Univerzális programozás

Írd meg a saját programozás tankönyvedet!

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

COLLABORATORS			
	TITLE : Univerzális programozás		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY	Bátfai, Norbert Ã©s Szabó, Nikolett	2019. május 9.	

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	2019-02-20	Turing	szabon
0.2.0	2019-03-04	Chomsky	szabon
0.3.0	2019-03-11	Olvasónapló	szabon
0.4.0	2019-03-17	Ceasar	szabon

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME
0.5.0	2019-03-25	Mandel	szabon
0.6.0	2019-04-01	Welch	szabon
0.7.0	2019-04-08	Conway	szabon
0.7.0	2019-05-09	Befejezés	szabon

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

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 (hogy lássuk mást is) példával.

Hogyan nyomjuk?

Rántsd 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 találod az informatikai szövegek jelölésére használható gazdag „API” elemenkénti bemutatását.

I. rész

Bevezetés

1. fejezet

Vízió

1.1. Mi a programozás?

1.2. Milyen doksikat olvassak el?

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

1.3. Milyen filmeket nézzek meg?

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

II. rész

Tematikus feladatok

Bátf41 Haxor Stream
A feladatokkal kapcsolatos élő adásokat sugároz a https://www.twitch.tv/nbatfai csatorna, melynek permanens archívuma a https://www.youtube.com/c/nbatfai csatornán található.

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:

100% dolgoztat 1 magot:

```
fordítás: gcc vegtelen.c .o vegtelen
          ./vegtelen
```

```
Program vegtelen
{

  #include <stdio.h>

        int main(){
        int t=1;
            while(t==1);
                  }
}
```

0% dolgoztat 1 magot:

```
fordítás: gcc sleep.c -o sleep
          ./sleep
```

```
Program sleep
{

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

```
        int main()
        {
            int t=1;
            while(t==1)
    sleep(1000);
  }
}
```

100% dolgoztat minden magot:

```
fordítás: gcc 100x8.c -o 100x8 -fopenmp
          ./100x8
```

```
Program 100x8
{

  #include <stdio.h>

        int main()
        {
            int t=1;
            #pragma omp parallel
            while(t==1);
}
```

Tanulságok, tapasztalatok, magyarázat...

100%-on dolgoztat 1 magot: A ciklus addig ismétlődik amíg a 't' értéke egy, mivel a t értékét semmi nem változtatja, így ez egy végtelen ciklus. Egy végtelen ciklus egy magot 100%-on dolgoztat.

0%-on dolgoztat 1 magot:A ciklus addig hajtódig végre amíg a 't' értéke egy, ezt azonban nem változtatja semmi. A sleep parancs hajtódik végre egymás után -míg a programot le nem állítjuk- minek hatására a CPU egy magja vár, így 0%-on dolgozik.

100%-on dolgoztat minden magot: Mint az első esetnél, csak itt a '#pragma omp parallel' hatására az összes mag 100%-on dolgozik.

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 vlgtelen ciklus:

```
Program T100
{

  boolean Lefagy(Program P)
```

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

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

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

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

akár önmagára

```
T100(T100)
false
```

ezt a kimenetet adja.

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

```
Program T1000
{

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

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

  main(Input Q)
  {
    Lefagy2(Q)
  }
```

```
}
```

Mit for kiírni erre a `T1000(T1000)` futtatásra?

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

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

Tanulságok, tapasztalatok, magyarázat...

Mivel az elméletben lehetetlen ilyen programot írni, így a gyakorlaban is.

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:

```
Program osszeadas
{

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

int main()
{
int a = 10;
int b = 20;

a=a+b;
b=a-b;
a=a-b;

printf("%d %d\n",a ,b);

}
```

Tanulságok, tapasztalatok, magyarázat...

A program az összeadás és kivonás műveleteket használja. A két változót összeadja, ha ebből az összegből kivonja az egyiket akkor megkapja a másikat.

2.4. Labdapattogás

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

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

Megoldás forrása: https://bhaxor.blog.hu/2018/08/28/labdapattogas

if-fel:

fordíitása: gcc labda.c -o labda -lncurses

```
Program labda
{

#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;

    for ( ;; ) {

        getmaxyx ( ablak, my , mx );

        mvprintw ( y, x, "O" );

        refresh ();
        usleep ( 100000 );

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

        if ( x>=mx-1 ) { // elerte-e a jobb oldalt?
            xnov = xnov * -1;
        }
        if ( x<=0 ) { // elerte-e a bal oldalt?
```

```
            xnov = xnov * -1;
        }
        if ( y<=0 ) { // elerte-e a tetejet?
            ynov = ynov * -1;
        }
        if ( y>=my-1 ) { // elerte-e a aljat?
            ynov = ynov * -1;
        }

  clear ();

    }
}
}
}
```

If nélkül:

Kódot írta és tutorált: Nagy László Mihály

fordítása: gcc ifnelkul.c -o ifnelkul -lncurses -lm

```
Program ifnelkul
{
#include <stdio.h>
#include <curses.h>
#include <unistd.h>
#include <math.h>

int main (void)
{
    WINDOW *ablak;
    ablak = initscr();

    int xcurrent = 0, ycurrent = 0;
    int xoffset = 1, yoffset = 1;
    int xmax, ymax;

    for (;;)
    {
  clear();
        getmaxyx(ablak, ymax, xmax);
        mvprintw(ycurrent, xcurrent, "O");

        refresh();
        usleep(33330);

        xcurrent += xoffset;
        ycurrent += yoffset;

        xoffset *= pow(-1, (xcurrent / (xmax - 1) + ((xmax - 1) - xcurrent) ↩
            / (xmax - 1)));
```

```
        yoffset *= pow(-1, (ycurrent / (ymax - 1) + ((ymax - 1) - ycurrent) ↩
            / (ymax - 1)));
    }

    return 0;
}



}
```

Tanulságok, tapasztalatok, magyarázat...

Az if-es és az if nélküli program is ugya azt a célt szolgálja. A képernyőn mozgatnak (látszólag) egy karaktert faltól falig. A programokban a végtelen ciklus a koordinátákat számolja és figyeli mikor éri el a karaktert az ablak szélét. Ekkor irányt kell változtatnia. A ciklus mindig kirajzol egy képet majd törli azt az új rajzolása előtt, ettől tűnik mozgónak a kép.

If nélküli: A mainen belül először az ablak kerül meghatározásra. Az x- és ycurreant az aktuális koordinátákat határozza meg, az x- és yoffset az elmozdulás mértékét az x- és ymax pedig a koordináták maximumát adja meg. A for végtelen ciklusban a clear mindig törli az előzőleg kirajzolt "o" betűt. A getmaxyx meghatározza az adott ablakban a legnagyobb lehetéges koordinátákat. A mvprintw meghatározza hová kerüljön az "o", a refresh pedig kirajzolja. Az unsleep a két kirajzolá közti időt határozza meg. Ezután meghatározásra kerül a következő kirajzolandó "o" koordinátái. Végezetül az elmozdulást változtatjuk az alapján, hogy elérte-e a labda egy szélet vagy sem az alábbi módon: Az x koordinátára vonatkozóan: Az "o"-tól balra és jobbra lévő részt is elosztjuk a az egész résszel, majd ezeket összeadjuk. A c-ben való osztás a tört rész elhagyja, tehát ha nem a szélén van akkor az összeg 0 lesz, ha pedig a szélén akkor 1. Ennek megfelelően emeljük a -1-et 0-ra vagy 1-re, és ezzel szorozzuk az xoffsetet.

If-es: Az előzőtől annyiban tér el, hoy máshogy vizsgálja elérte-e az ablak szélét. Itt egyesével figyeli az if-ekkel az ablak jobb és bal oldalát valamint a felső és alsó határt. Ha a labda eléri az egyiket akkor az annak megfelelő tengelyen lévő elmozdulást szorozza -1-el.

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:

```
Program szoh
{

  #include "std_lib_facilities.h"

            int main()
            {
                int a=1;
```

```
                int x=0;

                while (a!=0)
                {
                    a<<=1;
                    x++;

                }
            cout << x;
            }
}
}
```

Tanulságok, tapasztalatok, magyarázat...

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:

```
Program pagerank
{
#include <stdio.h>
#include <math.h>

void
kiir (double tomb[], int db)
{
  int i;
  for (i=0; i<db; i++)
  printf("PageRank [%d]: %lf\n", i, tomb[i]);
}

double tavolsag(double pagerank[],double pagerank_temp[],int db)
{
  double tav = 0.0;
  int i;
  for(i=0;i<db;i++)
  tav +=abs(pagerank[i] - pagerank_temp[i]);
  return tav;
}

int main(void)
{
  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};

  long int i,j,h;
  i=0; j=0; h=5;

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

  if ( tavolsag(PR,PRv, 4) < 0.00001)
  break;
  }
kiir (PR,4);
return 0;

}
}
```

Tanulságok, tapasztalatok, magyarázat...

A PageRank a Google alapja. A weboldalak jóságát számolja a rájuk- és belőlük kimutató linkekből. Alapelve, hogy egy oldal linke egy általa jónak tarot oldalra mutat. Minden kimutató link egy szavzatnak fogható fel, a szavazat értéke pedig függ az oldal jóságától.

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

Tanulságok, tapasztalatok, magyarázat...

Brun tétel szerint az ikerprímek reciprokösszege konvergens sorozatot alkot.

```
            library(matlab)
```

```
stp <- function(x){

    primes = primes(x)
    diff = primes[2:length(primes)]-primes[1:length(primes)-1]
    idx = which(diff==2)
    t1primes = primes[idx]
    t2primes = primes[idx]+2
    rt1plust2 = 1/t1primes+1/t2primes
    return(sum(rt1plust2))
}

x=seq(13, 1000000, by=10000)
y=sapply(x, FUN = stp)
plot(x,y,type="b")
```

stp függvény működése: stp(x): x-ig vagy ahoz legközelebbi nála kisebb prímig kiszámolja a prímpárok tagonkénti reciprokainak az összegének az összegét.

Első sor: Kiírja a prímeket x-ig egy egy dimenziós vektorba.

Második sor: Megadja az egymást követő prímek különbségét. Egy vektorban eltárolja a 2. prímtől a prímeket x-ig. Egy másikban pedig az első prímtől x-1-ig (vagyis az x előtti prímig). Majd a két vektort kivonja egymásból. Ezzel létrehozva egy harmadik vektort.

Harmadik sor: Megadja, hogy a különbség vektor hanyadik elemei 2. prímtől a prímeket x-ig. Egy másikban pedig az első prímtől x-1-ig (vagyis az x előtti prímig)

Negyedik sor: Megadja az iker prímek első tagjait, az indexük segítségével.

Ötödik sor: Megadja az ikerprímek második tagját úgy, hogy az első tagokhoz hozzáad kettőt.

Hatodik sor: Megadja az ikerprímpárok reciprokainak összegét.

Hetedik sor:Összeadja az előző sor adatait.

Kód utolsó 3 sora: Kirajzolja a grafikont.

2.8. A Monty Hall probléma

Írj R szimulációt a Monty Hall problémára!

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

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

Tanulságok, tapasztalatok, magyarázat...

A Monty Hall probléma:

Van 3 ajtó, egy mögött a nyeremény a másik kettő mögött semmi. A játékos választ egy ajtót majd a műsorvezető a másik két ajtó közül kinyit egyet ami mögött nincs semmi, majd a játékos dönthet, hogy megváltoztatja-e a döntését. A kérdés: Mikor van több esély nyerni, ha változtat vagy ha nem.

```
           kiserletek_szama=10000000
kiserlet = sample(1:3, kiserletek_szama, replace=T)
jatekos = sample(1:3, kiserletek_szama, replace=T)
```

Megadja a kísérletek számát, majd véletlen szerűen generál ennyi választást a nyeremény elhelyezésére és a játékos választására.

```
musorvezeto=vector(length = 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)]

}
```

A műsorvezető döntése: Ez a választás függ a nyeremény helyétől és a játékos választásától, ha a játékos eltalálta akkor a megmaradt két ajtó közül választ véletlenszerűen, ha nem találta el akkor a megmaradt egyből választ véletlenszrűen.

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

for (i in 1:kiserletek_szama) {

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

}

valtoztatesnyer = which(kiserlet==valtoztat)


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

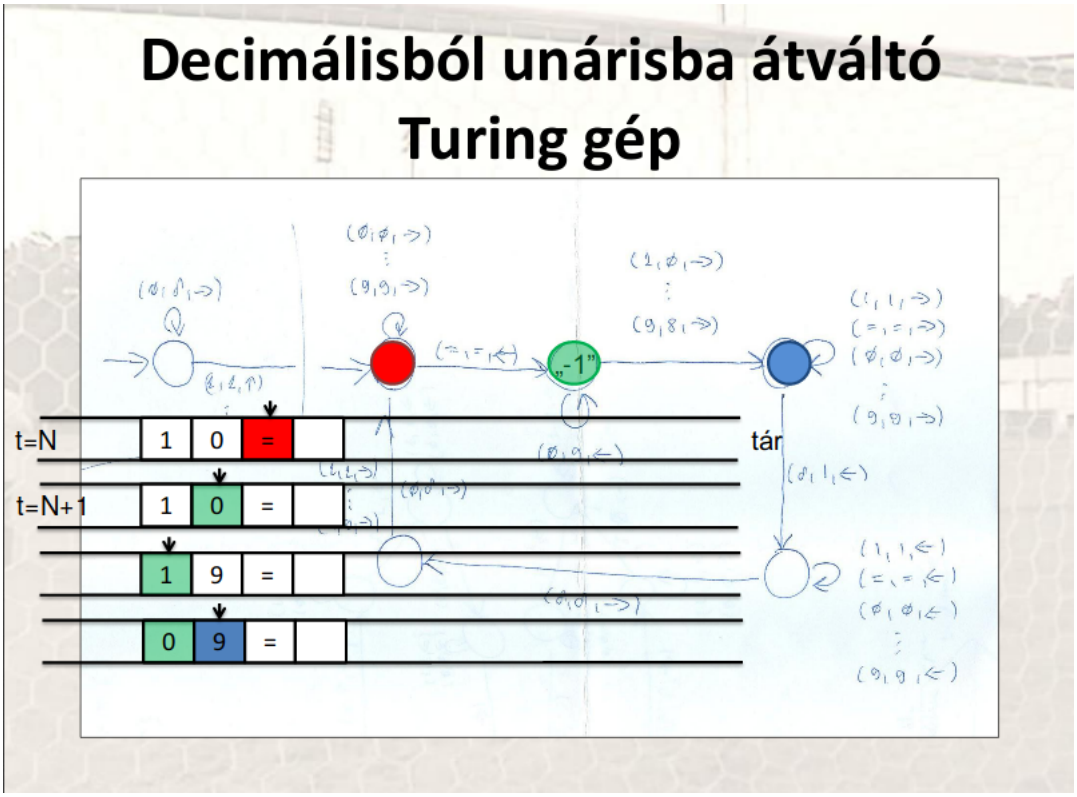
Végkövetkeztetés: többször nyerünk ha változtatunk.

3. fejezet

Helló, Chomsky!

3.1. Decimálisból unárisba átváltó Turing gép

Kép forrása: Prog1 fólia



Állapotátmenet gráfjával megadva írd meg ezt a gépet!

Megoldás videó:

Megoldás forrása:

