VILLÁM BEMUTATÓ

miről szól majd a tanfolyam

Motiváció:

Kell egy program ami felismer egy bizonyos hangmagasságot

+ megmutatni, hogy c++-ban is lehet gyorsan programot írni

(+ megmutatni, hogy milyen hibákat lehet c++-ban elkövetni)

Csináljunk projectet

Az elején Eclipse:

- Mert minden platformra van.
- Mert jó a kódkiegészítője
- Nagyon jó shortcutok

make:

- Flexibilis
- Megy parancssorból
- Speciálisabb dolgokra(pl. profilozós build) már kényelmesebb mint az eclipse

Szerkezet

bin/

x86, x64, arm, ...

debug, release

lib/

Bináris vagy forrás

src/

Közös és platform specifikus részek

src/main.cpp:

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

Makefile, hogy forduljon

```
SoundStuff: main.cpp
   g++ -o SoundStuff main.cpp
```

ESZKÖZÖK

IDE, Fordítók, Makefile, Verziókezelők

Eszközök

IDE

- Eclipse/CDT
- Xcode
- Visual Studio Express
- Qt Creator

Fordítók

- Clang
- Gcc
- VS VC++

Eszközök - folytatás

Makefile

- Makefile
- cmake
- qmake

Verziókezelő

- Git
- Perforce

Hangfelismerés?

Fourier transzformáció

hangminta -> hangmagasságokhoz tartozó erősség

FFT: 2^x a minta mérete, cserébe gyorsabb

Mi példánk, 48000 sample rate, 1024 méretű minta -> 46,875 Hz-es felbontás

A hang 440Hz -> 9,38667 -> a visszatérési érték 9. eleme

Keressünk libet!

google: fft cpp lib

http://www.librow.com/articles/article-10
4 fájl, egyszerű

Bemásolva:
lib/fft/

Oké, de hogy kapjuk a hangot?

stdin-ről

arecord: Hangkártya kimenete az stdoutra, szabvány wav formátumban.

Bármit lehet konvertálni .wav-ra és azzal tesztelni

Könnyű hozzáilleszteni a hangbemenet később

Rakjuk össze!

```
#include <iostream>

using namespace std;

typedef unsigned char sample_t;

int main() {
    sample_t sample;
    while (true) {
        cin >> sample;
        cout << sample << " ";
    }
    return 0;
}
```

Rakjuk össze!

```
#include <iostream>

using namespace std;

typedef unsigned char sample_t;

int main() {
    sample_t sample;
    while (true) {
        cin >> sample;
        cout << +sample << " ";
    }
    return 0;
}
```

Rakjuk össze!

```
#include <iostream>

using namespace std;

typedef unsigned char sample_t;

int main() {
    sample_t sample;
    while (!cin.eof()) {
        cin >> sample;
        cout << +sample << " ";
    }
    return 0;
}
```

Közös deklarációk külön headerbe

```
#ifndef TYPES_H_
#define TYPES_H_

typedef unsigned char sample_t;

#endif /* TYPES_H_ */
```

pragma once

```
#pragma once

typedef unsigned char sample_t;
```

Külön class a hangmintának!

```
#define sampleLength  1024

class SoundSample {
   sample_t sample[sampleLength];

   SoundSample& operator<<(sample_t sample);
   operator char*();
};
```

Külön class a hangmintának!

```
class SoundSample {
   static const int sampleLength;
   sample_t sample[sampleLength];

   SoundSample& operator<<(sample_t sample);
   operator char*();
};
```

Külön class a hangmintának!

```
class SoundSample {
public:
   static const int sampleLength;

   SoundSample& operator<<(sample_t sample);
   operator char*();
private:
   std::vector<sample_t> samples;
};
```

Külön class a hangmintának!

```
class SoundSample {
public:
   static const int sampleLength;

   void append(sample_t sample);
   operator char*();
private:
   std::vector<sample_t> samples;
};
```

Külön class a hangmintának!

```
class SoundSample {
public:
   static const int sampleLength;

   void append(sample_t sample);
   std::string toString();
private:
   std::vector<sample_t> samples;
};
```

Namespace

```
namespace soundstuff {

class SoundSample {
public:
   static const int sampleLength;
   void append(sample_t sample);
   std::string toString();
private:
   std::vector<sample_t> samples;
};

}
```

