

NTL Number Theory Library



Zdroje

- ▶ <https://libntl.org/>
 - Download, dokumentácia
- ▶ <https://uim.fei.stuba.sk/slovenska-prirucka-ntl/>
 - Dokumentácia v slovenčine

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
 - Čínska zvyšková veta, Jacobiho symbol
 - Faktorizácie polynómov nad \mathbb{Z} , $\text{GF}(p)$, $\text{GF}(p^k)$
 - Maticové algoritmy (GEM, inverzia, ...)
 - LLL algoritmus
 - Testovanie prvočíselnosti, generovanie ireducibilných polynómov, interpolácia, ...

NTL-inštalácia (Unix, Cygwin)

- ▶ <https://libntl.org/doc/tour-unix.html>
- ▶ V Unix-e odporúčam (tak ako autor knižnice) používať aj knižnice GMP a gf2x

NTL-inštalácia (Windows)

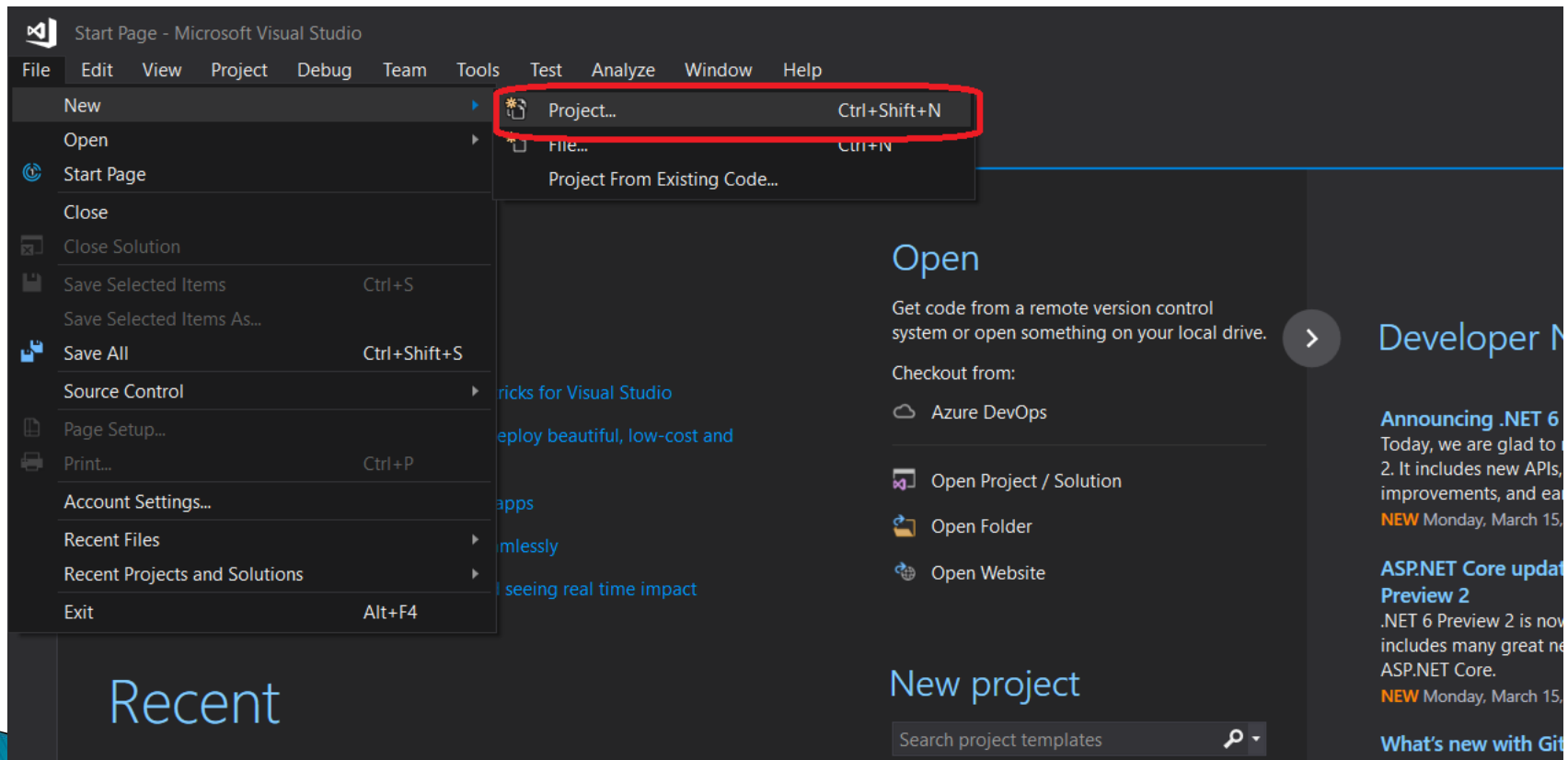
- ▶ <https://libntl.org/doc/tour-win.html>
- ▶ Ukážem kompiláciu NTL vo Visual Studiu (2017).
- ▶ Stiahneme si aktuálnu verziu NTL
- ▶ <https://libntl.org/download.html>
- ▶ (aktuálna verzia je 29.3.2021 v. 11.4.4)
- ▶ Stiahnutú knižnicu treba rozbaľiť do nejakého adresára.

NTL – inštalácia (Win)

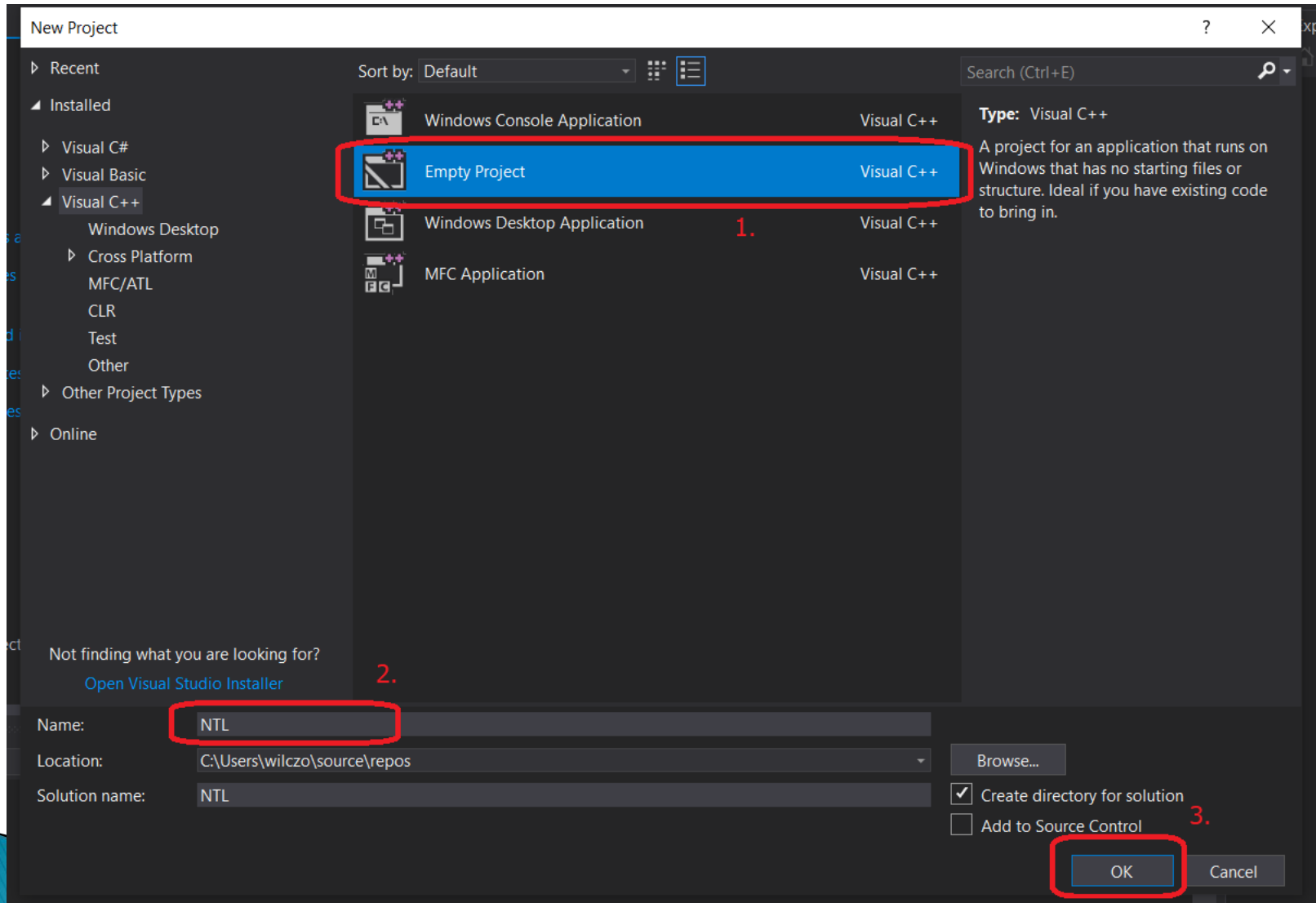
- ▶ Postup pre “inštaláciu” NTL:
 - 1) Stiahnuť
 - 2) Vytvoriť statickú knižnicu
 - 2.1) zo zdrojových kódov z adresára “src”
 - 2.2) Nastaviť cestu k hlavičkovým súborom v “include”
 - -----
 - 3) Prilinkovať statickú knižnicu k programu
 - 4) Taktiež nastaviť cestu k hlavičkovým súborom
 - 5) Pridať makro NTL_CLIENT

