Magas szintű programozási nyelvek II.

Labor jegyzőkönyv

Ed. PROG2, DEBRECEN,
2020. szeptember 8, v. 0.0.1

COLLABORATORS

	TITLE : Magas szintű programozási nyelvek II.	

ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY	Dude, Dude	2020. szeptember 8.	

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME
0.0.1	2020-09-08	Inited	rkeeves

Ajánlás

Tartalomjegyzék

Ábrák jegyzéke

Előszó

...

Használat

Használj Linuxot. Én múlt félévben Cygwinnel és Mingw-vel trükköztem. Meg lehet csinálni de kétszer annyi idő lesz.

Főleg Ubuntun és Debian-on könnyen felmegy apt-tal a dblatex. Amit ne felejts el, az a hungarian language pack. texlive-lang-hungarian vagy texlive-lang-european.

I. rész

Bevezetés

1. fejezet

Labor jegyzőkönyv

1.1. Mi a cél?

A laborjegyzőkönyvnek a saját munka dokumentálása a célja.

II. rész

Tematikus feladatok

Bátf41 Haxor Stream
Erősen javaslom ezen jegyzőkönyv olvasása helyett Bátfai Tanár Úr csatornáját. Perma YT archive.

2. fejezet

Helló, Berners-Lee!

2.1. C++ Java összehasonlítás

Ebből a két könyvből pár oldalas esszé jellegű kidolgozást kérek, Java és C++ összehasonlítás mentén, pl. kb.: kifejezés fogalom ua., Javában minden objektum referencia, mindig dinamikus a kötés, minden függvény virtuális, klónozásstb.

Tanulságok, tapasztalatok, magyarázat...

2.2. Python Java Mobil

Itt a kijelölt oldalakból (35-51) egy 1 oldalas élmény-olvasónaplóra gondoltam.

Tanulságok, tapasztalatok, magyarázat...

3. fejezet

Helló, Arroway!

3.1. OO szemlélet

A módosított polártranszformációs normális generátor beprogramozása Java nyelven. Mutassunk rá, hogy a mi természetes saját megoldásunk (az algoritmus egyszerre két normálist állít elő, kell egy példánytag, amely a nem visszaadottat tárolja és egy logikai tag, hogy van-e tárolt vagy futtatni kell az algot.) és az OpenJDK, Oracle JDK-ban a Sun által adott OO szervezés ua.!Lásd még fóliák!Ismétlés: (16-22 fólia) Ugyanezt írjuk meg C++ nyelven is! (lásd még UDPROG repó: source/labor/polargen)

A Java implementáció a következő:

```
1  import java.lang.Math;
2
3  public class PolarGen
4  {
5    boolean store_is_empty;
6
7    double stored;
8
9    public PolarGen()
10   {
11     store_is_empty = true;
12   }
13
14   public double next()
15   {
16     if(store_is_empty){
17       double u1, u2, v1, v2, w;
18       do{
19         u1 = Math.random();
20         u2 = Math.random();
21         v1 = 2 * u1 - 1;
22         v2 = 2 * u2 - 1;
23         w = v1 * v1 + v2 * v2;
24       }while(w > 1);
25       double r = Math.sqrt((-2 * Math.log(w)) /w);
```

```
26        stored = r * v2;
27        store_is_empty = !store_is_empty;
28        return r * v1;
29      }else{
30        store_is_empty = !store_is_empty;
31        return stored;
32      }
33    }
34
35
36    public static void main(String[] args){
37      PolarGen g = new PolarGen();
38      for(int i = 0; i < 10; ++i){
39        System.out.println(g.next());
40      }
41    }
42  }
```

Látható hogy egy fordítási egységen belül definiáltunk egy PolarGen osztályt és implementáltuk is.

