1. Špecifikácia

Aplikácia CBlake3 CLI je dostupná pre OS Windows a to pre 64-bitovú platformu (adresár ***CBlake3_cli_64bit***) ale aj 32-bitovú platformu (adresár ***CBlake3_cli_32bit***). Ide o takmer totožné adresáre, no adresár ***CBlake3_cli_32bit*** obsahuje niekoľko súborov navyše a generovanie DLL knižnice sa vykonáva pomocou špecifických prepínačov (viď kapitola 4). Opis súborov v tejto dokumentácii platí pre obidva adresáre (až na zmeny pri 32 bit platforme viď. kap. 4). Pre otestovanie aplikácie (bez potreby generovania DLL knižníc) je možné postupovať podľa návodu:

1) CLI aplikácia sa nachádza v adresári ***CBlake3_cli_64bit/testprogram/*** pre 64-bitovú platformu alebo ***CBlake3_cli_32bit/testprogram/*** pre 32-bitovú platformu (OS Winodws).

2) Aplikáciu je potrebné preložiť pomocou batch súboru ***preklad.bat*** (vhodné pre OS windows). Po úspešnom preklade sa vygeneruje spustiteľný súbor Blake3cli.exe.
*Poznámka: je nutné mať nainštalovaný GCC prekladač (uistite sa, že cesta (PATH) pre gcc prekladač je definovaná v systémových premenných) .

3) Spustenie aplikácie: ***Blake3cli.exe -f test3.txt -t 8*** (-f je prepínač pre cestu k súboru, -t je prepínač na počet vlákien | poradie prepínačov je možné zameniť).

4) Napísal som aj jednoduchý help výpis ktorý je možné spustiť napr. príkazom ***Blake3cli.exe -help***.

Verzie použitých nástrojov:

Nástroj	**Verzia**	**Zdroj**
Rust	1.61.0	https://www.rust-lang.org/tools/install
gcc (MinGW-W64 x86_64-ucrt-posix-seh, built by Brecht Sanders) 11.2.0	11.2.0	https://winlibs.com/
gcc (MinGW-W64 i686-ucrt-posix-dwarf, built by Brecht Sanders) 11.3.0	11.3.0	https://winlibs.com/
Blake3	1.3.1	https://github.com/BLAKE3-team/BLAKE3
cbindgen	0.20.0	https://docs.rs/cbindgen/latest/cbindgen/
rayon	1.5.1	https://docs.rs/rayon/latest/rayon/
libc	0.2	https://docs.rs/libc/latest/libc/

2. CBlake3 CLI

CBlake3 CLI je CLI aplikácia, ktorá využíva hašovaciu funkciu Blake3 na hašovanie súborov. Cblake3 berie na vstup 2 parametre a to cestu k súboru, ktorý chceme zašifrovať a počet vlákien, ktorý chceme pri šifrovani použiť. CBlake3 CLI vracia odtlačok zahašovaného súboru (hašovací kód) a čas, ktorý reprezentuje rýchlosť hašovania súboru. Výsledný čas predstavuje iba čas hašovania súboru (nezahŕňa otváranie a čítanie súboru alebo iné aspekty). Aplikácie je postavená na základe prepojenia optimalizovanej a paralizovateľnej implementácie hašovacej funkcie Blake3, ktorá je napísaná v jazyku Rust. Aplikáciu je možné využívať na OS Windows (pre 32 bit. Aj 64 bit. platformu).

2.1. Použitie

Prvý krok, ktorý je potrebné vykonať pre spustenie aplikácie je preložiť daný kód a vytvoriť spustiteľný súbor. Tento krok je možné vykonať spustením batch súboru ***preklad.bat***. Súbor preklad.bat sa nachádza v adresári ***testprogram/*** a jeho úlohov je preložiť súbor main.c s prilinkovanou DLL knižnicou (súbor rust-blake.dll).

Poznámka: DLL knižnica ***rustblake.dll,*** ktorá sa nachádza v adresári ***testprogram/*** slúži na preklad a vygenerovanie spustiteľného súboru pre 64-bit. platformu Windows (za predpokladu, že pracujeme na 64-bitovom OS s 64-bitovým gcc prekladacom). Ak chceme aplikáciu preložiť na 32-bitovom systéme, je potrebné nahradiť túto DLL knižnicu knižnicou z adresára ***testprogram/DLL_32/***.

