KIV/TSI - Seminář C++

12. Knihovny C++ - Boost, Qt. Nástroj CMake

Martin Úbl

KIV ZČU

2020/2021

Knihovny pro C++

- C++ má poměrně velký ekosystém knihoven
- také díky existenci knihoven pro C a zpětnou kompatibilitu
- ale existuje i spousty ryze C++ knihoven

Knihovny pro C++

- Boost - obecná knihovna rozšíření std knihovny a funkcí
- Qt - knihovna pro GUI, expandovala i do podpůrných funkcí
- Wt - webový framework, DBO
- Eigen - knihovna matematických struktur a operací
- ACE - vysokoúrovňová abstrakce nad sítí
- SDL - knihovna pro 2D grafiku
- OpenCV - počítačové vidění a manipulace s obrazem
- Ogre3D, Irrlicht - knihovny pro 3D grafiku
- Vulkan - 3D grafické API
- Box2D - fyzikální simulace
- Intel TBB - paralelizace (KIV/PPR)
- Google Test, cppunit - testovací frameworky
- Log4cpp - logovací knihovna
- a další...

Boost

- Boost
- obecná knihovna, rozšíření standardní knihovny a funkcí
- „předbíhá“ standard C++
- vývojáři standardu C++ se účastní vývoje knihovny Boost
- spousty konceptů a principů bylo z Boost přímo přejato
  - vlákna
  - any, variant, optional, tuple
  - filesystem
  - smart pointery
  - generování (pseudo-)náhodných čísel
  - `enable_if`
  - chrono
  - některé standardní algoritmy
  - networking (C++20)
  - a spousty dalších

Boost

- Boost
- jmenný prostor `boost`
- velká část je součástí standardu a všech moderních standardních knihoven
- neznamená to, že se pro některé věci nevyplatí použít
- např. zpětná kompatibilita se staršími kompilátory
  - kompilace na zastaralém kompilátoru s podporou pouze C++03
  - pro některé věci stačí zaměnit `std::` za `boost::`
  - rozhraní je přizpůsobeno standardu

Boost

- Boost
- stažení
  - `https://www.boost.org/users/download/`
  - `https://github.com/boostorg/boost`
  - balíčkovací systém (apt, ...)
- kompletní dokumentace
  - `https://www.boost.org/doc/libs/`
- přeložení
  - v případě, že jsme stahnuli zdrojové soubory

Boost

- přeložení - Windows

```
bootstrap.bat
b2.exe --toolset=msvc-14.1
       --address-model=64
       --architecture=x86
       --runtime-link=static,shared
       --link=static
       --build-dir=build\x64
       --prefix="C:\boost"
       threading=multi
       install
```

- výsledek bude zkopírován do zadané složky (prefix), je nutno zadat cesty do MSVS

Boost

- GNU/Linux (Debian)
- kompletní vývojový balíček

```
apt install libboost-all-dev
```

- části

```
apt install libboost-tools-dev
            libboost-thread-dev ...
```

- potřebné knihovny jsou uloženy do standardních cest, totéž hlavičkové soubory

Boost

- některé moduly knihovny Boost
  - Algorithm - sbírka algoritmů, některé jsou v STL
  - Asio - síťová komunikace
  - Beast - HTTP a WebSocket klient/server
  - Bimap - „dvojmapa“ - klíčem můžou být obě hodnoty
  - Compute - vysokoúrovňový obal nad OpenCL
  - CRC - výpočet CRC kódů
  - DLL - vysokoúrovňová abstrakce nad prací s knihovnami
  - Fiber - knihovna pro uživatelská vlákna
  - Geometry - knihovna pro geometrické výpočty
  - Graph - tvorba grafů a grafové algoritmy
  - Log - logovací knihovna
  - Math - další matematické algoritmy a funkce
  - Meta State Machine - knihovna pro konečné automaty

Boost

- některé moduly knihovny Boost
  - Multiprecision - knihovna pro výpočty s vyšší přesností
  - Polygon - algoritmy pro práci s mnohoúhelníky
  - Process - multiplatformní práce s procesy
  - Python - vytváření Python bindings do C++
  - Stacktrace - výpis stacktrace za běhu
  - Test - knihovna pro testování
  - Timer - knihovna časovačů
  - a další...
  - `https://www.boost.org/doc/libs/`