```
Program valt
{

  #include <stdio.h>
```

```
#include "std_lib_facilities.h"

int main(){
int a=0;

cin >> a;
while(a>0)
{cout << 1;
  a=a-1;
}

}
```

Tanulságok, tapasztalatok, magyarázat...

Az unáris az egyes számrendszer. A program úgy alakítja át a kapott számot, hogy annyi egyest ír ki ahányat ki tud vonni a számból addíg amíg az 0 nem lesz.

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:

Tanulságok, tapasztalatok, magyarázat...

3.3. Hivatkozási nyelv

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

Megoldás videó:

Megoldás forrása:

```
Program c
{

#include <stdio.h>

int main(){

for(int i=1; i<10;i++);

}
```

Tanulságok, tapasztalatok, magyarázat...

gcc -o -stb=c89 c.c parancsra a program nem fordul le a for után lévő int miatt, viszont gcc -o -stb=99 c.c -vel már lefordul-

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/nbatfai/bhax/blob/master/thematic_tutorials/bhax_textbook_IgyNeveldaProg Chomsky/realnumber.l

```
Program realnumber
%{
#include <stdio.h>
int realnumbers = 0;
%}
digit [0-9]
%%
{digit}*(\.{digit}+)?  {++realnumbers;
  printf("[realnum=%s %f]", yytext, atof(yytext));}
%%
int
main()
{
 yylex ();
 printf("The number of real numbers is %d\n", realnumbers);
 return 0;
}

}
```

Tanulságok, tapasztalatok, magyarázat...

A forrás 3 részből áll melyeket %% választ el egymástól.

Első rész: az integer kap egy 0 kezdőértéket, és meghatározzuk hogy a digit szám.

Második rész:Megadjuk hogy nézhet ki gy valós szám, ha taálunk egyet akkor hozzáadjuk az integerhez.

Harmadik rész: Eredményt kiírjuk a standard outputra.

3.5. l33t.l

Lexelj össze egy l33t ciphert!

Megoldás videó:

Megoldás forrása: https://gitlab.com/nbatfai/bhax/blob/master/thematic_tutorials/bhax_textbook_IgyNeveldaProg
Chomsky/l337d1c7.l

```
%{
  #include <stdio.h>
  #include <stdlib.h>
  #include <time.h>
  #include <ctype.h>

  #define L337SIZE (sizeof l337d1c7 / sizeof (struct cipher))

  struct cipher {
    char c;
    char *leet[4];
  } l337d1c7 [] = {

  {'a', {"4", "4", "@", "/-\\"}},
  {'b', {"b", "8", "|3", "|}"}},
  {'c', {"c", "(", "<", "{"}},
  {'d', {"d", "|)", "|]", "|}"}},
  {'e', {"3", "3", "3", "3"}},
  {'f', {"f", "|=", "ph", "|#"}},
  {'g', {"g", "6", "[", "[+"}},
  {'h', {"h", "4", "|-|", "[-]"}},
  {'i', {"1", "1", "|", "!"}},
  {'j', {"j", "7", "_|", "_/"}},
  {'k', {"k", "|<", "1<", "|{"}},
  {'l', {"l", "1", "|", "|_"}},
  {'m', {"m", "44", "(V)", "|\\/|"}},
  {'n', {"n", "|\\|", "/\\/", "/V"}},
  {'o', {"0", "0", "()", "[]"}},
  {'p', {"p", "/o", "|D", "|o"}},
  {'q', {"q", "9", "O_", "(,)"}},
  {'r', {"r", "12", "12", "|2"}},
  {'s', {"s", "5", "$", "$"}},
  {'t', {"t", "7", "7", "'|'"}},
  {'u', {"u", "|_|", "(_)", "[_]"}},
  {'v', {"v", "\\/", "\\/", "\\/"}},
  {'w', {"w", "VV", "\\/\\/", "(/\\)"}},
  {'x', {"x", "%", ")(", ")("}},
  {'y', {"y", "", "", ""}},
  {'z', {"z", "2", "7_", ">_"}},

  {'0', {"D", "0", "D", "0"}},
  {'1', {"I", "I", "L", "L"}},
  {'2', {"Z", "Z", "Z", "e"}},
  {'3', {"E", "E", "E", "E"}},
  {'4', {"h", "h", "A", "A"}},
  {'5', {"S", "S", "S", "S"}},
  {'6', {"b", "b", "G", "G"}},
```

```
  {'7', {"T", "T", "j", "j"}},
  {'8', {"X", "X", "X", "X"}},
  {'9', {"g", "g", "j", "j"}}

// https://simple.wikipedia.org/wiki/Leet
  };

%}
%%
. {

    int found = 0;
    for(int i=0; i<L337SIZE; ++i)
    {

      if(l337d1c7[i].c == tolower(*yytext))
      {

        int r = 1+(int) (100.0*rand()/(RAND_MAX+1.0));

          if(r<91)
          printf("%s", l337d1c7[i].leet[0]);
          else if(r<95)
          printf("%s", l337d1c7[i].leet[1]);
        else if(r<98)
          printf("%s", l337d1c7[i].leet[2]);
        else
          printf("%s", l337d1c7[i].leet[3]);

        found = 1;
        break;
      }

    }

    if(!found)
       printf("%c", *yytext);

  }
%%
int
main()
{
  srand(time(NULL)+getpid());
  yylex();
  return 0;
}
```

Tanulságok, tapasztalatok, magyarázat...

A program előre meghatározott karaktereket cserélki előre meghatározott karakterekre.

3.6. A források olvasása

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

```
if(signal(SIGINT, jelkezelo)==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 jelkezelo 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, jelkezelo);
```

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

for ciklusban az i értéke kezdetben 0 majd mindig növeli egyel és addig fut amíg kisebb mint 5. (ötször fog lefutni)

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

Mint az előző.

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

Ez bugos. tomb[i]=i++ nál hibás.

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

A for ciklus addig fut míg *d++ = *s++ -al és i kisebb mint n

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

Bugos.

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

Kiírja az f fügvény értékét ha "a" a bemenete, majd az eredeti "a"-t.

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

Buggos.

Megoldás forrása:

Megoldás videó:

Tanulságok, tapasztalatok, magyarázat...

3.7. Logikus

Hogyan olvasod természetes nyelven az alábbi Ar nyelvű formulákat?

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

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

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

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

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

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

Tanulságok, tapasztalatok, magyarázat...

1.:Minden x-nél van nagyobb y ami prím.

2.:Minden x-nél van nagyobb y ikerprím

3.:Minden x prím számnál létezik nagyobb y.

4.:Létezik minden nem prím x-nél nagyobb y.

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;
  ```
- ```
  int *b = &a;
  ```
- ```
  int &r = a;
  ```
- ```
  int c[5];
  ```
- ```
  int (&tr)[5] = c;
  ```
- ```
  int *d[5];
  ```
- ```
  int *h ();
  ```
- ```
  int *(*l) ();
  ```
- ```
  int (*v (int c)) (int a, int b)
  ```
- ```
  int (*(*z) (int)) (int, int);
  ```

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

```
Program dek
{
#include <stdio.h>
int main(){

int a;
int *b = &a;
int &r = a;
int c[5];
int (&tr)[5] = c;
int *d[5];
int *h ();
int *(*l) ();
int (*v (int c)) (int a, int b);
int (*(*z) (int)) (int, int);

}
}
```

4. fejezet

Helló, Caesar!

4.1. double** háromszögmátrix

Megoldás videó:

Megoldás forrása: https://gitlab.com/nbatfai/bhax/blob/master/thematic_tutorials/bhax_textbook_IgyNeveldaProg
Caesar/tm.c

Tanulságok, tapasztalatok, magyarázat...

Fordítás/futattás:

gcc -Wall tm.c -o tm

./tm

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

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

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

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

```
    }

    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");
    }

    tm[3][0] = 42.0;
    (*(tm + 3))[1] = 43.0;  // mi van, ha itt hiányzik a külső ()
    *(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");
    }

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

    free (tm);

    return 0;
}
```

A mátrix egy n sorból és n oszlopból álló táblázat. Speciáli fajtája a háromszogmátrix, amely négyzetes, vagyis ugyan annyi sora és oszlopa van. Megkülönbözetünk alsó és felső háromszögmátrixokat. Alsó háromszögmátrix estén a főátlóban és alatta vannak az értékes elemek míg felette csupa nulla, míga felsőben fordítva, az értékes elemek a főátlóban és fölöte találhatóak. A nullákat nem kell eltárolnunk, a többi elemet pedig alsó háromszögmátrix esetén sor-, felső háromszögmátrix esetén oszlopfolytonosan tudjuk tárolni.

4.2. C EXOR titkosító

Írj egy EXOR titkosítót C-ben!

Megoldás videó:

Megoldás forrása: https://progpater.blog.hu/2011/02/15/felvetelt_hirdet_a_cia?fbclid=IwAR1q4--FgCA_tRhLST

Tanulságok, tapasztalatok, magyarázat...

```
Program e
```

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

#define MAX_KULCS 100
#define BUFFER_MERET 256

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

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

  int kulcs_index = 0;
  int olvasott_bajtok = 0;

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

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

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

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

  }

      write (1, buffer, olvasott_bajtok);

    }
}


}
```

A program az EXOR művelettel titkosít és dekódól is. Ugyanis olvasható szöveg EXOR kulcs = titkosított szöveg, és titkosított szöveg EXOR kulcs = olvasható szöveg.

4.3. Java EXOR titkosító

Írj egy EXOR titkosítót Java-ban!

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

4.4. C EXOR törő

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

Megoldás videó:

Megoldás forrása: https://progpater.blog.hu/2011/02/15/felvetelt_hirdet_a_cia?fbclid=IwAR1q4--FgCA_tRhLST

Tanulságok, tapasztalatok, magyarázat...

```
Program t
{
#define MAX_TITKOS 4096
#define OLVASAS_BUFFER 256
#define KULCS_MERET 8
#define _GNU_SOURCE

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


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

    return (double) titkos_meret / sz;
}

int
tiszta_lehet (const char *titkos, int titkos_meret)
{
    // a tiszta szoveg valszeg tartalmazza a gyakori magyar szavakat
    // illetve az átlagos szóhossz vizsgálatával csökkentjük a
    // potenciális töréseket

    double szohossz = atlagos_szohossz (titkos, titkos_meret);

    return szohossz > 6.0 && szohossz < 9.0
           && strcasestr (titkos, "hogy") && strcasestr (titkos, "nem")
           && strcasestr (titkos, "az") && strcasestr (titkos, "ha");

}
```

```
void
exor (const char kulcs[], int kulcs_meret, char titkos[], int titkos_meret)
{

    int kulcs_index = 0;

    for (int i = 0; i < titkos_meret; ++i)
    {

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

    }

}

int
exor_tores (const char kulcs[], int kulcs_meret, char titkos[],
            int titkos_meret)
{

    exor (kulcs, kulcs_meret, titkos, titkos_meret);

    return tiszta_lehet (titkos, titkos_meret);

}

int
main (void)
{
    char titkos[MAX_TITKOS];
    char *p = titkos;
    int olvasott_bajtok;

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

    // maradek hely nullazasa a titkos bufferben
    for (int i = 0; i < MAX_TITKOS - (p - titkos); ++i)
        titkos[p - titkos + i] = '\0';

    // osszes kulcs eloallitasa
    for (int ii = '0'; ii <= '9'; ++ii)
        for (int ji = '0'; ji <= '9'; ++ji)
```

```
            for (int ki = '0'; ki <= '9'; ++ki)
                for (int li = '0'; li <= '9'; ++li)
                    for (int mi = '0'; mi <= '9'; ++mi)
                        for (int ni = '0'; ni <= '9'; ++ni)
                            for (int oi = '0'; oi <= '9'; ++oi)
                                for (int pi = '0'; pi <= '9'; ++pi)
                                {

                                    char *kulcs;

                                    if ((kulcs = (char *)malloc(sizeof(char ↩
                                       )*KULCS_MERET)) == NULL)
                                    {
                                        printf("Memoria (kulcs) faliora\n") ↩
                                            ;
                                        exit(-1);
                                    }

                                    kulcs[0] = ii;
                                    kulcs[1] = ji;
                                    kulcs[2] = ki;
                                    kulcs[3] = li;
                                    kulcs[4] = mi;
                                    kulcs[5] = ni;
                                    kulcs[6] = oi;
                                    kulcs[7] = pi;

                                    exor_tores (kulcs, KULCS_MERET, titkos, ↩
                                        p - titkos);

                                    free(kulcs);
                                }

    return 0;
}

}
```

A program feltételezi,hogy a kulcs 8 számjegy. Előállítja az összes kulcsot, és mindet kipróbálja, az így kapott szövegeken pedig a magyar nyelvre jellemuő tulajdonságokat keres.

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

Tanulságok, tapasztalatok, magyarázat...

4.6. Hiba-visszaterjesztéses perceptron

C++

Megoldás videó:

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

Tanulságok, tapasztalatok, magyarázat...

A feladathoz 3 három fájl szükséges.

mandelpng.cpp , ez előállítja a Mandelbrot halmaz képét, fordítása: g++ mandelpng.cpp -o mandel -lpng . A -lpng kapcsolóhoz szükséges a következő telepítés: sudo apt-get install libpng++-dev. Futtatása: ./mandel mandel.png

```
// Mandelbrot png
// Programozó Páternoszter
//
// Bátfai Norbert, nbatfai@inf.unideb.hu, nbatfai@gmail.com
// http://progpater.blog.hu/2011/03/26/kepes_egypercesek
//
// Fordítás:
// g++ mandelpng.c++ `libpng-config --ldflags` -o mandelpng

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

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

    // 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";
    // Végigzongorázzuk a szélesség x magasság rácsot:
    for (int j=0; j<magassag; ++j) {
        //sor = j;
```

```
        for (int k=0; k<szelesseg; ++k) {
            // c = (reC, imC) a rács csomópontjainak
            // megfelelő komplex szám
            reC = a+k*dx;
            imC = d-j*dy;
            // z_0 = 0 = (reZ, imZ)
            reZ = 0;
            imZ = 0;
            iteracio = 0;
            // z_{n+1} = z_n * z_n + c iterációk
            // számítása, amíg |z_n| < 2 vagy még
            // nem értük el a 255 iterációt, ha
            // viszont elértük, akkor úgy vesszük,
            // hogy a kiinduláci 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;

            }

            kep.set_pixel(k, j, png::rgb_pixel(255-iteracio%256,
                                               255-iteracio%256, 255- ↩
                                                  iteracio%256));
        }
        std::cout << "." << std::flush;
    }

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

}
```

perc.cpp

Bemenete az előzőleg létrehozott kép lesz. az ml.hpp pedig egy header fájlja.

Fordítása: g++ ml.hpp perc.cpp -o perc -lpng -std=c++11 Futtatása: ./perc mandel.png

5. fejezet

Helló, Mandelbrot!

5.1. A Mandelbrot halmaz

Megoldás videó:

Megoldás forrása:

A Hiba-visszaterjesztéses perceptron feladatben már feltűnt ez a program, melynek feladata a mandelbrot halmaz kirajzolása:

A Madelbrot halmaz Benoît Mandelbrotról kapta nevét, aki felfedezte a fraktálokat. A fraktálok végtelenül komplex alakzatok, melyek szemmel láthatóan ismétlődést tartalmaznak. A Mandelrot-halmaz is egy ilyen fraktál, képe a végtelenségig nagyítható, és mindig felfedezhető benne önmaga hasonmása. A halmaz kirajzolása: A komplex számsíkon veszünk egy C pontot, és a következő sorozatot képezzük:

$Z_0 := C$ és $Z_{i+1} := Z_i * Z_i + C$

Azok a C-k elemei a halmaznak ahol a sorozat nullához tart, ezeket feketére színezzük. Azok a C-k amik nem elemei a halmaznak ott a sorozat a végtelenbe tart.

