NTL
Number Theory Library

Zdroje

- http://www.shoup.net/ntl/
  - Download, Dokumentácia
- http://147.175.106.2/Tim6Download
  - Dokumentácia v slovenčine – stiahnuť 4. verziu

NTL

- C++ knižnica, podporujúca prácu s:
  - Celými / reálnymi číslami ľubovoľnej dĺžky
  - Konečnými poliami GF(p), GF(p^k)
  - Okruhmi polynómov nad Z[x], GF(p)[x], GF(p^k)[x]
  - Vektormi, maticami nad Z, R, GF(p), GF(p^k)

NTL

- Obsahuje implementácie viacerých užitočných algoritmov:
  - Euklidov algoritmus, rozšírený Euklidov algoritmus
  - CRT, Jacobiho symbol
  - Faktorizácie polynómov nad Z, GF(p), GF(p^k)
  - Maticové algoritmy (GEM, inverzia, ...)
  - Implementácia LLL algoritmu
  - Testovanie prvočíselnosti, generovanie ireducibilných polynómov, interpolácia,...

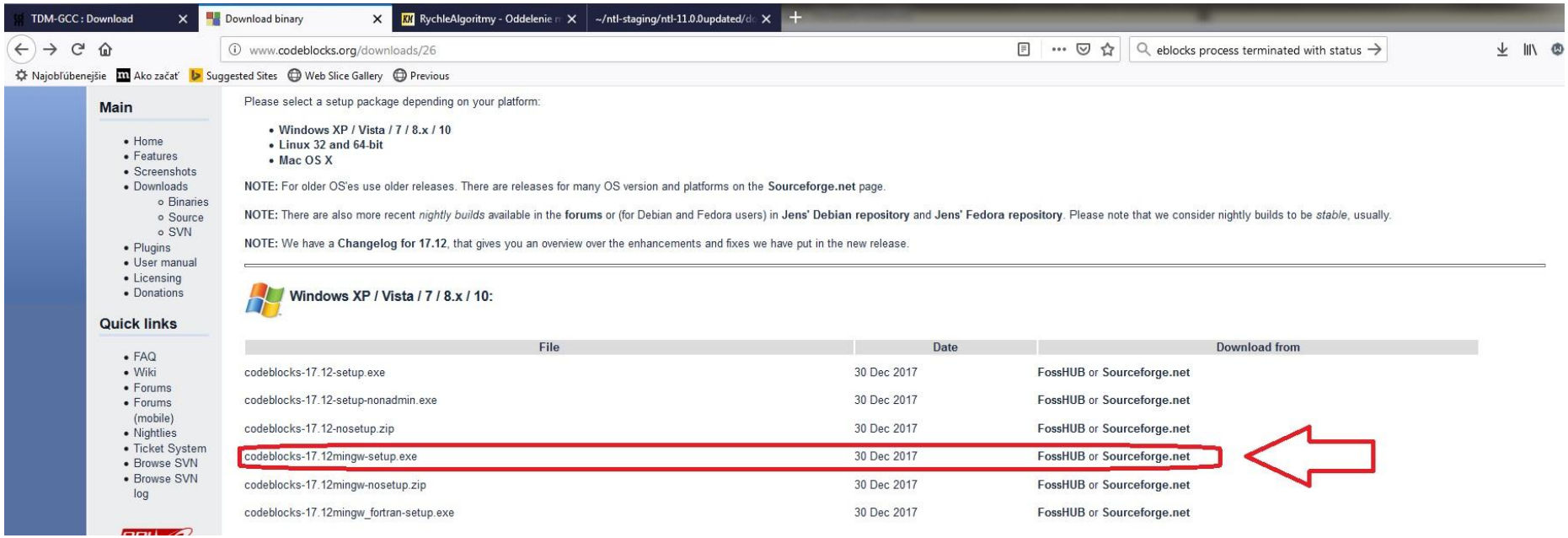
NTL – inštalácia (Unix)

- http://www.shoup.net/ntl/doc/tour-unix.html

- V Unix-e odporúčam (tak ako autor knižnice) používať aj knižnice GMP a gf2x

NTL – inštalácia (Windows)

- Od verzie 11.x.x Je potrebný kompilátor s podporou C++11
- Napr. TDM-GCC obsahujúci GCC/G++ 5.1.0, http://tdm-gcc.tdragon.net/download
- Prípadne priamo nejaké IDE, napr. Code::Blocks s GCC/G++, http://www.codeblocks.org/downloads/26 a tam vybrať verziu s MinGW


TDM-GCC : Download
Download binary
RychleAlgoritmy - Oddelenie
~/ntl-staging/ntl-11.0.0updated/
www.codeblocks.org/downloads/26
eblocks process terminated with status
Najobľúbenejšie
Ako začať
Suggested Sites
Web Slice Gallery
Previous
Main
Home
Features
Screenshots
Downloads
Binaries
Source
SVN
Plugins
User manual
Licensing
Donations
Quick links
FAQ
Wiki
Forums
Forums (mobile)
Nightlies
Ticket System
Browse SVN
Browse SVN log
Please select a setup package depending on your platform:
Windows XP / Vista / 7 / 8.x / 10
Linux 32 and 64-bit
Mac OS X
NOTE: For older OS'es use older releases. There are releases for many OS version and platforms on the Sourceforge.net page.
NOTE: There are also more recent nightly builds available in the forums or (for Debian and Fedora users) in Jens' Debian repository and Jens' Fedora repository. Please note that we consider nightly builds to be stable, usually.
NOTE: We have a Changelog for 17.12, that gives you an overview over the enhancements and fixes we have put in the new release.
Windows XP / Vista / 7 / 8.x / 10:
File
Date
Download from
codeblocks-17.12-setup.exe
30 Dec 2017
FossHUB or Sourceforge.net
codeblocks-17.12-setup-nonadmin.exe
30 Dec 2017
FossHUB or Sourceforge.net
codeblocks-17.12-nosetup.zip
30 Dec 2017
FossHUB or Sourceforge.net
codeblocks-17.12mingw-setup.exe
30 Dec 2017
FossHUB or Sourceforge.net
codeblocks-17.12mingw-nosetup.zip
30 Dec 2017
FossHUB or Sourceforge.net
codeblocks-17.12mingw_fortran-setup.exe
30 Dec 2017
FossHUB or Sourceforge.net

NTL – inštalácia (Windows)

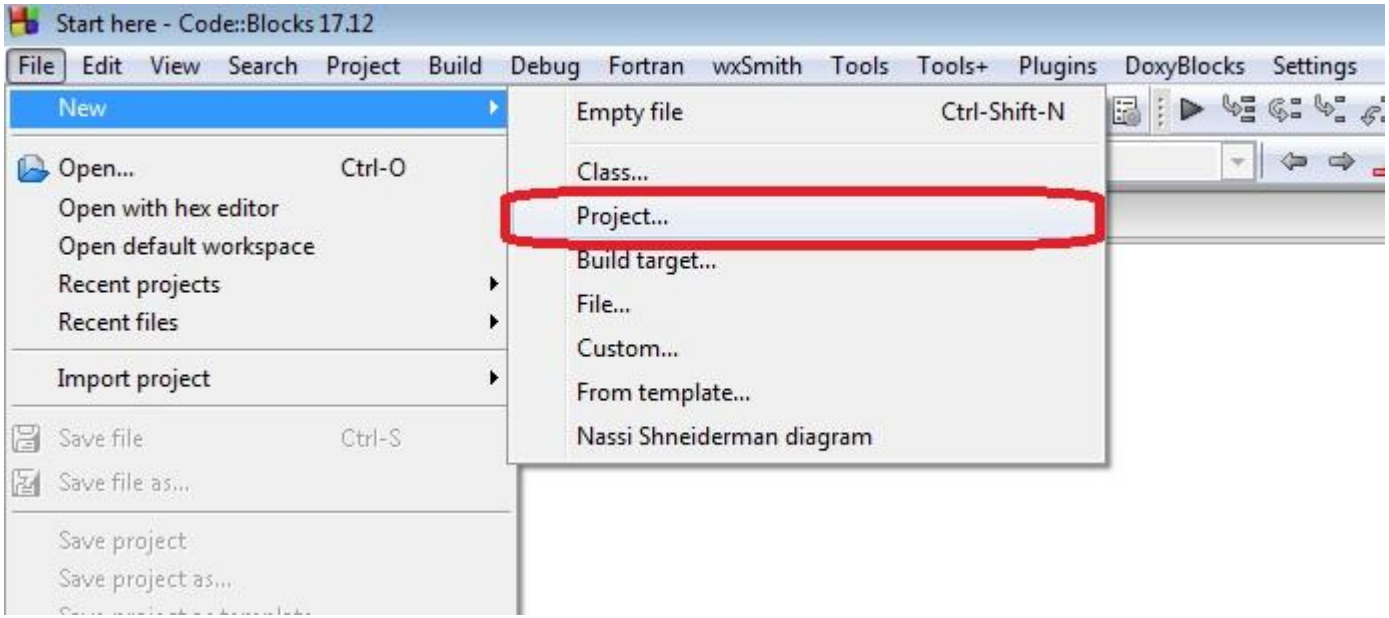
- Ďalej stiahnuť NTL pre Windows (aktuálne je najnovšia verzia 11.3.2 ku 02/04/2019 https://www.shoup.net/ntl/WinNTL-11_3_2.zip
- Avšak na starších IDE / verziách GCC je možno vhodnejšia verzia 10.5.0 (posledná verzia pred 11.x.x)
- Archív rozbaliť do nejakého adresára

NTL – inštalácia (Windows)

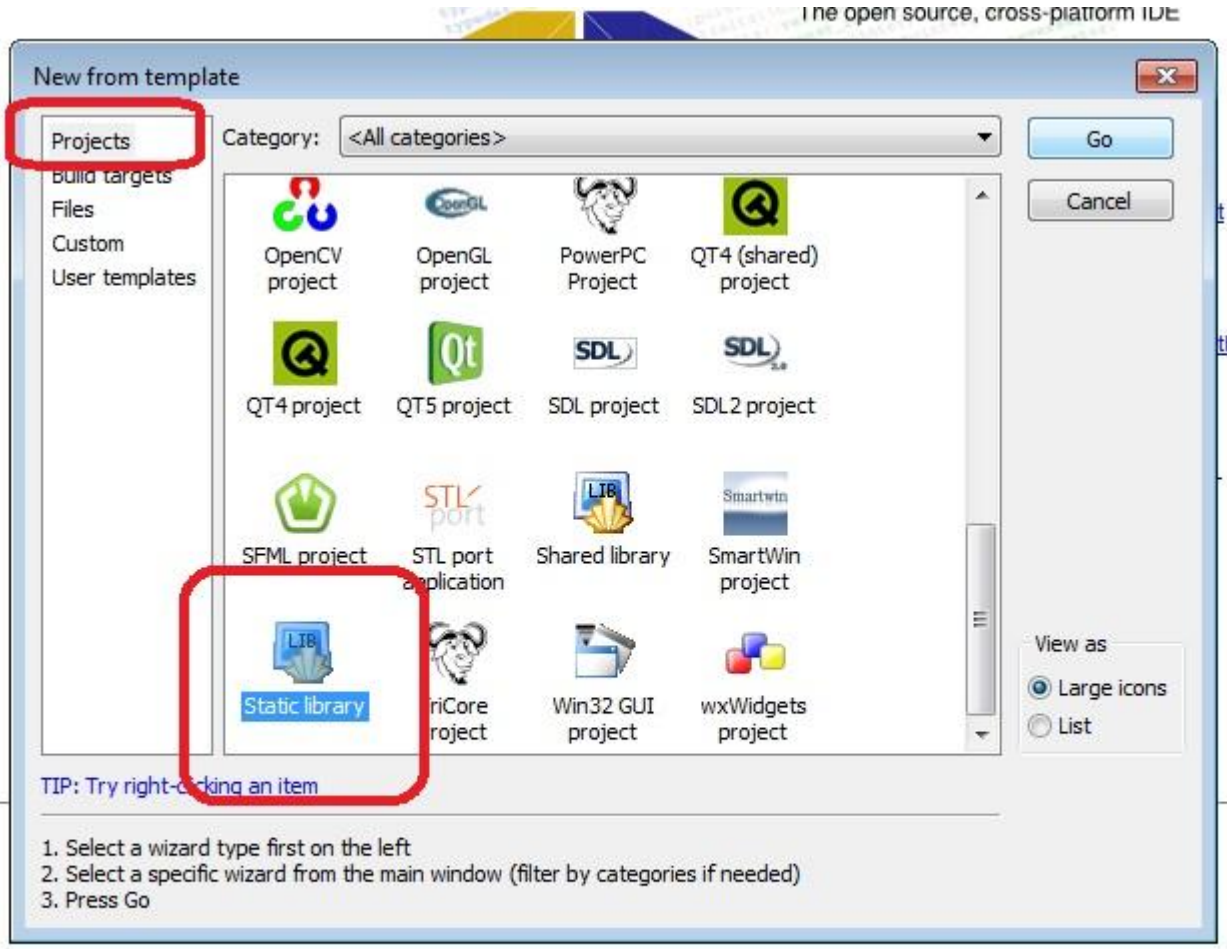
- „Your code will be much snappier, and your quality of life will be much better.“
- Postup pre „inštaláciu“ NTL
  - 1) Stiahnuť
  - 2) Vytvoriť statickú knižnicu
    - 2.1) zo zdrojových kódov z adresára „src“
    - 2.2) Nastaviť cestu k hlavičkovým súborom v „include“
  - ----------
  - 3) Prilinkovať statickú knižnicu k programu
  - 4) Taktiež nastaviť cestu k hlavičkovým súborom
  - 5) pridať makro NTL_CLIENT
- http://www.shoup.net/ntl/doc/tour-win.html

