: utasítások  
    .  
    .  
    .  
    default: utasítások  
}
```

A while ciklus esetén a program először kiértékeli a kifejezést. A ciklus addig folytatódik, amíg a kifejezés nullává (false) nem válik.

```
while (kifejezés)  
    utasítás
```

A for utasítás mindhárom komponense kifejezés. Egyszerű inicializálás és újrainicializálás esetén előnyösebb a while ciklusnál, mivel a ciklust vezérlő utasítások jól látható formában helyezkednek el.

```
for (1. kifejezés, 2. kifejezés, 3. kifejezés)  
    utasítás
```

A harmadik ciklusszervező utasítás a do-while. A gép először végrehajtja az utasítást és csak utána értékeli a kifejezést. Addig tart a ciklus, amíg a kifejezés értéke hamis nem lesz.

```
do  
    utasítás  
while (kifejezés);
```

A break utasítás lehetővé teszi a for, while vagy do utasításokkal szervezett ciklusok idő előtti elhagyását, valamint a switch utasításból való kilépést. A ciklusmagban található continue utasítás hatására azonnal megkezdődik a következő iterációs lépés. A goto utasítással megadott címkékre ugorhatunk. Használnánk egyik legelterjedtebb esete, amikor több szinten egymásba ágyazott szerkezet belsejében kívánjuk abbahagyni a feldolgozást és egyszerre több, egymásba ágyazott ciklusból szeretnénk kilépni.

Utasítások: Végrehajtásuk a hatásukban nyilvánul meg és nem rendelkeznek értékkel.

```
utasítás:  
    címkézett_utasítás  
    kifejezésutasítás  
    összetett_utasítás  
    kiválasztó_utasítás  
    iterációs_utasítás  
    vezérlésátadó_utasítás
```


Az utasításokhoz előtagként megadott címke tartozhat.

```
címkézett_utasítás  
  azonosító: utasítás  
  case állandó_kifejezés: utasítás  
  default: utasítás
```

A címkék önmagukban nem módosítják az utasítások végrehajtásának sorrendjét. Az utasítások többsége kifejezésutasítás. Funkcióját tekintve a legtöbb kifejezésutasítás értékadás vagy függvényhívás.

```
kifejezésutasítás:  
  kifejezés
```

A C nyelvben lehetőség van összetett utasításokra is, melyek több utasítás egyetlen utasításkénti kezelését teszi lehetővé.

```
összetett_utasítás:  
  {deklarációs_lista utasítás_lista}  
deklarációs_lista:  
  deklaráció  
  deklarációs_lista deklaráció  
utasítás_lista:  
  utasítás  
  utasítás_lista utasítás
```

A kiválasztó utasítások minden esetben a lehetséges végrehajtási sorrendek egyikét választják ki.

```
kiválasztó_utasítás:  
  if (kifejezés) utasítás  
  if (kifejezés) utasítás else utasítás  
  switch (kifejezés) utasítás
```

Az iterációs utasítások egy ciklust határoznak meg.

```
iterációs_utasítás:  
  while (kifejezés) utasítás  
  do utasítás while (kifejezés):  
  for (kifejezés; kifejezés; kifejezés) utasítás
```

A vezérlésátadó utasítások a vezérlés feltétel nélküli átadására alkalmasak.

```
vezérlésátadó_utasítás:  
  goto azonosító;  
  continue;  
  break;  
  return kifejezés;
```

11.3. Benedek-Levendovszky: Szoftverfejlesztés C++ nyelven

A C és a C++ nyelv: A C++ nyelvben az üres paraméterlista egy void paraméter megadásával ekvivalens: a függvénynek nincs paramétere.

```
void f(void)
{
    // Az f függvény törzse...
}
```

A szabványos C++ nyelvben a main függvénynek két formája létezik:

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

és

```
int main(int argc, char* argv[])
{
    ...
}
//argc: parancssor-argumentumok száma
//argv: parancssor-argumentumok
```

A C++ nyelvben került bevezetésre a bool típus, ami logikai igaz/hamis értéket tud reprezentálni. Pl. bool success = false. A bool típus olvashatóbb kódot eredményez, lehetőség van a függvénynevek és operátorok bool és int típusokra vonatkozó túlterhelésére. A C++ nyelvben a wchar_t beépített típus lett, így használatához a típusdefinícióra nincs szükség.

```
wchar_t c = L's';
wchar_t* text = L"sss";
```

A függvényeket a nevük és az argumentumlistájuk együttesen azonosítja, tehát a C++ nyelvben van lehetőség azonos nevű függvények létrehozására. A függvények argumentumainak alapértelmezett értékét is meg tudjuk már adni. Amennyiben a függvény hívásakor nem adunk meg értéket, a függvény az adott argumentum alapértelmezett értékével kerül meghívásra. A C++ bevezette a referenciatípust, ami megszünteti a pointerok szerepét a cím szerinti paraméterátadásban.

```
#include <stdio.h>
void f (int& i)
{
    i = i + 2;
}
int main(void)
{
    int a = 0;
    f(a);
    printf("%d\n", a);
}
//Ebben a C++ programban látható, hogy nem kell a változó címét képezni és ←
    az i szimbólumot ugyanolyan szintaxissal használhatjuk, mint egy int ←
    típusú változót.
//A program kimenete 2 és egy soremelés lesz.
```

```
//Egy adott típusú referenciát a referencia neve elé írt & jellel ↔  
    deklarálunk.  
//Az & jel egyargumentumú operátor is egyben, amely a mögötte álló változó ↔  
    címét adja vissza.
```

DRAFT

IV. rész

Irodalomjegyzék

DRAFT

11.4. Általános

[MARX] Marx, György, *Gyorsuló idő*, Typotex , 2005.

11.5. C

[KERNIGHANRITCHIE] Kernighan, Brian W. & Ritchie, Dennis M., *A C programozási nyelv*, Bp., Műszaki, 1993.

11.6. C++

[BMECPP] Benedek, Zoltán & Levendovszky, Tihamér, *Szoftverfejlesztés C++ nyelven*, Bp., Szak Kiadó, 2013.

11.7. Lisp

[METAMATH] Chaitin, Gregory, *META MATH! The Quest for Omega*, http://arxiv.org/PS_cache/math/pdf/0404/0404335v7.pdf , 2004.

Köszönet illeti a NEMESPOR, <https://groups.google.com/forum/#!forum/nemespor>, az UDPROG tanulószoba, <https://www.facebook.com/groups/udprog>, a DEAC-Hackers előszoba, <https://www.facebook.com/groups/DEACHackers> (illetve egyéb alkalmi szerveződésű szakmai csoportok) tagjait inspiráló érdeklődésükért és hasznos észrevételeikért.

Ezen túl kiemelt köszönet illeti az említett UDPROG közösséget, mely a Debreceni Egyetem reguláris programozás oktatása tartalmi szervezését támogatja. Sok példa eleve ebben a közösségben született, vagy itt került említésre és adott esetekben szerepet kapott, mint oktatási példa.