

Úvod do C++11

PV173 Programování v C++11

Vladimír Štill, Jiří Weiser

Fakulta Informatiky, Masarykova Univerzita

15. září 2014

Představení C++11 – auto

Označení typu

Typ `auto` říká, že překladač si má sám doplnit správný typ. Překladač nic nehádá, odvození funguje podle (mnohdy komplikovaných) pravidel.

Představení C++11 – nullptr

Speciální hodnota

- význam jako NULL
- typově bezpečné
 - **NULL** je typu **int**
 - **nullptr** má typ konvertibilní pouze na ukazatele

Představení C++11 – default & delete

Vlastnosti metod/funkcí

```
class A {  
    A() = default;  
    A( const A &) = delete;  
};  
void foo( int ) = delete;  
template< typename T > void foo( T ){}
```

Představení C++11 – static_assert

User-defined kontrola při překladu.

```
template< typename T >
class A {
    static_assert(
        !std::is_same< T, bool >::value,
        "T cannot be bool!" );
};
```

Představení C++11 – strong enum

Enum pro lidi

Uzavře hodnoty výčtu do jmenného prostoru výčtu.
Lze též definovat typ, nad kterým je **enum** realizován.

```
enum class colors : char {  
    red = 'r',  
    blue = 'b',  
    green = 'g',  
};  
colors c = colors::red;
```

Představení C++11 – nový zápis funkce

V C++14 není povinné zapsat šipku a typ.

```
auto foo() -> int { return 1 }
```

Představení C++11 – decltype

Automatické odvození typu z výrazu; doplněk k auto.

Velmi vhodné v místě, kdy sice známe výraz, ale absolutně netušíme typy, které se ve výrazu použijí – situace v například případě použití šablon.

```
template< typename T >
auto foo( T t )
    -> decltype( squeeze( t ) )
{
    return squeeze( t );
}
```


Představení C++11 – range-for

Nový zápis cyklu

```
for ( auto i : { 1, 2, 3, 4, 5 } )  
    std::cout << i;  
  
{  
    auto &&__c = {1, 2, 3, 4, 5};  
    auto __i = beginexpr( __c );  
    auto __e = endexpr( __c );  
    for ( ; __i != __e; ++__i ) {  
        auto i = *__i;  
        std::cout << i;  
    }  
}
```

Zopakování šablon – jednoduchá šablona

Klíčová slova **typename** a **class** jsou si v těchto šablonových definicích rovny.

```
template< typename T >  
void foo( T ) {}
```

```
template< class T >  
void bar( T ) {}
```

```
template< typename T >  
struct A {};
```

Zopakování šablon – specializace

```
template< typename T >  
class A {};
```

```
template< typename T >  
class A< T * > {};
```

```
template<>  
class A< bool > {};
```

Zopakování šablon – šablona v šabloně

Zde nelze **class** nahradit za **typename**!

```
template<
    template< typename, typename > class T,
    typename U,
    typename V
>
struct A {
    T< U, V > data;
};
```

Samostatný úkol

- Implementujte šablonový vektor.
- Požadavky
 - Při realokaci se zvětší na dvojnásobek; od velikosti 4096 již jenom přičítá 4096.
 - Nesmí být implementován za pomoci žádného STL kontejneru.
 - Nesmí v něm docházet k memory leakům.
 - Musí splňovat následující rozhraní

Samostatný úkol – rozhraní

- Konstruktory
 - Bezparametrický
 - Kopírovací
 - `vector(int n);` předalokuje paměť pro n položek.
- Operátory
 - Porovnání `==, !=`
 - Indexace `[]`
 - Přiřazení `=`
- `size() const;`
- `capacity() const;`
- `push_back(T);`
- `begin();`
- `end();`
- `erase(typename vector::iterator);`