Gmail 黃詠翔 <jamesbond0705@gmail.com>

(Long list) Incorrect Statements and Typos

黃詠翔 <jamesbond0705@gmail.com> 2023年2月19日 晚上7:55
收件者: tmplbook2@josuttis.com

Dear authors,

After reading your amazing book thoroughly, I found some new incorrect statements and 45+ typos in your template book (printed version), which have NOT been documented in http://www.tmplbook.com/errata2_1.html:

Errors, Incorrect Statements/Results:

Section 19.5 IsConvertibleT

(Page 430, Handling Special Cases)

The statements about void types have some typos/flaws:

1. the 3rd item/ **Conversion to** (…) **void** types should yield **true**./ **Conversion to** (…) **void** types should yield **false, except when both FROM and TO are (const/volatile-qualified) void**./
2. In line -7, s/ or if **FROM** is void and **TO** is not/ or if **TO** is void and **FROM** is not/ (Reason: the original statement "... and TO is not," is logically included in the previous case "IsVoidT<TO> is false")

myTest to ensure that it yields true only when both TO and FROM are void (gcc, clang): https://godbolt.org/z/beY5Pc7sY

19.6.4 Using Generic Lambdas to Detect Members

(Page 439, traits/isvalid1.cpp)

1. `hasLess(42, type<char>) and hasLess(type<string>, "hello")` yield **false**, **not true.**

myTest: https://godbolt.org/z/GsTb7r6Kh

The reason is that 42 and "hello" cannot convert to the arguments of valueT, whose types are TypeT<>. Hence isValidImpl SFINAE out the template with true_type and yields the one with false_type.

2. Moreover, the paragraph in p.439 says **hasSizeType uses std::decay.** However, in the code, **it does NOT**. Therefore, I suggest the following correction in **traits/isvalid1.cpp:**

```
hasSizeType
   = isValid([](auto x) -> typename std::decay_t<decltype(valueT(x))>::size_type {
         });
```

Section 25.1 Basic Tuple Design

1. In the beginning sentence of Page 579, it mentions a code example **tuples/tuple.hpp**, but I **cannot find it in** http://www.tmplbook.com/code/toc.html.

2. In **tuples/tupleget.hpp (Page 577), the functions apply(...) and get(...)'s return types use "auto"**, but I think **we should use auto&& or decltype(auto) so that we can get a reference to tuple's element.**

myTest (gcc & clang): https://godbolt.org/z/ja8jhYT7r

Appendix D: Standard Type Utilities

(Page 719)

In line 5 of main() function, is_**trivially**_copy_constructible_v<C> should be **false**, **not true**.

myTest (gcc & clang) : https://godbolt.org/z/Wbb6b8EMr

(Page 730, add_lvalue_reference)

In line 6 of 1st code snippet, **int &&** should be **int const &&**.

myTest (gcc & clang) : https://godbolt.org/z/57MTs6vo5

Typos:

1.5 Overloading Function Templates

(Page 19, line 3)

s/ There temporaries are created for the arguments (7, 42, and 68)/ **Their** temporaries are created for the arguments (7, 42, and 68)/

11. Generic Libraries

(Page 170)

1st code snippet/ ~~**int**~~ **int** size = 10/

(Page 171, 2 lines before 11.5)

s/ see **Section** 7/ see **Chapter** 7/

(Page 172, end of 2nd paragraph)

s/ declaration of struct **node**/ declaration of struct **Node**/ (capitalize)

12.2 Template Parameters

(page 188, line 2)

s/ These names **be** used/ These names **can be** used/

12.4 Variadic Templates

(page 203, line 7)

s/ page 207./ page 207)./ (Missing a right parenthesis)

13.3 Parsing Templates

(page 230)

line 1, s/ are disallowed by the **second** item/ are disallowed by the **third** item/

Reason: The keyword "base" in the next sentence appeared in the "3." item in previous page 229.

Similarly, we have to correct:

line 8, s/ it satisfies the first **two** rules/ it satisfies the first **three** rules/ (1., 2. & 3. in page 229)

(page 232, line -3)

code/ using Magic = ...:: template Magic**<T>**;/ using Magic = ...:: template Magic**<U>**;/

14.5 Explicit Instantiation

(page 260, 1st code snippet)

1. code/ ~~template void f(char)~~/
2. code/ // **four** valid explicit instantiations/ // **three** valid explicit instantiations/

Reason:

template void f(long) and ***template void f(char)*** are duplicated because they come from the same prototype of void f(T). By 1st edition's section 10.5 (page 159), we can figure out this duplication comes from the pre-C++11 dynamic exception specification: "throw(T)".

Section 14.6 Compile-Time if Statements

(Page 264, code snippet)

Before **static bool f(T p)**, we should add **template<typename T>**, to declare the parameter **T** (2 places).