Druhým krokom je samotné spustenie aplikácie. Aplikáciu spustíme v príkazovom riadku príkazom:

Blake3cli.exe -f *cesta_k_suboru.txt* ***-t*** *pocet_vlakien.*

Napríklad:

Blake3cli.exe -f test3.txt -t 8,

Kde prepínač -f (filepath) reprezentuje cestu k súboru a prepínač -t (threads) predsatvuje počet vlákien, ktoré chceme pri šifrovani využiť.

Akýmkoľvek neplatným prepínačom, napr. ***Blake3cli.exe -h,*** vypíšeme tzv. help menu.

3. Vytvorenie DLL knižnice

Pre využívanie Rust kódu v jazyku C je nutné vytvoriť interfejs medzi jazykom Rust a jazykom C. Ide o tzv. C-friendly API (application programming interface) vytvorenú v jazyku Rust. Týmto zabezpečíme „komunikáciu“ medzi jazykmi. V praxi sa takéto rozhranie pre komunikáciu medzi 2 jazykmi nazýva aj FFI (Foreign function interface), no pre jednoduchosť budeme používať termín API. Prvým krokom je vytvorenie Rust prostredia (príkaz: **cargo init** resp. **cargo init**). Po vykonaní jedného z uvedených príkazov získame prostredie pre vývoj nášho Rust projektu (Rust balíček). Prekladom Rust balíčka (zdrojových a konfiguračných kódov) sa vytvorí adresár ***target***, ktorý obsahuje nejaké prídavne súbory, ktoré vznikli pri preklade balíčka. V adresári ***target/release/*** sa nachádza aj DLL knižnica, ktorá bola taktiež vytvorená pri preklade. Pre využitie Rust API v jazyku C budeme potrebovať túto DLL knižnicu (ktorú prilinkujeme k C-kódu pri preklade, viď súbore preklad.bat) a hlavičkové súbory, ktoré je potrebné zahrnúť v C-kóde (viď. Zdrojový kód main.c). V jednotlivých podkapitolách detailne opíšeme proces tvorby DLL knižnice, automatizované generovanie hlavičkových súborov pre jazyk C a mapovanie Rustu s jazykom C (vytvorenie API).

3.1.Princíp mapovania a tvorby DLL knižnice

Vytvorenie DLL knižnice v jazyku Rust spočíva v niekoľkých krokoch:

1) Import knižníc (v Ruste označované ako crates).
2) Mapovanie Rustu s jazykom C (vytvorenie API).
3) Pripraviť prostredie (kódy + konfiguračné súbory) pre automatizované generovanie hlavičkových súborov
4) Preklad Rust balíčka

3.2.Import knižníc v Cargo.toml

Importovanie knižníc (crates) je realizované v súbore ***Cargo.toml***. V súbore *Cargo.toml* je možné vyplniť aj základne informácie o našom Rust balíčku ako názov, verziu apod.
Zdrojový kód (nižšie) zobrazuje súbor *Cargo.toml* aplikácie CBlake3 CLI.

```
[package]
name = "rustblake"
version = "0.1.0"
edition = "2021"

# nezabudni pridat build.rs
build = "build.rs"

[build-dependencies]
cbindgen = "0.20.0"

[lib]
name = "rustblake"
crate-type = ["cdylib"]

[dependencies]
blake3 = { version = "1.3.1", features = ["rayon"] }
time = "*"
rayon = "1.5.1"
libc = "0.2"
```

Zdrojový kód Cargo.toml: pre aplikáciu Cblake3 CLI.