Implementáció

```
#include "SoundSample.h"

using namespace std;
using namespace soundstuff;

const int SoundSample::sampleLength = 1024;

void SoundSample::append(sample_t sample) {
   samples.push_back(sample);
}
```

Etessük

```
SoundSample samples;
for (int i = 0; i < SoundSample::
sampleLength; ++i) {
   samples.append(sample);
}

if (samples.analyze()) {
   cout << "Signal" << endl;
}
```

Analizálás

- A bemenetet át kell alakítani, hogy jó legyen a libnek (complex*)

```
bool CFFT::Forward(complex *const
Data, const unsigned int N)
```

- Meg kell találni a legerősebb frekvenciát

Használjuk a libet!

```
#include "../lib/fft/complex.h"
#include "../lib/fft/fft.h"
```

Használjuk a libet!

```
#include <fft/complex.h>
#include <fft/fft.h>
```

Átalakítás

```
bool SoundSample::analyze() {

   vector<complex> complexVector;
   for (vector<sample_t>::iterator it =
      samples.begin();
      it != samples.end();
      ++it) {
      ccomplexVector.push_back(complex(*it));
   }

   samples.clear();
   complex* a = &complexVector[0];
```

std::transform

```
bool SoundSample::analyze() {

   vector<complex> complexVector;

   complexVector.resize(samples.size());
   transform(samples.begin(), samples.end(),
complexVector.begin(), toComplex);
   samples.clear();
   complex* a = &complexVector[0];
```

Maga a hívás:

```
bool ok = CFFT::Forward(
      temp,
      complexVector.size());
   if (!ok) {
      // Ilyenkor mi van?
      // Maradjunk csendben? return false;
      // Lépjünk ki? exit(1);
   }
```

Exception dobás

```
bool ok = CFFT::Forward(
      temp,
      complexVector.size());
   if (!ok) {
      throw exception();
   }
```

LIBEK

STD, QT, Saját

LIBEK - C++ standard

- Tárolók
  - std::vector
  - std::list
  - std::map
- Iterátorok
- Algoritmusok
  - std::transform
  - std::find
  - std::sort
- String
- Stream

LIBEK - QT

- Korlátozott lehetőségek az std-ben
- Hasonló adatszerkezetek
- Grafikus felhasználói felület
- Hálózatkezelés
- Szálkezelés
- Saját előfordító
  - signal/slot
  - UI

LIBEK - Saját

- Hogyan készíthetünk?
- Dinamikus / Statikus
- Miért érdemes használni?

Vissza a transform-hoz

```
complex(double val): m_re(val), m_im(0.) {}
```

Így kell előállítani

```
complex toComplex(sample_t s) {
   return complex(s);
}
```

De ezt is megeszi

```
complex toComplex(sample_t s) {
   return s;
   // Két implicit konverzió is történik!
}
```

explicit konstruktor!

```
explicit complex(double val): m_re(val),
m_im(0.) {}
```

C++11 auto

```
for (auto it = samples.begin();
 it != samples.end();
 ++it) {
...
}
```

Sokkal rövidebb!

C++11 iteráció

```
for (auto it : samples) {

}
```

Még sokkal rövidebb

Lambda függvények

Az átalakítós rész:

```
transform(
        samples.begin(),
        samples.end(),
        complexVector.begin(),
        [](sample_t t){return complex(t);}
        );
```

Melyik frekvencia a legerősebb?

```
int counter = 0;
int strongest;
double strongestValue = 0.0;
for (auto c : complexVector) {
   if (strongestValue < c.norm()) {
      strongestValue = c.norm();
      strongest = counter;
   }
   ++counter;
}
return strongest == 9;
```

Melyik frekvencia a legerősebb?

```
int counter = 0;
int strongest;
double strongestValue = 0.0;
for (auto c : complexVector) {
   if (strongestValue < c.norm()) {
      strongestValue = c.norm();
      strongest = counter;
   }
   ++counter;
}
return strongest == 9;
```

Újabb randa hibalehetőség

```
int strongest;
```

Hiába kap mindenképpen értéket most, lehet hogy később lesz egy olyan verzió, amikor inicializálatlan marad.