Ezen pici fájl miatt nem érdemes elindítani az IDE-t. Alább egy fordítás és futtatás OpenJDK 11.0-val, mivel az Oracle volt olyan kedves hogy ... na ebbe most ne menjünk bele-

A Cpp esetén egy header-ben definiáltuk az osztályt.

```
1  #ifndef POLARGEN_H
2  #define POLARGEN_H
3
4  namespace prog2{
5    class PolarGen
6    {
7
8    public:
9      PolarGen();
10
11     ~PolarGen() = default;
12
13     double next();
14
15   private:
16
17     bool store_empty;
18
19     double stored;
20
21   };
22 } /* prog2 */
23 #endif /* POLARGEN_H */
```

Include guard-ra azért van szükségünk, mert több helyen is fogjuk használni a header-t és nem akarjuk hogy újradefiniálási hibába ütközzünk. Ezt a preprocessornak kiadott utasításokkal érjük el. Ez röviden

annyit tesz, hogyha még nem létezik a POLARGEN_H def akkor bejutunk az ifdef-be és kiértékelésre kerül a header tartalma (plusz definiáljuk POLARGEN_H, azaz effektíve egy "beolvastuk-e már egyszer?" flag-ként működik). Ha a POLARGEN_H már definiálva volt, akkor kiesünk ifdefből és végeztünk.

Eltértem Tanár Úr példájától, hisz nem láttam értelmét a header-be includeolni a rand-hoz szükséges dolgokat. Ezek elegendőek ha bekerülnek a cpp-be. Ehhez csak annyit kell módosítanunk hogy ctor implementációt a cpp-ben végezzük el.

```
#include "polargen.h"

#include <cstdlib>
#include <cmath>
#include <ctime>

namespace prog2{

    PolarGen::PolarGen() : store_empty(true), stored(0.0)
    {
      std::srand( std::time(NULL) );
    }


    double PolarGen::next()
    {
      if(store_empty){
        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);
        stored = r * v2;
        store_empty = !store_empty;
        return r * v1;
      }else{
        store_empty = !store_empty;
        return stored;
      }
    }
} /* prog2 */

```

Egyébként jó szokás saját namespace-ben dolgozni, emiatt vezettem be ezen fájlokba a prog2 namespace-t.

