

Variadické šablony

PB173 Programování v C++11

Vladimír Štill, Jiří Weiser

Fakulta Informatiky, Masarykova Univerzita

10. listopadu 2014

- Syntaxe
- Jak s tím pracovat
 - Jak zjistit počet
 - S funkcemi
 - Se třídami
- Kde se to používá
 - `std::thread`, `std::make_shared`, ...
 - `std::tuple`

Syntaxe

```
// normální šablona třídy  
template< typename T >  
struct A {};
```

```
// variadická šablona třídy  
template< typename... Args >  
struct B {};
```

```
// variadická šablona funkce  
template< typename... Args >  
void foo( Args... args ) {}
```

Jak s tím pracovat

Jak zjistit počet

```
template< typename... Args >  
void foo( Args... args ) {  
    return sizeof...( Args );  
}  
foo(); // 0  
foo( 1 ); // 1  
foo( "string", 3.14 ); // 2
```

Jak s tím pracovat

S funkcemi

- V případě, že **nepotřebujete** přistupovat k jednotlivým parametrům, je možné balíček parametrů poslat dál.
- V případě, že **potřebujete** přistupovat k jednotlivým parametrům, nevyhnete se rekurzivnímu volání.

Jak s tím pracovat

S funkcemi – přeposlání balíčku parametrů dál

```
template< typename T, typename... Args >
std::shared_ptr< T > make_shared( Args... args )
{
    std::shared_ptr< T > ptr( new T( args... ) );
    return ptr;
}
```

Příklad není správně

- Parametry se budou kopírovat, i když by nemusely

Jak s tím pracovat

S funkcemi – přeposlání balíčku parametrů dál

```
template< typename T, typename... Args >
std::shared_ptr< T > make_shared( Args... args )
{
    std::shared_ptr< T > ptr( new T( args... ) );
    return ptr;
}
```

Příklad není správně

- Parametry se budou kopírovat, i když by nemusely
- Chtěli bychom
 - Dočasné hodnoty přesunout
 - Trvalé hodnoty zkopírovat

Jak s tím pracovat

S funkcemi – přeposlání balíčku parametrů dál (správná varianta)

```
template< typename T, typename... Args >
auto make_shared( Args &&... args )
    -> std::shared_ptr< T >
{
    std::shared_ptr< T > ptr(
        new T( std::forward< Args >( args )... )
    );
    return std::move( ptr );
}
// auto je jenom kvůli místu
```


Jak s tím pracovat

S funkcemi – přístup na jednotlivé parametry

```
// varianta z STD, ale pouze pro 2 čísla  
int m = std::max( a, b );
```

```
// vylepšená varianta  
template< typename T >  
T max( T a, T b ) {  
    return std::max( a, b );  
}  
  
template< typename T, typename... Args >  
T max( T a, T b, Args... pack ) {  
    return std::max( a, max( b, pack... ) );  
}
```

Jak s tím pracovat

S třídami – přístup na jednotlivé parametry

Příklad

- Chtěli bychom explicitně ověřit, že všechny předané parametry jsou jednoho typu.

Jak s tím pracovat

S třídami – přístup na jednotlivé parametry

```
template< class T1, class T2, class... TN >
struct are_same {
    static const bool value =
        are_same< T1, T2 >::value &&
        are_same< T2, TN... >::value;
};

template< typename T1, typename T2 >
struct are_same< T1, T2 > {
    static const bool value = false;
};

template< typename T >
struct are_same< T, T > {
    static const bool value = true;
};
```

Jak s tím pracovat

Použití ve třídách a funkcích dohromady

```
template< typename T >
T max( T a, T b ) {
    return std::max( a, b );
}

template< typename T, typename... Args >
T max( T a, T b, Args... pack ) {
    static_assert(
        are_same< T, Args... >::value,
        "Types must be the same"
    );
    return std::max( a, max( b, pack... ) );
}
```

Kde se to používá

V rámci STL (vybrané funkce a třídy)

```
// konstruktor třídy std::thread  
template< class Function, class... Args >  
explicit thread( Function &&f, Args &&... args );
```

```
// funkce std::make_shared  
template< class T, class... Args >  
shared_ptr<T> make_shared( Args &&... args );
```

```
// třída std::tuple a funkce std::make_tuple  
template< class... Types > class tuple;  
  
template< class... Types >  
std::tuple< Types... > tuple( Types &&... args );
```

Kde se to používá

Třída `std::tuple` (hlavičkový soubor `<tuple>`)

```
// auto == std::tuple< int, double, std::string >  
auto tuple = std::make_tuple(  
    1, 3.14, std::string( "text" )  
);  
std::get< 2 >( tuple ) += "ik";  
std::cout << std::get< 0 >( tuple ); // 1  
std::cout << std::get< 2 >( tuple ); // textik  
int i;  
double d;  
std::string t;  
std::tie( i, d, t ) = tuple;  
// i == 1, d == 3.14, t == "text"
```