NTL – vytvorenie statickej knižnice v prostredí Code::Blocks

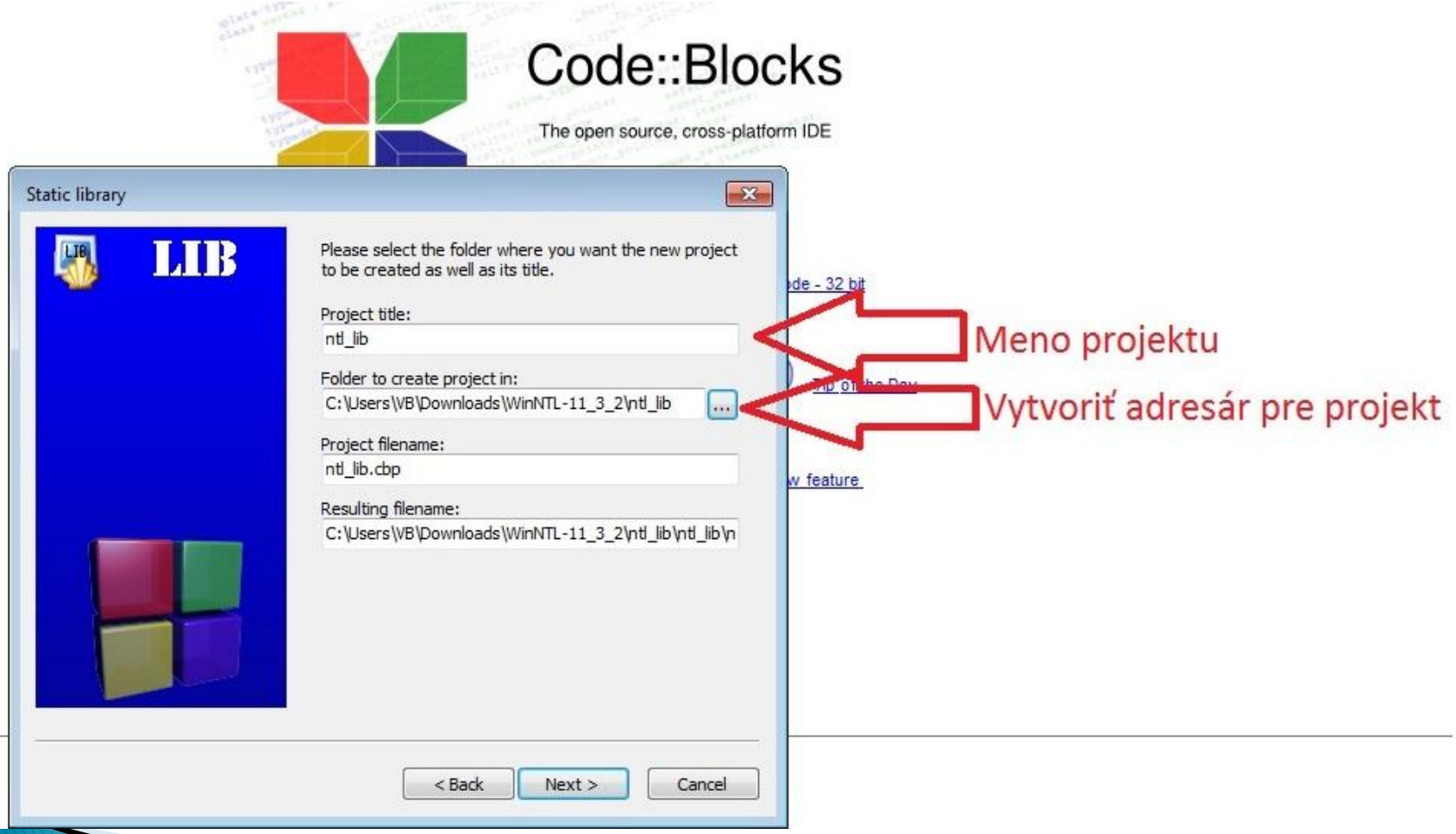
Vytvoríme nový projekt typu „Static Library“



NTL – vytvorenie statickej knižnice v prostredí Code::Blocks

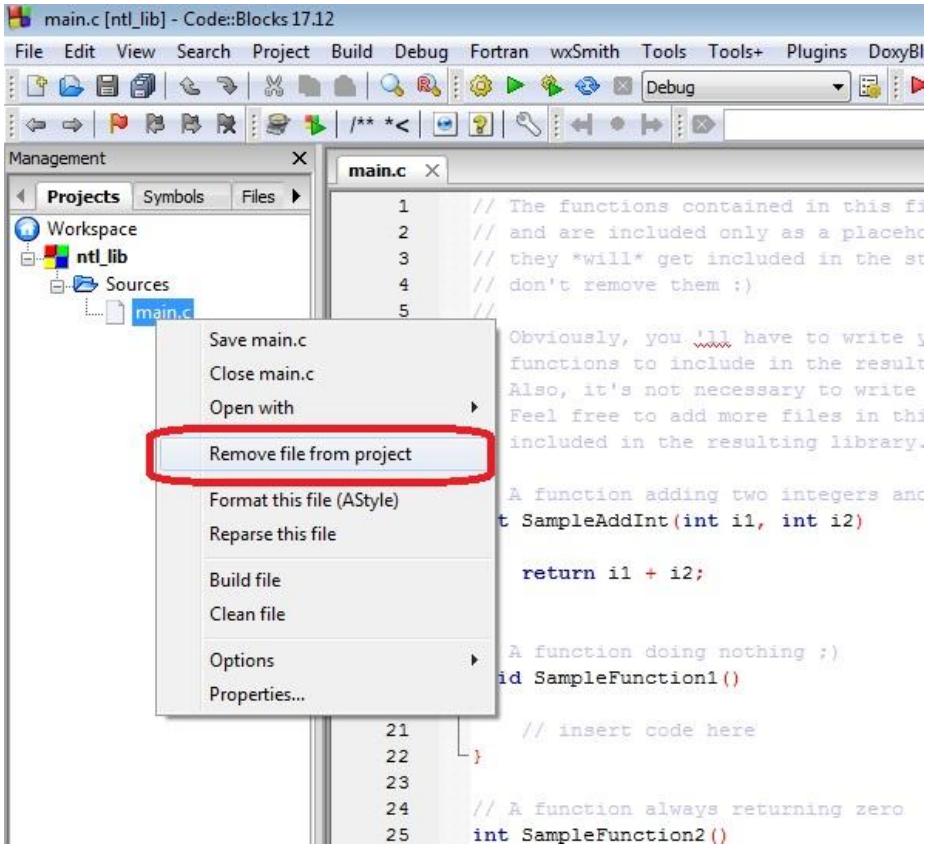


NTL – vytvorenie statickej knižnice v prostredí Code::Blocks



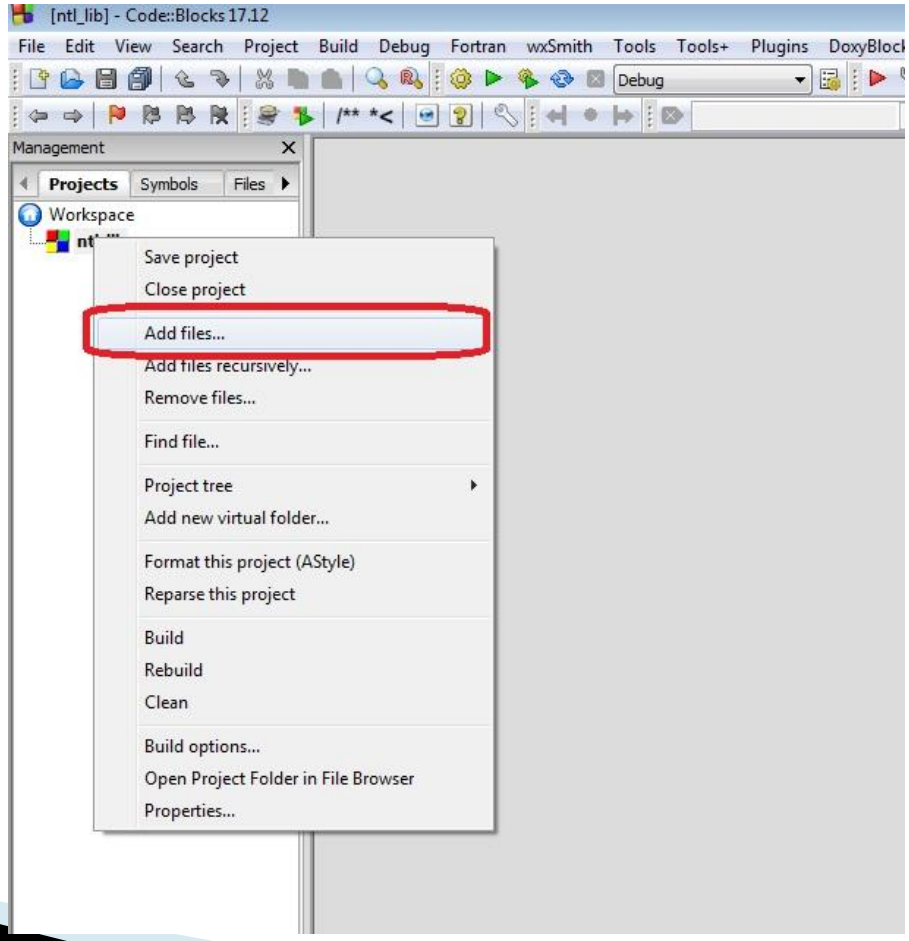
NTL – vytvorenie statickej knižnice v prostredí Code::Blocks

