Titkosírás haladóknak I.

(Cipher - exercises for advanced users - Part I)

Pitlik Marcell, Pitlik László, Pitlik Mátyás, Pitlik László (jun) – MY-X team

Kivonat: A titkosírások története számtalan megoldást ismer és vélelmezhetően végtelen sok trükk alakítható ki szövegek a triviális olvashatóságának nehezítésére. A három részes cikksorozat első részében bemutatandó kódoló, majd a második részben prezentálandó dekódoló eljáráshoz az alapötletet a mobiltelefonok tárcsázási logikája adta, ahol egy gombon több betű is elhelyezkedik. A harmadik rész a kódoló program finomhangolásáról szól, vagyis a kezdő és a haladó programozók szemlélet-különbségei kerülnek példaértékűen bemutatásra. Egy rel. nehéz titkosírás esetén elvárható, hogy a szavak határát ne lehessen felismerni, ill. az is elvárható, hogy egyes betűket/jeleket ne csak egy jelsorozat helyettesítsen, ill. lehetőség szerint legyen a titkosírásban értelmetlen/zavaró jelből is valamennyi – minél inkább véletlenszerűnek tűnő módon, sőt egy-egy üzenet ne azonos jelsorozattá alakuljon a titkosítás során mindenkor lehetőség szerint. A mellékelt titkosító demo egy C++ állomány, melyet a repl.it oldalon lehet futtatni. A cikk tartalmaz demo-üzeneteket is a titkosítási folyamat input-output-viszonyainak demonstrálására.

Kulcsszavak: titkosírás tervezése, titkosírások nehézségi szintjei

Abstract: The history of the ciphering offers a lot of solutions and theoretically, quasi unlimited ideas should be existing for causing problems during the interpretation of messages. This article is one of series having three parts. The first part presents a cipher algorithm created by an author having just a little experience in programming. The part II. demonstrates the appropriate decipher algorithm where the basic idea was delivered through the old-style mobile phone’s keyboard offering more letters per button. The last (third) part shows the same functionality like in the first part – however this source code got realized by a professional author. Based on the comparing of the more and less trained thinking methodologies (of more and less optimized source codes), it is possible to demonstrate the development possibilities concerning abstraction/complexity of approximations. A rel. robust cipher ensures that the starting and/or ending position of words can not be identified. It is also a sign of higher complexity of ciphering, if one particular letter in the raw message will be substituted through more than one code. Besides, it is necessary to have meaningless/random parts in the ciphered messages. Parallel, it is a more complex level, when the same message can be described through different ciphers in an identical way. The presented demo is a C++ program which can be used through the online services of repl.it. The article also includes demo messages and their ciphered versions.

Keywords: planning of ciphering, levels of ciphering

# Bevezetés

Előzményként említésre méltó egy új évi jó kívánság 2013-ból: <https://miau.my-x.hu/miau2013_2014.png>, melyre megfejtés azóta sem érkezett – bár vélhetően csak azért nem, mert nem feltétlenül áll az érdeklődés középpontjában egy olyan kép, mely kódtartalmára szinte semmi nem utal egyébként (<https://miau.my-x.hu/miau2009/index.php3?x=news>).

Friss impulzusként említendő, hogy 2018-ban a KJE sikeresen akkreditálta a BPROF képzését (üzemmérnök informatikus területen), mely képzés a tananyagfejlesztés kapcsán jelentős fejlesztéseket vár el. Ennek a folyamatnak a szerves része ez a cikk és ennek folytatása. Hiszen a BA és a BSC képzések mellett egyedülálló BPROF képzés gyakorlatiassága megkérdőjelezhetetlen kell, hogy legyen. Az akkreditáció során két specializáció került kialakításra: az agrárinformatika és az IT-biztonság, mely utóbbi értelemszerűen kötődik a titkosírásokhoz is – még esetlegesen, ha csak gamifikációs effektként és/vagy szemléletformáló erőtérként. A kódírás és a kódtörés a leendő programozók/programtervezők/rendszerüzemeltetők számára vélhetően kellően érdekes terület ahhoz, hogy egy hagyományos példánál nagyobb motiváltságot lehessen ezen keresztül kiváltani az érintett Hallgatóságból.

További részletek: <https://miau.my-x.hu/miau/253/titkosiras/>

# Üzenetek

Természetesen, aki vérbeli kódfejtő/kódíró, annak az alábbi üzeneteket előbb illik saját hatáskörben megfejtenie, mielőtt tovább olvassa a cikket:

Üzenet1:

00000000000100072516061170002116131040002102000702150000912190002012160140065000802120200060051411723018181401070030000711506101750280061190618800082000412318408130500009313000811209018000800510301420151202200050170040053111780518000460311411650320616060008140604900040319180005000703600313171036000410803790081305051300033010700007160310400300315103156000211801093630188030500020792015505106006000331500051261040151611070000902318418031511506700088020051205140004161310411411170004060310417310071181140003120003300650401608111111111

Üzenet2:

00000000000140060061130000908121600081840079160160180004603114000480161022000091189363118000513114010908118803170003131351603106512107020600211500050202201216000540604900009016136301800061506111911236000801200020431123130711353018000096301600052911610616001740852000201841179061060180000700219108000601701301603150031060000710171401537815180000911511650003061310410650321614000610406140009000361040161130360008311419307000070516030165121070004811301413600061677910303108100943018116000410506131160050028000423000091062070008116016017118000811417000481341028000305310711601220000760600510310179118143008106000250137900041126014115160700062000811301305131165140069020017800031141017000316030045211840801616

A megoldásokat természetesen a cikk maga is tartalmazza.

# A kódoló eljárás

Az alábbi (tudatosan nem a végső készültségi állapotot jelentő) kód tovább-gondolásra hívja fel az Olvasókat és maguk az üzenetek is a fejlesztőmunka egy adott állapotában felmerült gondolatokat osztják meg az Érdeklődőkkel:

// Example program

#include <iostream>

#include <string>

#include <map>

#include <sstream>

#include <bitset>

#include <stdlib.h>

#include <time.h>

using namespace std;

struct kod2 {

int mod;

int kong;

int tarcsa;

kod2(){};

kod2(int m, int k, int t) {

mod = m;

kong = k;

tarcsa = t;

}

};

/\*map<char,kod2> kod;

struct kodolo {

//int forgatas;

string uzenet;

kodolo(string u) {

uzenet = u;

}

string kodol() {

ide ki kene meg valahogy szedni

na meg a tobbi alegyseget is

}

};\*/

int main()

