Koncepty v C++

Študent: Marek Lošonský

Školiteľ: doc. RNDr. Robert Lukoťka, PhD.

Cieľ práce

- prehľad problematiky konceptov
- použitie v jazyku – písanie a viazanie na parametrizované typy
- diagnostika kompilačných chýb s konceptami
- motivácia k použitiu konceptov
- porovnanie s predchádzajúcimi spôsobmi na vymedzovanie typov
- smernice na správne použitie konceptov

Generické programovanie

- štýl programovania, konkrétny dátový typ → parametrizovaný dátový typ
- podpora v C++ – **šablóny** (*templates*)
- aktualizovanie, oprava, redukcia duplicity, viacúčelovosť
- sémantické obmedzenia

```
template<typename C>
void sort(C container) {
        /* algoritmus na utriedenie zoznamu prvkov */
}

...

sort(c);
```

Spôsoby vymedzovania typov

- komentáre
- operand **typeid** – testovanie počas behu programu
- **static assert** (statické tvrdenia) – testovanie v kompilačnom čase, zložitá diagnostika kompilačných chýb

```
template<typename C>
void sort(C container) {
        /* podmienky na typ C,
        aby reprezentoval zoznam prvkov
        (ktorý je usporiadateľný)
        */
        static_assert(...);

        /* algoritmus triedenia */
        ...
}
```

Spôsoby vymedzovania typov

- **SFINAE** – testovanie v kompilačnom čase, náročnosť implementácie

```
template<typename C>
std::enable_if_t<!constraints, void> sort(C container) {
        /* prípad, že podmienky na typ C nie sú splnené) */
        ...
}

template<typename C>
std::enable_if_t<constraints, void> sort(C container) {
        /* typ C reprezentuje zoznam prvkov,
        ktorý je usporiadateľný) */

        /* algoritmus triedenia */
        ...
}
```

Koncept

- pomenovaná množina požiadaviek pre parametrizované typy (*constraints*)
- booleovský predikát počítajúci sa (vyhodnotený) v čase kompilácie
- C++20, idea C++11
- knižnica concepts
- **syntax** – constraint-expression: **requires** klauzula, typová charakteristika, **koncept** + booleovské operácie

template<*template-parameter-list*>
concept *concept-name* = *constraint-expression*;

Viazanie konceptu na parametrizované typy

- funkcia **sort** a koncept **sortable** (z knižnice concepts)

```
#include<concepts>

template<std::sortable C>
void sort(C container)

template<typename C>
requires std::sortable<C>
void sort(C container)

void sort(std::sortable auto C)
```

Chybové výpisy kompilátora

- chybové výpisy súvisiace s šablónovým kódom môžu byť zložité

```
#include<list>
#include<algorithm>

int main() {
        auto list = std::list{1, 2, 3};
        std::sort(list.begin(), list.end());

        //list.sort();
}
```

```
In file included from main.cpp:1:
In file included from
/root/emsdk/upstream/emscripten/cache/sysroot/include/c++/v1/list:186:
In file included from
/root/emsdk/upstream/emscripten/cache/sysroot/include/c++/v1/algorithm:716:
/root/emsdk/upstream/emscripten/cache/sysroot/include/c++/v1/__algorithm/make
_heap.h:29:34: error: invalid operands to binary expression
('std::__list_iterator<int, void *>' and 'std::__list_iterator<int, void *>')
    difference_type __n = __last - __first;
                          ~~~~~~ ^ ~~~~~~
/root/emsdk/upstream/emscripten/cache/sysroot/include/c++/v1/__algorithm/part
ial_sort.h:38:12: note: in instantiation of function template specialization
'std::__make_heap<std::__less<int> &, std::__list_iterator<int, void *>>'
requested here
    _VSTD::__make_heap<_Compare>(__first, __middle, __comp);
           ^
/root/emsdk/upstream/emscripten/cache/sysroot/include/c++/v1/__algorithm/sort
.h:540:12: note: in instantiation of function template specialization
'std::__partial_sort<std::__less<int> &, std::__list_iterator<int, void *>>'
requested here
    _VSTD::__partial_sort<_Comp_ref>(__first, __last, __last,
_Comp_ref(__comp));
           ^
/root/emsdk/upstream/emscripten/cache/sysroot/include/c++/v1/__algorithm/sort
.h:551:12: note: in instantiation of function template specialization
'std::sort<std::__list_iterator<int, void *>, std::__less<int>>' requested
here
    _VSTD::sort(__first, __last, __less<typename
iterator_traits<_RandomAccessIterator>::value_type>());
           ^
main.cpp:7:10: note: in instantiation of function template specialization
'std::sort<std::__list_iterator<int, void *>>' requested here
    std::sort(list.begin(), list.end());
          ^
/root/emsdk/upstream/emscripten/cache/sysroot/include/c++/v1/__iterator/move_
iterator.h:152:6: note: candidate template ignored: could not match
'move_iterator' against '__list_iterator'
auto operator-(const move_iterator<_Iter1>& __x, const move_iterator<_Iter2>&
__y)

a ďalších 48 riadkov chybového výpisu
```