Code::Blocks v projekte vytvorí súbor main.c – ten zmažeme

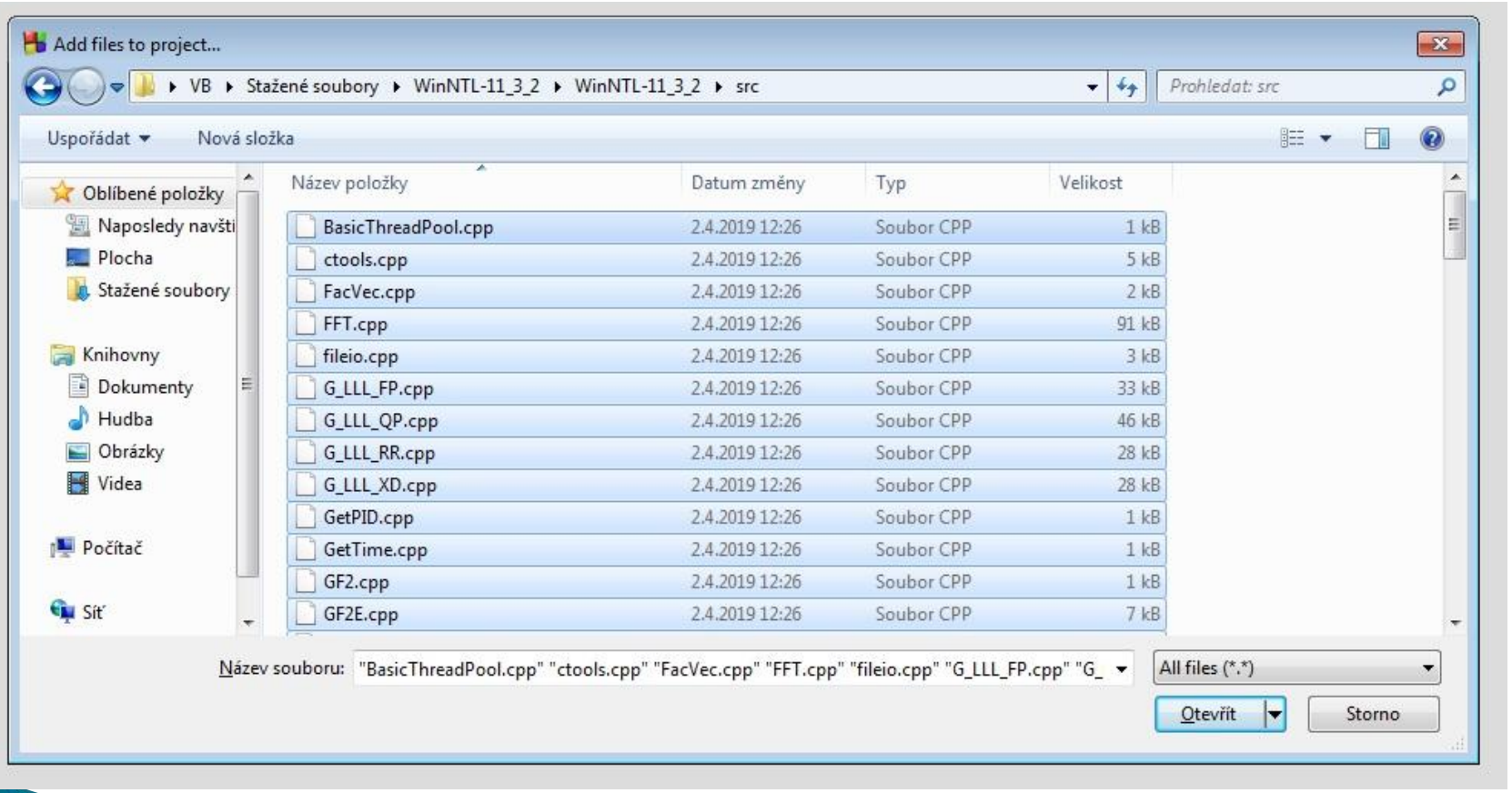


NTL – vytvorenie statickej knižnice v prostredí Code::Blocks

Následne do projektu pridáme **všetky** súbory z adresára „src“ knižnice NTL

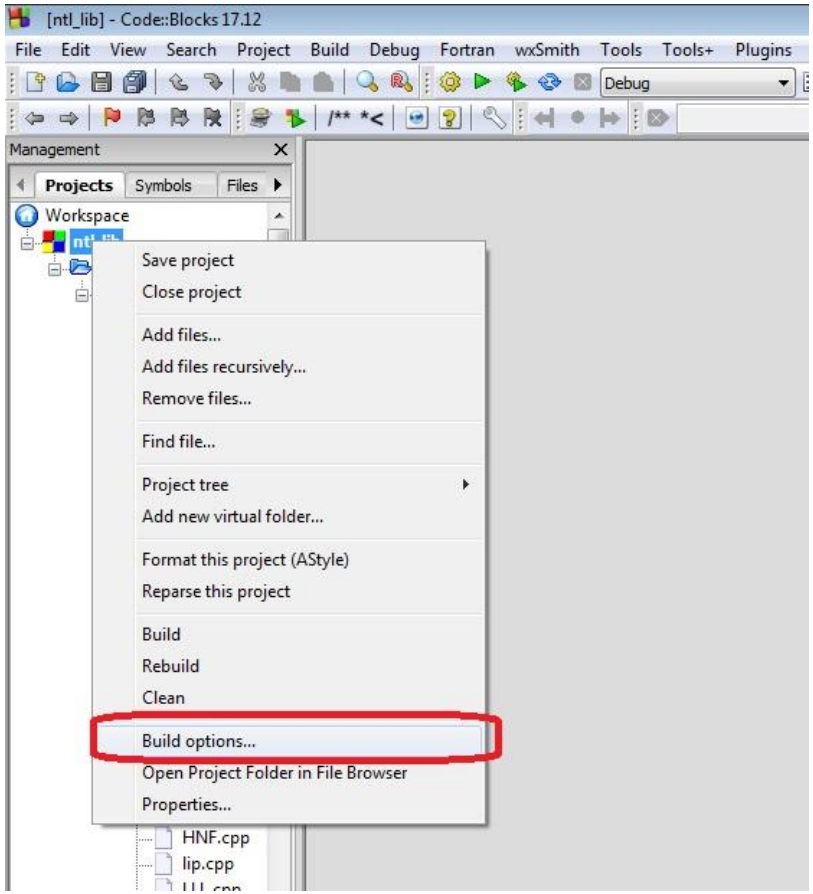


NTL – vytvorenie statickej knižnice v prostredí Code::Blocks



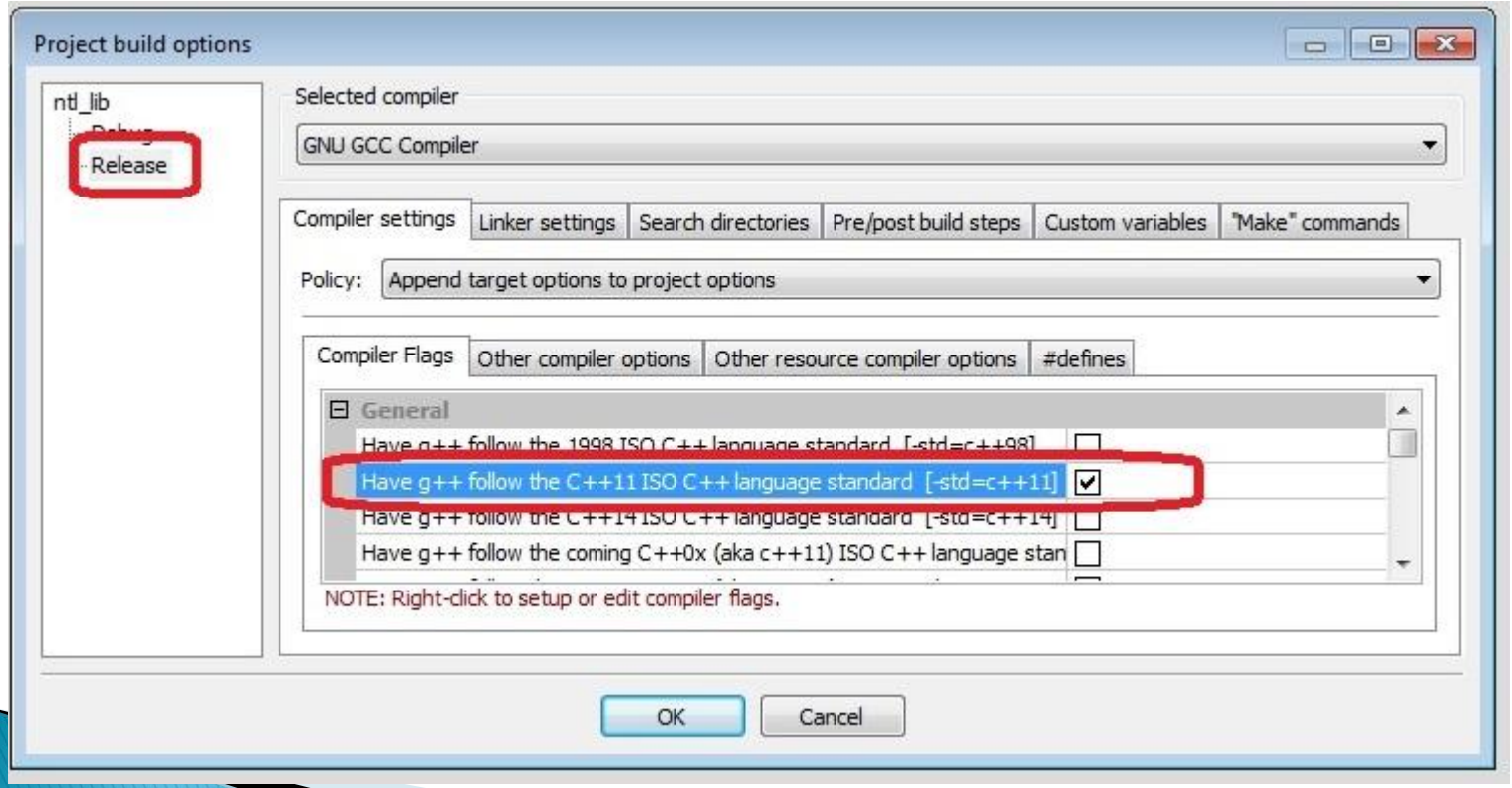
NTL – vytvorenie statickej knižnice v prostredí Code::Blocks

Ďalej musíme nastaviť cestu k hlavičkovým (*.h) súborom a nastaviť kompilátor na verziu C++11 (pre verziu NTL 11.x.x) cez „Build options...“



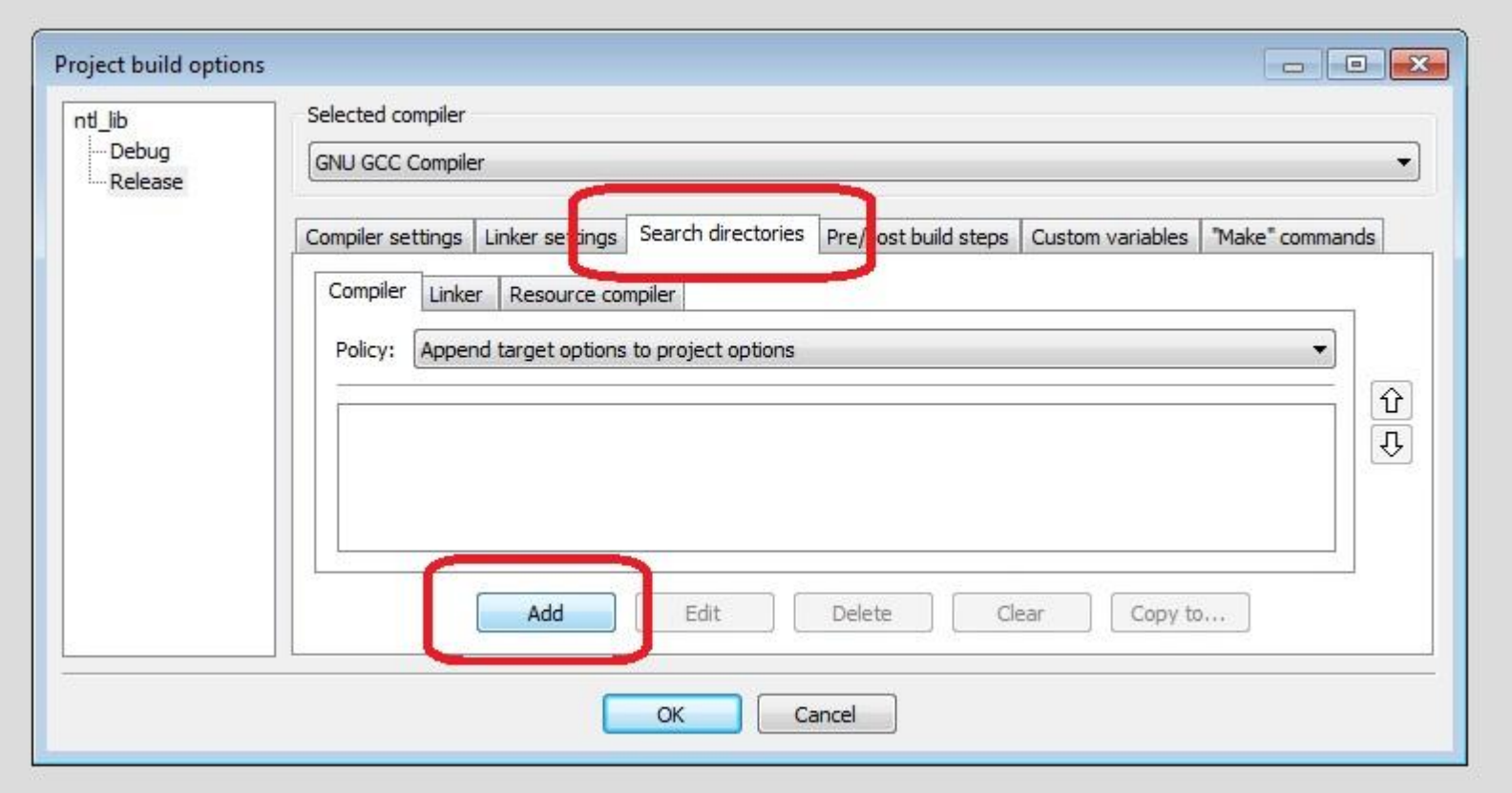
NTL – vytvorenie statickej knižnice v prostredí Code::Blocks

Zmeny budeme robiť pre Release verziu knižnice (ten istý postup sa dá použiť aj pre Debug verziu) – tu nastavíme kompilátor na C++11 – tento krok vynechať pre verziu 10.5.0 a menej



