CodeBlocks

- Vývojové prostředí pro Windows a Linux, zcela zdarma. Instalace na http://www.codeblocks.org/download/5 soubor codeblocks-8.02mingw-setup.exe. Nainstaluje se včetně překladače GCC.
- Uživatelský manuál lze nalézt na www.codeblocks.org.
- Nejdůležitější zkratky: `Ctrl+F9` zkompiluj, `F9` spusť

GMP&NTL

- GMP je knihovna pro práci s libovolně velkými čísly v jazyce `C` a `C++`.
- NTL je knihovna pro práci s libovolně velkými čísly, prvky konečných těles, polynomy, vektory a maticemi pro `C++`.
- Instalace na Windows:
  - stáhněte si http://fox.ucw.cz/gmpntl/gmpntl.zip
  - rozbalte do podadresáře MinGW v místě, kam jste si nainstalovali CodeBlocks, tj. typicky do adresáře C:\Program Files\CodeBlocks\MinGW
- Instalace na Linuxu:
  - stáhněte si zdrojáky knihoven z www.gmplib.org a z www.shoup.net/ntl/
  - rozbalte obě knihovny
  - v rozbalené GMP proveďte `./configure`, pak `make` a na konec `make install`.
  - v rozbalené NTL proveďte `./configure NTL_GMP_LIP=on`, pak `make` a nakonec `make install`.

GMP

- Knihovna pro práci s libovolně velkými čísly.
- Budeme používat především C++ rozhraní.
- Jednoduchý příklad v CodeBlocks:
  - vytvořte nový projekt typu `Console application`
  - v `Project/Build options` vlevo vyberte projekt (ne `Debug` ani `Release`), vpravo `Linker settings` a přidejte dvě knihovny `gmpxx` a `gmp` (na pořadí záleží)
  - do `main.cpp` napište následující kód

```
#include <iostream>
#include <time.h>
#include <gmpxx.h>

using namespace std;

/* vraci, zda-li je cislo prvocislo */
bool rabin_miller(mpz_class a, mpz_class n) {
    mpz_class e=n-1;
    while (e%2 == 0) e>>=1;
    mpz_powm(a.get_mpz_t(), a.get_mpz_t(),
        e.get_mpz_t(), n.get_mpz_t());
    if (a==1) return 1;
    while (e<n-1) {
        mpz_class a2=a*a; a2%=n;
        if (a2==1) return a==n-1;
        a=a2; e<<=1;
    }
    return a==1;
}

int main() {
    mpz_class n; cin >> n;
    if (n<10 || n%2==0) {
        cout << "Trivialni" << endl;
        return 0;
    }
    gmp_randclass r(gmp_randinit_default);
    r.seed(time(0));
    mpz_class a=2+r.get_z_range(n-3);
    if (rabin_miller(a, n))
        cout << "Možná prvočíslo" << endl;
    else cout << "Určitě složené" << endl;
    return 0;
}
```