NTL – vytvorenie statickej knižnice vo Visual Studiu

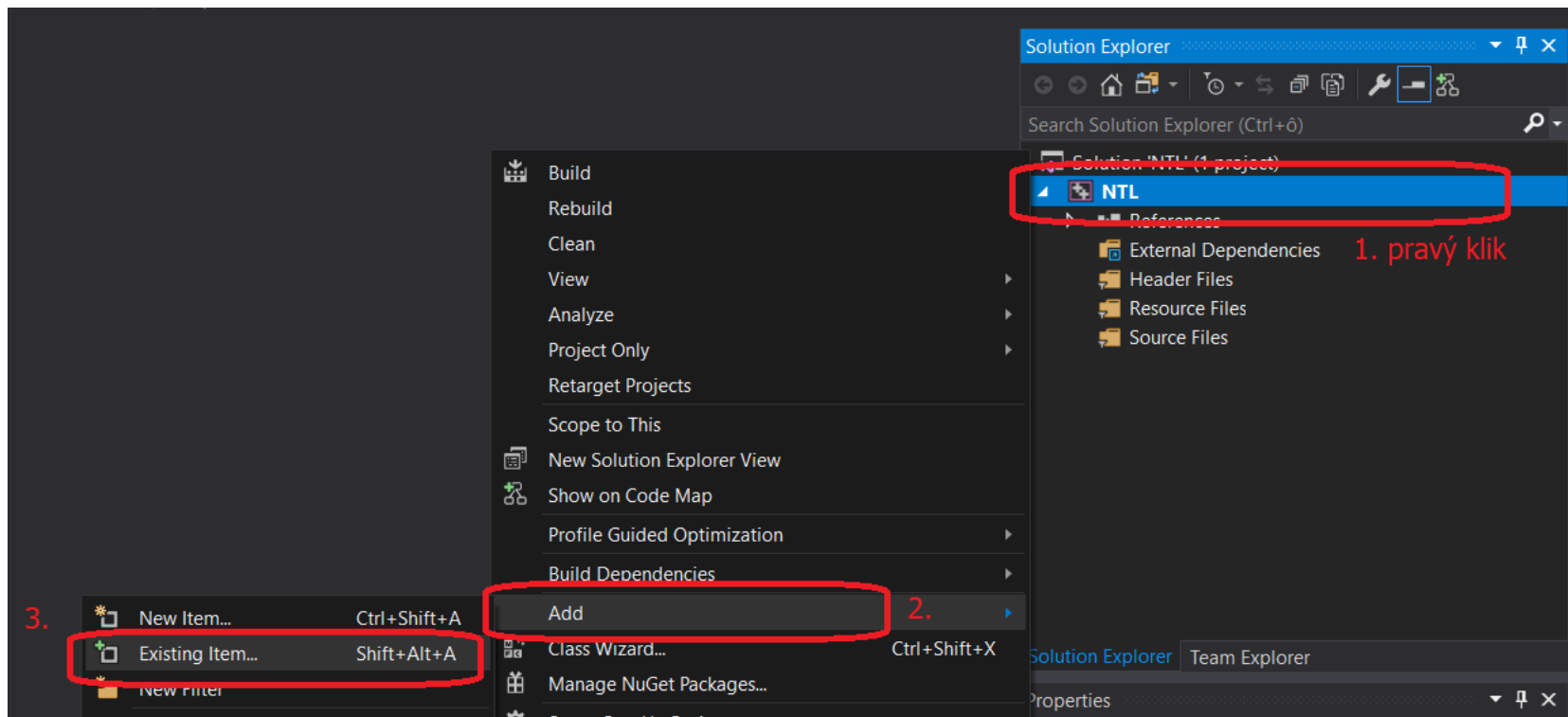
► Vytvoríme nový Empty Project



► Prázdny projekt pomenujeme napríklad NTL



- ▶ Do projektu přidáme všechny súbory z adresára “src” knižnice NTL



Add Existing Item - NTL



« Desktop > VisualStudio > WinNTL-11_4_4 > src



Search src

Organize ▾ New folder



Downloads

Documents

Pictures

2021

Knihy

NA DISKU

Programovanie

Microsoft Visual S

This PC

3D Objects

Desktop

Name

Date modified

Type

Size

BasicThreadPool

3/5/2021 5:49 PM

C++ Source File

ctools

3/5/2021 5:49 PM

C++ Source File

FacVec

3/5/2021 5:49 PM

C++ Source File

FFT

3/5/2021 5:49 PM

C++ Source File

fileio

3/5/2021 5:49 PM

C++ Source File

G_LLL_FP

3/5/2021 5:49 PM

C++ Source File

G_LLL_QP

3/5/2021 5:49 PM

C++ Source File

G_LLL_RR

3/5/2021 5:49 PM

C++ Source File

G_LLL_XD

3/5/2021 5:49 PM

C++ Source File

GetPID

3/5/2021 5:49 PM

C++ Source File

GetTime

3/5/2021 5:49 PM

C++ Source File

File name: "FFT" "fileio" "G_LLL_FP" "G_LLL_QP" "G_LLL_RR" "G_LLL_XD" "Get

All Files (*.*)

Add

Cancel

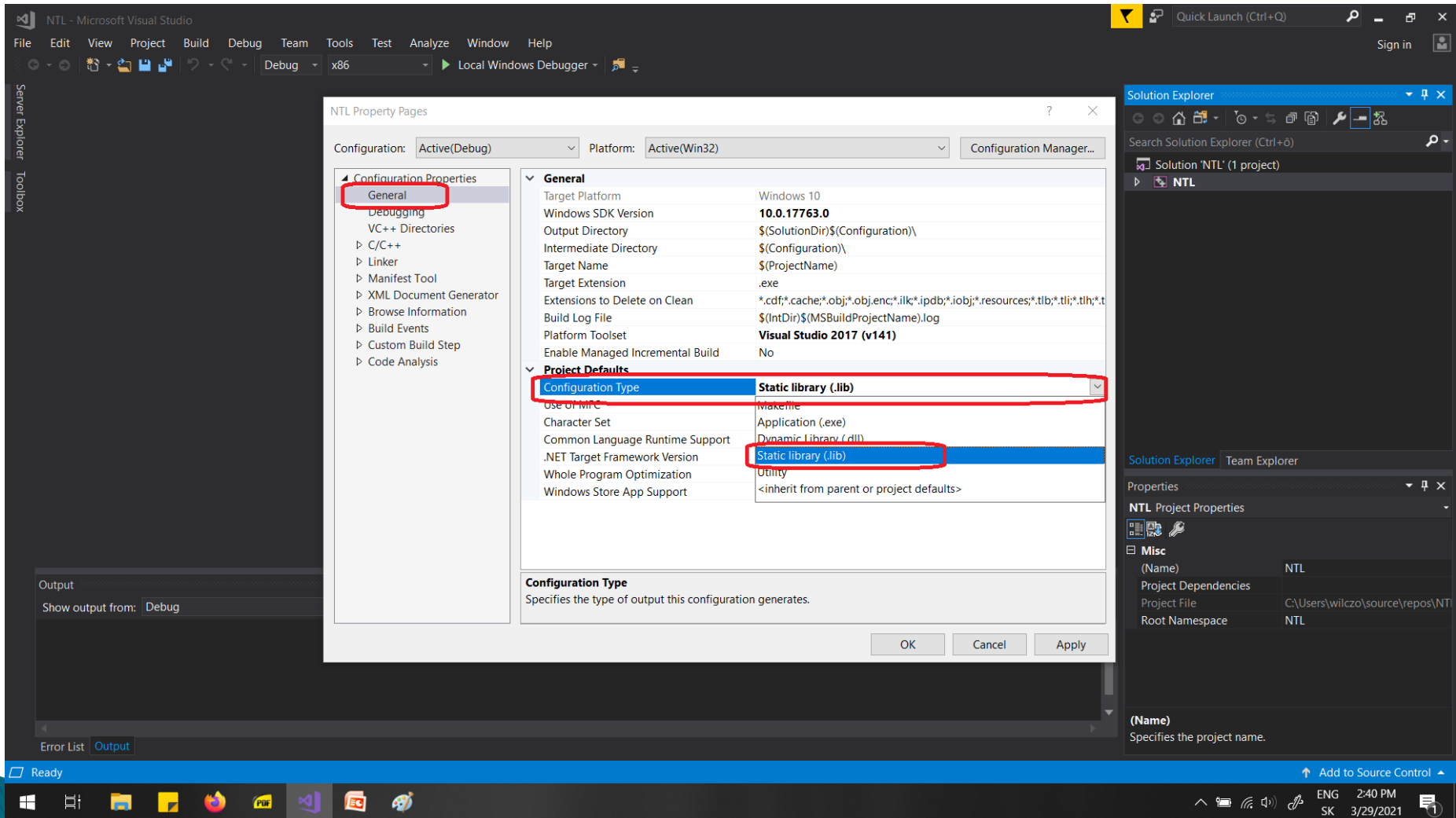
► Ďalej nastavíme vlastnosti projektu

The screenshot shows the Visual Studio interface with the Solution Explorer on the right. The project 'NTL' is selected and highlighted with a red box, with the text '1. pravý klik' (right click) written in red next to it. A context menu is open over the project, listing various actions. The 'Properties' option at the bottom of the menu is highlighted with a red box and labeled '2.'. The menu items include:

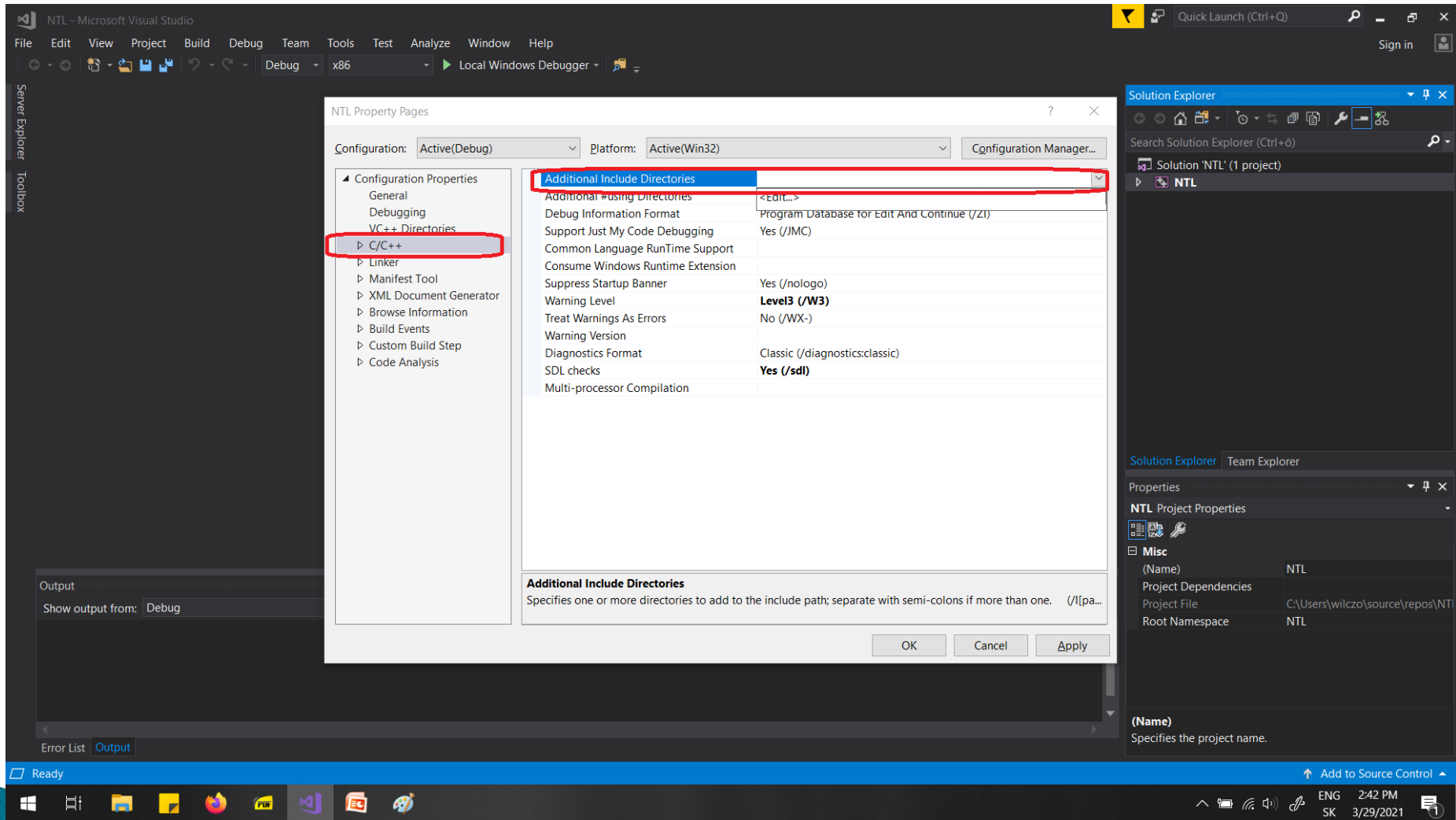
- Build
- Rebuild
- Clean
- View
- Analyze
- Project Only
- Retarget Projects
- Scope to This
- New Solution Explorer View
- Show on Code Map
- Profile Guided Optimization
- Build Dependencies
- Add
- Class Wizard... (Ctrl+Shift+X)
- Manage NuGet Packages...
- Set as StartUp Project
- Debug
- Source Control
- Cut (Ctrl+X)
- Paste (Ctrl+V)
- Remove (Del)
- Rename
- Unload Project
- Rescan Solution
- Display Browsing Database Errors
- Clear Browsing Database Errors
- Open Folder in File Explorer
- Properties (Alt+Enter)

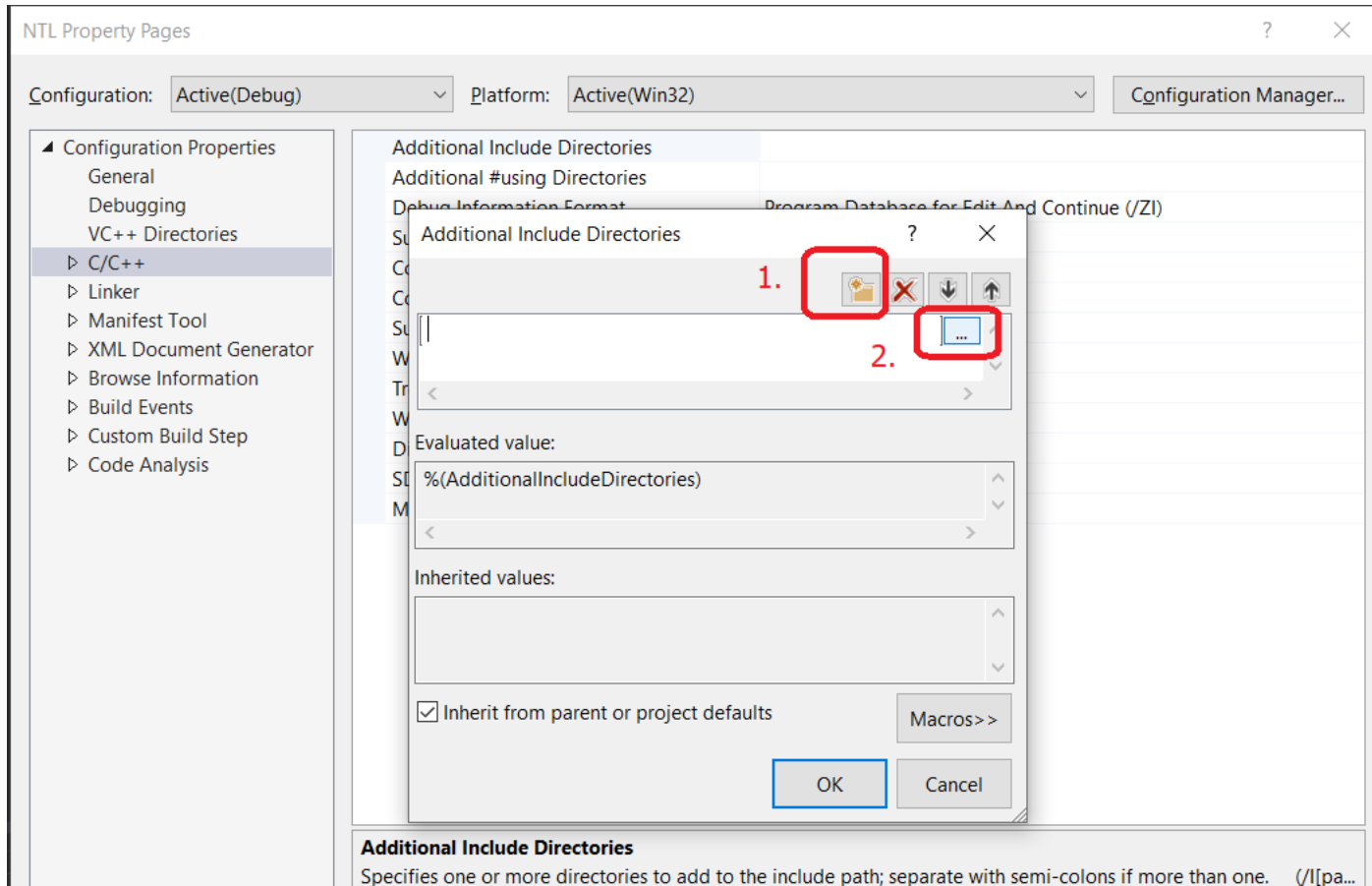
The Properties window on the right shows the 'NTL Project Properties' dialog, with the 'Misc' tab selected. The 'Name' property is set to 'NTL', and the 'Project File' is 'C:\Users\wilczo\source\repos\NTL'. The 'Root Namespace' is also 'NTL'.

► 1. nastavíme projekt ako Static Library (statická knižnica)