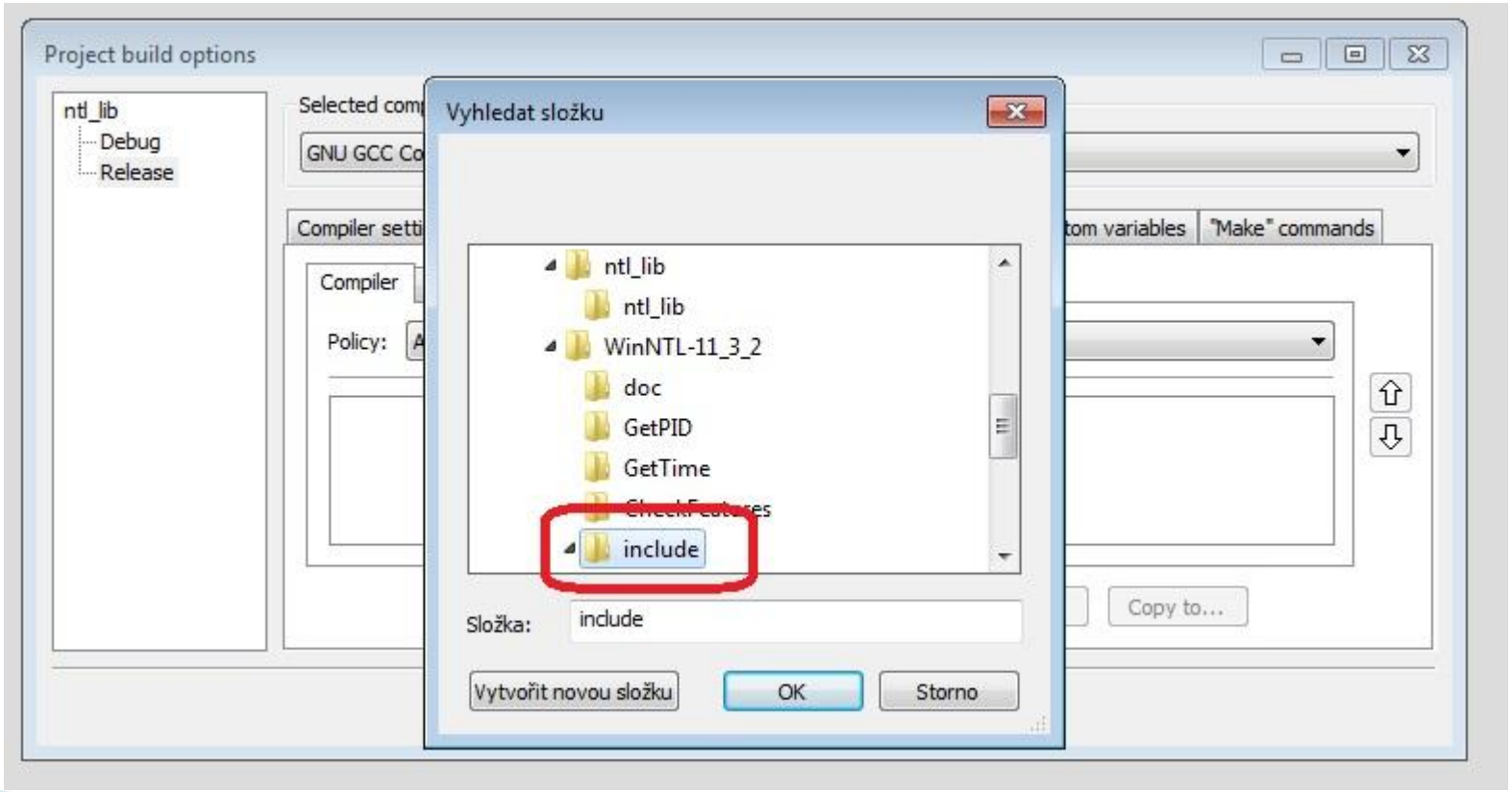
NTL – vytvorenie statickej knižnice v prostredí Code::Blocks

... A tu cestu k hlavičkovým súborom



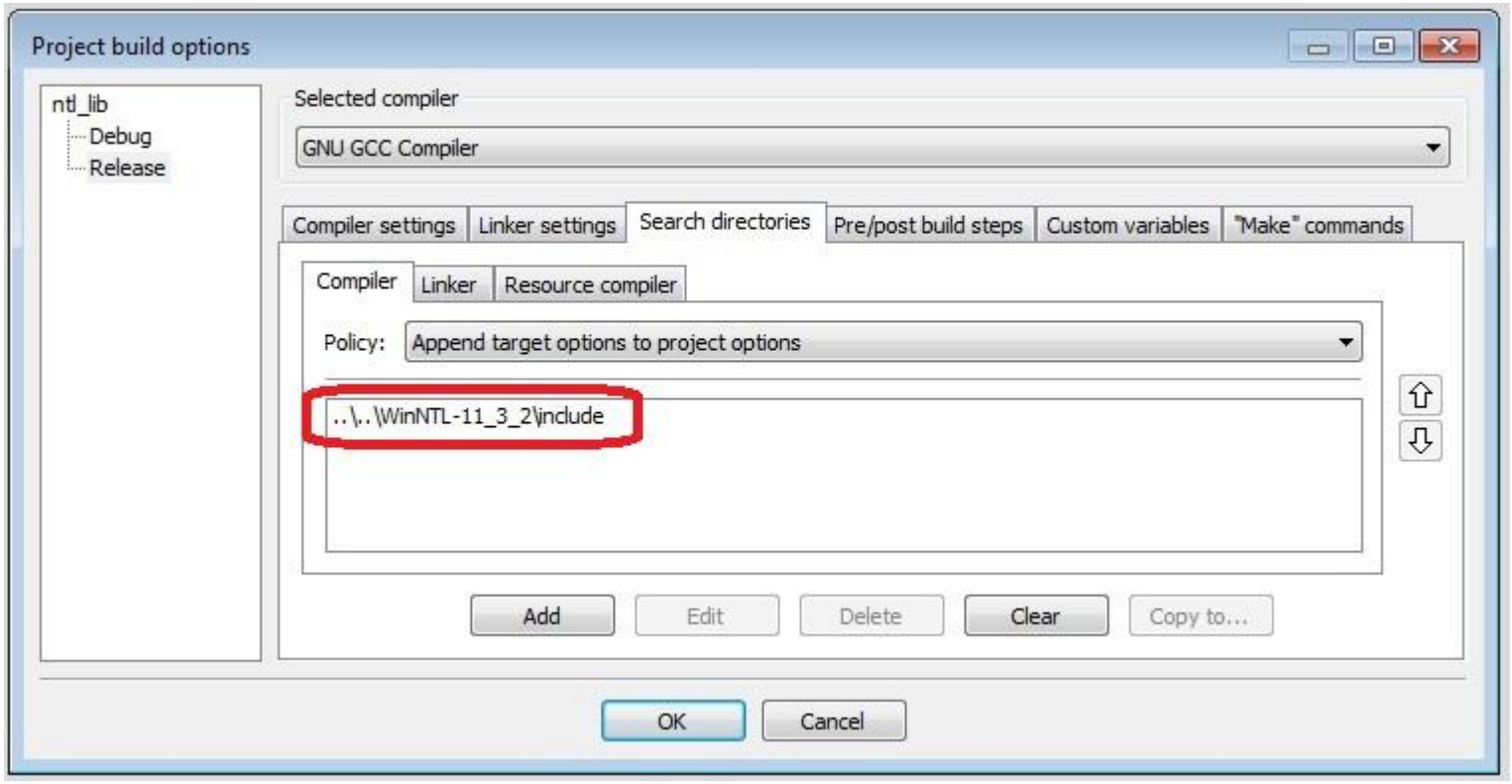
NTL – vytvorenie statickej knižnice v prostredí Code::Blocks

Tie sa nachádzajú v adresári „include“ knižnice NTL.



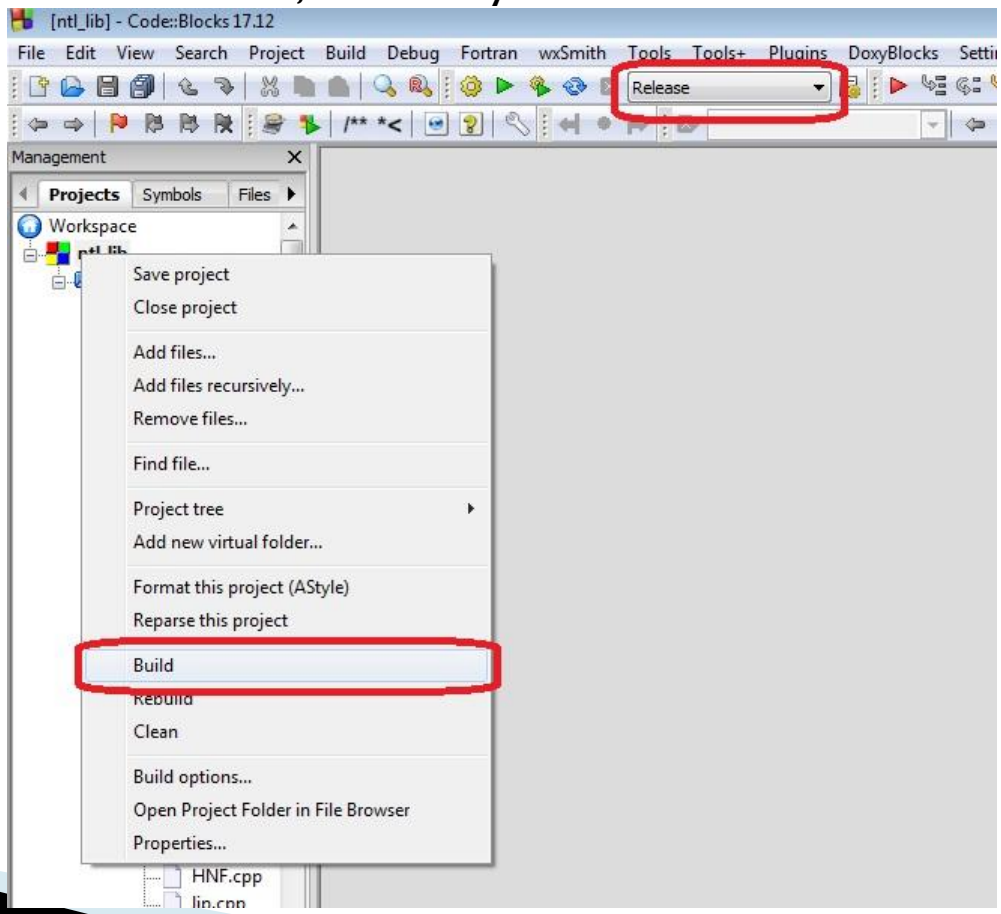
NTL – vytvorenie statickej knižnice v prostredí Code::Blocks

POZOR!!! Vkladajte cestu tak, aby končila „include“, NIE „include/NTL“ !!!