{

srand (time(NULL));

string uzenet;

string initdatum;

cout << "Mi az uzenet?"<< endl;

getline (cin, uzenet);

cout << "Esedekesseg datuma? (MM/DD)"<< endl;

getline (cin, initdatum);

stringstream sho, snap;

int ho,nap,nap1,nap2;

sho << initdatum[0] << initdatum[1];

sho >> ho;

snap << initdatum[3] << initdatum[4];

snap >> nap;

nap1=nap/10;

nap2=nap%10;

string bho = bitset<4>(ho).to\_string();

string bnap1 = bitset<2>(nap1).to\_string();

string bnap2 = bitset<4>(nap2).to\_string();

map<char,kod2> kod;

kod['a']= {3,0,2};

kod['b']= {3,1,2};

kod['c']= {3,2,2};

kod['d']= {3,0,3};

kod['e']= {3,1,3};

kod['f']= {3,2,3};

kod['g']= {3,0,4};

kod['h']= {3,1,4};

kod['i']= {3,2,4};

kod['j']= {3,0,5};

kod['k']= {3,1,5};

kod['l']= {3,2,5};

kod['m']= {3,0,6};

kod['n']= {3,1,6};

kod['o']= {3,2,6};

kod['p']= {4,0,7};

kod['q']= {4,1,7};

kod['r']= {4,2,7};

kod['s']= {4,3,7};

kod['t']= {3,0,8};

kod['u']= {3,1,8};

kod['v']= {3,2,8};

kod['w']= {4,0,9};

kod['x']= {4,1,9};

kod['y']= {4,2,9};

kod['z']= {4,3,9};

cout << endl << "A feldolgozott uzenet:" << endl << '"' << uzenet << '"' << endl;

int forgatas;

stringstream ss;

for (int i=0; i<uzenet.length(); i++) {

char betu = uzenet[i];

if (betu != 32 && (betu<97 || betu>122)) {

cout << "ERROR: UNEXPECTED CHAR!" << endl << "Csak az angol abc kisbetui megengedettek!" << endl;

exit(1);

}

//cout << betu << endl;

if (isspace(betu)) {

int sprand = rand()%8 + 1;

map<int,string> spkod;

spkod[1]= "0002";

spkod[2]= "0003";

spkod[3]= "0004";

spkod[4]= "0005";

spkod[5]= "0006";

spkod[6]= "00007";

spkod[7]= "0008";

spkod[8]= "00009";

ss << spkod[sprand];

//cout << ss.str() << endl;

} else {

forgatas = 0;

for (int j=0; j<ss.str().length(); j++) {

if (ss.str()[j] == '0' || ss.str()[j] == '1') {

forgatas++;

}

}

//cout << forgatas % kod[betu].mod << endl;

while (forgatas % kod[betu].mod != kod[betu].kong) {

bool forrand = rand()%2;

ss << forrand;

forgatas++;

//cout << ss.str() << endl;

}

ss << kod[betu].tarcsa;

//cout << ss.str() << endl;

}

}

while (forgatas % 12 != 0) {

ss << "1";

forgatas++;

}

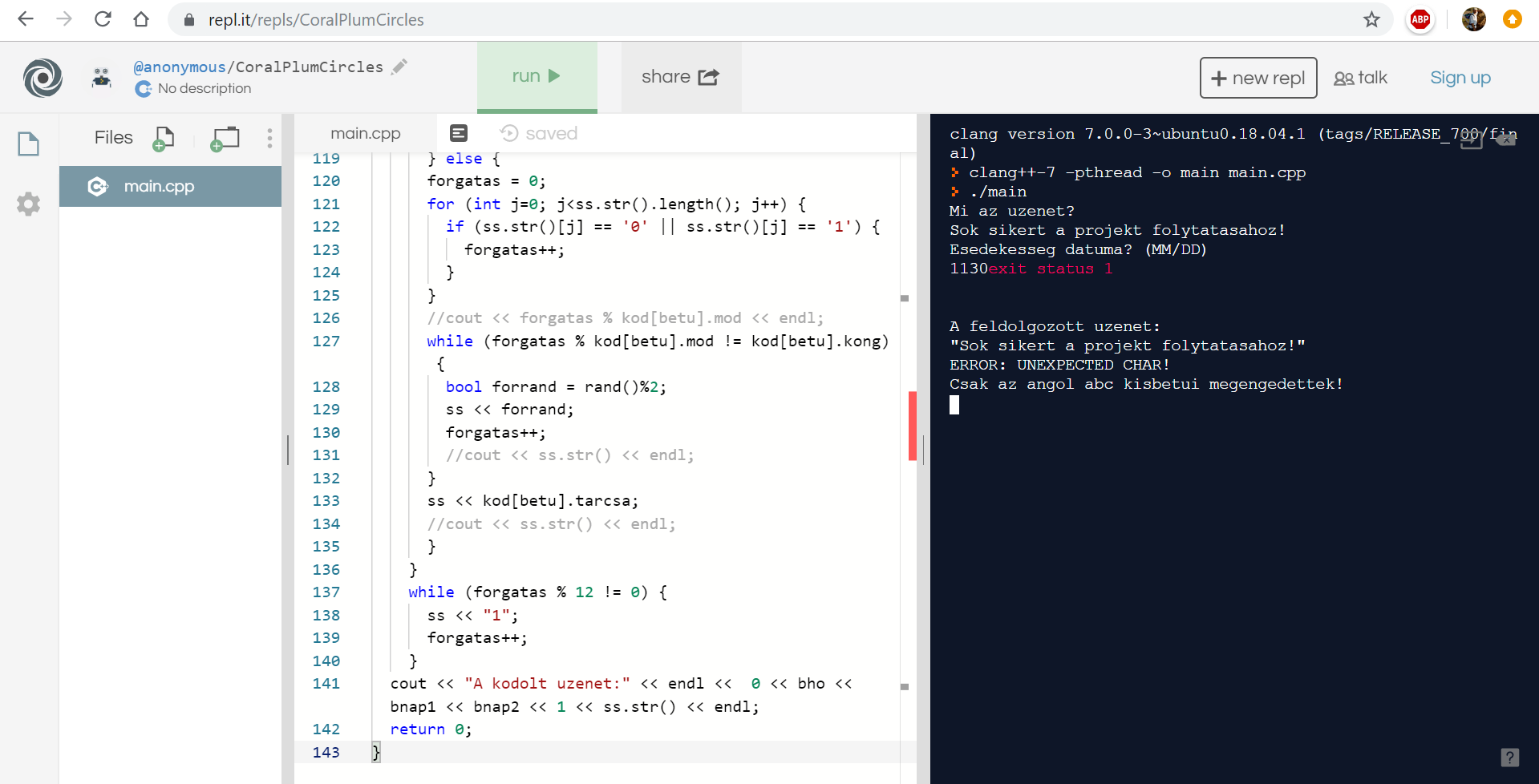
cout << "A kodolt uzenet:" << endl << 0 << bho << bnap1 << bnap2 << 1 << ss.str() << endl;