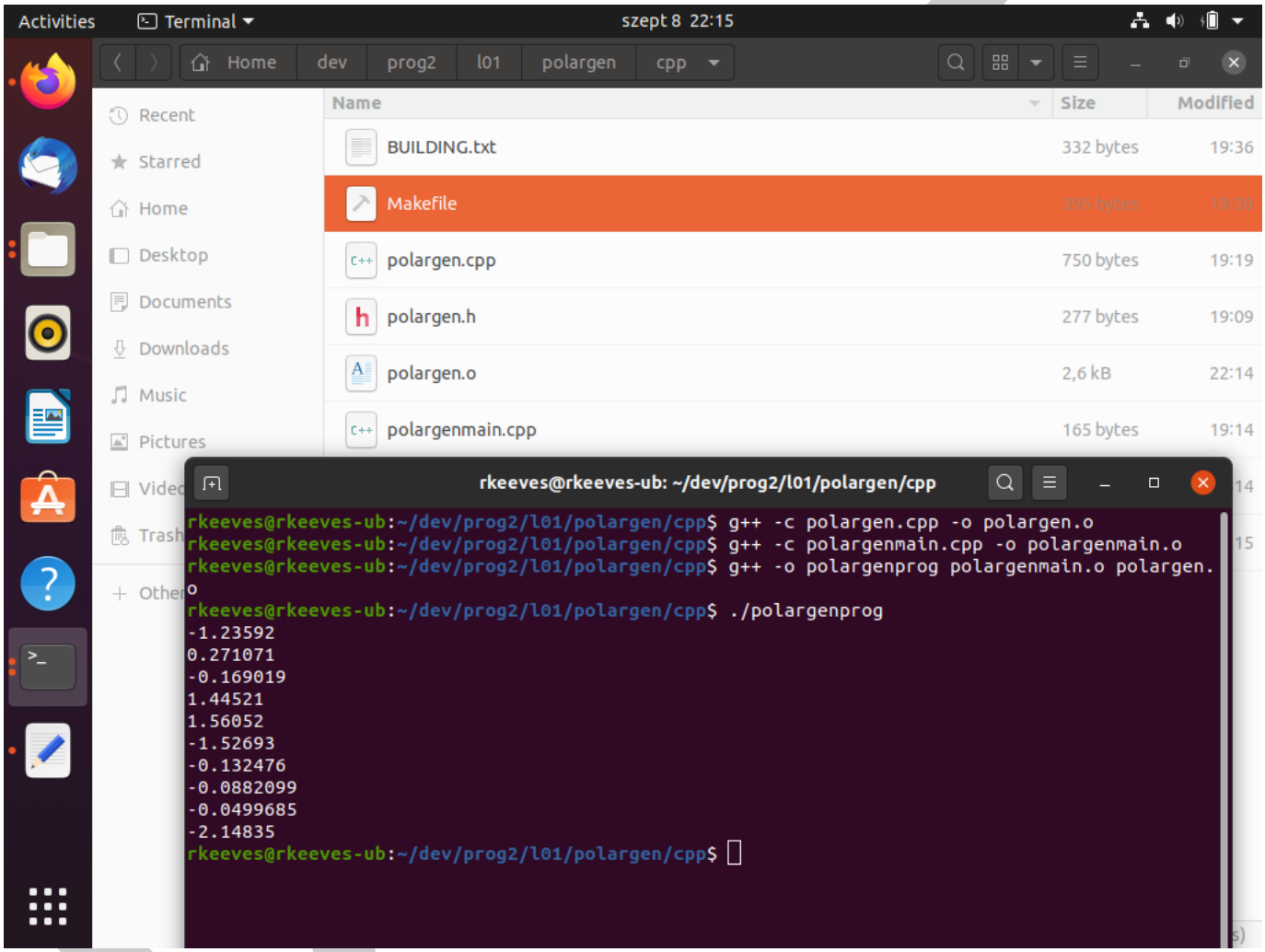
Ha mindez meg volt akkor már csak a "main" van hátra.

```
#include <iostream>
#include "polargen.h"

```

```
4  int main(int argc, char** argv){
5    prog2::PolarGen g;
6    for(int i = 0; i < 10; ++i)
7      std::cout<<g.next()<<std::endl;
8  }
```

A fordítás során ugye nem fordíthatjuk először a main-t. Előtte szükségünk van arra az objektumra, mely a polargen fordításából jön. Alább látható ez a folyamat.