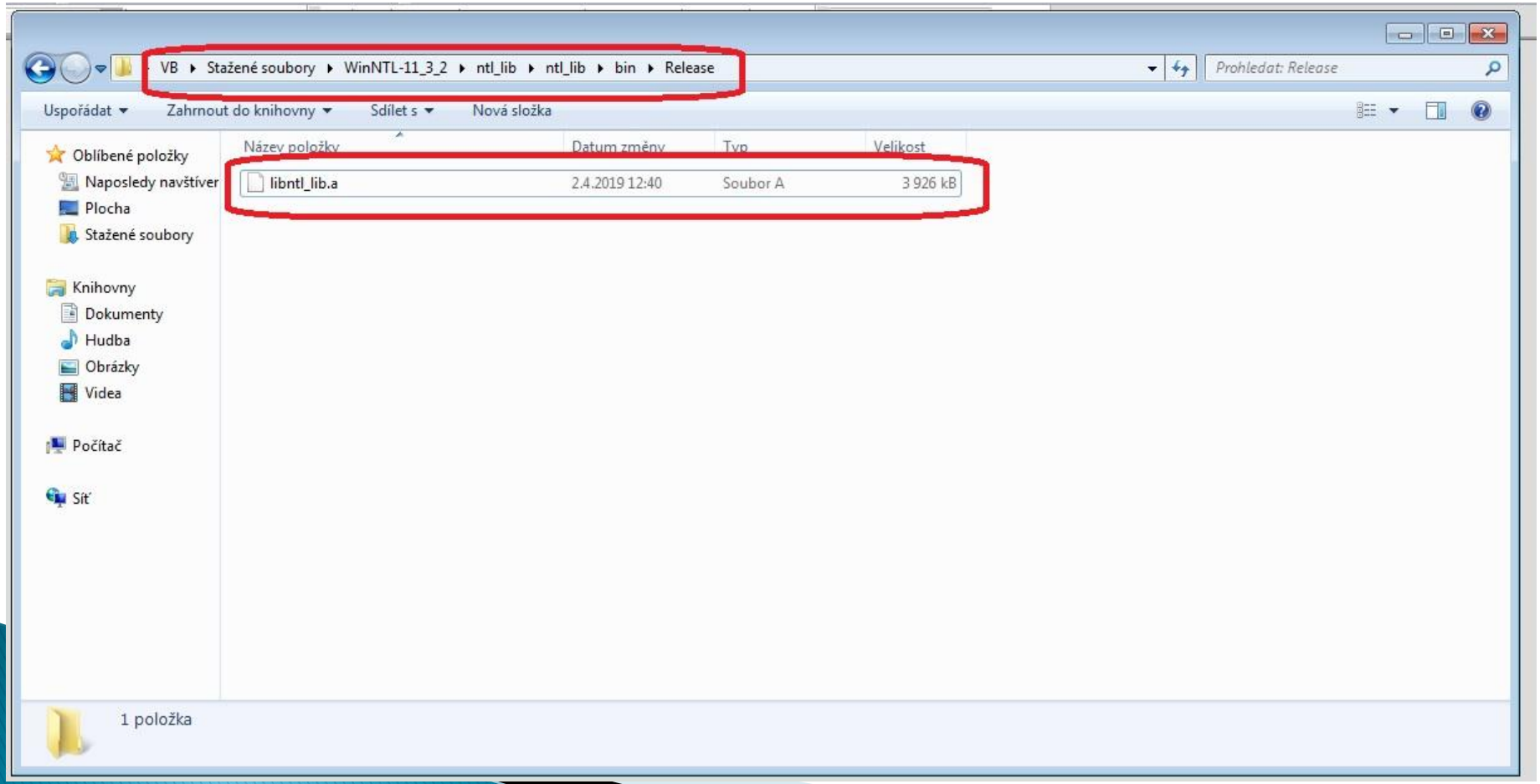
NTL – vytvorenie statickej knižnice v prostredí Code::Blocks

Následne už len nastavíme aktuálne nastavenie projektu na Release a vyberieme možnosť Build, ktorá vytvorí statickú knižnicu



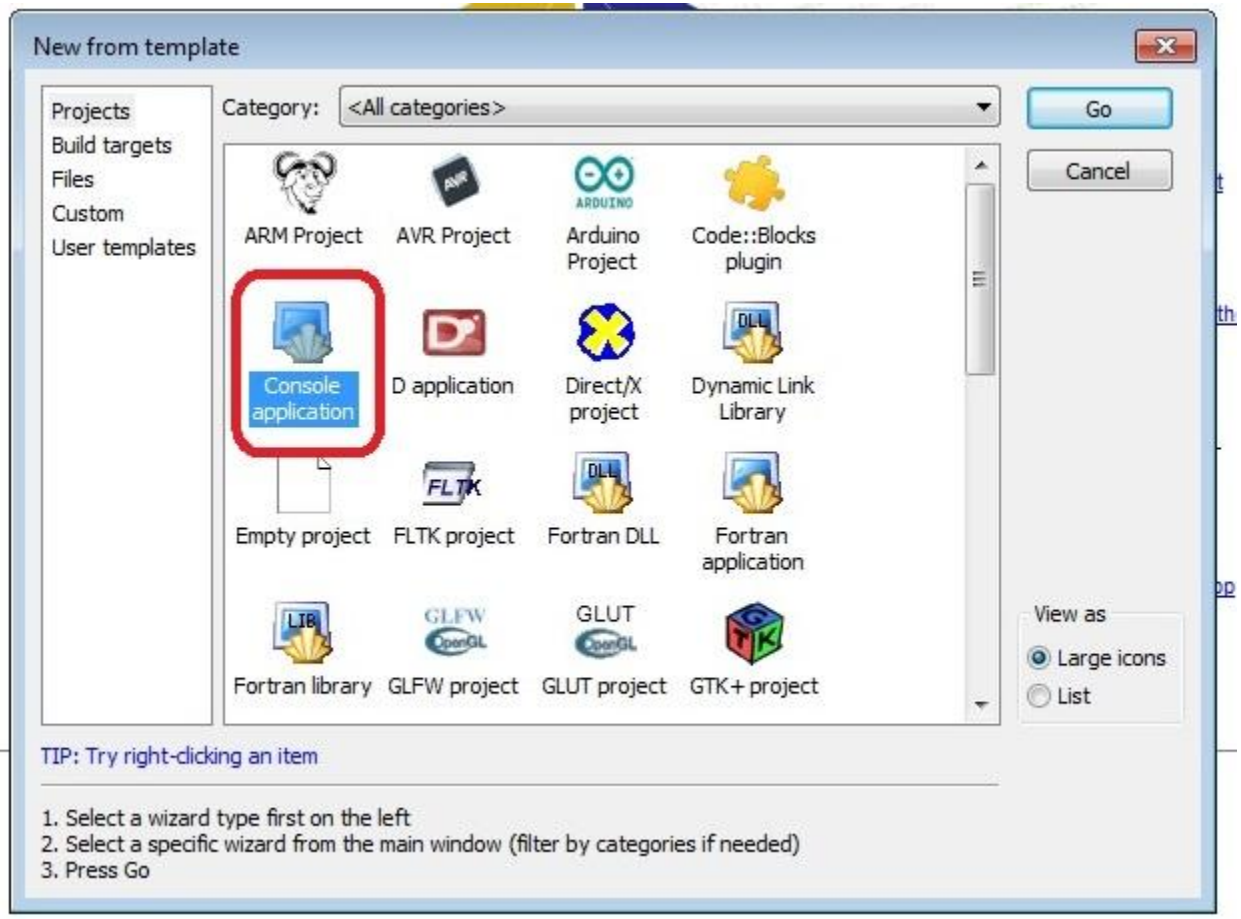
NTL – vytvorenie statickej knižnice v prostredí Code::Blocks

Výsledná knižnica (súbor s koncovkou .a) je v príslušnom podadresári projektu bin/Release



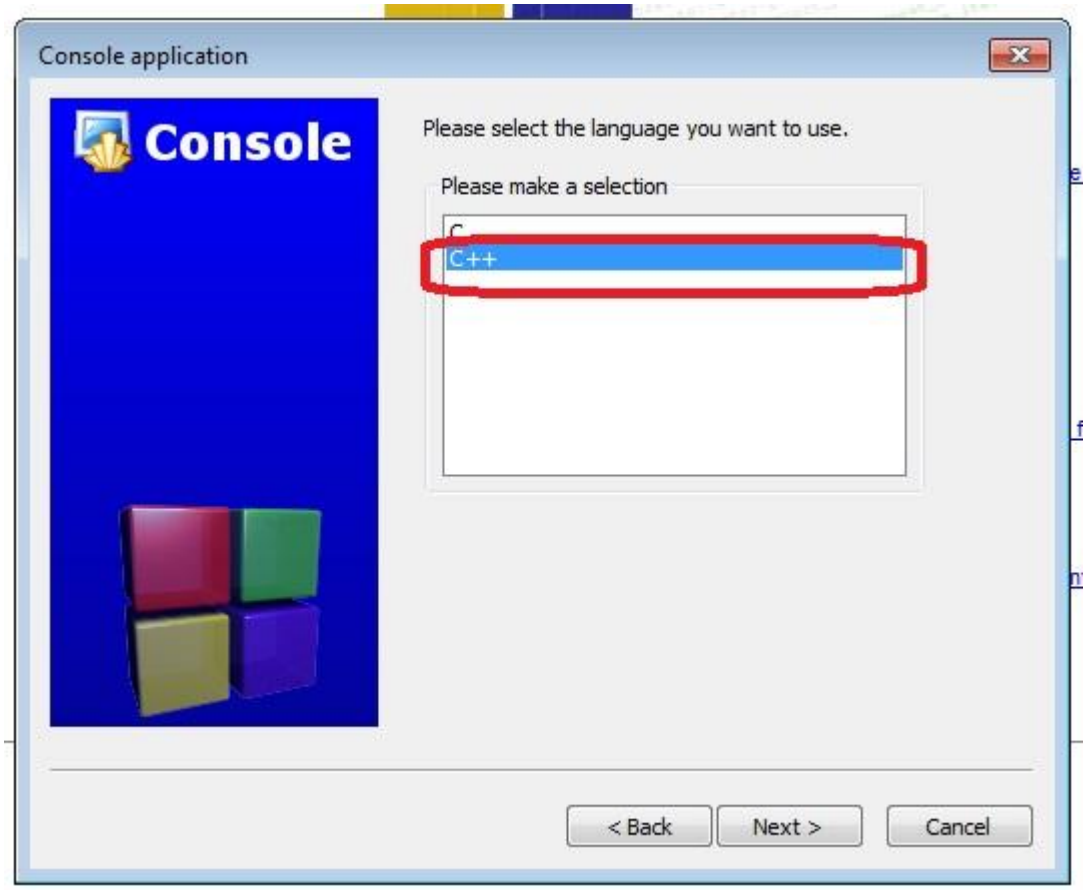
NTL – vytvorenie programu v prostredí Code::Blocks s NTL

Vytvoríme obyčajnú konzolovú aplikáciu využívajúcu NTL



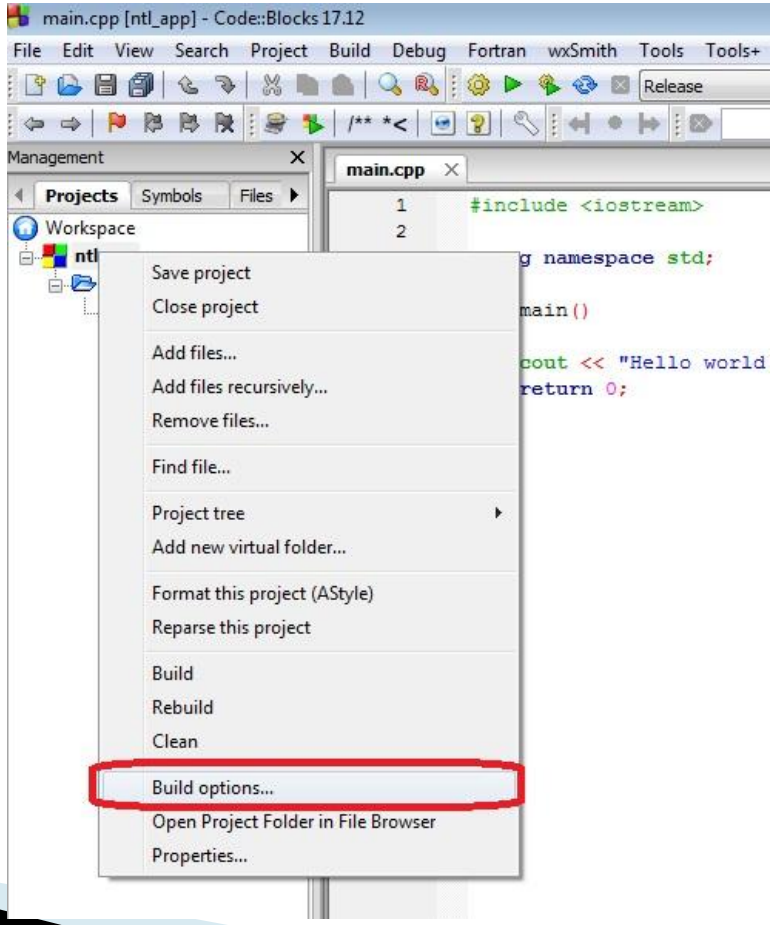
NTL – vytvorenie programu v prostredí Code::Blocks s NTL

