

# Konec a tak

PB173 Programování v C++11

Vladimír Štill, Jiří Weiser

Fakulta Informatiky, Masarykova Univerzita

15. prosince 2014

- Perf
- constexpr
- initializer list
- práce s časem
- hash a hashovací kontejnery
- regulární výrazy

Linuxový nástroj na profilování kódu.

- využívá hardwarových počítadel v CPU (rychlé)
- spouští se utilitou `perf` s několika podpříkazy
  - `top` momentální aktivita jednoho (s `--pid`) nebo všech procesů
  - `record` zaznamenává aktivitu programu po dobu jeho běhu
  - `report` analýza zaznamenané aktivity
- některé funkce (`top`, analýza systémových volání) vyžadují extra povolení v konfiguraci systému (na FI zakázáno)

# Perf: detekce náročných operací

Chceme zjistit, které funkce trvají nejdéle a kdo je volá.

- kompilujeme s `-g -fno-omit-frame-pointer`
- v ideálním případě linkujeme s `libc` a dalšími knihovnami s debug informacemi
- `perf record -a --call-graph fp`
  - `-a` všechna CPU
  - `--call-graph fp` zaznamenat call graf, použít analýzu frame-pointrů k vytvoření call-grafu (méně přesné, ale nevyžaduje rekompilaci knihoven)
  - vyprodukuje `perf.data` záznam (lze použít `-o soubor`)
- `perf report -g -G --stdio` k zobrazení záznamu
  - `-g` zobrazit call-graf
  - `-G` zobrazovat v grafu volající funkce, ne volané
  - `--stdio` méně grafický výstup (dlouhé C++ symboly jinak nelze číst)

# Perf: demo

Měření rychlosti práce s vektorem a deque v C++98 a C++11. `cecko.eu perf demo: bench.cpp + Makefile`

Měření rychlosti práce s vektorem a deque v C++98 a C++11. `cecko.eu perf demo: bench.cpp + Makefile`

- C++98 verze tráví hodně času v copy konstruktorech `std::string` (při `resize` `std::vector`)
- stejný kód zkompilovaný C++11 je výrazně rychlejší
- r-value reference mají smysl

# constexpr

Označení výrazu, že se má vypočítat v době překladu.

Lze aplikovat na

- proměnné

# constexpr

Označení výrazu, že se má vypočítat v době překladu.

Lze aplikovat na

- proměnné
- funkce/metody
  - nesmí být virtuální
  - musí vracet *LiteralType*
  - parametry musí být *LiteralType*
  - musí mít přesně jeden **return** (platí do C++14)



# constexpr

Označení výrazu, že se má vypočítat v době překladu.

Lze aplikovat na

- proměnné
- funkce/metody
  - nesmí být virtuální
  - musí vracet *LiteralType*
  - parametry musí být *LiteralType*
  - musí mít přesně jeden **return** (platí do C++14)
- konstruktory
  - Platí podobná omezení jako pro funkce/metody.

Pokud nelze poznačenou funkci vykonat v době překladu, stane se z ní **inline** funkce.

# constexpr

```
constexpr int factorial( int n ) {  
    return n > 1 ? n * factorial( n - 1 ) : 1;  
}
```

# constexpr

```
constexpr int factorial( int n ) {  
    return n > 1 ? n * factorial( n - 1 ) : 1;  
}
```

- Napište `constexpr` funkci Fibonacci.

# constexpr

```
constexpr int factorial( int n ) {  
    return n > 1 ? n * factorial( n - 1 ) : 1;  
}
```

- Napište `constexpr` funkci Fibonacci.
- Upravte ji, aby měla lineární složitost.

# `std::initializer_list`

- Umožňuje inicializovat třídu seznamem hodnot
  - Podobně jako inicializace pole v místě deklarace

# std::initializer\_list

- Umožňuje inicializovat třídu seznamem hodnot
  - Podobně jako inicializace pole v místě deklarace
- Jedná se o kontejner
  - má metody begin, end, size, empty, operator[]

# std::initializer\_list

- Umožňuje inicializovat třídu seznamem hodnot
  - Podobně jako inicializace pole v místě deklarace
- Jedná se o kontejner
  - má metody begin, end, size, empty, operator[]

```
#include <initializer_list>
struct A {
    // bere se hodnotou
    A( std::initializer_list< int > list ) : ....
    ...
    void add( std::initializer_list< int > list ) {
        ...
    }
};
```

# Práce s časem

hlavičkový soubor <chrono>

C++ definuje 3 různé hodiny:

- `std::chrono::system_clock`
  - čas systému
- `std::chrono::steady_clock`
  - čas, který nelze posouvat (letní čas, přestupné sekundy)
- `std::chrono::high_resolution_clock`
  - nejpřesnější čas<sup>1</sup>

Každé hodiny mají statickou metodu `now`, která udává aktuální čas.

---

<sup>1</sup>při práci s malými časovými úseky



# Práce s časem

hlavičkový soubor <chrono>

- `std::chrono::time_point`
  - časový okamžik
- `std::chrono::duration`
  - časový úsek
  - výsledek operace rozdílu dvou `std::chrono::time_point`
- `std::chrono::duration_cast`
  - umožňuje definovat rozlišení času
    - `std::chrono::hours`
    - `std::chrono::minutes`
    - `std::chrono::seconds`
    - `std::chrono::milliseconds`
    - `std::chrono::microseconds`
    - `std::chrono::nanoseconds`

# Práce s časem – měřič času

```
namespace T = std::chrono;
class Timer {
    using Clock = T::high_resolution_clock;
    using TimePoint = T::time_point< Clock >;
    TimePoint _startTime, _stopTime;
public:
    void start() { _startTime = now(); }
    void stop() { _stopTime = now(); }
    static TimePoint now() { return Clock::now(); }
    long long elapsed() const {
        return T::duration_cast< T::milliseconds >(
            _stopTime - _startTime
        ).count();
    }
};
```

# Hashovací kontejnery

- obdoba `std::set`, `std::map`
- datové kontejnery používající hash
  - `std::unordered_set`, `std::unordered_map`
  - `std::unordered_multiset`,  
`std::unordered_multimap`

# Hashovací kontejnery

- obdoba `std::set`, `std::map`
- datové kontejnery používající hash
  - `std::unordered_set`, `std::unordered_map`
  - `std::unordered_multiset`,  
`std::unordered_multimap`
- nutné specializovat třídu `std::hash`
  - Případně implementovat vlastní třídu se stejným rozhraním.
- mohou být rychlejší
- nepoužitelné s `std::set_union`,...

# Hashovací kontejnery

```
namespace std {  
    template<>  
    struct hash< A > {  
        size_t operator()( const A &a ) const {  
            size_t h;  
            // výpočet hashe  
            return h;  
        }  
    };  
}
```

# Regulární výrazy

- `<regex>`
- `std::regex`
- `std::regex_match` vyhodnotí regex nad celým výrazem (a umožní získat podvýrazy)
- `std::regex_search` vyhledávání podle regexu
- `std::regex_replace` nahrazení
- `std::regex_iterator` postupné hledání
- několik syntaxí regulárních výrazů: ECMAScript (výchozí), POSIX (extended), awk, (e)grep
- relativně špatná podpora v kompilátorech (gcc od 4.9, clang jen s libcpp, VS)
- <http://en.cppreference.com/w/cpp/regex>

# Regulární výrazy: demo

Parser dynamicky typovaných výrazů ze stringu.  
Viz [cecko.eu](http://cecko.eu), `regex.cpp`.