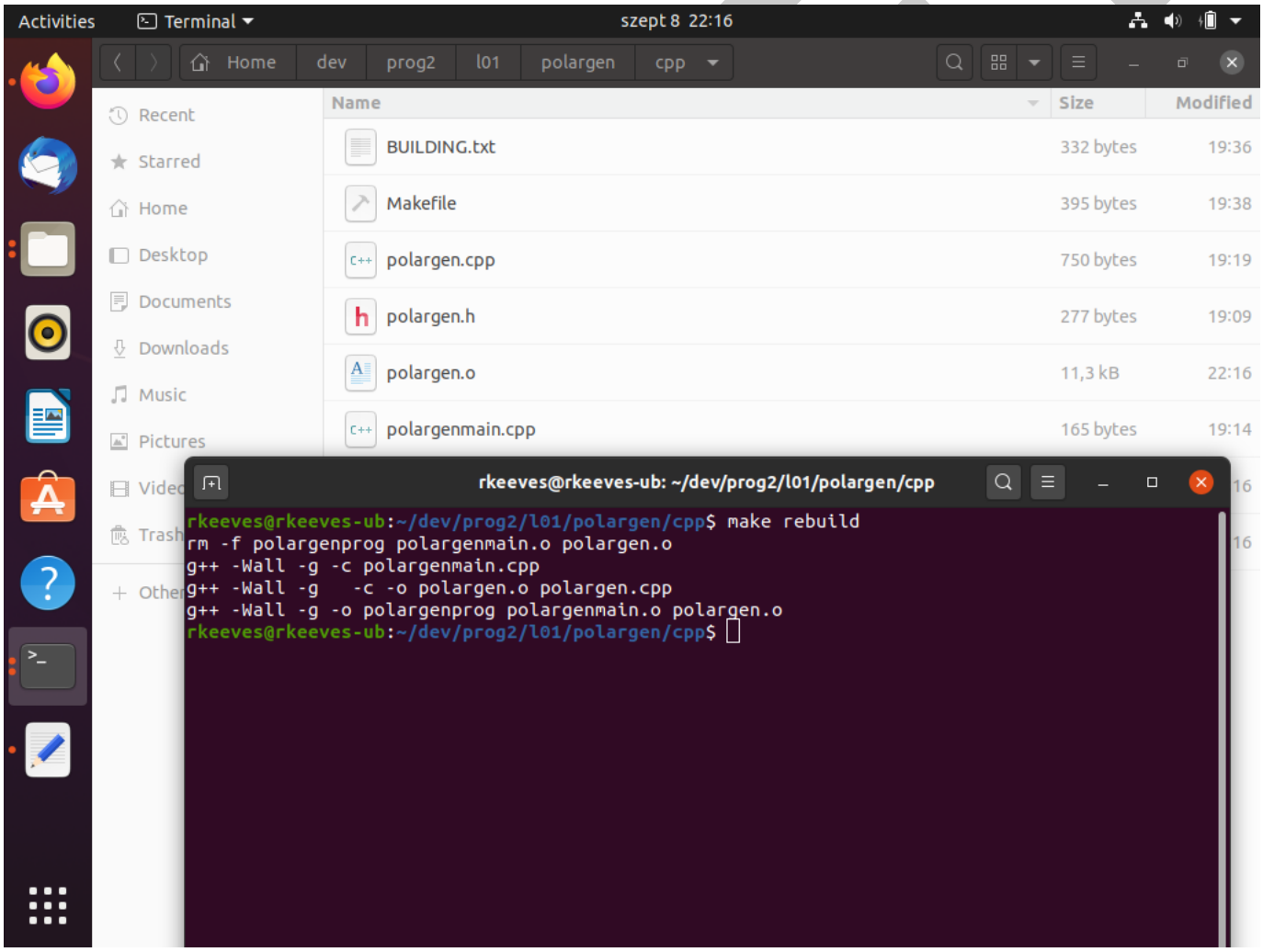
3.1. ábra. Manual Build

Persze egy olyan projektnél ahol 50+ file-unk van, ott érdemes lehet ezt automatizálni. Erre szolgálnak a makefile-ok. Alább egy primitív egyszerűségű példa.

```
1  # This is a really simplistic makefile
2  # For real projects use CMake, or Premake5
3
4  CXX = g++
5  CXXFLAGS = -Wall -g
6
```

```
7  rebuild: clean build
8
9  build: polargenmain.o polargen.o
10   $(CXX) $(CXXFLAGS) -o polargenprog polargenmain.o polargen.o
11
12 polargenmain.o: polargenmain.cpp polargen.h
13   $(CXX) $(CXXFLAGS) -c polargenmain.cpp
14
15 polargen.o: polargen.h
16
17 clean:
18   rm -f polargenprog polargenmain.o polargen.o
```

Alább egy példa a build-re make-el.



3.2. ábra. Make Build

A make feletti szintet a CMake képviseli, ezt nagyon széles körben használják és pont olyan borzalmas is mint minden amit sok ember szeret. Másik megoldás a premake ami nagyon jó annak aki szereti az átlátható

dolgokat, a Lua-t. Apró hátulütője, hogy jó esetben kinevetik miatta az embert rossz esetben pedig ki is közösítik.