```
// Mandelbrot png
// Programozó Páternoszter
//
// Bátfai Norbert, nbatfai@inf.unideb.hu, nbatfai@gmail.com
// http://progpater.blog.hu/2011/03/26/kepes_egypercesek
//
// Fordítás:
// g++ mandelpng.c++ `libpng-config --ldflags` -o mandelpng

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

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

```
// 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";
// Végigzongorázzuk a szélesség x magasság rácsot:
for (int j=0; j<magassag; ++j) {
    //sor = j;
    for (int k=0; k<szelesseg; ++k) {
        // c = (reC, imC) a rács csomópontjainak
        // megfelelő komplex szám
        reC = a+k*dx;
        imC = d-j*dy;
        // z_0 = 0 = (reZ, imZ)
        reZ = 0;
        imZ = 0;
        iteracio = 0;
        // z_{n+1} = z_n * z_n + c iterációk
        // számítása, amíg |z_n| < 2 vagy még
        // nem értük el a 255 iterációt, ha
        // viszont elértük, akkor úgy vesszük,
        // hogy a kiinduláci 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;

        }

        kep.set_pixel(k, j, png::rgb_pixel(255-iteracio%256,
                                           255-iteracio%256, 255- ↩
                                              iteracio%256));
    }
    std::cout << "." << std::flush;
```

```
    }

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

}
```

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

Megoldás videó:

Megoldás forrása:

A program ugyan azt a képet álítja elő, mint az előző, de ez már komplex számokkal számol.

```
// Verzio: 3.1.2.cpp
// Forditas:
// g++ 3.1.2.cpp -lpng -O3 -o 3.1.2
// Futtatas:
// ./3.1.2 mandel.png 1920 1080 2040  ↩
   -0.019473810573093663922605855987058021128 18  ↩
   -0.019473810572541341845642648422654019668 7  ↩
   0.798505756933826860155534177465597167611 1  ↩
   0.798505756934379196110285192844457924366
// ./3.1.2 mandel.png 1920 1080 1020  ↩
   0.412765541820958925534057470940751954913 1  ↩
   0.412765541824581805308014281763462349772 5  ↩
   0.213538705176874649138696327099751215428 1  ↩
   0.213538705180497528912653137922461610287 4
// Nyomtatas:
// a2ps 3.1.2.cpp -o 3.1.2.cpp.pdf -1 --line-numbers=1  --left-footer=" ↩
   BATF41 HAXOR STR34M" --right-footer="https://bhaxor.blog.hu/" --pro= ↩
   color
// ps2pdf 3.1.2.cpp.pdf 3.1.2.cpp.pdf.pdf
//
//
// Copyright (C) 2019
// Norbert Bátfai, batfai.norbert@inf.unideb.hu
//
//  This program is free software: you can redistribute it and/or modify
//  it under the terms of the GNU General Public License as published by
//  the Free Software Foundation, either version 3 of the License, or
//  (at your option) any later version.
//
//  This program is distributed in the hope that it will be useful,
//  but WITHOUT ANY WARRANTY; without even the implied warranty of
//  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
//  GNU General Public License for more details.
```

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


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

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

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

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

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

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

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

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

```
      // k megy az oszlopokon

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

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

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

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

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

              ++iteracio;
            }

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

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

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

}
```

5.3. Biomorfok

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

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

Tanulságok, tapasztalatok, magyarázat...

A program futtatásához meg kell adnunk 12 argomentumot, amit ha nem teszünk meg, a program a következő üzeneetl kilép: Hasznalat: ./3.1.2 fajlnev szelesseg magassag n a b c d reC imC R.

```
// Verzio: 3.1.3.cpp
```

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

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

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

    int szelesseg = 1920;
    int magassag = 1080;
    int iteraciosHatar = 255;
    double xmin = -1.9;
    double xmax = 0.7;
    double ymin = -1.3;
    double ymax = 1.3;
    double reC = .285, imC = 0;
```

```
    double R = 10.0;

    if ( argc == 12 )
    {
        szelesseg = atoi ( argv[2] );
        magassag =  atoi ( argv[3] );
        iteraciosHatar =  atoi ( argv[4] );
        xmin = atof ( argv[5] );
        xmax = atof ( argv[6] );
        ymin = atof ( argv[7] );
        ymax = atof ( argv[8] );
        reC = atof ( argv[9] );
        imC = atof ( argv[10] );
        R = atof ( argv[11] );

    }
    else
    {
        std::cout << "Hasznalat: ./3.1.2 fajlnev szelesseg magassag n a b c ↩
            d reC imC R" << std::endl;
        return -1;
    }

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

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

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

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

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

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

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

            int iteracio = 0;
            for (int i=0; i < iteraciosHatar; ++i)
            {

                z_n = std::pow(z_n, 3) + cc;
                //z_n = std::pow(z_n, 2) + std::sin(z_n) + cc;
```

```
                if(std::real ( z_n ) > R || std::imag ( z_n ) > R)
                {
                    iteracio = i;
                    break;
                }
            }

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

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

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

}
```

5.4. A Mandelbrot halmaz CUDA megvalósítása

Megoldás videó:

Megoldás forrása:

5.5. Mandelbrot nagyító és utazó C++ nyelven

Tutorálta: Nyitrai Dávid

Kódot írta: Petrus József Tamás

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

Megoldás videó:

Megoldás forrása: https://github.com/Matchbox1233/Mandelbrot/blob/master/5.cpp

A frakszál számolja ki a mandelbrot halmazt, amit rajzolásra átad a frakablaknak. A main függvényben w,show() kirajzolja a halmazt. A nagyítást a frakablak.cpp végzi

frakablak.cpp

```
        #include "frakablak.h"
```

```
FrakAblak::FrakAblak(double a, double b, double c, double d,
                     int szelesseg, int iteraciosHatar, QWidget *parent)
                         : QMainWindow(parent)
{
    setWindowTitle("Mandelbrot halmaz");

    int magassag = (int)(szelesseg * ((d-c)/(b-a)));

    setFixedSize(QSize(szelesseg, magassag));
    fraktal= new QImage(szelesseg, magassag, QImage::Format_RGB32);

    mandelbrot = new FrakSzal(a, b, c, d, szelesseg, magassag,  ↩
       iteraciosHatar, this);
    mandelbrot->start();

}

FrakAblak::~FrakAblak()
{
    delete fraktal;
    delete mandelbrot;
}

void FrakAblak::paintEvent(QPaintEvent*) {
    QPainter qpainter(this);
    qpainter.drawImage(0, 0, *fraktal);
    qpainter.end();
}

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

frakablk.h

```
        #ifndef FRAKABLAK_H
#define FRAKABLAK_H

#include <QMainWindow>
#include <QImage>
#include <QPainter>
#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) ;

protected:
    void paintEvent(QPaintEvent*);

private:
    QImage* fraktal;
    FrakSzal* mandelbrot;

};

#endif // FRAKABLAK_H
```

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 sûrû 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
            // megfelelõ 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 kiinduláci 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 < iterációsHatár sérülésével lépett ki, azaz
            // feltesszük a c-rõl, hogy itt a z_{n+1} = z_n * z_n + c
            // sorozat konvergens, azaz iteráció = iterációsHatár
            // ekkor az iteráció %= 256 egyenlõ 255, mert az esetleges
            // nagyítasok 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);
    }
}
```

frakszal.h

```
        #ifndef FRAKSZAL_H
#define FRAKSZAL_H

#include <QThread>
#include <QImage>
#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?
    // (tk. most a nagyítási pontosság)
    int iteraciosHatar;

    FrakAblak* frakAblak;
    int* egySor;

};

#endif // FRAKSZAL_H
```

main.cpp

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

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    FrakAblak w1,
    w2(-.08292191725019529, -.082921917244591272,
```

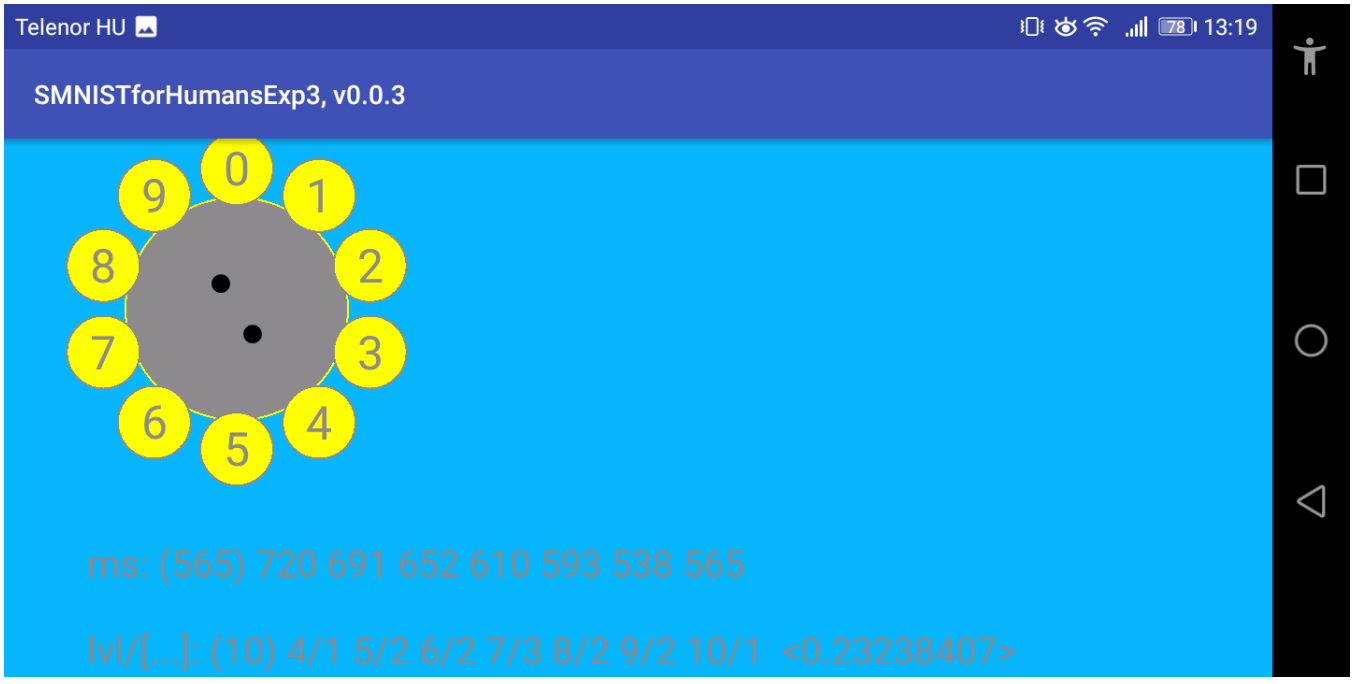
```
        -.9662079988595939, -.9662079988551173, 1200, 3000),
    w3(-.08292191724880625, -.0829219172470933,
        -.9662079988581493, -.9662079988563615, 1200, 4000),
    w4(.14388310361318304, .14388310362702217,
        .6523089200729396, .6523089200854384, 1200, 38655);
    w1.show();
    w2.show();
    w3.show();
    w4.show();

    return a.exec();
}
```

5.6. Mandelbrot nagyító és utazó Java nyelven

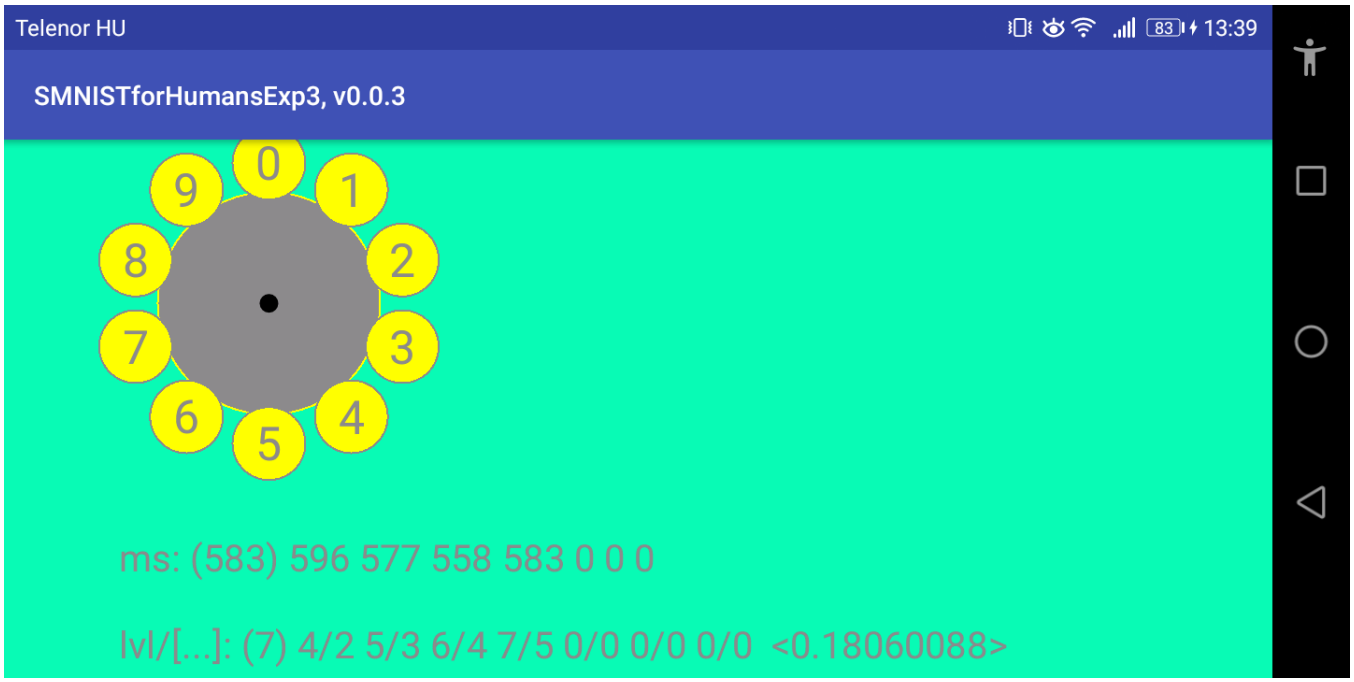
5.7. SMNIST

A 2 feladatot kiváltható screenshot.

