

Univerzális programozás

Írd meg a saját programozás tankönyvedet!

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

Copyright © 2019 Dr. Bátfai Norbert, Heinrich László

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. november 10.	

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
0.1.0	2019-03-01	Első csokor kész.	heinrichlaszlo
0.2.0	2019-03-12	Második csokor kész.	heinrichlaszlo
0.3.0	2019-03-19	Harmadik csokor befejezve.	heinrichlaszlo
0.4.0	2019-03-19	Negyedik csokor kész.	heinrichlaszlo

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME
0.5.0	2019-03-26	Ötödik csokor kész.	heinrichlaszlo
0.6.0	2019-04-02	Hatodik csokor kész.	heinrichlaszlo
0.7.0	2019-04-09	Hetedik csokor kész.	heinrichlaszlo
0.7.1	2019-04-13	Az eddigi fejezetek javítása.	heinrichlaszlo
0.8.0	2019-04-23	Nyolcadik csokor kész.	heinrichlaszlo
0.9.0	2019-04-29	Kilencedik csokor kész.	heinrichlaszlo
0.9.3	2019-05-04	Az eddigi fejezetek javítása.	heinrichlaszlo
0.9.4	2019-05-05	Az eddigi fejezetek javítása.	heinrichlaszlo
0.9.5	2019-05-07	Az eddigi fejezetek javítása.	heinrichlaszlo
0.9.6	2019-05-08	Az eddigi fejezetek javítása.	heinrichlaszlo

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	8
2.4. Labdapattogás	10
2.5. Szóhossz és a Linus Torvalds féle BogomIPS	13
2.6. Helló, Google!	13
2.7. 100 éves a Brun tétel	15
2.8. A Monty Hall probléma	15
3. Helló, Chomsky!	18
3.1. Decimálisból unárisba átváltó Turing gép	18
3.2. Az $a^n b^n c^n$ nyelv nem környezetfüggetlen	18
3.3. Hivatkozási nyelv	19
3.4. Saját lexikális elemző	20
3.5. l33t.1	21
3.6. A források olvasása	23
3.7. Logikus	24
3.8. Deklaráció	25

4. Helló, Caesar!	27
4.1. int *** háromszögmátrix	27
4.2. C EXOR titkosító	29
4.3. Java EXOR titkosító	31
4.4. C EXOR törő	32
4.5. Neurális OR, AND és EXOR kapu	35
4.6. Hiba-visszaterjesztéses perceptron	35
5. Helló, Mandelbrot!	37
5.1. A Mandelbrot halmaz	37
5.2. A Mandelbrot halmaz a <code>std::complex</code> osztállyal	39
5.3. Biomorfok	42
5.4. A Mandelbrot halmaz CUDA megvalósítása	44
5.5. Mandelbrot nagyító és utazó C++ nyelven	49
5.6. Mandelbrot nagyító és utazó Java nyelven	55
6. Helló, Welch!	59
6.1. Első osztályom	59
6.2. LZW	62
6.3. Fabejárás	65
6.4. Tag a gyökér	68
6.5. Mutató a gyökér	71
6.6. Mozgató szemantika	73
7. Helló, Conway!	76
7.1. Hangyaszimulációk	76
7.2. Java életjáték	93
7.3. Qt C++ életjáték	101
7.4. BrainB Benchmark	107
8. Helló, Schwarzenegger!	109
8.1. Szoftmax Py MNIST	109
8.2. Mély MNIST	111
8.3. Minecraft-MALMÖ	112

9. Helló, Chaitin!	113
9.1. Iteratív és rekurzív faktoriális Lisp-ben	113
9.2. Gimp Scheme Script-fu: króm effekt	115
9.3. Gimp Scheme Script-fu: név mandala	120
10. Helló, Gutenberg!	125
10.1. Juhász István: Magas szintű programozási nyelvek - olvasónapló	125
10.2. KR: A C programozási nyelv - olvasónapló	129
10.3. Benedek Zoltán, Levendovszky Tihamér: Szoftverfejlesztés C++ nyelven - olvasónapló	132
III. Második felvonás	137
11. Helló, Arroway!	139
11.1. OO szemlélet	139
11.2. Homokozó	140
11.3. "Gagyi"	141
11.4. Yoda	141
11.5. Kódolás from scratch	142
12. Helló, Berners-Lee!	146
12.1. Forstner Bertalan, Ekler Péter, Kelényi Imre : Bevezetés a mobil programozásba	146
12.2. C++ és Java nyelv összehasonlítása	146
13. Helló, Liskov!	148
13.1. Liskov helyettesítés sértése	148
13.2. Szülő-gyerek	150
13.3. Anti OO	152
13.4. Hello, Android!	153
14. Helló, Mandelbrot!	155
14.1. Reverse engineering UML osztálydiagram	155
14.2. Forward engineering UML osztálydiagram	156
14.3. BPMN	157
14.4. Egy esettan	158

15. Helló, Chomsky!	159
15.1. Encoding	159
15.2. l334d1c4 5	160
15.3. Full screen	161
15.4. Hiba-visszaterjesztéses perceptron	164
16. Helló, Chomsky!	166
16.1. Változó argumentumszámú ctor	166
16.2. JDK osztályok	168
16.3. Hibásan implementált RSA törése és Összefoglaló	170
17. Helló, Gödel!	175
17.1. Alternatív Tabella rendezése	175
17.2. STL map érték szerinti rendezése	178
17.3. GIMP Scheme hack	180
17.4. Gengszterek	185
18. Helló, !	187
18.1. OOCWC Boost ASIO hálózatkezelése	187
18.2. BrainB	188
18.3. SamuCam	191
18.4. FUTURE tevékenység editor	194
IV. Irodalomjegyzék	196
18.5. Általános	197
18.6. C	197
18.7. C++	197
18.8. Lisp	197

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 dblatex 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
dblatex 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 'dblatex' 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

DRAFT

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 videó:

Megoldás forrása:

- <https://gitlab.com/heinrichlaszlo/prog1codes/blob/master/vegtln1.c>
- <https://gitlab.com/heinrichlaszlo/prog1codes/blob/master/vegtln2.c>
- <https://gitlab.com/heinrichlaszlo/prog1codes/blob/master/vegtln3mag.c>

Végtelen ciklus , ami 100 százalékban dolgoztat meg egy magot:

```
#include <stdio.h>

int main()
{
    for(;;)
    {

    }

    return 0;
}
```

Ez a program nem túl bonyolult. Leegyszerűsítve annyi történik, hogy létrehozunk egy olyan ciklust, amelynek a ciklusfejrésze üres. Az így kapott ciklusban a futási feltétel rész is üres (első pontosvessző utáni rész), magyarul mindvégig igaz lesz, így a ciklus addig, amíg valaki kívülről meg nem szakítja (pl.: ctrl+c), folyamatosan futni fog egy processzormagot 100 százalékban dolgoztatva.

Végtelen ciklus , ami 0 százalékban dolgoztat meg egy magot:


```
#include <stdio.h>

int main()
{
    for(;;)
    {
        sleep(1);
    }

    return 0;
}
```

A második programot ha összehasonlítjuk az első programmal csupán két különbség üti fel a fejét. Az egyik különbséget az include-olás során fedezhetjük fel : az `unistd.h` header fájlt include-oljuk a `stdio.h` helyett. Ez a fájl tartalmazza a `sleep()` functiont. A `sleep()` function a zárójelében megadott időmennyiségig (ha nem adunk meg mást akkor alapértelmezetten másodpercben értve) késlelteti a szál feladatainak végrehajtását. Mivel a megírt ciklusunk a "végtelenségig" megy, így a `sleep()` számtalanszor végrehajtódik, 0 százalékban dolgoztatva meg az adott magot.

Végtelen ciklus , ami 100 százalékban dolgoztat meg minden magot:

```
#include <omp.h>

int main()
{
    #pragma omp parallel for
    {
        for(;;);
    }

    return 0;
}
```

A harmadik programban szintén egy új header-rel találkozunk: `omp.h`. Ez header az OpenMP-t include-ol, ami lehetővé teszi számunkra a párhuzamos kódolást.

A fenti programok eredménye a `htop` paranccsal lett letesztelve.

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 videó:

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 T1000-res programot elkészítjük a T100-as programot felhasználva.

```
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...

A T100-as program eldönti neki átadott programról, hogy van-e benne végtelen ciklus, majd vissza tér a true-val vagy false-sal. A T100-as visszatérési értékét átadjuk a T1000-es programnak, ha a kapott érték igaz, a program futása megáll, ha hamis, akkor végtelen ciklust kapunk. A T1000-es program akkor áll meg, ha a kapott érték igaz, vagyis ha a programban van végtelen ciklus, ellenkező esetben a T1000-es végtelen ciklusba kerül.

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:

- <https://gitlab.com/heinrichlaszlo/proglcodes/blob/master/csereexor.c>
- <https://gitlab.com/heinrichlaszlo/proglcodes/blob/master/cserekulonbseg.c>
- <https://gitlab.com/heinrichlaszlo/proglcodes/blob/master/cserekulonbseg.c>

```
{
    int elso= 5;
    int masodik= 10;

    printf("elso erteke:%d, masodik erteke:%d\n",elso, masodik);

    elso = elso-masodik; //elso = -5, masodik = 10
    masodik = elso+masodik; // masodik = 15, elso = 5
    elso = masodik-elso; // elso = 10, masodik = 5

    printf("Ertekeik a csere utan: ");

    printf("elso:%d, masodik:%d\n",elso, masodik);
}
```

```
int elso = 5;
int masodik = 10;

printf("%d, %d \n",elso,masodik);
printf("%s\n", "A csere után:");

elso = elso^masodik; //0000 1111 = 15 - elso
masodik=elso^masodik; //0000 0101 = 5 - masodik
elso=elso^masodik; //0000 1010 = 10 - elso

printf("%d, %d \n",elso,masodik);
```

```
int elso = 5;
int masodik = 10;

printf("%d, %d \n", elso, masodik);
printf("%s\n", "A csere után:");

elso = elso*masodik; //elso = 5*10=50;
masodik=elso/masodik; //masodik = 50/10=5;
elso=elso/masodik; //elso = 50/5=10;

printf("%d, %d \n", elso, masodik);
```

Minden programot egyszerű matematikai műveletekkel, bármiféle logikai utasítás vagy kifejezés nélkül hajtottuk végre. Először bekérünk a felhasználótól 2 tetszőleges számot, majd 3 lépésben felcseréljük őket. Végül kiírjuk egy egyszerű tetszőleges kiíratással.

A BogoMips

Mi az a BogoMips? A Bogomiaz a processzor sebességét hivatott mutatott mérőszám.

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

void delay (unsigned long long int loops)
{
    unsigned long long int i;
    for (i = 0; i < loops; i++);
}

unsigned long long int loops_per_sec = 1;
unsigned long long int ticks;

printf ("Calibrating delay loop..");
fflush (stdout);

while ((loops_per_sec <= 1))
{
    ticks = clock ();
    delay (loops_per_sec);
    ticks = clock () - ticks;

    printf ("%llu %llu\n", ticks, loops_per_sec);
```

A program elején includoljuk a `stdio` és `time` könyvtárakat. Majd létrehozuk a `delay` függvényt, ami `i=0`-tól "elszámol" `loops`-ig. Deklaráljuk a `loops_per_sec` és a `ticks` változókat. A `fflush` függvény miatt a megelőző sori `printf` kiíratásnak minden féle képpen végbe kell mennie. Következik egy `while` ciklus, ami addig tolja egyel balra a `loops_per_sec` értékét amíg az egész sor 0 nem lesz. A `clock()` függvény eredményét megkapja a `ticks` változó. Meghívjuk a `delay` függvényt, megadjuk

neki a `loops_per_sec` paraméterként. A `ticks` új értéket kap : `clock` függvény értékéből kivonjuk a `ticks` értékét. Majd kiíratunk `printf` segítségével.

```
if (ticks >= CLOCKS_PER_SEC)
{
    loops_per_sec = (loops_per_sec / ticks) * CLOCKS_PER_SEC;

    printf ("ok - %llu.%02llu BogoMIPS\n", loops_per_sec / 500000,
        (loops_per_sec / 5000) % 100);

    return 0;
}

printf ("failed\n");
return -1;
}
```

If feltétel vizsgálat : Ha a `ticks` értéke nagyobb vagy egyenlő mint `CLOCKS_PER_SEC`-é, a `loops_per_sec` értékét módosítjuk. Majd ezt a számot kiíratjuk. Ha a feltétel vizsgálat hamis akkor a programunk -1-gyel tér vissza.

2.4. Labdapattogás

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

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

Megoldás forrása:

- <https://gitlab.com/heinrichlaszlo/bhax/blob/codes/codes/turing/pat.c>
- <https://gitlab.com/heinrichlaszlo/bhax/blob/codes/codes/turing/patif.c>

if-es megoldás:

```
#include <stdio.h>
#include <curses.h>
#include <unistd.h>

int
main ( void )
{

    WINDOW *ablak;
    ablak = initscr ();
```

```
int x = 0;
int y = 0;

int xnov = 1;
int ynov = 1;

int mx;
int my;
```

A programunk legelején includoljuk a `stdio`, `curses` és `unistd` header fájlokat, könyvtárakat. A `stdio` teszi lehetővé számunkra az adat outputra történő kiíratását. A többi header fájlról, könyvtárról is ejtsünk pár szót: `curses` segítségével tudjuk meghívni a `initscr` függvényt, míg a `unistd`-re a `usleep` függvény miatt van szükségünk. Deklaráljuk az ablak nevű mutatót, majd inicializáljuk az `initscr` segítségével. Deklaráljuk még a azokat a változókat amik az ablak méretét tárolják, illetve a `xnov` és `ynov` változókat, ezek a labda pozíció változtatására vonatkozó adatokat tárolják.

```
for ( ;; )
{
    getmaxyx ( ablak, my , mx );

    mvprintw ( y, x, "O" );

    refresh ();
    usleep ( 100000 );
```

Következik egy végtelen ciklus. A `getmaxyx` tárolja az ablak méreteit, és a labdánk kiindulási pontját. A `mvprintw` írja ki folyton folyvást az `x` és `y` koordinátára azt az értéket amit harmadik paraméterként megadtunk ("O"). A `refresh` frissíti a képernyőt, hogy mindig az aktuális, friss képet láthassuk a képernyőnkön. A `usleep(x)` felel a labdánk lassításáért, a zárójelben megadott érték tetszőleges mikroszekundummal lassítja a program futását.

```
x = x + xnov;
y = y + ynov;

if ( x>=mx-1 ) {
    xnov = xnov * -1;
}
if ( x<=0 ) {
    xnov = xnov * -1;
}
if ( y<=0 ) {
    ynov = ynov * -1;
}
if ( y>=my-1 ) {
    ynov = ynov * -1;
}
```

Ebben a kódrészletben kerül sor a labdánk mozgatására, egyszerűen növeljük az `x` és `y` értékeket. Következik 4 feltétel vizsgálat amelyek lehetővé teszik, hogy a labdánk visszapattanjon a falról, olyan módon, hogy -1-el szorozza meg azokat a változókat amik a labda pozíciójának a módosításáért felelnek.

if nélküli megoldás:

```
#include <stdio.h>
#include <urses.h>
#include <stdlib.h>
#include <unistd.h>

int main (void)
{
    int xj = 0, xk = 0, yj = 0, yk = 0;
    int mx = 160, my = 48;

    WINDOW *ablak;
    ablak = initscr();
```

A program elején includoljuk az előző feladatban már kifejtett header fájlokat és az `stdlib.h`-t, ami az `abs()` függvény használatához lesz szükségünk. A `main`-ben megadjuk a szükséges változókat, illetve az ablak méreteit amik a maximum koordináták. Deklaráljuk az ablak nevű pointert az `initscr` segítségével.

```
for (;;)
{
    xj = (xj - 1) % mx;
    xk = (xk + 1) % mx;
    yj = (yj - 1) % my;
    yk = (yk + 1) % my;
    clear ();

    mvprintw (0, 0,
               " ←
               -----
               ");
    mvprintw (24, 0,
               " ←
               -----
               ");
    mvprintw (abs ((yj + (my - yk)) / 2),
               abs ((xj + (mx - xk)) / 2), "O");

    refresh ();
    usleep (100000);
}
```

Következik egy végtelen ciklus. A ciklusunk első 4 sorában definiáljuk a labdánk helyzetének meghatározására szolgáló változókat. Következik a `clear()` függvény ami a képernyő letisztítására szolgál. A `mvprintw` segítségével kiiratjuk a pálya alját és tetejét, amik jelen esetben szaggatott vonalak. Majd kiiratjuk magát a labdát is. Majd az előző `if`-es feladatban megismert `refresh()` és `usleep` függvénnyel találkozunk, annyi különbséggel, hogy az `usleep` értékének 1 tizedmsp lett megadva.

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 videó:

Megoldás forrása:

- <https://gitlab.com/heinrichlaszlo/prog1codes/blob/master/sz%C3%B3hossz.c>
- <https://gitlab.com/heinrichlaszlo/bhax/blob/codes/codes/turing/bogomips.c>

```
#include "std_lib_facilities.h"

int main()
{
    int szam=1;
    int db=0;

    while (szam!=0) {
        szam = szam<<1;
        ++db;
    }
    cout <<db<<"\n";
}
```

Deklarálunk két Integer típusú változót `szam` névvel 1-es értéket adva neki, majd `db` néven 0-ás értéket adva neki. Elindítunk egy `while` ciklust, a ciklus akkor fejezi be a futást, ha a `szam` nem lesz 0. Ezután a left shifting-et fogjuk használni, ami annyit tesz, hogy a `i` változó értéke minden egyes ciklus lefutásának a végén egyel balra ugrik.

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: <https://gitlab.com/heinrichlaszlo/prog1codes/blob/master/pagerank.c>

A PageRank a Google által, a kereséskor alkalmazott algoritmus. Ez alapján állít fel a google egy úgynevezett fontossági sorrendet a honlapok között. Minél nagyobb az adott honlap PageRank értéke annál előkelőbb helyen helyezkedik el ezen a bizonyos rangsorban.

```
#include <stdio.h>
#include <math.h>
```


Legelső sorban include-olnunk kell az `stdio` könyvtárat az eredmények képernyőn való megjelenítéséhez és a `math` könyvtárat a matematikai műveletek elvégzéséhez

```
void kiir(double tomb[], int db)
{
    int i;

    for(i=0; i<db;++i)
    {
        printf("%d%f \n", i, tomb[i]);
    }
}
```

Deklaráljuk a `kiir` függvényt. Deklaráljuk az `int` típusú `i` változót. A `for` ciklus `i=0`-tól `db`-ig megy, és kiírja a sorszámot, illetve a tömb `i`-edik elemét. Mivel a függvényünk `void` típusú, ezért a visszatérési értéke null (nincs)

```
double tavolsag( double PR[], double PRv[], int n)
{
    int i;
    double osszeg=0.0;

    for(i =0; i<n; ++i)
        osszeg+= (PRv[i] - PR[i]) * (PRv[i]-PR[i]);
    return sqrt (osszeg);
}
```

Deklaráljuk a `tavolsag` függvény, két `double` típusú tömböt adunk neki paraméterként, `PR` és `PRv` névvel, illetve deklarálunk egy `Integer` típusú, `n` nevű változót. Deklaráljuk az `i` `Integer` tipussal és az `osszeg` `double` típusú változót. A `for` ciklus `i=0`-tól `n`-ig megy, és a `for` ciklus `i=0`-tól `db`-ig megy, és az `osszeg` változó értéke a `PRv` tömb `i`-edik eleme és a `PR` tömb `i`-edik elemének a különbségének a szorzata. Eredményül a függvény az `sqrt` functionnel az `osszeg` változó négyzetgyökét adja vissza.

```
double L[4][4] =
{
    {0.0, 0.0, 1.0 / 3.0, 0.0},
    {1.0, 1.0 / 2.0, 1.0/ 3.0, 1.0},
    {0.0, 1.0 / 2.0, 0.0, 0.0},
    {0.0, 0.0, 1.0 / 3.0, 0.0}
};

double PR[4] = {0.0, 0.0, 0.0, 0.0};
double PRv[4]= {1.0/4.0, 1.0/4.0, 1.0/4.0, 1.0/4.0};

int i, j;
```

Létrehozzuk az `L` egy `double` típusú kétdimenziós, 4 soros, 4 oszlopos és a `PR` és `PRv` tömböket. Ezen kívül deklaráljuk `i` és `j`, `int` típusú változók is.

```
for(;;)
{
```

```
    for(i=0; i<4; ++i)
    {
        PR[i]=0.0;
        for(j=0; j<4; ++j)
            PR[i] += (L[i][j] * PRv[j]);
    }
}
```

Létrehozunk egy végtelen ciklust. Ezt követi egy belső `for` ciklus, ami 0-tól 4-ig fut, a `PR` tömb *i*-edik elemét 0.0-ra állítjuk be. Majd indítunk még egy ciklust, ami szintén 0-tól 4-ig megy. A `PR` tömb *i*-edik elemének értékét növeljük az `L` tömb *i*-edik, *j*-edik elemének és a `PRv` tömb *j*-edik elemének szorzatával.

```
if(tavolsag(PR, PRv, 4)<0.00001)
    break;

for(i=0; i<4; ++i)
    PRv[i]=PR[i];
}

kiir(PR, 4);
```

Feltételvizsgálat : Ha a `PR`, `PRv` és 4 paraméterekkel ellátott `tavolsag` függvény értéke kisebb, mint 0.00001, meghívódik a `kiir` függvény két paraméterrel: `PR`-rel és a 4-el. Ha a feltételvizsgálat hamis, elindul egy `for` ciklus 0-tól 4-ig, ami futása során `PRv` *i*-edik elemhelyére `PR` *i*-edik eleme kerül. Végül kiiradjuk.

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

Ez a szimuláció a Brun tételt ábrázolja szemléletesen. Tétel szerint a 2 különbségű prímek reciprokainak összege egy konstanshoz közelít. A `primes` a prímekből álló vektor, a `diff` pedig ezeknek különbségeit számolja ki, majd az `idx` az ikerprímek sorszámait tárolja, amik értékei a `t1`- és `t2primes`-ban tárolódnak. Aztán az `rt1plusrt2` ezek reciprokösszegeit veszi, majd a függvény visszatéríti a reciprokösszegek összegeit.

Ez után maga a szimuláció ábrázolja ezeket az értékeket, bizonyos maximális értékekig vett prímekekkel vett függvényeredményként, és valóban úgy tűnik, egy bizonyos számhoz konvergálnak.

forrás : <https://hu.wikipedia.org/wiki/Brun-t%C3%A9tel>

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

A probléma : Az egyik mögött rejlik a főnyeremény, egy autó, a másik kettő mögött pedig nincs semmi. Ön a második ajtót választja. Ekkor a műsorvezető kinyitja a másik két ajtó közül az egyiket, mondjuk a harmadikat. ami üres. Majd felteszi a kérdést, hogy szeretne-e változtatni és esetleg másik ajtót választani?

A válasz a kérdésre: IGEN, érdemes változtatni.

```
kiserletek_szama=100
kiserlet = sample(1:3, kiserletek_szama, replace=TRUE)
jatekos = sample(1:3, kiserletek_szama, replace=TRUE)
musorvezeto=vector(length = kiserletek_szama)
```

Deklaráljuk `kiserletek_szama`, a `kiserlet` és `jatekos` változókat a `sample` function-nel, amely mindkét esetben `kiserletek_szama` darabszámú (vagyis 100), 1 és 3 közötti, ismétlődhető véletlen számot generál. A `kiserlet` változó azt fogja tárolni, hogy éppen melyik ajtó mögött van a nyeremény, míg a `jatekos` azt, hogy melyik ajtót választottuk. A `musorvezeto` egy vector lesz, aminek mérete a `kiserletek_szama`.

```
for (i in 1:kiserletek_szama) {

  if(kiserlet[i]==jatekos[i]){

    mibol=setdiff(c(1,2,3), kiserlet[i])

  }else{

    mibol=setdiff(c(1,2,3), c(kiserlet[i], jatekos[i]))

  }

  musorvezeto[i] = mibol[sample(1:length(mibol),1)]

}
```

Egy ciklust hívunk meg, amelyben az 1-től indul egészen a `kiserletek_szama` által tárolt értékig. majd feltétel vizsgálat : ha a `kiserlet` *i*-edik eleme megegyezik a `jatekos` *i*-edik elemével, akkor a `mibol` vektor értéke egyenlő lesz az 1,2,3 számok és a `kiserlet` *i*-edik elemének "különbségével". Lényegében egy olyan sorszám lesz ez esetben a `mibol` értéke, amely mögött biztosan nincs már a nyeremény, ugyanis a játékos elsőre beletrafált, melyik mögött van a nyeremény, igaz, ő még nem tud róla. Ha ez nem teljesül, jön az else ág, amely lényegében ugyanez fordítva. A `mibol` értéke ugyanúgy az az érték lesz, ami az 1,2,3 számsorban megtalálható, azonban nem *i*-edik eleme sem a `kiserlet`nek, sem a `jatekos` *i*-edik eleme, ugyanis a műsorvezető csak olyan ajtót nyithat ki, amelyet a játékos még nem választott, és nyeremény sincs mögötte. A `musorvezeto` *i*-edik eleme pedig a `mibol` azon indexű eleme lesz, melyet az 1: `mibol` hossza intervallumból képzett, egy darab olyan szám lesz, melyet az előző feltételeknek megfelelő, elemei a `mibol` elemei lesznek.

```
nemvaltoztatesnyer= which(kiserlet==jatekos)
valtoztat=vector(length = kiserletek_szama)

for (i in 1:kiserletek_szama) {

  holvált = setdiff(c(1,2,3), c(musorvezeto[i], jatekos[i]))
  valtoztat[i] = holvált[sample(1:length(holvált),1)]

}

valtoztatesnyer = which(kiserlet==valtoztat)
```

A `which` függvénnyel megkapjuk azt az indexet, ahol a `kiserlet` és a `jatekos` értékek megegyeznek, ez lesz a `nemvaltoztatesnyer` értéke. Ez az az eset, amikor a játékos elsőre jól tippel és kitart a tippje mellett. A `valtoztat` egy vector lesz, aminek mérete a `kiserletek_szamaval` egyenlő. A következő cikluson belül történnek még érdekességek: a `holvált` értéke az lesz, ami 1,2,3 számhalmazban megtalálható, és cserébe sem a `musorvezeto`, sem pedig a `jatekos` *i*-edik eleme, vagyis se nem a játékos előzőleg, se nem a műsorvezető nem választotta ki még. A `valtoztat` *i*-edik eleme a `holvált` azon indexű eleme lesz, amely indexet az 1: '`holvált` hossza' (`length(holvált)`) intervallumból kapunk, elemei az előző feltételeknek megfelel, vagyis a `holvált` elemei. `valtoztatesnyer` értékét egyenlővé tesszük azzal az indexszel, amely indexen `kiserlet` és a `valtoztat` értékek megegyeznek.

```
sprintf("Kiserletek szama: %i", kiserletek_szama)
length(nemvaltoztatesnyer)
length(valtoztatesnyer)
length(nemvaltoztatesnyer)/length(valtoztatesnyer)
length(nemvaltoztatesnyer)+length(valtoztatesnyer)
```

Kiiratjuk az eredményt, amiből megállapíthatjuk, hogy tényleg nagyon esély van a győzelmere, ha új ajtót választunk.

3. fejezet

Helló, Chomsky!

3.1. Decimálisból unárisba átváltó Turing gép

Állapotátmenet gráfiával megadva írd meg ezt a gépet!

Megoldás videó:

Megoldás forrása:

- <https://gitlab.com/heinrichlaszlo/bhax/blob/codes/codes/chomsky/decitounar.c>
<https://gitlab.com/heinrichlaszlo/bhax/blob/codes/codes/chomsky/decitounar.c>

```
#include <iostream>
using namespace std;

int main(){
    int decimal;
    string unary;
    cin >> decimal;
    while(decimal>0){
        decimal--;
        unary+="1";
    }
    cout << unary;
}
```

Decimális számrendszerből vagy magyarul 10-es számrendszerből váltunk át egy számot egyes számrendszerbe.

Az 1-es számrendszerben mindössze 1 számjegyből állnak a számok, annyi darab 1-es áll a helyiértékek helyén amennyi a számunk valós értéke.

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 videó:

Megoldás forrása:

Vannak nyelven belüli szimbólumok, ezekhez a nyelvi szimbólumokhoz vannak hozzárendelési szabályok, melyek ezekhez az eredeti szimbólumtól eltérő szimbólumokat vagy szimbólumsorozatokat rendel.

A szimbólumoknak két fő típusa van: terminális és nem terminális.

A terminális szimbólum : amihez nem tartozik hozzárendelési szabály, tehát atomi, nem bontható tovább.

A nem terminális szimbólum : ezekhez a szimbólumokhoz tartozik hozzárendelés.

Ha van olyan grammatika ami őt generálja és szintén az akkor az adott nyelv környezet független.

1. $S \rightarrow aBC$
2. $S \rightarrow aSBC$
3. $CB \rightarrow CZ$
4. $CZ \rightarrow WZ$
5. $WZ \rightarrow WC$
6. $WC \rightarrow BC$
7. $aB \rightarrow ab$
8. $bB \rightarrow bb$ 9. $bC \rightarrow bc$
10. $cC \rightarrow cc$

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.

Megoldás videó:

Megoldás forrása:

- <https://gitlab.com/heinrichlaszlo/bhax/blob/codes/codes/chomsky/hivnyelv.c>

Tanulságok, tapasztalatok, magyarázat...

```
#include <stdio.h>

int main()
{
    for(int i=0; i<11; ++i)
    {
        printf("%d ", i );
    }

    return 0;
}
```

A program main részében egy for ciklus áll, ami 1-től 10-ig fut, minden futáskor a ciklusváltozó értékét növeljük egyel. Majd ezt az értéket kiiratjuk.

Nem sikerül a gcc-nek lefordítania a programot a C89-es szabvánnyal, ugyanis a változó ciklusfejben való deklarálása nem volt engedélyezett C89-ben. A fordító hibaüzenetben segítséget nyújt abbah, hogyan érdemes kijavítanunk a programunkat, hogy sikerre jussunk. Ez a for ciklus a legszemléletesebb példa a különbségek bemutatására.

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:

- <https://gitlab.com/heinrichlaszlo/bhax/blob/codes/codes/chomsky/valosszamok.l>

A Lex egy program, ami előre definiált szabályok alapján C kódot állít elő számunkra. A mi feladatunk ezeket a szabályokat definiálni.

```
%{  
#include <stdio.h>  
int realnumbers = 0;  
%}
```

Az egyes részeket százaléklelek választják el egymástól. Az első rész klasszik C kód, a definíciós rész, itt deklaráljuk változókat, lásd : számláló. Include-oljuk `stdio` függvénykönyvtárat

```
digit [0-9]  
%%  
{digit}*({digit}+)? {++realnumbers;  
    printf("[realnum=%s %f]", yytext, atof(yytext));}  
%%
```

Definiáljuk a szabályokat. Számjegyeinknek a `digit` elnevezést adtuk, későbbiekben ezen a néven fogunk rájuk hivatkozni.

```
%%  
{digit}*({digit}+)? {++realnumbers;  
    printf("[realnum=%s %f]", yytext, atof(yytext));}  
%%
```

A dupla százaléklelek között definiáljuk a szabályokat. Definiáljuk azokat a számokat, amiket a program keresni fog az inputon. Ha találtunk a szabályunknak megfelelő számot, a számlálónk értékét növeljük és egy egyszerű `printf` függvénnyel kiiratjuk magát a számot, ami a `yytext` változóban tárolódik.

Ez azonban egy string, amit double típusúvá szeretnénk átalakítani, ehhez használjuk az `atof` függvényt használjuk.

Azokat a számokat amiket keresünk, formailag a következők: Pont előtti rész: nulla vagy több darab számjegy. A pont utáni rész: 1 vagy több darab számjegyet tartalmaz.

```
int
main ()
{
    yylex ();
    printf("The number of real numbers is %d\n", realnumbers);
    return 0;
}
```

A mainben meghívásra kerül a `yylex` függvény.

3.5. l33t.l

Lexelj össze egy l33t ciphert!

Megoldás videó:

Megoldás forrása:

- <https://gitlab.com/heinrichlaszlo/bhax/blob/codes/codes/chomsky/l33t.l>

```
%{
#include <string.h>
%}
```

Az első százalékjelek közötti rész a definíciós rész. A feladat megoldásához include-oljuk `string` függvénykönyvtárat használjuk, mert a feladat során `Stringek`-re lesz szükség a megoldáshoz.

```
%%
[a-zA-Z]
%
```

A dupla százalékjellel definiáljuk a szabályokat. Szögletes zárójelek között megadjuk, hogy mikkel kell majd dolgoznia, vagyis mik lesznek az átalakítandó karakterek. Esetünkben ezek az ábécé kis- és nagybetűi lesznek.

```
switch(yytext[0])
{
    case 'a' :
        printf("4");
        break;

    case 'b' :
        printf("8");
        break;
```



```
    case 'e' :
        printf("3");
        break;

    case 'g' :
        printf("9");
        break;

    case 'l' :
        printf("1");
        break;

    case 's' :
        printf("5");
        break;

    case 'z' :
        printf("2");
        break;

    case 'o' :
        printf("0");
        break;

    default :
        printf("%s", yytext);
        break;
}
}
%%
%}
```

Egy `switch` szerkezetben megvizsgáljuk az input string karaktereit. Az `yytext` egy egyelemű karaktertömböt jelöl, aminek első tagjára kíváncsiak. A következőkben minden `case`-ben egy-egy Leet-beli számot rendelünk a betűkhöz, majd `printf`-fel kiíratjuk ezt.

```
int
main()
{
    yylex();
    return 0;
}
}
```

Létrehozzuk a "main"-ünket és meghívjuk az "yylex" függvényt , ami a programunk második fő része.

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 megy ránézésre, elkapja valamelyiket esetleg a splint vagy a frama?

i.

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

Ha mindenet jól csináltunk, akkor a programunk nem fog reagálni a billentyűzetről érkező jelekre.

ii.

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

A ciklusunk i=0-ról indul, i-nek az értékét addig növelem amíg az kisebb, mint 5, magyarul 4-ig.

iii.

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

A ciklusunk i=0-ról indul, i-nek az értékét addig növelem amíg az kisebb, mint 5, magyarul 4-ig.

iv.

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

a tömb i-edik eleme tárolja az i változó aktuális értékét, az i érték, ameddig a feltétel igaz, növekszik.

v.

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

vi.

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

A kiírásban kétszer hívjuk meg az f függvényt ami két paraméterrel rendelkezik. Paraméter átadás-kor növeljük először a második paramétert, majd az első paramétert.

vii.

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

Kiíratatjuk az f függvényt "a" paraméterrel és az "a" változóval.

viii.

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

Kiíratatjuk az f függvényt "a" paraméterrel és annak a címével és az "a" változóval.

Megoldás forrása:

Megoldás forrása:

- https://gitlab.com/heinrichlaszlo/bhax/blob/codes/codes/chomsky/forrasok_olvasasa.c
- https://gitlab.com/heinrichlaszlo/bhax/blob/codes/codes/chomsky/forrasok_olvasasa2.c

stb. stb.

- https://gitlab.com/heinrichlaszlo/bhax/blob/codes/codes/chomsky/forrasok_olvasasa9.c

Tanulságok, tapasztalatok, magyarázat...

3.7. Logikus

Hogyan olvasod természetes nyelven az alábbi Ar nyelvű formulákat?

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

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

```
)$
```

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

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

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

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

N interpretáció mellett a írt formulák:

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

vagyis

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

Prímek száma végtelen.

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

```
)$
```

vagyis

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

Ikerprímek száma végtelen.

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

vagyis

```

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

```

Prímek száma véges.

```

$$\$ (\text{\texttt{\textbackslash exists}} y \text{\texttt{\textbackslash forall}} x (y < x) \text{\texttt{\textbackslash supset}} \text{\texttt{\textbackslash neg}} (x \text{\texttt{\textbackslash text{ prím}}})) \$$$

```

vagyis

```

$$\exists y \forall x (y < x) \supset \text{\texttt{\textbackslash ensuremath}} \{ \text{\texttt{\textbackslash lnot}} \} (x \text{ prím})$$

```

Prímek száma véges.

3.8. Deklaráció

Vezesd be egy programba (forduljon le) a következőket:

- egész
- egészre mutató mutató
- egész referenciája
- egészek tömbje
- egészek tömbjének referenciája (nem az első elemé)
- egészre mutató mutatók tömbje
- egészre mutató mutatót visszaadó függvény
- egészre mutató mutatót visszaadó függvényre mutató mutató
- egészet visszaadó és két egészet kapó függvényre mutató mutatót visszaadó, egészet kapó függvény
- függvénymutató egy egészet visszaadó és két egészet kapó függvényre mutató mutatót visszaadó, egészet kapó függvényre

Mit vezetnek be a programba a következő nevek?

- ```
int a;
```

egész

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

egészre mutató mutató

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

egész referenciáját

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

egészek tömbje

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

egészek tömbjének referenciája

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

egészre mutató mutatók tömbje

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

egészre mutató mutatót visszaadó függvény

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

egészre mutató mutatót visszaadó függvényre mutató mutató

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

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

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

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

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

4. fejezet

Helló, Caesar!

4.1. int *** háromszögmátrix

Megoldás videó:

Megoldás forrása:

- <https://gitlab.com/heinrichlaszlo/bhax/blob/codes/codes/cesar%20haromszogmatrix.c>

Tutor : Kincs Ákos <https://gitlab.com/kincsa>

Mi is az a háromszögmátrix? A háromszögmátrix egy kvadratikus (négyzetes) mátrix, sorainak és oszlopainak a száma megegyezik. Két fajta létezik belőle: alsó-, és felső háromszögmátrix. Az alsó háromszögmátrix főátlója felett csupa nulla érték szerepel, míg a felsőnek a főátló alatti elemei mind nullák.

A mi programunk egy ilyen mátrixot fog készíteni: soook

A háromszögmátrix tartalmáról tudjuk, hogy bizonyos pozíciókon nulla áll, így felesleges minden elemét, vagyis $n*n$ darab értéket egyesével tárolni, elég csak $n*(n+1)/2$ darab elemet. A háromszögmátrixokat egy vektorba, vagyis egy dimenziós tömbbe lehet leképezni. A felső háromszögmátrixokat oszlopfolytonosan, míg az alsó háromszögmátrixokat sorfolytonosan szokás leképezni.

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

int
main ()
{
    int nr = 5;
    double **tm;

    printf("%p\n", &tm);
```

Includeoljuk a szükséges könyvtárakat, az stdio-val már többször találkoztunk, illetve az stdlib.h-ra is szükségünk lesz. Deklarálunk egy egészet nr néven, ő a sorok illetve oszlopok számát tartalmazza. Deklaráljuk továbbá a **tm mutatót is, ami egy pointerekre mutató pointer, majd kiíratjuk a címét a memóriában. Az eredményt hexadecimális számrendszerben kapjuk.

```
if ((tm = (double **) malloc (nr * sizeof (double *))) == NULL)
{
    return -1;
}

printf("%p\n", tm);
```

If feltétel vizsgálat : egy `double*` méretének `nr`-szeresét alokáljuk, vagyis lefoglalunk a memóriában, `double**`-nak, vagyis `double*`-okra mutató mutatónak. A `malloc void*`-ot ad vissza alaptól, de ezt típuskényszerítéssel meg tudjuk változtatni. Amennyiben ez `NULL` érték, nem sikerült a helyfoglalás, így a program hibával tér vissza. Kiírjuk a lefoglalt tárterület címét.

```
for (int i = 0; i < nr; ++i)
{
    if ((tm[i] = (double *) malloc ((i + 1) * sizeof (double))) == NULL) ←
    {
        return -1;
    }
}

printf("%p\n", tm[0]);
```

Egy `for` ciklussal megyünk `nr`-szer, és a következőt hajtjuk végre: lefoglalunk memóriát a `double` méretének `i+1`-szeresére. Erre `double*` -ot kényszerítünk, és rendre a `tm[i]` `i`-edik helyének adjuk értékül, vagyis ezekre a mutatókra fog mutatni `tm[i]`. A mutatót lényegében tömbként kezeljük. Ismét amennyiben ez `NULL` érték, nem sikerült a helyfoglalás, így a program hibával tér vissza. Kiírjuk az első mutató memóriacímét.

```
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");
}
```

Az első `for`-ciklusban feltöltjük a mátrixot tartalommal. `i`-vel és `j`-vel végigmegyünk a két dimenzión, majd az $i * (i + 1) / 2 + j$ értéket tesszük az adott helyre, vagyis szimplán sorba lesznek benne a számok nullától.

A második `for` ciklussal kiírjuk a tömbünk elemeit soronként.

```
tm[3][0] = 42.0;
(* (tm + 3)) [1] = 43.0;
*(tm[3] + 2) = 44.0;
*(* (tm + 3) + 3) = 45.0;

for (int i = 0; i < nr; ++i)
```

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

Négyszer megváltoztatjuk a mátrix tartalmát. Az első a legérthetőbb, a negyedik sor első eleme 42 lesz. Mindig a negyedik sor elemeit változtatjuk. A második alkalommal a második elem 43 lesz, majd a harmadik 44, illetve a negyedik 45.

For ciklussal kiírjuk a mátrix tartalmát.

```
for (int i = 0; i < nr; ++i)  
    free (tm[i]);  
  
free (tm);  
  
return 0;  
}
```

Felszabadítjuk a lefoglalt memóriát, először a for ciklusban a tm[]-ben levő double*-okat, majd a for után magát a double** tm-et is felszabadítjuk, majd a program hibamentesen tér vissza.

4.2. C EXOR titkosító

Írj egy EXOR titkosítót C-ben!

Megoldás videó:

Megoldás forrása:

- <https://gitlab.com/heinrichlaszlo/bhax/blob/codes/codes/cesar%20/e.c>

Az EXOR titkosítás lényegében nem más, mint a következő:

Van egy szöveged, amit titkosítani szeretnél és van egy megoldó kulcsod a titkosításhoz. A titkosítani kívánt szövegen a kulccsal exor műveletet hajtasz végre, ennek hatására a szövegből olvashatatlan dolgot hozol létre. A szépsége abban rejlik, hogy ezen az olvashatatlan szövegen újra alkalmazva a műveletet, ugyanazzal a megoldó kulccsal, visszakapjuk az eredeti szöveget, immáron olvasható módon. Magyarán az EXOR titkosítás lényege röviden és egyszerűen összefoglalva: az olvashatatlan szöveget csak akkor olvashatod el ha tudod a kulcsot hozzá.

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

A kód elején deklaráljuk azokat a könyvtárakat amiket szeretnénk használni. Deklaráljuk a unistd.h a read() és write() miatt, majd deklaráljuk a string.h (strlen(), strncpy) String függvények miatt. Mindezek mellett deklarálásra kerül a stdio.h könyvtár is.


```
#define MAX_KULCS 100
#define BUFFER_MERET 256
```

A kódunk legeslegelején megadunk két konstans értéket, a kulcs maximális méretét(MAX_KULCS) néven emellett az ideiglenes tároló méretét(BUFFER_MERET) néven.

```
int
main (int argc, char **argv)
{
```

Létrehozuk a main() metódust. A main() paraméterei a következők : Az első az argc az argumentumok számát tárolja, a második pedig az argv az argumentumokat tároló vektor. Ezeknek az argumentumoknak a jelszó megadásánál lesz jelentős szerepe.

```
char kulcs[MAX_KULCS];
char buffer[BUFFER_MERET];
```

Létrehozunk 2 char típusú tömböt (azaz Stringet). Az egyiket MAX_KULCS, a másikat BUFFER_MERET mérettel.

```
int kulcs_index = 0;
int olvasott_bajtok = 0;
```

Deklarálunk és inicializálunk 2 egész típusú változót amiknek 0 értéket adunk. Az egyikben azt tároljuk, hogy a kulcs hanyadik indexénél járunk, a másikban pedig eltároljuk a bufferbe olvasott bajtok számát.