Ako jazyk zvolíme C++



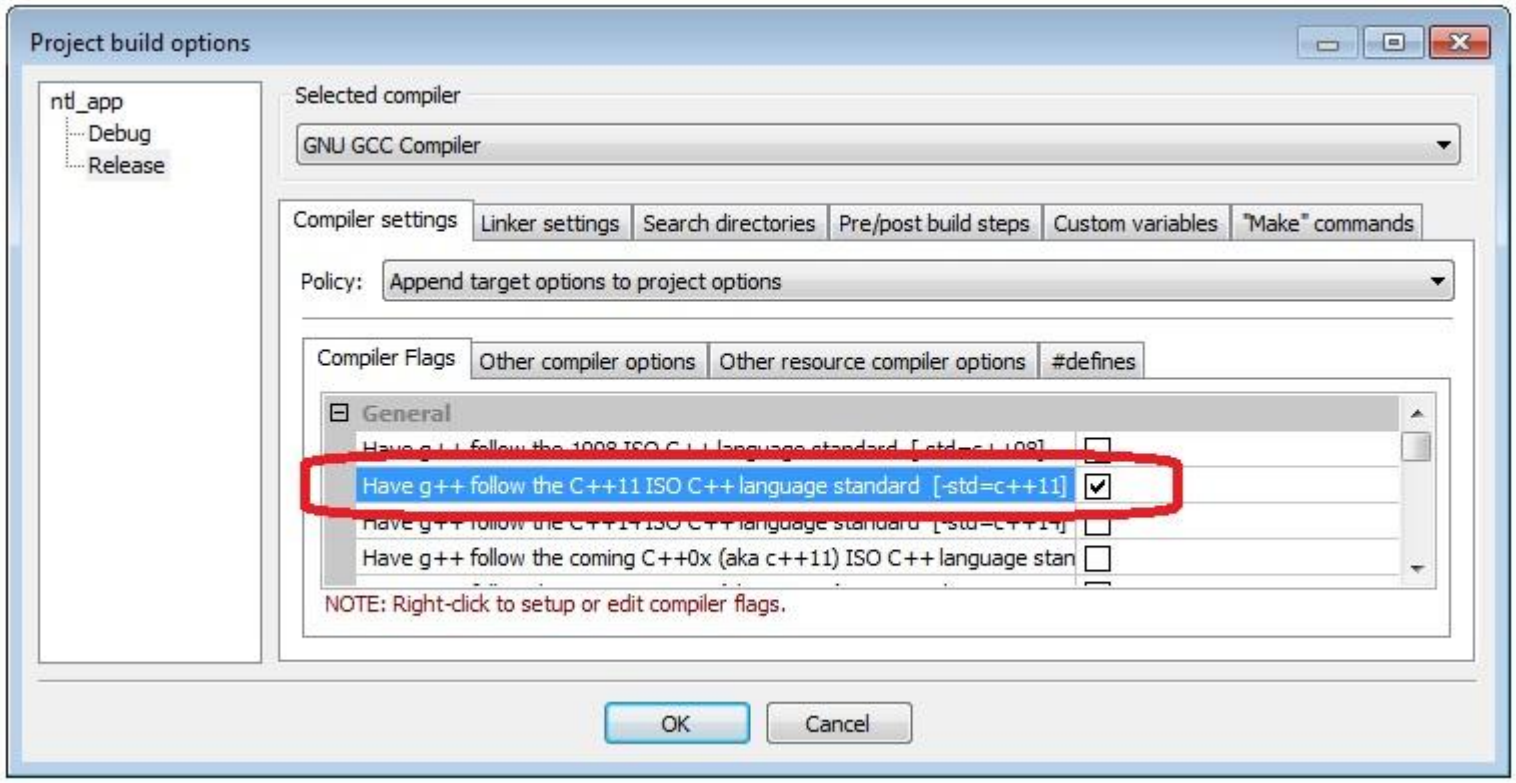
NTL – vytvorenie programu v prostredí Code::Blocks s NTL

Rovnakým spôsobom ako pri vytváraní statickej knižnice zmeníme „Build options...“ nového projektu (konzolovej aplikácie)



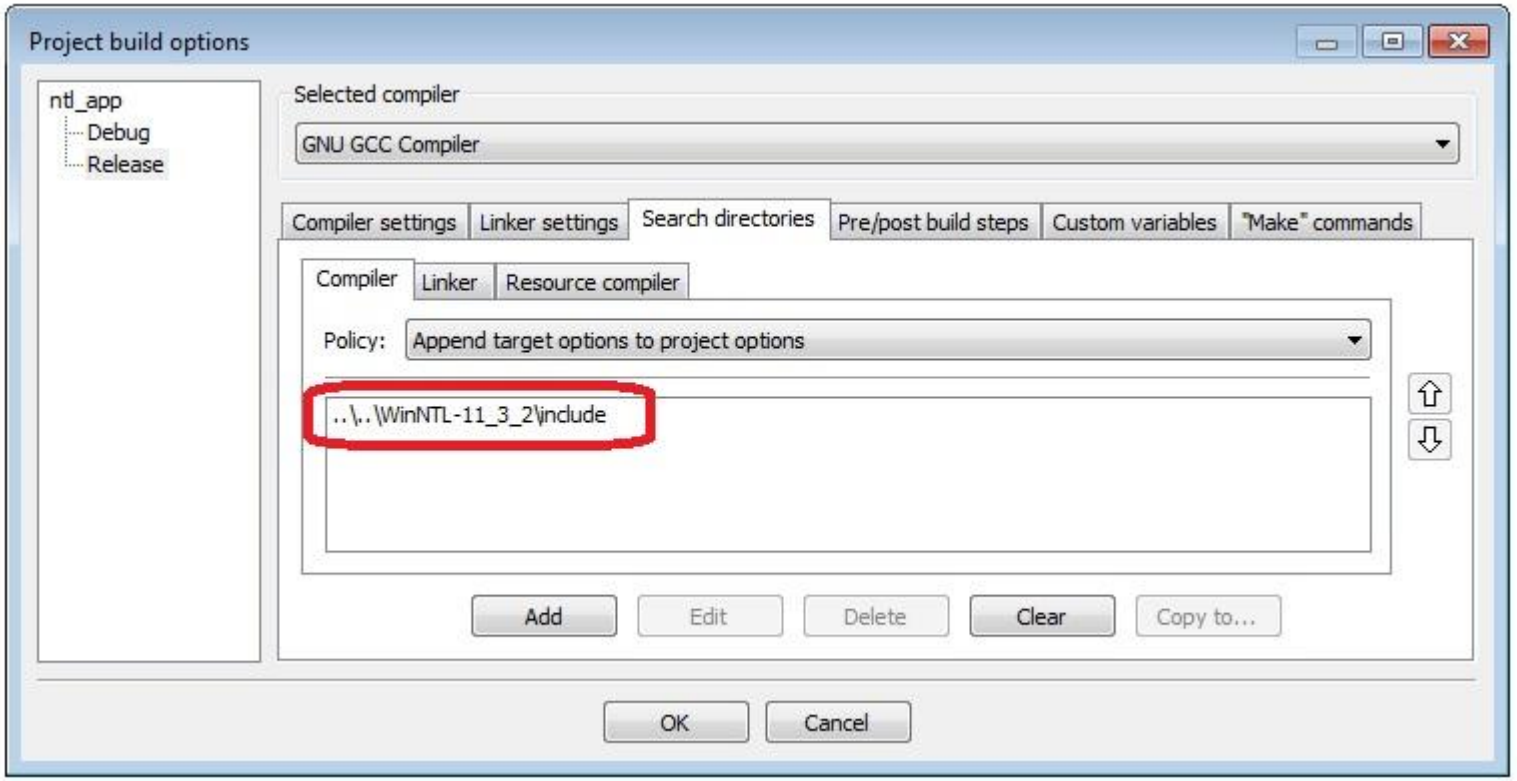
NTL – vytvorenie programu v prostredí Code::Blocks s NTL

Znovu – ak používame verziu NTL od 11.x.x, tak zmeníme nastavenie kompilátora na C++11 --- stačí pre verziu Release aplikácie (ak nechceme vytvárať Debug verziu) --- pre NTL 10.5.0 a menej vynecháme tento krok



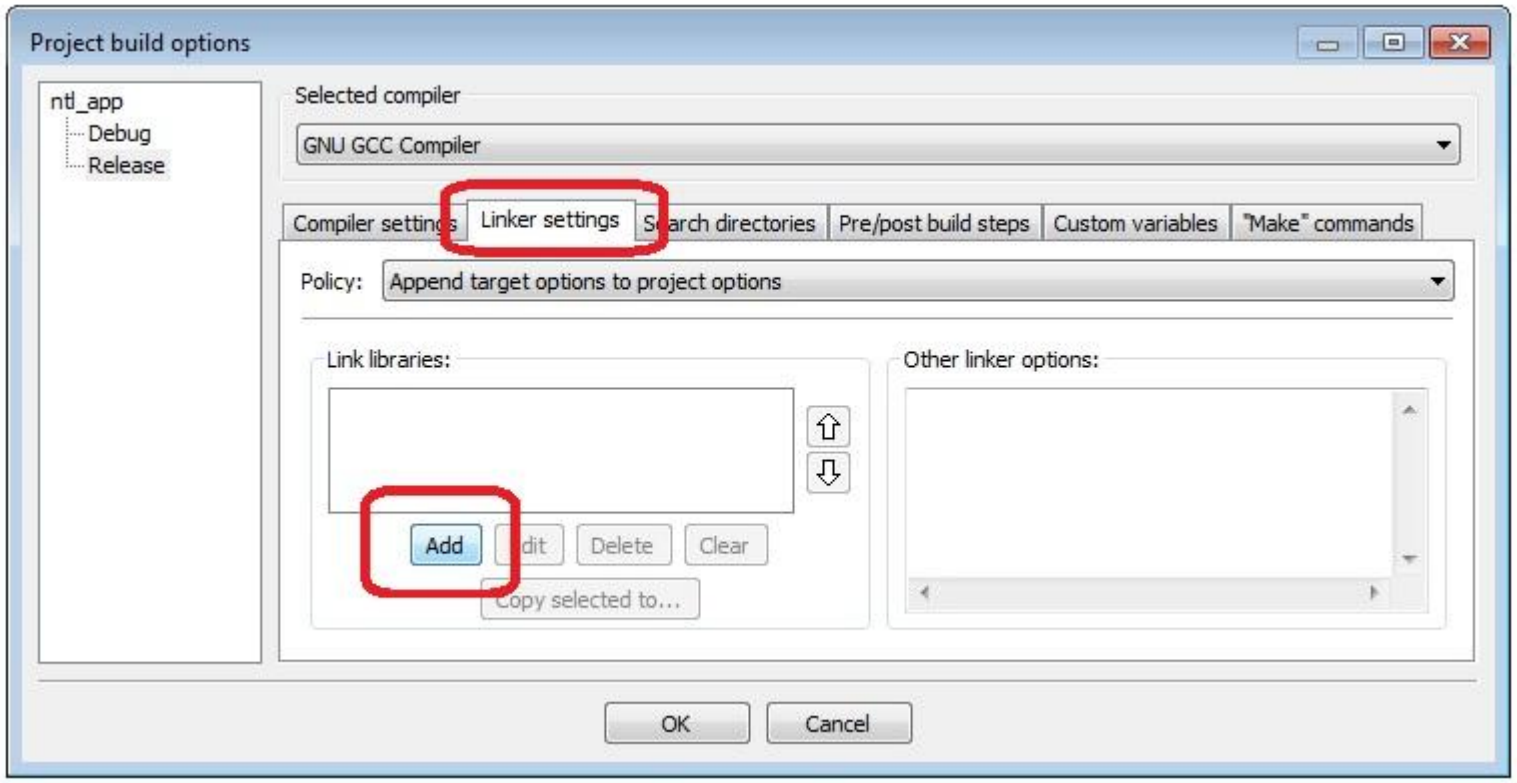
NTL – vytvorenie programu v prostredí Code::Blocks s NTL

Znovu je nutné nastaviť cestu aj k hlavičkovým súborom --- nezabudnúť na to, že musí končit „\include“