Első nekifutás: cél: lvl 10.



Második nekifutás: cél: /5.


Telenor HU
13:39
SMNISTforHumansExp3, v0.0.3
0
1
2
3
4
5
6
7
8
9
ms: (583) 596 577 558 583 0 0 0
lvl/[...]: (7) 4/2 5/3 6/4 7/5 0/0 0/0 0/0 <0.18060088>

6. fejezet

Helló, Welch!

Fejezetet tutorálta: Nyitrai Dávid

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 videó:

Megoldás forrása: java: https://github.com/Matchbox1233/Welch/blob/master/1a.java c++: https://github.com/-Matchbox1233/Welch/blob/master/1b.cpp

Tutorálta: Nyitrai Dávid

Kódot írta: Petrus József Tamás

Java:

```
import java.util.Random;

    public class PolarGen
    {
        private double tarolt;
        private boolean nincsTarolt;
        private Random r;
        private int RAND_MAX;

        public PolarGen()
        {
            nincsTarolt = true;
            r = new Random();
            r.setSeed(20);
            this.RAND_MAX=100;
        }
```

```
        public PolarGen(Integer RAND_MAX)
        {
            nincsTarolt = true;
            r = new Random();
            r.setSeed(20);
            this.RAND_MAX=RAND_MAX;
        }

        public double kovetkezo()
        {
            if (nincsTarolt)
            {

                double u1, u2, v1, v2, w;
                int i=0;
                do
                {
                      u1 = r.nextInt() / (RAND_MAX + 1.0);
                      u2 = r.nextInt() / (RAND_MAX + 1.0);
                      v1 = 2 * u1 - 1;
                      v2 = 2 * u2 - 1;
                      w = v1 * v1 + v2 * v2;
                }
                while (w > 1 && i++ < 40000000);
                double r = Math.sqrt ((2 * Math.log10(w)) / w);
                tarolt = r * v2;
                nincsTarolt = !nincsTarolt;
                return r * v1;
            }
            else
            {
                nincsTarolt = !nincsTarolt;
                return tarolt;
            }
        }
}
```

C++:

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

class PolarGen
{
public:
  PolarGen ()
  {
    nincsTarolt = true;
    std::srand (std::time (NULL));
```

```
  }
   ~PolarGen ()
  {
  }
  double kovetkezo ()
  {
    if (nincsTarolt)
      {
        double u1, u2, v1, v2, w;
        do
    {
      u1 = std::rand () / (RAND_MAX + 1.0);
      u2 = std::rand () / (RAND_MAX + 1.0);
      v1 = 2 * u1 - 1;
      v2 = 2 * u2 - 1;
      w = v1 * v1 + v2 * v2;
    }
        while (w > 1);

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

        tarolt = r * v2;
        nincsTarolt = !nincsTarolt;

        return r * v1;
      }
    else
      {
        nincsTarolt = !nincsTarolt;
        return tarolt;
       }
   }



private:
  bool nincsTarolt;
  double tarolt;

};


int
main (int argc, char **argv)
{
  PolarGen pg;

  for (int i = 0; i < 10; ++i)
    std::cout << pg.kovetkezo () << std::endl;
```

```
  return 0;
}
```

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

A program két részre osztható: class PolarGen és main. A polargen tovább bontható private és plublic részre. A private részben megadott változókat a public részben használjuk, ahol a számolás is történik. A main rész feladata az eredmények kiiratása.

6.2. LZW

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

Megoldás videó:

Megoldás forrása: https://github.com/Matchbox1233/Welch/blob/master/2.c

Tutorálta: Nyitrai Dávid

Kódot írta: Petrus József Tamás

Node* create_empty() : itt állítjuk be a fa gyökerének a kezdőértékét. A tmp a fa mutató mely értéke nulla. A kitüntetett gyökérelem a /.

Node* create_node(char val): Mint az előben, azzal a kivétellel, hogy itt a jobb és bal oldali ág kezdőértéket adjuk meg.

void insert_tree(char val): Itt döntjük el melyik elem megy a jobb vagy a bal oldalra.

void inorder(Node* elem,int depth): A fa bejárása inorder, feldolgozási sorrend: bol, gyökér, jobb. return : kilépési feltétel. depth változó: mélység, minnél "magasabb" szinten van egy elem (tehét minnél mélyebben), a terminálban annál jobbrább kerül kiírásra. spaces pointer: terminálban kirajzolja a "-" -eket, az osztály végén free paranccsal felszabadításra kerül.

void destroy_tree(Node* elem): postorder módon (bal,jobb, gyökér) bejárja a fát és felszabadítja a pointereket.

int main(int argc, char** argv): Véletlenszerűen generált számok 2-vel való maradékos osztásával, meghatározza a fa értékeit és kirajzolja a fát.

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <string.h>
typedef struct node{
    char c;
    struct node* left;
    struct node* right;
} Node;

Node* fa;
Node gyoker;
```

```
#define null NULL

Node* create_empty()
{
    Node* tmp = &gyoker;
    tmp->c= '/';
    tmp->left = null;
    tmp->right = null;
    return tmp;
}

Node* create_node(char val)
{
    Node* tmp = (Node*)malloc(sizeof(Node));
    tmp->c=val;
    tmp->left = null;
    tmp->right = null;
    return tmp;
}

void insert_tree(char val)
{
    if(val=='0')
    {
        if(fa->left == null)
        {
            fa->left = create_node(val);
            fa = &gyoker;
            //printf("Inserted into left.");
        }
        else
        {
            fa = fa->left;
        }
    }
    else
    {
        if(fa->right == null)
        {
            fa->right = create_node(val);
            fa = &gyoker;
            //printf("Inserted into left.");
        }
        else
        {
            fa = fa->right;
        }
    }
}
```

```
void inorder(Node* elem,int depth)
{

    if(elem==null)
    {
        return;
    }
    inorder(elem->left,depth+1);
    if(depth)
    {
        char *spaces;
        spaces =(char*) malloc(sizeof(char)*depth*2+1);
        for(int i=0;i<depth;i+=2)
        {
            spaces[i]='-';
            spaces[i+1]='-';
        }
        spaces[depth]='\0';

        printf("%s%c\n",spaces,elem->c);
        free (spaces);
    }
    else
    {
        printf("%c\n",elem->c);
    }
    inorder(elem->right,depth+1);
}

void destroy_tree(Node* elem)
{
    if(elem==null)
    {
        return;
    }
    destroy_tree(elem->left);
    destroy_tree(elem->right);
    if(elem->c == gyoker.c)
    {

    }
    else
    {
        free(elem);
    }
}

int main(int argc, char** argv)
{
    srand(time(null));
```

```
    fa = create_empty();
    for(int i=0;i<100;i++)
    {
        int x=rand()%2;
        if(x)
        {
            insert_tree('1');
        }
        else
        {
            insert_tree('0');
        }
    }

    inorder(&gyoker,0);

    destroy_tree(&gyoker);
    return 0;
}
```

6.3. Fabejárás

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

Megoldás videó:

Megoldás forrása: https://github.com/Matchbox1233/Welch/blob/master/3.c

Tutorálta: Nyitrai Dávid

Kódot írta: Petrus József Tamás

Preorder bejárás: Először kerül feldoglozásra a gyökér majd a bal aztán a jobb oldali ág.

Posztorder bejárás: Először kerül feldolgozásra a bal oldal, majd a jobb, végül pedig a a gyökér.

Az előző programon kell elvégezni egy kis módosítást:

void preorder(Node* elem,int depth) osztály a preorder bejárást valósítja meg a void postorder(Node* elem,int depth) pedig a postorder bejárást. A három féle bejárás közül a futtatás során egy kapcsolóval lehet dönteni.

```
   #include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <string.h>
typedef struct node{
  char c;
  struct node* left;
  struct node* right;
} Node;
```

```
Node* fa;
Node gyoker;

#define null NULL

Node* create_empty()
{
  Node* tmp = &gyoker;
  tmp->c= '/';
  tmp->left = null;
  tmp->right = null;
  return tmp;
}

Node* create_node(char val)
{
  Node* tmp = (Node*)malloc(sizeof(Node));
  tmp->c=val;
  tmp->left = null;
  tmp->right = null;
  return tmp;
}

void insert_tree(char val)
{
  if(val=='0')
  {
    if(fa->left == null)
    {
      fa->left = create_node(val);
      fa = &gyoker;
      //printf("Inserted into left.");
    }
    else
    {
      fa = fa->left;
    }
  }
  else
  {
    if(fa->right == null)
    {
      fa->right = create_node(val);
      fa = &gyoker;
      //printf("Inserted into left.");
    }
    else
    {
      fa = fa->right;
    }
```

```
  }
}

void inorder(Node* elem,int depth)
{

  if(elem==null)
  {
    return;
  }
  inorder(elem->left,depth+1);
  if(depth)
  {
    char *spaces;
    spaces =(char*) malloc(sizeof(char)*depth*2+1);
    for(int i=0;i<depth;i+=2)
    {
      spaces[i]='-';
      spaces[i+1]='-';
    }
    spaces[depth]='\0';

    printf("%s%c\n",spaces,elem->c);
  }
  else
  {
    printf("%c\n",elem->c);
  }
  inorder(elem->right,depth+1);
}

void preorder(Node* elem,int depth)
{
  if(elem==null)
  {
    return;
  }
  if(depth)
  {
    char *spaces;
    spaces =(char*) malloc(sizeof(char)*depth*2+1);
    for(int i=0;i<depth;i+=2)
    {
      spaces[i]='-';
      spaces[i+1]='-';
    }
    spaces[depth*2]='\0';

    printf("%s%c\n",spaces,elem->c);
  }
```

```
  else
  {
    printf("%c\n",elem->c);
  }
  preorder(elem->left,depth+1);
  preorder(elem->right,depth+1);
}
void postorder(Node* elem,int depth)
{
  if(elem==null)
  {
    return;
  }
  postorder(elem->left,depth+1);
  postorder(elem->right,depth+1);
  if(depth)
  {
    char *spaces;
    spaces =(char*) malloc(sizeof(char)*depth*2+1);
    for(int i=0;i<depth;i+=2)
    {
      spaces[i]='-';
      spaces[i+1]='-';
    }
    spaces[depth*2]='\0';

    printf("%s%c\n",spaces,elem->c);
    free(spaces);
  }
  else
  {
    printf("%c\n",elem->c);
  }
}

void destroy_tree(Node* elem)
{
  if(elem==null)
  {
    return;
  }
  destroy_tree(elem->left);
  destroy_tree(elem->right);
  if(elem->c == gyoker.c)
  {

  }
  else
  {
    free(elem);
```

```
  }
}

void usage()
{
  printf("Használat: ./binfa KAPCSOLÓ\n");
  printf("Az KAPCSOLÓ lehet:\n");
  printf("--preorder\tA bináris fa preorder bejárása\n");
  printf("--inorder\tA bináris fa inorder bejárása\n");
  printf("--postorder\tA bináris fa postorder bejárása\n");
}

int main(int argc, char** argv)
{
  srand(time(null));
  fa = create_empty();
  //gyoker = *fa;
  for(int i=0;i<10000;i++)
  {
    int x=rand()%2;
    if(x)
    {
      insert_tree('1');
    }
    else
    {
      insert_tree('0');
    }
  }
  if(argc == 2)
  {
    if(strcmp(argv[1],"--preorder")==0)
    {
      preorder(&gyoker,0);
    }
    else if(strcmp(argv[1],"--inorder")==0)
    {
      inorder(&gyoker,0);
    }
    else if(strcmp(argv[1],"--postorder")==0)
    {
      postorder(&gyoker,0);
    }
    else
    {
      usage();
    }
  }
  else
  {
```

```
    usage();
  }
  destroy_tree(&gyoker);
  return 0;
}
```

6.4. Tag a gyökér

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

Megoldás videó:

Megoldás forrása: https://github.com/Matchbox1233/Welch/blob/master/4.cpp

Tutorálta: Nyitrai Dávid

Kódot írta: Petrus József Tamás

A C programból C++ lett, és gazdagodott egy binfa osztállyal melynek van public és private része.

```
            #include <iostream>
#include <cstdlib>
#include <ctime>
#include <string.h>

#define null NULL

class Binfa
{
private:
    class Node
    {
    public:
        Node(char c='/')
        {
            this->c=c;
            this->left = null;
            this->right = null;
        }
        char c;
        Node* left;
        Node* right;
    };
    Node* fa;


public:
    Binfa(): fa(&gyoker)
    {
```

```
}

void operator<<(char c)
{
    if(c=='0')
    {
        if(fa->left == null)
        {
            fa->left = new Node('0');
            fa = &gyoker;
        }
        else
        {
            fa = fa->left;
        }
    }
    else
    {
        if(fa->right == null)
        {
            fa->right = new Node('1');
            fa = &gyoker;
        }
        else
        {
            fa = fa->right;
        }
    }
}

void preorder(Node* elem,int depth=0)
{
    if(elem==null)
    {
        return;
    }
    if(depth)
    {
        char *spaces;
        spaces =(char*) malloc(sizeof(char)*depth*2+1);
        for(int i=0;i<depth;i+=2)
        {
            spaces[i]='-';
            spaces[i+1]='-';
        }
        spaces[depth*2]='\0';

        printf("%s%c\n",spaces,elem->c);
    }
```

```
        else
        {
            printf("%c\n",elem->c);
        }
        preorder(elem->left,depth+1);
        preorder(elem->right,depth+1);
    }

    void inorder(Node* elem,int depth=0)
    {
        if(elem==null)
        {
            return;
        }
        inorder(elem->left,depth+1);
        if(depth)
        {
            char *spaces;
            spaces =(char*) malloc(sizeof(char)*depth*2+1);
            for(int i=0;i<depth;i+=2)
            {
                spaces[i]='-';
                spaces[i+1]='-';
            }
            spaces[depth*2]='\0';

            printf("%s%c\n",spaces,elem->c);
        }
        else
        {
            printf("%c\n",elem->c);
        }
        inorder(elem->right,depth+1);
    }

    void postorder(Node* elem,int depth=0)
    {
        if(elem==null)
        {
            return;
        }
        postorder(elem->left,depth+1);
        postorder(elem->right,depth+1);
        if(depth)
        {
            char *spaces;
            spaces =(char*) malloc(sizeof(char)*depth*2+1);
            for(int i=0;i<depth;i+=2)
            {
                spaces[i]='-';
```

```
                spaces[i+1]='-';
            }
            spaces[depth*2]='\0';

            printf("%s%c\n",spaces,elem->c);
        }
        else
        {
            printf("%c\n",elem->c);
        }
    }

    void destroy_tree(Node* elem)
    {
        if(elem==null)
        {
            return;
        }
        destroy_tree(elem->left);
        destroy_tree(elem->right);
        if(elem->c!='/') delete elem;
    }

    Node gyoker;

};

void usage()
{
    printf("Használat: ./binfa KAPCSOLÓ\n");
    printf("Az KAPCSOLÓ lehet:\n");
    printf("--preorder\tA bináris fa preorder bejárása\n");
    printf("--inorder\tA bináris fa inorder bejárása\n");
    printf("--postorder\tA bináris fa postorder bejárása\n");
}

int main(int argc, char** argv)
{
    srand(time(0));
    Binfa bfa;
    for(int i=0;i<100;i++)
    {
        int x=rand()%2;
        if(x)
        {
            bfa<<'1';
        }
        else
        {
            bfa<<'0';
```

```
        }
    }
    if(argc == 2)
    {
        if(strcmp(argv[1],"--preorder")==0)
        {
            bfa.preorder(&bfa.gyoker);
        }
        else if(strcmp(argv[1],"--inorder")==0)
        {
            bfa.inorder(&bfa.gyoker);
        }
        else if(strcmp(argv[1],"--postorder")==0)
        {
            bfa.postorder(&bfa.gyoker);
        }
        else
        {
            usage();
        }
    }
    else
    {
        usage();
    }
    bfa.destroy_tree(&bfa.gyoker);
return 0;
}
```

6.5. Mutató a gyökér

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

Megoldás videó:

Megoldás forrása: https://github.com/Matchbox1233/Welch/blob/master/5.cpp

Tutorálta: Nyitrai Dávid

Kódot írta: Petrus József Tamás

Ismételte az előző feladat lett módosítva: a binfa osztály public részében lévő binfa függvény módosult. Az előző feladatban a gyökér objectum volt, itt pedig pointer aminek értket kell adni hogy hova mutasson.

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

```
#include <string.h>

#define null NULL

class Binfa
{
private:
    class Node
    {
    public:
        Node(char c='/')
        {
            this->c=c;
            this->left = null;
            this->right = null;
        }
        char c;
        Node* left;
        Node* right;
    };
    Node* fa;


public:
    Binfa()
    {
        gyoker=fa=new Node();
    }


    void operator<<(char c)
    {
        if(c=='0')
        {
            if(fa->left == null)
            {
                fa->left = new Node('0');
                fa = gyoker;
            }
            else
            {
                fa = fa->left;
            }
        }
        else
        {
            if(fa->right == null)
            {
                fa->right = new Node('1');
                fa = gyoker;
```

```
            }
            else
            {
                fa = fa->right;
            }
        }
    }

    void preorder(Node* elem,int depth=0)
    {
        if(elem==null)
        {
            return;
        }
        if(depth)
        {
            char *spaces;
            spaces =(char*) malloc(sizeof(char)*depth*2+1);
            for(int i=0;i<depth;i+=2)
            {
                spaces[i]='-';
                spaces[i+1]='-';
            }
            spaces[depth*2]='\0';

            printf("%s%c\n",spaces,elem->c);
        }
        else
        {
            printf("%c\n",elem->c);
        }
        preorder(elem->left,depth+1);
        preorder(elem->right,depth+1);
    }

    void inorder(Node* elem,int depth=0)
    {
        if(elem==null)
        {
            return;
        }
        inorder(elem->left,depth+1);
        if(depth)
        {
            char *spaces;
            spaces =(char*) malloc(sizeof(char)*depth*2+1);
            for(int i=0;i<depth;i+=2)
            {
                spaces[i]='-';
                spaces[i+1]='-';
```

```
            }
            spaces[depth*2]='\0';

            printf("%s%c\n",spaces,elem->c);
        }
        else
        {
            printf("%c\n",elem->c);
        }
        inorder(elem->right,depth+1);
    }

    void postorder(Node* elem,int depth=0)
    {
        if(elem==null)
        {
            return;
        }
        postorder(elem->left,depth+1);
        postorder(elem->right,depth+1);
        if(depth)
        {
            char *spaces;
            spaces =(char*) malloc(sizeof(char)*depth*2+1);
            for(int i=0;i<depth;i+=2)
            {
                spaces[i]='-';
                spaces[i+1]='-';
            }
            spaces[depth*2]='\0';

            printf("%s%c\n",spaces,elem->c);
        }
        else
        {
            printf("%c\n",elem->c);
        }
    }

    void destroy_tree(Node* elem)
    {
        if(elem==null)
        {
            return;
        }
        destroy_tree(elem->left);
        destroy_tree(elem->right);
        if(elem->c!='/') delete elem;
    }
```

```
    Node* gyoker;

};

void usage()
{
    printf("Használat: ./binfa KAPCSOLÓ\n");
    printf("Az KAPCSOLÓ lehet:\n");
    printf("--preorder\tA bináris fa preorder bejárása\n");
    printf("--inorder\tA bináris fa inorder bejárása\n");
    printf("--postorder\tA bináris fa postorder bejárása\n");
}

int main(int argc, char** argv)
{
    srand(time(0));
    Binfa bfa;
    for(int i=0;i<100;i++)
    {
        int x=rand()%2;
        if(x)
        {
            bfa<<'1';
        }
        else
        {
            bfa<<'0';
        }
    }
    if(argc == 2)
    {
        if(strcmp(argv[1],"--preorder")==0)
        {
            bfa.preorder(bfa.gyoker);
        }
        else if(strcmp(argv[1],"--inorder")==0)
        {
            bfa.inorder(bfa.gyoker);
        }
        else if(strcmp(argv[1],"--postorder")==0)
        {
            bfa.postorder(bfa.gyoker);
        }
        else
        {
            usage();
        }
    }
    else
    {
```

```
        usage();
    }
    bfa.destroy_tree(bfa.gyoker);
    return 0;
}
```

6.6. Mozgató szemantika

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

Megoldás videó:

Megoldás forrása: https://github.com/Matchbox1233/Welch/blob/master/6.cpp

Tutorálta: Nyitrai Dávid

Kódot írta: Petrus József Tamás

Nagy szerepet játszik az argomentumátadásban és a visszatérési érték használatában az újonnan beírt másoló konstruktor.

```
#include <iostream>
#include <cmath>
#include <fstream>
#include <vector>


class LZWBinFa {
public:


    LZWBinFa () :fa ( &gyoker ) {

    }
    ~LZWBinFa () {
        std::cout << "LZWBinFa dtor" << std::endl;
        szabadit ( gyoker.egyesGyermek () );
        szabadit ( gyoker.nullasGyermek () );
    }

    LZWBinFa ( const LZWBinFa & regi ) {
        std::cout << "LZWBinFa copy ctor" << std::endl;

        gyoker.ujEgyesGyermek ( masol ( regi.gyoker.egyesGyermek (), regi ↩
            .fa ) );
        gyoker.ujNullasGyermek ( masol ( regi.gyoker.nullasGyermek (),  ↩
            regi.fa ) );
```

```
        if ( regi.fa == & ( regi.gyoker ) )
            fa = &gyoker;

    }

    LZWBinFa ( LZWBinFa && regi ) {
        std::cout << "LZWBinFa move ctor" << std::endl;

        gyoker.ujEgyesGyermek ( regi.gyoker.egyesGyermek() );
        gyoker.ujNullasGyermek ( regi.gyoker.nullasGyermek() );

        regi.gyoker.ujEgyesGyermek ( nullptr );
        regi.gyoker.ujNullasGyermek ( nullptr );

    }


    LZWBinFa& operator<< ( char b ) {

        if ( b == '0' ) {

            if ( !fa->nullasGyermek () ) {
                Csomopont *uj = new Csomopont ( '0' );

                fa->ujNullasGyermek ( uj );

                fa = &gyoker;
            } else {

                fa = fa->nullasGyermek ();
            }
        }

        else {
            if ( !fa->egyesGyermek () ) {
                Csomopont *uj = new Csomopont ( '1' );
                fa->ujEgyesGyermek ( uj );
                fa = &gyoker;
            } else {
                fa = fa->egyesGyermek ();
            }
        }

        return *this;
    }

    void kiir ( void ) {
        /
        melyseg = 0;
```

```
          kiir ( &gyoker, std::cout );
     }

     {
       szabadit (gyoker.egyesGyermek ());
       szabadit (gyoker.nullasGyermek ());

     }

     int getMelyseg ( void );
     double getAtlag ( void );
     double getSzoras ( void );


     friend std::ostream & operator<< ( std::ostream & os, LZWBinFa & bf )  ↩
        {
          bf.kiir ( os );
          return os;
     }
     void kiir ( std::ostream & os ) {
          melyseg = 0;
          kiir ( &gyoker, os );
     }

private:
     class Csomopont {
     public:

          Csomopont ( char b = '/' ) :betu ( b ), balNulla ( 0 ), jobbEgy ( ↩
              0 ) {
          };
          ~Csomopont () {
          };

          Csomopont *nullasGyermek () const {
               return balNulla;
          }

          Csomopont *egyesGyermek () const {
               return jobbEgy;
          }

          void ujNullasGyermek ( Csomopont * gy ) {
               balNulla = gy;
          }

          void ujEgyesGyermek ( Csomopont * gy ) {
               jobbEgy = gy;
          }
```

```
        char getBetu () const {
            return betu;
        }

    private:


        char betu;

        Csomopont *balNulla;
        Csomopont *jobbEgy;

        Csomopont ( const Csomopont & );
        Csomopont & operator= ( const Csomopont & );

    };


    Csomopont *fa;

    int melyseg, atlagosszeg, atlagdb;
    double szorasosszeg;


    void kiir ( Csomopont * elem, std::ostream & os ) {

        if ( elem != NULL ) {
            ++melyseg;
            kiir ( elem->egyesGyermek (), os );

            for ( int i = 0; i < melyseg; ++i )
                os << "---";
            os << elem->getBetu () << "(" << melyseg - 1 << ")" << std:: ↩
               endl;
            kiir ( elem->nullasGyermek (), os );
            --melyseg;
        }
    }

    void szabadit ( Csomopont * elem ) {

        if ( elem != NULL ) {
            szabadit ( elem->egyesGyermek () );
            szabadit ( elem->nullasGyermek () );

            delete elem;
        }
    }
```

```
     Csomopont *  masol ( Csomopont * elem, Csomopont * regifa ) {

          Csomopont * ujelem = NULL;

          if ( elem != NULL ) {
               ujelem = new Csomopont ( elem->getBetu() );

               ujelem->ujEgyesGyermek ( masol ( elem->egyesGyermek (),  ↩
                  regifa ) );
               ujelem->ujNullasGyermek ( masol ( elem->nullasGyermek (),  ↩
                  regifa ) );

               if ( regifa == elem )
                    fa = ujelem;

          }

          return ujelem;
     }

protected:          /
     Csomopont gyoker;
     int maxMelyseg;
     double atlag, szoras;

     void rmelyseg ( Csomopont * elem );
     void ratlag ( Csomopont * elem );
     void rszoras ( Csomopont * elem );

};


int
LZWBinFa::getMelyseg ( void )
{
     melyseg = maxMelyseg = 0;
     rmelyseg ( &gyoker );
     return maxMelyseg - 1;
}

double
LZWBinFa::getAtlag ( void )
{
     melyseg = atlagosszeg = atlagdb = 0;
     ratlag ( &gyoker );
     atlag = ( ( double ) atlagosszeg ) / atlagdb;
     return atlag;
}

double
```

```
LZWBinFa::getSzoras ( void )
{
     atlag = getAtlag ();
     szorasosszeg = 0.0;
     melyseg = atlagdb = 0;

     rszoras ( &gyoker );

     if ( atlagdb - 1 > 0 )
          szoras = std::sqrt ( szorasosszeg / ( atlagdb - 1 ) );
     else
          szoras = std::sqrt ( szorasosszeg );

     return szoras;
}

void
LZWBinFa::rmelyseg ( Csomopont * elem )
{
     if ( elem != NULL ) {
          ++melyseg;
          if ( melyseg > maxMelyseg )
               maxMelyseg = melyseg;
          rmelyseg ( elem->egyesGyermek () );

          rmelyseg ( elem->nullasGyermek () );
          --melyseg;
     }
}

void
LZWBinFa::ratlag ( Csomopont * elem )
{
     if ( elem != NULL ) {
          ++melyseg;
          ratlag ( elem->egyesGyermek () );
          ratlag ( elem->nullasGyermek () );
          --melyseg;
          if ( elem->egyesGyermek () == NULL && elem->nullasGyermek () ==  ↩
             NULL ) {
               ++atlagdb;
               atlagosszeg += melyseg;
          }
     }
}

void
LZWBinFa::rszoras ( Csomopont * elem )
{
     if ( elem != NULL ) {
```

```
            ++melyseg;
            rszoras ( elem->egyesGyermek () );
            rszoras ( elem->nullasGyermek () );
            --melyseg;
            if ( elem->egyesGyermek () == NULL && elem->nullasGyermek () ==  ↩
               NULL ) {
                ++atlagdb;
                szorasosszeg += ( ( melyseg - atlag ) * ( melyseg - atlag )  ↩
                   );
            }
        }
}



void
usage ( void )
{
        std::cout << "Usage: lzwtree in_file -o out_file" << std::endl;
}


void
fgv ( LZWBinFa binFa )
{
        binFa << '1';

        std::cout << binFa;

        std::cout << "depth = " << binFa.getMelyseg () << std::endl;
        std::cout << "mean = " << binFa.getAtlag () << std::endl;
        std::cout << "var = " << binFa.getSzoras () << std::endl;
}

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



        if ( argc != 4 ) {

            usage ();

            return -1;
        }


        char *inFile = *++argv;
```

```
    if ( * ( ( *++argv ) + 1 ) != 'o' ) {
        usage ();
        return -2;
    }


    std::fstream beFile ( inFile, std::ios_base::in );


    if ( !beFile ) {
        std::cout << inFile << " nem letezik..." << std::endl;
        usage ();
        return -3;
    }

    std::fstream kiFile ( *++argv, std::ios_base::out );

    unsigned char b;
    LZWBinFa binFa;


    binFa << '0' << '1' << '0' << '1' << '1' << '1' << '1' << '1' << '1'  ↩
       << '1';

    fgv ( binFa );

    binFa << '0';

    kiFile << binFa;

    kiFile << "depth = " << binFa.getMelyseg () << std::endl;
    kiFile << "mean = " << binFa.getAtlag () << std::endl;
    kiFile << "var = " << binFa.getSzoras () << std::endl;

    LZWBinFa binFa3 = std::move ( binFa );

    kiFile << "depth = " << binFa3.getMelyseg () << std::endl;
    kiFile << "mean = " << binFa3.getAtlag () << std::endl;
    kiFile << "var = " << binFa3.getSzoras () << std::endl;

    kiFile.close ();
    beFile.close ();


    return 0;
}
```

7. fejezet

Helló, Conway!