Boost

- práce s vybranými - viz příklady

Qt

- Qt
- `https://www.qt.io/`
- multiplatformní knihovna pro tvorbu GUI
- expanze - rozšíření o další funkce
  - parsování XML
  - síťová komunikace
  - komunikace s databází
  - komunikace přes Bluetooth
  - mobilní aplikace - senzory, GPS, in-app purchases, NFC, ...
  - a další...
  - `http://doc.qt.io/qt-5/qtmodules.html`
- centrální zůstává GUI

Qt

- Qt
- `https://www.qt.io/`
- dvě distribuce
  - komerční
  - open-source (LGPLv3) - zdarma pro nekomerční použití
- instalátor stahne vybrané komponenty
- volitelně propojí s MSVS (Windows)
- lze použít s C++, ale i Pythonem, Rust, Go, C#, Haskellem, ...

Qt

- Qt
- problém s GUI mezi platformami
  - Windows používá zprávy
  - Unixové systémy signály/X11
- Qt sjednocuje API pomocí signálů a slotů (observer pattern)
- signál
  - událost, kterou objekt vyvolá při nějaké změně
  - tělo se neimplementuje
  - např. kliknutí na tlačítko, změna hodnoty, ...
- slot
  - příjemce signálu
  - tělo se implementuje
  - např. obsluha, co se stane po stisku tlačítka, ...

Qt

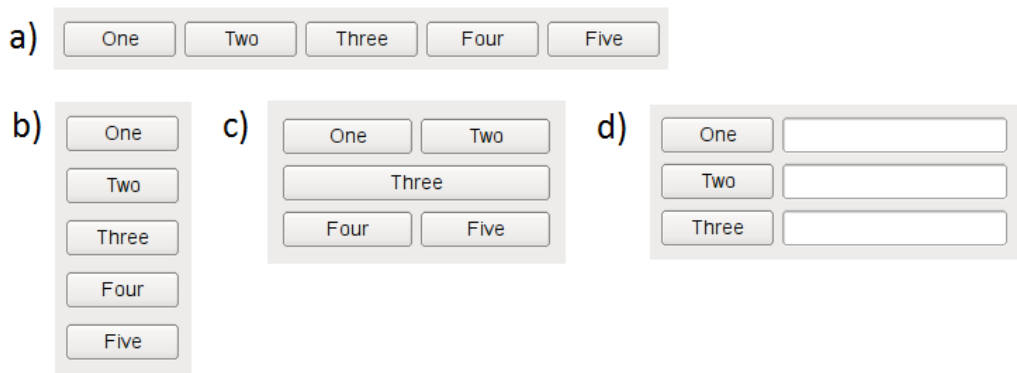
- Qt
- definice GUI
  - programově přidáváním prvků
  - QML
  - Qt Creator (používá výše uvedené)
- my se budeme zaobírat definicí v C++ kódu

Qt

- Qt
- různé prvky GUI
  - QWidget - obecný (prázdný) widget
  - QPushButton - tlačítko
  - QLabel - textový prvek
  - QLineEdit - pole pro zadávání textu
  - QGroupBox - seskupování prvků do vizuální oblasti
  - QDateEdit, QTimeEdit - pole pro zadávání data a času
  - QCheckBox - zaškrtávací pole
  - QRadioButton - přepínač
  - layouty

Qt

- Qt
- layouty
  - QVBoxLayout - vertikálně uspořádané prvky
  - QHBoxLayout - horizontálně uspořádané prvky
  - QGridLayout - uspořádání do mřížky
  - QFormLayout - dvousloupcpvé uspořádání
  - a další...