```
int kulcs_meret = strlen (argv[1]);
strncpy (kulcs, argv[1], MAX_KULCS);
```

Ebben a kódrészletben létrehozunk egy egész típusú változót, amiben a kulcs tényleges méretét fogjuk eltárolni.Ezt úgy kapjuk meg, hogy az strlen() függvényt alkalmazzuk az argumentum vektor 2. elemére(argv[1]), magyarul a megadott jelszóra. A strncpy() metódust alkalmazva a már létrehozott kulcs char tömbbe másoljuk a program 2. argumentumát, magyarul a kulcsot. A függvény paraméterként a következőket várja: hova másoljon, mit másoljon és hogy hány bajtot másoljon. Utóbbiként a MAX_KULCS-ot adjuk meg.

```
while ((olvasott_bajtok = read (0, (void *) buffer, BUFFER_MERET)))
{
```

Következik egy while ciklus aminek a feltétele a következő : az olvasott_bajtok változóban tároljuk a read() által visszaadott értéket, ami egészen addig pozitív lesz amíg a függvény sikeresen lefut, tehát amíg van mit beolvasni a programunk futni fog. Visszatérési értéként pedig nem csak egy egyszerű pozitív számot ad, hanem a beolvasni sikerült bajtok számát is.

```
for (int i = 0; i < olvasott_bajtok; ++i)
{

    buffer[i] = buffer[i] ^ kulcs[kulcs_index];
    kulcs_index = (kulcs_index + 1) % kulcs_meret;

}
```

A while cikluson belül található egy for ciklus, ami végig megy a beolvasott szövegen. Miközben végig megy, az aktuális elem és a kulcs kulcs_index-edik eleme között exor műveletet hajt végre. ezt követően a kulcs_index-et növeljük egyel és elosztjuk az így kapott értéket kulcs_meret-tel a maradékát véve, ezzel biztosítjuk azt, hogy az érték sose fogja túllépni a kulcs méretén.

```
        write (1, buffer, olvasott_bajtok);  
  
    }  
}
```

A write() függvény segítségével kiírjuk a standard outputra(1) a buffer tartalmát, ami olvasott_bajtok méretű. Ezzel amit beolvastunk, exoroztunk, most kiíratjuk. A ciklus így megy tovább, amíg az egész bemenet feldolgozásra nem kerül. A program teljes lefutása után a kimenet a titkosított vagy éppen már a visszakódolt szöveg lesz.

4.3. Java EXOR titkosító

Írj egy EXOR titkosítót Java-ban!

Megoldás videó:

Megoldás forrása: <https://www.tankonyvtar.hu/hu/tartalom/tkt/javat-tanitok-javat/ch01.html> 1.12. példája - Titkosítás kizáró vaggyal és [https://gitlab.com/heinrichlaszlo/bhax/blob/codes/codes/ceasar%20ExorTitkos%C3%](https://gitlab.com/heinrichlaszlo/bhax/blob/codes/codes/ceasar%20ExorTitkos%C3%9C)

Tanulságok, tapasztalatok, magyarázat...

```
public class ExorTitkosító {  
  
    public ExorTitkosító(String kulcsSzöveg,  
        java.io.InputStream bejövőCsatorna,  
        java.io.OutputStream kimenőCsatorna)
```

C programozási nyelvtől eltérően a programunk elején nincs szükségünk könyvtárak includolására. Létrehozzuk az ExorTitkosító objektumunkat három paraméterrel : kulcsként használt szöveg , input , output.

```
    throws java.io.IOException {  
  
        byte [] kulcs = kulcsSzöveg.getBytes();  
        byte [] buffer = new byte[256];  
        int kulcsIndex = 0;  
        int olvasottBajtok = 0;
```

Definiálunk kettő byte típusú tömböt kulcs és buffer, kulcs-ba kerül majd a kulcsszöveg , buffer-be megadjuk a méretét. A kulcs indexelésének és a beolvasott bajtok számolására létrehozunk két integer típusú változót , úgy nevezett számlálót

```
        while((olvasottBajtok =  
            bejövőCsatorna.read(buffer)) != -1) {  
  
            for(int i=0; i<olvasottBajtok; ++i) {
```

```
        buffer[i] = (byte)(buffer[i] ^ kulcs[kulcsIndex]);
        kulcsIndex = (kulcsIndex+1) % kulcs.length;
    }

    kimenőCsatorna.write(buffer, 0, olvasottBájtok);
}
```

Egy while ciklus fut addig, amíg van beolvasandó bájtunk. Azokat a bájtokat amiket nem tudunk beolvasni, úgy nevezett olvashatatlan szöveg megegyezik a buffer tömb *i*-edik elemével, ami a *buffer* *i*-edik eleme és az *kulcs* tömb *kulcs_index*-edik elemének vett EXOR művelet eredménye. A művelet eredménye byte típusú. Majd növeljük a *kulcs_index* értékét. Ezek után kiiratjuk a *write* függvénnnyel a buffer tömb elemeit nullától az *olvasottBájtok*-ig.

```
public static void main(String[] args) {

    try {

        new ExorTitkosító(args[0], System.in, System.out);

    } catch (java.io.IOException e) {

        e.printStackTrace();

    }

}
```

A Try - Catch -en belül példányosítunk. Létrehozunk egy új ExorTitkosítót ami a parancsori argumentumként kapott szöveget fogja átalakítani. A catch-en belül meghívjuk a *printStackTrace*-t, ami visszajelzést ad arról, hogy mi a hiba és hol található.

4.4. C EXOR törő

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

Megoldás videó:

Megoldás forrása:

- <https://gitlab.com/heinrichlaszlo/bhax/blob/codes/codes/ceasar%20t.c>

```
#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;
}
```

Kiszámítjuk az átlagos szóhosszt. Először megszámloljuk a szövegben a szóközöket ezután visszaosztjuk vele az összes karakterek számát.

```
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");
}
```

A tiszta szöveg nagy valószínűséggel tartalmaz olyan gyakori magyar szavakat mint: nem, hogy, ha, az. Az átlagos szóhossz segítségével csökkenthetjük a végrehajtandó törések számát is.

```
void
xor (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
xor_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);
}
```

Bájtónként végrehajtjuk az EXOR-t. A % segítségével a kulcs nem lépi át a keresett hosszt.

```
int
main (void)
{

    char kulcs[KULCS_MERET];
    char titkos[MAX_TITKOS];
    char *p = titkos;
    int olvasott_bajtok;

    while ((olvasott_bajtok =
        read (0, (void *) p,
            (p - titkos + OLVASAS_BUFFER <
                MAX_TITKOS) ? OLVASAS_BUFFER : titkos + MAX_TITKOS - p)))
        p += olvasott_bajtok;

    for (int i = 0; i < MAX_TITKOS - (p - titkos); ++i)
        titkos[p - titkos + i] = '\\0';

    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);

                                    exor (kulcs, KULCS_MERET, titkos, p - titkos);
                                }
            }
}
```

```
    }  
  
    return 0;  
}
```

A while ciklus addig fog futni, amíg van bemenet. Egymásba ágyazott for ciklusokkal előállítjuk az összes lehetséges kulcsot majd ha a kulcsok előálltak ki is próbáljuk őket. A végén újra exor-ozunk.

4.5. Neurális OR, AND és EXOR kapu

R

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

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

Ez az R szimuláció egy neurális háló használatával oldja meg a leggyakrabbi logikai műveleteket.

OR vagy VAGY: ha mindkét bit 0, akkor az or is 0, egyébként pedig 1.

AND vagy ÉS: csak akkor 1, ha mindkét bit 1, egyébként 0.

EXOR vagy KIZÁRÓ VAGY.

A tanuló algoritmust "megetetjük" ezen műveletek két argumentumával és ezek kimeneteivel, aztán ezen halmazok alapján az idegháló megpróbálja rekreálni azokat.

Ez láthatóan az or és and műveletnél rendkívül pontos eredményre vezet, tehát konklúzióként levonható, hogy a program megtanulta a műveleteket.

4.6. Hiba-visszaterjesztéses perceptron

C++

Megoldás videó: <https://youtu.be/XpBnR31BRJY>

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

Tanulságok, tapasztalatok, magyarázat...

A Neurális OR, AND és EXOR kapu feladatnál már találkozhattunk a neuron és a gépi tanulás fogalmával. A perceptron leegyszerűsítve nem más mint ezen neuron mesterséges intelligenciában használt változata. Tanulásra képes, a bemenő 0-k és 1-esek sorozatából mintákat tanul meg és súlyozott összegzést végez. A következő feladat során egy ilyen perceptront fogunk elkészíteni aminek alapvetően a feladata az, hogy a `mandelbrot.cpp` programunk által létrehozott Mandelbrot-halmazt ábrázoló PNG kép egyik színkódját vegye és az a színkód legyen a többretegű neurális háló inputja. !!!!! ÁTFOGALMAZNI!!!!

```
#include <iostream>  
#include "mlp.hpp"  
#include <png++/png.hpp>
```

Includoljuk `aziostream`, az `mlp` és a `png++/png` könyvtárakat. A `mlp` könyvtárra azért van szükségünk mert több rétegű perceptront fogunk létrehozni. A `png++/png` könyvtár a PNG kiterjesztású képállományokkal való munkát teszi számunkra lehetővé.

```
using namespace std;

int main(int argc, char **argv)
{
    png::image <png::rgb_pixel> png_image(argv[1]);

    int size = png_image.get_width()*png_image.get_height();

    Perceptron *p = new Perceptron(3, size, 256, 1);
```

A `main` függvényünk első sorában megmondjuk, hogy a képállományunk beolvasása az 1-es parancsori argumentum alapján történik. Eltároljuk egy segédváltozóban a kép méretét a `get_width` és a `get_height` szorzatát, majd létrehozuk a perceptront a `new` operátor segítségével.

```
double* image = new double[size];

for(int i=0; i<png_image.get_width(); ++i)
    for(int j=0; j<png_image.get_height(); ++j)
        image[i*png_image.get_width()+j] = png_image[i][j].red;

double value = (*p)(image);

cout<<value;

delete p;
delete [] image;
```

Az egyik `for` ciklussal végig megyünk a kép szélességét alkotó pontokon, majd a másik `for` ciklussal végig megyünk a magasságát alkotó pontokon. Az `image`-ben fogjuk tárolni a vörös képpontokat. A `value` értéke adja meg a Perceptron `image`-ra történő meghívását, ami egy `double` típusú változót tárol. Kiirattjuk és töröljük azokat az elemeket amiket már nem használunk, így az addig lefoglalt memória egységeket újra használhatóvá tettük.

Fordítjuk és futtatjuk a képen látható módon:

5. fejezet

Helló, Mandelbrot!

5.1. A Mandelbrot halmaz

Megoldás videó:

Megoldás forrása: https://progpater.blog.hu/2011/03/26/kepess_egypercetek <https://gitlab.com/heinrichlaszlo/-/blob/codes/codes/mandelbrot/mandelpng.c++>

Tutor : Kincs Ákos <https://gitlab.com/kincsa>

Benoît Mandelbrot egy Lengyel származású matematikus, aki a 20. században élte életét és csinálta munkásságát, az Ő nevéhez fűződik többek között a Mandelbrot-halmaz fogalom is, amit a komplex számsíkon ábrázolunk és amelyet Mandelbrot fedezett fel az 1970-es évek végén. A Mandelbrot-halmazok ábrázolására a komplex számokat használjuk. A komplex számok egyik közös jellemzője, hogy abban az esetben tartoznak Mandelbrot halmaz elemei közé, ha őket $x_{n+1} = x_n^2 + c$ sorozatba behelyettesítve egy konvergens sorozatot kapunk. Az előzőekben említett halmaz ábrázolása egy fraktál alakzatot eredményez, amely egy olyan alakzat, aminek körvonala nem egyenes, hanem "kitülemkedések" figyelhetők meg rajta illetve ezen alakzatok hasonlítanak egymásra. Forrás : https://hu.wikipedia.org/wiki/Komplex_sz%C3%A1mok

Ebben a feladatban egy olyan programot fogunk vizsgálni, ami egy ilyen Mandelbrot-halmazt állít elő. A program nem más mint a következő:

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

Mindenek előtt A programunk legelején a png++/png és a iostream könyvtárakat inkludáljuk, a png++/png könyvtár lehetővé teszi számunkra a PNG kiterjesztésű képállományokkal való munkát. Ez nagyon fontos számunkra, ugyanis egy PNG képet szeretnénk megrajzoltatni programunkkal. Amennyiben a png++/png könyvtár még nincs telepítve a számítógépünkre, a feladatunk megoldásához feltétlenül szükséges ezt telepíteni.

```
int main (int argc, char *argv[])
{
    if (argc != 2) {
        std::cout << "Hasznalat: ./mandelpng fajlnev";
        return -1;
    }
}
```


Ebben a kódrészletben parancssori argumentumként adjuk meg, hogy milyen fájlba legyen mentve a képünk, és ha ez a futtatás során elmarad, felhívjuk a felhasználó figyelmét a helyes futtatási módra.

```
// számítás adatai
double a = -2.0, b = .7, c = -1.35, d = 1.35;
int szelesseg = 600, magassag = 600, iteraciosHatar = 1000;

// png-t készítünk a png++ csomaggal
png::image <png::rgb_pixel> kep (szelesseg, magassag);

// a számítás
double dx = (b-a)/szelesseg;
double dy = (d-c)/magassag;
double reC, imC, reZ, imZ, ujreZ, ujimZ;
// Hány iterációt csináltunk?
int iteracio = 0;
std::cout << "Szamitas";
```

Létrehozzuk a számolásunkat szolgáló változókat. A double típusú változókkal a számsík határait adjuk meg az X és az Y tengelyen, majd újabb double típusú változók kerülnek létrehozásra, amik a kép szélességének és magasságának tárolására hivatott változók, illetve egy int típusú változó, ami az iterációk darabszámának felső korlátját jelentő érték.

Létrehozzuk a képünket, aminek mérete: szelesseg x magassag, Ez a kép jelenleg még egy sima, üres kép. A dx és dy változókat a számsíkon való lépésköz meghatározására létrehozzuk. Majd létrehozzuk az im és re előtagú változókat Z és C változókhoz, amelyek az adott komplex szám (C vagy Z) valós és képzetes részét jelölik.

Létrehozunk egy int típusú változót 0 értékkel, ami az iterációk számának nyilvántartására szolgál.

```
// 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;
    }
}
```

Két for ciklussal végigmegyünk a szelesseg x magassag "rács"on", majd inicializáljuk C valós és képzetes részeit tartalmazó változókat, ezek után ugyanezt tesszük Z esetében is. Létrehozásra kerül az iterációk számát tartalmazó változó.

```
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;
```

A következőekben megvizsgáljuk egy while ciklusban, hogy z_n abszolútértéke kisebb-e mint 2 és, hogy elértük-e az iterációs határt. Ha a ciklusfejben található feltétel teljesül, akkor módosítjuk a Z változó értékeit, hogy Z változó értéke: $z_n^2 + c$ -re módosuljon. A Z változó új értékeit Z eredeti változóiba másoljuk. Az `iteracio` számlálót növeljük, mivel ez egy iteráció volt. Ha az iterációk száma eléri a határt, az a következőt jelenti: az iterációnak van határértéke, vagyis a C szám eleme a Mandelbrot-halmaznak. Mivel a C szám eleme a Mandelbrot-halmaznak a hozzá tartozó képpontot színezzük.

```
kep.set_pixel(k, j, png::rgb_pixel(255-iteracio%256,  
                                   255-iteracio%256, 255-iteracio%256));  
  
kep.write (argv[1]);  
std::cout << argv[1] << " mentve" << std::endl;
```

A `set_pixel` függvény hozzáférést ad nekünk a k, j koordinátákon található képpontokhoz, majd ezen képpontok színét az `rgb_pixel` függvénnyel módosítjuk. A `kep` fájlba kiírjuk kirajzolt Mandelbrot-halmazunkat. A `png` fájl létrejöttéről a program egy üzenetet küld a felhasználónak --> "mentve".

Fordítjuk és futtatjuk a programot a következő parancsokkal:

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

Megoldás videó:

Megoldás forrása: https://gitlab.com/nbatfai/bhax/blob/master/attention_raising/Mandelbrot/3.1.2.cpp <https://gitlab.com/heinrichlaszlo/bhax/blob/master/codes/codes/mandelbrot/3.1.2.cpp>

A feladat lényegében ugyanaz, mint az előbb. Ugyanúgy egy olyan program elkészítése a cél, ami egy Mandelbrot-halmazt generál, azonban most az `std::complex` osztály használatával/segítségével. A megoldásnak a lényege annyiban más mint az előző feladatban, hogy nem szükséges a feladat megoldásához szükséges számok valós- és képzetes részét külön-külön változóba tárolni, ugyanis megoldható egyetlen egy darabban is.

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

Az első két könyvtárat már az előző feladatban kiveséztük. Továbbá ahogy a feladat is kérte, include-oljuk a `complex` függvénykönyvtárat is, ami komplex számokkal való könnyebb számolást teszi nekünk lehetővé.

```
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;
```

A main függvényben megadjuk a készülő képünk méreteit majd megadjuk a maximálisan megengedett iterációk számát is, ez az iterációs határ. Ezek mellett is inicializálásra kerülnek a komplex számsík határértékei, magyarul, hogy az alaphalmaz mely tartományai között dolgozunk.

```
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: ./mandelbrot_komplex fajlnev szelesseg ↔
    magassag n a b c d" << std::endl;

    return -1;
}
```

Kezdődik az if feltétel vizsgálat. A feltétel vizsgálatunk a következő : ha a futtatáskor megadott parancssori argumentumok száma 9, akkor a változók értékének beállítjuk a parancssori argumentumokat. Mindez az atoi és az atof függvényeknek köszönhetően jön létre. Röviden, hogy mit csinál a két függvény ? Az atoi egy olyan függvény amely stringet alakít át integerré. Az atof egy olyan függvény ami pedig stringet alakít át double -lé.

A függvény vizsgálatunk else része : Hogyha a felhasználó által megadott parancssori argumentumok száma nem 9, akkor egyszerűen kiíratjuk, mi az argumentumok megfelelő sorrendje . Majd -1-gyel tér vissza a program.

```
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;
```

Az image függvénnyel létrehozuk szelesseg*magassag felbontású képünket. Még ez a kép egy üres png kiterjesztésű állomány. A dx, dy változókat definiáljuk, a reC, imC, reZ, imZ változókat pedig deklaráljuk. Definiálunk double típusú változókat és az int típusú iteracio változó értékét 0-ra állítjuk. A "re" előtag a komplex szám valós részét és az "im" előtag a komplex szám képzetes részét jelöli.

```
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;
```

Elindítunk két for ciklust, az egyiket magassagig, a másikat pedig szelessegig. Magyarul egy úgynevezett rácsot járunk be a komplex számok számsíkján.

Az reC és a imC változó értékét a képen látható módon adjuk meg. A program elején inkludált complex osztály segítségével létrehozuk a c változót, ami megkapja a komplex szám valós részét, majd a képzetes részét is. Hasonló módon megkapjuk z_n értékét is. Ugyanúgy, a zárójelben lévő 1. szám a valós rész, míg a 2. szám a képzetes része lesz a számnak. A c változó lesz a vizsgált rácspont programunk során.

```
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 ) );
}
```

Egy while ciklus segítségével történik meg halmaz színezése. A ciklus futási feltétele a következő: meghívjuk az abs függvényt z_n-re, ha z_n abszolútértéke kisebb, mint 4 és az iteracio számláló elérte a program elején definiált iterációshatárt, akkor van határértéke az iterációnak. A z_n értékét $z_n^2 + c$ -re változtatjuk. Az iteracio számlálót egyel növelem, mert a ciklus akkor fejeződik be, ha elértük az iterációs határt. Ez azt jelenti, hogy az adott pont Mandelbrot-halmaz elem, ezért azt be kell jelölnünk a szám-

síkon. A kepre meghívjuk a `set_pixel` függvényt, ami kiszínezi a `k`, `j`-edik képpontot az `rgb_pixel` függvénynek átadott színnel.

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

Létrehozuk az integer típusú `szazalek` változót, ami futtatás után azt jelöli majd, hány %-ban van kész a kép kirajzoltatása. Meghívjuk a `flush` függvényt. A `flush` függvény biztosítja azt, hogy a bufferben lévő összes bájtt kiíratásra kerül.

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

A `write` függvénnyel kiíratjuk az elkészült képünket az 1-es indexű parancssori argumentumként megadott fájlba. A legutolsó sorban kiíratunk egy üzenetet, hogy a képünk el lett mentve.

Fordítjuk és futtatjuk a programot:

5.3. Biomorfok

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

Megoldás forrása: https://gitlab.com/nbatfai/bhax/tree/master/attention_raising/Biomorf<https://gitlab.com/heinrichlaszlo/bhax/blob/codes/codes/mandelbrot/3.1.3.cpp>

Tutoriált : Kincs Ákos <https://gitlab.com/kincsa>

Tanulságok, tapasztalatok, magyarázat...

A biomorfok leginkább olyan formák vagy alakzatok amik egy úgynevezett biológiai organizmusra hasonlítanak. Pickover amerikai kutató egy Julia-halmazhoz kapcsolódó programban található hiba következtében fedezte fel ezeket az alakzatokat. Aktuális feladatunk során egy ilyen biomorfot fogunk megrajzoltatni felhasználva az előző, Mandelbrot-halmazos feladatot. A kirajzolt alakzat az előzőhöz hasonlóan szintén fraktál lesz. Forrás és a Biomorfokról bővebben : 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>
```

Az inkludált könyvtárakat már kiveséztük , nincs bennük változás az előző feladatban inkludált könyvtárakhoz képest.

```
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;
```

A main függvényben definiált változók nagy része kis mértékben tér el az előző feladatban definiált változóktól, így ez most nem igényel különösebb magyarázást. Definiáljuk a C vizsgált rácspontot definiáló `reC` `double` típusú változót, ami a C komplex szám valós része és az `imC` `double` típusú változót, ami a C komplex szám képzetes része.

```
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;
}
```

Feltétel vizsgálat :ellenőrizzük a program helyesen történő futtatását. Ha nem, kiiratjuk neki a helyes módot. Ha igen, a megfelelő indexű parancssori argumentumokat az `atof` és `atoi` függvényekkel átalakítjuk megfelelő típusú értékekké, ezután az így kialakított értékeket hozzárendeljük a változóinkhoz.

```
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";
```

Ebben a kódrészletben hozzuk létre a képünket, melynek mérete: `szelesseg*magassag`. Definiáljuk a `cc` komplex számot is, `cc` paraméterei azok a valós és képzetes részek lesznek, amit az előbbieken parancssori argumentumként kapott.

```
// 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 ));
    }
}
```

Két egyszerű forciklussal végig megyünk a rácson . A complex osztály és a valós és képzetes részek felhasználásával létrehozzuk a `z_n` változónkat. Egy harmadik for cikluson kerül meghívásra, ami iterációs határ eléréséig fut. A `z_n` értékét a képlet szerint $z_n = z_n^3 + c$ -re változtatjuk meg és egyszerűen az érintett koordinátákon megtalálható képpontokat színezzük.

```
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;
```

Létrehozásra kerül a százalék nyilvántartására szolgáló változó, amit már az előző feladatban kiveséztünk.

5.4. A Mandelbrot halmaz CUDA megvalósítása

Megoldás videó:

Megoldás forrása: https://gitlab.com/nbatfai/bhax/blob/master/attention_raising/CUDA/mandelpngc_60x60_100.
https://gitlab.com/heinrichlaszlo/bhax/blob/codes/codes/mandelbrot/mandelpngc_60x60_100.cu

Tapasztalat, tanulságok, magyarázat...

Mi is az a CUDA? A CUDA az Nvidia által kifejlesztett, jellemzően C, C++ nyelvekkel remek összhangban működő technológia ami lehetővé teszi a programot író számára a videokártya erőforrásait felhasználva a párhuzamos számítást. A programot sajnos nem áll módomban futtatni, ugyanis nem áll rendelkezésemre CUDA kártya. A programkód .cu kiterjesztésű de leginkább C++ szintaktika található benne.

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

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

#define MERET 600
#define ITER_HAT 32000
```

A program legelején inkludáljuk a megfelelő könyvtárakat. A `sys/times.h` header fájl teszi lehetővé a processzoridővel való munkát. Ahogy azt már az előző Mandelbrotos programok során kiveséztük, most is nevesített konstansokat használunk a méret és az iterációs határ méretére vonatkozóan.

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

    ++iteracio;
}
return iteracio;
}

```

A `__device__` a függvény neve előtt azt adja meg, hogy a függvényt a kártya hajtsa végre. Létrehozásra kerülnek változók amik a számításhoz szükséges adatokat tartalmazzák, mint a szélességet, magasságot, iterációs határt, lépésközt és iterációk számát tartalmazó változók. A függvényen belül ugyanaz történik, mint az előző feladatok esetében. Nem áll rendelkezésre az `std::complex` osztályunk, ezért a z és c komplex számok valós és képzetes részeit külön változóban kell tárolnunk. Majd végigmegyünk a szélesség \times magasság rácson. Következik egy `while` ciklus, amiben a következőt vizsgáljuk: z_n abszolútértéke kisebb-e mint 2 és hogy elértük-e az iterációs határt. Ha ez igaz, akkor módosítjuk a Z változó értékeit: $z_n^2 + c - re$. A Z változó új értékeit Z eredeti változóiba másoljuk. Az `iteracio` számlálót növeljük. Majd a végén a függvény az iterációknak a számát adja vissza.

```

/*
__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);

}

```

A `__global__` a függvényünk előtt azt jelzi, hogy ezt a functiont a videokártya fogja végrehajtani. A CPU hívja meg a függvényt, de a GPU hajtja végre. A feladatban 60x60 darab blokkal dolgozunk, blokkonként 100 darab szállal, így a `threadIdx.x` és `threadIdx.y` azt jelölik, hogy az értékek kiszámítása melyik szálon fut, ezeket az adatokat `tj` és `tk` változóban tároljuk. Majd ezt az értékeket újra felhasználjuk a `j` és `k` változókat számoljuk ki. A `blockIdx.x` és `blockIdx.y` változókat is amik a blokkokat azonosítják. A `j` és `k` változó értéke: `blokk*10+szál` Az ilyen módon megkapott értékeket paraméterként adjuk át a `mandel` függvénynek.

```

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

    int *device_kepadat;
    cudaMalloc ((void **) &device_kepadat, MERET * MERET * sizeof ( ←
        int));

    dim3 grid (MERET / 10, MERET / 10);
    dim3 tgrid (10, 10);
    mandelkernel <<< grid, tgrid >>> (device_kepadat);

    cudaMemcpy (kepadat, device_kepadat,
        MERET * MERET * sizeof (int), cudaMemcpyDeviceToHost);
    cudaFree (device_kepadat);

}

```

A függvénynek átadunk egy tömböt. Létrehozunk egy `int` típusú pointert vagy mutatót majd a `cudaMalloc` függvény segítségével memóriát foglalunk számára. Létrehozunk két 3 dimenziós vektort amikben számértékeket tárolunk. A `cudaMemcpy` függvénnyel a `device_kepadat` értékét a `kepadat`-ba másoljuk. A függvény harmadik paramétere az átmásolandó bájtokat, míg az utolsó a másolás típusát adja meg ami Device to Host. Majd a legvégén `cudaFree` függvény segítségével felszabadítjuk a lefoglalt memóriát.

```

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

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

```

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

    int kepadat[MERET][MERET];

    cudamandel (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;

    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;
```

Egy változóban tároljuk a `clock` függvény értékét, ami a program által elhasznált processzoridőt adja vissza. Akár csak az előzőekben, ha helytelenül futtatná az illető a programot, figyelmeztetjük őt. Meghívásra kerül a `cudamandel` függvény és létrejön a képünk, ami természetesen még egy üres állomány. Már ismert módon végigmegyünk a sorokon és oszlopokon és ha az adott képpontról kiderül, hogy a Mandelbrot-halmaz eleme, akkor kiszínezzük azt. a legvégén tájékoztatjuk a felhasználót a kép mentésének sikerességéről: mentve és arról, hogy mennyi időnkbe került a kép elkészülése.

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:

Megoldás forrása: https://sourceforge.net/p/udprog/code/ci/master/tree/source/kozepes/Qt/mandel_nagyito/-
https://gitlab.com/heinrichlaszlo/bhax/tree/codes/codes/mandelbrot/mandel_nagyito

Tanulságok, tapasztalatok, magyarázat...

A feladat megoldásához a Qt-t fogjuk használni, ha a Qt eszköztár nincs letöltve, le kell töltenünk. A Qt egy olyan eszköztár, amivel grafikus felhasználói felületű programokat tudunk létrehozni.

A következő 5 fájlra van szükségünk, hogy minden a tervek szerint hiba mentesen történjen:

- main.cpp
- frakszal.cpp
- frakszal.h
- frakablak.cpp
- frakablak.h

main.cpp

```
#include <QApplication>
#include "frakablak.h"

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

    FrakAblak w1;
    w1.show();
    return a.exec();
}
```

Inkludolásra kerülnek a Qt-hez szükséges könyvtárak, a `QApplication` illetve egy másik `.h` kiterjesztésű, amelyet mi hoztunk létre. A `QApplication` segítségével példányosítunk, létre hozásra kerül egy objektum, majd a konstruktor kerül meghívásra.

frakablak.h

```
#ifndef FRAKABLAH_H
#define FRAKABLAH_H

#include <QMainWindow>
#include <QImage>
#include <QPainter>
```

```
#include <QMouseEvent>
#include <QKeyEvent>
#include "frakszal.h"

class FrakSzal;

class FrakAblak : public QMainWindow
{
    Q_OBJECT

public:
    FrakAblak(double a = -2.0, double b = .7, double c = -1.35,
              double d = 1.35, int szelesseg = 600,
              int iteraciosHatar = 255, QWidget *parent = 0);
    ~FrakAblak();
    void vissza(int magassag , int * sor, int meret) ;
    void vissza(void) ;
    // A komplex sík vizsgált tartománya [a,b]x[c,d].
    double a, b, c, d;
    // A komplex sík vizsgált tartományára feszített
    // háló szélessége és magassága.
    int szelesseg, magassag;
    // Max. hány lépésig vizsgáljuk a  $z_{n+1} = z_n * z_n + c$  ←
    // iterációt?
    // (tk. most a nagyítási pontosság)
    int iteraciosHatar;

protected:
    void paintEvent (QPaintEvent*);
    void mousePressEvent (QMouseEvent*);
    void mouseMoveEvent (QMouseEvent*);
    void mouseReleaseEvent (QMouseEvent*);
    void keyPressEvent (QKeyEvent*);

private:
    QImage* fraktal;
    FrakSzal* mandelbrot;
    bool szamitasFut;
    // A nagyítandó kijelölt területet bal felső sarka.
    int x, y;
    // A nagyítandó kijelölt terület szélessége és magassága.
    int mx, my;
};

#endif // FRAKABLAK_H
```

Ez a header fájl, amely egyaránt tartalmaz public protected és private hozzáférésű részeket. A protected functionok dolgozzák fel a billentyűzetlenyomásokat és egérmozgatásokat- és kattintásokat. Ezek a függvények itt még csak deklarálva vannak, a definiálásra később kerül majd sor.

frakablak.cpp (részlet)

```
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) {

    // A nagyítandó kijelölt területet bal felső sarka:
    x = event->x();
    y = event->y();
    mx = 0;
    my = 0;

    update();
}

void FrakAblak::mouseMoveEvent(QMouseEvent* event) {

    // A nagyítandó kijelölt terület szélessége és magassága:
    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();

    }
```

Lényegében az előző header fájlban deklarált funkcionok definiálásár kerül sor ebben a fájlban.

frakszal.h

```
#ifndef FRAKSZAL_H
#define FRAKSZAL_H

#include <QThread>
#include <math.h>
#include "frakablak.h"

class FrakAblak;

class FrakSzal : public QThread
{
    Q_OBJECT

public:
    FrakSzal(double a, double b, double c, double d,
             int szelesseg, int magassag, int iteraciosHatar, FrakAblak ←
             *frakAblak);
    ~FrakSzal();
    void run();
};
```

```
protected:
    // A komplex sík vizsgált tartománya [a,b]x[c,d].
    double a, b, c, d;
    // A komplex sík vizsgált tartományára feszített
    // háló szélessége és magassága.
    int szelesseg, magassag;
    // Max. hány lépésig vizsgáljuk a  $z_{n+1} = z_n * z_n + c$  iterációt  $\leftrightarrow$ 
    // ?
    // (tk. most a nagyítási pontosság)
    int iteraciosHatar;
    // Kinek számolok?
    FrakAblak* frakAblak;
    // Soronként küldöm is neki vissza a kiszámoltakat.
    int* egySor;

};

#endif // FRAKSZAL_H
```

Ebben az osztályban kerül deklarálásra azok a változókat, amiket a számolás és a rajzolás során fogunk alkalmazni.

frakszal.cpp

```
#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()
{
    // A [a,b]x[c,d] tartományon milyen sűrű a
```



```
// megadott szélesség, magasság háló:
double dx = (b-a)/szelesseg;
double dy = (d-c)/magassag;
double reC, imC, reZ, imZ, ujreZ, ujimZ;
// Hány iterációt csináltunk?
int iteracio = 0;
// Végigzongorázzuk a szélesség x magasság hálót:
for(int j=0; j<magassag; ++j) {
    //sor = j;
    for(int k=0; k<szelesseg; ++k) {
        // c = (reC, imC) a háló rácspontjainak
        // 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

            ujreZ = reZ*reZ - imZ*imZ + reC;
            ujimZ = 2*reZ*imZ + imC;

            reZ = ujreZ;
            imZ = ujimZ;

            ++iteracio;
        }
        // ha a < 4 feltétel nem teljesült és a
        // iteráció < iterációsHatar sérülésével lépett ki, ←
        // azaz
        // feltesszük a c-ről, hogy itt a z_{n+1} = z_n * z_n ←
        // + c
        // sorozat konvergens, azaz iteráció = iterációsHatar
        // ekkor az iteráció %= 256 egyenlő 255, mert az ←
        // esetleges
        // nagyítások során az iteráció = valahány * 256 + ←
        // 255

        iteracio %= 256;
```

```
        //a színezést viszont már majd a FrakAblak osztályban ←  
        lesz  
        egySor[k] = iteracio;  
    }  
    // Ábrázolásra átadjuk a kiszámolt sort a FrakAblak-nak.  
    frakAblak->vissza(j, egySor, szelesseg);  
}  
frakAblak->vissza();  
  
}
```

A Mandelbrot halmaz kirajzolását segítő osztály, már megszokottan végigmegyünk a szélesség X magasság rácson minden pontra megnézve, eleme-e a Mandelbrot halmaznak.

Hogyan bírjuk működni a Qt-t?

5.6. Mandelbrot nagyító és utazó Java nyelven

Megoldás videó:

Megoldás forrása: <https://www.tankonyvtar.hu/hu/tartalom/tkt/javat-tanitok-javat/apbs02.html> - Mandelbrot halmaz nagyító programja https://gitlab.com/heinrichlaszlo/bhax/tree/codes/codes/mandelbrot/mandel_nagyito

A már előzőleg C++ nyelven megírt Mandelbrot-halmaz kirajzoló programunk továbbfejlesztett változatát, a Mandelbrot nagyító és utazót fogjuk megírni Java nyelven.

```
public class MandelbrotHalmazNagyító extends MandelbrotHalmaz {  
    /** A nagyítandó kijelölt területet bal felső sarka. */  
  
    private int x, y;  
    /** A nagyítandó kijelölt terület szélessége és magassága. */  
  
    private int mx, my;
```

Létrehozuk a MandelbrotHalmazNagyító osztályunkat aminek a neve után megtalálható az extends szó. Ez teszi lehetővé számunkra, hogy a program a MandelbrotHalmaz.java program változóit és függvényeit is tudja használni. Majd létre hozzuk a nagyítandó területet tároló, privát hozzáférésű változókat is.

```
public MandelbrotHalmazNagyító(double a, double b, double c, double ←  
    d,  
    int szélesség, int iterációsHatár) {  
  
    super(a, b, c, d, szélesség, iterációsHatár);  
    setTitle("A Mandelbrot halmaz nagyításai");  
  
    // Egér kattintó események feldolgozása:  
    addMouseListener(new java.awt.event.MouseAdapter() {  
        // Egér kattintással jelöljük ki a nagyítandó területet
```

```
// bal felső sarkát:
public void mousePressed(java.awt.event.MouseEvent m) {
    // A nagyítandó kijelölt területet bal felső sarka:
    x = m.getX();
    y = m.getY();
    mx = 0;
    my = 0;
    repaint();
}

public void mouseReleased(java.awt.event.MouseEvent m) {
    double dx = (MandelbrotHalmazNagyító.this.b
        - MandelbrotHalmazNagyító.this.a)
        /MandelbrotHalmazNagyító.this.szélesség;
    double dy = (MandelbrotHalmazNagyító.this.d
        - MandelbrotHalmazNagyító.this.c)
        /MandelbrotHalmazNagyító.this.magasság;

    // Az új Mandelbrot nagyító objektum elkészítése:
    new MandelbrotHalmazNagyító(MandelbrotHalmazNagyító.this.a+ ←
        x*dx,
        MandelbrotHalmazNagyító.this.a+x*dx+mx*dx,
        MandelbrotHalmazNagyító.this.d-y*dy-my*dy,
        MandelbrotHalmazNagyító.this.d-y*dy,
        600,
        MandelbrotHalmazNagyító.this.iterációsHatár);
}
});

// Egér mozgás események feldolgozása:
addMouseMotionListener(new java.awt.event.MouseMotionAdapter() {
    // Vonszolással jelöljük ki a négyzetet:
    public void mouseDragged(java.awt.event.MouseEvent m) {
        // A nagyítandó kijelölt terület szélessége és magassága:
        mx = m.getX() - x;
        my = m.getY() - y;
        repaint();
    }
});
}
```

Ebben a függvényben hívjuk meg az eredeti osztályban található konstruktort. A különböző egérműveleteket a megfelelő functionök kezelik, ilyen például a kattintás és az egérgomb felengedése. Az egérgomb felengedése esetén egy új MandelbrotHalmazNagyító objektum is létrehozásra kerül ugyanis ilyenkor lehet megtudni, mekkora területet akarunk nagyítani. Ezeken kívül nyomon követjük az egér mozgását. Az itt található repaint teszi lehetővé, hogy a halmazunk újra kirajzoltatható legyen a nagyítások elvégzése után.

```
public void pillanatfelvétel() {
```

```
// Az elmentendő kép elkészítése:

java.awt.image.BufferedImage mentKép =
    new java.awt.image.BufferedImage(szélesség, magasság,
        java.awt.image.BufferedImage.TYPE_INT_RGB);
java.awt.Graphics g = mentKép.getGraphics();
g.drawImage(kép, 0, 0, this);
g.setColor(java.awt.Color.BLUE);
g.drawString("a=" + a, 10, 15);
g.drawString("b=" + b, 10, 30);
g.drawString("c=" + c, 10, 45);
g.drawString("d=" + d, 10, 60);
g.drawString("n=" + iterációsHatár, 10, 75);
if(számításFut) {
    g.setColor(java.awt.Color.RED);
    g.drawLine(0, sor, getWidth(), sor);
}
g.setColor(java.awt.Color.GREEN);
g.drawRect(x, y, mx, my);
g.dispose();
// A pillanatfelvétel képfájl nevének képzése:
StringBuffer sb = new StringBuffer();
sb = sb.delete(0, sb.length());
sb.append("MandelbrotHalmazNagyitas_");
sb.append(++pillanatfelvételSzámláló);
sb.append("_");
// A fájl nevébe bele vesszük, hogy melyik tartományban
// találtuk a halmazt:

sb.append(a);
sb.append("_");
sb.append(b);
sb.append("_");
sb.append(c);
sb.append("_");
sb.append(d);
sb.append(".png");
// png formátumú képet mentünk

try {
    javax.imageio.ImageIO.write(mentKép, "png",
        new java.io.File(sb.toString()));
} catch (java.io.IOException e) {
    e.printStackTrace();
}
}
```

Itt jön létre az üres képünk és a megfelelő képpontok színezése.

```
public void paint(java.awt.Graphics g) {
    // A Mandelbrot halmaz kirajzolása
```

```
g.drawImage(kép, 0, 0, this);  
// Ha éppen fut a számítás, akkor egy vörös  
// vonallal jelöljük, hogy melyik sorban tart:  
if(számításFut) {  
    g.setColor(java.awt.Color.RED);  
    g.drawLine(0, sor, getWidth(), sor);  
}  
// A jelző négyzet kirajzolása:  
g.setColor(java.awt.Color.GREEN);  
g.drawRect(x, y, mx, my);  
}
```

Megtörténik a keret kirajzoltatása amit a nagyításkor használunk.

```
public static void main(String[] args) {  
    // A kiinduló halmazt a komplex sík [-2.0, .7]x[-1.35, 1.35]  
    // tartományában keressük egy 600x600-as hálóval és az  
    // aktuális nagyítási pontossággal:  
    new MandelbrotHalmazNagyító(-2.0, .7, -1.35, 1.35, 600, 255);  
}
```

A main-ben sor kerül példányosításra.

Fordítjuk és futtatjuk:

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 elspájzolod és egy további logikai taggal az osztályban jelzed, hogy van vagy nincs eltéve kiszámolt szám.

Megoldás forrása: <https://www.tankonyvtar.hu/hu/tartalom/tkt/javat-tanitok-javat/ch01.html/>

Megoldás forrása: <https://www.tankonyvtar.hu/hu/tartalom/tkt/javat-tanitok-javat/ch01.html/>

```
#include <iostream>
#include <cstdlib>
#include <cmath>
#include <ctime>

using namespace std;

class Random
{
public:
    Random();
    ~Random() {}

    double get();
private:
    bool exist;
    double value;
};
```

Deklaráljuk a Random osztályt, a publikus részben az osztály konstruktora és destruktora és az a függvény ami random számokat fog lekérni kap helyet, míg a privát részben azt fogjuk tárolni, hogy létezik-e korábban kiszámolt másik random és ennek a kiszámolt randomnak az értéke.

```
Random::Random()
{
    exist = false;
    srand (time(NULL));
};
```

A konstruktorban inicializáljuk a randomszám generáló srand függvényt.

```
double Random::get()
{
    if (!exist)
    {
        double u1, u2, v1, v2, w;

        do
        {
            u1 = rand () / (RAND_MAX + 1.0);
            u2 = rand () / (RAND_MAX + 1.0);
            v1 = 2 * u1 - 1;
            v2 = 2 * u2 - 1;
            w = v1 * v1 + v2 * v2;
        }
        while (w > 1);

        double r = sqrt ((-2 * log (w)) / w);

        value = r * v2;
        exist = !exist;

        return r * v1;
    }

    else
    {
        exist = !exist;
        return value;
    }
};
```

Lekérő függvény. Abban az esetben , ha nincs eltárolva random számunk, akkor létrehozunk két számot. Az elsőt ki fogjuk írni, a másodikat eltároljuk. Ellenkező esetben az eltárolt számot kiíratjuk.

```
int main()
{

    Random rnd;
```

```
for (int i = 0; i < 2; ++i) cout << rnd.get() << endl;

}
```

A main függvényben meghívjuk a random osztály lehívó függvényét.

Polártranszformációs algoritmus JAVA:

```
public class PolárTranszF {

    boolean létezik_tárolt = false;
    double tárolt;

    public PolárTranszF() {

        létezik_tárolt = false;

    }

}
```

A publikus PolárTranszF osztályunkban deklaráljuk a logikai tagot : van-e korábban eltárolt random, és ha van mi annak az értéke.