Vysvetlivky zdrojového kódu *Cargo.toml:*

Sekcia [package]:
name – názov nášho Rust projektu (balíčka). Poznámka: Tento súbor Cargo.toml je modifikovanou kópiou súboru Cargo.toml, ktorý bol vytvorený pre prepojenie Pythonu a Rustu (viď. https://github.com/Alg0ritmus/CryptographyInPython_Thesis), preto je meno Rust projektu rovnaké aj v tomto projekte.
version – verzia daného projektu (v princípe nezáleží na označení – je to na nás)
edition – podobne ako pri verzii projektu (je to na nás)
build – dôležitá položka, ide o cestu k skriptu ***build.rs***, ktorý sa vykoná pri preklade (súvisí automatickým generovaním hlavičkových súborov)

Sekcia [build-dependencies]:
cbindgen – knižnica potrebná pre automatické vytvorenie hlavičkových súborov pre jazyk C

Sekcia [lib]:
name – názov nami vytvorenej knižnice, názov sa môže líšiť od názvu Rust balíčka no pre jednoduchosť volíme názor rovnaký
crate-type – dôležitá položka, hovorí o type našej knižnice. V tomto prípade volíme ["cdylib"] teda C dynamickú knižnicu (https://users.rust-lang.org/t/what-is-the-difference-between-dylib-and-cdylib/28847)

Sekcia [dependencies] – potrebné knižnice:
blake3 – importovanie knižnice (crate) hašovacej funkcie Blake3, ktorú použijeme na hašovanie súborov (https://github.com/BLAKE3-team/BLAKE3)
time – knižnica pre meranie casu/doby hašovania súboru
rayon – knižnica slúžiaca na konfiguráciu vlániek procesora (pre nastavenie počtu aktívnych vlákien pri hašovaní)
libc – knižnica, ktorá definuje niektoré dátové typy kompatibilné s jazykom C

3.3.Mapovanie

Mapovanie resp. vytvorenie API pre komunikáciu Rust kódu s jazykom C je realizované v súbore ***src/lib.rs***. Dátové štruktúry, funkcie a pod., ktoré chceme využívať (komunikovať s nimi z jazyka C) je nutné špeciálne „označiť“ (viď tabuľa nižšie).

#[no_mangle] pub const	funkcie
pub const pub static	globálne premenné
pub const	konštanty
#[repr(C)]	Dátové štruktúry

Viac na:
https://docs.rust-embedded.org/book/interoperability/rust-with-c.html
https://github.com/eqrion/cbindgen/blob/master/docs.md

Niektoré „označenia“ z tabuľky sú podporované iba za použitia knižnice **cbingen**.

Príklad:

```
#[no_mangle]
pub extern "C" fn rust_function() {
    // kód funkcie
}
```

Funkcie, ktoré budeme volať z jazyka C, musia brať na vstup parametre, ktorých dátový typ je kompatibilný s jazykom C. Rovnaká podmienka platí aj s návratovou hodnotou danej funkcie. Čiže výstup funkcie musí byť kompatibilný s jazykom C. Knižnica **cbingen** podporuje dátové typy: https://github.com/eqrion/cbindgen/blob/master/docs.md#supported-types. Niektoré dátové typy („libc“ types) sú dostupné v knižnici **libc**. V zdrojovom kóde ***src/lib.rs*** môžeme vidieť detailne okomentovanú funkciu spolu s dátovou štruktúrov.

TIP: Jazyk C nepozná dátový typ reťazec (string), preto práca s reťazcom nemusí byť triviálna. Na nasledujúcich linkách môžeme vidieť spôsob ktorým je možné pracovať s reťazcami. Prezentácia, kde je vysvetlené mapovanie Rust-C, spolu s mapovaním reťazcov: https://speakerdeck.com/dbrgn/calling-rust-from-c-and-java?slide=38, zdrojový kód k danej prezentácii: https://github.com/dbrgn/candidateparser/blob/master/candidateparser-ffi/src/lib.rs. Aplikácia CBlake3 využíva takéto spracovanie reťazca.

3.4.Automatizované generovanie hlavičkových súborov

Generovanie hlavičkových súborov je možné vykonať manuálne, no existuje komfortnejší spôsob. Pre automatizované generovanie hlavičkových súborov pre jazyk C vyžijeme knižnicu **cbindgen** (https://github.com/eqrion/cbindgen/blob/master/docs.md). Generovanie hlavičkových súborov pomocou knižnice **cbindgen** možno realizovať v 3 krokoch:

1) Inštalácia knižnice **cbindgen** príkazom: ***cargo install --force cbindgen***
2) V tzv. Manifeste (hlavný adresár, v ktorom sa nachádza súbor Cargo.toml) vytvoríme konfiguračný súbor ***cbingen.toml***, ktorý bude obsahovať iba kód: **language = "C"**
3) V adresári hlavnom adresári (Manifest) vytvoríme súbor ***build.rs*** (súbor build.rs môže byť vytvorený aj v inom adresári napr. src/, no cesta k súboru build.rs sa musí zhodovať z cestou ktorú sme zadali pre parameter „build“ v súbore cargo.toml). Ide o skript, ktorý je zodpovedný za zostavenie hlavičkových súborov (tento krok je možne vykonať aj z príkazového riadku, no kvôli zachovaniu komfortu využijeme krátky skript viď ***src/build.rs***).

3.5.Preklad Aplikácie + jej využitie v jazyku C

Preklad nášho Rust projektu je možné vykonať pomocou príkazu: ***cargo build --release***. Prekladom získame DLL knižnicu (knižnicu nájdeme v adresári **target/release/**), hlavičkové súbory dostupne v adresári ***testprogram/headers.***

Využitie Rust API v jazyku C je možné vykonať nasledovne:

1) Vygenerovanú DLL knižnicu (v našom prípade s názvom ***rustblake.dll***) nakopírujeme do adresára ***testprogram/***.
2) Volanie funkcie z jazyka C. Prvým krokom k zavolaniu Rust funkcie z jazyka C je pripísanie hlavičkového súboru, ktorý je vygenerovaný v adresári ***testprogram/headers/***.
3) Preklad C-kódu je nutné vykonať s prilinkovanou DLL knižnicou (viď. ***preklad.bat***)

4. Generovanie DLL pre 32 bit systemy

Generovanie DLL knižnice pre 32-bitový OS Windows je možné vykonať v niekoľkých krokoch:

1) Zvoliť tzv „target“ resp. cieľový systém (pre výpis všetkých podporovaných targetov slúži príkaz: ***rustc --print target-list***). Target musí odpovedať prekladaču, ktorý použijeme pri vygenerovaní DLL knižnice (máme tým na mysli prekladač pre jazyk C – keďže vytvárame Rust-C API).
2) Inštalácia daného toolchainu (jeden z targetov). Pre 32-bitovú platformu použijeme príkaz: ***rustup target add i686-pc-windows-gnu***
3) Stiahnutie MinGW gcc prekladača pre 32-bitovú Windows platformu: https://winlibs.com/.
4) Po stiahnutí MinGW prekladača pre jazyk C, vytvoríme v hlavnom adresári nášho Rust projektu adresár s názvom **.cargo**, v ktorom vytvoríme konfiguračný súbor ***config.toml***. V súbore ***.cargo/config.toml*** je nutné vytvoriť linker s naším 32-bitovým gcc prekladačom. To nám zabezpečí, že pri preklade Rust projektu využijeme práve 32bitový prekladač gcc, ktorý je potrebný pre vytvorenie DLL knižnice pre 32-bitovú platformu. Konfiguračný súbor by mohol vyzerať následovne:

```
[target.i686-pc-windows-gnu] // nazov tergetu (pre 32-bit MinGW)
linker = "C:\\mingw32\\bin\\i686-w64-mingw32-gcc" (nahradte cestu k suboru i686-w64-mingw32-gcc.exe -> adresar bin/ v stiahnutom 32-bit gcc MinGW prekladaci)
```

5) Preklad Rust projektu pomocou príkazu: **cargo build --target i686-pc-windows-gnu --release**, čím vygenerujeme DLL knižnicu pre 32-bit OS windows. DLL knižnicu nájdeme vo vygenerovanom adresári **target/i686-pc-windows-gnu/release/**.

5. Inštalácia jazyka Rust (zaručenie kombatibility s prekladačom MinGW)

Táto kapitola je venovaná inštalácii jazyka Rust. Inštalácia zahŕňa vybranie vhodného toolchainu (pre zachovanie kompatibility s prekladacom MinGW pre jazyk C). Kapitola taktiež opisuje postup zmeny toolchainu, ak sme Rust nainštalovali s iným toolchainom.