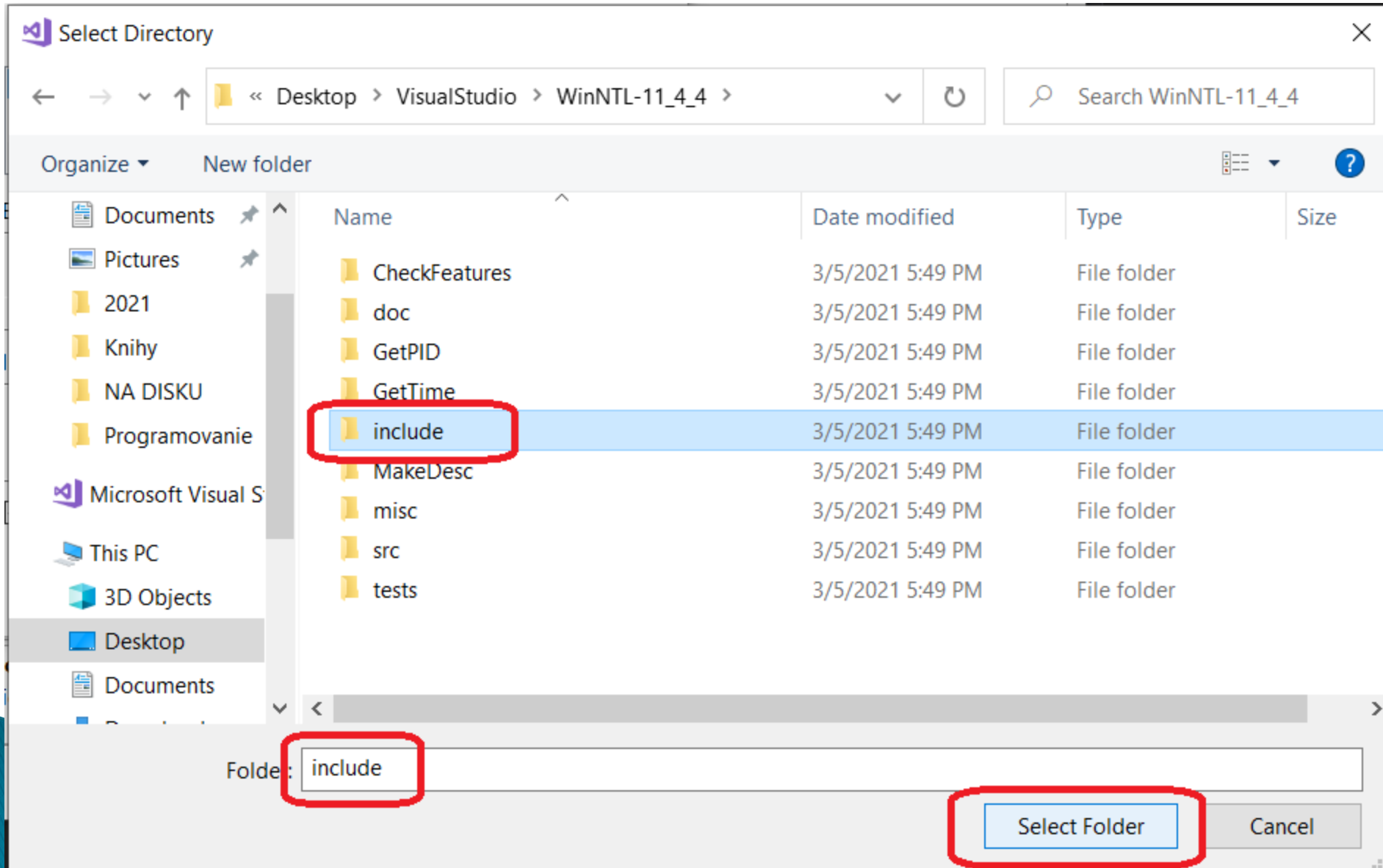


► 2. nastavíme cestu k hlavičkovým súborm v časti “Additional Include Directories”

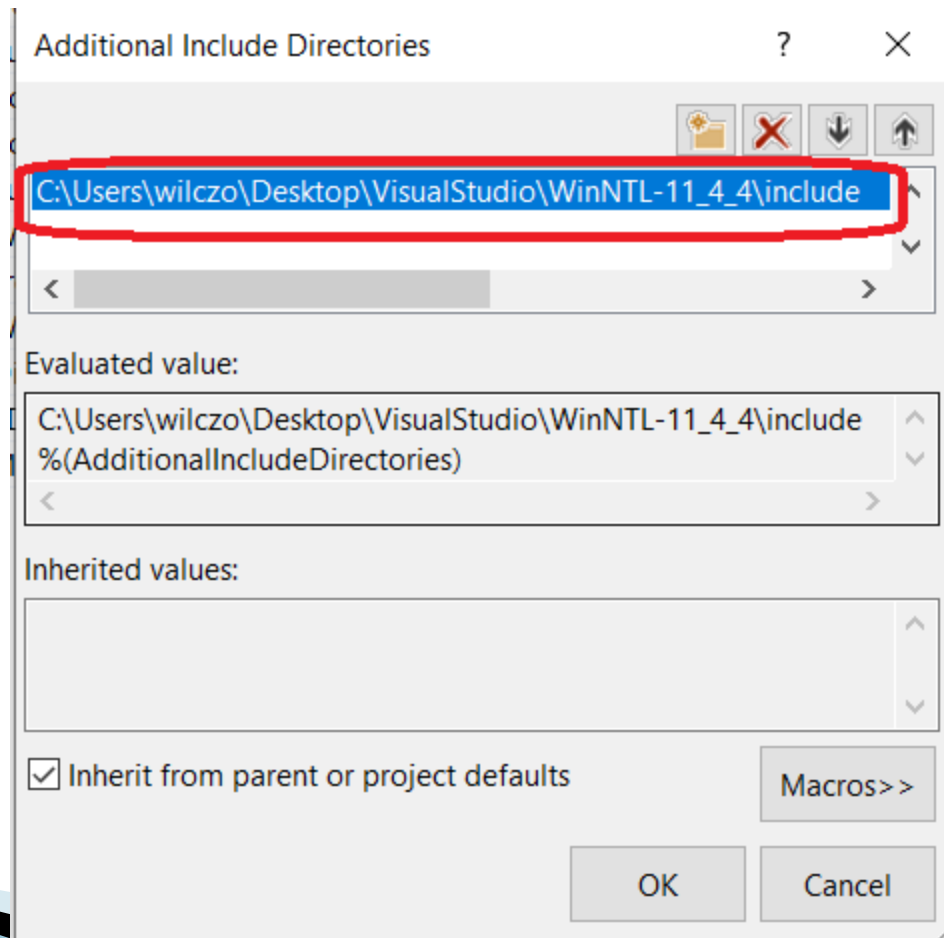




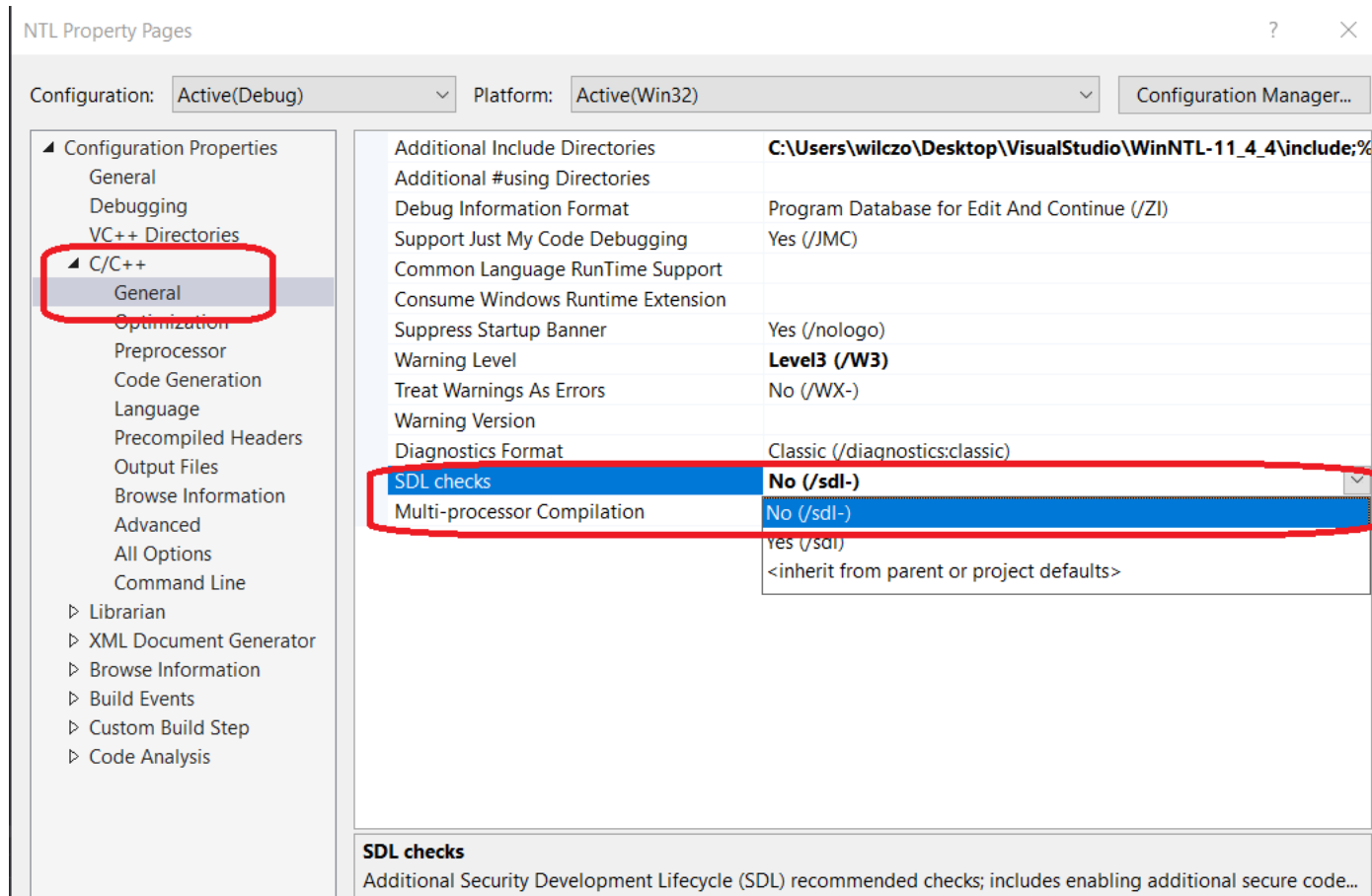
- ▶ Hlavičkové súbory sú v adresári “include” knižnice NTL.