Nové typy

- `mpz_class` (libovolně velká celá čísla)
- `mpq_class` (libovolně velká racionální čísla)
  - `get_num() get_den()` vrací čitatel a jmenovatel
  - `canonicalize()`
  - nastavení hodnoty `mpq_class p(12,23)`
- `mpf_class` (libovolně přesná desetinná čísla)
  - `mpf_set_default_prec(b)` nastavuje přesnost nově vytvořených typů, `x.set_prec(b)` jednoho.
  - funkce `ceil floor trunc`
- Společné funkce všech tří typů
  - fungují číselné operace `+ - * / %`
    - pracují na aktuálních typech, takže `mpq_class p = 1 / 3` nastaví do `p` nulu
  - bitové operace `& | ^ ~ << >>` Pozor na `^`
  - funkce `abs cmp sgn sqrt`
  - při nastavení hodnoty proměnné lze použít i řetězec, tj. `mpq_class p("123/456"); p="321/654";`
  - `get_str()` vrátí řetězec s hodnotou, `get_si()` vrací `long`, `get_d()` vrací double. Test velikosti pomocí `fits_sshort_p() fits_sint_p() fits_slong_p() fits_ushort_p() fits_uint_p() fits_ulong_p()`
  - načítání `cin >> n`
  - výpis `cout << n`
- `gmp_randclass` (náhodný generátor)
  - při vytvoření specifikace typu generátoru
  - `seed` pro různá počáteční nastavení (`time(0)`)
  - `get_z_bits(n)` vrací `mpz_class` s `n` náhodnými bity
  - `get_z_range(n)` vrací `mpz_class` v rozsahu [0; n-1]
  - `get_f(n)` vrací 0<=`mpf_class`<1 s `n` náhodnými bity
- Další funkce
  - existuje množství dalších funkcí, které ale mají jenom rozhraní pro `C`. Místo `mp?_class` je `mp?_t`, výsledek se předává jako první parametr.
  - `mp?_class` mají funkce `get_mp?_t()`
- Mocnění
  - `mpz_powm(x,a,e,n)` $x = a^e \bmod n$
  - `mpz_pow_ui(x,a,ulong e)` $x = a^e$
- Kořeny
  - `mpz_root(x,a,ulong n)` $x = \lfloor \sqrt[n]{a} \rfloor$
  - `mpz_rootrem(x,y,a,ulong n)` $x = \uparrow\ y = a - x^n$
  - `mpz_sqrt(x,a), mpz_sqrtrem(x,y,a)`
  - `mpz_perfect_power_p(x)`
  - `mpz_perfect_square_p(x)`
- Bity
  - `mpz_setbit(x, ulong bit)`
  - `mpz_clrbit mpz_combit mpz_tstbit`
  - `ulong d=mpz_hamdist(x,y)`
  - `ulong y=mpz_sizeinbase(x,2)` počet bitů čísla x
- Číselně teoretické funkce
  - `mpz_probab_prime_p(n, 10)` Rabin Miller
  - `mpz_nextprime(y,x)` y>x 1. prvočíslo
  - `mpz_gcd(g,a,b)` $g = \mathrm{NSD}(a,b)$
  - `mpz_gcdext(g,s,t,a,b)` $\uparrow = g = sa + tb$
  - `mpz_lcm(n,a,b)` $n = \mathrm{NSN}(a,b)$
  - `mpz_invert(x,a,n)` vrací nula při neúspěchu, nenula při úspěchu a pak $x = a^{-1} \bmod n$
  - `int x=mpz_legendre(a,p)` $x = \left(\frac{a}{p}\right)$
  - `mpz_jacobi(a,b) mpz_kronecker (a,b)`
  - `mpz_bin_ui(x,n,ulong k)` $x = \binom{n}{k}$
- A několik dalších, viz manuál http://gmplib.org/manual/

Aktuální verzi tohoto povídání, stejně jako všechny zmiňované odkazy a binárky naleznete na adrese http://fox.ucw.cz/gmpntl/.