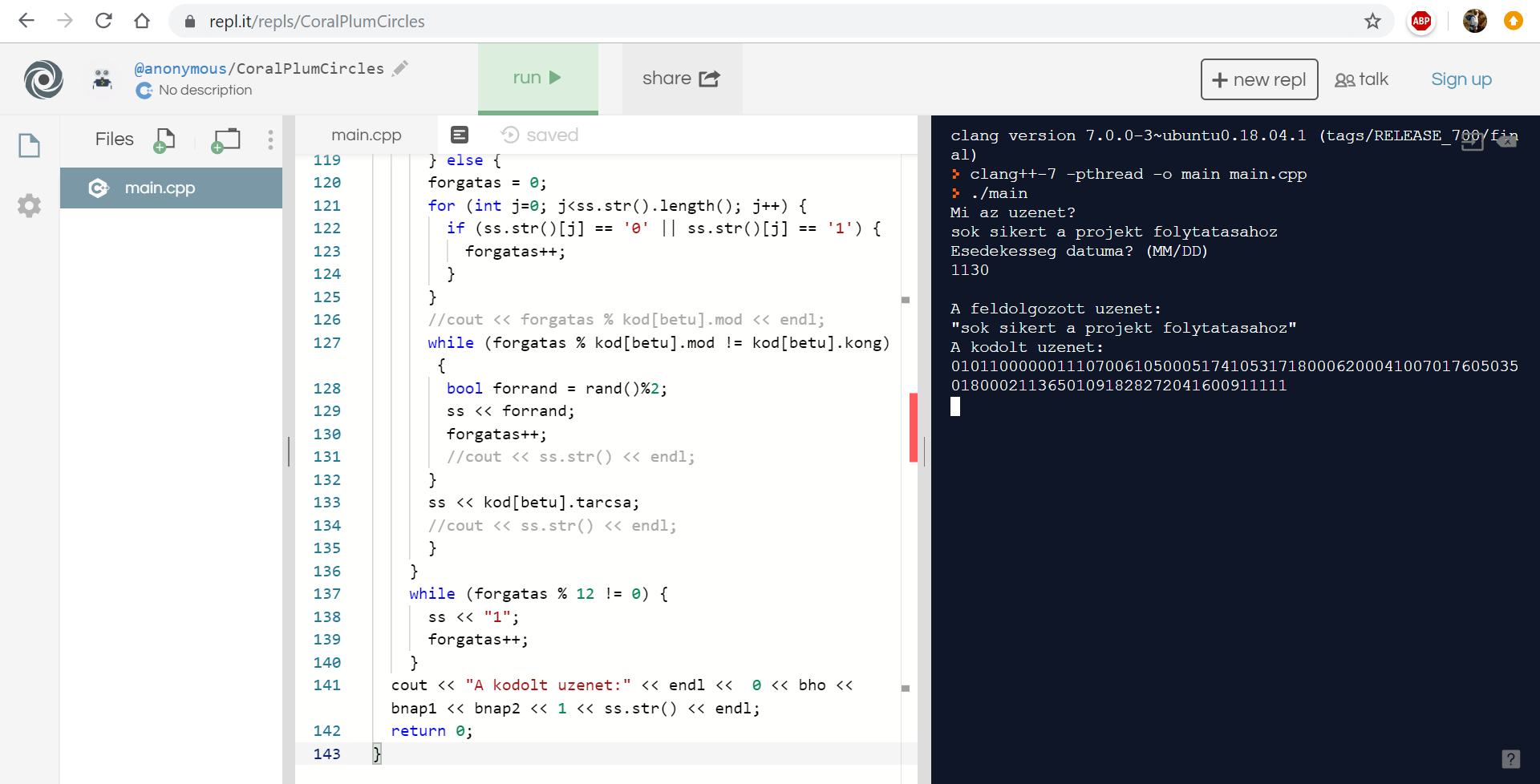
return 0;