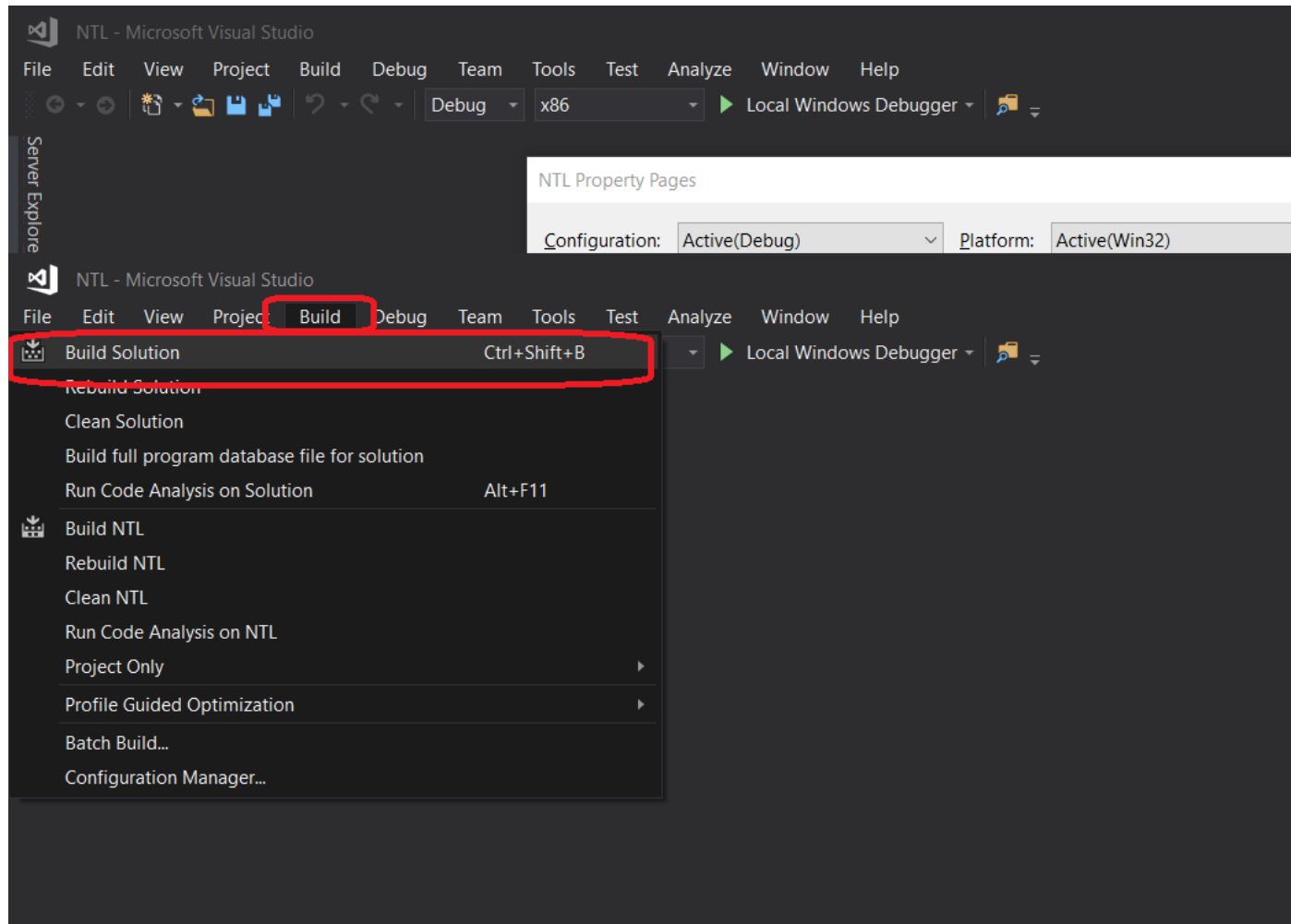
- Po korektnom nastavení cesty k “include”.
POZOR! Cesta musí končiť “\include”, nie
“\include\NTL” !!!



- ▶ Ešte je potrebné v nastavení projektu vypnúť tzv. SDL checks (inak bude preklad hlásiť chyby)

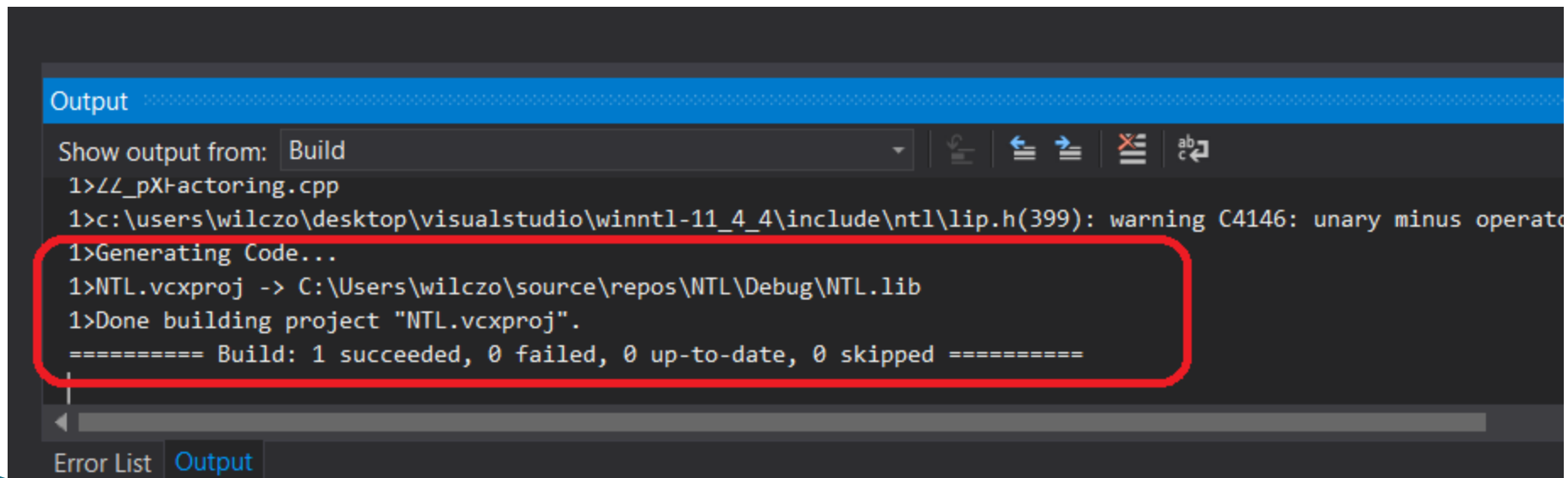


► Následne stačí už len spustiť “Build Solution”



Vytvorenie statickej knižnice

- ▶ Ak všetko zbehlo v poriadku, mali by sme získať súbor so statickou knižnicou – s koncovkou .lib.



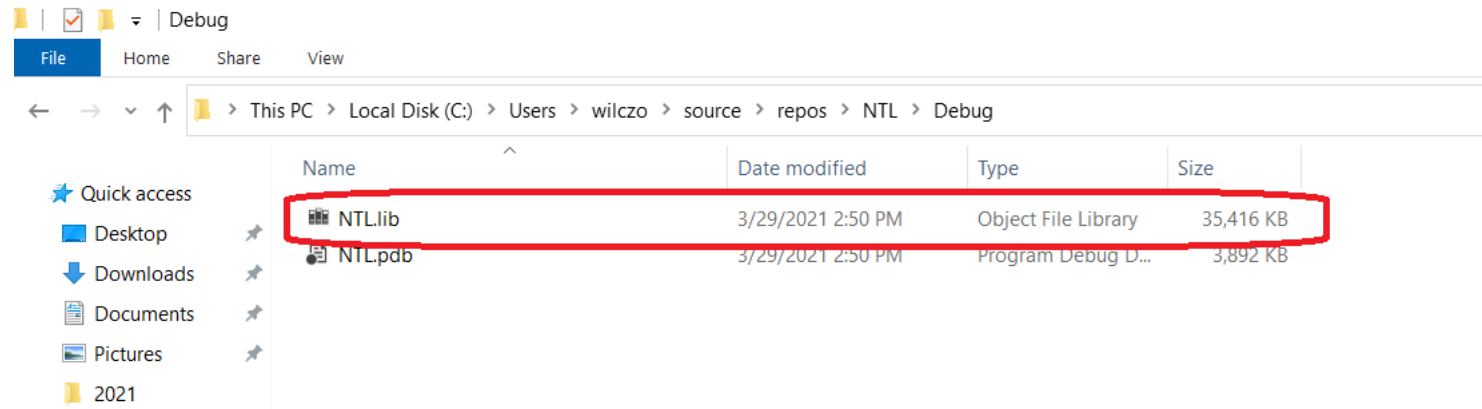
The screenshot shows the Visual Studio Output window with the 'Build' tab selected. The output text is as follows:

```
Output
Show output from: Build
1>\\_pX-factoring.cpp
1>c:\users\wilczo\desktop\visualstudio\winnt1-11_4_4\include\ntl\lip.h(399): warning C4146: unary minus operator
1>Generating Code...
1>NTL.vcxproj -> C:\Users\wilczo\source\repos\NTL\Debug\NTL.lib
1>Done building project "NTL.vcxproj".
===== Build: 1 succeeded, 0 failed, 0 up-to-date, 0 skipped =====
```

A red rectangular box highlights the lines from '1>Generating Code...' to '===== Build: 1 succeeded, 0 failed, 0 up-to-date, 0 skipped ====='. At the bottom of the window, there are tabs for 'Error List' and 'Output'.

Statická knižnica

- V mojom prípade súbor o veľkosti cca 35 MB.