Ha nem számít a sebesség, sosem hagyunk inicializálatlan változót

C++11

C++11

- Sok hasznos újdonság, két kategória
  - új nyelvi elmek
  - std kiegészítés
- Produktivitás
  - Gyorsabban és pontosabban érjük el célunkat
  - Kapunk védőfelszerelést is, nem csak fegyvert
- További újdonságok várható
  - Teljesen új szabvány "1 év" múlva
  - Részeredmények időközben (pl.: hálózat)
- Tesztüzem a cégnél

C++11 - új nyelvi elemek, részlet

- λ függvények
- { }
- static_assert
- nullptr
- konstruktor delegáció
- rvalue reference
- raw string
- auto
- explicit
- enum class
- override
- final

C++11 - standard kiegészítés, részlet

- std::move
- szálkezelés
- reguláris kifejezések
- std::tuple
- std::initializer_list
- std::unique_ptr / std::shared_ptr

Magic Makefile

```
BINDIR=bin
SRCDIR=src
LIBDIR=lib

CXXFLAGS = -std=c++11 -O3 -I$(LIBDIR)

OBJS=$(SRCDIR)/main.o $(SRCDIR)/SoundSample.o $(LIBDIR)/fft/fft.o $(LIBDIR)
/fft/complex.o
HEADERS=SoundSample.h type.h $(LIBDIR)/fft/fft.h $(LIBDIR)/fft/complex.h

$(SRCDIR)/%.o: %.cpp $(HEADERS)
    $(CXX) -c $(CXXFLAGS) -o $@ &<

$(BINDIR)/SoundStuff: $(OBJS)
    $(CXX) -o $@ $^

clean:
    rm $(OBJS) $(BINDIR)/SoundStuff
```

Próbáljuk ki:

```
arecord -r 48000 | bin/SoundStuff
```

Első futtatásra mindig minden hibás. Jó lenne, ha debuggolás közben nem kéne mindig a mikrofont használni.

Megy szabvány .wav fájlokkal is!

```
bin/SoundStuff < test.wav
```

Következő motiváció

Morse kód olvasás!

A teljes kód fent van a githubon, itt csak részek lesznek.

szöveg <-> belső reprezentáció <-> jelzés/csend

Hogy működjön?

```
/// Expects raw string
 MorseText(const std::string& rawString);

 /// Interprets high/low values
 MorseText(
     int signalUnitLength,
     const std::vector<bool>&
  signalVector);
```

Hogy működjön?

```
void toSignals(
    std::vector<bool>& appendTo);
```

VS.

```
std::vector<bool> toSignals();
```

Ami innen még érdekes

C++11-es inicializáló listák

```
map<char, vector<MorseText::Signal> >
MorseText::signalMap = {
    {'a', {SHORT, LONG} },
    {'b', {LONG, SHORT, SHORT, SHORT} },
    {'c', {LONG, SHORT, LONG, SHORT} },
```

Teszteljünk

```
#include <gtest/gtest.h>
#include "morsetext.h"

TEST(MorseTextTest, DefaultConstructor)
{
    EXPECT_EQ("", MorseText().toString());
}
```

Egyszerű esettel érdemes kezdeni

```
[==========] Running 1 test from 1 test case.
[----------] Global test environment set-up.
[----------] 1 test from MorseTextTest
[ RUN      ] MorseTextTest.DefaultConstructor
[       OK ] MorseTextTest.DefaultConstructor (0 ms)
[----------] 1 test from MorseTextTest (0 ms total)

[----------] Global test environment tear-down
[==========] 1 test from 1 test case ran. (0 ms total)
[  PASSED  ] 1 test.
```

```
TEST(MorseTextTest, Reversible)
{
    MorseText mt("alma");

    std::vector<bool> sv;
    mt.toSignals(sv);

    // Nem egyezik meg a bemenet és a kimenet - Szándékos?
    std::vector<int> iv;
    for(auto i: sv) { iv.push_back(i); } // !!HACK!!

    EXPECT_EQ("alma", MorseText(1, iv).toString());
}
```

```
[ RUN      ] MorseTextTest.Reversible
src/test/morse.cpp:29: Failure
Value of: MorseText(1, iv).toString()
  Actual: "a l m a "
Expected: "alma"
[  FAILED  ] MorseTextTest.Reversible (1 ms)

[==========] 3 tests from 1 test case ran. (1 ms total)
[  PASSED  ] 2 tests.
[  FAILED  ] 1 test, listed below:
[  FAILED  ] MorseTextTest.Reversible
```

UNIT TESZT

Unit teszt

- Mit értünk egy unit teszt alatt?
- Hogyan készíthetünk?
- Miért érdemes?
- Hogyan érdemes?