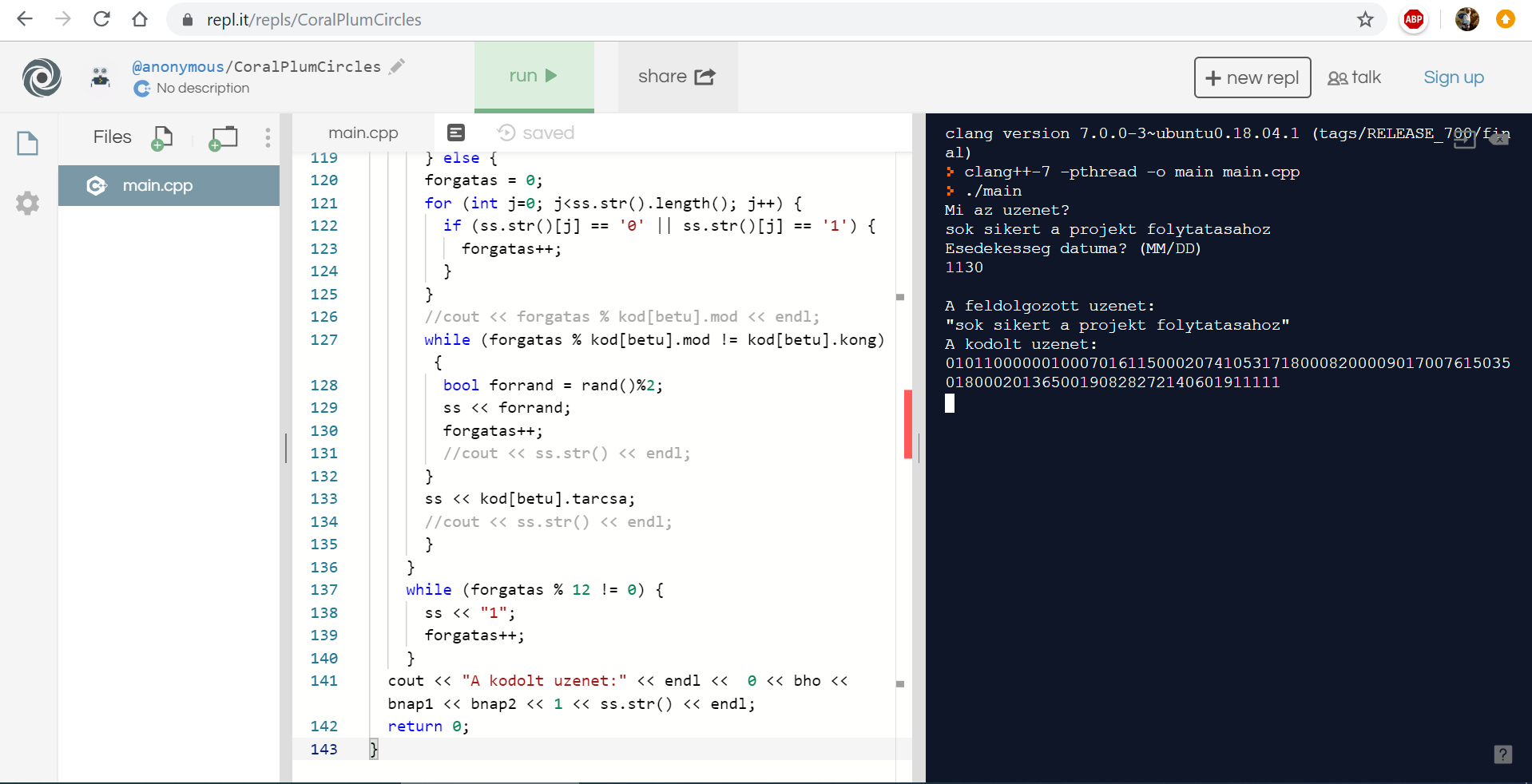
}

# Futtatási keretek

A repl.it oldalon a jobb felső sarokban látható „new repl” gomb mögött lehet választani programozási nyelvet (jelen esetben a C++-t). Majd a felkínált kódszerkesztési felületre a már kész (jelen esetben a titkosírást előállító) kód kerül bemásolásra. A „run” gomb (középen felül zöld háttérrel) teszi majd lehetővé magát a futtatást a megfelelő előkészítő lépések után. Amint fut a program, akkor a dialógus logikáját követve kell bemásolni a kódolandó üzenetet, mely alapvetően egyelőre az angol karakterkészletet engedi meg. Ugyanazon üzenet nem ugyanazon kóddá fordul minden esetben, vagyis egy üzenetnek több alternatív kódnézete is van:







# Konklúziók

Az oktatás gamifikálása alapvető kihívásként nehezedik az oktatási rendszerre. Látszat célok esetén a motiváció alábbhagy a Hallgatóságban. Valódi (piacképes/innovatív) célok esetén pedig a megfutamodás eseteivel állhat olykor/gyakran szemben az oktató. Jelen esetben gyakorló feladatként a kódolás/dekódolás érdekes játék – valódi innováció azonban akkor keletkezik az újabb és újabb kódolási logikákon túl, ha a már kódolt (ismeretlen logikájú) üzenet visszafejtésére alkalmas egyre inkább univerzális robot kerül kialakításra (vö. KNUTH: tudás az, ami forráskódba átírható – minden más emberi aktivitás művészet…)

Titkosírás haladóknak II.

(Cipher - exercises for advanced users - Part II)

Pitlik Marcell, Pitlik László, Pitlik Mátyás, Pitlik László (jun) – MY-X team

Kivonat: A KNUTH-i elv értelmében dekódolásról akkor beszélhetünk, ha a teljes folyamat programozott képességként kerül felkínálásra. Vagyis az, hogy emberi intuícióval (találgatással) adott üzenet megfejtettnek minősül, még nem felel meg annak az elvárásnak, hogy a dekódolás hibátlanul képes futni automatikusan. Jelen cikk egy három részes sorozat második eleme, ahol az első cikk a titkosítási eljárást már bemutatta. Itt és most a visszafejtő/dekódoló program kerül bemutatásra, míg a harmadik cikk a kódoló/titkosító eljárás finomhangolásának elvi és gyakorlati vonatkozásait mutatja be. A cikksorozat célja a gamifikált tanulás támogatása.

Kulcsszavak: titkosírás/kódfejtés tervezése, titkosírások nehézségi szintjei

Abstract: Based on the principle of KNUTH, it is just possible to speak about deciphering, if the whole process could be transferred into source code and therefore it can be used in an automatically way. It means the intuitive solution (guessing) is rather a kind of artistic performance as knowledge – independent from its success. This article is the second part of a series having three parts altogether. The first part already presented the ciphering algorithm. Here and now, it is possible to see the deciphering algorithm. The third part will demonstrate the same functionality of ciphering but instead of a solution created by a less trained programmer, the last part will present a more professional (optimized/efficient) source code. One of the goals of the article is the support the gamified learning.

Keywords: planning of de/ciphering, levels of ciphering

# Bevezetés

Ahhoz, hogy a titkosírások magasabb szintjén elvárt szempontok pl.

* szóhatárok felismerhetetlensége
* adott input karakter sokféle alternatív (eltérő hosszúságú) jelsorozattal való helyettesítése
* véletlenhatások bevonása
* adott üzenet alternatív megoldásokkal való lefedni tudása
* felesleges jelek (véletlenszerű) elhelyezése, stb.

algoritmikusan érvényesülni tudjanak, a titkosítás tervezésekor azonnal a visszafejtés mechanizmusait is tervezni kell.

# A dekódoló eljárás

Az alábbi eljárás a cikksorozat első részében bemutatott titkosítás visszafejtésének tapasztalatlan programozó által írt kódja:

#include <iostream>

#include <sstream>

#include <numeric>

#include <bitset>

#include <map>

using namespace std;

int main() {

string titkod;

int inits [12];

int forgatas = 0;

cout << "Mi a kodolt uzenet?" << endl;

getline (cin, titkod);

//cout << titkod << endl;

if (titkod[0] != '0' || titkod[11] != '1') {

goto ERR;

}

for (int i=0; i<12; i++) {

inits[i] = titkod[i] - '0';

if (inits[i] == 0 || inits[i] == 1) continue;

else goto ERR;

}

if (accumulate(inits, inits+12, 0) == 1) {

cout << endl << "Esedekesseg:\tISMERETLEN" << endl;

}

else if (accumulate(inits, inits+12, 0) < 10) {

stringstream ss;

string ho, nap1, nap2;

for (int i=1; i<5; i++) {

ss << inits[i];

}

ss >> ho;

if (bitset<4>(ho).to\_ulong() > 12) {

goto ERR;

}

stringstream().swap(ss);

ss << inits[5] << inits[6];

ss >> nap1;

stringstream().swap(ss);

for (int i=7; i<11; i++) {

ss << inits[i];

}

ss >> nap2;

if (bitset<4>(nap2).to\_ulong() > 9) {

goto ERR;

}

cout << endl << "Esedekesseg:\t" << bitset<4>(ho).to\_ulong() << "/" << bitset<2>(nap1).to\_ulong() << bitset<4>(nap2).to\_ulong() << endl;