NTL

- Knihovna pro práci s libovolně velkými čísly, konečnými tělesy, polynomy, vektory a maticemi
- Obsahuje velké množství číselně teoretických algoritmů
- Jen pro `C++`
- Sama interně používá `GMP` pro operace s celými čísly
- Jednoduchý příklad v CodeBlocks:
  - vytvořte nový projekt typu `Console application`
  - v `Project/Build options` vlevo vyberte projekt (ne `Debug` ani `Release`), vpravo `Linker settings` a přidejte dvě knihovny `ntl` a `gmp` (na pořadí záleží)
  - do `main.cpp` napište jeden z následujících kódů:

```
//// celá čísla////
#include <NTL/ZZ.h>
NTL_CLIENT
int main() {
   ZZ acc, val; acc = 0;
   while (SkipWhiteSpace(cin))
      cin >> val, acc += val*val;
   cout << acc << endl;    return 0;
}
//// vektory ////
#include <NTL/vec_ZZ.h>
NTL_CLIENT
ZZ sum(const vec_ZZ& v) {
   ZZ acc; acc=0;
   for (long i = 0; i < v.length(); i++) acc += v[i];
   return acc;
}
int main() {
   vec_ZZ v; cin >> v;
   long n = v.length(); v.SetLength(2*n);
   for (long i=0; i<n; i++) v[n+i] = v[n-1-i];
   cout << sum(v) << endl; return 0;
}
//// polynomy ////
#include <NTL/ZZXFactoring.h>
NTL_CLIENT
int main() {
   ZZX f; cin >> f;
   vec_pair_ZZX_long factors; ZZ c;
   factor(c, factors, f);
   cout << c << endl << factors << endl;  return 0;
}
//// modulární aritmetika ////
#include <NTL/ZZ_pX.h>
#include <NTL/ZZ_pXFactoring.h>
NTL_CLIENT
int main() {
   ZZ_p::init(to_ZZ(17));
   ZZ_p a; cin >> a; cout << 1/a << endl;
   ZZ_pX f; cin >> f;
   vec_pair_ZZ_pX_long factors;
   berlekamp(factors, f); /*nebo*/ CanZass(factors, f);
   cout << factors << endl;    return 0;
}
//// rozšíření okruhů a těles ////
#include <NTL/ZZ_pEXFactoring.h>
NTL_CLIENT
int main() {
   ZZ_p::init(to_ZZ(17));
   ZZ_pX P; BuildIrred(P, 10);
   ZZ_pE::init(P);                  // GF(17^10)
   ZZ_pEX f; random(f, 20); SetCoeff(f, 20);
   vec_pair_ZZ_pEX_long factors;
   CanZass(factors, f);
   cout << factors << endl;
   return 0;
}
```

- Třídy NTL se musí před použitím `#include <NTL/??.h>`
- Vše jde načítat a vypisovat pomocí `cin >>` a `cout <<`
- Všechny číselné a polynomiální typy implementují
  - `+ - += -= ++ -- * *= / /=`
  - operátory `% %=` implementují všechna netělesa
  - `== !=` a když to má smysl, tak i `< <= >= >`
  - `>> >>= << <<=` násobení a dělení 2-kou či x-em
- Až na výjimky (přiřazení výrazu do proměnné) nelze používat čísla nebo řetězce tam, kde NTL čeká ZZ, ZZ_p, nelze použít ZZ kde NTL čeká ZZX atd. Převody jdou dělat funkcí `to_??: to_ZZ(5), to_ZZX(to_ZZ(5)), ...`
- Všechny funkce jde volat jako `x=f(a,b)` či `f(x,a,b)`. Pokud je třeba rozlišit `f` dle typu `x`, tak `x=f_`*`TypeOfX`*`(a,b)`.