Diagnostika kompilačných chýb s konceptom

- jedna z hlavných výhod konceptov
- jednoduchá diagnostika kompilačnej chyby na rozdiel od starších spôsobov na vymedzovanie typov
- ukážka na funkcii súčtu čísel po dosadení objektov typu **vector**

```
main.cpp:17:16: error: no matching function for call to 'add_numbers'
 auto result = add_numbers(first, second);
               ^~~~~~~~~~~
main.cpp:8:6: note: candidate template ignored: constraints not satisfied
[with U = std::vector<int>, V = std::vector<int>]
auto add_numbers(U a, V b)
     ^
main.cpp:7:10: note: because 'std::vector<int>' does not satisfy 'Number'
template<Number U, Number V>
         ^
main.cpp:5:18: note: because 'std::vector<int>' does not satisfy 'integral'
concept Number = std::integral<N> || std::floating_point<N>;
                 ^
/root/emsdk/upstream/emscripten/cache/sysroot/include/c++/v1/__concepts/arith
metic.h:26:20: note: because 'is_integral_v<std::vector<int> >' evaluated to
false
concept integral = is_integral_v<_Tp>;
                   ^
main.cpp:5:38: note: and 'std::vector<int>' does not satisfy 'floating_point'
concept Number = std::integral<N> || std::floating_point<N>;
                                     ^
/root/emsdk/upstream/emscripten/cache/sysroot/include/c++/v1/__concepts/arith
metic.h:35:26: note: because 'is_floating_point_v<std::vector<int> >'
evaluated to false
concept floating_point = is_floating_point_v<_Tp>;
                         ^
1 error generated.
```

Motivácia k použitiu

- čistejší kód
- čitateľnejší kód
- diagnostika chybových výpisov kompilátora
- precízne zachytenie myšlienky
- vymedzenie typu a kontrola v čase kompilácie

Smernice na používanie konceptov

- menovacie konvencie
- sémantická precíznosť
- využitie preddefinovaných konceptov a ďalšie

```
/* cast kodu */
template<typename T>
// predpokladame, ze operatory +, -, * a / splnaju mat.
// definicie ako komutativnost, distributivnost, ...
concept Number = requires (T a, T b)
{
    {a + b} -> convertible_to<T>;
    {a - b} -> convertible_to<T>;
    {a * b} -> convertible_to<T>;
    {a / b} -> convertible_to<T>;
};
```

Zhrnutie

- predstavenie konceptu ako plnohodnotnej súčasti jazyka ako spôsobu vymedzovania parametrizovaných typov
- predstavenie správnych spôsobov na písanie konceptov
- pokračovanie v práci: preskúmanie funkcionality kompilátora s konceptami, pôvodnej verzie konceptov alebo projektov z praxe

Ďakujem za pozornosť!