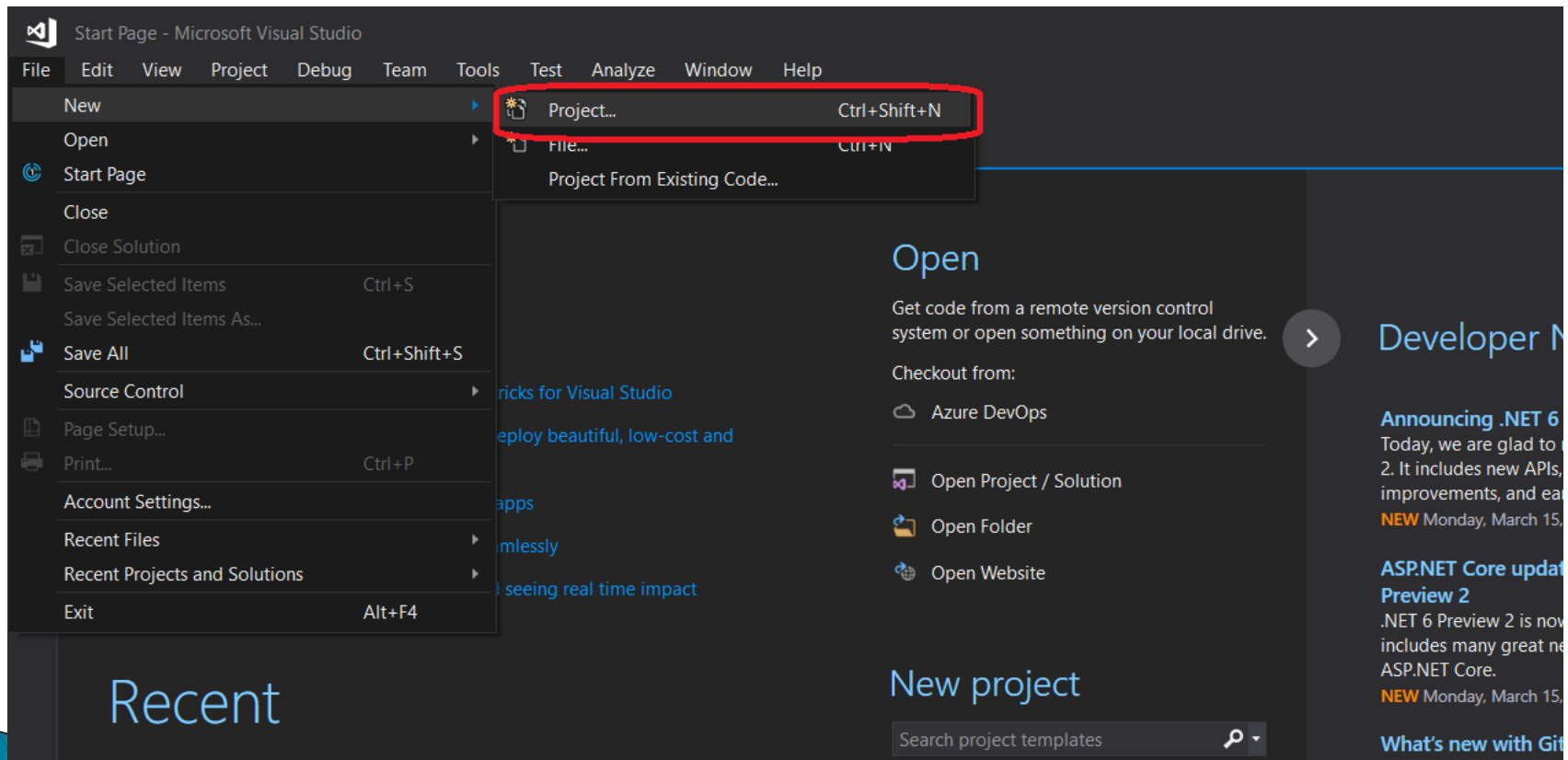


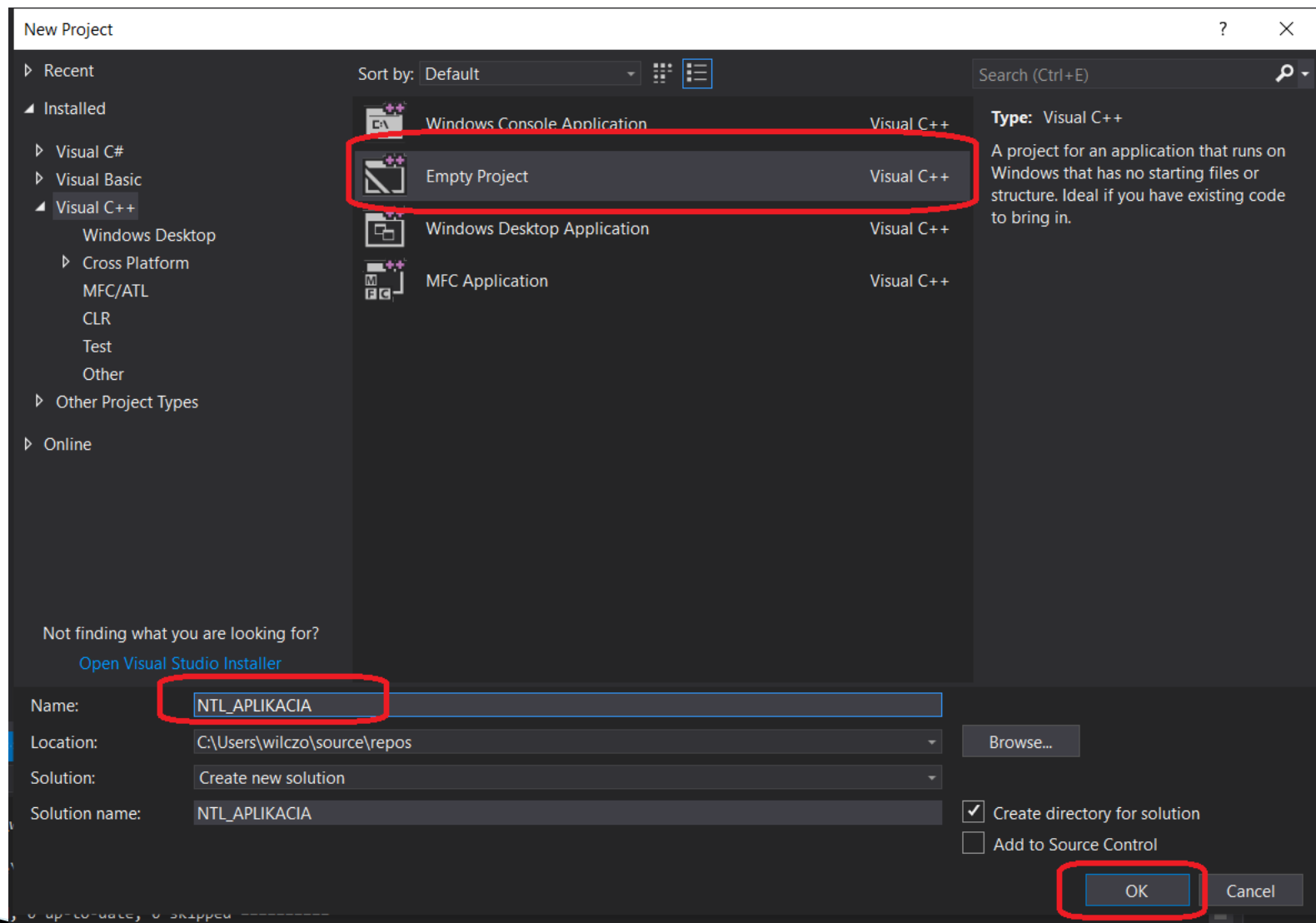
Vytvorenie aplikácie

- ▶ Po “vybuildovaní” statickej knižnice môžeme následne pristúpiť k programovaniu aplikácie využívajúcej knižnicu NTL.
- ▶ Pre využitie knižnice potrebujeme k samotnej aplikácii:
 - Pridať znovu NTL hlavičkové súbory
 - Pridať vytvorenú statickú knižnicu
 - Pridať do kódu makro NTL_CLIENT