Hlavní třídy NTL

- Číselné typy (dále označované jako *`NumType`*)
  - `ZZ`: celá čísla, libovolně dlouhá. Kromě operátorů umí ještě `g=GCD(a,b)`, `d=XGCD(s,t,a,b)`, modulární aritmetiku `x=AddMod(a,b,n)`, `SubMod(a,b,n)`, `NegateMod(a,n)`, `MulMod(a,b,n)`, `SqrMod(a,n)`, `InvMod(a,n)`, `PowerMod(a,e,n)`, bitové operace `NumBits(x)`, `bit(x,k)`, `SetBit(x,k)`, `SwitchBit(x,k)`, náhodná čísla `SetSeed(ZZ)`, `x=RandomBnd(n)` $\lfloor x\in[0,n-1]\rfloor$, `x=RandomBits(k)` $\lfloor x\in[0,2^k-1]\rfloor$, `x=RandomLen(k)` $\lfloor x\in[2^{k-1},2^k-1]\rfloor$, prvočíselné operace `ProbPrime(p, long pokusu)` (Rabin Miller), `p=RandomPrime(l)` (`p` náhodné `l`-bitové prvočíslo), `p=NextPrime(p0)` a spoustu dalších
  - `ZZ_p`: čísla modulo `p`. Před prvním použití musíte zavolat `ZZ_p::init(ZZ p)`. Kromě standardních operátorů umí ještě `ZZ rep()` (vrací hodnotu jako `ZZ`), `x=power(a,e)`, `x=random_ZZ_p()`. Nepředpokládá se změna modulu, ale jde provést pomocí `ZZ_pBak`.
  - `zz_p`: rychlá varianta `ZZ_p` pro `p`<2^30; `NTL/lzz_p.h`
  - `GF2`: rychlá varianta `ZZ_p` pro `p=2`.
  - `ZZ_pE zz_pE GF2E`: rozšíření příslušných okruhů / těles. Před prvním použitím je třeba provést *`Type`*`::init(p)` pro `p` polynom stupně >0. Kromě operátorů podporuje `x=random_`*`Type`*`()`, `power`, `x=trace(a)`, `x=norm(a)`. Modul lze změnit – *`Type`*`Bak`.
  - `RR`: čísla s plovoucí desetinnou čárkou s přesností na `p`(výchozí 150) bitů. Lze změnit pomocí `RR::SetPrecision(p)`, `RR:SetOutputPrecision(p)` nastavuje počet desítkových cifer při výpisu. Kromě operátorů podporuje klasické funkce jako `trunc`, `floor`, `ceil`, `round`, `exp`, `log`, `log10`, `pow`, `sin`, `cos`, `ComputePi_RR`, `random`, `SqrRoot`, `power` a jiné.
- Vektory
  - `vec_`*`NumTyp`*: vektory daného typu. Mají délku `v.length()`, ta lze změnit pomocí `v.SetLength()`. Umí se pomocí operátorů sčítat, odčítat, násobit skalárem a počítat skalární násobení. Přístup k prvkům vektoru pomocí `v[i]=4`, čísluje se od nuly.
- Matice
  - `mat_`*`NumTyp`*: matice daného typu s rozměry `m.NumRows()` a `m.NumCols()`. Nastavit rozměry jde pomocí `m.SetDims(r,c)`. K prvkům se přistupuje `m[i]` a `m[i][j]=m(i,j)`. Kromě sčítání, násobení skalárem a maticí podporují `power`, `n=transpose(m)`, `determinant(m)`, `solve(d,X,a,b)` (spočte det do `d` a je-li !=0 vyřeší soustavu `X*a=b`), `inv(d,X,Y)` (Y=X^{-1} pro d! =0), `image(X,A)`, `kernel(X,A)`.
- Polynomy
  - *`NumTypeX`*: Kromě klasických operací podporují `GCD` a `XGCD`, `deg(f)`, `random(deg)`, `coef(f,n)`, `LeadCoef(f)`, `SetCoef(f,n,ZZ a=1)`, modulární operace `MulMod(a,b,f)`, `SqrMod(a,f)`, dále `TraceMod(a,f)`, `NormMod(a,f)`, `resultant(a,b)`, `discriminant(f)`. Jednotlivé koeficienty uloženy ve vektoru `f.rep`.
  - *`NumTypeX`*`Factoring`: metody pro faktorizaci daných polynomů. Všechny kromě `ZZX` podporují funkci `CanZass(vec_pair_`*`NumTypeX`*`_long factors, f)`. V příslušných případech také `berlekamp` se stejnými parametry. Dále `BuildIrred_`*`NumTypeX`*`(deg)` hledá monický ired. polynom, `BuildSparseIrred_GF2X(deg)` hledá monický ired. tvaru x^n+x^k+1 či $x^n+x^{k1}+x^{k2}+x^{k3}+1$ (pro rychlé GF2E). Na `ZZX` je `factor(cont,factors,f)`
- `HNF`: funkce `HNF(mat_ZZ W,mat_ZZ A,ZZ d)`
- `LLL`: funkce `LLL(ZZ det2,mat_ZZ B)` provede LLL. Dále `NearVector(vec_ZZ w,mat_ZZ B,vec_ZZ a)` hledání aproximace nejbližšího vektoru pro B LLL reduk. (Babai).
- Podrobnosti na www.shoup.net/ntl/doc/tour.html.

Aktuální verzi tohoto povídání, stejně jako všechny zmiňované odkazy a binárky naleznete na adrese http://fox.ucw.cz/gmpntl/.