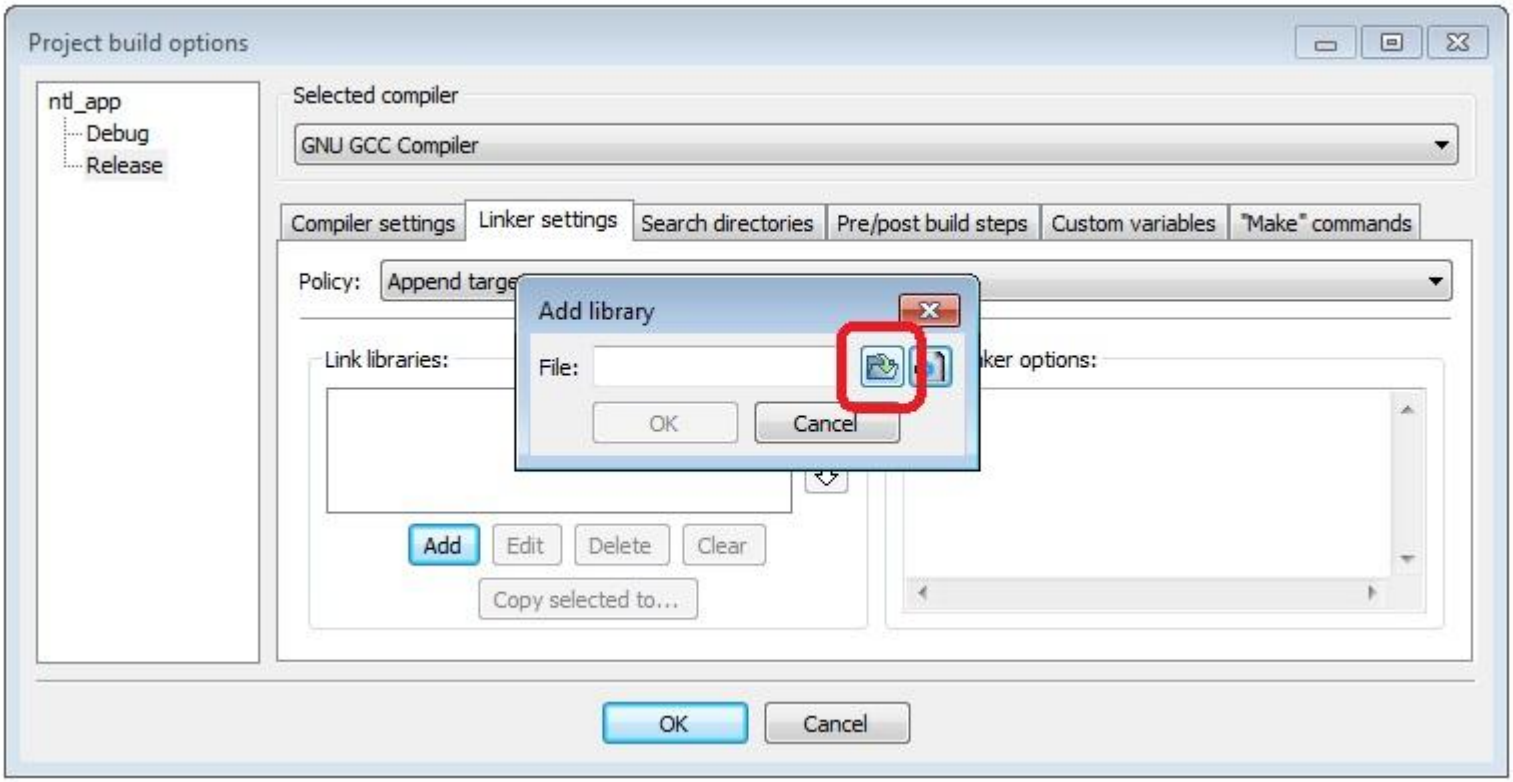
NTL – vytvorenie programu v prostredí Code::Blocks s NTL

Navyše, musíme teraz nastaviť cestu k statickej knižnici, ktorú sme vytvorili!



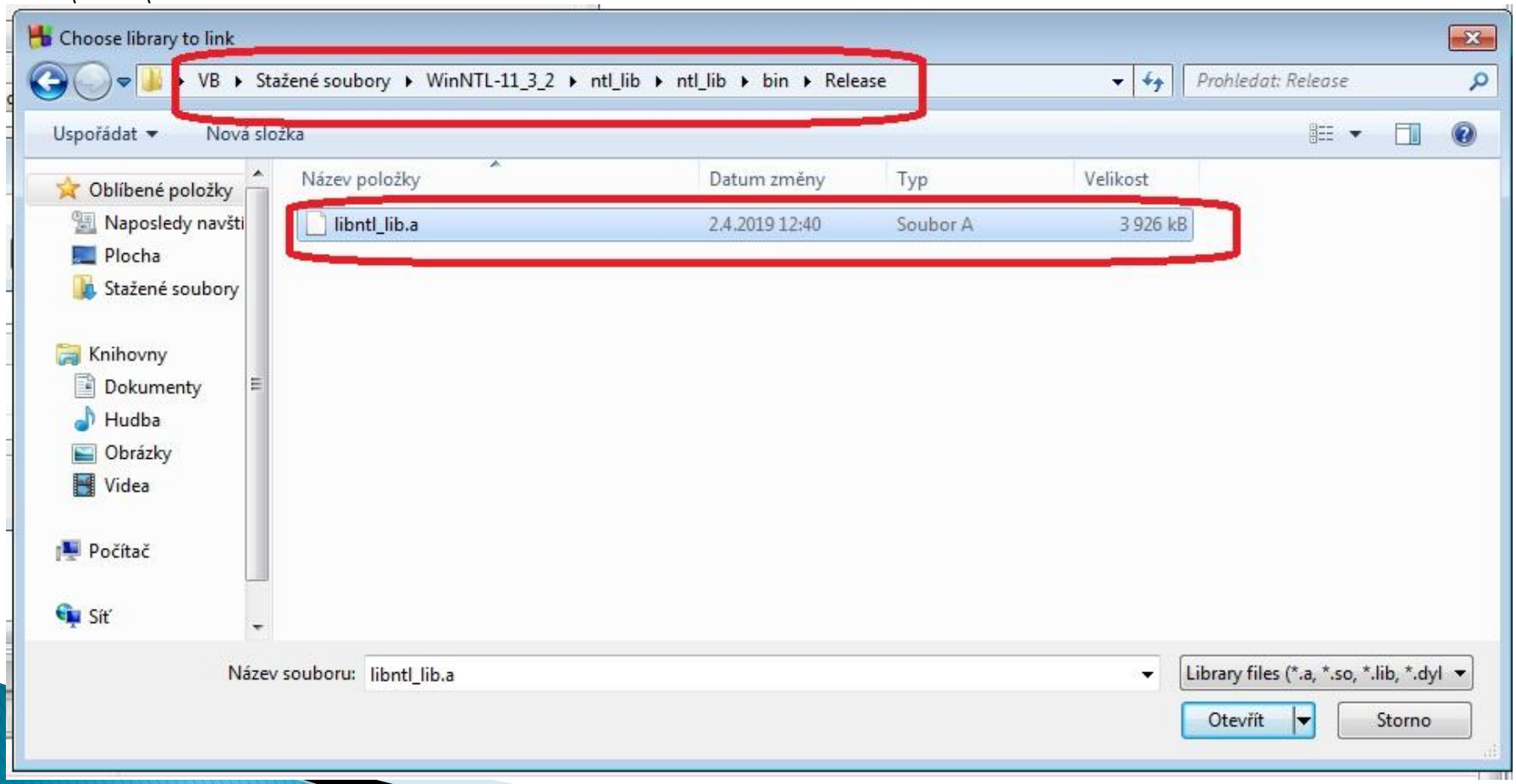
NTL – vytvorenie programu v prostredí Code::Blocks s NTL

Navyše, musíme teraz nastaviť cestu k statickej knižnici, ktorú sme vytvorili!



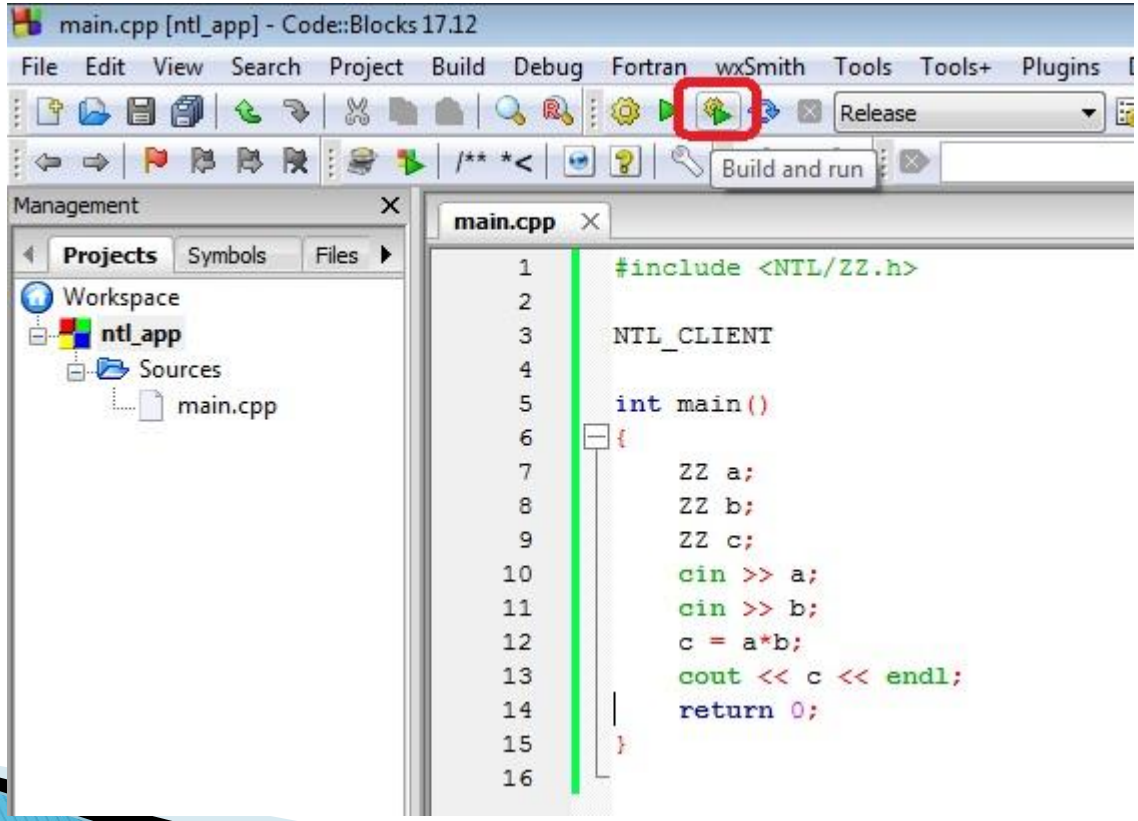
NTL – vytvorenie programu v prostredí Code::Blocks s NTL

Knižnica sa nachádza v adresári s projektom statickej knižnice v podadresári \bin\Release



NTL – vytvorenie programu v prostredí Code::Blocks s NTL

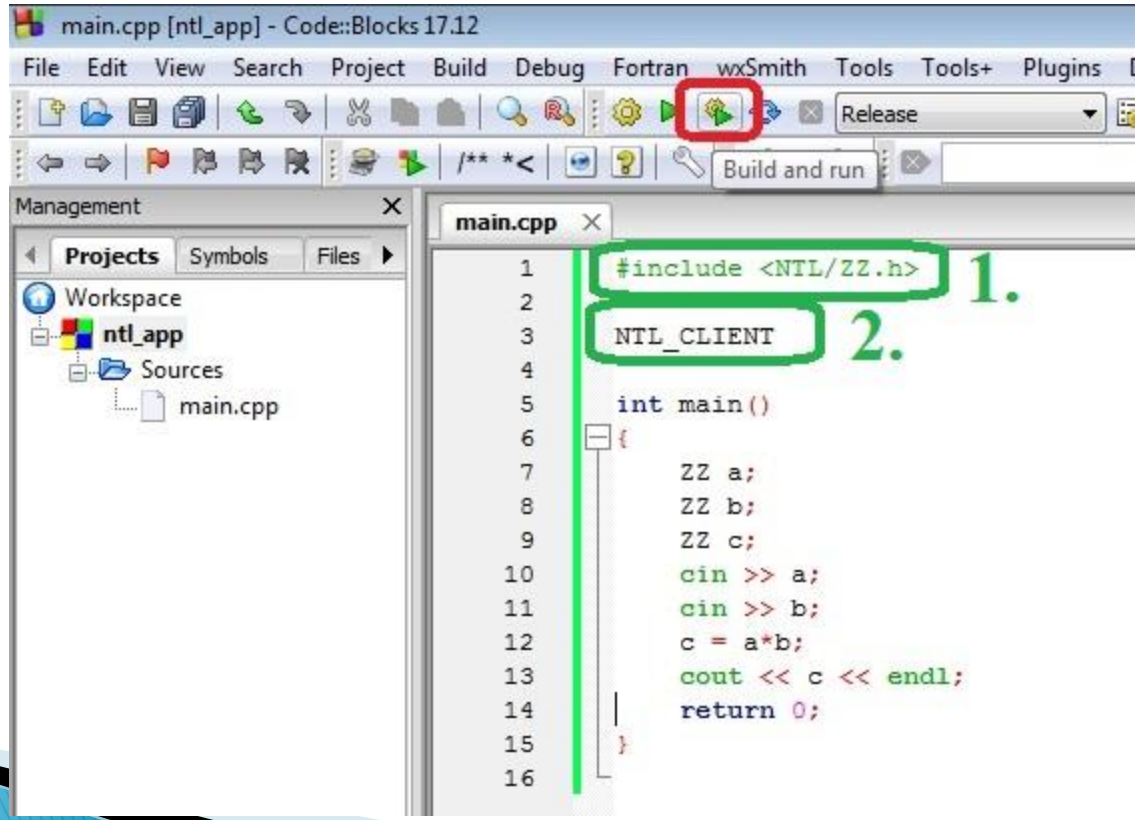
Teraz by už malo byť všetko nastavené pre použitie knižnice NTL. Napíšeme program v jazyku C++ s použitím NTL, skompilujeme a ~~podávame s opekanými zemiakmi...~~ spustíme výslednú aplikáciu.