```
public double matek_rész() {

    if(!létezik_tárolt) {

        double u1, u2, v1, v2, w;
        do {
            u1 = Math.random();
            u2 = Math.random();

            v1 = 2*u1 - 1;
            v2 = 2*u2 - 1;

            w = v1*v1 + v2*v2;

        } while(w > 1);

        double r = Math.sqrt((-2*Math.log(w))/w);

        tárolt = r*v2;
        létezik_tárolt = !létezik_tárolt;

        return r*v1;

    } else {
        létezik_tárolt = !létezik_tárolt;
        return tárolt;
    }

}
```


Ha nincs korábban eltárolt random értékünk, akkor meghívjuk a misztikus random-generáló matematikai eljárást. Ellenkező esetben az értékét adjuk vissza és a logikai tagot átállítjuk.

```
public static void main(String[] args) {  
  
    PolárTranszF g = new PolárTranszF();  
  
    for(int i=0; i<2; ++i)  
        System.out.println(g.matek_rész());  
  
}  
  
}
```

Két hívás következik : Ki iratjuk a kiszámolt számok egyikét, második hívásnál pedig a kiszámolt, majd elmentett értéket adjuk vissza.

6.2. LZW

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

Tutoriált : Kincs Ákos <https://gitlab.com/kincsakos>

Megoldás forrása : https://gitlab.com/heinrichlaszlo/bhax/blob/codes/codes/welch/binfa_z.c

Az Lempel–Ziv–Welch (LZW) algoritmust az 1980-as évek közepén, Abraham Lempel és Jacob Ziv már létező algoritmusát továbbfejlesztve, Terry Welch publikálta. Az algoritmus a UNIX alapú rendszerek fájl-tömörítő segédprogramja által terjed el leginkább, továbbá GIF kiterjesztésű képek és PDF fájlok veszteségmentes tömörítéséhez is használják. Az USA-ban 2003-ban, a világ többi részén 2004-ben az algoritmus szabadalma lejárt, ezért azóta alkalmazása a háttérbe szorult. Forrás : [Wikipédia](#)

A bináris fát úgy építjük fel, hogy az input nullákat és egyeseket beolvassuk. Ha olyat olvasunk be, amivel már találkoztunk , akkor olvassuk tovább. Az így kapott új egységet fogjuk a fa gyökerétől kezdve kiírni. Ha az adott egység ábrázolva van, visszaugrunk a gyökérbe és olvassuk tovább az inputot.

```
#include <stdio.h>  
#include <stdlib.h>  
#include <unistd.h>  
  
typedef struct binfa  
{  
    int ertek;  
    struct binfa *bal_nulla;  
    struct binfa *jobb_egy;  
}  
BINFA, *BINFA_PTR;
```

Definiáljuk a binfa struktúrát.

```
BINFA_PTR  
uj_elem ()
```

```
{
    BINFA_PTR p;

    if ((p = (BINFA_PTR) malloc (sizeof (BINFA))) == NULL)
    {
        perror ("memoria");
        exit (EXIT_FAILURE);
    }
    return p;
}
```

A új elem létrehozásakor memóriát szabadítunk majd fel. Ha hiba lép fel akkor kilépünk a programból.

```
extern void kiir (BINFA_PTR elem);
extern void szabadit (BINFA_PTR elem);
```

Deklaráljuk, de még nem definiáljuk a fa kiírásához és a lefoglalt memória felszabadításához használt függvényeket. (Egyelőre nem foglalunk le nekik helyet a memóriában.)

```
int
main (int argc, char **argv)
{
    char b;

    BINFA_PTR gyoker = uj_elem ();
    gyoker->ertek = '/';
    BINFA_PTR fa = gyoker;

    while (read (0, (void *) &b, 1))
    {
        write (1, &b, 1);
        if (b == '0')
        {
            if (fa->bal_nulla == NULL)
            {
                fa->bal_nulla = uj_elem ();
                fa->bal_nulla->ertek = 0;
                fa->bal_nulla->bal_nulla = fa->bal_nulla->jobb_egy = NULL;
                fa = gyoker;
            }
            else
            {
                fa = fa->bal_nulla;
            }
        }
        else
        {
            if (fa->jobb_egy == NULL)
            {
                fa->jobb_egy = uj_elem ();
                fa->jobb_egy->ertek = 1;
            }
        }
    }
}
```

```
        fa->jobb_egy->bal_nulla = fa->jobb_egy->jobb_egy = NULL;
        fa = gyoker;
    }
    else
    {
        fa = fa->jobb_egy;
    }
}

printf ("\n");
kiir (gyoker);
extern int max_melyseg;
printf ("melyseg=%d\n", max_melyseg);
szabadit (gyoker);
}
```

Először létrehozunk a gyökeret és ráállítjuk a fa pointert. Olvassuk az inputon érkező 0-kat és 1-eseket.

- Ha 0-t olvasunk és az aktuális nodenak nincs nullás gyermeke, akkor az `uj_elem` függvénnyel létrehozunk neki egyet, majd megkapja értékül a 0-t. Ezután ennek az újonnan létrehozott gyermeknek beállítjuk a bal és jobb gyermekeit NULL pointerre és visszaugrunk a pointerrel a gyökérre. Ha a beolvasáskor már volt nullás gyermeke az aktuális nodenak, akkor a mutatót ráállítjuk.
- Ha 1-t olvasunk és az aktuális nodenak nincs egyes gyermeke, akkor az `uj_elem` függvénnyel létrehozunk neki egyet, majd megkapja értékül az 1-t. Ezután ennek az újonnan létrehozott gyermeknek beállítjuk a bal és jobb gyermekeit NULL pointerre és visszaugrunk a pointerrel a gyökérre. Ha a beolvasáskor már volt egyes gyermeke az aktuális nodenak, akkor a mutatót ráállítjuk.

```
static int melyseg = 0;
int max_melyseg = 0;

void
kiir (BINFA_PTR elem)
{
    if (elem != NULL)
    {
        ++melyseg;
        if (melyseg > max_melyseg)
            max_melyseg = melyseg;
        kiir (elem->jobb_egy);
        for (int i = 0; i < melyseg; ++i)
            printf ("---");
        printf ("%c(%d)\n", elem->ertek < 2 ? '0' + elem->ertek : elem->ertek,
            melyseg);
        kiir (elem->bal_nulla);
        --melyseg;
    }
}
```

A kiír függvénnyel jelenítjük meg magát a fát a parancssorban. Mivel 90 fokkal el van forgatva az eredmény balra és fentről lefelé írjuk ki az ágakat ezért inorder bejárás esetén először a jobb oldali ágat, majd a gyökeret, majd végül a bal oldali ágat rajzoltatjuk ki.

```
void
szabadit (BINFA_PTR elem)
{
    if (elem != NULL)
    {
        szabadit (elem->jobb_egy);
        szabadit (elem->bal_nulla);
        free (elem);
    }
}
```

Rekurzívan hívjuk a függvényt. Ezzel a lépéssel felszabadítjuk azokat az elemeket amiket korábban lefoglaltunk.

Fordítjuk majd futtatjuk:

```
heinrichlaszlo@heinrichlaszlo-VirtualBox:~/Asztal/welch_codes$ gcc binfa_z.c -o binfa -lm
heinrichlaszlo@heinrichlaszlo-VirtualBox:~/Asztal/welch_codes$ ./binfa <bemenet.txt> out.txt
heinrichlaszlo@heinrichlaszlo-VirtualBox:~/Asztal/welch_codes$ cat out.txt

-----1(2)
-----1(1)
-----1(4)
-----1(3)
-----0(2)
-----0(3)
---/(0)
-----1(2)
-----0(1)
-----0(2)
melyseg=4
altag=2.600000
szoras=0.894427
heinrichlaszlo@heinrichlaszlo-VirtualBox:~/Asztal/welch_codes$
```

6.3. Fabejárás

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

Megoldás forrása:

- https://gitlab.com/heinrichlaszlo/bhax/blob/codes/codes/welch/binfa_z_pre.c
- https://gitlab.com/heinrichlaszlo/bhax/blob/codes/codes/welch/binfa_z_post.c

Példa Inorder bejárásra:

```
static int melyseg = 0;
int max_melyseg = 0;

void
kiir (BINFA_PTR elem)
```

```
{
    if (elem != NULL)
    {
        ++melyseg;
        if (melyseg > max_melyseg)
            max_melyseg = melyseg;
        kiir (elem->jobb_egy);
        for (int i = 0; i < melyseg; ++i)
            printf ("---");
        printf ("%c(%d)\n", elem->ertek < 2 ? '0' + elem->ertek : elem->ertek,
            melyseg);
        kiir (elem->bal_nulla);
        --melyseg;
    }
}
```

A preorder bejárás : a gyökeret dolgozzuk fel, majd a bal oldali részfát és végül a jobb oldali részfát .

Példa Preorder bejárásra:

```
void
kiir (BINFA_PTR elem)
{
    if (elem != NULL)
    {
        ++melyseg;
        if (melyseg > max_melyseg)
            max_melyseg = melyseg;

        kiir (elem->jobb_egy);
        kiir (elem->bal_nulla);
        for (int i = 0; i < melyseg; ++i)
            printf ("---");
        printf ("%c(%d)\n", elem->ertek < 2 ? '0' + elem->ertek : elem->ertek,
            melyseg);

        --melyseg;
    }
}
```

Fordítjuk majd futtatjuk:

```

heinchlaszlo@heinchlaszlo-VirtualBox:~/Asztal/welch_codes$ gcc binfa_z_pre.c -o pre -lm
heinchlaszlo@heinchlaszlo-VirtualBox:~/Asztal/welch_codes$ ./pre <bemenet.txt> preki.txt
heinchlaszlo@heinchlaszlo-VirtualBox:~/Asztal/welch_codes$ cat preki.txt

---/(1)
-----0(2)
-----0(3)
-----1(3)
-----1(2)
-----0(3)
-----0(4)
-----1(4)
-----1(5)
-----1(3)
melyseg=4
altag=2.600000
szoras=0.894427

```

A postorder bejárás : bal oldali részfát majd a jobb oldali részfát majd a gyökeret dolgozzuk fel.

```

void
kiir (BINFA_PTR elem)
{
    if (elem != NULL)
    {
        ++melyseg;
        if (melyseg > max_melyseg)
            max_melyseg = melyseg;

        for (int i = 0; i < melyseg; ++i)
            printf ("---");
        printf ("%c(%d)\n", elem->ertek < 2 ? '0' + elem->ertek : elem->ertek ->
            , melyseg);
        kiir (elem->jobb_egy);
        kiir (elem->bal_nulla);
        --melyseg;
    }
}

```

Fordítjuk majd futtatjuk:

```

heinchlaszlo@heinchlaszlo-VirtualBox:~/Asztal/welch_codes$ gcc binfa_z_post.c -o post -lm
heinchlaszlo@heinchlaszlo-VirtualBox:~/Asztal/welch_codes$ ./post <bemenet.txt> postki.txt
heinchlaszlo@heinchlaszlo-VirtualBox:~/Asztal/welch_codes$ cat postki.txt

-----0(3)
-----1(3)
-----0(2)
-----0(4)
-----1(5)
-----1(4)
-----0(3)
-----1(3)
-----1(2)
---/(1)
melyseg=4
altag=2.600000
szoras=0.894427

```

6.4. Tag a gyökér

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

Megoldás forrása:

- <https://gitlab.com/heinrichlaszlo/bhax/blob/codes/codes/welch/z3a7.cpp>

```
#include <iostream>

class LZWTree
{
public:
    LZWTree () : fa(&gyoker){}

    ~LZWTree ()
    {
        szabadit (gyoker.egyesGyermekek ());
        szabadit (gyoker.nullasGyermekek ());
    }

    void operator<<(char b)
    {
        if (b == '0')
        {
            if (!fa->nullasGyermekek ())
            {
                Node *uj = new Node ('0');
                fa->ujNullasGyermekek (uj);
                fa = &gyoker;
            }
            else
            {
                fa = fa->nullasGyermekek ();
            }
        }
        else
        {
            if (!fa->egyesGyermekek ())
            {
                Node *uj = new Node ('1');
                fa->ujEgyesGyermekek (uj);
                fa = &gyoker;
            }
            else
            {

```

```

        fa = fa->egyenesGyermekek ();
    }
}
void kiir (void)
{
    melyseg = 0;
    kiir (&gyoker);
}
void szabadit (void)
{
    szabadit (gyoker.jobbEgy);
    szabadit (gyoker.balNulla);
}

```

A fa konstruktora és destruktora után az LZWTree osztályon belül definiáljuk a balra eltoló bitshift operátort . Ez fogja az inputról érkező karaktereket eltolni egészen a LZWTree objektumba. Így fog felépülni a fa.

private:

```

class Node
{
public:
    Node (char b = '/') : betu (b), balNulla (0), jobbEgy (0) {};
    ~Node () {};
    Node *nullasGyermekek () {
        return balNulla;
    }
    Node *egyenesGyermekek ()
    {
        return jobbEgy;
    }
    void ujNullasGyermekek (Node * gy)
    {
        balNulla = gy;
    }
    void ujEgyenesGyermekek (Node * gy)
    {
        jobbEgy = gy;
    }

private:
    friend class LZWTree;
    char betu;
    Node *balNulla;
    Node *jobbEgy;
    Node (const Node &);
    Node & operator=(const Node &);
};

```


Ha a Node konstruktor paraméter nélküli, akkor az alapértelmezett '/'-jellel fogja azt létrehozni. Egyébként a meghívó karakter kerül a betű helyére. A bal és jobb gyermekekre mutató mutatókat nulára állítjuk majd az aktuális Node mindig meg tudja mondani, hogy mi a bal illetve jobb gyermeke. Deklaráljuk az LZWTree osztályt a csomópontok kezelésére.

```
Node gyoker;
Node *fa;
int melyseg;

LZWTree (const LZWTree &);
LZWTree & operator=(const LZWTree &);

void kiir (Node* elem)
{
    if (elem != NULL)
    {
        ++melyseg;
        kiir (elem->jobbEgy);

        for (int i = 0; i < melyseg; ++i)
            std::cout << "---";
        std::cout << elem->betu << "(" << melyseg - 1 << ")" << std::endl;
        kiir (elem->balNulla);
        --melyseg;
    }
}

void szabadit (Node * elem)
{
    if (elem != NULL)
    {
        szabadit (elem->jobbEgy);
        szabadit (elem->balNulla);
        delete elem;
    }
}

};
```

Mindig az aktuális csomópontra mutatunk. A ír függvénnyel jelenítjük meg magát a fát.

```
int
main ()
{
    char b;
    LZWTree binFa;

    while (std::cin >> b)
    {
        binFa << b;
    }
}
```

```
    binFa.kiir ();  
    binFa.szabadit ();  
  
    return 0;  
}
```

Bitenként olvassuk a bemenetet.

Fordítjuk majd futtatjuk:

```
heinchlaszlo@heinchlaszlo-VirtualBox:~/Asztal/welch_codes$ g++ z3a7.cpp -o z3a7  
heinchlaszlo@heinchlaszlo-VirtualBox:~/Asztal/welch_codes$ g++ z3a7.cpp -o fa  
heinchlaszlo@heinchlaszlo-VirtualBox:~/Asztal/welch_codes$ ./fa bemenet.txt -o faki.txt  
heinchlaszlo@heinchlaszlo-VirtualBox:~/Asztal/welch_codes$ cat faki.txt  
-----1(2)  
-----0(3)  
-----0(4)  
-----1(1)  
-----0(2)  
-----1(4)  
-----0(3)  
-----1(5)  
-----0(4)  
-----0(5)  
---/(0)  
-----1(3)  
-----0(4)  
-----0(5)  
-----1(2)  
-----0(3)  
-----1(6)  
-----1(5)  
-----0(4)  
-----0(1)  
-----1(4)  
-----0(5)  
-----0(6)  
-----0(7)  
-----1(3)  
-----0(4)  
-----1(6)  
-----0(5)  
-----0(2)  
-----1(5)  
-----1(4)  
-----0(3)  
-----1(5)  
-----0(4)  
-----0(5)  
-----1(7)  
-----0(6)  
depth = 7  
mean = 5.36364  
var = 1.02691
```

6.5. Mutató a gyökér

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

Megoldás forrása: https://gitlab.com/heinchlaszlo/bhax/blob/codes/codes/welch/z3a7_new.cpp

Eddig a LZWTree-ben a fa gyökere egy objektum volt. Mostantól a gyökér is és a fa is egy-egy mutató lesz. Tehát a fa mutatónkak a gyökér mutatót adjuk.

```
class LZWTree
{
public:
    LZWTree ()
    {
        gyoker = new Node();
        fa = gyoker;
    }

    ~LZWTree ()
    {
        szabadit (gyoker->egyesGyermekek ());
        szabadit (gyoker->nullasGyermekek ());
        delete gyoker;
    }
}
```

A csomópontokra mutató mutatóként létrehozuk a gyökeret, a destruktorban fel szabadítjuk . Az eddig a gyökér előtt álló referenciajeleket mindenhol kitöröljük.

```
Node *gyoker;
Node *fa;
int melyseg;
```

Ahol eddig objektumként szerepelt ott átírjuk pointerre.

Fordítjuk majd futtatjuk:

```
heinchlaszlo@heinchlaszlo-VirtualBox:~/Asztal/welch_codes$ g++ z3a7_new.cpp -o fanew
heinchlaszlo@heinchlaszlo-VirtualBox:~/Asztal/welch_codes$ ./fanew bemenet.txt -o fanewki.txt
heinchlaszlo@heinchlaszlo-VirtualBox:~/Asztal/welch_codes$ cat fanewki.txt
-----1(2)
-----0(3)
-----0(4)
-----1(1)
-----0(2)
-----1(4)
-----0(3)
-----1(5)
-----0(4)
-----0(5)
---/(0)
-----1(3)
-----0(4)
-----0(5)
-----1(2)
-----0(3)
-----1(6)
-----1(5)
-----0(4)
-----0(1)
-----1(4)
-----0(5)
-----0(6)
-----0(7)
-----1(3)
-----0(4)
-----1(6)
-----0(5)
-----0(2)
-----1(5)
-----1(4)
-----0(3)
-----1(5)
-----0(4)
-----0(5)
-----1(7)
-----0(6)
depth = 7
mean = 5.36364
var = 1.02691
```

6.6. Mozgató szemantika

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

Megoldás forrása: https://gitlab.com/heinchlaszlo/bhax/blob/codes/codes/welch/z3a7_mozgato.cpp

```
LZWTree& operator= (LZWTree& copy)
{
    szabadit(gyoker->egyenesGyermekek ());
    szabadit(gyoker->nullasGyermekek ());

    bejar(gyoker, copy.gyoker);

    fa = gyoker;
    copy.fas = copy.gyoker;

}
```

Törölöm a régi értékeket majd bejárom a fát és átmásolom az értékeket, ezek után mindkét fában visszaugrok a gyökérhez.

```
void bejar (Node * masolat, Node * eredeti)
{
    if (eredeti != NULL)
    {
        if ( !eredeti->>nullasGyermek() )
        {
            masolat->ujNullasGyermek(NULL);
        }
        else
        {
            Node* uj = new Node ('0');
            masolat->ujNullasGyermek (uj);
            bejar(masolat->>nullasGyermek(), eredeti->>nullasGyermek());
        }

        if ( !eredeti->egyenesGyermek() )
        {
            masolat->ujEgyenesGyermek(NULL);
        }
        else
        {
            Node *uj = new Node ('1');
            masolat->ujEgyenesGyermek (uj);
            bejar(masolat->egyenesGyermek(), eredeti->egyenesGyermek());
        }
    }
    else
    {
        masolat = NULL;
    }
}
```

Fordítjuk majd futtatjuk:

```
heinrichlaszlo@heinrichlaszlo-VirtualBox:~/Asztal/welch_codes$ g++ z3a7_mozgato.cpp -o famozg
heinrichlaszlo@heinrichlaszlo-VirtualBox:~/Asztal/welch_codes$ ./famozg bemenet.txt -o famozgki.txt
LZWBinFa mozgato ctor
LZWBinFa mozgato ctor
LZWBinFa mozgato ctor
heinrichlaszlo@heinrichlaszlo-VirtualBox:~/Asztal/welch_codes$ cat famozgki.txt
-----1(2)
-----0(3)
-----0(4)
-----1(1)
-----0(2)
-----1(4)
-----0(3)
-----1(5)
-----0(4)
-----0(5)
---/(0)
-----1(3)
-----0(4)
-----0(5)
-----1(2)
-----0(3)
-----1(6)
-----1(5)
-----0(4)
-----0(1)
-----1(4)
-----0(5)
-----0(6)
-----0(7)
-----1(3)
-----0(4)
-----1(6)
-----0(5)
-----0(2)
-----1(5)
-----1(4)
-----0(3)
-----1(5)
-----0(4)
-----0(5)
-----1(7)
-----0(6)
depth = 7
mean = 5.36364
var = 1.02691
```

Utóirat : a std::move semmit nem mozgat.

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: <https://gitlab.com/heinrichlaszlo/bhax/tree/codes/codes/conway/myrmecologist>

Kezdjük elsősorban azzal h miért is érdekesek a hangyák ? A hangyák feromon nevezetű kémiai anyagot bocsátanak ki. A feromon hatására a kiszemelt hangya az anyagot kibocsátó ellentétes nemű hangya iránt vonzalmat kezd érezni. A feromon leginkább a párválasztásban játszik fontos szerepet. Merül fel bennünk a kérdés tehát , hogy a hangyák feromon kibocsátása hogyan befolyásolja a többi hangya mozgását , illetve a többi hangya elhelyezkedését és viselkedését. Logikusan belegondolva , a hangyák azokhoz a szomszédjaikhoz fognak közel kerülni akiknek a feromonszintje a legmagasabb. Nos a mi programunk is ezt hivatott bemutatni.

A main.cpp

```
#include <QApplication>
#include <QDesktopWidget>
#include <QDebug>
#include <QDateTime>
#include <QCommandLineOption>
#include <QCommandLineParser>

#include "antwin.h"

/*
 *
 * ./myrmecologist -w 250 -m 150 -n 400 -t 10 -p 5 -f 80 -d 0 -a ↵
 * 255 -i 3 -s 3 -c 22
 *
 */

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

```
{

    QApplication a ( argc, argv );

    QCommandLineOption szeles_opt ( {"w","szelesseg"}, "Oszlopok ( ←  
cellakban) szama.", "szelesseg", "200" );
    QCommandLineOption magas_opt ( {"m","magassag"}, "Sorok ( ←  
cellakban) szama.", "magassag", "150" );
    QCommandLineOption hangyaszam_opt ( {"n","hangyaszam"}, " ←  
Hangyak szama.", "hangyaszam", "100" );
    QCommandLineOption sebesseg_opt ( {"t","sebesseg"}, "2 lepes ←  
kozotti ido (millisec-ben).", "sebesseg", "100" );
    QCommandLineOption parolgas_opt ( {"p","parolgas"}, "A parolgas ←  
erteke.", "parolgas", "8" );
    QCommandLineOption feromon_opt ( {"f","feromon"}, "A hagyott ←  
nyom erteke.", "feromon", "11" );
    QCommandLineOption szomszed_opt ( {"s","szomszed"}, "A hagyott ←  
nyom erteke a szomszedokban.", "szomszed", "3" );
    QCommandLineOption alapertek_opt ( {"d","alapertek"}, "Indulo ←  
ertek a cellakban.", "alapertek", "1" );
    QCommandLineOption maxcella_opt ( {"a","maxcella"}, "Cella max ←  
erteke.", "maxcella", "50" );
    QCommandLineOption mincella_opt ( {"i","mincella"}, "Cella min ←  
erteke.", "mincella", "2" );
    QCommandLineOption cellamerete_opt ( {"c","cellameret"}, "Hany ←  
hangya fer egy cellaba.", "cellameret", "4" );
    QCommandLineParser parser;

    parser.addHelpOption();
    parser.addVersionOption();
    parser.addOption ( szeles_opt );
    parser.addOption ( magas_opt );
    parser.addOption ( hangyaszam_opt );
    parser.addOption ( sebesseg_opt );
    parser.addOption ( parolgas_opt );
    parser.addOption ( feromon_opt );
    parser.addOption ( szomszed_opt );
    parser.addOption ( alapertek_opt );
    parser.addOption ( maxcella_opt );
    parser.addOption ( mincella_opt );
    parser.addOption ( cellamerete_opt );

    parser.process ( a );

    QString szeles = parser.value ( szeles_opt );
    QString magas = parser.value ( magas_opt );
    QString n = parser.value ( hangyaszam_opt );
    QString t = parser.value ( sebesseg_opt );
    QString parolgas = parser.value ( parolgas_opt );
    QString feromon = parser.value ( feromon_opt );
```



```

        QString szomszed = parser.value ( szomszed_opt );
        QString alapertek = parser.value ( alapertek_opt );
        QString maxcella = parser.value ( maxcella_opt );
        QString mincella = parser.value ( mincella_opt );
        QString cellameret = parser.value ( cellamerete_opt );

        qsrand ( QDateTime::currentMSecsSinceEpoch() );

        AntWin w ( szeles.toInt(), magas.toInt(), t.toInt(), n.toInt(), ←
                    feromon.toInt(), szomszed.toInt(), parolgas.toInt(),
                    alapertek.toInt(), mincella.toInt(), maxcella. ←
                    toInt(),
                    cellameret.toInt() );

        w.show();

        return a.exec();
    }

```

Szokásos módon includoljuk a szükséges könyvtárakat , jelenleg ez a Qt-s és az antwin.h könyvtárakat. Majd a mainben példányosítjuk a QApplication objektumot , illetve megadjuk és feldolgozzuk a futtatáskor szükséges kapcsolókat.

A ant.h

```

#ifndef ANT_H
#define ANT_H

class Ant
{
public:
    int x;
    int y;
    int dir;

    Ant(int x, int y): x(x), y(y) {

        dir = qrand() % 8;

    }

};

typedef std::vector<Ant> Ants;

#endif

```

Először is vegyük a grand függvényt , ennek a függvénynek a segítségével állítjuk be a hangya irányát , ami véletlenszerű. A hangya mozgásának leírására hivatott változók mind publikusak. Abból a célból , hogy

megkönnyítsük a magunk dolgát Typedef -et használunk , ami annyit tesz , hogy azokra a vektorokra amik Ant-okat tartalmaznak a későbbiekben Ants néven fogunk hivatkozni.

Az antthread.h

```
#ifndef ANTTHREAD_H
#define ANTTHREAD_H

#include <QThread>
#include "ant.h"

class AntThread : public QThread
{
    Q_OBJECT

public:
    AntThread(Ants * ants, int ***grids, int width, int height,
              int delay, int numAnts, int pheromone, int ←
              nbrPheromone,
              int evaporation, int min, int max, int cellAntMax);

    ~AntThread();

    void run();
    void finish()
    {
        running = false;
    }

    void pause()
    {
        paused = !paused;
    }

    bool isRunnung()
    {
        return running;
    }

private:
    bool running {true};
    bool paused {false};
    Ants* ants;
    int** numAntsinCells;
    int min, max;
    int cellAntMax;
    int pheromone;
    int evaporation;
    int nbrPheromone;
    int ***grids;
    int width;
```

```
int height;
int gridIdx;
int delay;

void timeDevel();

int newDir(int sor, int oszlop, int vsor, int voszlop);
void detDirs(int irány, int& ifrom, int& ito, int& jfrom, int& ←
    jto );
int moveAnts(int **grid, int row, int col, int& retrow, int& ←
    retcol, int);
double sumNbhs(int **grid, int row, int col, int);
void setPheromone(int **grid, int row, int col);

signals:
    void step ( const int &);

};

#endif
```

Deklaráljuk azokat a függvényeket amiknek feladata az ablak képét megállítani , illetve a szimuláció befejezése. Ebben a kódrészletben deklaráljuk a konstruktort és a destruktort is. Továbbá létrehozunk azokat a privát változókat amik az úgynevezett "rács" vagy "háló"-ra vonatkoznak. Ezek például a : min, max , width , height , running , pause.

Az antthread.h

```
#include "antthread.h"
#include <QDebug>
#include <cmath>
#include <QDateTime>

AntThread::AntThread ( Ants* ants, int*** grids,
    int width, int height,
    int delay, int numAnts,
    int pheromone, int nbrPheromone,
    int evaporation,
    int min, int max, int cellAntMax)
{
    this->ants = ants;
    this->grids = grids;
    this->width = width;
    this->height = height;
    this->delay = delay;
    this->pheromone = pheromone;
    this->evaporation = evaporation;
    this->min = min;
    this->max = max;
    this->cellAntMax = cellAntMax;
    this->nbrPheromone = nbrPheromone;
```

```
numAntsinCells = new int*[height];
for ( int i=0; i<height; ++i ) {
    numAntsinCells[i] = new int [width];
}

for ( int i=0; i<height; ++i )
    for ( int j=0; j<width; ++j ) {
        numAntsinCells[i][j] = 0;
    }

qsrand ( QDateTime::currentMSecsSinceEpoch() );

Ant h {0, 0};
for ( int i {0}; i<numAnts; ++i ) {

    h.y = height/2 + qrand() % 40-20;
    h.x = width/2 + qrand() % 40-20;

    ++numAntsinCells[h.y][h.x];

    ants->push_back ( h );

}

gridIdx = 0;
}

double AntThread::sumNbhs ( int **grid, int row, int col, int ←
dir )
{
    double sum = 0.0;

    int ifrom, ito;
    int jfrom, jto;

    detDirs ( dir, ifrom, ito, jfrom, jto );

    for ( int i=ifrom; i<ito; ++i )
        for ( int j=jfrom; j<jto; ++j )

            if ( ! ( ( i==0 ) && ( j==0 ) ) ) {
                int o = col + j;
                if ( o < 0 ) {
                    o = width-1;
                } else if ( o >= width ) {
                    o = 0;
                }

                int s = row + i;
```

```
        if ( s < 0 ) {
            s = height-1;
        } else if ( s >= height ) {
            s = 0;
        }

        sum += (grid[s][o]+1)*(grid[s][o]+1)*(grid[s][o] ←
            ]+1);

    }

    return sum;
}

int AntThread::newDir ( int sor, int oszlop, int vsor, int ←
    voszlop )
{

    if ( vsor == 0 && sor == height -1 ) {
        if ( voszlop < oszlop ) {
            return 5;
        } else if ( voszlop > oszlop ) {
            return 3;
        } else {
            return 4;
        }
    } else if ( vsor == height - 1 && sor == 0 ) {
        if ( voszlop < oszlop ) {
            return 7;
        } else if ( voszlop > oszlop ) {
            return 1;
        } else {
            return 0;
        }
    } else if ( voszlop == 0 && oszlop == width - 1 ) {
        if ( vsor < sor ) {
            return 1;
        } else if ( vsor > sor ) {
            return 3;
        } else {
            return 2;
        }
    } else if ( voszlop == width && oszlop == 0 ) {
        if ( vsor < sor ) {
            return 7;
        } else if ( vsor > sor ) {
            return 5;
        } else {
            return 6;
        }
    }
}
```

```
    } else if ( vsor < sor && voszlop < oszlop ) {
        return 7;
    } else if ( vsor < sor && voszlop == oszlop ) {
        return 0;
    } else if ( vsor < sor && voszlop > oszlop ) {
        return 1;
    }

    else if ( vsor > sor && voszlop < oszlop ) {
        return 5;
    } else if ( vsor > sor && voszlop == oszlop ) {
        return 4;
    } else if ( vsor > sor && voszlop > oszlop ) {
        return 3;
    }

    else if ( vsor == sor && voszlop < oszlop ) {
        return 6;
    } else if ( vsor == sor && voszlop > oszlop ) {
        return 2;
    }

    else { //(vsor == sor && voszlop == oszlop)
        qDebug() << "ZAVAR AZ EROBEN az iranynal";

        return -1;
    }
}

void AntThread::detDirs ( int dir, int& ifrom, int& ito, int& ←
    jfrom, int& jto )
{

    switch ( dir ) {
    case 0:
        ifrom = -1;
        ito = 0;
        jfrom = -1;
        jto = 2;
        break;
    case 1:
        ifrom = -1;
        ito = 1;
        jfrom = 0;
        jto = 2;
        break;
    case 2:
        ifrom = -1;
        ito = 2;
```

```
        jfrom = 1;
        jto = 2;
        break;
    case 3:
        ifrom =
            0;
        ito = 2;
        jfrom = 0;
        jto = 2;
        break;
    case 4:
        ifrom = 1;
        ito = 2;
        jfrom = -1;
        jto = 2;
        break;
    case 5:
        ifrom = 0;
        ito = 2;
        jfrom = -1;
        jto = 1;
        break;
    case 6:
        ifrom = -1;
        ito = 2;
        jfrom = -1;
        jto = 0;
        break;
    case 7:
        ifrom = -1;
        ito = 1;
        jfrom = -1;
        jto = 1;
        break;
    }

}

int AntThread::moveAnts ( int **racs,
                          int sor, int oszlop,
                          int& vsor, int& voszlop, int dir )
{
    int y = sor;
    int x = oszlop;

    int ifrom, ito;
    int jfrom, jto;
```

```
detDirs ( dir, ifrom, ito, jfrom, jto );

double osszes = sumNbhs ( racs, sor, oszlop, dir );
double random = ( double ) ( grand() %1000000 ) / ( double ←
    ) 1000000.0;
double gvalseg = 0.0;

for ( int i=ifrom; i<ito; ++i )
    for ( int j=jfrom; j<jto; ++j )
        if ( ! ( ( i==0 ) && ( j==0 ) ) )
        {
            int o = oszlop + j;
            if ( o < 0 ) {
                o = width-1;
            } else if ( o >= width ) {
                o = 0;
            }

            int s = sor + i;
            if ( s < 0 ) {
                s = height-1;
            } else if ( s >= height ) {
                s = 0;
            }

            //double kedvezo = std::sqrt((double)(racs[s][o ←
                ]+2)); //(racs[s][o]+2)*(racs[s][o]+2);
            //double kedvezo = (racs[s][o]+b)*(racs[s][o]+b ←
                );
            //double kedvezo = ( racs[s][o]+1 );
            double kedvezo = (racs[s][o]+1)*(racs[s][o]+1) ←
                *(racs[s][o]+1);

            double valseg = kedvezo/osszes;
            gvalseg += valseg;

            if ( gvalseg >= random ) {

                vsor = s;
                voszlop = o;

                return newDir ( sor, oszlop, vsor, voszlop ←
                    );

            }

        }

}

qDebug() << "ZAVAR AZ EROBEN a lepesnel";
```



```
        vsor = y;
        voszlop = x;

        return dir;
    }

    void AntThread::timeDevel()
    {

        int **racsElotte = grids[gridIdx];
        int **racsUtana = grids[ ( gridIdx+1 ) %2];

        for ( int i=0; i<height; ++i )
            for ( int j=0; j<width; ++j )
            {
                racsUtana[i][j] = racsElotte[i][j];

                if ( racsUtana[i][j] - evaporation >= 0 ) {
                    racsUtana[i][j] -= evaporation;
                } else {
                    racsUtana[i][j] = 0;
                }
            }

        for ( Ant &h: *ants )
        {

            int sor {-1}, oszlop {-1};
            int ujirany = moveAnts( racsElotte, h.y, h.x, sor, ←
                oszlop, h.dir );

            setPheromone ( racsUtana, h.y, h.x );

            if ( numAntsinCells[sor][oszlop] <cellAntMax ) {

                --numAntsinCells[h.y][h.x];
                ++numAntsinCells[sor][oszlop];

                h.x = oszlop;
                h.y = sor;
                h.dir = ujirany;
            }
        }

        gridIdx = ( gridIdx+1 ) %2;
    }
}
```

```
void AntThread::setPheromone ( int **racs,
                               int sor, int oszlop )
{
    for ( int i=-1; i<2; ++i )
        for ( int j=-1; j<2; ++j )
            if ( ! ( ( i==0 ) && ( j==0 ) ) )
            {
                int o = oszlop + j;
                {
                    if ( o < 0 ) {
                        o = width-1;
                    } else if ( o >= width ) {
                        o = 0;
                    }
                }
                int s = sor + i;
                {
                    if ( s < 0 ) {
                        s = height-1;
                    } else if ( s >= height ) {
                        s = 0;
                    }
                }

                if ( racs[s][o] + nbrPheromone <= max ) {
                    racs[s][o] += nbrPheromone;
                } else {
                    racs[s][o] = max;
                }
            }

    if ( racs[sor][oszlop] + pheromone <= max ) {
        racs[sor][oszlop] += pheromone;
    } else {
        racs[sor][oszlop] = max;
    }
}

void AntThread::run()
{
    running = true;
    while ( running ) {

        QThread::msleep ( delay );
    }
}
```

```
        if ( !paused ) {
            timeDevel();
        }

        emit step ( gridIdx );

    }

}

AntThread::~AntThread()
{
    for ( int i=0; i<height; ++i ) {
        delete [] numAntsinCells[i];
    }

    delete [] numAntsinCells;
}
```

Ebben a kódrészletben röviden annyi történik , hogy definiáljuk azokat a függvényeket amelyeket már az előző header fájlban deklaráltunk.

Az antwin.h

```
##ifndef ANTWIN_H
#define ANTWIN_H

#include <QMainWindow>
#include <QPainter>
#include <QString>
#include <QCloseEvent>
#include "antthread.h"
#include "ant.h"

class AntWin : public QMainWindow
{
    Q_OBJECT

public:
    AntWin(int width = 100, int height = 75,
          int delay = 120, int numAnts = 100,
          int pheromone = 10, int nbhPheromon = 3,
          int evaporation = 2, int cellDef = 1,
          int min = 2, int max = 50,
          int cellAntMax = 4, QWidget *parent = 0);

    AntThread* antThread;

    void closeEvent ( QCloseEvent *event ) {
```

```

        antThread->finish();
        antThread->wait();
        event->accept();
    }

    void keyPressEvent ( QKeyEvent *event )
    {

        if ( event->key() == Qt::Key_P ) {
            antThread->pause();
        } else if ( event->key() == Qt::Key_Q
                    || event->key() == Qt::Key_Escape ) {
            close();
        }

    }

    virtual ~AntWin();
    void paintEvent(QPaintEvent*);

private:

    int ***grids;
    int **grid;
    int gridIdx;
    int cellWidth;
    int cellHeight;
    int width;
    int height;
    int max;
    int min;
    Ants* ants;

public slots :
    void step ( const int &);

};

#endif

```

Deklaráljuk a `keyPressEvent` függvényt ami a billentyűzetről érkező includokat kezeli , emellett deklaráljuk az ablak bezárását kezelő `closeEvent` függvényt

Az `antwin.cpp`

```

#include "antwin.h"
#include <QDebug>

AntWin::AntWin ( int width, int height, int delay, int numAnts,
                 int pheromone, int nbhPheromon, int evaporation, ←

```

```
        int cellDef,
        int min, int max, int cellAntMax, QWidget *parent ←
        ) : QMainWindow ( parent )
{
    setWindowTitle ( "Ant Simulation" );

    this->width = width;
    this->height = height;
    this->max = max;
    this->min = min;

    cellWidth = 6;
    cellHeight = 6;

    setFixedSize ( QSize ( width*cellWidth, height*cellHeight ) );

    grids = new int**[2];
    grids[0] = new int*[height];
    for ( int i=0; i<height; ++i ) {
        grids[0][i] = new int [width];
    }
    grids[1] = new int*[height];
    for ( int i=0; i<height; ++i ) {
        grids[1][i] = new int [width];
    }

    gridIdx = 0;
    grid = grids[gridIdx];

    for ( int i=0; i<height; ++i )
        for ( int j=0; j<width; ++j ) {
            grid[i][j] = cellDef;
        }

    ants = new Ants();

    antThread = new AntThread ( ants, grids, width, height, delay, ←
        numAnts, pheromone,
                                nbhPheromon, evaporation, min, max, ←
                                cellAntMax);

    connect ( antThread, SIGNAL ( step ( int) ),
              this, SLOT ( step ( int) ) );

    antThread->start();
}

void AntWin::paintEvent ( QPaintEvent* )
{
```

```
QPainter qpainter ( this );

grid = grids[gridIdx];

for ( int i=0; i<height; ++i ) {
    for ( int j=0; j<width; ++j ) {

        double rel = 255.0/max;

        qpainter.fillRect ( j*cellWidth, i*cellHeight,
                               cellWidth, cellHeight,
                               QColor ( 255 - grid[i][j]*rel,
                                         255,
                                         255 - grid[i][j]*rel) );

        if ( grid[i][j] != min )
        {
            qpainter.setPen (
                QPen (
                    QColor ( 255 - grid[i][j]*rel,
                            255 - grid[i][j]*rel, 255),
                    1 )
            );

            qpainter.drawRect ( j*cellWidth, i*cellHeight,
                                cellWidth, cellHeight );
        }

        qpainter.setPen (
            QPen (
                QColor (0,0,0 ),
                1 )
        );

        qpainter.drawRect ( j*cellWidth, i*cellHeight,
                            cellWidth, cellHeight );

    }
}

for ( auto h: *ants) {
    qpainter.setPen ( QPen ( Qt::black, 1 ) );

    qpainter.drawRect ( h.x*cellWidth+1, h.y*cellHeight+1,
                        cellWidth-2, cellHeight-2 );
}
```

```
        QPainter.end();
    }

    AntWin::~AntWin()
    {
        delete antThread;

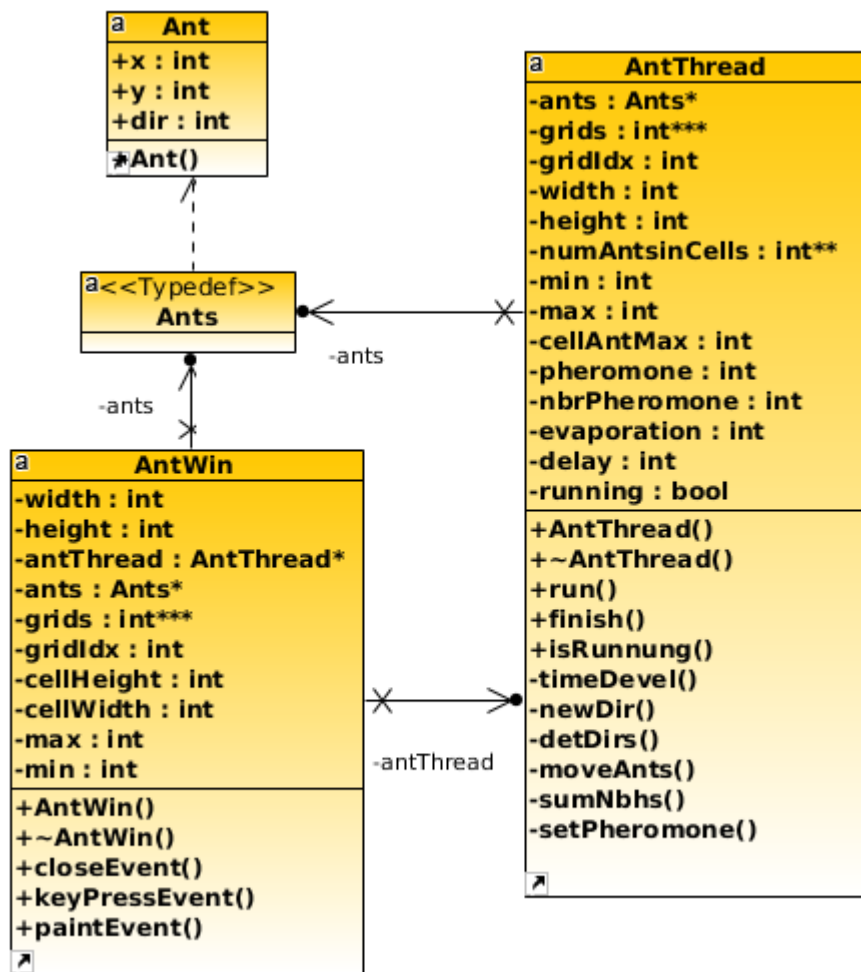
        for ( int i=0; i<height; ++i ) {
            delete[] grids[0][i];
            delete[] grids[1][i];
        }

        delete[] grids[0];
        delete[] grids[1];
        delete[] grids;

        delete ants;
    }

    void AntWin::step ( const int &gridIdx )
    {
        this->gridIdx = gridIdx;
        update();
    }
```

Definiáljuk a headerben includolt könyvtárakat. Definiáljuk a konstruktort és a destruktort. Értéket adunk a "cella" vagy "rács" magasságának és szélességének, majd a főprogramban megtörténik a hangyák kirajzoltatása.