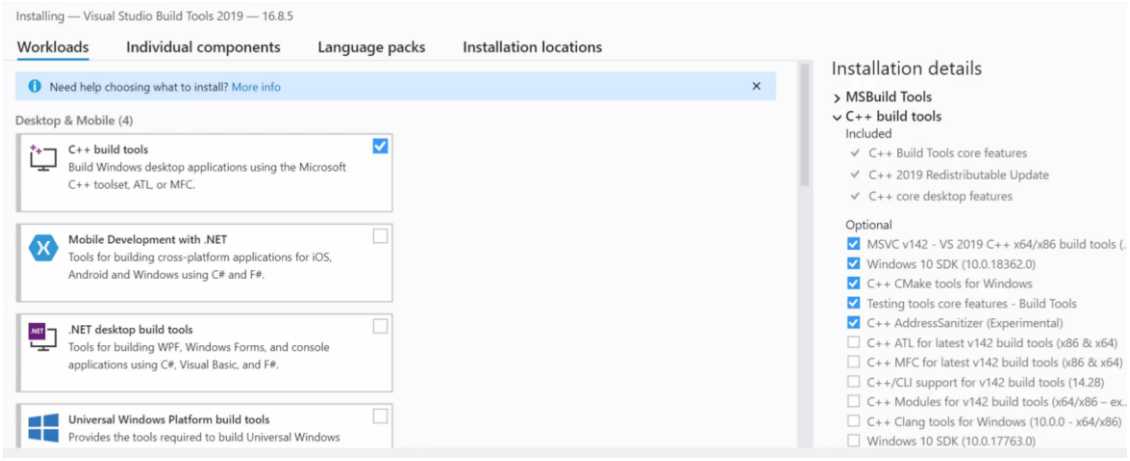
5.1.Základne vysvetlivky:

rustup – inštalátor pre jazyk Rust
rustc – kompilátor pre jazyk Rust
cargo – balíčkový manažér a tzv. build system
toolchain – špecifická verzia skupiny programov, ktoré sú potrebné pre zostavenie/preklad Rust aplikácie
(https://stackoverflow.com/questions/62417389/what-exactly-is-a-rust-toolchain#answer-62419829)

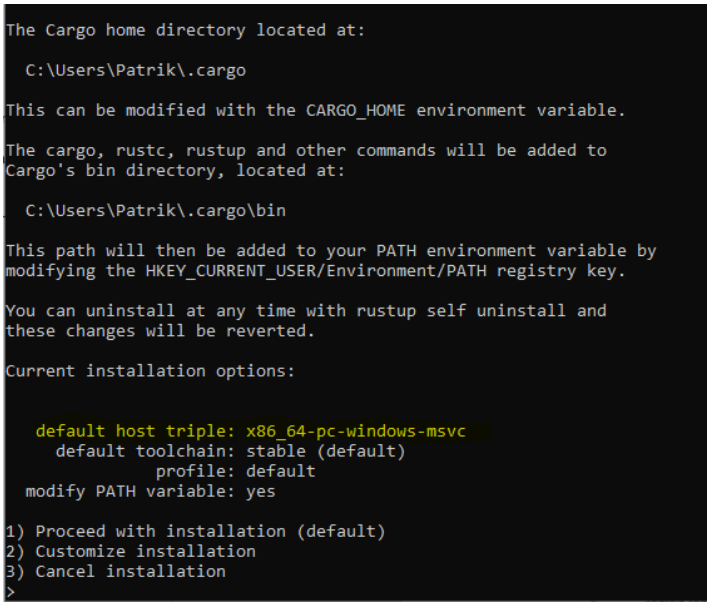
5.2.Inštalácia Rustu

Najjednoduchší spôsob ako nainštalovať Rust pre OS Windows je použiť rustup (https://www.rust-lang.org/tools/install). Na uvedenej linke je možné stiahnuť command-line rustup inštalátor a jazyk Rust nainštalovať v niekoľkých krokoch:

1) Pre OS Windows je vyžadovaná prítomnosť softvéru C++ build tools for Visual Studio 2013 alebo akúkoľvek novšiu verziu (https://visualstudio.microsoft.com/visual-cpp-build-tools/). Pri inštalácii C++ build tools for Visual Studio 2013 je nutné zvoliť inštaláciu C++ build tools.