Fejezetet tutorálta: Nyitrai Dávid

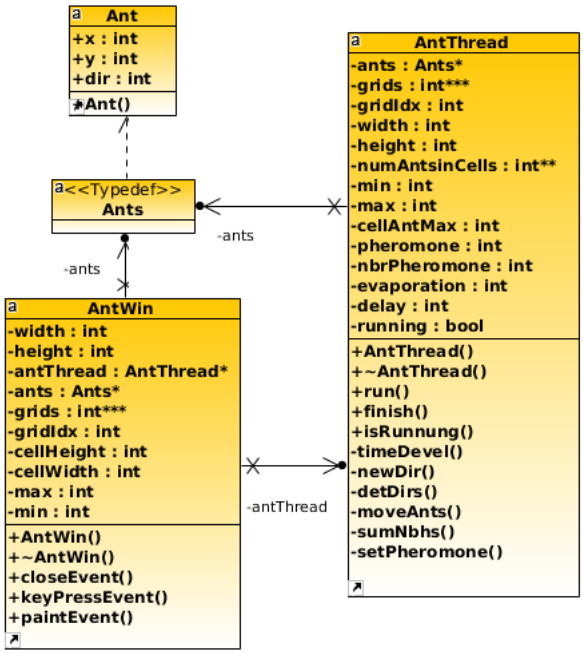
7.1. Hangyaszimulációk

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/nbatfai/bhax/tree/master/attention_raising/Myrmecologist?fbclid=IwAR28U
jJNiCGYkG-fYlLgwqk-YyOyErWjrgrpJPg

Kép forrása: https://bhaxor.blog.hu/2018/10/10/myrmecologist



Tanulságok, tapasztalatok, magyarázat...

A kód négy osztályból áll: Ant, Typdef, AntWin, AntThread.

Az Ant osztály tartalmazza a hangya tulajdonságait. x és y koordinátát, és az irányát (dir).

Typdef: definiálja az Ants típust

AntWin: itt található az ablak méretei (width, hight). Itt kerül meghívásra többek között az AntThread osztály. Az ablak bezárása is itt szerepel. A billentyűzet parancsok is dolgozódnak fel és a rajzolás is itt történik.

AntThread: Itt kerül meghatározásra a hangyák útvonala, mely a feromont közeti ami szintén itt van megadva. Egy cellában a maximális hangyaszám is megszámolásra kerül.

A diagrammon a + jel jelentése, hogy a tag más osztályból is látható, a - meg pedig nem.

7.2. Java életjáték

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

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

7.3. Qt C++ életjáték

Most Qt C++-ban!

Megoldás videó:

Megoldás forrása: Megoldás forrása: https://github.com/Matchbox1233/Conway/blob/master/3.cpp

Tutorálta: Nyitrai Dávid

Kódot írta: Petrus József Tamás

Tanulságok, tapasztalatok, magyarázat...

Program fordítása: g++ 3.cpp -o sfml-app -lsfml-graphics -lsfml-window -lsfml-system

Grid: public részben kerül meghatározásra az ablak nagysága a void draw pedig megrajzolja azt.

Square: A égyzethálón való rajzolásért felelős.

A harmadik rész egy vektor, ami a kijelölt négyzeteket mozgatja, a szabályoknak megdelelően:

Egy sejttel 3 dolog történhet:

-túléli a kört,ha: 2 vagy3 szomszédja van.

-elpusztul, ha: 2-nél kevesebb vagy 3-nál több szomszédja van.

-új születik, ha: pontosan 3 szomszédja van

A negyedik rész, void killall: előző részben használt pointerek kerülnek felszabadításra. while függvény addig fut amíg nyitva van az ablak. Billentyű parancsok: Q: bezárja a programot, C: megöli az összes négyzetet, E: szerkesztő mód.

```
#include <SFML/System.hpp>
#include <SFML/Graphics.hpp>

#include <vector>
#include <iostream>
using namespace sf;
using std::vector;
using std::cout;
using std::endl;


class Grid
{
public:
    Grid(unsigned int x = 1000, unsigned int y = 1000, unsigned int diffs = ↩
        50) : w(x),h(y),diff(diffs)
    {

    }

    void draw(RenderWindow & window)
    {
        for(int i=0;i<w;i+=diff)
        {
            Vertex line[] =
            {
                sf::Vertex(sf::Vector2f(i,0)),
                sf::Vertex(sf::Vector2f(i, h))
            };
            line[0].color = Color(0,0,0);
            line[1].color = Color(0,0,0);
            window.draw(line, 2, sf::Lines);
        }
        for(int i=0;i<h;i+=diff)
        {
            Vertex line[] =
            {
                sf::Vertex(sf::Vector2f(0,i)),
                sf::Vertex(sf::Vector2f(w,i))
            };
            line[0].color = Color(0,0,0);
            line[1].color = Color(0,0,0);

            window.draw(line, 2, sf::Lines);
        }
    }

    unsigned int w;
    unsigned int h;
```

```
    unsigned int diff;
};

class Square
{
public:
    Square()
    {

    }
    Square(int x_pos, int y_pos, float w, bool alive = false)
    {
        square = new RectangleShape(Vector2f(w,w));
        square->setPosition(Vector2f(x_pos,y_pos));
        aliveState = alive;
    }
    /*Square (const Square& other )
    {
        if(this != &other)
        {
            delete this->square;
            this->square = other.square;
        }
    }
    Square& operator=(const Square& other)
    {
        if(this != &other)
        {
            delete this->square;
            this->square = other.square;
        }
        return *this;
    }*/
    ~Square()
    {
        delete square;
    }

    void update()
    {
        if(aliveState)
        {
            square->setFillColor(Color::Black);
        }
        else
        {
            square->setFillColor(Color::White);
        }

    }
```

```
    void setFill(Color c = Color::White)
    {
        square->setFillColor(c);
    }

    void draw(RenderWindow &window)
    {
        window.draw(*square);
    }
    RectangleShape* square;

    bool aliveState;
private:

};

vector<vector<Square*>> update(vector<vector<Square*>> v)
{
    vector<vector<Square*>> tmp ;//= v;

    for(int i=0;i<v.size();i++)
    {
        tmp.push_back(vector<Square*>());
        for(int j=0;j<v[0].size();j++)
        {
            tmp[i].push_back(new Square(v[i][j]->square->getPosition().x,v[ ↩
               i][j]->square->getPosition().y,v[i][j]->square->getSize().x, ↩
               v[i][j]->aliveState));
        }
    }

    for(int i=0;i<v.size();i++)
    {
        for(int j=0;j<v[0].size();j++)
        {
            int live_neighbours = 0;
            live_neighbours +=  v[(i-1)%v.size()][(j-1)%v[0].size()]-> ↩
               aliveState;
            live_neighbours +=  v[(i-1)%v.size()][(j)%v[0].size()]-> ↩
               aliveState;
            live_neighbours +=  v[(i-1)%v.size()][(j+1)%v[0].size()]-> ↩
               aliveState;

            live_neighbours +=  v[(i)%v.size()][(j-1)%v[0].size()]-> ↩
               aliveState;
            live_neighbours +=  v[(i)%v.size()][(j+1)%v[0].size()]-> ↩
               aliveState;

            live_neighbours +=  v[(i+1)%v.size()][(j-1)%v[0].size()]-> ↩
```

```
                aliveState;
            live_neighbours +=  v[(i+1)%v.size()][(j)%v[0].size()]-> ↩
                aliveState;
            live_neighbours +=  v[(i+1)%v.size()][(j+1)%v[0].size()]-> ↩
                aliveState;
            //cout <<" X:"<<i << " y:"<< j << " Live neighbours:"<<  ↩
                live_neighbours<<endl;
            if(v[i][j]->aliveState)
            {
                if(live_neighbours < 2)
                {
                    tmp[i][j]->aliveState = false;
                }
                else if(live_neighbours > 3)
                {
                    tmp[i][j]->aliveState = false;
                }
            }
            else
            {
                if(live_neighbours == 3)
                {
                    tmp[i][j]->aliveState = true;
                }
            }
        }
    }

    return tmp;
}


/*void update(vector<vector<Square*>> &v)
{
    vector<vector<Square*>> tmp = v;

    for(int i=0;i<v.size();i++)
    {
        for(int j=0;j<v[0].size();j++)
        {
            int live_neighbours = 0;
            live_neighbours +=  v[(i-1)%v.size()][(j-1)%v[0].size()]-> ↩
                aliveState;
            live_neighbours +=  v[(i-1)%v.size()][(j)%v[0].size()]-> ↩
                aliveState;
            live_neighbours +=  v[(i-1)%v.size()][(j+1)%v[0].size()]-> ↩
                aliveState;
            live_neighbours +=  v[(i)%v.size()][(j-1)%v[0].size()]-> ↩
                aliveState;
            live_neighbours +=  v[(i)%v.size()][(j+1)%v[0].size()]-> ↩
```

```
                aliveState;
            live_neighbours +=  v[(i+1)%v.size()][(j-1)%v[0].size()]-> ↩
                aliveState;
            live_neighbours +=  v[(i+1)%v.size()][(j)%v[0].size()]-> ↩
                aliveState;
            live_neighbours +=  v[(i+1)%v.size()][(j+1)%v[0].size()]-> ↩
                aliveState;

            //cout <<" X:"<<i << " y:"<< j << " Live neighbours:"<<  ↩
                live_neighbours<<endl;
            if(v[i][j]->aliveState)
            {
                if(live_neighbours < 2)
                {
                    //dead.push_back(Vector2i(i,j));
                    tmp[i][j]->aliveState = false;
                }
                else if(live_neighbours > 3)
                {
                    tmp[i][j]->aliveState = false;
                }
            }
            else
            {
                if(live_neighbours == 3)
                {
                    //nextGen[i][j] = true;
                    tmp[i][j]->aliveState = true;
                }


            }

        }
    }

    v=tmp;

}*/

/*vector<vector<Square*>> update(vector<vector<Square*>> v)
{
    vector<vector<Square*>> tmp = v;
    int size1 = v.size(), size2 = v[0].size();
    for(int i=0;i<size1;i++)
    {
        for(int j=0;j<size2;j++)
        {
            int alive = 0;
            for(int k=-1;k<=1;k++)
```

```
            {
                for(int l = -1; l<=1;l++)
                {
                    if(!(l==0 && k ==0))
                    {
                        alive += v[(i+k)%size1][(j+l)%size2]->aliveState;
                    }
                }
            }
            if(v[i][j]->aliveState)
            {
                if(alive < 2)
                {
                    tmp[i][j]->aliveState = false;
                }
                else if(alive > 3)
                {
                    tmp[i][j]->aliveState = false;
                }
            }
            else
            {
                if(alive == 3)
                {
                    tmp[i][j]->aliveState = true;
                }
            }
        }
    }
}*/

void killall(vector<vector<Square*>> &v)
{
    for(int i=0;i<v.size();i++)
    {
        for(int j=0;j<v[0].size();j++)
        {
            v[i][j]->aliveState=false;
        }
    }
}

int main()
{
    RenderWindow window(VideoMode(1000,1000),"Game of Life");
    window.setFramerateLimit(10);
    window.setActive();

    Vector2u size = window.getSize();
```

```
    Grid g(size.x,size.y,1000/40);


    int h = g.h/g.diff+1;
    int w = g.w/g.diff+1;
    //Square squares[h][w];

    std::vector<std::vector<Square*>> squares;

    bool edit_mode = true;


    for(int i=0;i<h;i++)
    {
        squares.push_back(vector<Square*>());
        for(int j=0;j<w;j++)
        {
            squares[i].push_back(new Square(i*g.diff+1,j*g.diff+2,g.diff-3) ↩
               );
        }
    }
    //squares[4][5]->aliveState=true;


    while (window.isOpen())
    {
        window.clear(sf::Color::White);
        // check all the window's events that were triggered since the last ↩
            iteration of the loop
        sf::Event event;
        while (window.pollEvent(event))
        {
            // "close requested" event: we close the window
            if (event.type == sf::Event::Closed)
            {
                window.close();
            }
            else if(event.type == Event::MouseButtonPressed)
            {
                if(edit_mode && event.mouseButton.button == Mouse::Button:: ↩
                   Left)
                {
                    /*cout<<event.mouseButton.x<<" "<<event.mouseButton.y<< ↩
                       endl;
                    cout<<event.mouseButton.x/g.diff<< " "<< event. ↩
                       mouseButton.y/g.diff<<endl;*/
                    squares[event.mouseButton.x/g.diff][event.mouseButton.y ↩
                       /g.diff]->aliveState= !squares[event.mouseButton.x/g ↩
                       .diff][event.mouseButton.y/g.diff]->aliveState;
                    cout<< "Changed state on entity at X:"<< event. ↩
```

```
                    mouseButton.x/g.diff << " Y:"<<event.mouseButton.y/g ↩
                    .diff << " to "<< (squares[event.mouseButton.x/g. ↩
                    diff][event.mouseButton.y/g.diff]->aliveState? " ↩
                    Alive" : "Dead")<<endl;
            }
        }
        else if(event.type == Event::KeyPressed)
        {
            if(event.key.code == Keyboard::Q)
            {
                cout<<"Close request recieved. Application will exit." ↩
                    <<endl;
                window.close();
            }
            if(edit_mode && event.key.code == Keyboard::C)
            {
                cout<< "Killed all entities." <<endl;
                killall(squares);
            }
            if(event.key.code == Keyboard::E)
            {
                edit_mode = !edit_mode;
                if(edit_mode)
                {
                    cout<< "Changed to edit mode."<<endl;
                }
                else
                {
                    cout<< "Changed to simulation mode."<<endl;
                }

            }
        }


    }

    /*s.draw(window);
    s.square->setPosition(Vector2f(s.square->getPosition().x+1,s.square ↩
       ->getPosition().y));*/
    g.draw(window);

    for(int i=0;i<h;i++)
    {
        for( int j=0; j<w;j++)
        {
            squares[i][j]->draw(window);
        }
    }
```

```
        window.display();
        if(!edit_mode) squares = update(squares);
        for(int i=0;i<h;i++)
        {
            for( int j=0; j<w;j++)
            {
                squares[i][j]->update();
            }
        }
    }

    return EXIT_SUCCESS;
}
```

7.4. BrainB Benchmark

Megoldás videó:

Megoldás forrása: https://github.com/nbatfai/esport-talent-search?fbclid=IwAR0n58sBL-LrZ5N9rpC-zftv0CeiW

Tutorálta: Nyitrai Dávid

Tanulságok, tapasztalatok, magyarázat...