}

else {

//cout << "teszt: lefutott" << endl;

ERR:cout << "ERROR: MSG HDR COMPROMISED!" << endl;

exit(1);

}

stringstream uzenet;

for (int i=12; i<titkod.length(); i++) {

char c = titkod[i];

if (c == '0' || c == '1') {

forgatas++;

}

else {

if (i>14 && titkod[i-3] == '0' && titkod[i-2] == '0' && titkod[i-1] == '0') {

if (c == '7' || c == '9') {

if (titkod[i-4] == '0' && i>15) {

uzenet << " ";

} else goto MEGIS\_BETU;

} else {

uzenet << " ";

}

} else {

MEGIS\_BETU:if (c == '2') {

map<int,char> tabla2;

tabla2[0]= 'a';

tabla2[1]= 'b';

tabla2[2]= 'c';

uzenet << tabla2[forgatas%3];

}

else if (c == '3') {

map<int,char> tabla3;

tabla3[0]= 'd';

tabla3[1]= 'e';

tabla3[2]= 'f';

uzenet << tabla3[forgatas%3];

}

else if (c == '4') {

map<int,char> tabla4;

tabla4[0]= 'g';

tabla4[1]= 'h';

tabla4[2]= 'i';

uzenet << tabla4[forgatas%3];

}

else if (c == '5') {

map<int,char> tabla5;

tabla5[0]= 'j';

tabla5[1]= 'k';

tabla5[2]= 'l';

uzenet << tabla5[forgatas%3];

}

else if (c == '6') {

map<int,char> tabla6;

tabla6[0]= 'm';

tabla6[1]= 'n';

tabla6[2]= 'o';

uzenet << tabla6[forgatas%3];

}

else if (c == '7') {

map<int,char> tabla7;

tabla7[0]= 'p';

tabla7[1]= 'q';

tabla7[2]= 'r';

tabla7[3]= 's';

uzenet << tabla7[forgatas%4];

}

else if (c == '8') {

map<int,char> tabla8;

tabla8[0]= 't';

tabla8[1]= 'u';

tabla8[2]= 'v';

uzenet << tabla8[forgatas%3];

}

else if (c == '9') {

map<int,char> tabla9;

tabla9[0]= 'w';

tabla9[1]= 'x';

tabla9[2]= 'y';

tabla9[3]= 'z';

uzenet << tabla9[forgatas%4];

} else {

uzenet << "\*err\*";