Az UML Unified Modeling Language , magyarul Egységes Modellező Nyelvnek nevezzük. A programozásban a legelterjedt módja a programunk osztályainak és az osztályok közötti kapcsolatainak ábrázolására.

Mit kell tudni az UML osztálydiagramról? Az osztályok reprezentálása történik benne. A fenti képen minden sárga cella egy-egy osztály. Minden osztálynak vannak tulajdonságai és emellett viselkedése. A tulajdonságokat a változóknak és a viselkedéseket a függvényeknek nevezzük. A kettő közötti elválasztó vonal is be van jelölve a diagramunkon minden egyes osztály esetében, a vonal felett találhatók meg a változók míg alatta a függvények helyezkednek el.

Szembevetendő dolog még a + vagy - jel a változók vagy függvények előtt : A + jel azt jelenti , hogy az adott változó vagy függvény `public` , tehát más osztályokból is hozzá tudunk férni. A - jel `private` változókat vagy függvényeket jelöl , amikre csak az osztályon belül hivatkozhatunk.

7.2. Java életjáték

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

Megoldás videó:

Megoldás forrása: https://gitlab.com/heinrichlaszlo/bhax/tree/codes/codes/conway/sejtauto_java

Tanulságok, tapasztalatok, magyarázat...

Az életjáték John Horton Conway, angol egyetemi matematikus nevével vált egyggyé. Mi is az életjáték? Egy olyan játék ahol a játékos feladata, hogy megad egy kiiunduló alakzatot majd várja az eredményt. A játék alapját egy négyzetrácsos ablak adja meg, mezőket celláknak, az itt megtalálható korongokat pedig sejteknek nevezzük. A sejt környezetének a hozzá legközelebb eső 8 mezőt tekintjük. A sejt szomszédjai a környezetében található sejtek.

A játék elején a játékosnak lehetősége van cellákba sejteket helyezni.

A játék körökre van osztva. Egy sejttel a különböző körökben különböző dolgok történhetnek, melyek a következők:

A sejt túléli a kört akkor, ha pontosan 2 vagy pontosan 3 élő szomszédja van.

A sejt elpusztul akkor, ha 2-nél kevesebb vagy 3-nál több élő szomszédja van.

Új sejt is születhet akkor, ha az adott cella környezetében pontosan 3 sejt található.

Több, egymás mellett található sejt alakzatot alkot. Az egyik ilyen alakzat a sikló. A sikló átlósan mozog mindig egy-egy kockányit miközben változtatja formáját, mozgása végén visszatér a kiindulási formájába.

```
public class Sejtautomata extends java.awt.Frame implements Runnable {  
    /** Egy sejt lehet élő */  
    public static final boolean ÉLŐ = true;  
    /** vagy halott */  
    public static final boolean HALOTT = false;  
    /** Két rácsot használunk majd, az egyik a sejttér állapotát  
     * a t_n, a másik a t_n+1 időpillanatban jellemzi. */  
    protected boolean [][][] rácsok = new boolean [2][][];  
    /** Valamelyik rácsra mutat, technikai jellegű, hogy ne kelljen  
     * a  
     * [2][][]-ból az első dimenziót használni, mert vagy az  
     * egyikre  
     * állítjuk, vagy a másikra. */  
    protected boolean [][] rács;  
    /** Megmutatja melyik rács az aktuális: [rácsIndex][][] */  
    protected int rácsIndex = 0;  
    /** Pixelben egy cella adatai. */  
    protected int cellaSzélesség = 20;  
    protected int cellaMagasság = 20;  
    /** A sejttér nagysága, azaz hányszor hány cella van? */  
    protected int szélesség = 20;  
    protected int magasság = 10;  
    /** A sejttér két egymást követő t_n és t_n+1 diszkrét id  
     * őpillanata  
     * közötti valós idő. */  
    protected int várakozás = 1000;  
    // Pillanatfelvétel készítéséhez  
    private java.awt.Robot robot;  
    /** Készítsünk pillanatfelvételt? */  
    private boolean pillanatfelvétel = false;  
    /** A pillanatfelvételek számozásához. */
```

```
private static int pillanatfelvételSzámláló = 0;
```

A Sejtautomata osztályban az osztály a `java.awt.Frame` osztályt használja a keret létrehozásához vagyis a grafikus megjelenítéshez. Létrehozásra kerülnek a sejtek körönkénti állapotát jelző változók, a cella adatokat tároló változók, egy logikai változó ami azt az információt tárolja, hogy készítettünk-e már pillanatképet vagy sem. Majd létrehozásra kerül a rács.

```
public Sejtautomata(int szélesség, int magasság) {
    this.szélesség = szélesség;
    this.magasság = magasság;
    // A két rács elkészítése
    rácsok[0] = new boolean[magasság][szélesség];
    rácsok[1] = new boolean[magasság][szélesség];
    rácsIndex = 0;
    rács = rácsok[rácsIndex];
    // A kiinduló rács minden cellája HALOTT
    for(int i=0; i<rács.length; ++i)
        for(int j=0; j<rács[0].length; ++j)
            rács[i][j] = HALOTT;
    // A kiinduló rácsra "élőlényeket" helyezünk
    //sikló(rács, 2, 2);
    siklóKilövő(rács, 5, 60);
    // Az ablak bezárásakor kilépünk a programból.
    addWindowListener(new java.awt.event.WindowAdapter() {
        public void windowClosing(java.awt.event.WindowEvent e) {
            setVisible(false);
            System.exit(0);
        }
    });
};
```

Létrehozunk egy `Sejtautomata` objektumot. Esetünkben ez a `Sejtautomata` objektum a konstruktor. A cellák állapotát beállítjuk, a játék kezdetén minden cella halott. Egy for ciklus segítségével végig megyünk sorokon és oszlopokon. Létre hozunk egy functiont, ami azt hivatott figyelni, hogy mikor zárjuk be az ablakot. Ha ez megtörténik, kilép a programból.

```
addKeyListener(new java.awt.event.KeyAdapter() {
    // Az 'k', 'n', 'l', 'g' és 's' gombok lenyomását figyeljük
    public void keyPressed(java.awt.event.KeyEvent e) {
        if(e.getKeyCode() == java.awt.event.KeyEvent.VK_K) {
            // Felezzük a cella méreteit:
            cellaSzélesség /= 2;
            cellaMagasság /= 2;
            setSize(Sejtautomata.this.szélesség*cellaSzélesség,
                Sejtautomata.this.magasság*cellaMagasság);
            validate();
        } else if(e.getKeyCode() == java.awt.event.KeyEvent.VK_N) {
            // Duplázzuk a cella méreteit:
            cellaSzélesség *= 2;
            cellaMagasság *= 2;
            setSize(Sejtautomata.this.szélesség*cellaSzélesség,
                Sejtautomata.this.magasság*cellaMagasság);
        }
    }
});
```

```
        validate();
    } else if(e.getKeyCode() == java.awt.event.KeyEvent.VK_S)
        pillanatfelvétel = !pillanatfelvétel;
    else if(e.getKeyCode() == java.awt.event.KeyEvent.VK_G)
        várakozás /= 2;
    else if(e.getKeyCode() == java.awt.event.KeyEvent.VK_L)
        várakozás *= 2;
    repaint();
}
});

// Egér kattintó események feldolgozása:
addMouseListener(new java.awt.event.MouseAdapter() {
    // Egér kattintással jelöljük ki a nagyítandó területet
    // bal felső sarkát vagy ugyancsak egér kattintással
    // vizsgáljuk egy adott pont iterációit:
    public void mousePressed(java.awt.event.MouseEvent m) {
        // Az egérmutató pozíciója
        int x = m.getX()/cellaSzélesség;
        int y = m.getY()/cellaMagasság;
        rácsok[rácsIndex][y][x] = !rácsok[rácsIndex][y][x];
        repaint();
    }
});

// Egér mozgás események feldolgozása:
addMouseMotionListener(new java.awt.event.MouseMotionAdapter() {
    // Vonszolással jelöljük ki a négyzetet:
    public void mouseDragged(java.awt.event.MouseEvent m) {
        int x = m.getX()/cellaSzélesség;
        int y = m.getY()/cellaMagasság;
        rácsok[rácsIndex][y][x] = ÉLŐ;
        repaint();
    }
});

// Cellaméretek kezdetben
cellaSzélesség = 10;
cellaMagasság = 10;
// Pillanatfelvétel készítéséhez:
try {
    robot = new java.awt.Robot(
        java.awt.GraphicsEnvironment.
            getLocalGraphicsEnvironment().
            getDefaultScreenDevice());
} catch(java.awt.AWTException e) {
    e.printStackTrace();
}
```

```
// A program ablakának adatai:
setTitle("Sejtautomata");
setResizable(false);
setSize(szélesség*cellaSzélesség,
        magasság*cellaMagasság);
setVisible(true);
// A sejttér életrekeltése:
new Thread(this).start();
}
/** A sejttér kirajzolása. */
public void paint(java.awt.Graphics g) {
    // Az aktuális
    boolean [][] rács = rácsok[rácsIndex];
    // rácsot rajzoljuk ki:
    for(int i=0; i<rács.length; ++i) { // végig lépked a sorokon
        for(int j=0; j<rács[0].length; ++j) { // s az oszlopok
            // Sejt cella kirajzolása
            if(rács[i][j] == ÉLŐ)
                g.setColor(java.awt.Color.BLACK);
            else
                g.setColor(java.awt.Color.WHITE);
            g.fillRect(j*cellaSzélesség, i*cellaMagasság,
                      cellaSzélesség, cellaMagasság);
            // Rács kirajzolása
            g.setColor(java.awt.Color.LIGHT_GRAY);
            g.drawRect(j*cellaSzélesség, i*cellaMagasság,
                      cellaSzélesség, cellaMagasság);
        }
    }

    // Készítünk pillanatfelvételt?
    if(pillanatfelvétel) {
        // a biztonság kedvéért egy kép készítése után
        // kikapcsoljuk a pillanatfelvételt, hogy a
        // programmal ismerkedő Olvasó ne írja tele a
        // fájlrendszerét a pillanatfelvételekkel
        pillanatfelvétel = false;
        pillanatfelvétel(robot.createScreenCapture
            (new java.awt.Rectangle
             (getLocation().x, getLocation().y,
              szélesség*cellaSzélesség,
              magasság*cellaMagasság)));
    }
}
```

Nyomon követjük az egérről és a billentyűzetről érkező inputokat. Majd Beállításra kerülnek a cellaméretek és a programablakok . Majd a new szóval létrehozunk egy új Thread-et, amit a későbbiekben a start function-nel el is indítunk. A paint függvényen a "sejttér" kerül megrajzolásra. A Java awt.Graphics G osztálya teszi lehetővé a sejttér megrajzolását. A rajzoláshoz a fekete , fehér és a szürke színeket hasz-

náljuk. A megrajzolás után megvizsgáljuk, hogy a rács jelenlegi állapotáról készült -e pillanatfelvétel.

```
public int szomszédokSzama(boolean [][] rács,
    int sor, int oszlop, boolean állapot) {
    int állapotúSzomszéd = 0;
    // A nyolcszomszédok végigzongorázása:
    for(int i=-1; i<2; ++i)
        for(int j=-1; j<2; ++j)
            // A vizsgált sejtet magát kihagyva:
            if(!((i==0) && (j==0))) {
                // A sejtteréből szélének szomszédai
                // a szembe oldalakon ("periódikus határfeltétel")
                int o = oszlop + j;
                if(o < 0)
                    o = szélesség-1;
                else if(o >= szélesség)
                    o = 0;

                int s = sor + i;
                if(s < 0)
                    s = magasság-1;
                else if(s >= magasság)
                    s = 0;

                if(rács[s][o] == állapot)
                    ++állapotúSzomszéd;
            }

    return állapotúSzomszéd;
}
```

Megvizsgáljuk az adott sejt 8 szomszédjának az állapotát , mert ez befolyásolja , hogy a sejtünk életben marad, meghal , megszületik-e.Legvégül a függvény visszaadja a szomszédok számát

```
public void időFejlődés() {

    boolean [][] rácsElőtte = rácsok[rácsIndex];
    boolean [][] rácsUtána = rácsok[(rácsIndex+1)%2];

    for(int i=0; i<rácsElőtte.length; ++i) { // sorok
        for(int j=0; j<rácsElőtte[0].length; ++j) { // oszlopok

            int élők = szomszédokSzama(rácsElőtte, i, j, ÉLŐ);

            if(rácsElőtte[i][j] == ÉLŐ) {
                /* Élő élő marad, ha kettő vagy három élő
                szomszedja van, különben halott lesz. */
                if(élők==2 || élők==3)
                    rácsUtána[i][j] = ÉLŐ;
                else
                    rácsUtána[i][j] = HALOTT;
            }
        }
    }
}
```

```
        } else {
            /* Halott halott marad, ha három élő
               szomszédja van, különben élő lesz. */
            if(élők==3)
                rácsUtána[i][j] = ÉLŐ;
            else
                rácsUtána[i][j] = HALOTT;
        }
    }
}
rácsIndex = (rácsIndex+1)%2;
}
/** A sejtter időbeli fejlődése. */
public void run() {

    while(true) {
        try {
            Thread.sleep(várakozás);
        } catch (InterruptedException e) {}

        időFejlődés();
        repaint();
    }
}
```

Megvizsgáljuk, hogy élő vagy halott az adott sejt, esetleg éppen megszületik-e. A `run` függvényben egy végtelen ciklus foglal helyet aminek eredménye képpen a programunk addig fut amíg mi kívülről meg nem szakítjuk.

```
public void sikló(boolean [][] rács, int x, int y) {

    rács[y+ 0][x+ 2] = ÉLŐ;
    rács[y+ 1][x+ 1] = ÉLŐ;
    rács[y+ 2][x+ 1] = ÉLŐ;
    rács[y+ 2][x+ 2] = ÉLŐ;
    rács[y+ 2][x+ 3] = ÉLŐ;

}

public void siklóKilövő(boolean [][] rács, int x, int y) {

    rács[y+ 6][x+ 0] = ÉLŐ;
    rács[y+ 6][x+ 1] = ÉLŐ;
    rács[y+ 7][x+ 0] = ÉLŐ;
    rács[y+ 7][x+ 1] = ÉLŐ;

    rács[y+ 3][x+ 13] = ÉLŐ;

    rács[y+ 4][x+ 12] = ÉLŐ;
```

```
rács[y+ 4][x+ 14] = ÉLŐ;

rács[y+ 5][x+ 11] = ÉLŐ;
rács[y+ 5][x+ 15] = ÉLŐ;
rács[y+ 5][x+ 16] = ÉLŐ;
rács[y+ 5][x+ 25] = ÉLŐ;

rács[y+ 6][x+ 11] = ÉLŐ;
rács[y+ 6][x+ 15] = ÉLŐ;
rács[y+ 6][x+ 16] = ÉLŐ;
rács[y+ 6][x+ 22] = ÉLŐ;
rács[y+ 6][x+ 23] = ÉLŐ;
rács[y+ 6][x+ 24] = ÉLŐ;
rács[y+ 6][x+ 25] = ÉLŐ;

rács[y+ 7][x+ 11] = ÉLŐ;
rács[y+ 7][x+ 15] = ÉLŐ;
rács[y+ 7][x+ 16] = ÉLŐ;
rács[y+ 7][x+ 21] = ÉLŐ;
rács[y+ 7][x+ 22] = ÉLŐ;
rács[y+ 7][x+ 23] = ÉLŐ;
rács[y+ 7][x+ 24] = ÉLŐ;

rács[y+ 8][x+ 12] = ÉLŐ;
rács[y+ 8][x+ 14] = ÉLŐ;
rács[y+ 8][x+ 21] = ÉLŐ;
rács[y+ 8][x+ 24] = ÉLŐ;
rács[y+ 8][x+ 34] = ÉLŐ;
rács[y+ 8][x+ 35] = ÉLŐ;

rács[y+ 9][x+ 13] = ÉLŐ;
rács[y+ 9][x+ 21] = ÉLŐ;
rács[y+ 9][x+ 22] = ÉLŐ;
rács[y+ 9][x+ 23] = ÉLŐ;
rács[y+ 9][x+ 24] = ÉLŐ;
rács[y+ 9][x+ 34] = ÉLŐ;
rács[y+ 9][x+ 35] = ÉLŐ;

rács[y+ 10][x+ 22] = ÉLŐ;
rács[y+ 10][x+ 23] = ÉLŐ;
rács[y+ 10][x+ 24] = ÉLŐ;
rács[y+ 10][x+ 25] = ÉLŐ;

rács[y+ 11][x+ 25] = ÉLŐ;

}
```

```
/** Pillanatfelvételek készítése. */
public void pillanatfelvétel(java.awt.image.BufferedImage felvetel) {
    // A pillanatfelvétel kép fájlneve
```

```
StringBuffer sb = new StringBuffer();
sb = sb.delete(0, sb.length());
sb.append("sejtautomata");
sb.append(++pillanatfelvetelSzamlalo);
sb.append(".png");
// png formátumú képet mentünk
try {
    javax.imageio.ImageIO.write(felvetel, "png",
        new java.io.File(sb.toString()));
} catch (java.io.IOException e) {
    e.printStackTrace();
}
}
// Ne villogjon a felület (mert a "gyári" update()
// lemeszelné a vászon felületét).
public void update(java.awt.Graphics g) {
    paint(g);
}
/**
 * Példányosít egy Conway-féle életjáték szabályos
 * sejtter obektumot.
 */
public static void main(String[] args) {
    // 100 oszlop, 75 sor mérettel:
    new Sejtautomata(100, 75);
}
}
```

A pillanatfelvétel függvény az S billentyű lenyomására hívódik meg. Aktiválásakor egy png kiterjesztésű képállományt hoz létre, ezen elkészült képállományok számát is eltároljuk egy változóban.

A main függvényben történik a példányosítás, létrehozunk egy objektumot 100 és 75 paraméterekkel ezek jelölik az ablak szélességet és magasságot.

Fordítás és futtatás után:

7.3. Qt C++ életjáték

Most Qt C++-ban!

Megoldás videó:

https://gitlab.com/heinrichlaszlo/bhax/tree/codes/codes/conway/sejtauto-qt_ej

Tanulságok, tapasztalatok, magyarázat...

A main.cpp-ben:

```
#include <QApplication>
#include "sejtablak.h"
#include <QDesktopWidget>
```



```
int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    SejtAblak w(100, 75);
    w.show();

    return a.exec();
}
```

Inkludáljuk azokat az osztályokat amik a Qt-vel való munkához szükségünk lesz. A `QApplication` segítségével fogunk példányosítani. Létrehozunk egy objektumot. Majd meghívásra kerül a konstruktor is itt adjuk meg az ablak méreteit is.

A `sejtablak.h`-ban:

```
#ifndef SEJTABLAK_H
#define SEJTABLAK_H

#include <QMainWindow>
#include <QPainter>
#include "sejtszal.h"

class SejtSzal;

class SejtAblak : public QMainWindow
{
    Q_OBJECT

public:
    SejtAblak(int szelesseg = 100, int magassag = 75, QWidget *parent ←
        = 0);

    ~SejtAblak();
    // Egy sejt lehet élő
    static const bool ELO = true;
    // vagy halott
    static const bool HALOTT = false;
    void vissza(int racsIndex);

protected:
    // Két rácsot használunk majd, az egyik a sejttér állapotát
    // a t_n, a másik a t_n+1 időpillanatban jellemzi.
    bool **racsok;
    // Valamelyik rácsra mutat, technikai jellegű, hogy ne kelljen a
    // [2][][]-ból az első dimenziót használni, mert vagy az egyikre
    // állítjuk, vagy a másikra.
    bool **racs;
    // Megmutatja melyik rács az aktuális: [racsIndex][][]
    int racsIndex;
    // Pixelben egy cella adatai.
```

```
int cellaSzelesseg;
int cellaMagassag;
// A sejtter nagysága, azaz hányszor hány cella van?
int szelesseg;
int magassag;
void paintEvent(QPaintEvent*);
void siklo(bool **racs, int x, int y);
void sikloKilovo(bool **racs, int x, int y);

private:
    SejtSzal* eletjatek;

};

#endif // SEJTABLAK_H
```

Létrehozzuk SejtAblak konstruktorát és destruktort. Deklarálunk számos functiont és a cella magassági és szélességi adatait tároló int típusú változókat.

A sejtablak.cpp-ben:

```
void SejtAblak::paintEvent(QPaintEvent*) {
    QPainter qpainter(this);
    // Az aktuális
    bool **racs = racsok[racsIndex];
    // racsot rajzoljuk ki:
    for(int i=0; i<magassag; ++i) { // végig lépked a sorokon
        for(int j=0; j<szelesseg; ++j) { // s az oszlopok
            // Sejt cella kirajzolása
            if(racs[i][j] == ELO)
                qpainter.fillRect(j*cellaSzelesseg, i*cellaMagassag,
                                   cellaSzelesseg, cellaMagassag, Qt:: ←
                                   black);
            else
                qpainter.fillRect(j*cellaSzelesseg, i*cellaMagassag,
                                   cellaSzelesseg, cellaMagassag, Qt:: ←
                                   white);
            qpainter.setPen(QPen(Qt::gray, 1));

            qpainter.drawRect(j*cellaSzelesseg, i*cellaMagassag,
                               cellaSzelesseg, cellaMagassag);
        }
    }

    qpainter.end();
}

SejtAblak::~SejtAblak()
{
}
```

```
delete eletjatek;

for(int i=0; i<magassag; ++i) {
    delete[] racsok[0][i];
    delete[] racsok[1][i];
}

delete[] racsok[0];
delete[] racsok[1];
delete[] racsok;

...
...

void SejtAblak::siklo(bool **racs, int x, int y) {

racs[y+ 0][x+ 2] = ELO;
racs[y+ 1][x+ 1] = ELO;
racs[y+ 2][x+ 1] = ELO;
racs[y+ 2][x+ 2] = ELO;
racs[y+ 2][x+ 3] = ELO;

}

void SejtAblak::sikloKilovo(bool **racs, int x, int y) {

racs[y+ 6][x+ 0] = ELO;
racs[y+ 6][x+ 1] = ELO;
racs[y+ 7][x+ 0] = ELO;
racs[y+ 7][x+ 1] = ELO;

}
```

Itt hívunk meg sok-sok functiont illetve itt definiáljuk a destruktort.

A sejt.szal.h-ban:

```
#ifndef SEJTSZAL_H
#define SEJTSZAL_H

#include <QThread>
#include "sejtablak.h"

class SejtAblak;

class SejtSzal : public QThread
{
    Q_OBJECT

public:
    SejtSzal(bool ***racsok, int szelesseg, int magassag,
```

```

        int varakozas, SejtAblak *sejtAblak);
~SejtSzal();
void run();

protected:
    bool ***racsek;
    int szelesseg, magassag;
    // Megmutatja melyik rács az aktuális: [rácsIndex][][ ]
    int racsIndex;
    // A sejtter két egymást követő t_n és t_n+1 diszkrét ←
    időpillanata
    // közötti valós idő.
    int varakozas;
    void idoFejlodes();
    int szomszedokSzama(bool ***racsek,
                        int sor, int oszlop, bool allapot);
    SejtAblak* sejtAblak;

};

#endif // SEJTSZAL_H

```

Egy header fájl kerül deklarálásra, amiben inkludáljuk a másik, header fájlunkat a fejezet elején említett céllal. Ebben a fájlban kapnak helyet public és protected változók, function-ök, a konstruktor és a destruktor deklarálása is.

A sejtsszal.cpp-ben:

```

#include "sejtsszal.h"

SejtSzal::SejtSzal(bool ***racsek, int szelesseg, int magassag, int ←
    varakozas, SejtAblak *sejtAblak)
{
    this->racsek = racsek;
    this->szelesseg = szelesseg;
    this->magassag = magassag;
    this->varakozas = varakozas;
    this->sejtAblak = sejtAblak;

    racsIndex = 0;
}

int SejtSzal::szomszedokSzama(bool ***racsek,
                                int sor, int oszlop, bool allapot) {
    int allapotuSzomszed = 0;
    // A nyolcszomszedok végigzongorázása:
    for(int i=-1; i<2; ++i)
        for(int j=-1; j<2; ++j)
            // A vizsgált sejtet magát kihagyva:
            if(!((i==0) && (j==0))) {

```

```
// A sejttérből szélének szomszédai
// a szembe oldalakon ("periódikus határfeltétel")
int o = oszlop + j;
if(o < 0)
    o = szelesseg-1;
else if(o >= szelesseg)
    o = 0;

int s = sor + i;
if(s < 0)
    s = magassag-1;
else if(s >= magassag)
    s = 0;

if(racs[s][o] == allapot)
    ++allapotuSzomszed;
}

return allapotuSzomszed;
}

void SejtSzal::idoFejlodes() {

    bool **racsElotte = racsok[racsIndex];
    bool **racsUtana = racsok[(racsIndex+1)%2];

    for(int i=0; i<magassag; ++i) { // sorok
        for(int j=0; j<szelesseg; ++j) { // oszlopok

            int elok = szomszedokSzama(racsElotte, i, j, SejtAblak ←
::ELO);

            if(racsElotte[i][j] == SejtAblak::ELO) {
                /* Élő élő marad, ha kettő vagy három élő
                szomszedja van, különben halott lesz. */
                if(elok==2 || elok==3)
                    racsUtana[i][j] = SejtAblak::ELO;
                else
                    racsUtana[i][j] = SejtAblak::HALOTT;
            } else {
                /* Halott halott marad, ha három élő
                szomszedja van, különben élő lesz. */
                if(elok==3)
                    racsUtana[i][j] = SejtAblak::ELO;
                else
                    racsUtana[i][j] = SejtAblak::HALOTT;
            }
        }
    }

    racsIndex = (racsIndex+1)%2;
```

```
    }

    /** A sejttér időbeli fejlődése. */
    void SejtSzal::run()
    {
        while(true) {
            QThread::msleep(varakozas);
            idoFejlodes();
            sejtAblak->vissza(racsIndex);
        }
    }

    SejtSzal::~SejtSzal()
    {
    }
}
```

Definiáljuk az `idoFejlodes` functiont amivel megvizsgáljuk , hogy az adott sejt , élő-e vagy halott , vagy éppen megszületik-e. Létrehozzuk a `A SejtSzal` konstruktort ami a sejt szomszédjainak számát határozza meg.

Ahogy az előző feladatban itt is megismerkedünk a `A run` function-nel , benne egy végtelen `while` ciklussal , ami lehetővé teszi a program megszakításig történő futását.

7.4. BrainB Benchmark

Megoldás videó:

Megoldás forrása : <https://gitlab.com/heinrichlaszlo/bhax/tree/codes/codes/conway/brainb>

Tanulságok, tapasztalatok, magyarázat...

A BrainB projekt célja a jövő esportolójának felkutatása. A program az agy úgynevezett kognitív képességét méri.

A játékos feladata a Samu Entropy nevű karakteren tartani az egérmutatót ami az idő teltével természetesen egyre nehezebb hiszen újabb és újabb Entropy karakterek jelennek meg a képernyőn. A program érzékeli, ha Samu Entropyt elvesztettük, ekkor csökkenti a többi Entropy számát.

```
#include <QApplication>
#include <QTextStream>
#include <QtWidgets>
#include "BrainBWin.h"

int main ( int argc, char **argv )
{
    QApplication app ( argc, argv );

    QTextStream qout ( stdout );
```

```
qout.setCodec ( "UTF-8" );

qout << "\n" << BrainBWin::appName << QString::fromUtf8 ( " ←
    Copyright (C) 2017, 2018 Norbert Bátfai" ) << endl;

qout << "This program is free software: you can redistribute it and ←
    /or modify it under" << endl;

.....
.....
.....

QRect rect = QApplication::desktop()->availableGeometry();
BrainBWin brainBWin ( rect.width(), rect.height() );
brainBWin.setWindowState ( brainBWin.windowState() ^ Qt:: ←
    WindowFullScreen );
brainBWin.show();
return app.exec();
}
```

Includoljuk a `BrainBWin` osztályt, a többi inkludolásra kerülő osztályt már az előző programokban megfigyelhettük, ugyanis ez a program is Qt grafikus felületet használ. Deklarálunk egy `app` `QApplication` típusú objektumot. Itt használjuk a `qout` functiont amivel a standard outputra lehet írni. Deklaráljuk az entropykát. Majd létrehozuk a `BrainBWin` objektumot. Az osztályban inkludolásra került függvények és változók itt is `.h` kiterjesztésű header fájlokban vannak.

BrainBWin.h

Deklaráljuk azt a függvényt ami az egér és a billentyűzetről történő inputok figyeléséért felelős. Deklaráljuk továbbá a konstruktort és a destruktort.

BrainBWin.cpp

Itt kerül definiálásra az előző header fájlban deklaráltak.

BrainBThread.h

A Hero-kat tartalmazó vectorra typedefet használunk, mostantól `Hero`sként hivatkozunk rá.

BrainBThread.cpp

Létrehozuk a Hero-kat. Definiáljuk a destruktort és deklaráljuk a `run`, `pause` és `set_paused` függvényeket

8. fejezet

Helló, Schwarzenegger!

8.1. Szoftmax Py MNIST

Python

Megoldás videó: <https://youtu.be/j7f9SkJR3oc>

Megoldás forrása: <https://github.com/tensorflow/tensorflow/releases/tag/v0.9.0> (/tensorflow-0.9.0/tensorflow/exa
https://progater.blog.hu/2016/11/13/hello_samu_a_tensorflow-bol

Mi is az a TensorFlow? Egy olyan könyvtár amit gépi tanulásra használnak. Ez az algoritmus hmindössze néhány éve jelent meg és néhány hónapja elérhető belőle a működő kiadás. Bővebben a Tensorflowról :<https://en.wikipedia.org/wiki/TensorFlow>

Mi az az Modified National Institute of Standards and Technology azaz röviden MNIST adatbázis? Egy olyan óriási elemszámú adatbázis, mely kézzel írott számjegyeket tartalmaz, amit fel fogunk használni a feladatmegoldásunk során. Bővebben a MNIST adatbázisról : https://en.wikipedia.org/wiki/MNIST_database

Ahhoz, hogy a feladatot megoldjuk, telepíteni szükséges a Pythont és a TensorFlow-t is.

```
from __future__ import absolute_import
from __future__ import division
from __future__ import print_function

import argparse

# Import data
from tensorflow.examples.tutorials.mnist import input_data

import tensorflow as tf

import matplotlib.pyplot
```

Importáljuk a szükséges könyvtárakat.

```
def readimg():
    file = tf.read_file("sajat8a.png")
```



```
print("-----")

print("-- A MNIST 42. tesztkepenek felismerese, mutatom a szamot, a ←
      tovabblepeshez csukd be az ablakat")

img = mnist.test.images[42]
image = img

matplotlib.pyplot.imshow(image.reshape(28, 28), cmap=matplotlib. ←
      pyplot.cm.binary)
matplotlib.pyplot.savefig("4.png")
matplotlib.pyplot.show()

classification = sess.run(tf.argmax(y, 1), feed_dict={x: [image]})

print("-- Ezt a halozat ennek ismeri fel: ", classification[0])
print("-----")

print("-- A saját kezi 8-asom felismerese, mutatom a szamot, a ←
      tovabblepeshez csukd be az ablakat")

img = readimg()
image = img.eval()
image = image.reshape(28*28)

matplotlib.pyplot.imshow(image.reshape(28, 28), cmap=matplotlib. ←
      pyplot.cm.binary)
matplotlib.pyplot.savefig("8.png")
matplotlib.pyplot.show()

classification = sess.run(tf.argmax(y, 1), feed_dict={x: [image]})

print("-- Ezt a halozat ennek ismeri fel: ", classification[0])
print("-----")

if __name__ == '__main__':
    parser = argparse.ArgumentParser()
    parser.add_argument('--data_dir', type=str, default='/tmp/tensorflow/ ←
        mnist/input_data',
                        help='Directory for storing input data')
    FLAGS = parser.parse_args()
    tf.app.run()
```

A hálózat tesztelése és megtudjuk miként is ismeri fel a hálózatunk az egyes számjegyeket.

8.2. Mély MNIST

Python

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

Mély MNIST feladat passzolva.

8.3. Minecraft-MALMÖ

Megoldás videó: <https://youtu.be/bAPSu3Rndi8>

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

Minecraft-MALMÖ feladat passzolva.

DRAFT

9. fejezet

Helló, Chaitin!

9.1. Iteratív és rekurzív faktoriális Lisp-ben

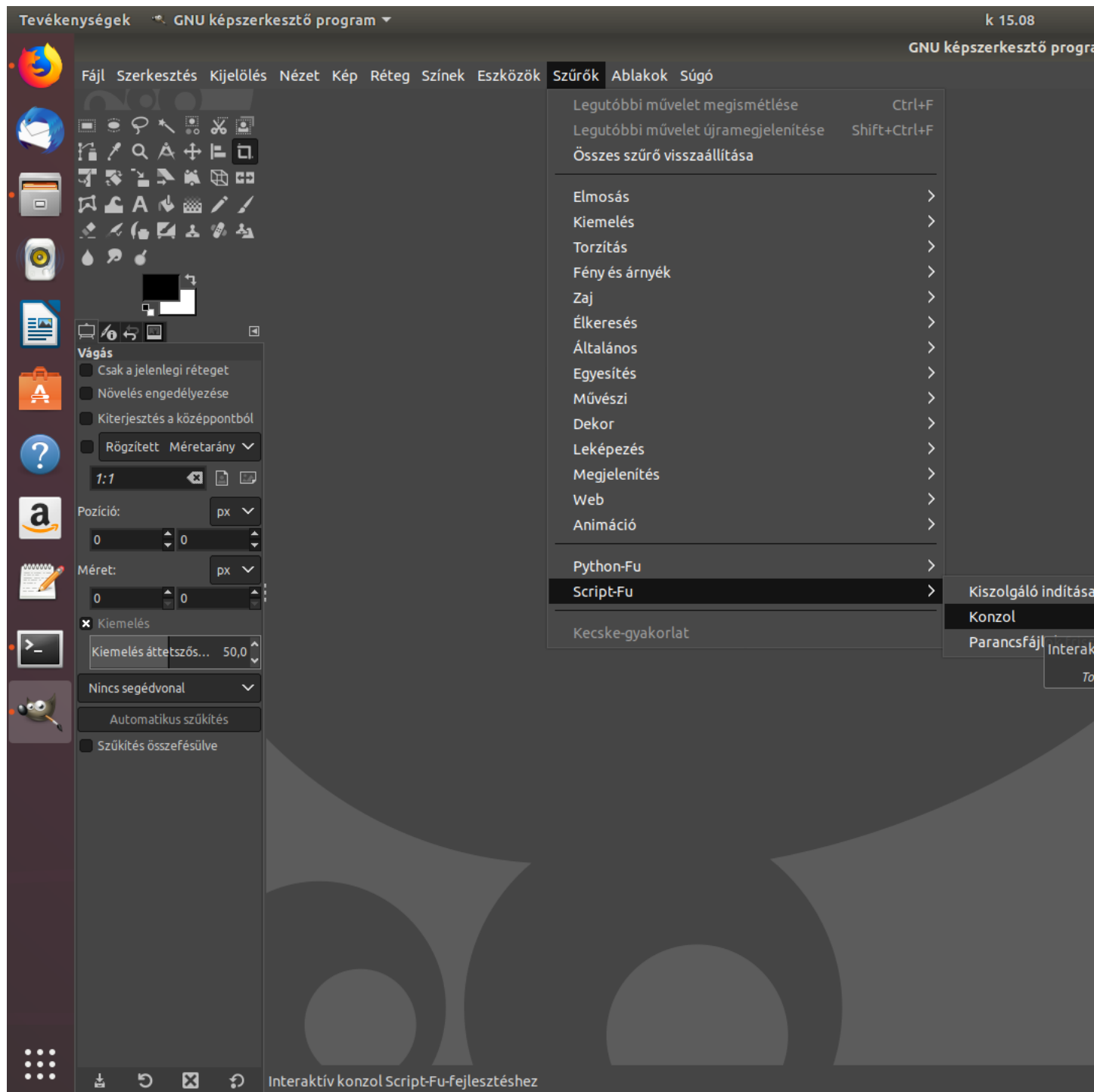
Megoldás videó: <https://youtu.be/z6NJE2a1zIA>

Megoldás forrása:

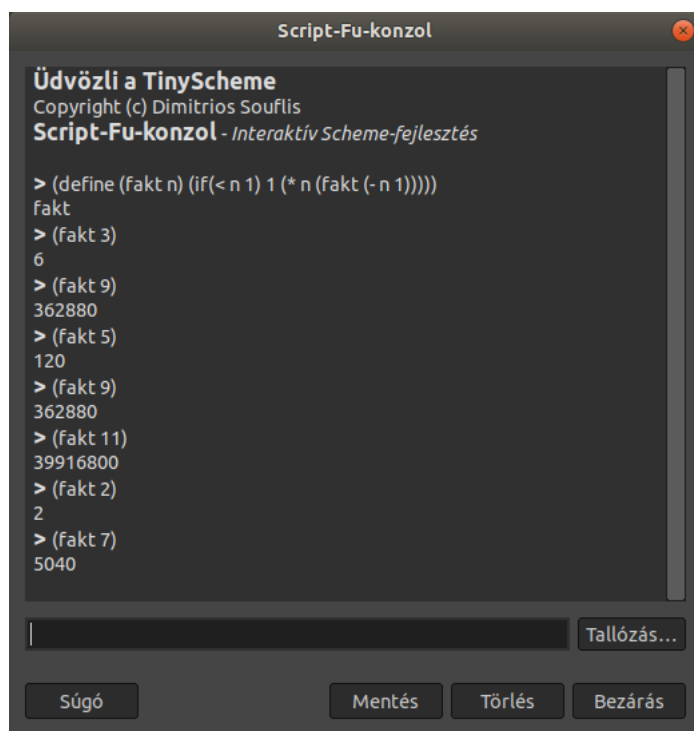
Tanulságok, tapasztalatok, magyarázat...

Most hogy már van egy elképzelésünk a nyelvről, nézzük is meg, hogyan sikerült megvalósítani a feladatot!

Mi is az a Lisp ? A Lisp nem egy újonnan létrejött nyelvcsalád. Az első Lisp nyelven írt program az 1958-as évek környékén jelent meg és hamar felkapottá vált a mesterséges intelligencia területén. Nevét az általa használt adatszerkezetből , a láncolt listáról kapta. Forrás és Lispről bővebben : [https://hu.wikipedia.org/wiki/Lisp_\(programoz%C3%A1si_nyelv\)](https://hu.wikipedia.org/wiki/Lisp_(programoz%C3%A1si_nyelv)) A következő Lisp kódot a GIMP (gimpről bővebb információk : <https://hu.wikipedia.org/wiki/GIMP>) nevű képszerkesztő program , scriptek írására alkalmas , programon belüli konzolban vizsgáljuk meg :



Következzen a kód :



```
Script-Fu-konzol

Üdvözlí a TinyScheme
Copyright (c) Dimitrios Souflis
Script-Fu-konzol - Interaktív Scheme-fejlesztés

> (define (fakt n) (if(< n 1) 1 (* n (fakt (- n 1)))))
fakt
> (fakt 3)
6
> (fakt 9)
362880
> (fakt 5)
120
> (fakt 9)
362880
> (fakt 11)
39916800
> (fakt 2)
2
> (fakt 7)
5040

Tallózás...

Súgó Mentés Törlés Bezárás
```

Ami először feltűnik, hogy maga a program csupán egyetlen sorból áll. Elemezzük balról jobbra haladva. A `define` szóval hívjuk meg a függvényünket `fakt` néven `n` paraméterrel, `n` a paraméter aminek a faktoriálisára kíváncsiak leszünk. Következik egy `if` függvény vizsgálat. A feltételvizsgálatban találkozhatunk azzal a furcsasággal, hogy a műveleti jel nem a két paraméter között áll, hanem előttük. Ha a feltételvizsgálat igaz, akkor a zárójel utáni első érték kerül visszaadásra. Ha a feltételvizsgálatunk hamis akkor a visszakapott értékünk: `n` paraméter szorzata a `fakt` függvénnyel, ami jelen esetben `n-1`-re kerül meghívásra. A függvényünk rekurzív, mert a függvény meghívja önmagát és mivel a végrehajtás ismétlődő ezért Iteratív is. Elmondhatjuk, hogy a kódunk rekurzív és iteratív egyszerre.

9.2. Gimp Scheme Script-fu: króm effekt

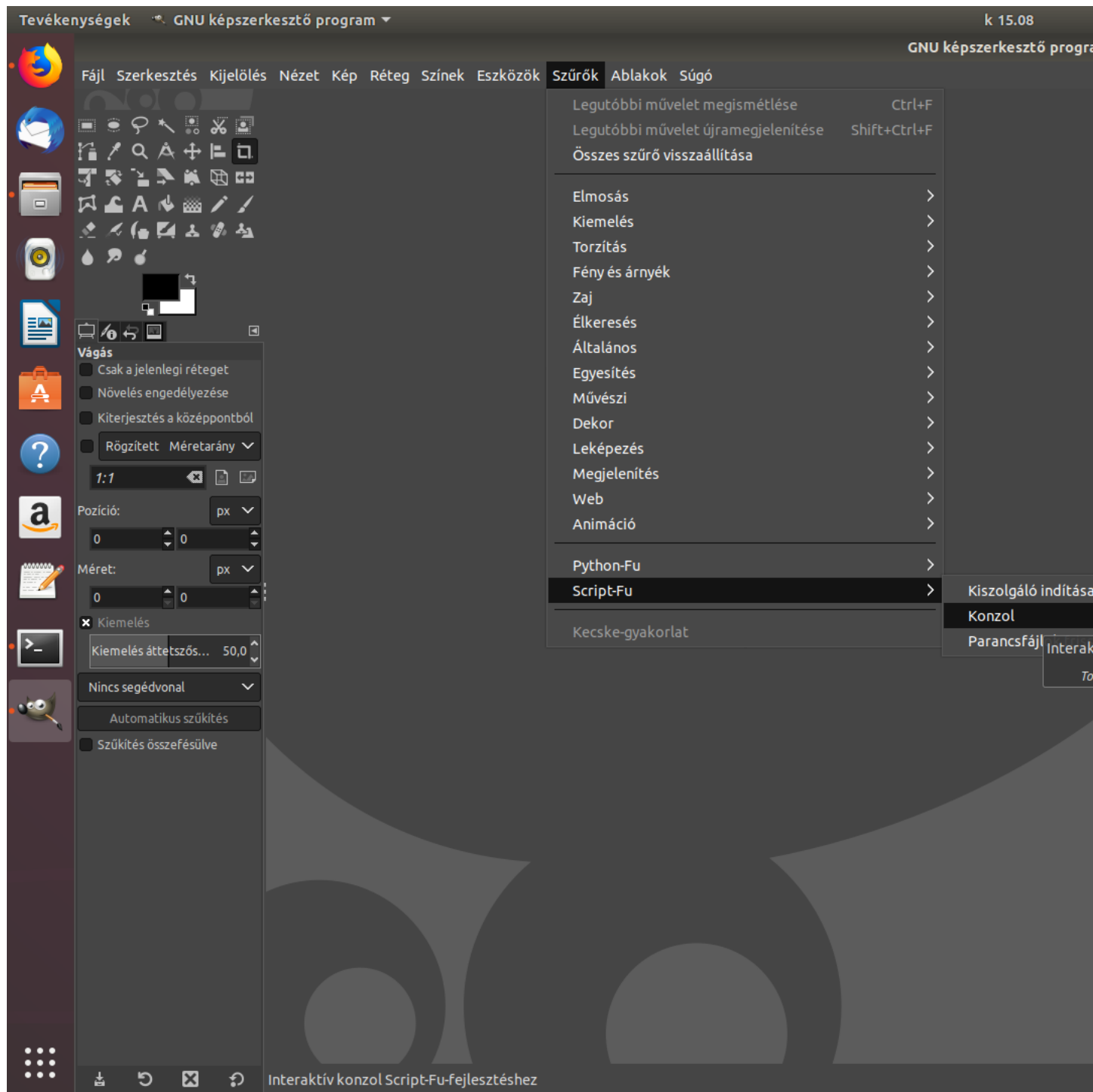
Írj olyan script-fu kiterjesztést a GIMP programhoz, amely megvalósítja a króm effektet egy bemenő szövegre!