A program indításakor megjelenik 4 darab négyzet, közepükön egy darab kék körrel. A feladat az hogy a bal egérgombot lenyomva a körön tartsuk a kurzort, miközben az mozog. Ha sikerül rajta tartani akkor a négyzetek száma nőni fog, ha pedig nem akkor csökkenni. A feladat nehézsége abban rejlik, hogy ha sok négyzet van a képernyőn akkor fedni is fogják egymást.

8. fejezet

Helló, Schwarzenegger!

8.1. Szoftmax Py MNIST

aa Python

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

8.2. Szoftmax R MNIST

R

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

8.3. Mély MNIST

Python

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

8.4. Deep dream

Keras

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

8.5. Robotpszichológia

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

9. fejezet

Helló, Chaitin!

9.1. Iteratív és rekurzív faktoriális Lisp-ben

Megoldás videó:

Megoldás forrása:

9.2. Weizenbaum Eliza programja

Éleszd fel Weizenbaum Eliza programját!

Megoldás videó:

Megoldás forrása:

9.3. Gimp Scheme Script-fu: króm effekt

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

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

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

Tanulságok, tapasztalatok, magyarázat...

9.4. Gimp Scheme Script-fu: név mandala

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

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

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

Tanulságok, tapasztalatok, magyarázat...

9.5. Lambda

Hasonlítsd össze a következő programokat!

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

9.6. Omega

Megoldás videó:

Megoldás forrása:

10. fejezet

Helló, Gutenberg!

10.1. Juhász István: Magas szintű programozási nyelvek 1 (Pici könyv)

2.4. Adattípusok

Az adattípus egy konkrét programozási eszköz egy része. Ennek van neve, ami egy azonosító. Attól függően, hogy egy nyelv ismeri-e az adattípust, lehet típusos és nem típusos. A előbbieknek vannak beépített -úgynevezett standard- típúsai. Bizonyos nyelvekben mi is létrehozhatunk típusokat. Az adattípusoknak két nagy csoportjuk van: skalár/egyszerű és strukturált/összetett.

2.5. A nevesített konstans

A nevesített konstanak három alkotórésze van: név, típus, érték. Ezt mindig deklarálni kell. A program szövegében a nevével jelenik meg, és az értékét jelenti. A nevesített konstans értéké a deklarációnál kell megadni, és ezt később nem lehet változtatni.

2.6. A változó

A változónak négy komponense van: név, attribútumok, cím, érték. A névhez rendeljük hozzá a 3 másik értéket. A változóhoz a deklarációval attribútum rendelődik, ami a változó viselkedését futás közben határozza meg. Ez lehet: Implicit deklaráció, Explicit deklaráció, Automatikus deklaráció.

A változó értékének a tárban elfoglalt helyét a cím határozza meg, amit az alábbi módokon lehet hozzárendelnii a változóhoz: Statikus tárkiosztás, Dinamikus tárkiosztás, A programozó által vezérelt tárkiosztás.

2.7. Alapelemek az egyes nyelvekben

C nyelvben egyszerű típus az aritmetikai, az összetett pedig a származtatott. Bool típusa nincs, ahol igaz/-hamis érték szükséges ott az int 0 a hamis és bármely más int az igaz. A void típus tartománya üres, így nincs reprezentációja és se műveletei.

3. KIFEJEZÉSEK

A kifejezések szintaktikai eszközök, két részük van, érték és típus. Formális összetevők: operandusok, operátorok, kerek zárójelek. A kifejezéseket 3 alakban lehet felírni, attól függően hol helyezkedik el az operátor: prefix alakban az legelől, infix alakban középen és postfix alakban leghátul. A műveletek végrehajtási sorrendje lehet: balról-jobbra vagy jobbról-balra

4. UTASÍTÁSOK

Az utasításoknak két nagy csoportjuk van: a deklarációs és a végrehajtható utasítások. Az előbbiek a fordítóprogramnak szólnak. Az utóbbiakból generálja a fordítóprogram a tárgykódot. Végrehajtható utasítások csoportjai: Értékadó utasítás, Üres utasítás, Ugró utasítás, Elágaztató utasítások, Ciklusszervező utasítások, Hívó utasítás, Vezérlésátadó utasítások, I/O utasítások, Egyéb utasítások.

1. Értékadó utasítás

Beállítja vagy módosítja egy változó értékkomponensét.

2. Üres utasítás

Egy üres utasításra a processzor egy üres gépi utasítást hajt végre.

3. Ugró utasítás

Az ugró utasítással a program egy adott pontjáról egy adott címkével ellátott végrehajtható utasításra adhatjuk át a vezérlést.

4. Elágaztató utasítások

-Kétirányú elágaztató utasítás

A kétirányú elágaztató utasítással el lehet dönteni, hogy két tevékenység közül melyiket válasszuk, illetve ,hogye egy tevékenységet végrehajtsuk-e.

-Többirányú elágaztató utasítás

A többirányú elágaztató utasítással egymást kölcsönösen kizáró akárhány tevékenység közül egyet végrehajthatunk.

5. Ciklusszervező utasítások

A ciklusszervező utasítással egy bizonyos tevékenységet tetszés szerint megismételhetünk.

Az alábbi ciklusfajtákat különböztetjük meg:

-Feltételes ciklus

Ennél az ismétlődést egy igaz vagy hamis érték szabályozza a feltételnek megfelelően.

-Előírt lépésszámú ciklus

Ennél az ismétlődést a ciklusváltozó határozza meg, aminek megadjuk, hoy változzon és meddig.

-Felsorolásos ciklus

Ennél ciklusváltozó értékeit egyesével megadhatjuk.

-Végtelen ciklus

A végtelen ciklus ismétlődése magától nem áll le. A magban kell olyan utasítást megadni, amelyi megállítja a ciklust.

-Összetett ciklus

Az előző ciklusfajták összetételéből áll össze.

6. Hívó utasítás

7. Vezérlésátadó utasítások

8. I/O utasítások

9. Egyéb utasítások

5. A PROGRAMOK SZERKEZETE

Az eljárásorientált nyelvekben a program szövege hozzávetőlegesen független részekre, vagyis programegységekre tagolható, amik az alábbiak lehetnek: alprogram, blokk, csomag, taszk. Az alprogramot egyszer kell megírni, majd a programban ahol szükség van rá csak hivatkozni kell. Így elkerülhető, hogy többször kelljen megírni ugyan azt a programrészt.

Formálisan az alprogram így épül fel: fej vagy specifikáció, törzs vagy implementáció, vég Az alprogram, mint programozási eszköz négy részbőlből áll: név, formális paraméter lista, törzs, környezet. Az alprogramoknak két fajtája van: eljárás és függvény. Az eljárás egy tevékenységet hajt végre és elhelyezhető bárhol, ahol egy végrehajtható utasítás is. Akkor fejeződik be ha elérjük a végét vagy külön utasításra. A függvény egyetlen értéket határoz meg.

Egy programegység bármikor meghívhat egy másikat, az pedig egy újabbat... Így alakulhat ki egy hívási lánc. A hívási lánc összes tagja aktív, viszont csak a legutolsó programegység működik. Legelőször az utoljára meghívott egység fejezi be a működését, majd a vezérlés visszatér az előzőbe. Rekurziónak nevezzük mikor egy még aktiv alprogramot hívunk meg. A rekurzó lehet közvetlen és közvetett. Egyes nyelvekben egy alprogramot meghívni nem csak a fejen keresztül lehet, hanem a törzsben is ki lehet alakítani ún. másodlagos belépési pontokat. A programegységek egyfajta kommunikációját paraméterátadással lehet megvalósítani, aminél van egy hívó és egy hívott ami mindig alprogram. Az alábbi paraméterátadási módok léteznek: érték szerinti, cím szerinti, eredmény szerinti, érték-eredmény szerinti, név szerinti, szöveg szerinti Típust nem lehet átadni paraméterként, ha alprogramokról van szó. Az alprogramok formális paramétereit három csoportra bonthatjuk: input paraméterek, output paraméterek, input-output paraméterek.

A blokk egy paraméter nélküli programegység ami csak egy másik egységben jelenhet meg, önállóan nem. Bármely olyan helyen elhelyezhető ahol egy végrehajtható utasítás is. Amely nyelvekben lehet neve, ott általában az a címke ami a kezdet előtt áll. GOTO utasítással is lehet aktivizálni is befejezni. Mint általában ha rákerül a vezérlés akkor aktivizálódik, ha eléjük a program végét akkor befejeződik.

Nevekhez kapcsolódik a hatáskör. Kétféle hatáskörkezelés létezik, a statikus és a dinamikus. A hatáskör terjedése csak és kizárólag befelé irányul.

A fordítási egységek önnállóan lefordíthatóak, az eljárásorientált nyelven írt program ilyen egységekből épül fel.

6. ABSZTRAKT ADATTÍPUS

Annak érdekében, hogy az értékeket se véletlenül, se pedig szándékosan ne lehessen elrontani, találták ki a bezárást vagy információ rejtést, amit az absztrakt adattípus valósít meg.

9. KIVÉTELKEZELÉS

Az operációs rendszertől át lehet venni a megszakítások kezelését, ezt a célt szolgálja a kivételkezekési eszközrendszer. Így megoldható az eseményvezérlés a programozásban. A kivételkezelést a program végzi, ha olyan események következnek be, amik megszakítást okoznak, vagyis kivételek. A kivételeknek legtöbbször van nevük és kódjuk. Némely kivétel figyelése engedélyezhető vagy letiltható. Letiltásnál a program egyszerűen figyelmen kívül hagyja.

10. GENERIKUS PROGRAMOZÁS

A generikus programozás bármely programozási nyelvbe beépíthető, lényege, hogy megadunk egy paraméterezhető forrásszöveg-mintát amit a fordító fog kezelni. Tetszőleges számú lefordítható szöveg állítható elő a mintaszövegből paraméterek segítségével. A mintaszöveg paraméterezhető típussal is.

11. PÁRHUZAMOS PROGRAMOZÁS

A párhuzamos programozás nyelvi alapfogalmai:

-Kommunikáció: A folyamatok között adatcsere zajlik.

-Szinkronizáció: Előfordulhat a párhuzamosan futó folyamatoknál, hogy ahoz, hogy az egyik továbblépjen szüksége van a másiktól valamilyen információra. Ahoz hogy kommunikálni tudjanek egymással, szinkronizációs pontokra van szükség, ahol létre jöhet az adatcsere.

-Konkurencia: Előfordul, hogy több erőforrásra lenne szükség, mint ami rendelkezésre áll, így a folyamatoknak versengenie kell.

-Kölcsönös kizárás: Egyszerre több folyamat nem férhet hozzá ugyan ahoz az adathoz.

A párhuzamos programozáshoz egy nyelvnek eszközökre van szüksége az alábbiak kivitelezésére: folyamatok elindítása és befejeztetése, folyamatok prioritásának meghatározása, folyamatok kódjának megadása, kölcsönös kizárás kérése, kommunikáció megvalósítása, szinkronizáció, folyamatok mőködésének felfüggesztése, folyamatok ütemezése.

13. INPUT/OUTPUT

A perifériákkal való kommunikációt az I/O eszközrendszer végzi, adatokat küld és fogad, ilyenkor a tárban és a periférián is van egy fajta ábrázolási mód. Az adatátvitel lehet folyamatos vagy bináris/rekord módú. Az előbbi esetén a ét helyen eltér az ábrázolás. kialakult eszközrendszerek: formátumos módú adatátvitel, szerkesztett módú adatátvitel, listázott módú adatátvitel. Az I/O-ban az állomány központi helyen foglal helyet. Funkció szempontjából ez lehet: input állomány, output állomány, input-output állomány. Valamilyen programban az állományok használatához a következők szükségesek: Deklaráció, Összerendelés, Állomány megnyitása, Feldolgozás, Lezárás.

10.2. Juhász István: Magas szintű programozási nyelvek 2

Kivételkezelés a Javaban

A Java működése közben módszerek hívodnak meg, ha ezek futása köben speciális esemény következik be, akkor létrejön egy kivétel-objektum. Ekkor a kivétel a módszertől -mely a futását be is fejezi- átkerül a Java virtuális gép hatáskörébe. A JVM feladata megtalálni egy megfelelő típust, egy kivételkezelőt ami képes kezelni a fennálló kivételt. A kivételkezelő akkor jó típusú, ha típusa megegyezik, vagy őse a kivétel típusának. A kivételkezelő egy blokk, mellyek egymásba ágyazhatóak és bárhol elhelyezhetők.