Obrázek: a) `QHBoxLayout`, b) `QVBoxLayout`, c) `QGridLayout`, d) `QFormLayout`

Qt

- každý potomek `QObject` může mít signály/sloty
  - jde to i bez dědění `QObject`
- každá třída používající signály a sloty musí obsahovat makro `Q_OBJECT`
- „nová sekce“ `signals` pro signály a `slots` pro sloty
  - ve skutečnosti jen makro, které Qt bere jako orientační
  - hlavičkové soubory se pak musí zpracovat nástrojem Qt `moc`

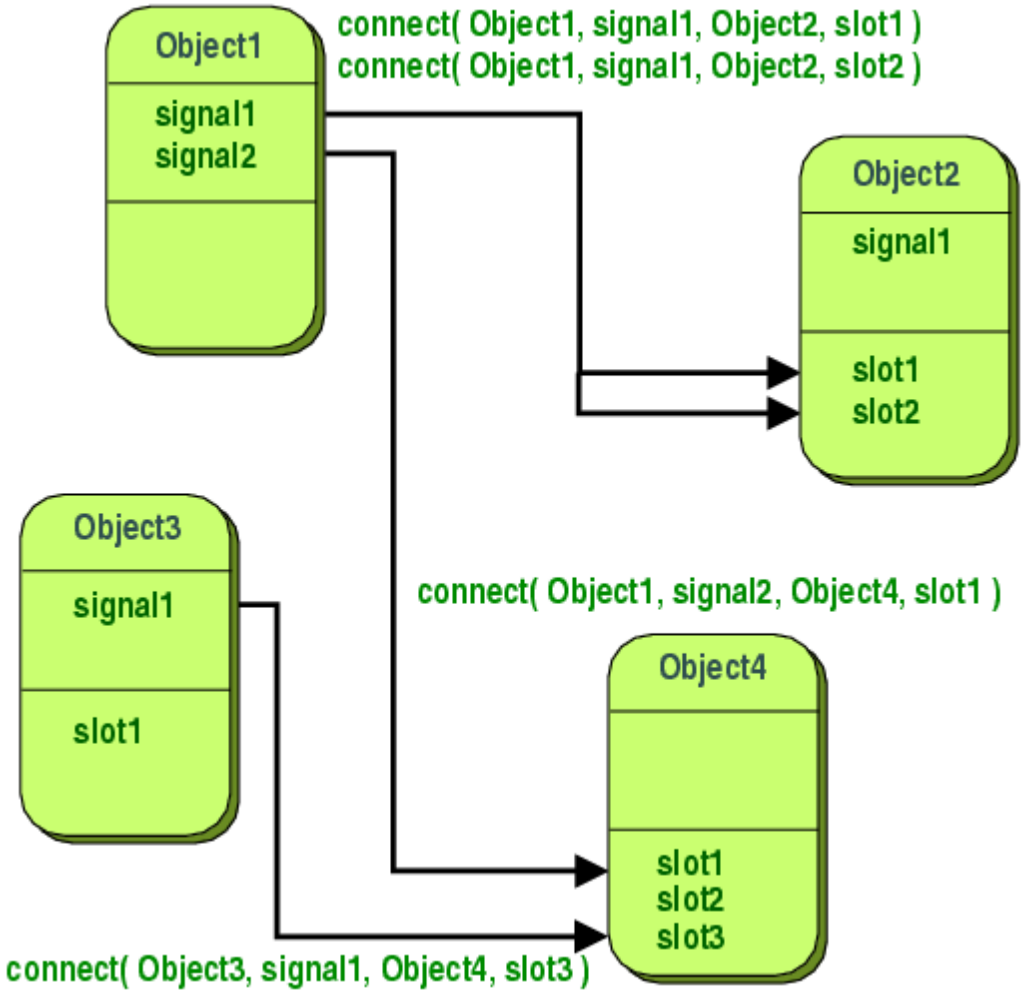
```
class MyQtObject : public QObject {
  Q_OBJECT

  private slots:
    // ...
  signals:
    // ...
};
```

- napojení slotu na signál
- metoda `connect()`
- lze propojovat signály a sloty různých objektů

```
QObject::connect(&a, SIGNAL(valueChanged(int)),
                 &b, SLOT(onValueChanged(int)));
```

- - když objekt a vyvolá signál `valueChanged`, je zavolána metoda `onValueChanged` objektu `b`