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

Megoldás forrása: https://gitlab.com/nbatfai/bhax/tree/master/attention_raising/GIMP_Lisp/Chrome <https://gitlab.com/heinrichlaszlo/bhax/tree/codes/codes/chaitin>

Tanulságok, tapasztalatok, magyarázat...

Az előző feladatban megismerkedtünk a Lisp nyelvvel, most a feladat során a Lisp nyelvcsalád egyik képviselőjével fogunk megismerkedni. A Scheme programnyelv az 1970-es évek közepén jelent meg és mai napig találkozhatunk Scheme kódokkal. Forrás és Scheme nyelvről bővebben: <https://hu.wikipedia.org/wiki/Scheme> A fájlunk .scm kiterjesztésű lesz, és ha berakjuk a GIMP erre hivatott mappájába, megtalálhatjuk mint használható szkriptet. A feladat során a Script-fut fogjuk használni. Az előző feladat során megismert GIMP képszerkesztő programhoz egy olyan szkriptet fogunk írni, ami a bemetként megadott szöveg króm effektezését teszi lehetővé. A megíráshoz használhatjuk a Script-Fu-konzolt is:



```
(define (color-curve)
  (let* (
    (tomb (cons-array 8 'byte))
  )
    (aset tomb 0 0)
    (aset tomb 1 0)
    (aset tomb 2 50)
    (aset tomb 3 190)
    (aset tomb 4 110)
```

```

        (aset tomb 5 20)
        (aset tomb 6 200)
        (aset tomb 7 190)
    tomb)
)

```

Megadjuk a függvényeket , olyan módon amit már az előző feladatban láthattunk , ezért erre most nem térnék ki külön. Létrehozunk egy 8 elemű tömböt a `color-curve` segítségével. Majd megadjuk a megfelelő értékeket.

Megadunk egy olyan függvényt is, ami egy lista x -edik elemét adja vissza, a `car`(lista első eleme) és `cdr`(a lista első elemén kívüli összes elem) használatával.

```

(define (elem x lista)

    (if (= x 1) (car lista) (elem (- x 1) (cdr lista) ) )

)

(define (text-wh text font fontsize)
(let*
  (
    (text-width 1)
    (text-height 1)
  )

  (set! text-width (car (gimp-text-get-extents-fontname text fontsize ↵
    PIXELS font)))
  (set! text-height (elem 2 (gimp-text-get-extents-fontname text ↵
    fontsize PIXELS font)))

  (list text-width text-height)
  )
)

```

Meghívunk egy függvényt. A függvényünk a listánk x -edik elemét adja vissza a `car` és a `cdr` használatával. `car` : a lista első eleme. `cdr` : a lista elsőn kívüli összes eleme. A soron következő függvény segítségével tudjuk kiszámolni a szöveg magasságát.

```

(define (script-fu-bhax-chrome text font fontsize width height color ↵
  gradient)
(let*
  (
    (image (car (gimp-image-new width height 0)))
    (layer (car (gimp-layer-new image width height RGB-IMAGE "bg" 100 ↵
      LAYER-MODE-NORMAL-LEGACY)))
    (textfs)
    (text-width (car (text-wh text font fontsize)))
    (text-height (elem 2 (text-wh text font fontsize)))
    (layer2)
  )
)

```


A `script-fu-bhax-chrome` metódus lesz felelős a króm effektért. A `let*` segítségével létrehozuk az objektumunkat a már megadott paraméterekből. Az `image`-be létrehozuk a képünket a `gimp-image-new` használatával. Ezután létrehozunk egy új réteget a `layer`-ben. Létrehozuk a `textfs`-t. A `text-wh` segítségével megadjuk a szöveg szélességét és magasságát. A legvégén pedig létrehozuk `layer2`-t.

9 lépésben megadjuk hogy krómosítson a szkriptünk.:

Első lépés:

```
;step 1
(gimp-image-insert-layer image layer 0 0)
(gimp-context-set-foreground '(0 0 0))
(gimp-drawable-fill layer FILL-FOREGROUND )
(gimp-context-set-foreground '(255 255 255))

(set! textfs (car (gimp-text-layer-new image text font fontsize PIXELS) ←
))
(gimp-image-insert-layer image textfs 0 0)
(gimp-layer-set-offsets textfs (- (/ width 2) (/ text-width 2)) (- (/ ←
height 2) (/ text-height 2)))

(set! layer (car(gimp-image-merge-down image textfs ←
CLIP-TO-BOTTOM-LAYER)))
```

A `gimp-image-insert-layer`) segítségével hozzáadjuk a képünkhöz a rétegünket. A `gimp-context-s` segítségével beállítjuk az elsődleges színt feketére majd a beállított színnel kiszinezük a rétegünket. Végük a színt fehérre állítjuk. A `textfs`-be létrehozunk egy új szövegréteget `gimp-text-layer-new` segítségével. Majd beállítjuk `gimp-layer-set-offsets` segítségével a réteg offsetjét. Összeolvasztjuk a létrehozott szöveget és a háttérrel. Ezzel el is értünk az első lépés végére.

Második lépés:

```
;step 2
(plug-in-gauss-iir RUN-INTERACTIVE image layer 15 TRUE TRUE)
```

A `plug-in-gauss-iir` segítségével elmosódást állítunk be a `layer`-en.

Harmadik lépés:

```
;step 3
(gimp-drawable-levels layer HISTOGRAM-VALUE .11 .42 TRUE 1 0 1 TRUE)
```

A `gimp-drawable-levels` segítségével beállítjuk a rétegünk szintezését.

Negyedik lépés:

```
;step 4
(plug-in-gauss-iir RUN-INTERACTIVE image layer 2 TRUE TRUE)
```

A `plug-in-gauss-iir` segítségével még egy elmosódást teszünk a `layer`-re.

Ötödik lépés:

```
;step 5
(gimp-image-select-color image CHANNEL-OP-REPLACE layer '(0 0 0))
(gimp-selection-invert image)
```

A `gimp-image-select-color` segítségével kijelöljük a layer rétegen a fehér pixeleket. Ezt a kijelölést megfordítjuk `gimp-selection-invert` segítségével.

Hatodik lépés:

```
;step 6
(set! layer2 (car (gimp-layer-new image width height RGB-IMAGE "2" 100 ←
  LAYER-MODE-NORMAL-LEGACY)))
(gimp-image-insert-layer image layer2 0 0)
```

Létrehozunk egy új réteget és ezt hozzáadjuk a képünkhöz.

Hetedik lépés:

```
;step 7
(gimp-context-set-gradient gradient)
(gimp-edit-blend layer2 BLEND-CUSTOM LAYER-MODE-NORMAL-LEGACY ←
  GRADIENT-LINEAR 100 0 REPEAT-NONE
  FALSE TRUE 5 .1 TRUE width (/ height 3) width (- height (/ height ←
  3)))
```

Átállítjuk aktívra a gradient-t, majd a `gimp-edit-blend` segítségével összemossuk a `layer2`-t a háttérrel

Nyolcadik lépés:

```
;step 8
(plug-in-bump-map RUN-NONINTERACTIVE image layer2 layer 120 25 7 5 5 0 ←
  0 TRUE FALSE 2)
```

A `plug-in-bump-map` segítségével domborítás effektet adunk a képhez.

Kilencedik lépés:

```
;step 9
(gimp-curves-spline layer2 HISTOGRAM-VALUE 8 (color-curve))

(gimp-display-new image)
(gimp-image-clean-all image)
)
)
```

Beállítjuk a `color-curve`-t, majd megjelenítjük egy új ablakban.

```
(script-fu-register "script-fu-bhax-chrome"
  "Chrome3"
  "Creates a chrome effect on a given text."
  "Norbert Bátfai"
  "Copyright 2019, Norbert Bátfai"
```

```
"January 19, 2019"
""
SF-STRING      "Text"      "Bátf41 Haxor"
SF-FONT        "Font"      "Sans"
SF-ADJUSTMENT  "Font size" '(100 1 1000 1 10 0 1)
SF-VALUE       "Width"     "1000"
SF-VALUE       "Height"    "1000"
SF-COLOR       "Color"     '(255 0 0)
SF-GRADIENT    "Gradient"  "Crown molding"
)
(script-fu-menu-register "script-fu-bhax-chrome"
  "<Image>/File/Create/BHAX"
)
```

Menü beállítása

9.3. 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 Megoldás forrása: <https://gitlab.com/heinrichlaszlo/bhax/tree/codes/codes/chaitin>

Tanulságok, tapasztalatok, magyarázat...

A feladatunk már ismerős az előző feladatból. Egy hasonló szkriptet írni, ami a megadott szövegből mandalát készít. A Szkriptünk eleje és vége majdnem ugyanaz mint az előző feladatban, ezért azt nem fogjuk most külön kitérni.

A szkriptünk eleje :

```
(define (elem x lista)

  (if (= x 1) (car lista) (elem (- x 1) (cdr lista) ) )

)

(define (text-width text font fontsize)
  (let*
    (
      (text-width 1)
    )
    (set! text-width (car (gimp-text-get-extents-fontname text fontsize ↵
      PIXELS font)))

    text-width
  )
)
```

```
(define (text-wh text font fontsize)
  (let*
    (
      (text-width 1)
      (text-height 1)
    )
    ;;
    (set! text-width (car (gimp-text-get-extents-fontname text fontsize ←
      PIXELS font)))
    ;; ved ki a lista 2. elemét
    (set! text-height (elem 2 (gimp-text-get-extents-fontname text ←
      fontsize PIXELS font)))
    ;;
    (list text-width text-height)
  )
)
```

A szkriptünk vége :(a menü felépítése egy kicsit más , mint az előző feladatban.)

```
(script-fu-register "script-fu-bhax-mandala"
  "Mandala9"
  "Creates a mandala from a text box."
  "Norbert Bátfai"
  "Copyright 2019, Norbert Bátfai"
  "January 9, 2019"
  ""
  SF-STRING      "Text"      "Bátf41 Haxor"
  SF-STRING      "Text2"     "BHAX"
  SF-FONT         "Font"      "Sans"
  SF-ADJUSTMENT   "Font size" '(100 1 1000 1 10 0 1)
  SF-VALUE        "Width"     "1000"
  SF-VALUE        "Height"    "1000"
  SF-COLOR        "Color"     '(255 0 0)
  SF-GRADIENT     "Gradient"  "Deep Sea"
)
(script-fu-menu-register "script-fu-bhax-mandala"
  "<Image>/File/Create/BHAX"
)
```

```
(define (script-fu-bhax-mandala text text2 font fontsize width height color ←
  gradient)
  (let*
    (
      (image (car (gimp-image-new width height 0)))
      (layer (car (gimp-layer-new image width height RGB-IMAGE "bg" 100 ←
        LAYER-MODE-NORMAL-LEGACY)))
      (textfs)
      (text-layer)
      (text-width (text-width text font fontsize))
    )
  )
```

```

    ;;
    (text2-width (car (text-wh text2 font fontsize)))
    (text2-height (elem 2 (text-wh text2 font fontsize)))
    ;;
    (textfs-width)
    (textfs-height)
    (gradient-layer)
)

```

Létrehozzuk a `script-fu-bhax-mandala` függvényt. Paraméter listája majdnem megegyezik az előző feladatban már kitárgyalt paraméterlistával. A `let*` segítségével megadjuk azokat a dolgokat amikre a továbbiakban szükségünk lesz.

```
(gimp-image-insert-layer image layer 0 0)
```

Hozzáadjuk a `layer`-t a képünkhöz.

```
(gimp-context-set-foreground '(0 255 0))
(gimp-drawable-fill layer FILL-FOREGROUND)
(gimp-image-undo-disable image)
```

Átszínezzük a hátteret , majd beállítjuk h ne lehessen visszavonnást alkalmazni.

```
(gimp-context-set-foreground color)
```

Átállítjuk a `foreground color`-t az előzőekben paraméterként megadott színre.

```
(set! textfs (car (gimp-text-layer-new image text font fontsize PIXELS) ←
))
(gimp-image-insert-layer image textfs 0 -1)
(gimp-layer-set-offsets textfs (- (/ width 2) (/ text-width 2)) (/ ←
height 2))
(gimp-layer-resize-to-image-size textfs)
```

Egy új réteggént hozzáadjuk a képünkhöz a megadott szöveget, beállítjuk az offsetet , és a rétegnek a méreteit hozzá igazítjuk a kép méreteihez.

```
(set! text-layer (car (gimp-layer-new-from-drawable textfs image)))
(gimp-image-insert-layer image text-layer 0 -1)
(gimp-item-transform-rotate-simple text-layer ROTATE-180 TRUE 0 0)
(set! textfs (car(gimp-image-merge-down image text-layer ←
CLIP-TO-BOTTOM-LAYER)))

(set! text-layer (car (gimp-layer-new-from-drawable textfs image)))
(gimp-image-insert-layer image text-layer 0 -1)
(gimp-item-transform-rotate text-layer (/ *pi* 2) TRUE 0 0)
(set! textfs (car(gimp-image-merge-down image text-layer ←
CLIP-TO-BOTTOM-LAYER)))

(set! text-layer (car (gimp-layer-new-from-drawable textfs image)))
(gimp-image-insert-layer image text-layer 0 -1)
(gimp-item-transform-rotate text-layer (/ *pi* 4) TRUE 0 0)
```

```
(set! textfs (car(gimp-image-merge-down image text-layer ↔
  CLIP-TO-BOTTOM-LAYER)))

(set! text-layer (car (gimp-layer-new-from-drawable textfs image)))
(gimp-image-insert-layer image text-layer 0 -1)
(gimp-item-transform-rotate text-layer (/ *pi* 6) TRUE 0 0)
(set! textfs (car(gimp-image-merge-down image text-layer ↔
  CLIP-TO-BOTTOM-LAYER)))
```

Lemásoljuk a szöveget , elforgatjuk és az eredetivel egybeolvasszuk. Aztán ezt újra és újra.

```
(plug-in-autocrop-layer RUN-NONINTERACTIVE image textfs)
(set! textfs-width (+ (car(gimp-drawable-width textfs)) 100))
(set! textfs-height (+ (car(gimp-drawable-height textfs)) 100))
```

A túllógó részeket levágjuk. Majd a textfs-width és textfs-height méreteit beállítjuk 100-al nagyobbra a textfsméreteinél.

```
(gimp-layer-resize-to-image-size textfs)
```

A képhez méretezzük a textfs rétegünket.

```
(gimp-image-select-ellipse image CHANNEL-OP-REPLACE (- (- (/ width 2) ↔
  (/ textfs-width 2)) 18)
  (- (- (/ height 2) (/ textfs-height 2)) 18) (+ textfs-width 36) (+ ↔
  textfs-height 36))
(plug-in-sel2path RUN-NONINTERACTIVE image textfs)

(gimp-context-set-brush-size 22)
(gimp-edit-stroke textfs)
```

Készítünk egy ellipszis kijelölést a szövegünk köré és kifestjük a határát.

```
(gimp-context-set-brush-size 22)
(gimp-edit-stroke textfs)

(set! textfs-width (- textfs-width 70))
(set! textfs-height (- textfs-height 70))

(gimp-image-select-ellipse image CHANNEL-OP-REPLACE (- (- (/ width 2) ↔
  (/ textfs-width 2)) 18)
  (- (- (/ height 2) (/ textfs-height 2)) 18) (+ textfs-width 36) (+ ↔
  textfs-height 36))
(plug-in-sel2path RUN-NONINTERACTIVE image textfs)

(gimp-context-set-brush-size 8)
(gimp-edit-stroke textfs)
```

A textfs-width és textfs-height méretét csökkentjük 70nel . Aztán elvégezzük a korábbi "festést" még egyszer.

```
(set! gradient-layer (car (gimp-layer-new image width height RGB-IMAGE ↵  
  "gradient" 100 LAYER-MODE-NORMAL-LEGACY)))
```

A gradient-layer-t állítjuk be.

```
(gimp-image-insert-layer image gradient-layer 0 -1)  
(gimp-image-select-item image CHANNEL-OP-REPLACE textfs)  
(gimp-context-set-gradient gradient)  
(gimp-edit-blend gradient-layer BLEND-CUSTOM LAYER-MODE-NORMAL-LEGACY ↵  
  GRADIENT-RADIAL 100 0  
REPEAT-TRIANGULAR FALSE TRUE 5 .1 TRUE (/ width 2) (/ height 2) (+ (+ ↵  
  (/ width 2) (/ textfs-width 2)) 8) (/ height 2))
```

Hozzáadjuk a legújabb rétegünket. Kijelöljük a textfs-t. Megadjuk a kapott paramétereket a gradient-nek. Aztán összeillesztjük a képünket.

```
(plug-in-sel2path RUN-NONINTERACTIVE image textfs)  
  
(set! textfs (car (gimp-text-layer-new image text2 font fontsize PIXELS ↵  
  )))  
(gimp-image-insert-layer image textfs 0 -1)  
(gimp-message (number->string text2-height))  
(gimp-layer-set-offsets textfs (- (/ width 2) (/ text2-width 2)) (- (/ ↵  
  height 2) (/ text2-height 2)))  
  
; (gimp-selection-none image)  
; (gimp-image-flatten image)  
  
(gimp-display-new image)  
(gimp-image-clean-all image)
```

Létrehozunk egy új szöveget majd hozzáadjuk új réteggént. Majd egy új ablakban megjelenítjük a felugró képet.

10. fejezet

Helló, Gutenberg!

10.1. Juhász István: Magas szintű programozási nyelvek - olvasónapló

1.2 Alapfogalmak

Megismerjük , hogy a programozási nyelveknek milyen 3 színje van. A szintek : a gépi nyelv , az assembly szintű nyelv illetve a magas szintű nyelv. Mi a magas szintű nyelvvel fogunk részletesen foglalkozni. A magas szintű programozási nyelven írt programok a forráspontok és a forrásszövegek. A forrásszövegre vonatkozó nyelvtani szabályokat szintaktikának nevezzük. A tartalmi szabályok összességét pedig szemantikának. A magas szintű programozási nyelveket ez a két tényező fogja meghatározni. Ezeknek a szabályoknak az összefoglaló neve a hivatkozásui nyelv.

A forrásszövegből szükséges a processzor által értelmezhető gép kóddá konvertálnunk a kódot , amire lét technika áll rendelkezésünkre : fordítóprogram és interpreter. A fordítóprogram egy magas szintű programnyelven megírt kódból tárgyprogramot képes előállítani, a következő lépésekkel : lexikális elemzés , szintaktikai elemzés , szemantikai elemzés, kódgenerálás. A lépések végrehajtódása után megkapjuk a gépi kódunkat. Ebből a gépi kódból a kapcsolatszerkesztő állít elő futtatható kódot. Az interpreteres megoldás nem készít tárgyprogramot. Ez a két technika együttesen is használható. Emelett léteznek még előszeretettel használt IDE-k (integrált fejlesztői környezetek) amitkomplex feladatkörrel láttak el a fejlesztők.

1.3 A programnyelvek osztályozása

A programozási nyelveket két főcsoportra oszthatjuk. A két fő csoport : Dekleratív és Imperatív nyelvek.

A deklaratív nyelvek jellemzői : ezek a nyelvek nem algoritmikus nyelvek , nem kötődnek a Neumann architektúrákhoz. Nincs bennük lehetőség memóriaműveletekre. Két alcsoportot különböztetünk meg : funkcionális és logikai.

Az imperatív nyelvek jellemzői : algoritmikus nyelvek, ami azt jelenti , hogy a leprogramozott algoritmus működteti a processzort. A program utasításokból áll. A legfőbb összetevői a változók. Alcsoportjai : eljárás és objektumorientált nyelvek

Alapelemek

2.1 - Karakterkészlet

A program legkisebb alkotórészeit karaktereknek nevezzük. Karakterekből épülnek fel a bonyolultabb nyelvi elemek : lexikális és szintaktikai egységek , utasítások , programegységek , fordítási egységek , és

maga a program. A karaktereket a programnyelvekben 3 különböző féle képpen csoportosíthatjuk : betűk , számjegyek , egyéb karakterek.

A fordító a lexikális elemzés során felismeri és tokenizálja a lexikális egységeket. Számos fajtát különböztetünk meg : többkarakteres szimbólum , szimbólikus név , címke , megjegyzés , literál stb.

A többkarakteres szimbólumok : A többkarakteres szimbólumoknak az esetek nagy részében a nyelv tulajdonít jelentést. Gyakran operátorok , pl : ++ , -- , stb.

A címke az utasítások jelölésére szolgál , hogy a program egy másik pontjáról hivatkozni lehessen rá.

A címke az utasítások jelölésére szolgál. A program egy másik pontjáról is lehet rá hivatkozni.

A megjegyzések szerepe fontos , mert ezeknek a megjegyzéseknek segítségével olyan dolgokat közölhetünk , ami segíti az olvasást és az értelmezést.

A literálok segítségével , meg nem változtatható értékeket vezethetünk be a programkódunkba. Két része van : típus és érték.

2.4 - Adattípusok

Egy absztrakt programozási eszköz az adatabsztrakció legelelső megjelenési módja. Két fajtája van : típusos és nem típusos nyelvek . Az adattípus neve egy azonosító, majd ezt követi egy belső ábrázolási mód is. Egy adattípust 3 dolog határoz meg:

- tartomány
- műveletek
- reprezentáció.

Minden típusos nyelv rendelkezik beépített típusokkal, de egyes nyelvek engedélyezik a programozó által definiált saját típusokat is. Egy ilyen definiáláshoz meg kell adni a típus tartományát, műveleteit és reprezentációját. Az adattípusoknak két nagy csoportja létezik:

- egyszerű,
- összetett típusok.

Az egyszerű típusok csoportjába tartoznak: egész és valós típusok , karakteres típus, egyes nyelvek esetében a logikai típus. Speciális egyszerű típus a felsorolásos típus és a sorszámozott típus.

Az eljárásorientált nyelvekben a két legismertebb összetett típus a tömb és a rekord. A tömb egy statikus és homogén típus.

2.4.3. Mutató típus

Tárcímeket tároló egyszerű típus, amivel megvalósítható az indirekt címzés. Fő feladata a megcímzett tárterületen található érték minél hamarabbi elérése. Ha nem mutat sehova , akkor NULL érték.

2.5 A nevesített konstans

A nevesített konstans 3 komponenssel rendelkezik : névvel , típussal és értékkel. Ezeket minden esetben deklarálni kell. Szerepük , hogy a gyakran előforduló értékeknek adhatunk neveket és ha az adott értéket megszeretnénk változtatni nagy segítséget nyújt ebben.

2.6 A változók

A változók olyan programozási eszközök aminek a komponensei a következők : név , attribútum , cím és érték. A név egy azonosító. Az attribútumok olyan jellemzők , amik a futás közbeni működését határozza meg a változóknak. A változóknak deklarációval értéket rendelhetünk. A változó címe a tárnak az a címe ahol megtalálhatjuk a változó értékét . Az értékkomponens a címen elhelyezkedő bitkombinációként mutatkozik meg.

2.7 Alapelemek az egyes nyelvekben

Két fő típust figyelhetünk meg : aritmetikai és származtatott típus. Aritmetikai az egyszerűek és a származtatott az összetett típusok .

3. Kifejezések

A kifejezések olyan szintaktikai eszközök , amik lehetővé teszik , hogy egy adott ponton a már eleve ismert értékekből új értékeket határozzunk meg. Kér részre bontjuk őket : érték és típus. A kifejezések összetevői : operandusok , operátorok , kerek zárójelek.

Az operandusok képviselik az értéket. Az operátorok a műveleti jelek. A zárójelek a műveleti sorrendet befolyásolják.

4. Utasítások

Az utasításoknak két fő típusa van : deklarációs és végrehajtható utasítások. A deklarációs utasítások a fordítóprogramnak szólnak.

A végrehajtható utasításoknak főbb típusai: értékeadó utasítás, üres utasítás, ugró utasítás, elágaztató utasítások, ciklusszervező utasítások, hívó utasítás, vezérlésátadó utasítások, Input/Output utasítások, egyéb utasítások.

Az értékeadó utasítás feladata a változó értékének beállítása.

Az üres utasításokat feladata átláthatóbb szerkezetűvé tenni a programot .

Az ugró utasítással pontról át lehet ugrani egy adott címkével jelölt utasításra.

Elágaztató utasítások:

Az elágaztató utasítások olyan , utasítások amiben a feltételvizsgálat eredménye dönti el, milyen művelet kerüljön végrehajtásra. A vizsgált feltétel egy logikai kifejezés.

Léteznek egy és több irányú utasítások.

Ciklusszervező utasítások:

Egy ciklusnak 3 fő része van: fej, mag és vég. A mag tartalmazza az utasításokat. A cikluson nem rendelteték szerű működésének két iskolapéldája az üres ciklus, ami egyszer sem fut le és a végtelen ciklus ami "végtelen" ideig fut.

Feltételes ciklusok

Az egyik fajtája a kezdőfeltételes ciklus amely esetében a fejben lévő feltétel kiértékelődése után hajtódik végre a ciklusmagban található kód addig amíg hamis nem lesz a feltétel. A végfeltételes ciklus esetében a feltétel a ciklus végében van, esetében először a mag hajtódik végre és csak ezután értékelődik ki a feltétel.

Vezérlő utasítások

CONTINUE, BREAK , RETURN.

A programok szerkezete

Programegységekről az eljárásorientált nyelvek esetében beszélhetünk, melyek a következők :

alprogram, blokk, csomag.

Az alprogramok az újrafelhasználás eszközei, akkor használjuk, ha többször szükséges elvégeznünk egy adott műveletet. Az alprogramot egyszer kell megírunk, utána hivatkoznunk kell csak rá ott, ahol az eredeti programrész szerepelt volna.

A fejben található a név. Ez azonosítja az alprogramot. Itt található még a formális paraméterlista is mely kerek zárójelek között áll. A paraméterlista a a paraméterkiértékelés során játszik fontos szerepet.

A törzsben találhatóak a deklarációs és végrehajtható utasítások.

Az alprogram környezete a globális változóinak együttese. Két fajtája van: eljárás és függvény. Az eljárás hajtja végre a tevékenységet , de nincs visszatérési értéke. A függvény egy tetszőleges típusú értékű eredményt ad vissza.

5.2 Hívási lánc, rekurzió

Akkor alakul ki a hívási lánc, ha egy programegység meghív egy másikat, az pedig egy következőt, és így tovább , tovább.

Ha egy aktív alprogramot hívunk meg, rekurzióról beszélünk, ami kétféle lehet. Ha egy alprogram önmagát hívja meg beszélhetünk közvetlen meghívásról. De ha a hívási láncban már szereplő alprogramot hívunk meg akkor közvetett meghívásról beszélünk.

5.4 Paraméterkiértékelés

A paraméterkiértékelés akkor figyelhető meg , amikor egy függvény vagy eljárás hívásnál megfigyelhető a mechanizmus, amely során egy alprogram formális-és aktuális paraméterei egymáshoz történő megfeleltetése megtörténik.

5.5 Paraméterátadás

Amikor paraméterátadásról beszélünk mindig van egy hívó és egy hívott. A hívott mindig az alprogram. Különböző paraméterátadási módokat ismerhetünk: érték-, cím-, eredmény-, érték-eredmény-, név- és szöveg szerinti. Érték szerinti paraméterátadáskor a formális paraméterek rendelkeznek címkomponenssel az alprogram területén. Esetében az információ áramlása egyirányú, működése során értékmásolás hajtódik végre.

5.6 A blokk

A blokk egy olyan programegység ami egy másik programegységben helyezkedhet el, szerkezetileg van kezdete, törzse és vége. A blokkban megtalálható elemek egyértelműen meghatározhatók. Paraméterrel Nem rendelkeznek paraméterrel.

5.7 Hatáskör

Egy név hatásköre a programnak az a része, ahol az adott névvel ugyanarra az eszközre hivatkozunk.

A blokk általános alakja:

```
{  
  deklarációk  
  vegrehajtható utasítások  
}
```

5.9 Az egyes nyelvek eszközei - C

A fejezet szemlélteti a blokk és függvény alakját.

6. Absztrakt adattípus

Az Absztrakt adattípusoknál nem ismerjük sem a reprezentációt sem pedig a műveletek implementációját, mert ezt az adattípus nem mutatja a külvilág számára. Hozzáférésük interfészeken keresztül történik.

10. Generikus programozás

A generikus programozás az újrafelhasználhatóság eszköze, ami egy olyan eszközrendszer ami a legtöbb nyelvbe beépíthető. Megadunk egy paraméterezhető forrásszöveg mintát. Amiből előállítható egy konkrét fordítható szöveg.

13. Input/output

Az Input/Output egy olyan eszköz rendszer ami felelős a perifériákkal való kommunikáció megvalósításáért és annak fenntartásáért, a memóriából adatokat küld a perifériákhoz vagy fordítva. Az állomány egy programban lehet fizikai illetve logikai. Az állományokat megkülönböztethetjük funkció szerint is, lehetnek: input állományok - csak olvasni lehet belőle; output - csak írni lehet bele; input-output - olvasni és írni is lehet. Az I/O során kétféle adatátviteli módot alkalmazunk amelyekkel az adatok a memória és a periféria közötti mozgásukat végzik: folyamatos vagy bináris.

A következő lépések szükségesek, ha a programunkban állományokkal szeretnénk dolgozni:

- deklaráció
- összerendelés
- állomány megnyitása
- feldolgozás
- lezárás

9. kivételkezelés

Kivételről beszélünk olyan névvel és kóddal rendelkező események esetében, amelyek megszakítást okoznak.

10.2. KR: A C programozási nyelv - olvasónapló

2. fejezet: Típusok

A C nyelv adattípusai :

- char: mérete 1 byte, a karakterkészlet egy elemét tartalmazza
- int: egész szám
- float: lebegőpontos szám

- double:lebegőpontos szám

Aritmetikai operátorok : +, -, *, / és %. Ezek a következők: összeadás, kivonás, szorzás, egész osztás és maradékos osztás operátorok.

Léteznek relációs és logikai operátorok is. A relációsak: >, >=, <, <=, =.

A logikai operátorok: || illetve && .

Inkrementáló és dekrementáló operátorokkal : ++ és --. Az inkrementáló operátor 1-et ad hozzá az operandushoz míg a dekrementáló 1-et von ki belőle. Ezek az operátorok állhatnak prefix- vagy postfix operátorként.

Bitenkénti logikai operátorok: &, |, ^, <<, >>, ~

Értékadó operátorok : a könyv 58. oldalától kezdődően találkozhatunk.

3. fejezet: Vezérlési szerkezetek

3.1 Utasítások és blokkok

A C nyelvben a pontosvessző az utasításokat lezáró jel. A kapcsos zárójelek segítségével a deklarációkat és utasításokat egy nagy blokkba lehet foglalni, ami szintaktikailag egyetlen utasítással ekvivalens.

3.2 Az if-else utasítás

Döntést, választást írunk le vele. A könyv 65. oldala fellelhető példa:

```
if (kifejezes)
1. utasitas
else
2. utasitas
```

3.3 Az else-if utasítás

A könyv példája:

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

3.4 A switch utasítás

Nagyon hasonló az if..else if..else szerkezethez, viszont olvashatóbb. Lásd a könyv 69. oldal :

```
switch (kifejezes)
case :
case :
case :
case :
case :
default :
```

3.5 A while és a for utasítás

A könyv példa:

```
while (kifejezes)
    utasítás
```

```
for (kifejezes1; kifejezes2; kifejezes3)
    utasítás
```

alakú for utasítás egyenértékű a

A for bármely 3 kifejezése elhagyható. A for és a while ciklus között szabadon választhatunk. d

3.6 A do-while utasítás

```
do
    utasítás
while (kifejezes);
```

3.7 A break utasítás

A break utasítással már a ciklusbeli feltételvizsgálat előtt is ki lehet ugrani a ciklusokból. A break utasítás eredményeképp a vezérlés a legbelső zárt ciklusból azonnal kilép.

3.8 A continue utasítás

A könyv 76. oldal:

```
for (i = 0; i < N; i++) {
    if (a[i] != 0) /*Páros elemek átugrása*/
        continue;
}
...
/*Páratlan elemek feldolgozása*/
```

A continue megkezdí a ciklus következő iterációját.

3.9 A goto utasítás; címkék

A goto utasítás segítségével lehet címkére ugrani.

77. oldal:

```
for ( . . . )
for ( . . . ) {
    . . .
    if (zavar)
        goto hiba;
    . . .
}
hiba: számold fel a zavart
```

Alapja megegyezik a változónevekével , de a kettőspont követi a nevüket.

Utasítások

Utasítások fő fajtái:

- feltételes utasítás
- while utasítás
- do utasítás
- for utasítás
- switch utasítás
- break utasítás
- continue utasítás
- goto utasítás
- címkézett utasítás

A return utasítás

```
return;  
return kifejezés;
```

Először határozatlan a visszadaott érték , másodszor a kifejezés értéke a visszaadott érték.

A nulla utasítás

Egy pontosvesszőből áll, amipéldául üres ciklustörzset is képezhet.

A kifejezés utasítás

Utasítások , függvényhívások vagy értékadások.

Az összetett utasítás vagy blokk

A blokkok segítségével tudunk egy utasításból több utasítást képezni úgy , hogy az lényegében olyan legyen mintha még mindig egy utasítás lenne.

10.3. Benedek Zoltán, Levendovszky Tihamér: Szoftverfejlesztés C++ nyelven - olvasónapló

A C++ nem objektumorientált újdonságai

2.1 A C és a C++ nyelv

A C nyelvtől eltérően nem tetszőleges számú paraméterrel hívható ha a függvényt üres paraméterlistával definiáltunk , hanem akkor az olyan, mintha egy void paraméterrel definiáltuk volna.

2.1 A main függvény

Két formája létezik C++-ban. Példa a 4.oldalon

2.1.3 A bool típus

A C++ nyelvben megismerkedünk a bool típussal , ami logikai értéket tárol.

A C++-ban már beépített típusként kaptak helyet a C-ben már jól ismert , több-bájtos sztringek megvalósítására alkalmas `wchar_t` típus.

2.2 Függvények túlterhelése

A C nyelvben a függvényt a neve azonosítja. C++-ban : a név és az argumentumlista együttesen azonosít egy függvényt. Megismerkedünk linker névelferdítési technikájával , amit azonos nevű függvények esetén alkalmazhatunk. Szó esik az `extern` kulcsszó használatáról , amit abban az esetben használhatunk , ha C-ben akarunk C++ függvényt meghívni.

2.3 Alapértelmezett függvényargumentumok

A C++ nyelvben, függvényeinknek meg lehet adni alapértelmezett értékeket.

2.4 Paraméterátadás referenciátípussal

A C nyelvben a paraméterátadás érték szerint lehetséges. C++ : létezik referenciával való paraméterátadás. Ez a cím szerinti paraméterátadás.

Objektumok és osztályok

3.1 Az objektumorientáltság alapelvei

A programok összetettségének növekedése miatt már a kódok nagyon átláthatatlanok lettek , lehetetlen volt már őket kezelni. Ezért terjedt el az objektumorientált programozás.

Az egységbe zárt adatstruktúra neve osztály. Az osztálynak vannak egyedei, amiket objektumoknak nevezünk. Beszélünk ezek mellett az öröklődés és az egységbe zárt fogalmakról is. Ezek az objektumorientált programozás alapelvei.

3.2 Egységbe zárt a C++-ban

Átvesszi a tagváltozók és tagfüggvények szerepét. A függvény rendelkezik egy láthatatlan, első paraméterrel, ami alapján tudni fogja, melyik struktúrát kell módosítani. A `this` kulcsszó.

3.3 Adatrejtés

A `private` kulcsszó. A `private` kulcsszó után álló változók és függvények csak az osztályon belül láthatóak. Ellentéte a `public`.

A változó létrehozását az osztályból magyarul példányosításnak hívjuk, a példány más néven objektum.

3.4 Konstruktorok és destruktorok

A konstruktor olyan speciális tagfüggvény, példányosításkor hívódik meg. A destruktor felel az objektumok által lefoglalt erőforrások felszabadításáért , az objektum megszűnésekor automatikusan meghívódik.

3.5 Dinamikus adattagot tartalmazó osztályok

A `new` a dinamikus memóriakezelésért felelős operátor , majd felszabadítjuk a `delete` operátorral a lefoglalt helyet. Tömbök esetén: `new []` és a `delete []` használhatóak.

3.5.3 Másoló konstruktor

Egy referenciát vár, aminek típusa megegyezik az osztály típusával. Az újonnan létrehozott objektumot inicializáljuk egy már létező objektum alapján .

3.6 Friend függvények és osztályok

3.6.1 Friend függvények

A `friend` kulcsszó feljogosíthat függvényeket, hogy hozzáférhessenek az osztály védett tagjaihoz.

3.6.1 Friend osztályok

Lényegében megegyeznek a `friend` függvényekkel. Az osztályunk egy másik osztályt jogosít fel a védett tagjaihoz való hozzáférésre.

3.7 Tagváltozók inicializálása

Az inicializálás és az értékadás különbség : Az inicializálás konstruktorhívás történik. Értékadás esetén az `=` operátor hívódik meg.

3.8 Statikus tagok

Nem az objektumhoz hanem az osztályhoz tartozó tagváltozók. A statikus tagfüggvények objektum nélkül, az osztály nevéen keresztül is használhatók. A `this` mutató statikus függvények esetében nem használható.

3.9 Beágyazott definíciók

A C++ nyelven belül lehetőségünk van az osztály, struktúra, típusdefiníciók osztályon belüli megadására. Ezt beágyazott definíciónak nevezzük.

Operátorok és túlterhelésük

6.1. Az operátorokról általában

Az átadott argumentumokon végzel el a műveletet , majd az eredményt visszatérési értéként adjuk meg. Mellékhatásnak nevezzük azt a jelenséget amikor az argumentum értékét megváltoztatja az operátok. A zárójel használata a bonyolultabb kifejezéseknél fontos.

6.2 Függvényszintaxis és túlterhelés

A C nyelvben nem lehetséges , csak ha a változóra mutató mutatót adunk át. C++-ban ez lehetséges, a referencia szerinti paraméterátadással. 94. oldal:

```
int noveles(int& ertekek)
{
    int eredmeny = ertekek;
    ertekek = ertekek+1; //mellékhatás
    return eredmeny;
}
```

C++-ban az operátorok függvényszintaxissal történő meghívása esetén is van lehetőség:

```
z = x+y;
z = operator+ (x,y);
```

C++ sablonok

Vannak olyan osztály- és függvénysablonok, amiknek néhány elem definiáláskor paraméternek veszünk. Gyakori az alkalmazásuk : tetszőleges típusú elemek tarolására alkalmas tároló osztályok létrehozásánál.

11. 1 C++ függvénysablonok

A függvényt sablonná alakítjuk át. A típus amivel dolgozunk, nem rögzített,a sablon felhasználásakor adjuk meg. A könyvben található példa:

```
template <class N> inline N max (N lhs, N rhs)
{
    return lhs > rhs ? lhs: rhs;
}
```

Felhasználása:

```
int y = max(3, 5);  
double z = max(3.1, 5.5);
```

11.1.2 Példák függvénysablonokra

A sablonparaméter nem csak típus, hanem lehet típusos konstans is. pl.:

```
template <int N> int Square()  
{  
    return N*N;  
}  
int main()  
{  
    const int x= 10;  
    cout <<x<<"négyzete:" <<Square<x>() << endl;  
}
```

11.1.3 A hívott függvény kiválasztása

Ha függvénynek több implementációja van , akkor bizonyos szabályok döntenek el annak kiválasztásakor, melyik függvény kerül meghívásra.

11.2.1 Osztálysablonok írása

1. lépés: Osztálydefinícióból sablondefiníció
2. lépés: Sablonparaméterek bevezetése
3. lépés: Nem implicit inline definiált tagfüggvények szintaktikája
4. lépés: Osztálynév helyett osztálynév ÉS sablonparaméterek szerepeltetése a stípusként való felhasználás során

A C++ I/O alapjai

5.1 A szabványos adatfolyamok

A C nyelvben 3 fájlleíró áll rendelkezésünkre : stdin, a stdout és stderr , ezek FILE* típusúak, magas szintű állományleírók.

A C++ nyelv adatfolyamokban gondolkodik. Az istream típusú objektumok alatt csak olvasható adatfolyamokat értünk, ostream típusú alatt pedig csak írhatókat. Ezek az adatfolyamok az << és >> operátorokat használják beolvasásra és kiíratásra.

Ahhoz, hogy használni tudjuk az I/O-t C++-ban, includoljuk az iostream állományt. Mivel egy objektum az adatfolyam , így az állapotát beállító konstansok: eofbit, failbit, badbit, goodbit. Az első 3 mind különböző hibát jelez vissza, az utolsó a helyes működésről tájékoztat. A C és C++-beli, beolvasást megvalósító függvények összehasonlítása is megtörténik a 79. oldalon. Ugyanezen az oldalon a getline függvény is említésre kerül, amit stringek beolvasásakor használunk arra, hogy a beolvasás a szóközöknél ne szakadjon meg .

5.2 Manipulátorok és formázás

A manipulátor egy speciális objektum, amelyet az adatfolyamokra alkalmazunk a be és ki meneti operátorok argumentumaként. Az előre definiált manipulátorok az iomanip fájlban találhatók amit programunkba

inkludálnunk kell ha használni akarunk. Manipulátorokra példa: `endl`, `flush`, `noskipws`, `setw`. Jelzőbitek és maszk fogalmak megjelenése. A leggyakoribb manipulátorok és tagfüggvényeik összefoglaló táblázata a 84-85. oldalon található meg.

5.3 Állománykezelés

A C++ programnyelv az állománykezeléshez is adatfolyamokat használ, amiket az `ifstream`, az `ofstream` és az `fstream` osztályok reprezentálnak. Az állományok megnyitását konstruktorok, míg bezárását a destruktorok végzik.

Kivételkezelés

A kódban a hibakódok számára egy külön osztály került létrehozásra melyben a kódok definiálása történt meg. A kivételkezelés az előnyös megoldás, amely számunkra lehetővé teszi azt, hogy kivétel esetén a vezérlés a kivételkezelőhöz kerüljön. A `main`-ben található `try-catch` szerkezet `try` részébe a helyes működés, míg a `catch` részébe a kivételkezelő kódja kerül.

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

11. fejezet

Helló, Arroway!

11.1. OO szemlélet

A módosított polártranszformációs normális generátor beprogramozása Java nyelven. Mutassunk rá, hogy a mi természetes saját megoldásunk (az algoritmus egyszerre két normálist állít elő, kell egy példánytag, amely a nem visszaadottat tárolja és egy logikai tag, hogy van-e tárolt vagy futtatni kell az algoritmust) és az OpenJDK, Oracle JDK-ban a Sun által adott OO szervezés ua.! <https://arato.inf.unideb.hu/batfai.norbert/UDPROG> (16-22 fólia) Ugyanezt írjuk meg C++ nyelven is! (lásd még UDPROG repó: source/labor/polargen)

Megoldás videó:

Megoldás forrása: <https://gitlab.com/heinrichlaszlo/prog2forras/blob/master/Arroway/PolarGenerator.java>

A polár generátor projekt egy klasszikus OO bevezető példa, ezért most ezt fogjuk megtekinteni.

```
boolean nincsTárolt = true;
double tárolt;
%%
%}
```

A class elején deklarálunk két változót, az egyik a nem visszatérő adatot tárolja, a másik egy logikai kifejezés, ami arra mutat rá, hogy van-e tárolt, vagy futtatnunk kell az algoritmust újra?

```
if (nincsTárolt)
{
double u1, u2, v1, v2, w;
do
{
u1 = Math.random();
u2 = Math.random();
v1 = 2 * u1 - 1;
v2 = 2 * u2 - 1;
w = v1 * v1 + v2 * v2;
}
while (w > 1);
tárolt = w;
%}
```

Az algoritmusunk elején megnézzük, hogy a logikai kifejezésünk igaz-e?