- doplňujúci materiál a odpovede na niektoré z otázok a pripomienok sú na nasledovných slidoch

Doplňujúce materiály

- chybový výpis kompilátora s použitím **statického tvrdenia** na vymedzenie typov generickej funkcie na súčet čísel s argumentami typu **vector**

```
main.cpp:7:2: error: static assertion failed due to requirement
'std::is_arithmetic<std::vector<int, std::allocator<int>>>::value'
        static_assert(
        ^
main.cpp:17:26: note: in instantiation of function template specialization
'add_numbers<std::vector<int>, std::vector<int>>' requested here
    auto assert_result = add_numbers(first, second);
                         ^
main.cpp:10:15: error: invalid operands to binary expression
('std::vector<int>' and 'std::vector<int>')
        return first + second;
               ~~~~~ ^ ~~~~~~
/root/emsdk/upstream/emscripten/cache/sysroot/include/c++/v1/__iterator/move_
iterator.h:170:1: note: candidate template ignored: could not match
'move_iterator' against 'vector'
operator+(typename move_iterator<_Iter>::difference_type __n, const
move_iterator<_Iter>& __x)
^
/root/emsdk/upstream/emscripten/cache/sysroot/include/c++/v1/__iterator/rever
se_iterator.h:219:1: note: candidate template ignored: could not match
'reverse_iterator' against 'vector'
operator+(typename reverse_iterator<_Iter>::difference_type __n, const
reverse_iterator<_Iter>& __x)
^
/root/emsdk/upstream/emscripten/cache/sysroot/include/c++/v1/__iterator/wrap_
iter.h:259:21: note: candidate template ignored: could not match
'__wrap_iter' against 'vector'
__wrap_iter<_Iter1> operator+(typename __wrap_iter<_Iter1>::difference_type
__n, __wrap_iter<_Iter1> __x) _NOEXCEPT
                    ^
2 errors generated.
```

Odpovede na otázky

Strana 8, Algoritmus 1.5:

```
... ( (U_string.compare(V_string) == 0) ||
      ((U_string.compare("i") == 0) && (V_string.compare("d") == 0)) ||
      ((U_string.compare("d") == 0) && (V_string.compare("i") == 0)) ) ...
```

Čo sa stane ak funkciu *add_numbers* zavoláte s dvomi argumentami typu *char* alebo *string*?

- neočakávané správanie (výsledok) funkcie, **chyba**

Odpovede na otázky

- ak by sme chceli byť dôslední, tak by podmienok pre typy bolo kvadraticky veľa (N^2, kde N je počet dátových typov pre čísla)
- opravená verzia časti kódu z práce

```
((U_string.compare("i") == V_string.compare("i"))
||
(U_string.compare("d") == V_string.compare("d"))
||
(U_string.compare("i") == V_string.compare("d"))
||
(U_string.compare("d") == V_string.compare("i")))
```

Odpovede na otázky

Strana 24:

template<typename U, typename V>
concept same_as = same<U, V> && same<V, U>;

Prečo je požadované *same<U, V>* aj *same<V, U>*? Nie je *same* relácia ekvivalencie?

```
template<typename U, typename V>
concept same = std::is_same_v<U, V>;

template<typename U, typename V>
concept same_as = same<U, V>;
```

```
template<typename U, typename V> requires std::same_as<U, V>
auto sum(U a, V b) {
        return a + b;
}

template<typename U, typename V>
requires std::same_as<V, U> && std::integral<U>
auto sum(U a, V b) {
        return a + b;
}
```

Poznámka z posudku

- funkcia **min** z STL a návrh skrátenej šablónovej funkcie min

```
template<class T>
const T& min(const T& a, const T& b)
{
        return a > b ? b : a;
}
```

```
const auto& min(const auto& a, const auto& b)
{
        return a > b ? b : a;
}
```

Experimenty

- testovanie časovej efektivity bolo realizované na jednoduchom teste kontroly typu – či je funkcia volaná so správnym typovým parametrom
- 30 behov programu; údaje v tabuľke sú priemerné časy udané v ms

Spôsob kontroly typu	REAL-TIME	EXECUTION-TIME (-O0) – 1 mil.	EXECUTION-TIME (-O1) – 10 mil.	EXECUTION-TIME (-O2) – 1 mld.	EXECUTION-TIME (-O3) – 10 mld.
typeid	659	2255	2914	3179	29596
static assert	660	2243	2852	825	8073
SFINAE	667	2246	2851	829	8092
koncept	665	2248	2847	826	8093