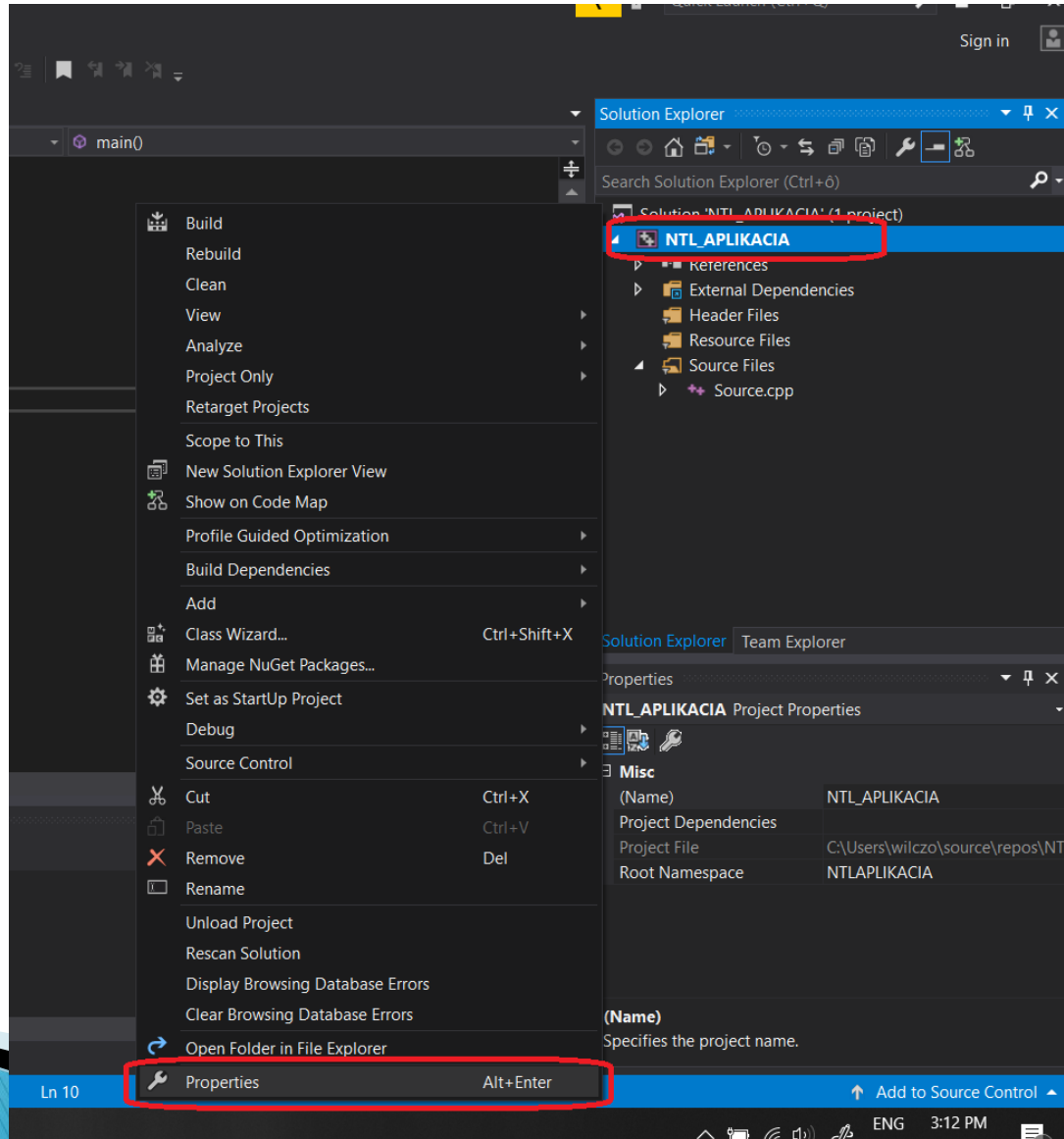
Vytvorenie aplikácie

- ▶ Vytvoríme znovu nový prázdny projekt

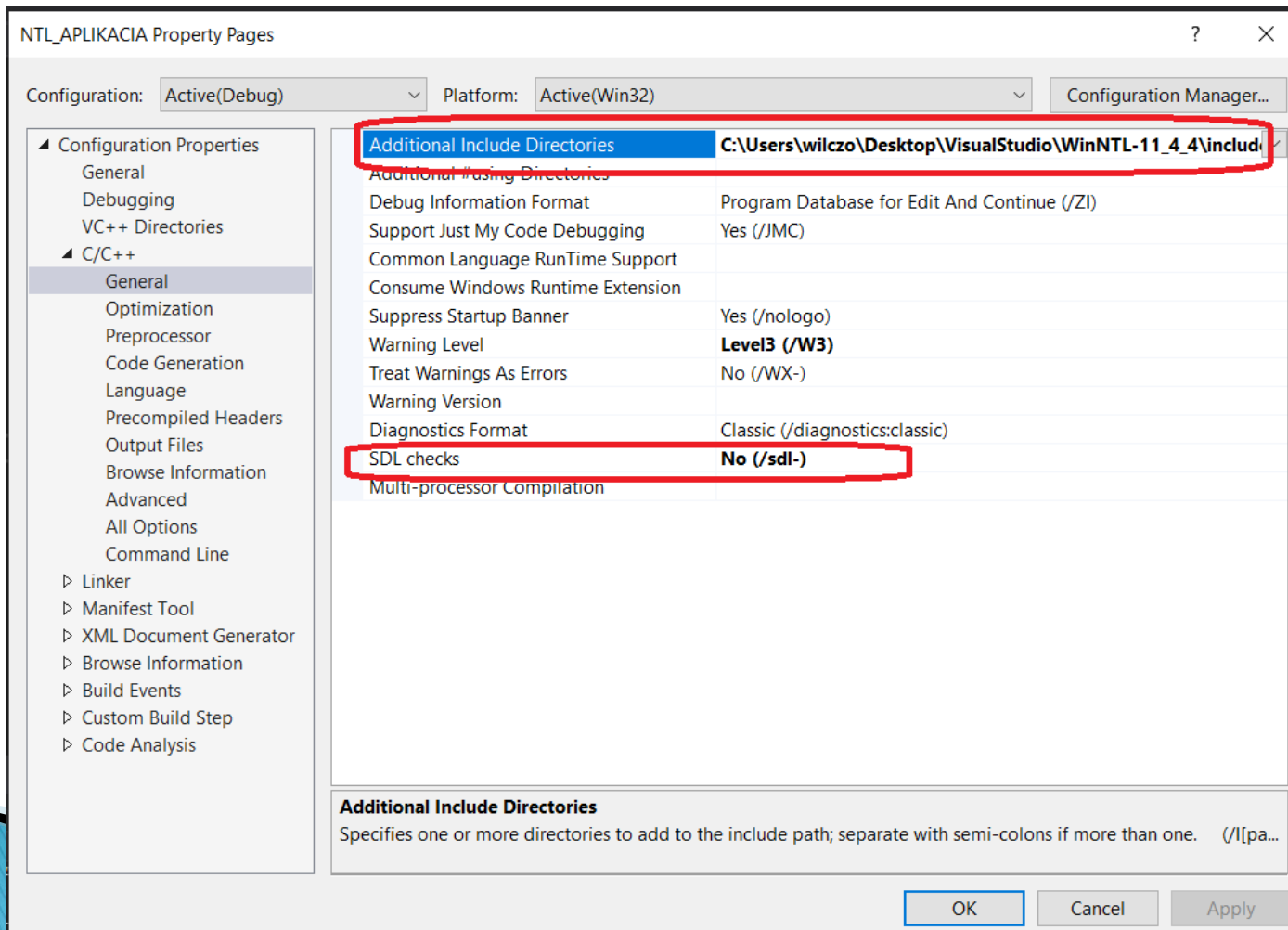




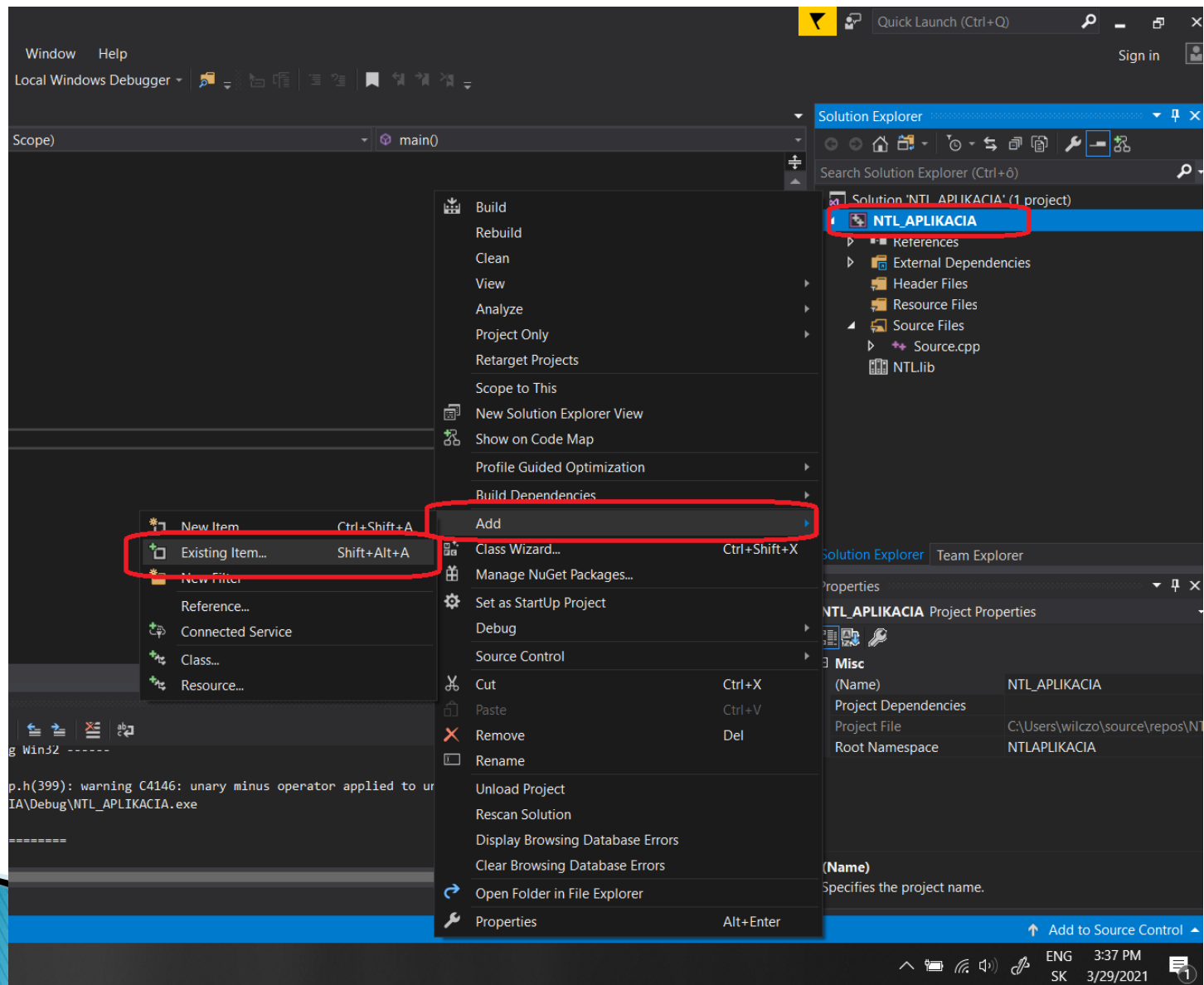
- ▶ Aj v aplikácii musíme nastaviť cestu k NTL hlavičkovým súborom



- ▶ Cestu k hlavičkovým súborom nastavíme rovnakým spôsobom, ako keď sme vytvárali statickú knižnicu.
- ▶ Taktiež vypneme znovu SDL checks.



- ▶ Následne k aplikácii pridáme vytvorenú statickú knižnicu:



Add Existing Item - NTL_APLIKACIA



> wilczo > source > repos > NTL > Debug



Search Debug

Organize ▾

New folder



★ Quick access

Desktop

Downloads

Documents

Pictures

2021

Knihy

NA DISKU

Programovanie

Microsoft Visual S

This PC

Name

Date modified

Type

Size

NTL.lib

3/29/2021 2:50 PM

Object File Library

35,416

NTL.pub

3/29/2021 2:50 PM

Program Debug D...