Ha igen akkor nézzük az If feltétel vizsgálatunk belsejét : Deklarálunk 2 változót amiket random számokkal határozunk meg. Ezután néhány matematikai műveletet végzünk el rajta, egészen addig amíg a "w" nagyobb nem lesz mint 1 (a w = a két szám kétszerese).

```
} while (w > 1);
double r = Math.sqrt((-2 * Math.log(w)) / w);
tárolt = r * v2;
nincsTárolt = !nincsTárolt;
return r * v1;
}
%%
%}
```

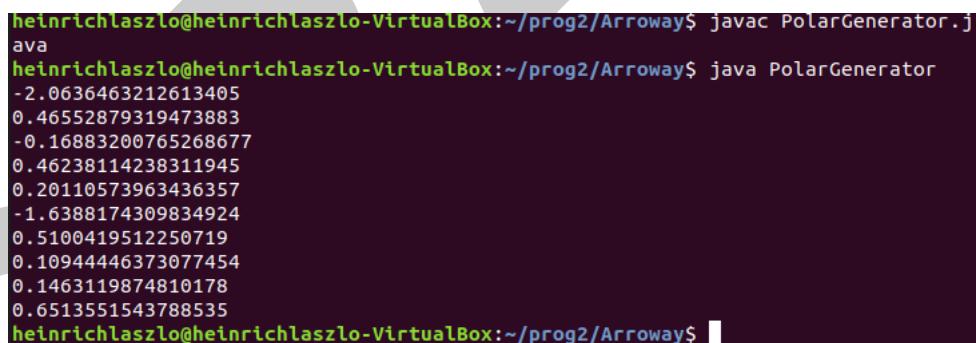
A tárol adatban elmentjük a legelősször létrehozott szorzatot (r) , majd a logikai változót az ellenkezőjére állítjuk be (!nincsTárolt). Ezután visszatérítjük a második random kétszeresét.

```
else
{
nincsTárolt = !nincsTárolt;
return tárolt;
}

%%
%}
```

Lefut az else ág , mert megváltoztattuk a logikai változónkat, ahol a "tárolt" adatainkat tároltuk.

Fordítjuk majd futtatjuk a programunkat.



```
heinchlaszlo@heinchlaszlo-VirtualBox:~/prog2/Arroway$ javac PolarGenerator.j
ava
heinchlaszlo@heinchlaszlo-VirtualBox:~/prog2/Arroway$ java PolarGenerator
-2.0636463212613405
0.46552879319473883
-0.16883200765268677
0.46238114238311945
0.20110573963436357
-1.6388174309834924
0.5100419512250719
0.10944446373077454
0.1463119874810178
0.6513551543788535
heinchlaszlo@heinchlaszlo-VirtualBox:~/prog2/Arroway$
```

11.2. Homokozó

Írjuk át az első védési programot (LZW binfa) C++ nyelvről Java nyelvre, ugyanúgy működjön! Mutasunk rá, hogy gyakorlatilag a pointereket és referenciákat kell kiirtani és minden máris működik (erre utal a feladat neve, hogy Java-ban minden referencia, nincs választás, hogy mondjuk egy attribútum pointer, referencia vagy tagként tartalmazott legyen). Miután már áttettük Java nyelvre, tegyük be egy Java Servletbe és a böngészőből GET-es kéréssel (például a böngésző címsorából) kapja meg azt a mintát, amelynek kiszámolja az LZW binfáját! 1

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

11.3. "Gagyi"

Az ismert formális 2 „while (x <= t && x >= t && t != x);” tesztkérdéstípusra adj a szokásosnál (miszerint x, t az egyik esetben az objektum által hordozott érték, a másikban meg az objektum referenciája) „mélyebb” választ, írd Java példaprogramot mely egyszer végtelen ciklus, más x, t értékekkel meg nem! A példát építsd a JDK Integer.java forrására 3, hogy a 128-nál inkluzív objektum példányokat poolozza!

Megoldás videó:

Megoldás forrása: <https://gitlab.com/heinrichlaszlo/prog2forras/blob/master/Arroway/gagyi128.java> <https://gitlab.com/heinrichlaszlo/prog2forras/blob/master/Arroway/gagyi129.java>

Ebben a feladatban a „while (x <= t && x >= t && t != x);” ciklusra vagyunk kíváncsiak. Hogyan lehetséges az, hogy van olyan szám amelynél végtelen ciklusba kerül miközben az értéket vizsgálja. Eközben egy másiknál befejeződik a ciklus, a referenciákat vizsgálva.

-129-nél : Mindkét objektumban tárolt érték -129 ezért x<=t és x>=t teljesülni fog. De a két integerben két objektum más-más címmel rendelkezik, ezért az egyenlőségük nem teljesülhet. A while ciklusunk ekkor végtelen ciklusba kerül, mert a programunk x != t része teljesülni fog.

```
heinrichlaszlo@heinrichlaszlo-VirtualBox:~/prog2/Arroway$ javac gagyi129.java
heinrichlaszlo@heinrichlaszlo-VirtualBox:~/prog2/Arroway$ java gagyi129
-129
-129
```

-128-nál : Ebben az esetben is teljesülni fog a x<=t, x>=t. Ebben az esetben ugyanarra a számra ugyanazt az objektumot kapjuk, ebben az esetben már az objektumot előre elkészített pool-ból kapjuk, referenciájuk megegyezik. Az objektumot felhasználva az azonos érték miatt azonos lesz. A while ciklusban az x!=t hamis lesz, emiatt a while ciklus leáll és nem kerülünk végtelen ciklusba.

```
heinrichlaszlo@heinrichlaszlo-VirtualBox:~/prog2/Arroway$ javac gagyi128.java
heinrichlaszlo@heinrichlaszlo-VirtualBox:~/prog2/Arroway$ java gagyi128
-128
-128
heinrichlaszlo@heinrichlaszlo-VirtualBox:~/prog2/Arroway$
```

11.4. Yoda

Írjunk olyan Java programot, ami java.lang.NullPointerException-el leáll, ha nem követjük a Yoda conditions-t! https://en.wikipedia.org/wiki/Yoda_conditions

Megoldás videó:

Megoldás forrása: <https://gitlab.com/heinrichlaszlo/prog2forras/blob/master/Arroway/gagyi129.java>

Mi is az a "Yoda körülmények"? A Yoda körülmények az utasítás rendjétől függően két egymástól különböző képpen tud lefordulni a program.

Ebben az állapotban a konstans, a feltételes utasítás a bal oldalon, míg a változó a bal oldalon fordul elő.

Nevét a Star Wars világhírű filmsorozatról kapta, ahol az említett "körülmény" névadója, Yoda mester beszéd közben nem tartotta be az angol nyelv nyelvtanát.


```
heinchlaszlo@heinchlaszlo-VirtualBox:~/prog2/Arroway$ javac yoda.java
heinchlaszlo@heinchlaszlo-VirtualBox:~/prog2/Arroway$ java yoda
heinchlaszlo@heinchlaszlo-VirtualBox:~/prog2/Arroway$
```

```
public class yoda {
    public static void main(String[] args){
        String s=null;
        if
            // (s.equals("yoda")) leáll NullPointerException-nal
            ("yoda".equals(s)) //lefordul de nem fut le.

        {

            System.out.println("Yoda");

        }
    }
}
```

A program lefordul. Majd a yoda "körülmenyek" elhagyása után leáll NullPointerException-nel, és nem fut tovább.

```
heinchlaszlo@heinchlaszlo-VirtualBox:~/prog2/Arroway$ javac yoda.java
heinchlaszlo@heinchlaszlo-VirtualBox:~/prog2/Arroway$ java yoda
Exception in thread "main" java.lang.NullPointerException
    at yoda.main(yoda.java:5)
heinchlaszlo@heinchlaszlo-VirtualBox:~/prog2/Arroway$
```

```
public class yoda {
    public static void main(String[] args){
        String s=null;
        if
            (s.equals("yoda")) //leáll NullPointerException-nal
            //("yoda".equals(s)) lefordul de nem fut le.

        {

            System.out.println("Yoda");

        }
    }
}
```

11.5. Kódolás from scratch

Induljunk ki ebből a tudományos közleményből: <http://crd-legacy.lbl.gov/~dhbailey/dhbpapers/bbp- alg.pdf> és csak ezt tanulmányozva írjuk meg Java nyelven a BBP algoritmus megvalósítását! Ha megakadsz, de csak végső esetben: https://www.tankonyvtar.hu/hu/tartalom/tkt/javat-tanitok-javat/apbs02.html#pi_jegyei (mert ha csak lemásolod, akkor pont az a fejlesztői élmény marad ki, melyet szeretném, ha átélnél).

Megoldás videó:

Megoldás forrása: <https://gitlab.com/heinchlaszlo/prog2forras/blob/master/Arroway/bbp.java>

A Számítás a BBP vagy a Bailey-Borwein-Plouffe algoritmus segítségével történik.

A BBP algoritmus segítségével az előző betűk ismerete nélkül tudunk meghatározni a Pi számjegyeit hexadecimális számrendszerben.

A mi programunk a 1000001. elemtől fogja számítani a hexadecimális számjegyeket.

$$\pi = \sum_{k=0}^{\infty} \frac{1}{16^k} \left(\frac{4}{8k+1} - \frac{2}{8k+4} - \frac{1}{8k+5} - \frac{1}{8k+6} \right)$$

```
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;
```

```
public class bbp {
    public static char betuzes(double pi)
    {
        int kovetkezojegy = (int) (pi*16);
        //System.out.println("Pi: "+pi*16);
        char betu = 0;
        if(kovetkezojegy>=0 && kovetkezojegy<=9 )
```

```
        betu = (char) (kovetkezojegy + '0');

    else{
        if(kovetkezojegy==10)
            betu='A';
        else{
            if(kovetkezojegy==11)
                betu='B';
            else{
                if(kovetkezojegy==12)
                    betu='C';
                else{
                    if(kovetkezojegy==13)
                        betu='D';
                    else{
                        if(kovetkezojegy==14)
                            betu='E';
                        else{
                            if(kovetkezojegy==15)
                                betu='F';
                        }
                    }
                }
            }
        }
    }
    //System.out.println("belepett");
    //System.out.println(betu);
    return betu;
}

public static double Hexalas(double pi)
{
    pi=(pi*16) - (int) (pi*16);
    //System.out.println("Pi jegyek2: "+pi);
    return pi;
}

public static double modulo(double n,double k){
    long t=1;
    double r=1;
    int b=16;

    while(t<=n){
        t=t*2;
    }

    t/=2;
    while(true)
    {
        if(n>=t)
        {
```

```
        r=b*r%k;
        n=n-t;
    }
    t/=2;

    if(t>=1) {
        r=Math.pow(r,2)%k;
    }
    else
        break;
    }
    return r;
}

public static void main(String[] args) throws IOException {
    int d=1000000;
    double S1 = 0,S4 = 0,S5 = 0,S6 = 0, pi=0;

    if(d>0)
    {
        for (int k = 0; k <= d; k++) {

            S1=S1+((modulo((d-k),(8*k+1)))/(8*k+1));
            S4=S4+((modulo((d-k),(8*k+4)))/(8*k+4));
            S5=S5+((modulo((d-k),(8*k+5)))/(8*k+5));
            S6=S6+((modulo((d-k),(8*k+6)))/(8*k+6));

        }

        S1=S1-Math.round(S1);
        S4=S4-Math.round(S4);
        S5=S5-Math.round(S5);
        S6=S6-Math.round(S6);

    }

    pi=4*S1-2*S4-S5-S6;

    String pijegyek = new String();
    while(pi>0)
    {
        //System.out.println("Pi jegyek: "+pi);
        pijegyek=pijegyek+betuzes(pi);
        //System.out.println("Pi jegyek: "+pi);
        pi=Hexalas(pi);
        //System.out.println(pi);

    }

    System.out.println(pijegyek);
}
```

```
    }  
  
}  
  
%%  
    % }
```

Fordítás és futtatás után, a következő eredményt kapjuk.

```
heinrichlaszlo@heinrichlaszlo-VirtualBox:~/prog2/Arrowway$ javac bbp.java  
heinrichlaszlo@heinrichlaszlo-VirtualBox:~/prog2/Arrowway$ java bbp  
6C65E5308
```

12. fejezet

Helló, Berners-Lee!

12.1. Forstner Bertalan, Ekler Péter, Kelényi Imre : Bevezetés a mobil programozásba

A Python egy dinamikus, objektumorientált és platformfüggetlen. Összetettebb problémák megoldására is használható, de inkább prototípusok tesztelésére és készítésére használják a fejlesztési fázisban.

A programozók munkáját nagyban könnyíti, hogy a forrást nem kell állandóan menteni és fordítani, hanem elég ha az interpreter megkapja, ami azonnal fordítja és futtatja azt.

A Python az alkalmazás fejlesztők munkáját is megkönnyíti köszönhetően annak, hogy rengeteg modul található benne és maga a nyelv nagyon könnyen elsajátítható.

A python programokra ha laikus szemmel ránézünk az első ami szembejön, hogy sokkal rövidebb és átláthatóbb, mint egy Java illetve C/C++ program.

A nyelv behúzásokra épül, ezeket szóközökkel vagy tabulátorokkal érjük el. Leegyszerűsítve a minden utasítás egy-egy sor. Nincs szükségünk zárójelekre vagy ";"-re. A Python minden sort tokenekre (azonosító, kulcsszó, operátor stb.) oszt, amik között whitespace helyezkedik el.

A megjegyzéseket a "#" -tel hozhatjuk létre.

Pythonban az objektumok reprezentálják az adatokat. A rajtuk végezhető műveletek tehát az objektum típusától függenek. Ennek megadására nincs szükség, ugyanis az interpreter valós időben határozza meg az adott változó típusát. pl. Számok, Stringek stb.. A számoknak lehetnek például: egészek (lehetnek oktális, hexadecimális, decimális), lebegő pontosak, vagy komplexek (pl C-ben a double típus). Stringeket idézőjelek és aposztrófok közé írjuk.

12.2. C++ és Java nyelv összehasonlítása

Az első hasonlóság, a C++ és a Java között, hogy mindkét esetben objektum orientált nyelvről beszélünk.

De mi az az objektumorientált nyelv? Leegyszerűsítve annyit jelent, hogy a program amit megírtunk Java nyelven objektumokból és osztályokból áll.

Mi az az osztály? Az osztályt metódusok és változók alkotják, ez ismerős lehet a C++ nyelvből is.

A könyv elején találkozhatunk egy régi jó baráttal, a "Hello World"-del. A könyvünk ez alapján mutatja be, hogyan működik a java fordítója. Itt találkozhatunk az első különbséggel a Java és C++ között, ugyanis a Java egy Java Virtuális Gépet használ a bájtkódok értelmezéséhez és ennek segítségével fordítja le a kódunkat.

Ha elindítjuk a programunkat, akkor a fordító program, az úgynevezett virtuális gép a megadott osztály main metódusát fogja fordítani.

A könyvben a szerző ismerteti az osztályokat, változókat, a megjegyzéseket, illetve a konstansokat.

Utóbbiról annyi, hogy megadása a final szóval lehetséges.

A megjegyzésnek két fajtáját ismerjük, létezik egysoros illetve több soros megjegyzés. Az egysoros jelölése `"/"`, emellett a több soros jelölése a következő: a megjegyzésünk elején `"/*"` jelölést, majd a végén `"*/"` jelölést alkalmazunk.

A megjegyzések a program, illetve a közös program egy elhagyhatatlan része, nagyban megkönnyíti a másokkal közös munkát.

Az osztályok: Az osztálynak az adattagjait és metódusait bármilyen sorrendben felsorolhatjuk. A Javában a `class` szóval tudunk létrehozni osztályokat.

A könyv a példában a `String` nevezetű típust hozza fel.

Változók: több típussal találkozhatunk: pl.: `int` (32 bites egész szám), `long` (64 bites egész szám), `char` (16 bites karakter) `boolean` (igaz vagy hamis).

A kivételkezelés a Javában a C++-ból már ismert `try-catch` szerkezet.

13. fejezet

Helló, Liskov!

13.1. Liskov helyettesítés sértése

Írjunk olyan OO, leforduló Java és C++ kódcsipetet, amely megsérti a Liskov elvet! Mutassunk rá a megoldásra: jobb OO tervezés. https://arato.inf.unideb.hu/batfai.norbert/UDPROG/deprecated/Prog2_1.pdf (93-99 fólia) (számos példa szerepel az elv megsértésére az UDPROG repóban, lásd pl. [source/binom/Batfai-Barki/madarak/](#))

Megoldás videó:

Megoldás forrása: <https://gitlab.com/heinrichlaszlo/prog2forras/blob/master/liskov/liskovhely.cpp> <https://gitlab.com/heinrichlaszlo/prog2forras/blob/master/liskov/liskovhely.java>

Lefutni lefut, de nem működik. A probléma nagyon leegyszerűsítve a következő : Vegyük a madarakat, a madarak tudnak repülni. Vegyük a pingvint, a pingvin nem tud repülni , pedig madár!!!

Ha P osztály M osztály leszármazottja, akkor P -t szabadon be tudjuk helyettesíteni minden egyes olyan helyre , ahol M típust várunk, hogy a program helyes futtatását eredményezze.

Az előbb felhozott madaras példa ennek a megsértésére tökéletesen működik. Az előbbi elv alapján, ha P lesz pingvin és M lesz madár, akkor ebben az esetben a repülési készséghez be kellene tudnunk helyettesíteni P-nek, mivel a pingvin madár és köztudottan a madarak tudnak repülni. Ennek ellenére azt is tudjuk ,hogy a pingvin egy röpképtelen madár így nem helyettesíthető be.

A leírtakat összegezve a pingvines madaras példánk sérti a Liskov elvet.

```
#include <iostream>

using namespace std;

class Madar {
public:
    virtual void repul() {
        cout<<"Tudok repülni!"<<endl;
    };
};

class Program {
```

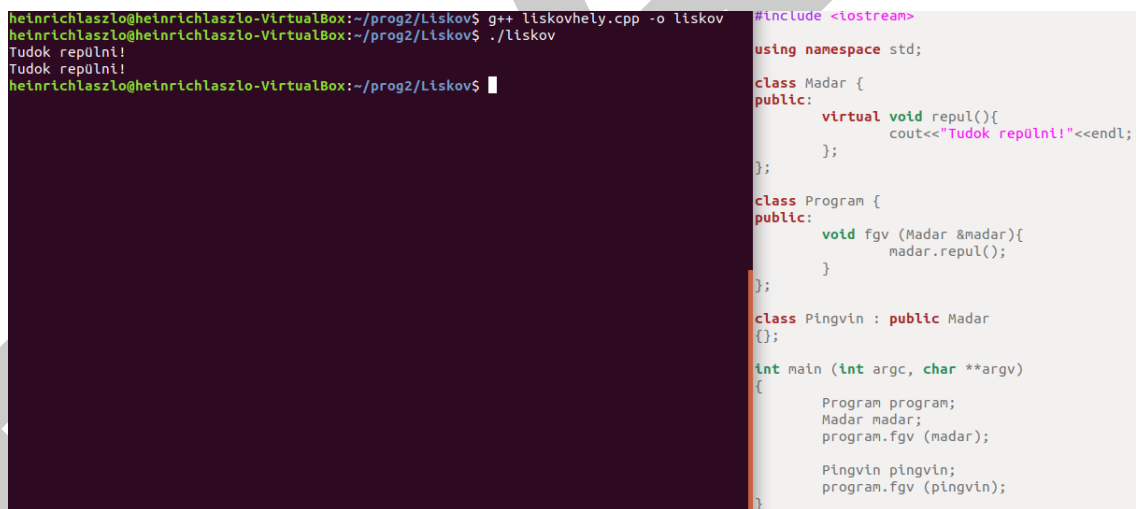
```
public:
    void fgv (Madar &madar){
        madar.repul();
    }
};

class Pingvin : public Madar
{};

int main (int argc, char **argv)
{
    Program program;
    Madar madar;
    program.fgv (madar);

    Pingvin pingvin;
    program.fgv (pingvin);
}
```

Létrehozzuk a Madar osztályt ami kap egy repul függvénnel. Majd létrehozzuk a Program osztályt, amiben lesz egy fgv függvény. Az fgv függvény feladata lesz a repul függvény meghívásra egy Madar típusu egyedre. Létrehozzuk a Pingvin osztályt, aminek szülőosztálya a Madar osztály lesz. Az utolsó lépés a példányosítás és a függvények meghívása.



```
#include <iostream>
using namespace std;

class Madar {
public:
    virtual void repul(){
        cout<<"Tudok repülni!"<<endl;
    };
};

class Program {
public:
    void fgv (Madar &madar){
        madar.repul();
    }
};

class Pingvin : public Madar
{};

int main (int argc, char **argv)
{
    Program program;
    Madar madar;
    program.fgv (madar);

    Pingvin pingvin;
    program.fgv (pingvin);
}
```

```
heinchlaszlo@heinchlaszlo-VirtualBox:~/prog2/Liskov$ g++ liskovhely.cpp -o liskov
heinchlaszlo@heinchlaszlo-VirtualBox:~/prog2/Liskov$ ./liskov
Tudok repülni!
Tudok repülni!
heinchlaszlo@heinchlaszlo-VirtualBox:~/prog2/Liskov$
```

Írjuk át a kódunkat Javara :

```
class Madar
{
    public void repul ()
    {
        System.out.print("Tudok repulni!\n");
    }
}

class Pingvin extends Madar
{
}
```



```
}

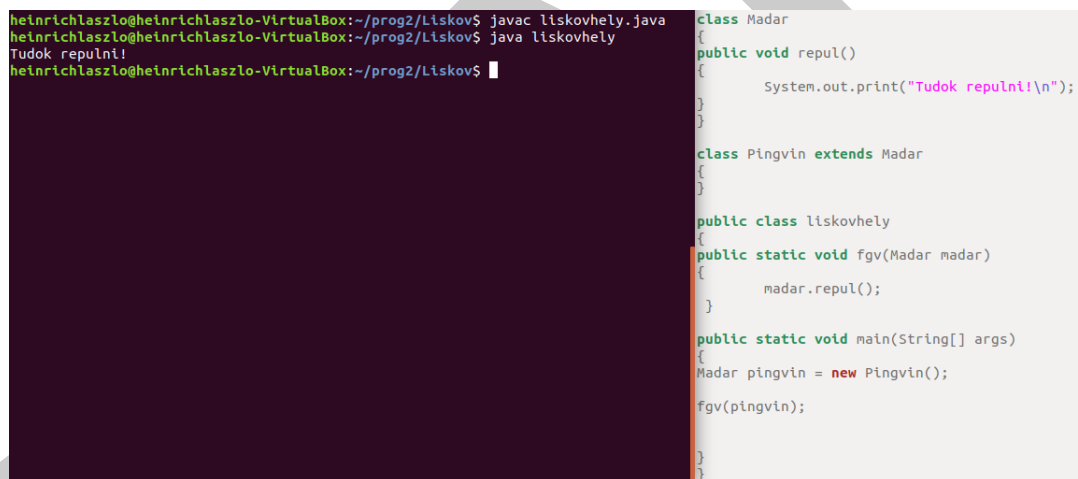
public class Liskovsert
{
public static void fgv(Madar madar)
{
    madar.repul();
}

public static void main(String[] args)
{
Madar pingvin = new Pingvin();

fgv(pingvin);

}
}
```

Fordítsuk majd futtasuk a programot



```
heinchlaszlo@heinchlaszlo-VirtualBox:~/prog2/Liskov$ javac liskovhely.java
heinchlaszlo@heinchlaszlo-VirtualBox:~/prog2/Liskov$ java liskovhely
Tudok repulni!
heinchlaszlo@heinchlaszlo-VirtualBox:~/prog2/Liskov$
```

```
class Madar
{
    public void repul()
    {
        System.out.print("Tudok repulni!\n");
    }
}

class Pingvin extends Madar
{
}

public class liskovhely
{
    public static void fgv(Madar madar)
    {
        madar.repul();
    }

    public static void main(String[] args)
    {
        Madar pingvin = new Pingvin();
        fgv(pingvin);
    }
}
```

Probléma nélkül lefut a programunk és újra megreptettük a mi röpképtelen madarunkat.

13.2. Szülő-gyerek

Írjunk Szülő-gyerek Java és C++ osztálydefiníciót, amelyben demonstrálni tudjuk, hogy az őson keresztül csak az ős üzenetei küldhetők! https://arato.inf.unideb.hu/batfai.norbert/UDPROG/deprecated/Prog2_1.pdf (98. fólia)

Megoldás videó:

Megoldás forrása: <https://gitlab.com/heinrichlaszlo/prog2forras/blob/master/liskov/szgy.cpp> <https://gitlab.com/heinrichlaszlo/prog2forras/blob/master/liskov/szgy.java>

A feladat lényege, hogy írjunk egy olyan programot, amely bemutatja, hogy az őson keresztül csak az ős üzenetei küldhetők.

Mit értünk ez alatt ?

Leegyszerűsítve a gyermek osztály használhatja , az ősz osztály változóit , metódusait. DE , ez a fordítva nem igaz, az ősz osztály nem tudja értelmezni a gyermekosztályban definiált metódusokat , változókat stb.

```
public class szgy{
    class Szulo{
    }

    class Gyerek extends Szulo
    {
        public void viselkedik()
        {
        }
    }
    public static void main(String[] args)
    {
        szgy SzGy = new szgy();

        Szulo szulo = SzGy.new Gyerek();
        Szulo.viselkedik();

    }
}
```

Létrehozásra kerül a Szulo osztály. Majd létrehozásra kerül a Gyerek osztály, ami a Szulo osztály leszármazotja. A származtatás az extends kulcsszóval történik. Létrehozzuk a viselkedik függvényt. Következik a main. MAjd végül meghívjuk a szülőre a gyerek metódusát.

Fordítjuk majd futtatjuk a programot.

```
heinchlaszlo@heinchlaszlo-VirtualBox:~/prog2/Liskov$ javac szgy.java
szgy.java:16: error: cannot find symbol
        Szulo.viselkedik();
              ^
symbol:   method viselkedik()
location: class szgy.Szulo
1 error
heinchlaszlo@heinchlaszlo-VirtualBox:~/prog2/Liskov$
```

Az alosztályunkban meghívott függvények nem használhatóak a Szulo osztályon, ezért a programunk hibát is dob ki.

Írjuk át a kódunkat C++-ra :

```
#include <iostream>

using namespace std;

class Szulo
{
};

class Gyerek: public Szulo
```

```
{  
void viselkedik()  
{  
cout<<"Viselkedek!";  
}  
};  
  
int main ()  
{  
Szulo* sz= new Gyerek;  
  
cout<<sz->viselkedik();  
}
```

Fordítjuk a programot :

```
heinrichlaszlo@heinrichlaszlo-VirtualBox:~/prog2/Liskov$ gcc szgy.cpp -o szgy  
szgy.cpp: In function 'int main()':  
szgy.cpp:22:11: error: 'class Szulo' has no member named 'viselkedik'  
  cout<<sz->viselkedik();  
               ^~~~~~  
heinrichlaszlo@heinrichlaszlo-VirtualBox:~/prog2/Liskov$
```

A fordító ismét hibát jelez , tehát újra bebizonyítottuk , hogy a szülőosztály nem tudja az alosztályának a metódusait értelmezni.

13.3. Anti OO

A BBP algoritmussal 4 a Pi hexadecimális kifejtésének a 0. pozíciótól számított 10⁶, 10⁷, 10⁸ darab jegyét határozzuk meg C, C++, Java és C# nyelveken és vessük össze a futási időket! <https://www.tankonyvtar.hu/hu/tartalom/tanitok-javat/apas03.html#id561066>

Megoldás videó:

Megoldás forrása: <https://gitlab.com/heinrichlaszlo/prog2forras/blob/master/liskov/PiBBPBench678.c> <https://gitlab.com/heinrichlaszlo/prog2forras/blob/master/liskov/PiBBPBench678.java> <https://gitlab.com/heinrichlaszlo/prog2forras/blob/master/liskov/PiBBPBench678.cpp>

A feladatunk az, hogy a programunk segítségével, határozzuk meg a pi számjegyeinek 10⁶, 10⁷ és 10⁸ darab jegyét.

Ezt a BBP algoritmus segítségével fogjuk végre hajtani, majd lemérjük ezeknek a futási idejét, és megnézzük melyik a leggyorsabb.

A programunk for ciklusának d ciklusváltozóját kell módosítanunk attól függően , hogy Pi hanyadik számjegyet szeretnénk megkapni

```
for (d = 1000000; d < 1000001; ++d)
```

```
heinrichlaszlo@heinrichlaszlo-VirtualBox:~/prog2/Liskov$ g++ PiBBPBench678.c -o pi
heinrichlaszlo@heinrichlaszlo-VirtualBox:~/prog2/Liskov$ ./pi
6
1.510720
heinrichlaszlo@heinrichlaszlo-VirtualBox:~/prog2/Liskov$ g++ PiBBPBench678.cpp -o pi
heinrichlaszlo@heinrichlaszlo-VirtualBox:~/prog2/Liskov$ ./pi
6
1.51042
heinrichlaszlo@heinrichlaszlo-VirtualBox:~/prog2/Liskov$ javac PiBBPBench678.java
heinrichlaszlo@heinrichlaszlo-VirtualBox:~/prog2/Liskov$ java PiBBPBench678
6
1.35
heinrichlaszlo@heinrichlaszlo-VirtualBox:~/prog2/Liskov$

heinrichlaszlo@heinrichlaszlo-VirtualBox:~/prog2/Liskov$ g++ PiBBPBench678.c -o
pi
heinrichlaszlo@heinrichlaszlo-VirtualBox:~/prog2/Liskov$ ./pi
7
18.076774
heinrichlaszlo@heinrichlaszlo-VirtualBox:~/prog2/Liskov$ g++ PiBBPBench678.cpp -
o pi
heinrichlaszlo@heinrichlaszlo-VirtualBox:~/prog2/Liskov$ ./pi
7
17.7307
heinrichlaszlo@heinrichlaszlo-VirtualBox:~/prog2/Liskov$ javac PiBBPBench678.jav
a
heinrichlaszlo@heinrichlaszlo-VirtualBox:~/prog2/Liskov$ java PiBBPBench678
7
15.874

heinrichlaszlo@heinrichlaszlo-VirtualBox:~/prog2/Liskov$ g++ PiBBPBench678.c -o
pi
heinrichlaszlo@heinrichlaszlo-VirtualBox:~/prog2/Liskov$ ./pi
12
205.863604
heinrichlaszlo@heinrichlaszlo-VirtualBox:~/prog2/Liskov$ g++ PiBBPBench678.cpp -
o pi
heinrichlaszlo@heinrichlaszlo-VirtualBox:~/prog2/Liskov$ ./pi
12
205.257
heinrichlaszlo@heinrichlaszlo-VirtualBox:~/prog2/Liskov$ javac PiBBPBench678.jav
a
heinrichlaszlo@heinrichlaszlo-VirtualBox:~/prog2/Liskov$ java PiBBPBench678
12
180.743
```

Mint látjuk mind 3 esetben a java verzió futott le leghamarabb , ezt követte a c++ majd a c verzió.

13.4. Hello, Android!

Élesszük fel az SMNIST for Humans projektet! <https://gitlab.com/nbatfai/smnist/tree/master/forHumans/SMNIST>
Apró módosításokat eszközölj benne, pl. színvilág.

Megoldás videó:

Megoldás forrása:

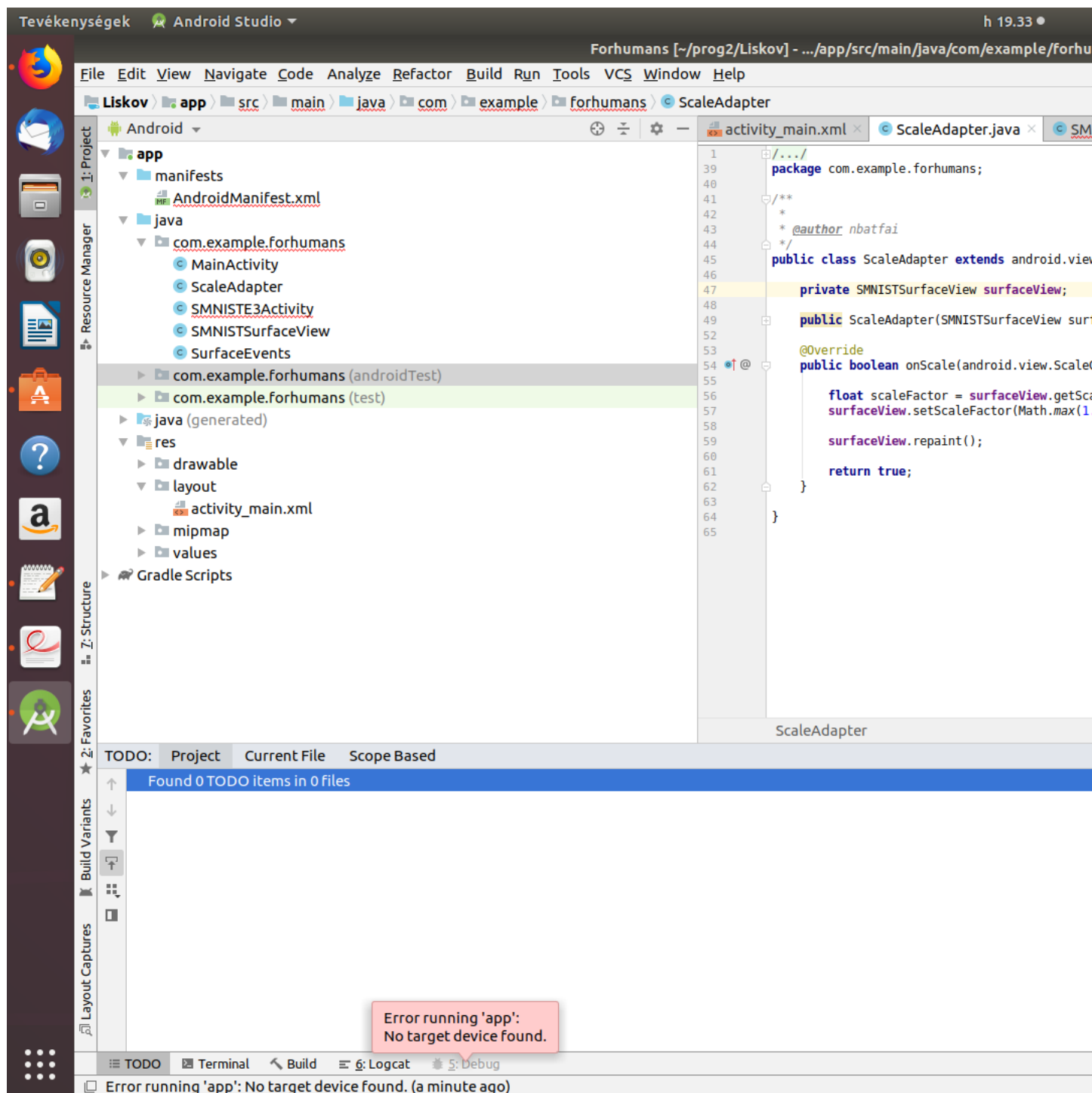
A feladatunk , hogy az SMNIST for Humans projektet felélesszük , úgy , hogy kisebb módosításokat eszközölünk rajta. A feladat megoldásához az SMNISTSurfaceView.java fájlban kell módosításokat elvégeznünk.

A feladatban példának a háttérszín módosítását említették , ennél az alábbi kódrészletbe kell belenyúlnunk.

```
int [] bgColor =
{
    android.graphics.Color.rgb(11,180,250),
```

```
android.graphics.Color.rgb(11,250,180),  
};
```

A Java forrásfájl Android Studio-ban szerettem volna futtatni. A telepítés és konfigurálást követően a program futtatása nem sikerült az alábbi probléma miatt :



14. fejezet

Helló, Mandelbrot!

14.1. Reverse engineering UML osztálydiagram

UML osztálydiagram rajzolása az első védési C++ programhoz. Az osztálydiagramot a forrásokból generáljuk (pl. Argo UML, Umbrello, Eclipse UML) Mutassunk rá a kompozíció és aggregáció kapcsolatára a forráskódban és a diagramon, lásd még: https://youtu.be/Td_nIERIEOs. <https://arato.inf.unideb.hu/batfai.norbert/UD> (28-32 fólia)

Megoldás videó:

A feladatunk az volt , hogy UML osztálydiagrammot készítsünk az első védési feladathoz, azaz a binfához.

De mi is az az UML ?

Az UML (Unified Modelling Language) egy egységes modellezőnyelv. A grafikus ábrázoláshoz szükséges eszközök gyűjteménye.

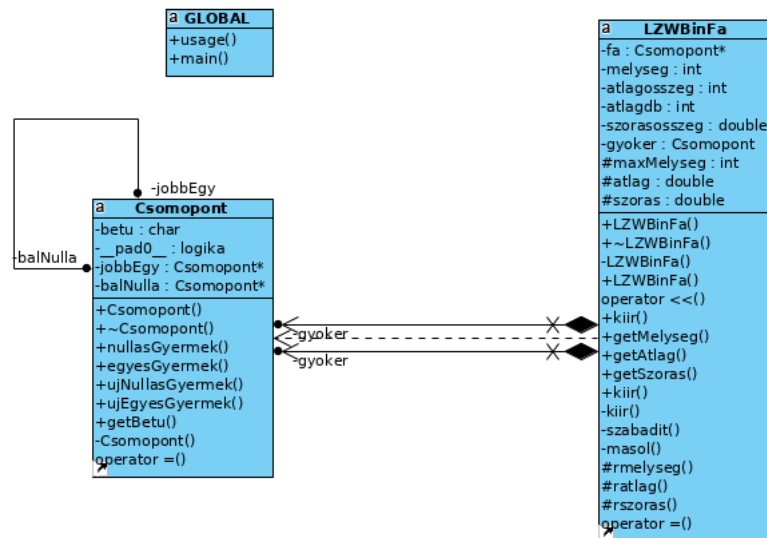
Az UML meghatározott jelölésrendszerrel dolgozik, így könnyen kezelhető és értelmezhető az így készített modell. Az osztályokat , interfészeket és azoknak a kapcsolatait írhatjuk le vele.

Az egyik legfontosabb tulajdonsága , hogy jól átlátható, a privát és public változók külön-külön vannak szedve. Ha ezt egy táblázatnak vesszük , akkor a legfelső sorban helyezkedik el a class neve, alatta az attribútumok és a legalján az operátorok.

Az "oszlopok" között nyilakat láthatunk , ezeket társításnak nevezzük. Az üres négyszög fejű nyíl a jelöli az aggregációt , ami egy adott osztály megalakulását jelenti. A teli négyszög fejű nyíl ennek az erős változatát jelöli , ezt kompozíciónak nevezzük, ekkor ezek a részek önmagában nem léteznek , együtt jönnek létre és együtt is szűnnek meg. A kör fejű nyíl elhatárolást jelent. A Nyilak mellett megjegyzéseket tekinthetünk meg.

Az UML generálásához a Visual Paradigmot használtam.

Lássuk :



14.2. Forward engineering UML osztálydiagram

UML-ben tervezzünk osztályokat és generáljunk belőle forrást!

Megoldás videó:

A feladatunk az volt, hogy UML-ben tervezzünk osztályokat és forrást generáljunk belőle.

A feladatban az előzőleg legenerált UML-t generáltam vissza forráskóddá.

Lássuk :

```

#ifndef LZWBINF_H
#define LZWBINF_H

class LZWBInFa {

private:
    Csomopont* fa;
    int melyseg;
    int atlagosszeg;
    int atlagdb;
    double szorasosszeg;
protected:
    Csomopont gyoker;
    int maxMelyseg;
    double atlag;
    double szoras;

public:
    LZWBInFa();
  
```

```
void ~LZWBinFa();

void kiir();

int getMelyseg();

double getAtlag();

double getSzoras();

void kiir(std::ostream& os);

private:
    LZWBinFa(const LZWBinFa& unnamed_1);

    void kiir(Csomopont* elem, std::ostream& os);

    void szabadit(Csomopont* elem);

protected:
    void rmelyseg(Csomopont* elem);

    void ratlag(Csomopont* elem);

    void rszoras(Csomopont* elem);
};

#endif

%%
    %}
```

14.3. BPMN

Rajzoljunk le egy tevékenységet BPMN-ben! <https://arato.inf.unideb.hu/batfai.norbert/UDPROG/deprecated/Prog>
(34-47 fólia)

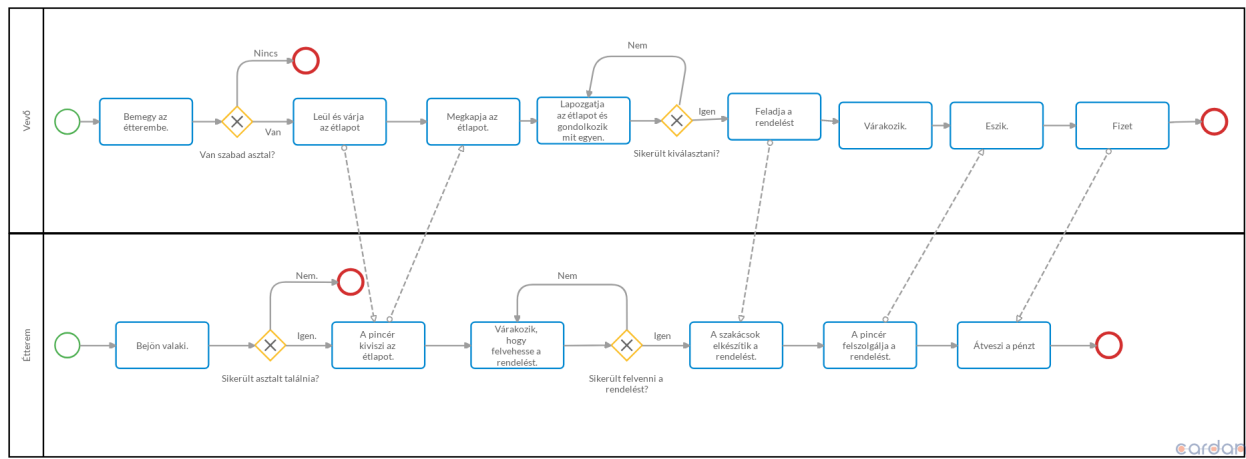
Megoldás videó:

A feladatunk az volt, hogy rajzoljunk le egy tevékenységet BPMN-ben.

De mi is az a BPMN ?

A BPMN (Business Process Model and Notation) egy folyamatábra lényegében , ami grafikai reprezentációk modellezésére szolgál üzleti folyamatok részére. Leegyszerűsítve a BPMN egy egyszerű grafikai modellező eszköz.

Nézzünk meg egy egyszerű projektet.



A képen egy éttermi étkezés folyamatát figyelhetjük meg. Először bemegyünk az étterembe és megnézzük, hogy van-e szabad asztal, ha van leülünk ha nincs akkor itt az esemény vége. Rendelünk a pincértől és a rendelésünket a szakácsok készítik el. A pincér felszolgálja a rendelést, én megeszem, kifizetem, a pincér átveszi a pénzt majd itt az esemény vége.

14.4. Egy esettan

A BME-s C++ tankönyv 14. fejezetét (427-444 elmélet, 445-469 az esettan) dolgozzuk fel!

Megoldás videó:

A feladatunk az volt, hogy dolgozzuk fel a BME-s C++ tankönyv kiadott fejezetét.

A feladatban látható egy UML diagramm kezelés.

Egy adatfolyamban található termékeket találhatunk benne. A Termékekhez különféle adatok vannak írva, például kód, ár, név és idő.