3.2. Homokozó

Írjuk át az első védési programot (LZW binfa) C++ nyelvről Java nyelvre, ugyanúgy működjön!Mutassunk rá, hogy gyakorlatilag a pointereket és referenciákat kell kiirtani és minden máris működik (erre utal a feladat neve, hogy Java-ban minden referencia, nincs választás, hogy mondjuk egy attribútum pointer, referencia vagy tagként tartalmazott legyen).Miután már áttettük Java nyelvre, tegyük be egy Java Servletbe és a böngészőből GET-es kéréssel (például a böngésző címsorából) kapja meg azt a mintát, amelynek kiszámolja az LZW binfáját!

Tanulságok, tapasztalatok, magyarázat...

3.3. Gagyi

Az ismert formális „while (x <= t && x >= t && t != x);” tesztkérdéstípusra adj a szokásosnál (miszerint x, t az egyik esetben az objektum által hordozott érték, a másikban meg az objektum referenciája) „mélyebb” választ, írj Java példaprogramot mely egyszer végtelen ciklus, más x, t értékekkel meg nem! Apéldát építsd a JDK Integer.java forrására4, hogy a 128-nál inkluzív objektum példányokat poolozza!

Tanulságok, tapasztalatok, magyarázat...

3.4. Yoda

Írjunk olyan Java programot, ami java.lang.NullPointerEx-el leáll, ha nem követjük a Yoda conditions-t!

Tanulságok, tapasztalatok, magyarázat...

3.5. Kódolás from scratch

Induljunk ki ebből a tudományos közleményből: és csak ezt tanulmányozva írjuk meg Java nyelven a BBP algoritmus megvalósítását! Ha megakadsz, de csak végső esetben: (mert ha csak lemásolod, akkor pont az a fejlesztői élmény marad ki, melyet szeretném, ha átélnél).

Tanulságok, tapasztalatok, magyarázat...

3.6. EPAM: Java Object metódusok

Mutasd be a Java Object metódusait és mutass rá mely metódusokat érdemes egy saját osztályunkban felüldefiniálni és miért. (Lásd még Object class forráskódja)

Tanulságok, tapasztalatok, magyarázat...

3.7. EPAM: Eljárásorientál vs Objektumorientált

Írj egy 1 oldalas értekező esszé szöveget, amiben összehasonlítod az eljárásorientált és az objektumorientált paradigmát, igyekezve kiemelni az objektumorientált paradigma előnyeit!

Tanulságok, tapasztalatok, magyarázat...

3.8. EPAM: Objektum példányosítás programozási mintákkal

Hozz példát mindegyik “creational design pattern”-re és mutasd be mikor érdemes használni őket!

Tanulságok, tapasztalatok, magyarázat...

III. rész

Irodalomjegyzék

3.9. Általános

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

[JAVA] Nyékyné Dr. Gaizler Judit et al., Judit, *Java 2 útikalauz programozóknak 5.0 I.-II.*, ELTE TTK Hallgatói Alapítvány , 2008.

[PYTHON] Forstner, Bertalan, Ekler, Péter, Ã©s Kelényi, Imre, *Bevezetés a mobil programozásba. Gyors prototípus-fejlesztés Python és Java nyelven*, Szak Kiadó Kft. , 2008.

3.10. My Little Ponys

[CHISOMORPH] Sorensen, Morten Heine B, *Lectures on the Curry-Howard Isomorphism*, Pdf , 1998.

[DENOTATIONALSEMANTICS] Allison, Lloyd, *A practical introduction to denotational semantics*, Cambridge University Press , 1986.

[NANDTOTETRIS] Nisan, Noam, *The Elements of Computing Systems*, Homepage of the book , 2005.

[PROGLANGS] Harper, Robert, *Practical Foundations for Programming Language*, Carnegie Mellon University , 2016.

Köszönet illeti Bátfai Norbert Tanár Urat, továbbá a tárgy gyakorlat tartóit, és az UDPROG-ot .