Unit teszt - google eszközök

gtest

- Teszthez szükséges állapot statikus kialakítása
- Elvárt eredmény megfogalmazása
- (Ál)biztonság

gmock

- Teszthez szükséges állapot dinamikus kialakítása
- Ritka hibák előidézése
- Erőforrásigény csökkentése

Unit teszt - TDD

- Test Driven Development
- Egy csoda!
- Végre lehet csúnya kódot írni!
- Időigényes
- Segíti a refactort
  - piros teszt
  - kódolás
  - zöld teszt
  - refactor
- Fél éve használjuk - kezd jó lenni
- Szerintem: sokat javít a szerkezeten

PROGRAM-TERVEZÉSI MINTÁK

Design patterns

Tervezési minták

- Nagyon egyszerű dolgok
- Már magától is mindenki így dolgozik
  - Gyártófüggvény / Factory Method
- Három nagyobb csoport
  - Létrehozási
  - Szerkezeti
  - Viselkedési
- Két megvalósítási mód
  - Osztály
  - Objektum

Memória mérés!

valgrind a barátunk

```
valgrind bin/SoundStuff < sample.wav
```

Memória mérés!

==29425== HEAP SUMMARY:

==29425== in use at exit: 0 bytes in 0 blocks

==29425== total heap usage: 554,997 allocs, 554,997 frees, 58,765,958 bytes allocated

==29425==

==29425== All heap blocks were freed -- no leaks are possible

==29425==

==29425== For counts of detected and suppressed errors, rerun with: -v

==29425== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 2 from 2)

Mit jelent?

Nem veszett el memória.

A programban nincs is dinamikus memória foglalás. Nem veszhet el memória.

shared_ptr, **unique_ptr**, **weak_ptr**: Ezekkel sokkal biztonságosabb a dinamikus memóriakezelés is

Profilozás

A program melyik része milyen gyorsan fut?

gprof

-g -gp fordítási opció a g++-nak -> gmon.out

A gprof csak értelmez

Profilozás

Ez jött ki: 1:14.78s alatt futott le

52.96	MorseText::reduceNoise
35.09	MorseText::MorseText
7.27	CFFT::Perform
1.38	CFFT::Forward

reduceNoise:

Csúnya is, lassú is:

```
auto it = signals.begin()+2;
while (it != signals.end()) {
   if (*(it - 2) == *(it)) {
      *(it - 1) = *it;
   }
   it++;
}
```

Optimizálva

```
int signalsSize = signals.size();
bool b0;
bool b1 = signals[0];
bool b2 = signals[1];
```

Optimizálva

```
for (int i = 2; i < signalsSize; ++i)
{
  b0=b1;
  b1=b2;
  b2=signals[i];
  if (b0 == b2) {
    signals[i-1] = b1 = b0;
  }
}
```

Új mérés

Futási idő: 1:05.50

43.98 MorseText::MorseText()
41.08 MorseText::reduceNoise()
9.11 CFFT::Perform()

10s különbség - 12%

A reduceNoise így is nagyon lassú

A vector<bool> egy spéci típus

Kis helyre van optimizálva ezért lassú

vector<int>-re cserélve a reduceNoise már meg sem jelenik profilozáskor

50s alatt fut le így

TOVÁBBI TÉMÁK

További témák

- Code smell
- Refactor
- Debug
- Crash dump elemzés
- Statikus analízis
- Profiling
- Continuous Integration
- ...

Arduino

A c++ szinte mindenen fut, ami digitális.

- 8 bites mikrovezérlő
- nincs operációs rendszer
- minimális memória
- nincs még c++11 STL, de c++11 van.

```
MorseText t("HelloWorld");
vector<bool> signals;
t.toSignals(signals);
```

```
for (vector<bool>::iterator it = signals.
begin(); it != signals.end(); ++it) {
    if (*it) {
      tone(TONE_PIN, TONE);
    } else {
      noTone(TONE_PIN);
    }
    delay(TONE_DURATION);
  }
```

Kérdések?