}}

}

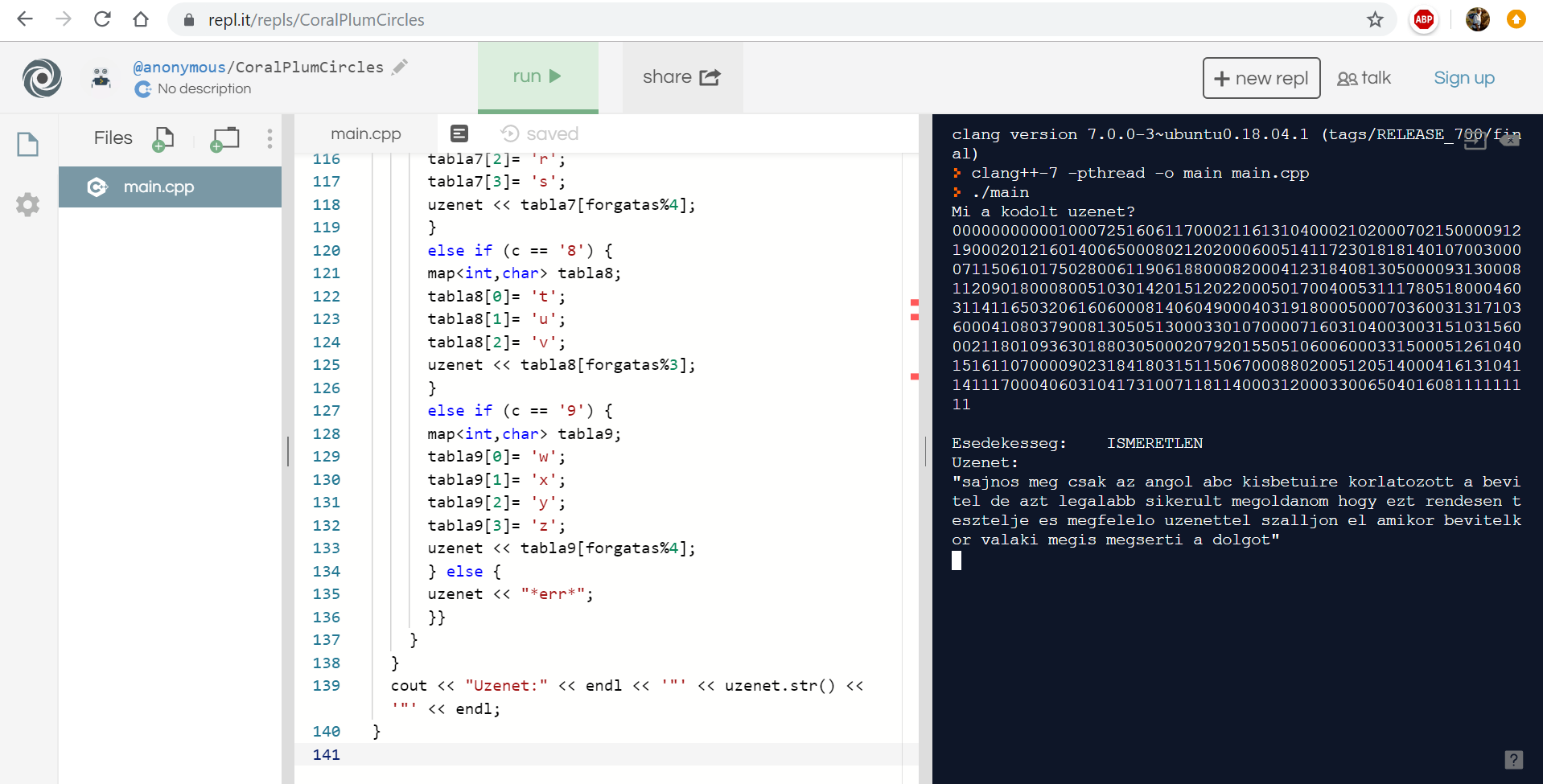
}

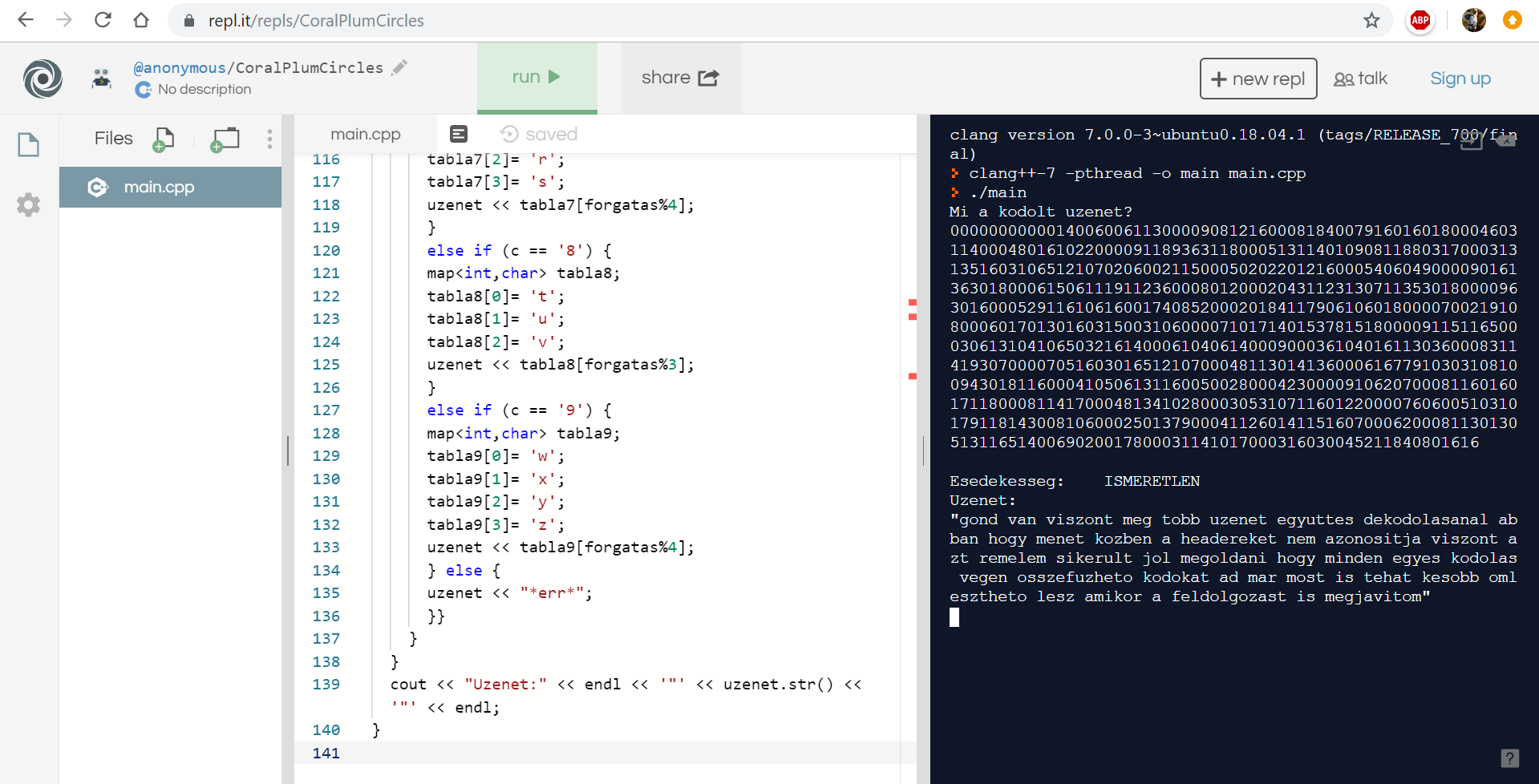
cout << "Uzenet:" << endl << '"' << uzenet.str() << '"' << endl;

}

# Üzenetek

A cikksorozat első részében jelzett programozói gondolatok titkosítás és visszafejtése tehát lehetséges:





# Konklúziók

A dekódolás és a titkosítás egyszerre, egymással párban fejlesztendő. Sajnos ez a gondolkodásmód csak azt garantálja, hogy a titkosítás és a visszafejtés mindvégig egymásnak megfelelő és kevésbé mozgatja meg az ismeretlen titkosírások visszafejtéséhez szükséges agyi területeket. Így a gamifikáció ilyen formája csak korlátozottan támogatja az innovativitást. Szerencsés esetben a valaki által kialakított titkosítások megértése és a visszafejtés algoritmikusságának garantálni tudása legalább új titkosítási koncepciók fejlesztését képes lehet katalizálni.

Titkosírás haladóknak III.

(Cipher - exercises for advanced users - Part III)

Pitlik Marcell, Pitlik László, Pitlik Mátyás, Pitlik László (jun) – MY-X team

Kivonat: A titkosító eljárások készítése során a programozó előbb-utóbb belefut a kód hatékonyságának kérdéskörébe. Hiszen a nagy mennyiségű szövegek átkódolása nem igényelhet tetszőlegesen sok időt valós üzenetváltási folyamatokban, ahol még a kommunikáció maga is időigényes lehet, ill. a kommunikáció zavarása miatt az időigény eleve jelentősen megnőhet. A kódhatékonyságot sokféleképpen lehet növelni. A cikksorozat harmadik, s egyben befejező részében a laikus és a profi kódolás közötti szemléleti/stratégiai és operatív szempontok kerülnek felvillantásra.

Kulcsszavak: hatékonyság, hatásosság, programozás

Abstract: During programming of ciphering algorithms, the authors will be confronted sooner or later with questions concerning code-efficiency. The ciphering/deciphering may not need unlimited time and/or computing resources in case of real processes where the communication itself also needs time and/or the disturbed communication can increase this demand in an extreme way. The efficiency of the source codes can be increased in different (partly parallel) ways. The third (final) part of the series a less and a more optimized source code will be compared where the functionality the same is.

Keywords: efficiency, effectivity, programming

# Bevezetés

A cikksorozat első két részében található programrészletek laikus programozó által kerültek kialakításra, ahol a cél a hatásosság (a matematikai megfelelés) és nem a futásidő-minimalizálás volt. Ebben a részben felvillantásra kerülnek azok a szempontok, melyek mentén maga a program-finomhangolás értelmezhető. Egyes beavatkozások adott kód apróbb változtatásait jelentik, de alapvetően a finomhangolás elvárása kikényszerítheti a kód teljes újra strukturálását.

A jelen cikkben szereplő profi-kód az Olvasók által tételesen összevetendő az első cikkben bemutatott laikus kóddal és az Olvasó saját személyiségfejlődésének szerves részeként elvárható, hogy a hogyanok és a miértek megfelelő számú nekifutásból összeálljanak minél több befogadó fejében.