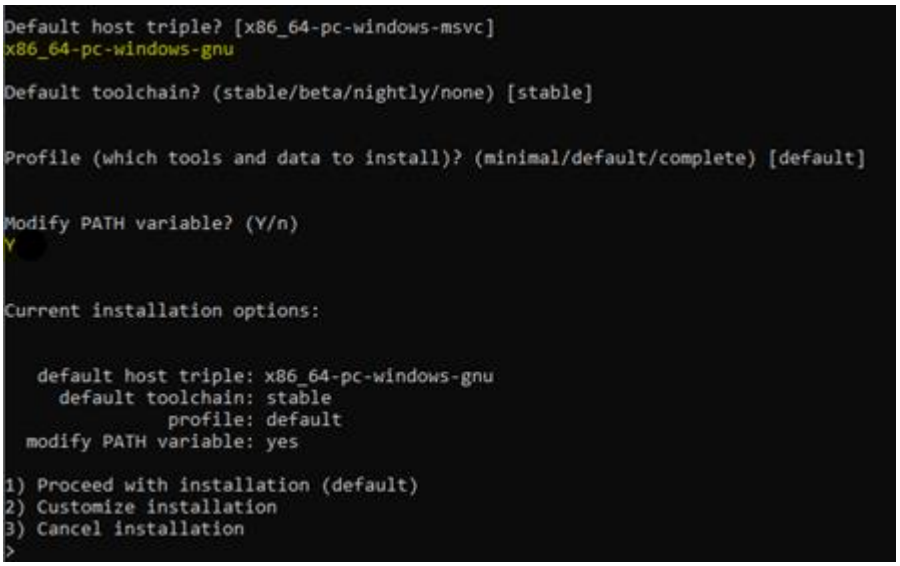


2) Inštalácia jazyka Rust pomocou rustup inštalátora. Dvojklikom na stiahnutý inštalátor sa nám zobrazí command-line inštalátor (za predpokladu, že máme správne nainštalovaný softvér C++ build tools):



Predvolený host triple ***X86_64-px-winodows-msvc*** je kompatibilný s MSVC kompilátorom, ak chceme voľbu zmeniť a inštaláciu dokončiť pre host triple, ktorý je kompatibilný s MinGW kompilátorom, zvolíme možnosť ***2) Customize instalation.***

3) Zvolením druhej možnosti bude od nás inštalátor požadovať verziu triple hostu, zadáme **x86_64-pc-windows-gnu** (kompatibilné s 64-bit. MinGW). Inštaláciu ďalej dokončíme s predvolenými (default) nastaveniami (stačí zadávať enter). Posledná možnosť súvisí s modifikovaním PATH premennej, zvolíme Y.



4) Posledným krokom je zvolenie možnosti číslo 1, teda nainštalovanie jazyka Rust.

Viac info o inštalacii jazyka Rust: https://doc.rust-lang.org/book/ch01-01-installation.html?highlight=rustc#installing-rustup-on-windows

5.3.Zmena toolchainu v Ruste

Ak sme Rust nainštalovali na inom toolchaine ako ***x86_64-pc-windows-gnu*** (povedzme, že na dafaultnom ***X86_64-px-winodows-msvc***), je možné túto konfiguráciu zmeniť. Zmena môže byť vykonaná viacerými spôsobmi ako napríklad za použitia command-line príkazov, metódou, ktorou sme prekladali Rust projekt (generovali DLL knižnicu) pre 32-bitovú platformu (s tým, že využijeme target a linker pre 64-bit. MinGW prekladac) alebo v globalnom nastavení (.cargo -> POZOR! - adresár, v ktorom je nainštalovaný programovací jazyk Rust, nie adresár nášho projektu). Najjednoduchším spôsobom je využitie command-line príkazu:

1) ***rustup default stable-x86_64-pc-windows-gnu,*** ktorý stiahne a zároveň nakonfiguruje tento toolchain ako predvolený (default).

Aktuálnu verziu toolchainu, ako aj všetky nainštalované toolchainy je možné skontrolovať pomocou príkazu: ***rustup show***.

Zoznam dostupných toolchainov: https://forge.rust-lang.org/infra/other-installation-methods.html alebo príkazom: ***rustc --print target-list***.