NTL – vytvorenie programu v prostredí Code::Blocks s NTL

2 upozornenia:

1. Každý „#include“ musí začínať „NTL/“, napr. #include<NTL/ZZ.h>
   pre prístup k súboru ZZ.h (v ktorom sú funkcie a dátové typy pre celé čísla)
2. Nezabudnúť na makro NTL_CLIENT



NTL – ukážky

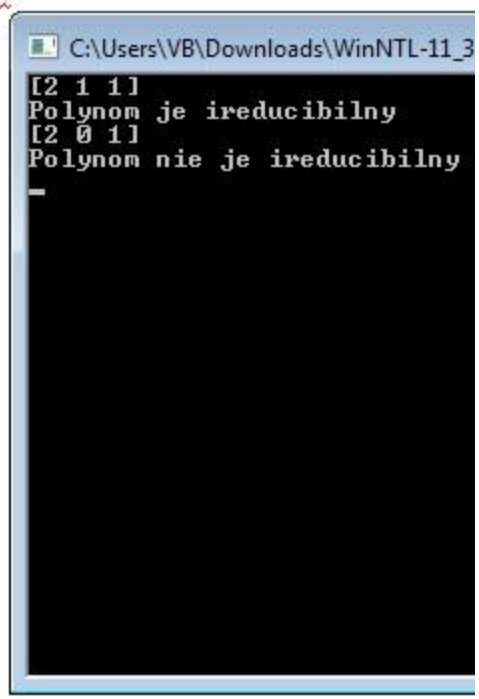
- Celé čísla– dátový typ ZZ
- Polynómy nad Z – dátový typ ZZX
- Okruh Z_p – dátový typ ZZ_p
- Polynómy nad Z_p – ZZ_pX
- Rozšírenie $Z_p/f(x)$ – ZZ_pE
- Polynómy nad $Z_p/f(x)$ – ZZ_pEX

- p nemusí byť prvočíslo, f(x) nemusí byť ireducibilný

```
#include<NTL/ZZ.h>   //pre datovy typ ZZ
NTL_CLIENT

int main()
{
    ZZ a,b,c;    //premenne typu ZZ (velke cele cislo)
    cin >> a;    //nacitanie a z klavesnice
    cin >> b;    //nacitanie b z klavesnice
    c = a*b;     //do c priradime sucin a*b
    cout << c << endl;  //vypis c na obrazovku

    return  0;
}
```

```
#include<NTL/ZZ.h>
#include<NTL/ZZ_p.h>  //pre datovy typ ZZ_p (cele cisla modulo cislo p)
NTL_CLIENT

int main()
{
   ZZ p;
   cin >> p;   //nacitanie p z klavesnice
   p = conv<ZZ>(3); //natvrdo zadane v kode a predchadzajuca hodnota sa prepise
                        // conv<T>(x) robi konverziu hodnoty x do datoveho typu T

   ZZ_p::init(p);   //vytvorenie okruhu modulo p pomocou funkcie init

   ZZ_p a,b,c;     //premenne a,b,c su teraz cele cislamodulo p

   cin >> a;   //nacitame a z klavesnice
   cin >> b;   //ak nacitame vacsie cislo ako p, automaticky sa zmoduluje

   c = a * b;  //vsetky operacie su modulo, aj nasobenie
   cout << c << endl;
   return  0;
}
```

```
#include<NTL/ZZ.h>
#include<NTL/ZZ_p.h>  //pre datovy typ ZZ_p (cele cisla modulo cislo p)
#include<NTL/ZZ_pX.h> //polynomy nad Z_p v jednej neurcitej
#include<NTL/ZZ_pXFactoring.h> //pre ireducibilne polynomy
NTL_CLIENT

int main()
{
    ZZ_p::init(conv<ZZ>(3));     //okruh Z_3

    ZZ_pX polynom;

    while(1)
    {
        cin >> polynom;

        if (IterIrredTest(polynom))
            cout << "Polynom je ireducibilny"<< endl;
        else
            cout << "Polynom nie je ireducibilny" << endl;
    }

    return  0;
}
```



[2 1 1] je NTL reprezentácia polynómu $2 + x + x^2$

[2 0 1] je NTL reprezentácia polynómu $2 + x^2$

Nad okruhom Z_3 je prvý polynóm ireducibilný, druhý nie!

```
#include<NTL/ZZ.h>
#include<NTL/ZZ_p.h>   //pre datovy typ ZZ_p (cele cisla modulo cislo p)
#include<NTL/ZZ_pX.h> //polynomy nad Z_p v jednej premennej
#include<NTL/ZZ_pXFactoring.h> //pre ireducibilne polynomy
#include<NTL/ZZ_pE.h>    //pre rozsirenie
NTL_CLIENT

int main()
{
    ZZ_p::init(conv<ZZ>(3));     //okruh Z_3

    ZZ_pX polynom;

    while(1)
    {
        cin >> polynom;

        if (IterIrredTest(polynom))
        {
            cout << "Polynom je ireducibilny"<< endl;
            break;
        }
        else
            cout << "Polynom nie je ireducibilny" << endl;
    }
    ZZ_pE::init(polynom);
    ZZ_pE prvok_ZZ_pE;

    random(prvok_ZZ_pE);
    cout << prvok_ZZ_pE << endl;

    return  0;
}
```

Vytvoríme rozšírenie Z_3 pomocou ireducibilného polynómu $2 + x + x^2$

Náhodne vygenerovaný prvok tohto okruhu zvyškov je $1 + 2*x$

Taktiež je v NTL implementovaný inkrementačný algoritmus na výpočet ČZV (CRT = Chinese Remainder Theorem)

```
#include<NTL/ZZ.h>
NTL_CLIENT

int main()
{
    ZZ A,P;
    A = conv<ZZ>(2);
    P = conv<ZZ>(3);

    CRT(A,P,5,7);

    cout << A << endl;
    cout << P << endl;
    cout << "-----------" << endl;
    CRT (A,P,1,5);
    cout << A << endl;
    cout << P << endl;


    return  0;
}
```

NTL – ukážky

- Pre prácu s okruhmi charakteristiky 2 existuje samostatný dátový typ GF2
- GF2, GF2X, GF2E, GF2EX

- Jeho použitie umožňuje efektívnejšiu prácu, ako pri použití ZZ_p, ktorý by sa inicializoval hodnotou 2.

```
#include<NTL/GF2.h>
#include<NTL/GF2X.h>
#include<NTL/GF2E.h>
#include<NTL/GF2EX.h>
NTL_CLIENT

int main()
{
   GF2X polynom;
   cin >> polynom;                    //nacitame polynom

   GF2E::init(polynom);               //okruh zvyskov nad GF(2) po deleni zadanym
                                      //polynomom
   GF2E x;                            //prvok rozsirenia

   cout << "Zadajte prvok x" << endl;
   cin >> x;          //napr [0 1] je prvok “x”, [0 1 1] je “x + x²”
   cout << "Vypis i-tych mocnin x" << endl;
   for (int i = 1; i < (1<<deg(polynom)) ; i++)
   {
      cout << i << ", " << power(x, i) << endl;
   }
   return  0;
}
```

NTL – ukážky

- Každý dátový typ reprezentujúci polynómy má implementované faktorizačné algoritmy
- ZZXFactoring, ZZ_pXFactoring, ZZ_pEXFactoring
- GF2XFactoring, GF2EXFactoring

NTL – náhodný generátor

- 1) Inicializácia NTL generátora – SetSeed
- 2) Generovanie potrebných náhodných hodnôt – GF2, GF2E, GF2X, ...
- 3) Generovanie náhodných **ireducibilných** polynómov je vo „Factoring“ hlavičkových súboroch – GF2XFactoring, atď.

NTL – ukážky

- Vektory, Vec<T>, vec_T
  - Vektory prvkov sú vo všeobecnosti deklarované ako Vec<T>, kde T je príslušný dátový typ, t.j. Vec<GF2>, Vec<ZZ>, Vec<GF2EX>, atď.
  - Rovnako funguje aj konvencia vec_T (pozor na veľké/malé písmená – Vec<T> vs. vec_T)
- Vektory vektorov, Vec<Vec<T>>, vec_vec_T
  - vec_vec_GF2, vec_vec_ZZ, vec_vec_ZZ_p, atď.
- Matice, Mat<T>, mat_T
  - mat_GF2, mat_GF2E, mat_ZZ, atď.
  - Vznikajú konverziou vec_vec_T

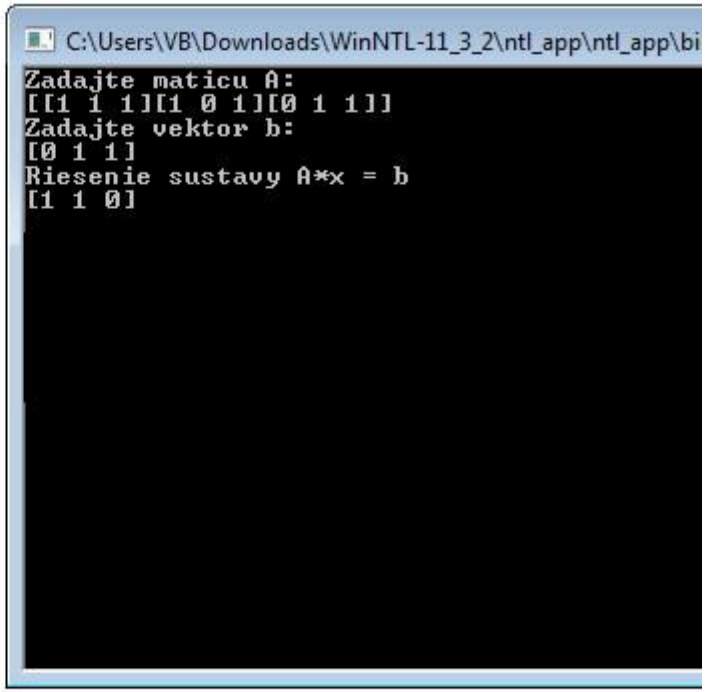
Sústava lineárnych rovníc

- Napr. nad GF(2)
- $x1 + x2 + x3 = 0$
- $x1 + x3 = 1$
- $x2 + x3 = 1$

- Pre GF2, GF2E rieši NTL sústavy **x*A = b**, resp. **A*x = b** podľa poradia argumentov metódy **solve.**

Sústava lineárnych rovníc

```
#include<NTL/mat_GF2.h>
#include<NTL/vec_vec_GF2.h>
#include<NTL/vec_GF2.h>
#include<NTL/GF2.h>
NTL_CLIENT

int main()
{
    vec_vec_GF2 vec_vec_A;
    vec_GF2 vec_b;
    mat_GF2 mat_A;
    GF2 det;
    vec_GF2 riesenie;

    cout << "Zadajte maticu A: " << endl;
    cin >> vec_vec_A;
    cout << "Zadajte vektor b: " << endl;
    cin >> vec_b;

    //konverzia na maticu
    conv(mat_A, vec_vec_A);

    //riesenie sustavy A*x = b
    solve(det, mat_A, riesenie, vec_b);

    cout << "Riesenie sustavy A*x = b" << endl;
    cout << riesenie << endl;

    return  0;
}
```



Dátový typ „pair“

- pair_GF2X_long, pair_GF2EX_long, atď.
- Využívajú sa pri faktorizácii, napr. faktorizácia polynómu

[1 1 0 0 0 0 1 1]

$1 + x + x^6 + x^7 = (1 + x + x^2)^2 (1+x)^3$

[[[1 1] 3] [[1 1 1] 2]]

T.j. faktor $(1+x)$ 3-násobný a $(1+x+x^2)$ 2-násobný

Dátový typ „pair“

```
#include<NTL/GF2X.h>
#include<NTL/GF2XFactoring.h>
NTL_CLIENT

int main()
{
    GF2X polynom;
    vec_pair_GF2X_long faktory;
    cout << "Zadajte polynom na faktorizaciu: " << endl;
    cin >> polynom;

    CanZass(faktory, polynom);
    cout << "Faktory zadaneho polynomu s nasobnostami: " << endl;
    cout << faktory << endl;


    return  0;
}
```

```
C:\Users\VB\Downloads\WinNTL-11_3_2\ntl_app\ntl_app\bin\Release
Zadajte polynom na faktorizaciu:
[1 1 0 0 0 0 1 1]
Faktory zadaneho polynomu s nasobnostami:
[[[1 1] 3] [[1 1 1] 2]]
```

Hint pre začiatočníkov

- Vo vlastných funkciách využívajúcich NTL odovzdávajte argumenty **vždy odkazom!**