# A kódoló eljárás másként

Az I. részben látható kódoló eljárás egy nem rutinos programozó gondolatmenetét tükrözi, mely az alábbi szempontok mentén került egy rutinos programozó által újragondolásra:

* a kódszervezés keretében (vö. refaktorálás) újra gondolásra kerültek a
  + struct-ok
  + metódusok
* a korábban már előállított titkosírás folyamatos újra és újra értelmezése helyett bevezetésre került az aktuális állapot leírásának lehetősége
* hatékonysági kérdés volt az ismétlődő kódrészletek előfordulásának minimalizálása és a változó élettartamának optimalizálása
* a dátumkezelés során több beépített C++ függvény került bevonásra a kódba
* a fenti változások érvényesítéséhez új változókra is szükség volt, vagy a meglévő változó nevek mögött tartalmi változás is történt

// Example program

#include <iostream>

#include <string>

#include <map>

#include <sstream>

#include <bitset>

#include <stdlib.h>

#include <time.h>

#include <exception>

using namespace std;

struct kod2 {

int mod;

int kong;

int tarcsa;

kod2(){};

kod2(int m, int k, int t) {

mod = m;

kong = k;

tarcsa = t;

}

};

struct Header {

string date;

Header(string d) {

date = d;

}

string encode() {

if (date.empty())

return string(10, '0');

if (formatInvalid())

throw invalid\_argument("Érvénytelen esedékességi dátum!");

int ho = atoi(date.substr(0, 2).c\_str());

int nap = atoi(date.substr(3, 2).c\_str());

int nap1=nap/10;

int nap2=nap%10;

stringstream ss;

ss << bitset<4>(ho) << bitset<2>(nap1) << bitset<4>(nap2);

return ss.str();

}

private:

bool formatInvalid() {

// TODO proper validation

return false;

}

};

struct Kodolo {

static const map<char, kod2> kod;

static const map<int, pair<string, int> > spkod;

int forgatas;

string uzenet;

stringstream ss;

Kodolo(string u) {

forgatas = 0;

uzenet = u;

}

string kodol() {

srand(time(NULL));

for (int i=0; i<uzenet.length(); i++) {

char betu = uzenet[i];

if (isUnsupported(betu))

throw invalid\_argument("Csak az angol abc kisbetui megengedettek!");

if (isspace(betu))

encodeSpace();

else

encodeLetter(betu);

}

while (forgatas % 12 != 0) {

ss << "1";

forgatas++;

}

return ss.str();

}

private:

bool isUnsupported(char c) {

return c != 32 && (c<97 || c>122);

}

void encodeSpace() {

int sprand = rand()%8 + 1;

auto sp = Kodolo::spkod.at(sprand);

ss << sp.first;

forgatas += sp.second;

forgatas = forgatas % 12;

}

void encodeLetter(char c) {

auto letterData = Kodolo::kod.at(c);

while (forgatas % letterData.mod != letterData.kong) {

int forrand = rand()%2;

ss << forrand;

forgatas++;

}

ss << letterData.tarcsa;

}

};

const map<char, kod2> Kodolo::kod = {

{ 'a', {3,0,2} },

{ 'b', {3,1,2} },

{ 'c', {3,2,2} },

{ 'd', {3,0,3} },

{ 'e', {3,1,3} },

{ 'f', {3,2,3} },

{ 'g', {3,0,4} },

{ 'h', {3,1,4} },

{ 'i', {3,2,4} },

{ 'j', {3,0,5} },

{ 'k', {3,1,5} },

{ 'l', {3,2,5} },

{ 'm', {3,0,6} },

{ 'n', {3,1,6} },

{ 'o', {3,2,6} },

{ 'p', {4,0,7} },

{ 'q', {4,1,7} },

{ 'r', {4,2,7} },

{ 's', {4,3,7} },

{ 't', {3,0,8} },

{ 'u', {3,1,8} },

{ 'v', {3,2,8} },

{ 'w', {4,0,9} },

{ 'x', {4,1,9} },

{ 'y', {4,2,9} },

{ 'z', {4,3,9} },

};

const map<int, pair<string, int> > Kodolo::spkod = {

{ 1, {"0002",3} },

{ 2, {"0003",3} },

{ 3, {"0004",3} },

{ 4, {"0005",3} },

{ 5, {"0006",3} },

{ 6, {"00007",4} },

{ 7, {"0008",3} },

{ 8, {"00009",4} },

};

int main()

{

string uzenet;

string initdatum;

cout << "Mi az uzenet?"<< endl;

getline(cin, uzenet);

cout << "Esedekesseg datuma? (MM/DD)"<< endl;

getline(cin, initdatum);

Header header(initdatum);

Kodolo encoder(uzenet);

try {

cout << 0 << header.encode() << 1 << encoder.kodol();

}

catch (invalid\_argument e) {

cout << "ERR: " << e.what();

