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…

A három rész egyben is letölthető: <https://miau.my-x.hu/miau/253/cipher1-2-3.docx>