Qt



Obrázek:
http://doc.qt.io/archives/qt-4.8/signalsandslots.html

Qt

- napojení slotu na signál
- např. stisk tlačítka

```
QPushButton* btn = new QPushButton("Stiskni␣me")
QObject::connect(btn, SIGNAL(clicked()),
                 this, SLOT(onBtnClicked()));

// v deklaraci tridy
  private slots:
    void onBtnClicked();

// v implementaci
void MyWindow::onBtnClicked() {
 // ...
}
```

Qt

- Qt
- knihovna je poměrně rozsáhlá
- spousty widgetů, chování
- spousty podpůrných knihoven
- `http://doc.qt.io/qt-5/index.html`

- definice GUI není tak pohodlná programově
- vyšší komfort - Qt Creator GUI, QML
- nebo jiný jazyk (C# + XAML, Java + JavaFX, Python, ...) a C++ použít pro backend formou knihovny

Wt

- Wt
- framework pro tvorbu webové aplikace
- návrhově se inspiruje u Qt
- lze použít pro menší aplikace s požadavkem na rychlost
- používá trochu jiné principy, než jiné frameworky/jazyky
- `https://www.webtoolkit.eu/wt`
- nebudeme se jím zabývat do podrobna, jen představení

Wt

- Wt
- aplikace je psaná v C++, framework generuje relativně velké množství Javascript kódu
- do prohlížeče je pak poslána kostra celé aplikace
- AJAX požadavky se donačítá zbytek stránky, jak uživatel prohlíží
- vybrané části lze poslat rovnou, aby AJAX nebyl nutný

Wt

- Wt
- podporuje ověřovací mechanismy
  - jméno/email + heslo
  - OAuth
- zapomenuté heslo
- „fail2ban“

Wt

- Wt
- obsahuje databázovou vrstvu - Wt::Dbo
  - MySQL, PostgreSQL, SQLite, ...
- Object Relational Mapping (ORM)
  - mapování tříd a atributů na tabulky a jejich sloupce
  - co řádek, to instance třídy
- databázi si sám generuje a aktualizuje při spuštění
- lze použít i samostatně bez Wt

Wt

- Wt
- výsledkem je buď program nebo dynamická knihovna
  - program - používá vestavěný HTTP server
  - dynamická knihovna - lze připojit jako CGI modul Apache/nginx/IIS/...
- vestavěný HTTP server podporuje šifrování a všechny standardní věci, co by takový daemon měl umět

Wt

- kostra aplikace

```
class HelloApplication : public Wt::WApplication
{
  public:
    HelloApplication(const Wt::WEnvironment& env) {
      // ... setup ...
    }
};

int main(int argc, char **argv)
{
  return Wt::WRun(argc, argv,
        [](const Wt::WEnvironment& env) {
    return std::make_unique<HelloApplication>(env);
  });
}
```

Wt

- má vlastní sadu widgetů
  - Wt::WPushButton - tlačítko
  - Wt::WLineEdit - pole pro vstup
  - Wt::WLabel - popisek
  - a další...
- používá obdobu signálů Qt, ale za použití Boost/C++ functional a bind vlastností
- nepotřebuje tvorbu rozhraní odstínit od konkrétního OS - generuje webový layout a styly interpretované až prohlížečem dle standardu
- kořenový prvek stránky - v instanci aplikace `root()` - v podstatě `<body>` element

Wt

```
mGreetBtn = root()->addWidget(
    std::make_unique<Wt::WPushButton>("Hello␣there"));

mText = root()->addWidget(
    std::make_unique<Wt::WText>());

auto greet = [this](){
  mText->setText("General␣Kenobi");
};

button->clicked().connect(greet);
```

Wt

- pro každé sezení jedna instance `Wt::WApplication`
- prvky uchované jako atributy jsou tedy unikátní pro každého
- instance je udržovaná po celou dobu trvání sezení
- lze se tedy spolehnout, že prvky budou v paměti při každé události (stisk tlačítka, časovač, ...)