A termékek között találhatóak összetettebbek és egyszerűbbek.

A könyv elolvasva, a feladatok elkészítése még folyamatban.

15. fejezet

Helló, Chomsky!

15.1. Encoding

Fordítsuk le és futtassuk a Javat tanítók könyv MandelbrotHalmazNagyító.java forrását úgy, hogy a fájl nevekben és a forrásokban is meghagyjuk az ékezetes betűket! <https://www.tankonyvtar.hu/hu/tartalom/tkt/javat-tanitok-javat/adatok.html>

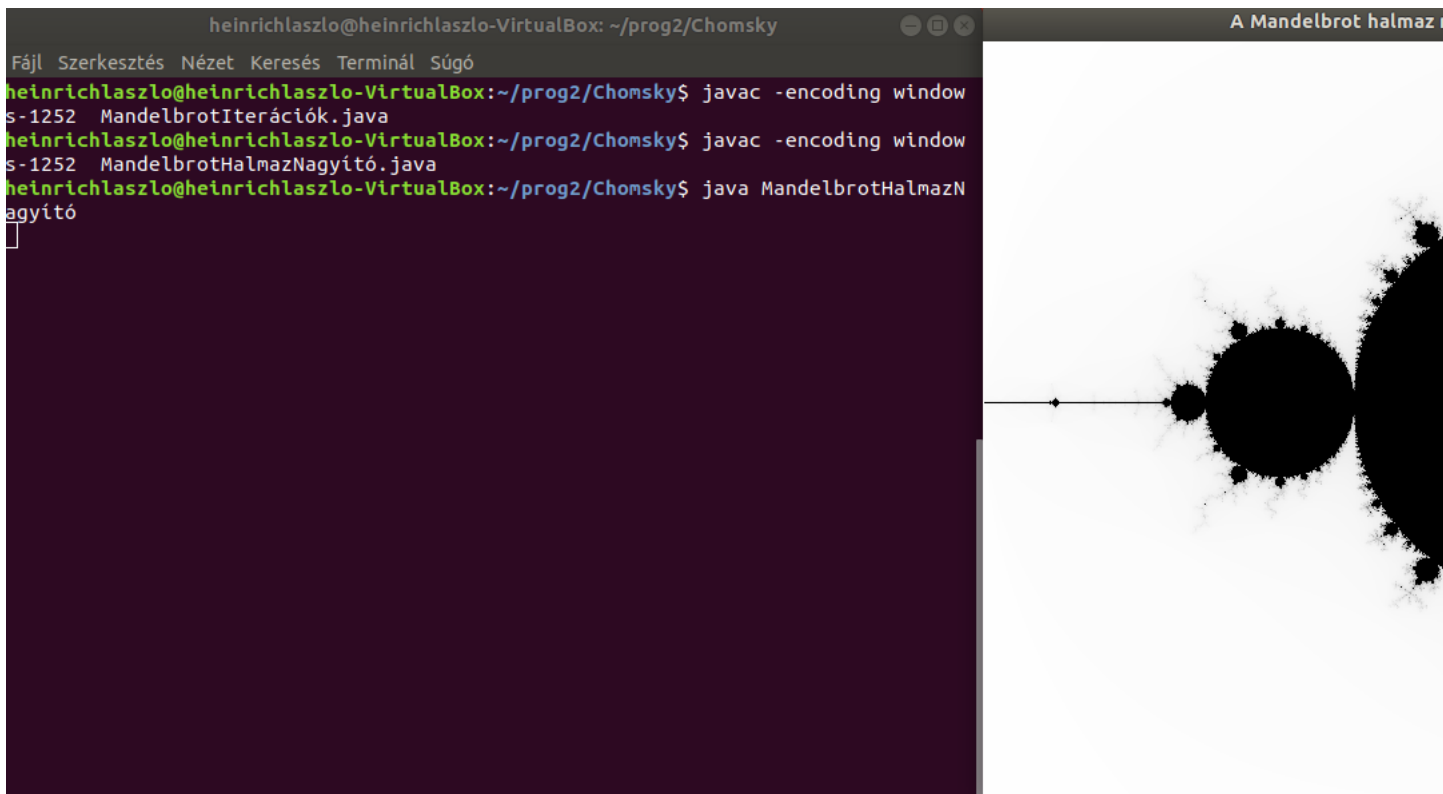
Megoldás videó:

A feladatunk az volt , hogy fordítsuk és futtassuk a Javat tanítók könyv MandelbrotHalmazNagyító.java forrását úgy, hpgy a fájl nevekben és magában a forrásban is meghagyjuk az ékezetes betűket.

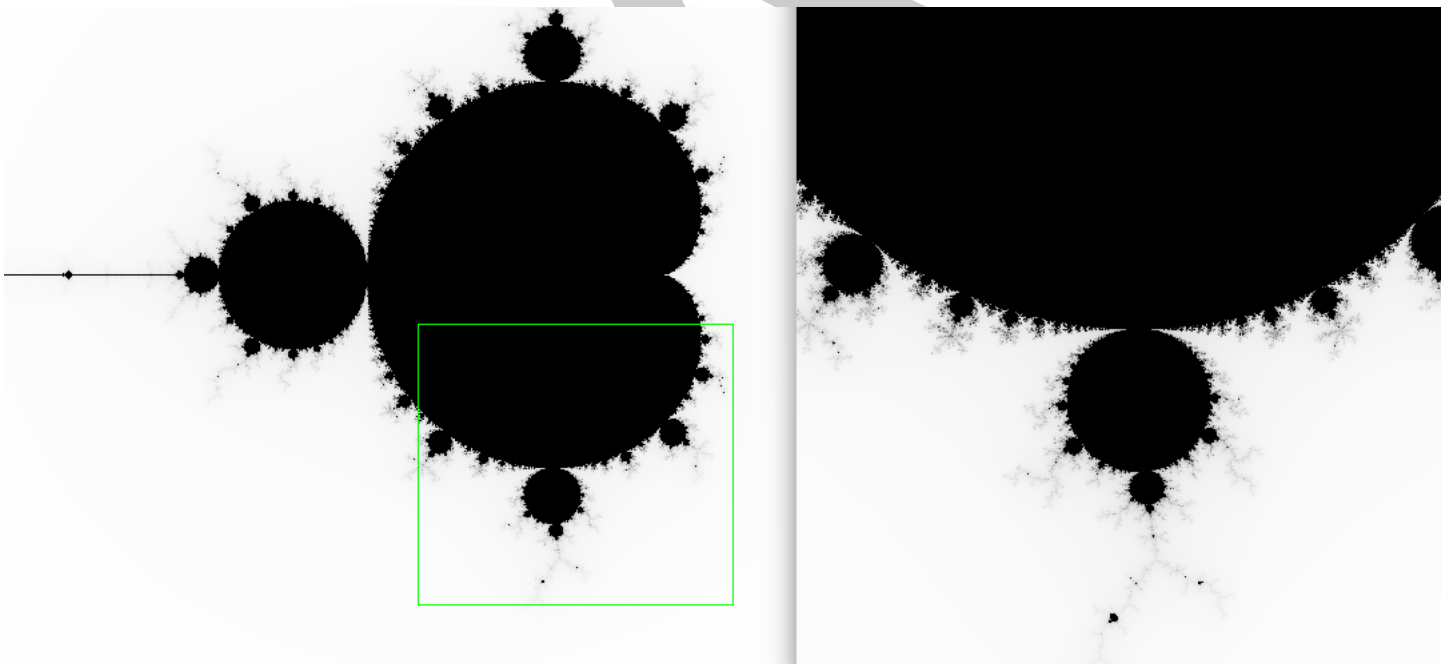
Fordítás után jó ár error fogad minket:

```
heinchlaszlo@heinchlaszlo-VirtualBox:~/prog2/Chomsky$ javac MandelbrotIterác
iók.java
MandelbrotIterációk.java:2: error: unmappable character for encoding UTF8
 * MandelbrotIterációk.java
      ^
MandelbrotIterációk.java:2: error: unmappable character for encoding UTF8
 * MandelbrotIterációk.java
      ^
MandelbrotIterációk.java:4: error: unmappable character for encoding UTF8
 * DIGIT 2005, Javat tanítók
      ^
MandelbrotIterációk.java:5: error: unmappable character for encoding UTF8
 * Botfai Norbert, nbatfai@inf.unideb.hu
      ^
MandelbrotIterációk.java:9: error: unmappable character for encoding UTF8
 * A nagyított Mandelbrot halmazok adott pontjában képes
      ^
MandelbrotIterációk.java:9: error: unmappable character for encoding UTF8
 * A nagyított Mandelbrot halmazok adott pontjában képes
      ^
MandelbrotIterációk.java:9: error: unmappable character for encoding UTF8
 * A nagyított Mandelbrot halmazok adott pontjában képes
      ^
```

A hiba üzenetekből azt tudjuk leszűrni , hogy a forráskód kódolásával van a gond. Pontosabban az , hogy nem található a karakter UTF-8 kódolás számára. Így hát a google segítségével , rákerestem olyan karak-
tekódolásra ami tartalmazza az ékezetes betűket, találtam is jópárat amik sajnos nem működtek. Végül
ráakadtam a egy olyanra amivel sikerült kiküszöbölni a problémát.



A nagyítás az egér segítségével kijelölt területen történik :



15.2. l334d1c4 5

Írj olyan OO Java vagy C++ osztályt, amely leet cipherként működik, azaz megvalósítja ezt a betű helyettesítést: <https://simple.wikipedia.org/wiki/Leet> (Ha ez első részben nem titted meg, akkor írasd ki és magyarázd meg a használt struktúratömb memórafoglalását!)

Megoldás forrása:

Megoldás videó:

A feladatunk az volt , hogy írjunk olyan Java vagy C++ programot ami megvalósítja , amely leet cipherként működik, azaz megvalósítja ezt a betű helyettesítést: <https://simple.wikipedia.org/wiki/Leet>.

Mi is ez a leet chipper?

A kódolási rendszer feltalálói Stephen McGreal és Alex Mole.

A leet kódolás szerint kódolja a megkapott szöveget, leegyszerűsítve az angol abc betűit helyettesíti a megfelelő ASCII karakterekkel.

Az interneten nagy népszerűségnek örvend ez a fajta kódolás. Használják például sajátos szimbólumszabályokon alapuló írásokon vagy csak , hogy egyszerűen kiéljék a kreativitásukat az emberek.

A szövegben felcserélt betűknek illetve számoknak nem csak 1 féle helyettesítése létezik. Pl egy betű lehet : l<, 1<, l<, l{ vagy l{- is.

Ezen a nyelven írt szoftverek sokáig az illegálisan munkálkodó emberek nyelve volt az interneten. Ugyanis könnyen hozhatóak vele létre rosszindulatú programok.

Fordítsuk és futtasuk a programunkat majd nézzünk meg egy egyszerű példát.

```
heInrichlaszlo@heInrichlaszlo-VirtualBox:~/prog2/Chomsky$ g++ leet.cpp -o leet
heInrichlaszlo@heInrichlaszlo-VirtualBox:~/prog2/Chomsky$ ./leet

Irja be a forditando szoveget.
>I am Batman.

I am Batman. -> I hm XhTmhn.
Do you want to enter another word y/n?
>n
heInrichlaszlo@heInrichlaszlo-VirtualBox:~/prog2/Chomsky$
```

15.3. Full screen

Készítsünk egy teljes képernyős Java programot! Tipp: https://www.tankonyvtar.hu/en/tartalom/tkt/javat-tanitok-javat/ch03.html#labirintus_jatek

Megoldás videó: <https://youtu.be/XpBnR31BRJY>

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

Az volt a feladatunk , hogy egy teljes képernyős Java programot. Egy már számunkra bevprogból és prog1ből is ismert feladat fullscreenes verzióját választottam , ami nem más mint a BouncingBall magyarul pattogó labda.

A labda mozgatása esetűben nem lényeges , annyi érdekesség van benne talán , hogy if nélkül történik. (mint anno bevprogon és prog1en.)

Annak érdekében , hogy full képernyőn fusson a programunk a konstruktorunkban be kell állítanunk , hogy frame undecorated legyen. Majd a teljes képernyő módba , setFullscreenWindows függvény segítségével jutunk el.

Lássuk a kódot :

```
import java.awt.*;
import java.awt.image.BufferStrategy;
public class BouncingBall {
    int x = 450;
    int y = 450;
    int radius = 50;
    int tempX, tempY;
    int maxX, maxY;
    private static DisplayMode[] BEST_DISPLAY_MODES = new DisplayMode[] {
        new DisplayMode(1920, 1080, 32, 0),
        new DisplayMode(1920, 1080, 16, 0),
        new DisplayMode(1920, 1080, 8, 0)
    };
    Frame mainFrame;
    public void move() {
        tempY = (int)(tempY + 1) % (int)(2 * (maxY - radius));
        y = y + (int)Math.pow(-1, Math.floor(tempY / (maxY - radius)));
        tempX = (int)(tempX + 1) % (int)(2 * (maxX - radius));
        x = x + (int)Math.pow(-1, Math.floor(tempX / (maxX - radius)));
    }
    public BouncingBall (int numBuffers, GraphicsDevice device) {
        try {
            GraphicsConfiguration gc = device.getDefaultConfiguration();
            mainFrame = new Frame(gc);
            mainFrame.setUndecorated(true);
            mainFrame.setIgnoreRepaint(true);
            device.setFullScreenWindow(mainFrame);
            if (device.isDisplayChangeSupported()) {
                chooseBestDisplayMode(device);
            }
            Rectangle bounds = mainFrame.getBounds();
            bounds.setSize(device.getDisplayMode().getWidth(), device.getDisplayMode().getHeight());
            maxX = device.getDisplayMode().getWidth();
            maxY = device.getDisplayMode().getHeight();
            tempX = x;
            tempY = y;
            mainFrame.createBufferStrategy(numBuffers);
            BufferStrategy bufferStrategy = mainFrame.getBufferStrategy();
            while(true) {
                Graphics g = bufferStrategy.getDrawGraphics();
                if (!bufferStrategy.contentsLost()) {
                    move();
                    g.setColor(Color.white);
                    g.fillRect(0,0,bounds.width, bounds.height);

                    g.setColor(Color.blue);
```

```
g.fillOval(x, y, radius, radius);
bufferStrategy.show();
g.dispose();
}
try {
Thread.sleep(5);
} catch (InterruptedException e) {}
}
} catch (Exception e) {
e.printStackTrace();
} finally {
device.setFullScreenWindow(null);
}
}
private static DisplayMode getBestDisplayMode(GraphicsDevice device) {
for (int x = 0; x < BEST_DISPLAY_MODES.length; x++) {
DisplayMode[] modes = device.getDisplayModes();
for (int i = 0; i < modes.length; i++) {
if (modes[i].getWidth() == BEST_DISPLAY_MODES[x].getWidth()
&& modes[i].getHeight() == BEST_DISPLAY_MODES[x].getHeight()
&& modes[i].getBitDepth() == BEST_DISPLAY_MODES[x].getBitDepth()) {
return BEST_DISPLAY_MODES[x];
}
}
}
return null;
}
public static void chooseBestDisplayMode(GraphicsDevice device) {
DisplayMode best = getBestDisplayMode(device);
if (best != null) {
device.setDisplayMode(best);
}
}
public static void main(String[] args) {
try {
int numBuffers = 2;
GraphicsEnvironment env = GraphicsEnvironment.
getLocalGraphicsEnvironment();
GraphicsDevice device = env.getDefaultScreenDevice();
BouncingBall ball = new BouncingBall(numBuffers, device);
} catch (Exception e) {
e.printStackTrace();
}
System.exit(0);
}
}
```

Forditjuk majd futtatjuk a kódot :

```
heinrichlaszlo@heinrichlaszlo-VirtualBox:~/prog2/Chomsky$ javac BouncingBall.java  
heinrichlaszlo@heinrichlaszlo-VirtualBox:~/prog2/Chomsky$ java BouncingBall
```

Végezetül , nézzük a teljes képernyős formáját.



15.4. Hiba-visszaterjesztéses perceptron

C++

Megoldás videó: <https://youtu.be/XpBnR31BRJY>

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

Tanulságok, tapasztalatok, magyarázat...

A Neurális OR, AND és EXOR kapu feladatnál már találkozhattunk a neuron és a gépi tanulás fogalmával. A perceptron leegyszerűsítve nem más mint ezen neuron mesterséges intelligenciában használt változata. Tanulásra képes, a bemenő 0-k és 1-esek sorozatából mintákat tanul meg és súlyozott összegzést végez. A következő feladat során egy ilyen perceptront fogunk elkészíteni aminek alapvetően a feladata az, hogy a `mandelbrot.cpp` programunk által létrehozott Mandelbrot-halmazt ábrázoló PNG kép egyik színkódját vegye és az a színkód legyen a többretegű neurális háló inputja. !!!!! ÁTFOGALMAZNI!!!!

```
#include <iostream>
#include "mlp.hpp"
#include <png++/png.hpp>
```

Includoljuk az `iostream`, az `mlp` és a `png++/png` könyvtárakat. A `mlp` könyvtárra azért van szükségünk mert több rétegű perceptront fogunk létrehozni. A `png++/png` könyvtár a PNG kiterjesztésű képállományokkal való munkát teszi számunkra lehetővé.

```
using namespace std;

int main(int argc, char **argv)
{
    png::image <png::rgb_pixel> png_image(argv[1]);

    int size = png_image.get_width()*png_image.get_height();

    Perceptron *p = new Perceptron(3, size, 256, 1);
```

A `main` függvényünk első sorában megmondjuk, hogy a képállományunk beolvasása az `1es` parancsori argumentum alapján történik. Eltároljuk egy segédváltozóban a kép méretét a `get_width` és a `get_height` szorzatát, majd létrehozuk a perceptront a `new` operátor segítségével.

```
double* image = new double[size];

for(int i=0; i<png_image.get_width(); ++i)
    for(int j=0; j<png_image.get_height(); ++j)
        image[i*png_image.get_width()+j] = png_image[i][j].red;

double value = (*p)(image);

cout<<value;

delete p;
delete [] image;
```

Az egyik `for` ciklussal végig megyünk a kép szélességét alkotó pontokon, majd a másik `for` ciklussal végig megyünk a magasságát alkotó pontokon. Az `image`-ben fogjuk tárolni a vörös képpontokat. A `value` értéke adja meg a Perceptron `image`-ra történő meghívását, ami egy `double` típusú változót tárol. Kiírjuk és töröljük azokat az elemeket amiket már nem használunk, így az addig lefoglalt memória egységeket újra használhatóvá tettük.

Fordítjuk és futtatjuk a képen látható módon:

16. fejezet

Helló, Chomsky!

16.1. Változó argumentumszámú ctor

Készítsünk olyan példát, amely egy képet tesz az alábbi projekt Perceptron osztályának bemenetére és a Perceptron ne egy értéket, hanem egy ugyanakkora méretű „képet” adjon vissza. (Lásd még a 4 hét/Perceptron osztály feladatot is.)

Megoldás videó:

A feladat nagyban megegyezik, az előző Percetronos feladattal.

Nézzük a kódot :

```
#include <iostream>
#include "mlp.hpp"
#include <png++/png.hpp>
```

A programunk elején includoljuk a számunkra szükséges könyvtárakat. Az mlp(multi layer perceptron) és png++/png-re a többretegű perceptron létrehozásánál lesz szükségünk. A PNG könyvtárunk a PNG képálmányokkal való munkát teszi nekünk lehetővé.

```
using namespace std;
int main(int argc, char **argv)
{
    png::image <png::rgb_pixel> png_image(argv[1]);
    int size = png_image.get_width()*png_image.get_height();
    Perceptron *p = new Perceptron(3, size, 256, 1);
```

Elérkezünk a main függvényhez , először is a képálmány kerül beolvasásra, az 1-es parancssori argumentum alapján.

A get_width és a get_height szorzatából megkapjuk a kép méretét.

Mindezek után létrehozunk egy új perceptront a new szócska (operátor) segítségével. Paraméterként kapja az 1. réteg neuronjai az inputrétegben , majd ezt követően 2.réteg neuronjai az inputrétegben , majd az utolsó az eredmény.

```
double* image = new double[size];
for(int i=0; i<png_image.get_width(); ++i)
```

```
for(int j=0; j<png_image.get_height(); ++j)
image[i*png_image.get_width()+j] = png_image[i][j].red;
```

Deklarálunk egy mutatót double típussal.

Majd jönnek a for ciklusok. Ezek a for ciklusok egike a képet alkotó pontok szélességén , míg a másik a képet alkotó pontok magasságán. A végig menések után , az image-ben tárolni fogjuk a képállomány vörös színtonponenseit.

```
double* newPicture = (*p) (image); // eddig: double value = (*p) ( ←-
image);
for (int i = 0; i<png_image.get_width(); ++i)
for (int j = 0; j<png_image.get_height(); ++j)
png_image[i][j].red = newPicture[i*png_image.get_width()+j];
png_image.write("kimeneti.png");
delete p;
delete [] image;
```

Egy double csillag típus segítségével nem egy értéket akarunk kiírni, hanem magát a képet.

Majd két for ciklus segítségével végigmegyünk , az eredeti kép szélességén és magasságán. Ezeket az színadatokat kapja meg az új képünk. Ezt követően a write függvénnyel létrehozuk kimeneti néven az új képállományunkat. A kimeneti állományunk png formátumu lesz.

```
double* operator() ( double image [] )
{
```

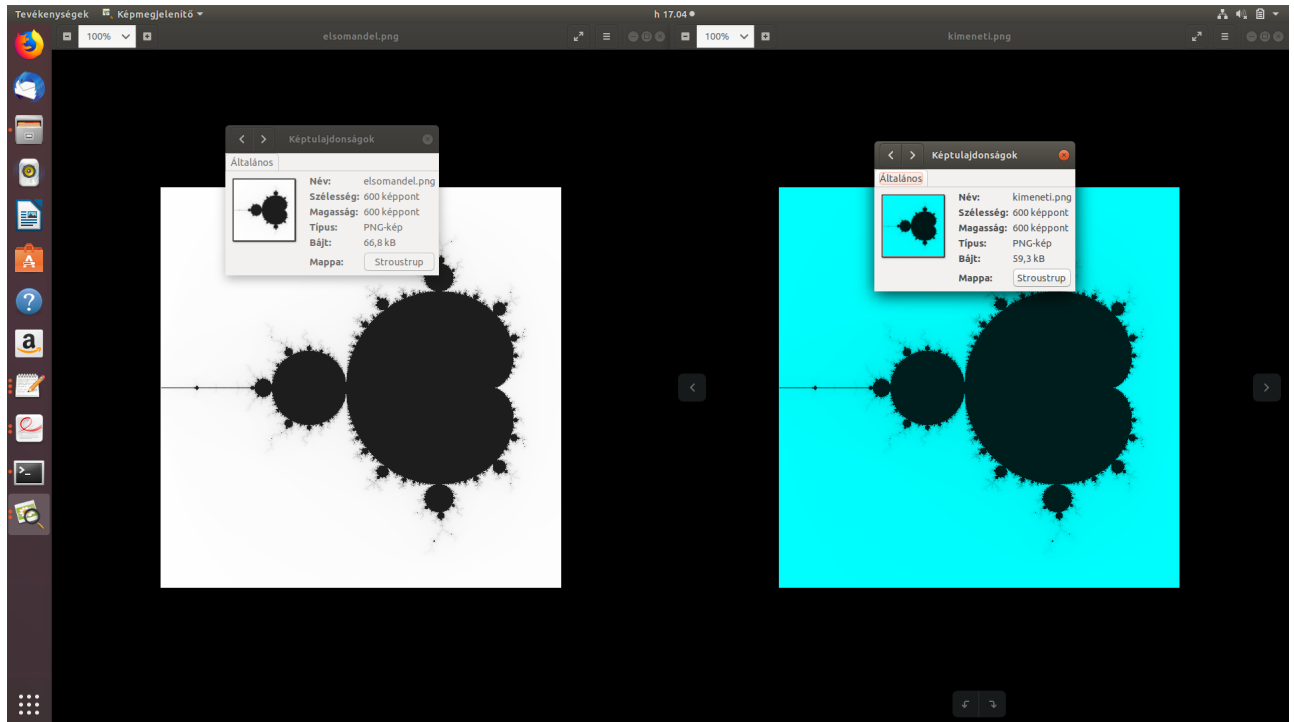
Kicsit bele kell nyúlni az mlp.hpp fájlunkba is.

Például az () operátor működésébe ami mostantól nem egy double értéket , hanem double pointert ad vissza.

Fordítjuk majd futtatjuk a programunkat .

```
heinrichlaszlo@heinrichlaszlo-VirtualBox:~/prog2/Stroustrup$ g++ mlp.hpp percept
ron.cpp -o perceptron -lpng -std=c++11
heinrichlaszlo@heinrichlaszlo-VirtualBox:~/prog2/Stroustrup$ ./perceptron elsoma
ndel.png
heinrichlaszlo@heinrichlaszlo-VirtualBox:~/prog2/Stroustrup$
```

Lássuk az eredményt.



16.2. JDK osztályok

Írjunk olyan Boost C++ programot (indulj ki például a fénykardból) amely kilistázza a JDK összes osztályát (miután kicsomagoltuk az src.zip állományt, arra ráengedve)!

Megoldás videó:

A feladatot a leírás alapján, a fénykard program alapján csináltam. Az alábbi linken tekinthetjük meg Bátfai tanárúr fénykard.cpp kódját : <https://github.com/nbatfai/future/blob/master/cs/F7/fenykard.cpp>

```
#include <iostream>
#include <string>
#include <map>
#include <iomanip>
#include <fstream>
#include <vector>

#include <boost/filesystem.hpp>
#include <boost/filesystem/fstream.hpp>
#include <boost/program_options.hpp>
#include <boost/tokenizer.hpp>
#include <boost/date_time/posix_time/posix_time.hpp>

using namespace std;
using namespace boost::filesystem;

int szamlalo = 0;
```

```

void read_file ( boost::filesystem::path path, std::vector<std::string> <←
    acts )
{
    if ( is_regular_file ( path ) ) {
        std::string ext ( ".java" );
        if ( !ext.compare ( boost::filesystem::extension ( path ) ) <←
            ) {
            cout<<path.string()<<'\\n';
            std::string actproppath = path.string();
            std::size_t end = actproppath.find_last_of ( "/" ) <←
                ;
            std::string act = actproppath.substr ( 0, end );
            acts.push_back(act);
            szamlalo++;
        }

        } else if ( is_directory ( path ) )
            for ( boost::filesystem::directory_entry & entry : <←
                boost::filesystem::directory_iterator ( path ) )
                read_file ( entry.path(), acts );
    }

int main ( int argc, char *argv[] )
{
    string path="src";
    vector<string> acts;
    read_file(path,acts);
    cout<<"szamlalo: "<<szamlalo<< std::endl;
}

```

Röviden magáról a kódról :

Az src mappán belül elhelyezkedő .java végződésű fájlokat számolhatjuk meg vele. Kiírja, hogy melyek ezek a fájlok src mappán belül és hol találhatóak. Majd utolsó sorban a végeredményt. A végeredmény az a szám , hogy összesen mennyit talált a számláló.

A segítségként megadott fenykard.cpp-ből kiindulva a read_acts részeket kell átírni és hozzáadni egy szamlalo integert. Majd ha a .java-val találkozunk egyel növeljük a számlálót , és kiírjuk a fájlok elérésének az útvonala és a neve. Majd végül maga a számláló.

Fordítjuk majd futtatjuk :

```

hejnrichlaszlo@hejnrichlaszlo-VirtualBox:~/prog2/Stroustrup$ g++ fenykardJDKosztaly.cpp -o fenykard -lboost_system -lboost_filesystem -lboost_program_options -std=c++14
hejnrichlaszlo@hejnrichlaszlo-VirtualBox:~/prog2/Stroustrup$ ./fenykard

```

Majd nézzük az eredményt :

```
heinrichlaszlo@heinrichlaszlo-VirtualBox: ~/prog2/S
Fájl Szerkesztés Nézet Keresés Terminál Súgó
src/jdk.jdi/com/sun/jdi/ReferenceType.java
src/jdk.jdi/com/sun/jdi/ClassObjectReference.java
src/jdk.jdi/com/sun/jdi/BooleanValue.java
src/jdk.jdi/com/sun/jdi/Accessible.java
src/jdk.jdi/com/sun/jdi/InternalException.java
src/jdk.jdi/com/sun/jdi/LongValue.java
src/jdk.jdi/com/sun/jdi/InvalidLineNumberException.java
src/jdk.jdi/com/sun/jdi/request/StepRequest.java
src/jdk.jdi/com/sun/jdi/request/ExceptionRequest.java
src/jdk.jdi/com/sun/jdi/request/MethodEntryRequest.java
src/jdk.jdi/com/sun/jdi/request/BreakpointRequest.java
src/jdk.jdi/com/sun/jdi/request/package-info.java
src/jdk.jdi/com/sun/jdi/request/VMDeathRequest.java
src/jdk.jdi/com/sun/jdi/request/MonitorContentedEnteredRequest.java
src/jdk.jdi/com/sun/jdi/request/EventRequest.java
src/jdk.jdi/com/sun/jdi/request/DuplicateRequestException.java
src/jdk.jdi/com/sun/jdi/request/MonitorContentedEnterRequest.java
src/jdk.jdi/com/sun/jdi/request/MonitorWaitRequest.java
src/jdk.jdi/com/sun/jdi/request/ThreadStartRequest.java
src/jdk.jdi/com/sun/jdi/request/AccessWatchpointRequest.java
src/jdk.jdi/com/sun/jdi/request/MethodExitRequest.java
src/jdk.jdi/com/sun/jdi/request/ThreadDeathRequest.java
src/jdk.jdi/com/sun/jdi/request/MonitorWaitedRequest.java
src/jdk.jdi/com/sun/jdi/request/EventRequestManager.java
src/jdk.jdi/com/sun/jdi/request/ClassUnloadRequest.java
src/jdk.jdi/com/sun/jdi/request/WatchpointRequest.java
src/jdk.jdi/com/sun/jdi/request/ModificationWatchpointRequest.java
src/jdk.jdi/com/sun/jdi/request/InvalidRequestStateException.java
src/jdk.jdi/com/sun/jdi/request/ClassPrepareRequest.java
src/jdk.jdi/com/sun/jdi/NativeMethodException.java
src/jdk.jdi/com/sun/jdi/InvalidCodeIndexException.java
src/jdk.jdi/com/sun/jdi/VirtualMachine.java
src/jdk.jdi/com/sun/jdi/CharValue.java
src/jdk.jdi/com/sun/jdi/VMMismatchException.java
src/jdk.jdi/com/sun/jdi/IncompatibleThreadStateException.java
src/jdk.jdi/com/sun/jdi/FloatValue.java
src/jdk.jdi/com/sun/jdi/Type.java
src/java.management.rmi/module-info.java
src/java.management.rmi/com/sun/jmx/remote/protocol/rmi/ClientProvider.java
src/java.management.rmi/com/sun/jmx/remote/protocol/rmi/ServerProvider.java
src/java.management.rmi/com/sun/jmx/remote/internal/rmi/ProxyRef.java
src/java.management.rmi/com/sun/jmx/remote/internal/rmi/RMIExporter.java
src/java.management.rmi/javax/management/remote/rmi/RMIConnector.java
src/java.management.rmi/javax/management/remote/rmi/RMIServer.java
src/java.management.rmi/javax/management/remote/rmi/NoCallStackClassLoader.java
src/java.management.rmi/javax/management/remote/rmi/RMIServerImpl_Stub.java
src/java.management.rmi/javax/management/remote/rmi/RMIServerImpl.java
src/java.management.rmi/javax/management/remote/rmi/RMIConnectorServer.java
src/java.management.rmi/javax/management/remote/rmi/RMIJRMPServerImpl.java
src/java.management.rmi/javax/management/remote/rmi/RMIConnectionImpl_Stub.java
src/java.management.rmi/javax/management/remote/rmi/RMIIOPServerImpl.java
src/java.management.rmi/javax/management/remote/rmi/RMIConnectionImpl.java
src/java.management.rmi/javax/management/remote/rmi/RMIConnection.java
szamlalo: 17593
heinrichlaszlo@heinrichlaszlo-VirtualBox: ~/prog2/Stroustrup$
```

16.3. Hibásan implementált RSA törése és Összefoglaló

Készítsünk betű gyakoriság alapú törést egy hibásan implementált RSA kódoló: https://arato.inf.unideb.hu/~batfai.norbert/UDPROG/deprecated/Prog2_3.pdf (71-73 fólia) által készített titkos szövegen.

Megoldás videó:

Megoldás forrása: <https://gitlab.com/heinrichlaszlo>

Tanulságok, tapasztalatok, magyarázat...

Tutor: [GitHub](#)

Mi is az az RSA? Az RSA nem más mint egy titkosító eljárás. Az eljárás az algoritmus alkotóiról kapta a nevét, R mint Ron Rivest, A mint Adi Shamir és L mint Len Adleman. Ezt az eljárást 1978-ban publikálták.

Működése a Fermat-tétel szerint működik. Egyik legérdekesebb tulajdonsága, hogy egy nyílt és egy titkos kulccsal dolgozik. Mindkét kulcs egy-egy számpárt alkot, így teljesen ketté lesz választva a titkosítás és a törés folyamara. A kódolás és a dekódolás nem lesznek ugyanazok, így az egyik a másiktól nem lesz meghatározható.

Forrás : [katt](#)

Említsük még meg a BigInteger osztályt is. Ez az osztály a JDK részét képezi, jelenleg a java.math csomagban található meg. Általában kriptográfiai kódolók és törők elkészítéséhez használják. Magát a típust egy egész értékből a 32-bites egészéből egy skálázó faktor alkotja.

Lássuk a programot :

```
public class RSA{
    public static void main(String[] args)
    {
        int meretBitekben = (int) (700 * (int) (java.lang.Math.log( (double ↵
            ) 10)) / java.lang.Math.log((double) 2));

        System.out.println("Méret bitekben: ");
        System.out.println(meretBitekben);

        java.math.BigInteger p_i = new java.math.BigInteger(meretBitekben, 100, ↵
            new java.util.Random() );

        System.out.println("p_i");
        System.out.println(p_i);
        System.out.println("p_i hexa");
        System.out.println(p_i.toString(16));

        java.math.BigInteger q_i = new java.math.BigInteger(meretBitekben, 100, ↵
            new java.util.Random() );

        System.out.println("q_i");
        System.out.println(q_i);

        java.math.BigInteger m_i =p_i.multiply(q_i);

        System.out.println("m_i");
        System.out.println(m_i);

        java.math.BigInteger z_i = p_i.subtract(java.math.BigInteger.ONE). ↵
            multiply(q_i.subtract(java.math.BigInteger.ONE));

        System.out.println("z_i");
        System.out.println(z_i);

        java.math.BigInteger d_i;
```



```

do {
    do {
        d_i = new java.math.BigInteger(meretBitekben, new java.util.Random ←
        ());

        } while(d_i.equals(java.math.BigInteger.ONE));
    } while(!z_i.gcd(d_i).equals(java.math.BigInteger.ONE));

    System.out.println("d_i");
    System.out.println(d_i);

    java.math.BigInteger e_i = d_i.modInverse(z_i);

    System.out.println("e_i");
    System.out.println(e_i);
}
}

```

Fordítás és futtatás után megkapjuk nyilvános és titkos kulcsainkat:

```

heinchlaszlo@heinchlaszlo-VirtualBox:~/prog2/Stroustrup$ javac RSA.java
heinchlaszlo@heinchlaszlo-VirtualBox:~/prog2/Stroustrup$ java RSA
Méret bitekben:
2019
p_i
33076538267449209987410353902236489690097697720751773362946745758120951882164249013638618823122621155829411089743385943
24375862445570109639132876374288436474878226996931497986799134872549991590182072216304651114366077033489273909122308582
29648759759028552176977251284200436720805504597282703165413740625017980040364960999461893784973782064915772551487524733
p_i hexa
4655a5b917618cebddd83b8bc1bf223049f21860a88df10a76e0acff664ce3c47d90a627dc2d3f94dd6acb62f1b15e1c8ba6135d48f5d448686d64fe
826ba26998db6bdec5e2692f0b4c6767c4142412b220a8b0209340bd69a883f5e42a69b08ef7c67eab3f66d528559a1b8d7fdaf19affba56b33991c
fd9f22457c98c195cd0dde9d95cf5706c7ba394c0760877ff912425cc47490146876eb0b9bc8d37f83d88bce53d181733
q_i
42270129703550942894097447734009427972791504111936147129664652342639050384053926746353693601851410267979180276338320888
60068772969729694754516466634696463797612181393583177707605192536121554546510381974801876345633678898112
05279318857075833005584991715963164997786018767541548423459685978834729524111212798730645720665782102473914908825448078
m_i
13981495627095442928525121039841852211005002853265223256125955466799001658603865757716696939601882767629656535371166230
93459892260027876752252872180905563147815941130547227035647717334192622124175993314813946221951872035950865988966551550
25538156628353455182828711328902300084280897007446495343267475619189647488393612257599448141114406521968156830728079982
95751302396533022687014974453956199174197662714017522130875092167006145767975027167518923170661851953677037964807343326
93691568374999852931368526369461318747173110443225092745802616868806359621906039615238278998369907277021368599311067350
07511914219765164692400288602223497765014506839750701395261200992610565315136723375598981077045032252340596162820150703
z_i
13981495627095442928525121039841852211005002853265223256125955466799001658603865757716696939601882767629656535371166230
93459892260027876752252872180905563147815941130547227035647717334192622124175993314813946221951872035950865988966551550
25538156628353455182828711328902300084280897007446495343267475619189647488393612257599448141114406521968156830728079982
29071592395004207608998611994779570282179335834812596016894084566983483586217427244394673430347613867763377147739020319
47337420060067081249867933730580103330206677330633043360870713661285265265062321114076097140119384749718940587244123656
26725753175913339069970286966206311849780858597234812660995162465514920553398741450203924681403358355465993033092031384
d_i
57503577746060713716316732225673771191294677033004118645938195689302900543758616576958077756965974889153519337269710377
38707959293508419361203147224768782082560081731961821484559781592797141334282122287823149882585239508332435785716002986
50377175806056102398153742918087123027290544761581499652473739153470135834818192072483233723418320867529067526004836107
e_i
41049091062470003185593256027428240473120750443162346525996833131911950941412120610979270635228721237374140657912170575
02417592909060328374669987808088002302695321118508162459475158630785122234840055413062968817762816398766083216589405122
18934970753519306939213987405819271690632521930584310551045043708860057349283406451216988862097021412733091666563073389
27460558029787806059686509589247650760026071814362075087113366286828788892075626227113011524067425257554325027153387610
36909649658933430745718410869170174489789733718737718233836800106802866287635135196304871221597851853958110475717953328
54497758411333693504227753758626479673893574204898440157266520426242363048703106355388205222960028545410747700060973842
public class RSA2{

static class KulcsPar{

```

```
java.math.BigInteger d, e, m;

public KulcsPar() {

    int meretBitekben = 700 * (int) (java.lang.Math.log((double) 10) / ←
        java.lang.Math.log((double) 2));

    java.math.BigInteger p = new java.math.BigInteger(meretBitekben, 100, ←
        new java.util.Random());

    java.math.BigInteger q = new java.math.BigInteger(meretBitekben, 100, ←
        new java.util.Random());

    m = p.multiply(q);

    java.math.BigInteger z = p.subtract(java.math.BigInteger.ONE). ←
        multiply(q.subtract(java.math.BigInteger.ONE));

    do {
        do {
            d = new java.math.BigInteger(meretBitekben, new java.util. ←
                Random());
        } while (d.equals(java.math.BigInteger.ONE));
    } while (!z.gcd(d).equals(java.math.BigInteger.ONE));

    e = d.modInverse(z);
}
}
```

Most , hogy már megvannak a kulcsaink nézzük meg titkosítást. A modulust a p és q számok szorzatából állítjuk elő. Ezek a fenti , KulcsPar osztályban kerülnek létrehozásra. m lesz a modulo, e a kitevő míg d lesz e inverze.

```
public static void main(String[] args) {

    KulcsPar jSzereplo = new KulcsPar();

    String tisztaSzoveg = "Ez a titkosított tartalom.";

    byte[] buffer = tisztaSzoveg.getBytes();
    java.math.BigInteger[] titkos = new java.math.BigInteger[buffer.length ←
        ];

    for(int i = 0; i < titkos.length; i++)
    {
        titkos[i] = new java.math.BigInteger(new byte[]{buffer[i]});
        titkos[i] = titkos[i].modPow(jSzereplo.e, jSzereplo.m);
    }
}
```



```
for(int i = 0; i < titkos.length; i++)
{
    titkos[i] = titkos[i].modPow(jSzereplo.d, jSzereplo.m);
    buffer[i] = titkos[i].byteValue();
}
System.out.println(new String(buffer));
}
```

A main-ben megtörténik a titkosítás. Megadjuk a tiszta szöveget. Majd következik egy for ciklus amivel végig megyünk a titkos szövegen és feltöltjük az eredeti "tiszta" szövegünk ASCII kódjával. A for cikluson belül találkozhatunk a modPow függvénnyel, ennek a függvénynek megadjuk a hatványkitevőt, és modulust amivel BigInteger-t kapunk.

```
for(int i = 0; i < titkos.length; i++)
{
    titkos[i] = titkos[i].modPow(jSzereplo.d, jSzereplo.m);
    buffer[i] = titkos[i].byteValue();
}
System.out.println(new String(buffer));
}
```

A titkosítás után következik a 'visszatitkosítás', Az pedig hasonlóan történik mint az előzőekben. Mivel a modPow függvény első paraméterként inverzet kap ezért visszafelé működik az algoritmus, a BigInteger-ből ismét ASCII kódsorozat lett. A bufferbe a visszafejtett szöveg kerül be, majd ezt iratjuk ki a képernyőre.

Fordítás és futtatás után megkapjuk eredeti szövegünket:

```
heinrichlaszlo@heinrichlaszlo-VirtualBox:~/prog2/Stroustrup$ javac RSA2.java
heinrichlaszlo@heinrichlaszlo-VirtualBox:~/prog2/Stroustrup$ java RSA2
Ez a titkosított tartalom.
```

17. fejezet

Helló, Gödel!

17.1. Alternatív Tabella rendezése

Mutassuk be a https://progpater.blog.hu/2011/03/11/alternativ_tabella a programban a java.lang Interface comparable<T> szerepét!

Megoldás videó:

Megoldás forrása: <https://gitlab.com/heinrichlaszlo/prog2forras/blob/master/Arroway/PolarGenerator.java>

A feladatunk az volt, hogy az alternatív tabella programban mutassuk be, az java.lang Interface comparable<T> szerepét.

De mielőtt , ezt megtennénk , ismerjük meg mi is az az alternatív tabella?

Az alternatív tabella nem más, mint az egyes futtbalbajnokságokhoz készített sorrend, ami nem a megszokott pontozás szerint számolja a csapatok sorrendjét, hanem azt veszi figyelembe, hogy az adott csapat , melyik másik csapattal szemben érte el az adott eredményt.

Az ilyen alternatív tabellák elkészítésének egyik módja, hogy a Google PageRank algoritmuást használjuk fel ehhez.

Leegyszerűsítve, az alternatív sorrend az alapján alakul ki, hogy az a csapat kerül előrébb, mely előrébb lévő csapat ellen szerez pontot.

A bevezető forrása : https://hu.wikipedia.org/wiki/Alternat%C3%ADv_tabella

Most, hogy már tudjuk mi is az az alternatív tabella nézzük meg mi is az az interface amire a feladatunk rákérdezett:

```
class Csapat implements Comparable<Csapat>
{
    protected String nev;
    protected double ertek;

    public Csapat(String nev, double ertek) {
        this.nev = nev;
        this.ertek = ertek;
    }
}
```

```
public int compareTo(Csapat csapat) {  
    if (this.ertek < csapat.ertek) {  
        return -1;  
    } else if (this.ertek > csapat.ertek) {  
        return 1;  
    } else {  
        return 0;  
    }  
}
```

Ha egy összetettebb adatszerkezetnél a sort függvénnyel való rendezés hibákhoz vezetne, ezért a `Comparable<T>` interface segítségével tudjuk ezeket kiküszöbölni.