}

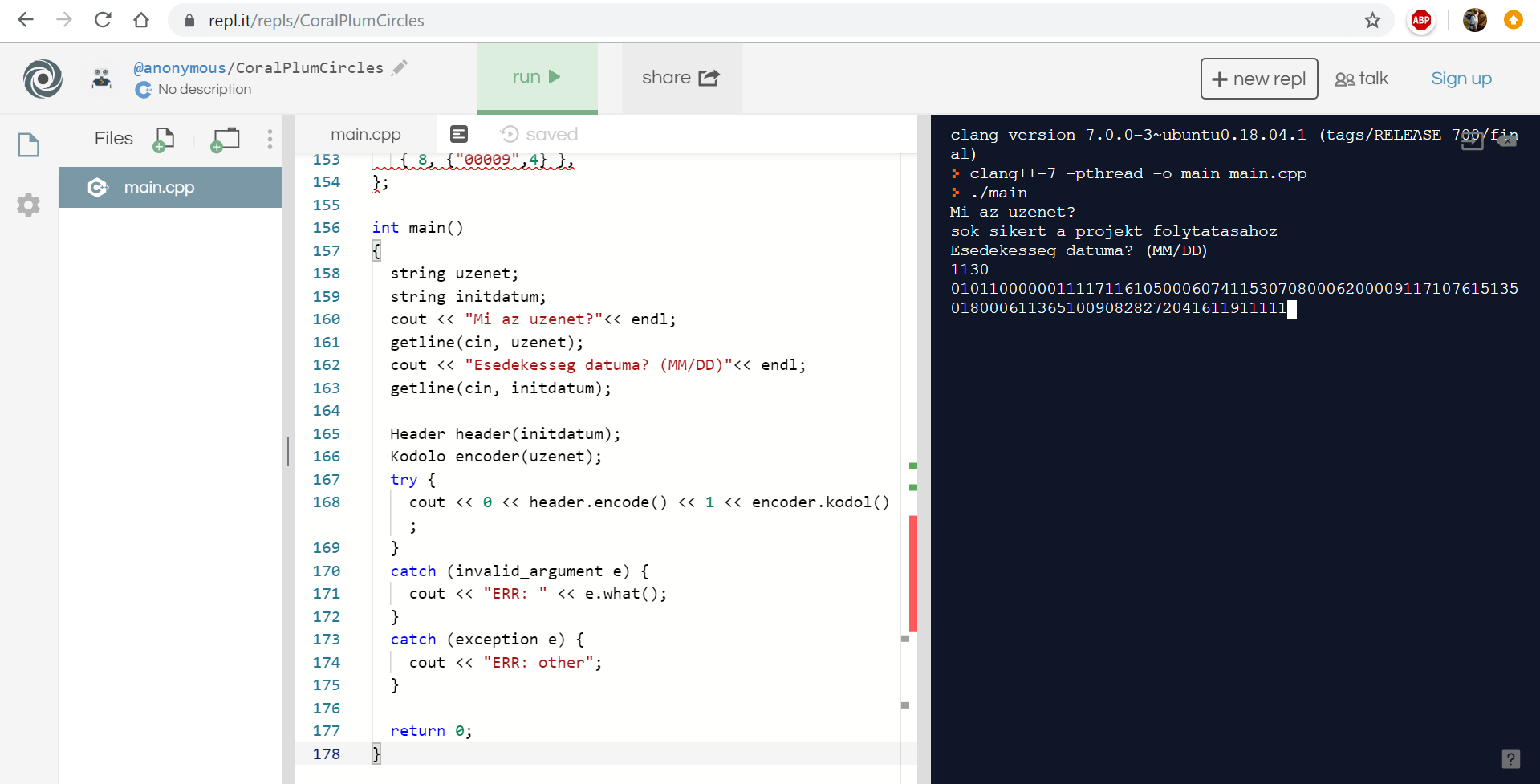
catch (exception e) {

cout << "ERR: other";

}

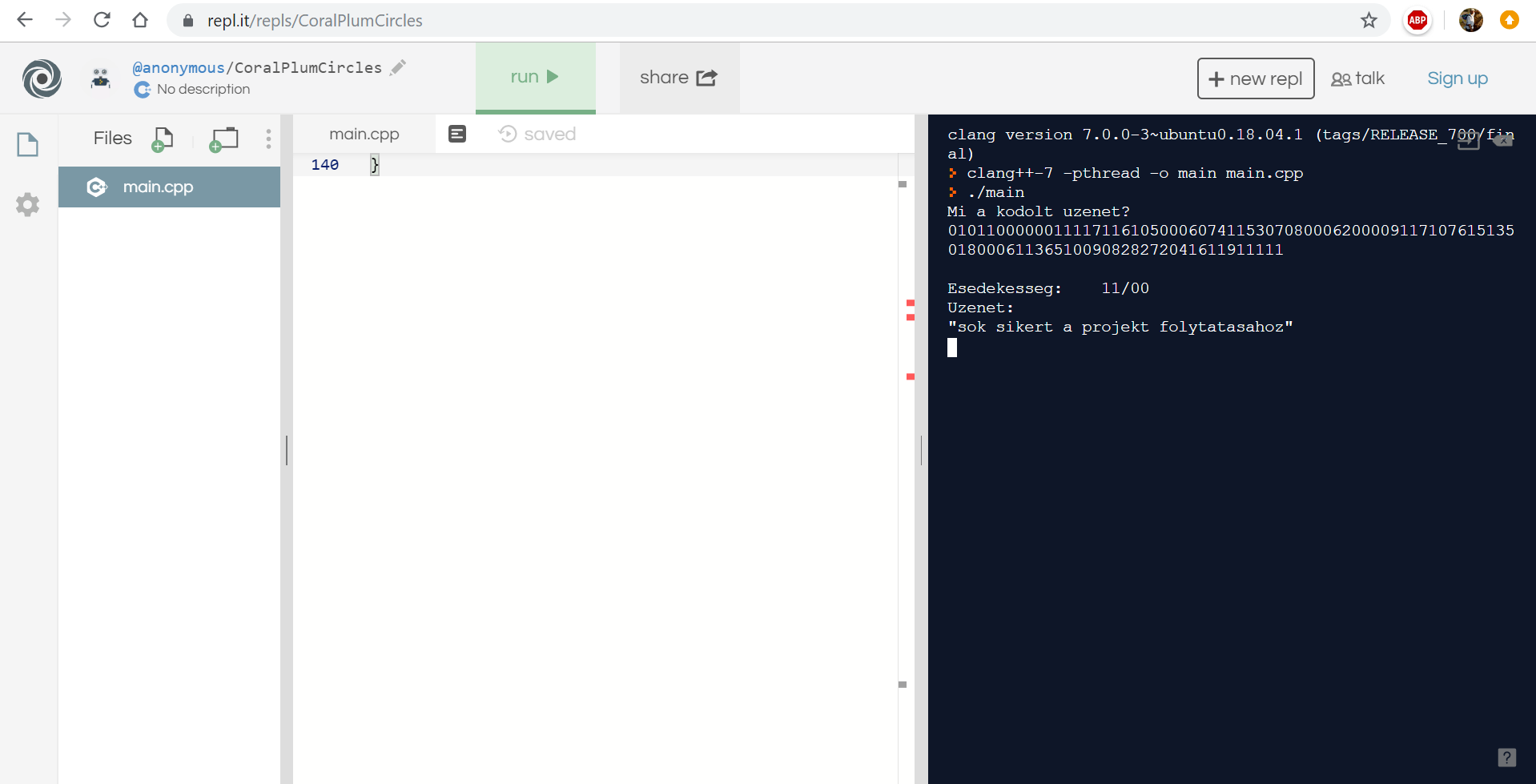
return 0;

}



Tesztüzenet kódolva: 0101100000011117116105000607411530708000620000911710761513501800061136510090828272041611911111

Dekódolási folyamat/ellenőrzés:



Megjegyzés: A dátumkezelés zavara tetten érhető, mert a kihagyott /-jel más dátumot enged vélelmezni, azonban a formátumellenőrzés hiányában a kódolási/dekódolási folyamatok lefutnak.

# Konklúziók

Mint látható, a hatásosság érintetlenül hagyása mellett a hatékonyság érdekében tett lépések jelentősen más programkódot eredményeznek. A programozás tehát csak laikus szinten egy-lendületű-tevékenység. Profi szinten a programtervezés komoly előkészületi időt vár el. Különbség körülbelül akkora, mint ami egy spontán elmondott népmese és a Dan Brown által kínos optimalizálással előállított Da Vinci Kód között érzékelhető bárki által…