Wt

- podpora šablon a překladů funkcí `tr()`
- v instanci aplikace např.:

```
messageResourceBundle().use("templates.xml");

auto tpl =
      std::make_unique<Wt::WTemplate>(tr("homepage"));
root()->addWidget(std::move(tpl));
```

- v `templates.xml`:

```
<messages>
  <message id="homepage">
    Localized greeting: ${tr:greeting}
  <message>
</messages>
```

Wt

- soubor s lokalizací
- např. `lang_cs.xml`:

```
<messages>
  <message id="greeting">Ahoj<message>
</messages>
```

- např. `lang_en.xml`:

```
<messages>
  <message id="greeting">Hello<message>
</messages>
```

Wt

- mixovaný layout s programově definovanými widgety

```
<messages>
  <message id="homepage">
    ${tr:label_username}: ${username_edit}
  <message>
</messages>
```

- v aplikaci

```
auto editbox = std::make_unique<Wt::WLineEdit>();

tpl->bindWidget("username_edit", editbox);
```

Wt

- showroom: `https://www.webtoolkit.eu/widgets/`
- je dobré vědět, že Wt existuje
- potencálně vhodné pro menší aplikace s kritériem výkonu
- weby které běží na Wt:
  - `https://www.webtoolkit.eu/wt`
  - `https://diabetes.zcu.cz`
- konkurent: CppCMS, CWF, treefrog, ...
- nemusí být dobrý nápad implementovat web v C++
- opět se lze vydat cestou jiného jazyka a použít C++ v backendové knihovně
- nebo použít Boost/POCO/... pro realizaci REST API nebo jiného rozhraní

Emscripten

- vsuvka: Emscripten
- kompilace C/C++ kódu do Javascriptu
- vysoký potenciál optimalizace díky LLVM
- spíše pro zajímavost
- ... a pro ty, co mají fobii z Javascriptu (a ne z C/C++)
- `https://kripken.github.io/emscripten-site/`
- kompilátor `emcc` (wrapper)
- výsledek lze interpretovat pomocí Node.js nebo v prohlížeči

Emscripten

- program píšeme, jako kdybychom psali dynamickou knihovnu
- „exportované“ funkce pak lze zavolat jako javascriptové
- lze tak mít webovou stránku propojenou s kompilovaným C++ do JS
- díky HTML5 a `<canvas>` prvku lze např. použít i SDL knihovnu pro grafický výstup
- některé věci je třeba obejít - např. práci se soubory (při kompilaci preload)

Emscripten

- používají emscripten:
  - `https://diabetes.zcu.cz/` - výpočet signálu modelem a generování SVG
  - `https://files.unity3d.com/jonas/AngryBots/` - portované Unity + WebGL
  - `http://www.quakejs.com/` - Quake 3
  - `https://jackbenfu.itch.io/pong` - Pong
  - a spousty dalších...
  - `https://github.com/kripken/emscripten/wiki/Porting-Examples-and-Demos`

CMake

- nástroj pro dynamické generování konfigurací k sestavení
- generuje MSVS solutions, makefile, Codeblocks projekty, atd...
- CLI utilita
- má i GUI nástavbu
- integrace do MSVS

CMake

- root soubor
- zpravidla obsahuje definici min. verze

```
CMAKE_MINIMUM_REQUIRED(VERSION 2.4)
```

- vždy obsahuje název celého projektu (workspace, solution)

```
PROJECT(my-fancy-cpp-app)
```

- pro přehlednost je možné, aby podsložky obsahovaly také CMakeLists.txt, který bude načten odděleně; vyvoláme ho přidáním podsložky

```
ADD_SUBDIRECTORY(src)
```

- nic z toho zatím nemá vliv na strukturu projektu

CMake

- přidání spustitelného souboru k sestavení (resp. projektu pro MSVS v rámci solution)

```
ADD_EXECUTABLE(<target-name> <file list...>)
```

- např.

```
ADD_EXECUTABLE(my-executable main.cpp module.cpp othermodule.cpp)
```

- toto vygeneruje k sestavení soubor `my-executable` (resp. `my-executable.exe` na Windows)