Ako odinštalovať Rust - príkaz: ***rustup self uninstall***

6. Ako vytvoriť Rust-C API od základov?

Táto kapitola stručne opíše princíp vytvorenia aplikácie od jej základov.

1) Vytvorenie Rust prostredia

   Prostredie pre programovanie aplikácie v jazyku Rust je možné jednoducho pripraviť príkazom **cargo init** ***nazov_projektu*** (vytvorí projekt v už existujúcom adresári) alebo **cargo new** ***nazov_projektu*** (vytvorí aj nový adresár).

2) Vytvorenie ***src/*** adresára

   Pri preklade projektu musí byť zachovaná určitá hierarchia adresárov, preto sa odporúča mať všetky zdrojové kódy v adresári ***src/***

3) Importovanie knižníc -> Cargo.toml

   V tomto kroku si vhodne nakonfigurujeme ***Cargo.toml***, čo je konfiguračný súbor ktorým sa riadi balíčkový manažer Cargo. V kapitole 3.2 sme si detailne opísali ako by súbor ***Cargo.toml*** mohol vyzerať pre danú aplikáciu.

4) Mapovanie (vytvorenie API)

   V adresári **src/** si vytvoríme súbor ***lib.rs***, v ktorom bude realizované mapovanie (viď kapitola 3.3). Detailne okomentovaný zdrojový kód pre aplikáciu CBlake3 CLI je dostupný v adresári **src/lib.rs**.

5) Automatizované generovanie hlavičkových súborov pre jazyk C

Automatizované generovanie hlavičkových súborov pre jazyk C je možné vykonať pomocou modulu **cbindgen**. Ten si nainštalujeme, vytvoríme konfiguračný súbor ***cbindgen.toml*** v hlavnom adresári nášho Rust projektu (Manifest) a v adresári ***src/build.rs*** pripravíme jednoduchý skript pre vytvorenie hlavičkových súborov (viď kapitola 3.4).

6) Preklad kódu + generovanie DLL knižnice

   Preklad kódu vykonáme príkazom: ***cargo build --release****,* pričom prepínač *--release* hovorí o preklade projektu v optimalizovanom móde (Viac v kapitole 3.5 a kapitole 4).

7. Využitie DLL knižnice v jazyku C

Prvým krokom k využitiu DLL knižnice je importovanie hlavičkových súborov, ktoré boli automatický vytvorené v adresári ***testprogram/headers/***. V adresári ***testprogram/*** vytvoríme súbor ***main.c***, v ktorom využijeme našu DLL knižnicu. Importovanie hlavičkových súboro vykonáme v súbore ***main.c*** nasledovne: ***#include "headers/mycrate.h".***

V hlavičkovom súbore testorogram/headers/mycrate.h môžeme vidieť, že naša Rust-C API využíva dátovu štruktúru MyString a funkciu Blake3C, pričom návratova hodnota funkcie Blake3C je práve dátova štruktúra MyString.

```
typedef struct MyString {
  const char *hash_code;
  float hash_time;
} MyString;
const struct MyString *Blake3C(int8_t threads_num, const char *s);
```

Volanie funkcie Blake3C v kóde je možné realizovať takto:

```
const MyString *Blake3Hash = Blake3C(pocet_vlakien, cesta_k_suboru); // Dátova štruktúra musí byť konštanta (kompatibilita s Rust API)
```

Po hašovaní súboru sa do dátovej štruktúry zapíšu hodnoty hašovacieho kódu (hash_code) a času (hash_time). Tieto hodnoty môžeme získať nasledovne:

```
Blake3Hash->hash_code;
Blake3Hash->hash_time; // v sec
```

Posledným krokom je preklad C-kódu a vytvorenie spustiteľného súboru, to je možné vykonať príkazom:

gcc -Wall -Werror main.c -L./ -lrustblake -o Blake3cli,

ktorý zároveň prilinkuje DLL knižnicu (DLL knižnica vytvorená pri preklade Rust projektu). Daná DLL knižnica (rustblake.dll) sa musí nachádzať v adresári **testprogram/** (rovnaký adresár v akom sa nachádza main.c).

Autor: P.Z.