3,892

File name: NTL.lib

All Files (*.*)

Add

Cancel

- ▶ Teraz už môžeme písať C++ kód s využitím NTL knižnice
- ▶ Nezabudnúť na 2 veci:
 - a) Každý include modulu NTL musí začínať “NTL/”
 - Napríklad `#include<NTL/ZZ.h>` pre modul ZZ pre prácu s celými číslami
 - b) Vložiť makro `NTL_CLIENT`

Source.cpp* [icon] [X]

[icon] NTL_APLIKACIA

(Global Scope)

```
1  #include<iostream>
2  #include<NTL/ZZ.h>
3
4  NTL_CLIENT
5
6  int main()
7  {
8      ZZ a, b, c;
9      cin >> a;
10     cin >> b;
11     c = a + b;
12     cout << c << endl;
13
14     //funguje len na WIN - pozastavi aplikaciju na keypress
15     system("pause");
16     return 0;
17 }
```

NTL – ukážky

- ▶ Celé čísla – dátový typ ZZ
 - ▶ Okruh Z_p – dátový typ ZZ_p
 - ▶ Polynómy nad Z_p – ZZ_pX
 - ▶ Rozšírenie $Z_p / f(x)$ – ZZ_pE
-
- ▶ p nemusí byť prvočíslo, $f(x)$ nemusí byť ireducibilný

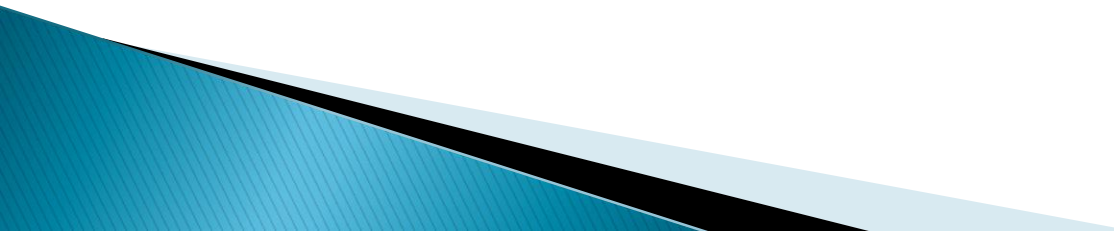
Program pre súčin celých čísiel $c = a * b$

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

```
NTL_CLIENT
```

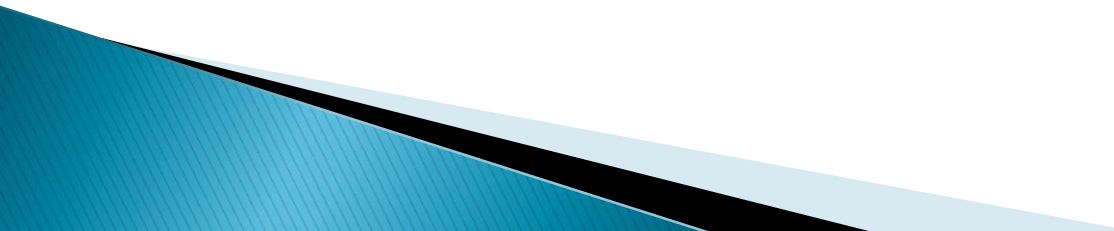
```
int main()
```

```
{  
    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 neucitatei
#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;
}

```

```

C:\Users\VB\Downloads\WinNTL-11_3
[2 1 1]
Polynom je ireducibilny
[2 0 1]
Polynom nie je ireducibilny
-

```

[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 zmennej
#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;
}

```

```

C:\Users\VB\Downloads\WinNTL-11_3_2\ntl_app\ntl
[2 1 1]
Polynom je ireducibilny
[1 2]

```

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

Náhodne vygenerovaný
prvok tohto okruhu
zvyškov je v našom
případe $1 + 2x$

- ▶ NTL má taktiež implementovaný inkrementačný algoritmus na výpočet čínskej zvyškovej vety (CRT)

```
#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;
}
```

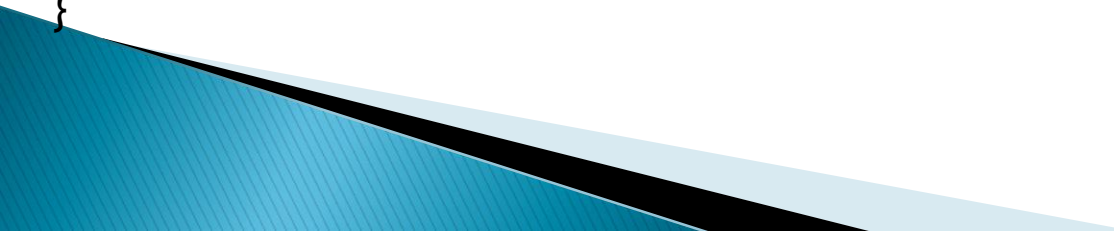


```
C:\Users\VB\...
5
21
-----
26
105
```

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 zvyiskov nad GF(2) po deleni zadany
                           //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 – náhodný generátor

- ▶ 1) Inicializácia náhodného generátora – SetSeed
- ▶ 2) Generovanie potrebných náhodných hodnôt – GF2, GF2E, GF2X, ...
- ▶ 3) Generovanie náhodných ireducibilných polynómov je vo “Factoring” hlavičkových súboroch – GF2XFactoring, atd’.

NTL – ukážky

- ▶ Vektory, $\text{Vec}\langle T \rangle$, vec_T
 - Vektory prvkov sú vo všeobecnosti deklarované ako $\text{Vec}\langle T \rangle$, kde T je príslušný dátový typ, t.j. $\text{Vec}\langle \text{GF2} \rangle$, $\text{Vec}\langle \text{ZZ} \rangle$, $\text{Vec}\langle \text{GF2EX} \rangle$, atď.
 - Rovnako funguje aj konvencia vec_T (pozor na veľké/malé písmená – $\text{Vec}\langle T \rangle$ vs vec_T)
- ▶ Vektory vektorov, $\text{Vec}\langle \text{Vec}\langle T \rangle \rangle$, vec_vec_T
 - vec_vec_GF2 , vec_vec_ZZ , vec_vec_ZZ_p , atď.
- ▶ Matice, $\text{Mat}\langle T \rangle$, mat_T
 - mat_GF2 , mat_GF2E , mat_ZZ , atď.
 - Dajú sa zostrojiť konverziou z vec_vec_T

Sústava lineárnych rovníc

- ▶ Napr nad $\text{GF}(2)$
 - ▶ $x_1 + x_2 + x_3 = 0$
 - ▶ $x_1 + x_3 = 1$
 - ▶ $x_2 + x_3 = 1$
-
- ▶ Pre $\text{GF}2$, $\text{GF}2E$ rieši NTL sústavy $\mathbf{x}^*\mathbf{A} = \mathbf{b}$, resp. $\mathbf{A}^*\mathbf{x} = \mathbf{b}$, podľa poradia argumentov metódy `solve`.

Sústava lineárnych rovníc

- ▶ Napríklad uvedená sústava

- $x_1 + x_2 + x_3 = 0$
- $x_1 + x_3 = 1$
- $x_2 + x_3 = 1$

- ▶ Je zapísateľná ako:

$$\begin{pmatrix} 1 & 1 & 1 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix}$$

- ▶ Na jej riešenie potrebujeme v NTL reprezentovať maticu koeficientov a vektor pravých strán

```

#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;
}

```

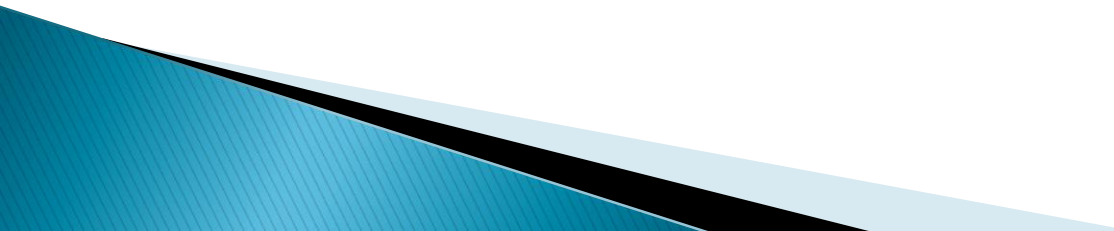
C:\Users\VB\Downloads\WinNTL-11_3_2\ntl_app\ntl_app\bi

```

Zadajte maticu A:
[[1 1 1][1 0 1][0 1 1]]
Zadajte vektor b:
[0 1 1]
Riesenie sustavy A*x = b
[1 1 0]

```

NTL – ukázky

- ▶ Každý datový typ reprezentující polynomy má implementované faktorizační algoritmy
 - ▶ ZZXFactoring, ZZ_pXFactoring, ZZ_pEXFactoring
 - ▶ GF2XFactoring, GF2EXFactoring
- 

Dátový typ “pair”

- ▶ pair_GF2X_long, pair_GF2EX_long, atd’.
- ▶ Využívajú sa pri faktorizácii, napr.
Faktorizácia polynómu
- ▶ Napr. nad GF(2)
[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 2] 2]]
T.j. 3-násobný faktor $(1 + x)$ a 2-násobný faktor $(1 + x + x^2)$

```

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

int main()
{
    GF2X polynom;
    vec_pair_GF2X_long faktory;
    cout << "Zadajte polynom na faktorizaciju: " << 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 faktorizaciju:
[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 odozdávajte argumenty **vždy odkazom!**
- ▶ Ak budete chcieť vyvíjať aplikáciu vo Visual Studio v móde “**Release**”, odporúčam mať vyrobenú aj verziu statickej knižnice v móde **Release**.