10.3. Brian W. Kernighan – Dennis M. Ritchie: A C programozási nyelv (K and R könyv)

Vezérlési szerkezetek

3.1. Utasítások és blokkok

A ";" utasítás lezárójel

A "{""}" utasításokat és deklarációkat foglal össze, blokk esetén utána nincs ";"

3.2. Az if-else utasítás

If kerül kiértékelésre ha a kifejezés igaz, az else ha hamis. Az else opcionális és ahoz az ifhez tartozik ami a legközelebb van hozzá fentről, ha nincs zárójelezés. A fordítóprogram a tagolást nem veszi figyelembe.

3.3. Az else-if utasítás

Töbszörös döntést lehet létrehozni több if-else egymásba ágyazásával.

3.4. A switch utasítás

A switch utasításban van elhelyezve több case utasítás és egy default. A switchben lévő kifejezés lesz összehasonlítva a case kifejezéseivel, ahol ez a kettő megegyezik abban a caseben lévő utastás lesz végrehajtva, ha sehol nincs egyezés akkor a default ég kerül végrehajtásra. A default elhagyható, ebben az estben ha nincs egyezés akkor nem történikk semmi.

3.5. Ciklusszervezés while és for utasítással

A while utasítást át lehet írni for utasításnak és fordítva, a következő módon: while utasítás felépítése:

1. kifejezés;

while(2. kifejezs)

utasítás;

3.kifejezés;

For utasítás felépítése:

for(1.kifejezés;2.kifejezés;3.kifejezés)

utasítás;

3.6. Ciklusszervezés do-while utasítással

A do-while utasítás előbb hajtja végre az utasítást aztán ellenőrzi, hogy megfelelt-e a feltételnek, így biztosítja, hogy a ciklus egyszer minimum lefut.

3.7. A break és continue utasítások

A break utasítással meg lehet szakítani a do, for, while és switch utasításokat. Több egymásba ágyazott ciklus esetén csak azt szakítja meg amibe bele van írva. A continue utasítás csak ciklusokban működik, hatására a cilkus további utasításai átugrásra kerülnek, majd a ciklus újra lefut.

3.8. A goto utasítás és a címkék A goto utacítással egy címkére ugorhatunk. A címke úgy alakítható ki, mint egy változó, és egy utasítás előtt áll. A goto képes több egymásba ágyazott ciklus legbelsejéből egy lépésben kilépni a break utasítással ellentétben. Azonban szinte mindig át lehet úgy írni a programot, hogy ne kelljen ezt az utasítást használni. Amennyiben lehetséges általában kerülendő, mivel a programot nehezebben követhetővé teszi.

10.4. Benedek Zoltán - Levendovszky Tihamér: Szoftverfejlesztés C++ nyelven (BME C++ könyv)

(1-16)

A C++ a C továbbfejlesztett verziója, kényelmesebb és biztonságosabb is.

Egy függvény üres paraméterlistával definiálva C nyelven tesztőlege számú paraméterrel hívható, C++ nyelven azonban egy void paraméter megadásával egyenértékű.

Bevezetésre került a bool típusú változó ami true vagy false értéket vehet fel. A wchar_t pedig beépített típus lett. C++-ban ahol utasítás állhat, ott változót is lehet deklarálni. Így a kód áttekinthetőbb. Így a változókat csak aztán lehet használni ahol deklarálva lettek, egészen a blokk végéig.

Újítás a C-hez képest, hogy lehetséges két azonos nevű függvény létrehozása, amennyiben az argomentum listájuk nem egyezik meg. A függvényeknek alapértelmezett argomentum értékeket is meg lehet adni.

(17-59)

Objektunok és osztályok

Az egységbe zárással a struktúrának lehetnek adattagjai (tagváltozó), és tagfüggvényei is. A tagváltozót lehet még attribútumnak, a tagfüggvényt metódusnak vagy műveletnek nevezni.

A tagváltozók a struktúra adattagjai. C++-ban nem lehet úgy kihasználni a memóriaképet mint C-ben ugyanis előfordul, hogy a tagváltozó után más adatot is tárolni kell.

Az osztálydefinícióban lehet megadni a tagfüggvényeket, valamint a struktúradefiníción kívül is megadhatjuk a függvény törzsét.

A C++ a struktúra koncepcióját fejlesztette tovább a változók és függvények egységebe zárásának megvalósításához.

Az adatrejtést lehet megvalósítani a private kulcsszóval, mivel az utána deklarált tagváltozók és függvények kívülről nem láthatók sem hozzáférhetők. A publik deklaráltak azonban már hozzáférhetők. Érdemes csak annyi hozzáférést engedni a változókhoz ami a felhasználónak feltétlen szükséges. Itt érdemes megjegyezni, hogy a struktúra alapértelmezésben publikus, az osztály azonban private. Az osztály változói az objektumok.

A konstruktor egy speciális tagfüggvény, a neve pedig megegyezik az osztály nevével, a konstruktor automatikusan meghívódik az osztály példányosításakor (meghívni nem lehet). A konstruktorokra is igaz az mint a függvényekre, azonosításuk név és paraméterlist alapján történik. Így lehetséges egyszerre több, különböző paraméterlistával rendelkező konstruktor megadása. Alapértelmezett konstruktornak nevezzük azt amelynek nincs argomentuma, amyelynek pedig csak egy van az a konverziós konstruktor. A konstruktor végzi az inicialiálást, így inicializáláshoz lehet használni a beépített típusok konstruktorát is. A destruktor feladata az objektum által használt erőforrások felszabadítása, mikor az objektum maga megszűnik vagy a blokknak vége lesz.

Dinamikus memóriakezelésre van szükség, ha nem adjuk meg a tároló méretét, akkor addig lehet bele elemeket pakolni amíg van elég szabad memória.

C-ben ezt a malloc és a free függvényekkel végezhettük és ezek variánsaival. C++-ban pedig a new nevű opertátor felelős a dinamikus memóriakezeléséert. A new osztályok példányosítására is alkalmas. A lefoglalt hely a delete operátorral szabadítható fel.

Ha a delete operátorokat olyan nem nulla értékű pointerre hívjuk me, amelyet nem valamelyik new operátor foglalt le, kiszámíthatatlan lesz a programunk működése.

Elemek betétele és kivétele: Egy elem betételekor két est léphet fel:a tároló mérete az új elemmelegyütt vagy meghaladja vagy nem haladja meg az előre lefoglalt méretet. Ha meghaladja akkor le kell foglalni egy nagyobb méretű tömböt amelyre egy ideiglenes pointerrel hivatkozunk, majd egy ciklussal az adatokat átmásoljuk, aztán az új elemet hozzáadjuk a tömb végéhez, végül az eredeti tömböt felszabadítjuk és a pointert az új területre irányítjuk. Ha van elég hely akkor csak hozzáfűzzük a tömb végére az elemet. Elem kivételekor előfordulhat hogy a tároló üres, ilyenkor hibával térünk vissza.

A másolókonstruktor egy speciális konstruktor így vele is lehet inicializálni objektumokat, azonban ezzel egy már létező objektum alapján hoz létre egy új objektumot. Valamint függvényparaméter érték szerinti átadásánál, az adott változót lemásolja, és a másolatot a függvénytörzsben való használat után felszabadítja. A beépített típusok másolása estén a fordító bitenként átmásolja az adott című és méretű memória területet. A bitenkénti másolás neve sekély másolás. Mély másolás pedig, mikor a dinamikus adattagok is másolásra kerülnek.

C++-ban megvan a lehetőség hogy a private osztályok, kívűl álló függvényenek vagy más osztályoknak hozzáférész bizosítson saját tagváltozóihoz és tagfüggvényeihez.

Ezt a friend kulcsszóval lehet megtenni. Az ilyen függvények a friend függvénynek, az osztályok pedig friend osztályok. Melyeknek épp olyan hozzáférési joga van mintha az adott osztály tagfüggvényei lennének.

Az értékadás és az inicializálás C++-ban nem ugyan azt jelenti. Inicializálás az objektumok és áltozók létrehozásakor történik. Az értékadásnál pedig egy már létező objektum vagy változó kap új értéket. Tagváltozókat is lehet inicializálni, erre van a konstruktorok inicializálási listája.

Speciális osztályváltozókat (statikus taggváltozókat) is lehet definiálni ami nem az osztály objektumához, hanem az osztályhoz kapcsolódik. Ehez nem szükséges hogy az osztálynak legyenek objektumai. Lényege, hogy az objektumok azonos értéket vesznek fel. Megvalósításához a static kulcsszó szükséges. Ezt függvényekkel is meg lehet csinálni. A statikus tagfüggvényekből azonban ami nem statikus az nem érhető el.

Akkor ajánlott ezt használni mikor az osztály minden objektumára közös változóra van szükségünk.

Beágyazott definíció az osztálydefiníción belüli megadása a enumeráció-, osztály-, és típusdefinícióknak.

(73-90)

A C++ I/O alapjai

A C nyelvnek 3 File típusú magas szintű állományleírója van: stdin (bemenet) ami csak olvasható, stdout (kimenet) és stderr (hibakimenet) ezen kettő csak írható. A C++ban ezek helyett objektumok vannak, valamint a nyelv adatfolyamokban gondolkodik, melyek operátorokat használnak kiírásra és beolvasásra. A "nyilak" mindig arra mutatnak amerre az adat áramlik. A cout irányába amikor kiiratás történik, és a cintől elfelé, amikor beolvasás történik. A cin működését javasolt ellenőrizni, ugyanis csak addig olvas amíg úlyan típusú karaktert kap amilyet vár is.

Hibák naplózására szolgál a clog, aminek nincs C megfelelője, ez a cerr bufferelt változata.

Egy isostate típusú tagváltozó jelzi az adatfolyam állapotát, amit konstruktorokkal lehet beállítani: eofbit: az állomány végét jelenti.

failbit: valamilyen kisebb hibát jelez.

badbit: valamilyen komoly hibát jelez.

goodbit: az előző három nem következett be.

Ha bármelyik beállításra kerül, akkor az utána lévő írási és olvasási műveletek nem hajtódnak végre.

Egy adatfolyam manipulálásához használhatunk manipuéáturokat. Egy ismert manipulátor például az endl, mely mielőtt kiüríti a buffert még elhelyez egy sor vége karaktert az adatfolyamban. A cin alap beállításain is lehet változtatni. a noskipws manipulátor hatására nam hagyja el a whitespace karaktereket, ideiglenes visszaállításhoz a ws, végleges vissszaállításhoz a skipws manipulátorok szükségesek.

A C szabványos állománykezelése egy File típusú leíró köré csoportosul, amelyet a fopen függvény ad vissza. A C++ban egy objektum zárja egységbe a leírót, az állományműveleteket pedig a tagfüggvények használatával végezhetők. A C++ adatolyamokat használ az állománykezeléshez is.

A C++ megtalálható az átirányítás funkció is. Ehhez meg kell nyitni az állományt, és az átiránytandó adatfolyam formázási beállításait átmásolni a megnyitott állomány adatfolyamába.

(93-96)

Operátorok és túlterhelésük.

C nyelvben mellékhatásnak nevezzük mikor egy operátor az argomentuma értékét is megváltozatja, ugyanis alap esetben az argomentumon végzett művelet eredményét a visszatérítési érték adja meg, az argumentum pedig nem változik. Az operátorok előre meghatározott sorrendben kerülnek kiértékelésre, amin zárójelekkel lehet változtatni. A C++ a C-hez képest több operátort is ismer, működésükben rugalmas.

A C nyelvben a függvény nem képes mellékhatásra, a C++ ban viszont igen. A függvényszintaxis segítségével az operátorok működését a függvényekkel megegyező módon lehet definiálni. Mivel az operátorok speciális függvények, és a függvénynevek túlterhelhetők, így az operátornevek is túlterhelhetők.

(187-197)

A C-ben a hibák kezelése a függvények által visszaadott hibakódok vagy egy globális változóban tárolt hibakód alapján történik, aminek megvannak a hátrányai. Egy hívási láncban feltűnő hibakódot a lánc minden szintjén ellenőrizni kell. Több hibakód keveredhet. Könnyen figyelmen kívül lehet hagyni egy hibát amivel a program hibásan de tovább tud futni. A C++-ban a hibakezelés is továbbfejleszésre került.

A kivételkezelés biztosítja, hogy ha valahol hibát találunk, akkor a futás a hibakezeló ágon folytatódjon. Nevéből adódóan, nem csak hibákat hanem kivételeket is lehet kezelni ezzel a módszerrel.

A könyvben található példa:

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

int main()
{
  try
  {
   double d;
   cout << "Enter a nonzero number: ";
   cin >> d;
   if(d == 0)
     throw "The number con not be zero.";
   cout << "The reciprocal is: " << 1/d << endl;
  }
  catch (const char* exc)
  {
    cout << "Error! The error text is: " << exc <<endl;
  }
  cout << "Done." << endl;
}
```

A pogram a beírt szám reciprokát adja vissza. A nullát nem lehet beírni, itt ezt kezeljük. A kód tartalmaz egy try-catch blokkot. A tryon belülre kerül a throw utasítás ami a kivételt fogja dobni, ha ez az utasítás lefut, akkor a catch ág következik, ami típus alapján kapja el a kivételeket. Ha a throw utasítás nem hajtódik végre akkor a catch sem.

A try-catch blokkok egymásba ágyazhatók.

A verem visszacsévélésére egy a könyvben szereplő példa:

```
int main()
{
 try
 {
  f1();
 }
 catch(const char* errorText)
 {
   cerr << errorText << endl;
 }
}

void f1()
{
 Fifo fifo; // T.f.h a Fifo egy általunk megírt osztály
 f2();
 ...
}

void f2()
{
  int i = 1;
  throw "error1";
}
```

Előszöris az f2 dob egy kivételt, majd felszabadul az i lokális változó. Felszabadul az f1-ben lefoglalt Fifo fifo objektum, ugyanis meghívódik a destruktora. Végül lefut a catch blokk. Ebből látszik, hogy a kivétek dobása és elkapása közt futhat le kód. Fontos szabály, hogy a kivétel dobása és elkapása közt nem szabad újabb kivételt dobni, mert az már kezelhetetlen.

III. rész

Második felvonás

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.

11. fejezet

Helló, Arroway!

11.1. A BPP algoritmus Java megvalósítása

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

11.2. Java osztályok a Pi-ben

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

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

IV. rész

Irodalomjegyzék

11.3. Általános

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

11.4. C

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

11.5. C++

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

11.6. 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.