CMake

- přidání knihovny k sestavení (resp. projektu pro MSVS v rámci solution) probíhá analogicky

```
ADD_LIBRARY(<target-name> [library type] <file list...>)
```

- kde `[library type]` může být např.:
  - `SHARED` - dynamicky linkovaná knihovna (.dll, .so, .dylib)
  - `STATIC` - staticky linkovaná knihovna (.lib, .a)
- např.

```
ADD_LIBRARY(my-toolset STATIC libmain.cpp a.cpp b.cpp)
```

- toto vygeneruje k sestavení knihovnu `my-toolset.a` (resp. `my-toolset.lib` na Windows)

CMake

- include cesty se dají přidat globálně / pro jeden cíl

```
INCLUDE_DIRECTORIES(<dirs>)
TARGET_INCLUDE_DIRECTORIES(<target> <specifier> <dirs>)
```

- hodnota `specifier` může být
  - `INTERFACE`
  - `PUBLIC` - vystačíme si s touto
  - `PRIVATE`
- např. pro přidání složky `include`

```
INCLUDE_DIRECTORIES(${INCLUDE_DIRECTORIES} include/)
TARGET_INCLUDE_DIRECTORIES(my-executable PUBLIC
                ${INCLUDE_DIRECTORIES} include/)
```

CMake

- linkování knihoven taktéž globálně / pro jeden cíl
- pozor - globálně je třeba před přidáním cílů

```
LINK_LIBRARIES(<libraries>)
TARGET_LINK_LIBRARIES(<target> <libraries>)
```

- např.

```
LINK_LIBRARIES(my-toolset)
TARGET_LINK_LIBRARIES(my-executable my-toolset)
```

- názvem knihovny je buď konkrétní knihovna ze systému/z cesty
- nebo nějaká z knihoven, kterou CMake má definovanou pro sestavení

CMake

- je potřeba modul CMake
- nějaké má CMake přibalené s sebou
- modul je možno vyvolat

```
FIND_PACKAGE(<packagename> [flag])
```

- `flag` nás bude zajímat jen jedna: `REQUIRED` - nelze bez této knihovny pokračovat
- např.

```
FIND_PACKAGE(OpenSSL REQUIRED)
```

- modul nastavuje určité proměnné
- ostatní silně specifické

CMake

- interní moduly CMake nastavují standardizované proměnné
- vždy `<packagename>_FOUND` pokud se knihovnu podaří nalézt
- často `<packagename>_INCLUDE_DIR` - kde hledat hlavičkové soubory
- často `<packagename>_LIBRARIES` - seznam knihoven k linkování
- všechny moduly které s sebou CMake nese jsou zdokumentovány: `https://cmake.org/cmake/help/latest/manual/cmake-modules.7.html`

CMake

- často je potřeba předat nějaký přepínač pro parametrizaci sestavení
- opět lze globálně i pro target

```
ADD_DEFINITIONS(<defines>)
TARGET_COMPILE_DEFINITIONS(<target> <specifier> <defines>)
```

- pozor, defines musí být včetně prefixu `-D` pro preprocesorové direktivy (aby se chovaly jako `#define`)
- např.

```
ADD_DEFINITIONS(-DDEV_BUILD)
TARGET_COMPILE_DEFINITIONS(my-executable PUBLIC -DDEV_BUILD)
```

CMake

- abychom nemuseli otrocky vypisovat seznam souborů, lze je nechat vyhledat
- po vyhledání jsou uloženy do proměnné
- využijeme příkaz `FILE`, který má širší využití

```
FILE(<operation> <parameters>)
```

- pro vyhledání souborů v dané složce s příponou .cpp a .h např.:

```
FILE(GLOB_RECURSE my_app_sources ./ *.cpp *.h)
```

- operace `GLOB_RECURSE` projde i podsložky
- pokud nechceme, lze použít jen `GLOB`

CMake

- a další...
- řídicí struktury - `IF`, `FOR`, ...
- zprávy do konzole - `MESSAGE`
- generování balíčků, např. dpkg - CPack
- ...
- `https://cmake.org/cmake/help/latest/`