A `Comparable<T>` interface, más néven természetes rendezés, egy teljes rendezést tesz az osztály objektumaira.

A `java.lang` csomag része, de ezt a csomagot nem szükséges importálni, mert ez az importálás automatikusan történik. Ez az interface egyetlen egy metódust tartalmaz, ami az `compareTo()`, amit más néven természetes összehasonlító metódusnak is neveznek.

A `compareTo()` összehasonlíja az objektumokat és három értékkel térhet vissza. Negatív értékkel, (a kódban ez -1) ha az objektum nagyobb mint a kiválasztott objektum. Nullával tér vissza ha a két objektum egyenlő. Pozitív értékkel (esetünkben 1) tér vissza, ha az objektum értéke kisebb mint a kiválasztott objektumé. Megadjuk, hogy a csapat melyik tagja alapján rendezzen, esetünkben a doubleként megadott érték alapján történik a rendezés. Majd a kiválasztott objektum nevéhez tartozó értéket hasonlítja össze a többi névhez tartozó értékkel. A csapat osztályban létrehozott értékeket hasonlítjuk össze, a többi névhez tartozó értékkel. Ezután a csapat classban létrehozott elemeket rendezett sorrendben adja vissza a sort függvény.

Az `AlternativTabella.java` programunkban a `rendezCsapatok` függvény hívja, majd meg a class-t de ez már 0 ; 1 és -1-ekből áll és rendezve lesz.

A `compareTo()` függvényben az objektumok sorrendjét az objektumok összehasonlításának sorrendje határozza meg.

A `Comparable<T>` interface implementálása a csapat nevű osztályban történik a következő módon :

```
java.util.List<Csapat> rendezettCsapatok = java.util.Arrays.asList( ←  
    csapatok);  
java.util.Collections.sort(rendezettCsapatok);
```

Fordítsuk és futtasuk :

```
heinchlaszlo@heinchlaszlo-VirtualBox:~/prog2/gödel$ javac AlternativTabella.java
heinchlaszlo@heinchlaszlo-VirtualBox:~/prog2/gödel$ java AlternativTabella
iteracio...
norma = 0.05918759643574294
qsszeg = 1.0
iteracio...
norma = 0.014871507967965858
qsszeg = 0.9999999999999999
iteracio...
norma = 0.006686056768932613
qsszeg = 1.0
iteracio...
norma = 0.007776749198562133
qsszeg = 1.0
iteracio...
norma = 0.007661483555477314
qsszeg = 0.9999999999999999
iteracio...
norma = 0.007501657116770109
qsszeg = 0.9999999999999998
iteracio...
norma = 0.007532790726590474
qsszeg = 0.9999999999999998
iteracio...
norma = 0.007538144256251714
qsszeg = 0.9999999999999998
iteracio...
norma = 0.007535025315190922
qsszeg = 1.0
iteracio...
norma = 0.00753510193655861
qsszeg = 0.9999999999999997
iteracio...
norma = 0.0075353297234758716
qsszeg = 0.9999999999999998
iteracio...
norma = 0.007535284725802345
qsszeg = 0.9999999999999999
iteracio...
norma = 0.007535275242694286
qsszeg = 0.9999999999999996
iteracio...
norma = 0.0075352801728795745
qsszeg = 0.9999999999999999
iteracio...
norma = 0.007535280078749908
qsszeg = 0.9999999999999998
iteracio...
norma = 0.007535279718816022
qsszeg = 0.9999999999999998
iteracio...
norma = 0.007535279786665789
qsszeg = 0.9999999999999998
iteracio...
norma = 0.007535279802432719
```

```
csapatok rendezve:
```

```
-  
Videoton  
40  
Videoton  
0.0841  
-  
Ferencváros  
34  
Debreceni VSC  
0.0828  
-  
Paks  
31  
Paksi FC  
0.0725  
-  
Debreceni VSC  
31  
BFC Siófok  
0.0698  
-  
Zalaegerszegi TE  
30  
Budapest Honvéd  
0.0697  
-  
Kaposvári Rakóczi  
29  
Ferencváros  
0.0689  
-  
Lombard Pápa  
27  
Gyuri ETO  
0.0649  
-  
Kecskeméti TE  
24  
Újpest  
0.0636  
-  
Újpest  
23  
Zalaegerszegi TE  
0.0636  
-  
Gyuri ETO  
23  
MTK Budapest  
0.0595  
-
```

```
Budapest Honvéd  
22  
Kaposvári Rakóczi  
0.0570  
-  
MTK Budapest  
22  
Lombard Pápa  
0.0560  
-  
Vasas  
21  
Szombathelyi Haladás  
0.0559  
-  
Szombathelyi Haladás  
20  
Vasas  
0.0543  
-  
BFC Siófok  
18  
Kecskeméti TE  
0.0439  
-  
Szolnoki MÁV  
9  
Szolnoki MÁV FC  
0.0329  
-
```

17.2. STL map érték szerinti rendezése

Például: <https://github.com/nbatfai/future/blob/master/cs/F9F2/fenykard.cpp#L180>

Megoldás videó:

Megoldás forrása: <https://gitlab.com/heinrichlaszlo/prog2forras/blob/master/Arroway/PolarGenerator.java>

Mi is az az STL (Standart Template Library)?

Az STL egy olyan C++ sablon- vagy mintaosztályok összessége, amelyek lehetővé teszik számos, széles körben használt algoritmus és adatszerkezet használatát.

Három fő dolgot találhatunk meg benne : tárolókat, iterátorokat és algoritmusokat.

A feladatunk szempontjából a tárolók lesznek érdekesek, név szerint a map fajtájuk.

Mit érdemes tudni a map fajtájú tárolókról?

Az elemeket a kulcsuk alapján tudjuk azonosítani, ún. Asszociatív tárolók.

Nézzük a megoldást.:

```
std::vector<std::pair<std::string, int>> sort_map ( std::map <std::string, int> &rank )
{
    std::vector<std::pair<std::string, int>> ordered;

    for ( auto & i : rank ) {
        if ( i.second ) {
            std::pair<std::string, int> p {i.first, i.second};
            ordered.push_back ( p );
        }
    }

    std::sort (
        std::begin ( ordered ), std::end ( ordered ),
        [= ] ( auto && p1, auto && p2 ) {
            return p1.second > p2.second;
        }
    );

    return ordered;
}
```

Számpárok vannak a std::map-ben, pontosabban egy kulcs és érték párosból áll a map, ahol mindig az első a kulcs és a második az érték. A kulcs alapján van rendezve minden esetben a map.

Mivel érték szerint rendezünk, ezért átmásoljuk a párokat egy vektorba, amelyben érték szerint lesznek rendezve az std::sort függvény segítségével. Vektor esetén már használható ez a függvény. Az std::sort-hoz meg kell adnunk mettől meddig rendezzen (std::begin(ordered),std::End(ordered)).

A második tagot az int alapján rendezzük, majd rendezetten visszatudjuk másolni őket egy új map-be, oda ahol már a rendezett sorrendre lesz szükségünk.

17.3. GIMP Scheme hack

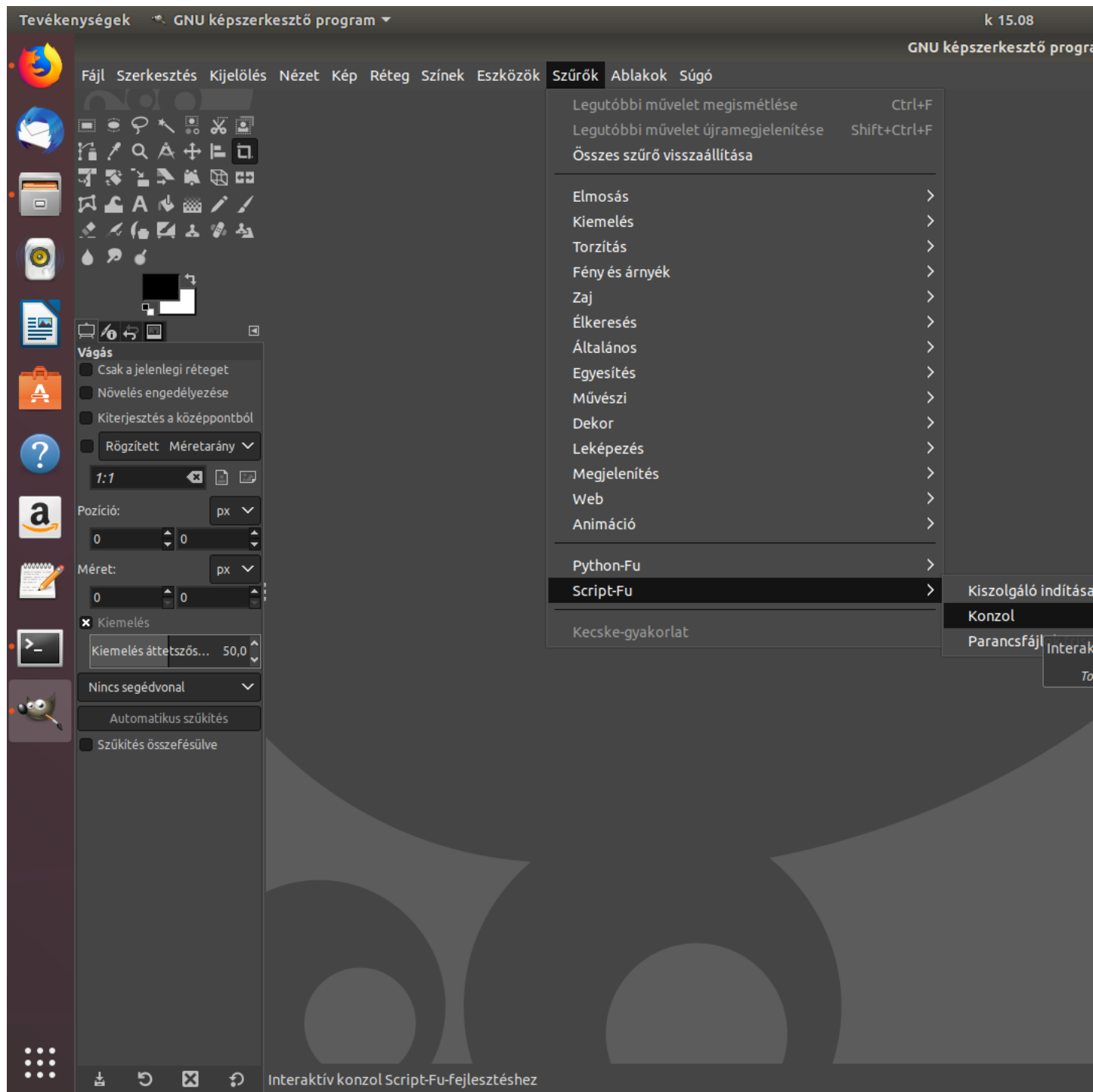
Ha az előző félévben nem dolgoztad fel a témát (például a mandalás vagy a króm szöveges dobozosat) akkor itt az alkalom!

Megoldás videó:

Megoldás forrása: https://gitlab.com/nbatfai/bhax/tree/master/attention_raising/GIMP_Lisp/Chrome <https://gitlab.com/heinrichlaszlo/bhax/tree/codes/codes/chaitin>

Tanulságok, tapasztalatok, magyarázat...

Az előző feladatban megismerkedtünk a Lisp nyelvvel, most a feladat során a Lisp nyelvcsalád egyik képviselőjével fogunk megismerkedni. A Scheme programnyelv az 1970-es évek közepén jelent meg és mai napig találkozhatunk Scheme kódokkal. Forrás és Scheme nyelvről bővebben: <https://hu.wikipedia.org/wiki/Scheme> A fájlunk .scm kiterjesztésű lesz, és ha berakjuk a GIMP erre hivatott mappájába, megtalálhatjuk mint használható szkriptet. A feladat során a Script-fut fogjuk használni. Az előző feladat során megismert GIMP képszerkesztő programhoz egy olyan szkriptet fogunk írni, ami a bemetként megadott szöveg króm effektezését teszi lehetővé. A megíráshoz használhatjuk a Script-Fu-konzolt is:



```
(define (color-curve)
  (let* (
    (tomb (cons-array 8 'byte))
  )
    (aset tomb 0 0)
    (aset tomb 1 0)
    (aset tomb 2 50)
    (aset tomb 3 190)
    (aset tomb 4 110)
```



```

        (aset tomb 5 20)
        (aset tomb 6 200)
        (aset tomb 7 190)
    tomb)
)

```

Megadjuk a függvényeket , olyan módon amit már az előző feladatban láthattunk , ezért erre most nem térnék ki külön. Létrehozunk egy 8 elemű tömböt a `color-curve` segítségével. Majd megadjuk a megfelelő értékeket.

Megadunk egy olyan függvényt is, ami egy lista x -edik elemét adja vissza, a `car`(lista első eleme) és `cdr`(a lista első elemén kívüli összes elem) használatával.

```

(define (elem x lista)

    (if (= x 1) (car lista) (elem (- x 1) (cdr lista) ) )

)

(define (text-wh text font fontsize)
(let*
    (
        (text-width 1)
        (text-height 1)
    )

    (set! text-width (car (gimp-text-get-extents-fontname text fontsize ↵
        PIXELS font)))
    (set! text-height (elem 2 (gimp-text-get-extents-fontname text ↵
        fontsize PIXELS font)))

    (list text-width text-height)
    )
)

```

Meghívunk egy függvényt. A függvényünk a listánk x -edik elemét adja vissza a `car` és a `cdr` használatával. `car` : a lista első eleme. `cdr` : a lista elsőn kívüli összes eleme. A soron következő függvény segítségével tudjuk kiszámolni a szöveg magasságát.

```

(define (script-fu-bhax-chrome text font fontsize width height color ↵
    gradient)
(let*
    (
        (image (car (gimp-image-new width height 0)))
        (layer (car (gimp-layer-new image width height RGB-IMAGE "bg" 100 ↵
            LAYER-MODE-NORMAL-LEGACY)))
        (textfs)
        (text-width (car (text-wh text font fontsize)))
        (text-height (elem 2 (text-wh text font fontsize)))
        (layer2)
    )
)

```

A `script-fu-bhax-chrome` metódus lesz felelős a króm effektért. A `let*` segítségével létrehozuk az objektumunkat a már megadott paraméterekből. Az `image`-be létrehozuk a képünket a `gimp-image-new` használatával. Ezután létrehozunk egy új réteget a `layer`-ben. Létrehozuk a `textfs`-t. A `text-wh` segítségével megadjuk a szöveg szélességét és magasságát. A legvégén pedig létrehozuk `layer2`-t.

9 lépésben megadjuk hogy krómosítson a szkriptünk.:

Első lépés:

```
;step 1
(gimp-image-insert-layer image layer 0 0)
(gimp-context-set-foreground '(0 0 0))
(gimp-drawable-fill layer FILL-FOREGROUND )
(gimp-context-set-foreground '(255 255 255))

(set! textfs (car (gimp-text-layer-new image text font fontsize PIXELS) ←
))
(gimp-image-insert-layer image textfs 0 0)
(gimp-layer-set-offsets textfs (- (/ width 2) (/ text-width 2)) (- (/ ←
height 2) (/ text-height 2)))

(set! layer (car(gimp-image-merge-down image textfs ←
CLIP-TO-BOTTOM-LAYER)))
```

A `gimp-image-insert-layer`) segítségével hozzáadjuk a képünkhöz a rétegünket. A `gimp-context-s` segítségével beállítjuk az elsődleges színt feketére majd a beállított színnel kiszinezük a rétegünket. Végük a színt fehérre állítjuk. A `textfs`-be létrehozunk egy új szövegréteget `gimp-text-layer-new` segítségével. Majd beállítjuk `gimp-layer-set-offsets` segítségével a réteg offsetjét. Összeolvasztjuk a létrehozott szöveget és a háttérrel. Ezzel el is értünk az első lépés végére.

Második lépés:

```
;step 2
(plug-in-gauss-iir RUN-INTERACTIVE image layer 15 TRUE TRUE)
```

A `plug-in-gauss-iir` segítségével elmosódást állítunk be a `layer`-en.

Harmadik lépés:

```
;step 3
(gimp-drawable-levels layer HISTOGRAM-VALUE .11 .42 TRUE 1 0 1 TRUE)
```

A `gimp-drawable-levels` segítségével beállítjuk a rétegünk szintezését.

Negyedik lépés:

```
;step 4
(plug-in-gauss-iir RUN-INTERACTIVE image layer 2 TRUE TRUE)
```

A `plug-in-gauss-iir` segítségével még egy elmosódást teszünk a `layer`-re.

Ötödik lépés:

```
;step 5
(gimp-image-select-color image CHANNEL-OP-REPLACE layer '(0 0 0))
(gimp-selection-invert image)
```

A `gimp-image-select-color` segítségével kijelöljük a layer rétegen a fehér pixeleket. Ezt a kijelölést megfordítjuk `gimp-selection-invert` segítségével.

Hatodik lépés:

```
;step 6
(set! layer2 (car (gimp-layer-new image width height RGB-IMAGE "2" 100 ←
  LAYER-MODE-NORMAL-LEGACY)))
(gimp-image-insert-layer image layer2 0 0)
```

Létrehozunk egy új réteget és ezt hozzáadjuk a képünkhöz.

Hetedik lépés:

```
;step 7
(gimp-context-set-gradient gradient)
(gimp-edit-blend layer2 BLEND-CUSTOM LAYER-MODE-NORMAL-LEGACY ←
  GRADIENT-LINEAR 100 0 REPEAT-NONE
  FALSE TRUE 5 .1 TRUE width (/ height 3) width (- height (/ height ←
  3)))
```

Átállítjuk aktívrá a gradient-t, majd a `gimp-edit-blend` segítségével összemossuk a layer2-t a háttérrel

Nyolcadik lépés:

```
;step 8
(plug-in-bump-map RUN-NONINTERACTIVE image layer2 layer 120 25 7 5 5 0 ←
  0 TRUE FALSE 2)
```

A `plug-in-bump-map` segítségével domborítás effektet adunk a képhez.

Kilencedik lépés:

```
;step 9
(gimp-curves-spline layer2 HISTOGRAM-VALUE 8 (color-curve))

(gimp-display-new image)
(gimp-image-clean-all image)
)
)
```

Beállítjuk a color-curvet-t, majd megjelenítjük egy új ablakban.

```
(script-fu-register "script-fu-bhax-chrome"
  "Chrome3"
  "Creates a chrome effect on a given text."
  "Norbert Bátfai"
  "Copyright 2019, Norbert Bátfai"
```

```
"January 19, 2019"
""
SF-STRING      "Text"      "Bátf41 Haxor"
SF-FONT        "Font"      "Sans"
SF-ADJUSTMENT  "Font size" '(100 1 1000 1 10 0 1)
SF-VALUE       "Width"     "1000"
SF-VALUE       "Height"    "1000"
SF-COLOR       "Color"     '(255 0 0)
SF-GRADIENT    "Gradient"  "Crown molding"
)
(script-fu-menu-register "script-fu-bhax-chrome"
  "<Image>/File/Create/BHAX"
)
```

Menü beállítása

17.4. Gengszterek

Gengszterek rendezése lambdával a Robotautó Világbajnokságban <https://youtu.be/DL6iQwPx1Yw> (8:05-től)

Megoldás videó:

Megoldás forrása: <https://gitlab.com/heinrichlaszlo/prog2forras/blob/master/Arroway/PolarGenerator.java>
Tutor

A Robocar World Championship egy olyan platform amely lehetőséget ad a robotautó kutatásra. A projekt központjában a Robocar City Emulator áll. A mi feladatunk az, hogy a carlexer.ll-ben található scanf függvényt értelmezzük.

Mik is azok a lambda kifejezések ?

A lambda kifejezések a C++ 11-es verziójában jelentek meg. Ezek a kifejezések lehetővé teszik az in-line functionok írását, vagyis egy vagy kevés soros függvényekét. Ezek a függvények úgynevezett "egyszer használatos" függvények. Nem adunk nekik nevet.

Példa egy lambda kifejezésre :

```
[] (int szam1, int szam2) -> { return szam1 + szam2; }
```

A szögletes zárójel jelzi a lambda kifejezés kezdetét. Ezután találhatóak a paraméterek , majd a nyilacs-kát követően a kapcsos zárójelek között található a függvény az elvégezni kívánt művelettel. Itt derül ki továbbá, hogy mi lesz a visszatérési érték típusa.

Most, hogy tisztáztuk mi is az a lambda kifejezés, nézzük a mi feladatunkat :

```
std::sort (gangsters.begin(), gangsters.end(), [this, cop] (Gangster x, ←
  Gangster y)
{
  return dst (cop, x.to) < dst (cop, y.to);
});
```

```
void sort (RandomAccessIterator first, RandomAccessIterator last, Compare ←  
    comp) ;
```

A gangsters vektor kerül rendezésre, a sort függvény segítségével. A függvény harmadik paramétere lesz a lambda kifejezés.

A lambda kifejezésben paraméterei két Gangster lesz. A kifejezés egy boolean-nal tér vissza. Ami akkor igaz, ha x gengszter és rendőr távolsága kisebb mint y gengszter és rendőr távolsága.

Fontos még, hogy jelen esetben a vektor távolság alapján lesz rendezve.

18. fejezet

Helló, !

18.1. OOCWC Boost ASIO hálózatkézelése

Mutassunk rá a scanf szerepére és használatára! <https://github.com/nbatfai/robocar-emulator/blob/master/justine/rcemu/src/carlexer.ll>

Megoldás videó:

Megoldás forrása: <https://gitlab.com/heinrichlaszlo/prog2forras/blob/master/Arroway/PolarGenerator.java>

Első sorban nézzük mit takar az OOCWC rövidítés : Az OOCWC a rObOCar World Championship szóból ered, ami egy olyan platform ami lehetőséget ad a robotautó-kutatásra. Az egész projekt középpontjában a Robocar City Emulator áll.

A mi feladatunk az lesz, hogy megvizsgáljuk a carlexer-ll-ben található scanf függvényt.

Vessünk egy pillantást az említett függvényre :

```
while (std::sscanf (data+nn, "<OK %d %u %u %u>%n", &idd, &f, &t, &s, &n) == 4)
{
    nn += n;
    gangsters.push_back (Gangster {idd, f, t, s});
}
```

A sscanf függvény formázott stringből olvas be adatokat. (a sima scanf függvénnyel ellentétben.)

Két részből állnak : buffer-ből és format-ból. A buffer egy karakterstringre mutató pointer. Innen kerülnek majd kiolvasásra az adatok. Ami pedig meghatározza, hogy az adatok hogyan kerüljenek kiolvasásra az a format lesz. Format-specifikátorokból áll, amik %-lel kezdődnek.

A d az int-ekre illeszkedik. Az u az unsigned integerre (magyarul : előjel nélküli integerre) illeszkedik. Az n pedig számon tartja a már beolvasott karakterek számát.

Addig tudjuk beolvasni az adatokat, ameddig nincs meg mind a 4 argumentum. (while == 4)

Ha mind a 4 várt argumentumunkat be tudtuk olvasni, új Gangster kerül létrehozásra a megfelelő argumentumokkal és a gangster vektorban tároljuk az új gengsterünket.

Az nn változóban tároljuk az összes beolvasott karakterek számát. Ez a változó a sscanf függvényben is megtalálható, nem máshol, mint a buffer részben. A data értékét tehát azért növeéjük mindig mindig nn értékkel, hogy onnan tudjunk adatokat beolvasni ahonnan azelőtt még ezt nem tettük.

18.2. BrainB

Mutassuk be a Qt slot-signal mechanizmust ebben a projektben: <https://github.com/nbatfai/esport-talent-search>

Megoldás videó:

Megoldás forrása: <https://gitlab.com/heinrichlaszlo/prog2forras/blob/master/Arroway/PolarGenerator.java>

A BrainB projekt célja a jövő esportolójának felkutatása. A program az agy úgynevezett kognitív képességét méri.

A játékos feladata a Samu Entropy nevű karakteren tartani az egérmutatót ami az idő teltével természetesen egyre nehezebb hiszen újabb és újabb Entropy karakterek jelennek meg a képernyőn. A program érzékeli, ha Samu Entropyt elvesztettük, ekkor csökkenti a többi Entropy számát.

```
#include <QApplication>
#include <QTextStream>
#include <QtWidgets>
#include "BrainBWin.h"

int main ( int argc, char **argv )
{
    QApplication app ( argc, argv );

    QTextStream qout ( stdout );
    qout.setCodec ( "UTF-8" );

    qout << "\n" << BrainBWin::appName << QString::fromUtf8 ( " ↵
    Copyright (C) 2017, 2018 Norbert Bátfai" ) << endl;

    qout << "This program is free software: you can redistribute it and ↵
    /or modify it under" << endl;

    .....
    .....
    .....

    QRect rect = QApplication::desktop()->availableGeometry();
    BrainBWin brainBWin ( rect.width(), rect.height() );
    brainBWin.setWindowState ( brainBWin.windowState() ^ Qt:: ↵
        WindowFullScreen );
    brainBWin.show();
    return app.exec();
}
```

Includoljuk a BrainBWin osztályt, a többi inkludolásra kerülő osztályt már az előző programokban megfigyelhettük, ugyanis ez a program is Qt grafikus felületet használ. Deklarálunk egy app QApplication típusú objektumot. Itt használjuk a qout functiont amivel a standard outputra lehet írni. Deklaráljuk az entropyskat. Majd létrehozuk a BrainBWin objektumot. Az osztályban inkludolásra került függvények és változók itt is .h kiterjesztésű header fájlokban vannak.

BrainBWin.h

Deklaráljuk azt a függvényt ami az egér és a billentyűzetről történő inputok figyeléséért felelős. Deklaráljuk továbbá a konstruktort és a destruktort.

BrainBWin.cpp

Itt kerül definiálásra az előző header fájlban deklaráltak.

BrainBThread.h

A Hero-kat tartalmazó vectorra typedefet használunk, mostantól Hero-sként hivatkozunk rá.

BrainBThread.cpp

Létrehozuk a Hero-kat. Definiáljuk a destruktort és deklaráljuk a run, pause és set_paused függvényeket

Most , hogy ezt tisztáztuk, nézzük meg, hogy mi az az a Qt slot-signal mechanizmusa ?

Egy Qt objektum képes signal-t , vagy magyarul jelet sugározni magából. Ez a sugárzás akkor történik meg, ha valami "fontos" történik. Az objektumoknak lehetnek slot-jaik, ezek várják majd "kapják" el a signalokat. Amikor egy signal-t és egy slot-ot összekötünk, akkor a slotnak mondjuk meg, hogy melyik signal-t várja. Magyarul a slotok felelnek a kapcsolatért. Ez a mechanizmus csak Qt-ben érhető el, magyarul teljes egészében Qt specifikus.

A BrainBWin.cpp fájlunkban helyezkedik el egy programrészlet ami a BrainBWin.h-ban létrehozott slotok-hoz köti a signalokat, ezek a connectek.

```
connect ( brainBThread, SIGNAL ( heroesChanged ( QImage, int, int ) ),
         this, SLOT ( updateHeroes ( QImage, int, int ) ) );

connect ( brainBThread, SIGNAL ( endAndStats ( int ) ),
         this, SLOT ( endAndStats ( int ) ) );
```

Első paraméter a küldő brainBThread, második a signal, harmadik a this mindkettőnél, magyarul az adott osztály, negyedik a slot.

A heroesChanged signal-t köti az updateHeroes slot-hoz a megadott paraméterekkel, majd a endAndStats signal-t köti az endAndStats slot-hoz a megadott paraméterekkel.

Első connect : ha a brainBThread jelet küld a hős helyzet változtatásáról, akkor ezt a slot fogadja és frissíti a hős helyzetét.

Második connect : ha a kísérletnek vége BrainBWin is befejezi a kísérletét.

Tekintsük meg a függvényeket amelyek meghívásra kerülnek :

egyik :

```
void BrainBWin::endAndStats ( const int &t )
{
    qDebug() << "\n\n\n";
    qDebug() << "Thank you for using " + appName;
    qDebug() << "The result can be found in the directory " + statDir;
    qDebug() << "\n\n\n";
```



```
        save ( t );  
        close();  
    }
```

másik :

```
void BrainBWin::updateHeroes ( const QImage &image, const int &x, ←  
    const int &y )  
{  
  
    if ( start && !brainBThread->get_paused() ) {  
  
        int dist = ( this->mouse_x - x ) * ( this->mouse_x - x ) + ←  
            ( this->mouse_y - y ) * ( this->mouse_y - y );  
  
        if ( dist > 121 ) {  
            ++nofLost;  
            nofFound = 0;  
            if ( nofLost > 12 ) {  
  
                if ( state == found && firstLost ) {  
                    found2lost.push_back ( brainBThread ←  
                        ->get_bps() );  
                }  
  
                firstLost = true;  
  
                state = lost;  
                nofLost = 0;  
                //qDebug() << "LOST";  
                //double mean = brainBThread->meanLost();  
                //qDebug() << mean;  
  
                brainBThread->decComp();  
            }  
        } else {  
            ++nofFound;  
            nofLost = 0;  
            if ( nofFound > 12 ) {  
  
                if ( state == lost && firstLost ) {  
                    lost2found.push_back ( brainBThread ←  
                        ->get_bps() );  
                }  
  
                state = found;  
                nofFound = 0;  
                //qDebug() << "FOUND";  
                //double mean = brainBThread->meanFound();  
                //qDebug() << mean;  
            }  
        }  
    }  
}
```

```
        brainBThread->incComp();
    }

    }

    }
    pixmap = QPixmap::fromImage ( image );
    update();
}
```

fordítás és futtatás után :

18.3. SamuCam

Mutassunk rá a webcam (pl. Androidos mobilod) kezelésére ebben a projektben: <https://github.com/nbatfai/SamuCam>

Megoldás videó:

Megoldás forrása: <https://gitlab.com/heinrichlaszlo/prog2forras/blob/master/Arroway/PolarGenerator.java>

A feladatunk az, hogy mutassunk rá a webcamra.

A következőkben a SamuCam.cpp fájlunkat fogjuk elemezni kódcsípetekre bontva :

```
#include "SamuCam.h"

SamuCam::SamuCam ( std::string videoStream, int width = 176, int height = 144 )
: videoStream ( videoStream ), width ( width ), height ( height )
{
    openVideoStream();
}

SamuCam::~SamuCam ()
{
}
```

A legelső lépésben inkludoljuk a header fájlunkat. Létrehozzuk a konstruktorunkat , ami 3 paraméterrel rendelkezik. Létrehozásra kerül a destruktork is.

```
void SamuCam::openVideoStream()
{
    videoCapture.open ( videoStream );

    videoCapture.set ( CV_CAP_PROP_FRAME_WIDTH, width );
    videoCapture.set ( CV_CAP_PROP_FRAME_HEIGHT, height );
    videoCapture.set ( CV_CAP_PROP_FPS, 10 );
}
```

Az `openVideoStream` függvényen belül található a `VideoCapture`, annak az `open` függvénye és az `open` argumentuma. Az eszköz indexét találhatjuk meg ott. Ezek az index 0 az alapértelmezett kamera eszközünk neve. Majd megtörténik a szélesség, magasság és fps szám megadása.

```
void SamuCam::run()
{

    cv::CascadeClassifier faceClassifier;

    std::string faceXML = "lbpcascade_frontalface.xml"; // https:// ↩ github.com/Itseez/opencv/tree/master/data/lbpcascades

    if ( !faceClassifier.load ( faceXML ) )
    {
        qDebug() << "error: cannot found" << faceXML.c_str();
        return;
    }

    cv::Mat frame;
```

Következik a `run` függvényünk. Szükségünk lesz egy `CascadeClassifier`-re.

Mi is az a `CascadeClassifier`?

A `CascadeClassifier`-t alapesetben egy adott tárgy detektálására szokták használni OpenCV-s programokban. Esetünkben a feladata az arc felismerés lesz. Ehhez viszont le kell tölteni a megjegyzésben található link segítségével egy xml fájlt és egy stringbe kimenteni. Ez az xml tartalmazza a arc felismeréséhez szükséges algoritmust. A `load` függvény segítségével hívjuk meg az xml-t , ha hiányzik az xml fájlunk akkor pedig hibaüzenetet kapunk vissza.

```
while ( videoCapture.isOpened() )
{

    QThread::msleep ( 50 );
    while ( videoCapture.read ( frame ) )
    {

        if ( !frame.empty() )
        {

            cv::resize ( frame, frame, cv::Size ( 176, 144 ), 0, 0, ↩
                cv::INTER_CUBIC );

            std::vector<cv::Rect> faces;
            cv::Mat grayFrame;

            cv::cvtColor ( frame, grayFrame, cv::COLOR_BGR2GRAY );
            cv::equalizeHist ( grayFrame, grayFrame );

            faceClassifier.detectMultiScale ( grayFrame, faces, 1.1, ↩
                3, 0, cv::Size ( 60, 60 ) );
```

```
        if ( faces.size() > 0 )
        {

            cv::Mat onlyFace = frame ( faces[0] ).clone();

            QImage* face = new QImage ( onlyFace.data,
                                        onlyFace.cols,
                                        onlyFace.rows,
                                        onlyFace.step,
                                        QImage::Format_RGB888 );

            cv::Point x ( faces[0].x-1, faces[0].y-1 );
            cv::Point y ( faces[0].x + faces[0].width+2, faces[0].y + faces[0].height+2 );
            cv::rectangle ( frame, x, y, cv::Scalar ( 240, 230, 200 ) );

            emit faceChanged ( face );
        }

        QImage* webcam = new QImage ( frame.data,
                                        frame.cols,
                                        frame.rows,
                                        frame.step,
                                        QImage::Format_RGB888 );

        emit webcamChanged ( webcam );

    }

    QThread::msleep ( 80 );
}
```

A while függvényen belül a program 50 ms-onként ellenőrzi, megvan-e nyitva a kameránk. Ha meg van nyitva a kameránk, akkor az adott képkockák beolvasásra majd tárolásra kerülnek. A kép átméretezésre kerül a resize segítségével. Következik a faces vektor, majd a képünket szürkévé színezzük a cvtColor segítségével. Ennek is létrehozunk egy tárolót aminek a neve GrayFrame lesz. Majd elkezdjük keresni az arcot a detectMultiScale segítségével. A megtalált arcok egy rectangle kerülnek tárolásra. QImage készül az arcról, majd egy webcamra nevű QImage is készül. Mindez 80ms-onként ismétlődik.

```
if ( ! videoCapture.isOpened() )
{
    openVideoStream();
}
```

Ha nincs megnyitva a kameránk, akkor megnyitja azt.

Fordítjuk majd futtatjuk a prgramunkat.

18.4. FUTURE tevékenység editor

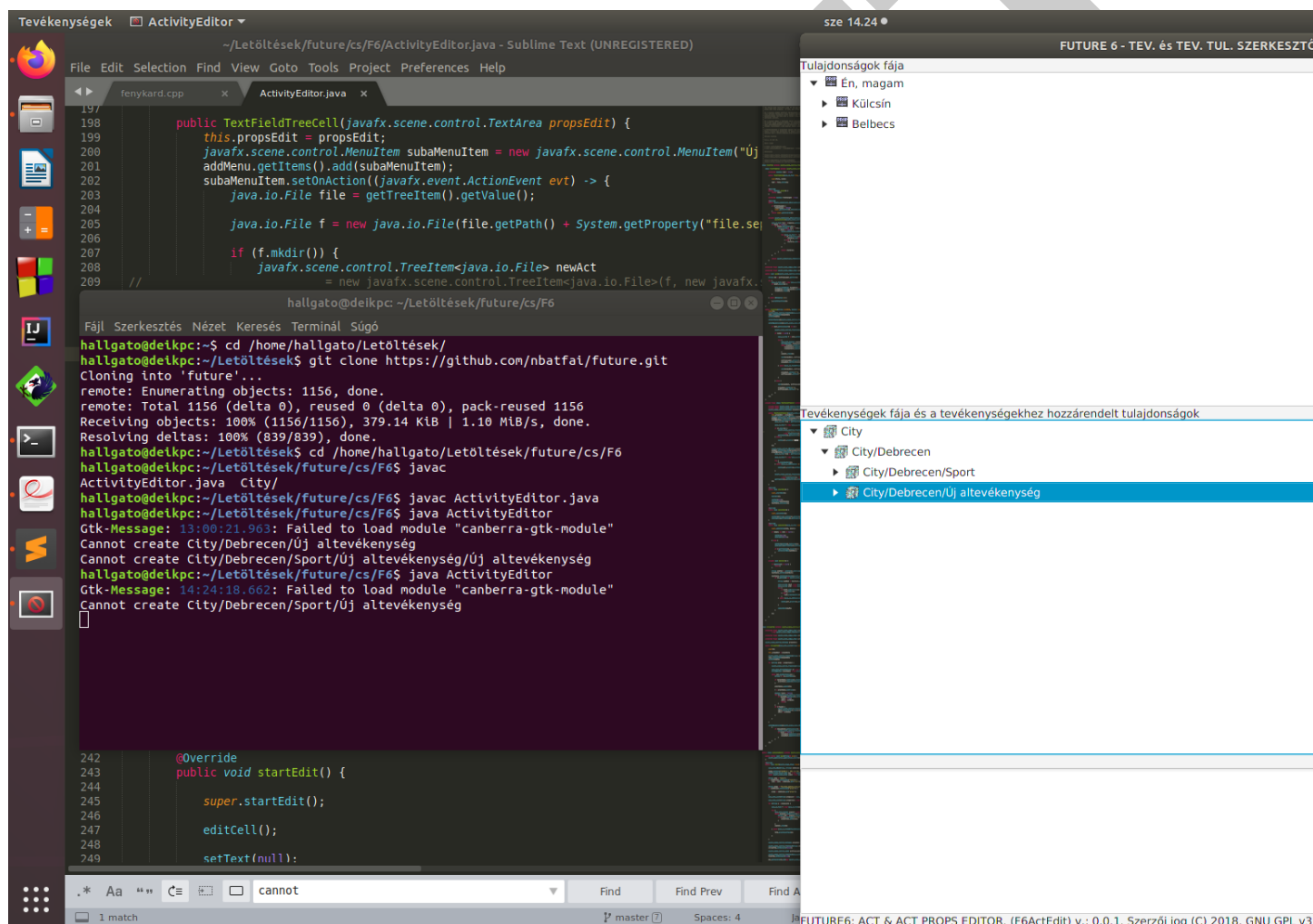
Javítsunk valamit a ActivityEditor.java JavaFX programon! <https://github.com/nbatfai/future/tree/master/cs/F6>
Itt láthatjuk működésben az alapot: <https://www.twitch.tv/videos/222879467>

Megoldás videó:

Megoldás forrása: <https://gitlab.com/heinrichlaszlo/prog2forras/blob/master/Arroway/PolarGenerator.java>

A feladatunk, hogy bugokat javítsunk a ActivityEditor.java fájlban.

Fordítás és futtatás után amikor megpróbálnánk új altevékenységet létrehozni, rögtön egy buggal találjuk szembe magunkat, ugyanis nem tudunk létrehozni új altvékenységet.



A bugra egy lehetséges megoldása egy bevezetett számláló segítségével :

```

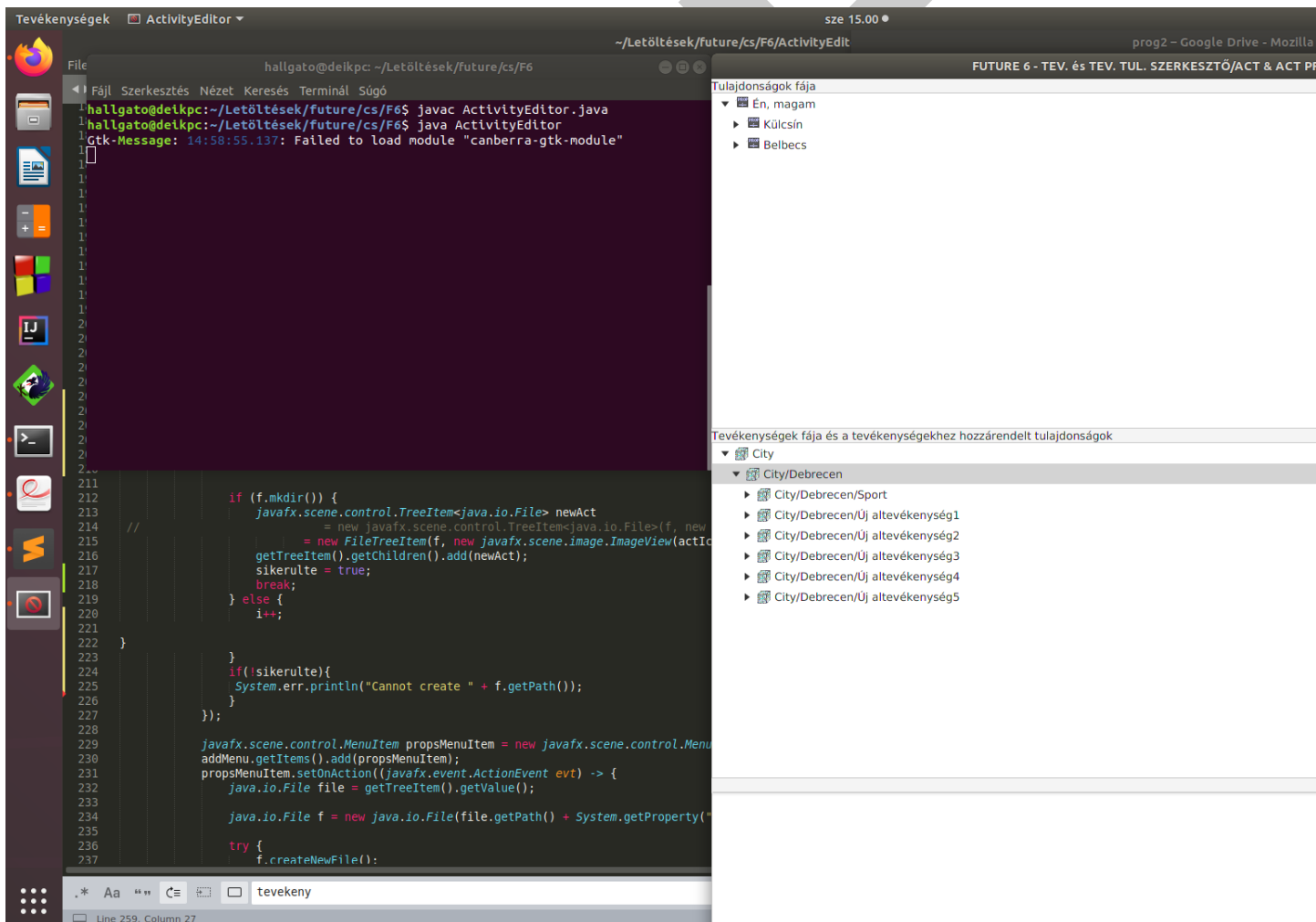
        boolean sikerulte = false;
        java.io.File f;
        int i = 1;
        while(true){
            f = new java.io.File(file.getPath() + System.getProperty("file.separator") + "Új altevékenység" + i);

            if (f.mkdir()) {
                javafx.scene.control.TreeItem<java.io.File> newAct
                = new javafx.scene.control.TreeItem<java.io.File>(f, new javafx.scene.image.ImageView(actIcon));
                = new FileTreeItem(f, new javafx.scene.image.ImageView(actIcon));
                getTreeItem().getChildren().add(newAct);
                sikerulte = true;
                break;
            } else {
                i++;
            }
        }

        if(!sikerulte){
            System.err.println("Cannot create " + f.getPath());
        }
    });
}

```

Mostmár ha egy új altevékenységet szeretnénk létrehozni, azt nyugodt szívvel megtehetjük :



IV. rész

Irodalomjegyzék

DRAFT

18.5. Általános

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

18.6. C

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

18.7. C++

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

18.8. 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.