15.1 The Deduction Process

(page 271, line 3)

s/ calling **g(x)**/ calling **g(arr)**/

15.5 Parameter Packs

(page 276, line -7)

s/ (e.g., **Types** for **h1()**)/ (e.g., **Types** for **f1()**)/

15.6 Rvalue References

(page 277, line -1)

s/ ref2ref = **i**;/ ref2ref = **r**;/

15.10 Deduction from Initializers and Expressions

(page 294)

1. **line 3,** s/ Section 15.10.**2** on page **298**/ Section 15.10.**3** on page **301**/ *Reason*: decltypo(auto)
2. **line -2,** s/ <typename **t**>/ <typename **T**>/ (capitalize t)

(page 295)

auto const ~~**S::***~~**pm** = &X<int>::m

Reason: The ***variable name* S::*pm** is invalid, myTest: https://godbolt.org/z/67nanoaGM

(page 304, line 3)

s/ initializing **both ~~both~~ oops and val**/

15.11 Alias Templates

(page 313, deduce/aliastemplate.cpp)

code/ f3(Stack<T, deq<T>)/ f3(Stack<T, deq<T>>)/ (Missing a right angle bracket)

15.12 Class Template Argument Deduction

(page 320, line 3)

1. s/ Tuple<Tuple<int, int>/Tuple<Tuple<int, int>>/ (Missing the ending right angle bracket)
2. s/ Tuple<Tuple<**int**>>/ Tuple<Tuple<**int, int**>>/ (2 places)

16.5 Explicit Instantiation

(page 351, **1st line of 16.5** & **code snippet**)

1. s/ draft C++**11** standard/ draft C++**14** standard/
2. s/ **null_ptr**/ **nullptr**/

19.4 SFINAE-Based Traits

(Page 420, **line 3 & 6**)

s/ Is**Convertible**T<…>/ Is**DefaultConstructible**T<…>/ (2 places)

(Page 421)

s/ looks **more** condensed **that** the first approach/ looks **more** condensed **than** the first approach/

(Page 426, 2 lines before code snippet *traits/hasplus.hpp*)

s/ Following the example of **HasLess**T described in the **previous** section/ Following the example of **IsDefaultConstructible**T described in the **previous** section/

Reason:

HasLessT's definition appears in the later section 19.6.3 (Page 436). By reading all the stuff in Section 19.4, I guess what you want to mention here is the one appeared in the beginning of 19.4.2 (Page 420).

(Page 428, line 1)

In line 1 of code snippet, bool = HasMemberT_value_type<C> is undefined until the later section 19.6.3, page 434. I suggest to point out this in an extra footnote.

19.5 IsConvertibleT

(Page 428)

This is a further correction to the existed typo:

1. code snippet/ <typename F, **typename ~~T~~**, …> /<typename F, **~~typename T~~**, …>/
2. code snippet (test() fallback)/ <typename**~~, typename~~**>/

myTest (gcc 12.2& clang 15.0.0): https://godbolt.org/z/x5a3csbWr

19.7 Other Traits Techniques

(Page 442, line 3)

s/ so **neither**…, **or** the program is likely/ so **neither**…, **nor** the program is likely/

(Page 443, end of 1st & last code snippet)

In the end of 1st code snippet for **UnsignedT**, remove the redundant "::Type" in the end of using declaration. (It's redundant since MakeUnsignedT<T>::Type has applied it.)

(Page 446, 1 line above the title: Alias Templates and Traits)
our **MemberPointerToIntT** example is **not found** in the whole book.

19.8 Type Classification

(Page 451, line 6)

s/ std::is_**integral**/ std::is_**arithmetic**/

Reason: Integral is a primary type category, not a composite one. On the other hand, Arithmetic is a composite type category.

21.1 The Empty Base Class Optimization (EBCO)

(Page 494, line 4)

s/ Section **25.1.1** on page **576**/ Section **25.5.1** on page **593**/

21.4 Named Template Arguments

(Page 515, 5 lines before Section 21.5)

It mentions a code example **inherit/namedtmpl.cpp**, but I **cannot find it in** http://www.tmplbook.com/code/toc.html

Although I can find it in the first edition's website: http://www.josuttis.com/tmplbook/toc.html

22.1 Function Objects, Pointers, and std::function<>

(Page 518)

s/ but if the **template were** large,/ but if the **templates were** large,/

23 Metaprogramming

(Page 535, line -4)

s/ has type Ratio<2003, **2000**>/ has type Ratio<2003, **3000**>/

myTest: https://godbolt.org/z/v5PqGshcT

(Page 543, Table 23.1's line -2)

s/ <Doublify<double, double>,/ **Doublify**<Doublify<double, double>,/

(Missing the begining Doublify, *Reason:* This line should equal to the first line of Trouble<2>::LongType)

24.2 Typelist Algorithms

(Page 561)

s/ an empty typelist(**TypeList<T>**)/ an empty typelist(**Typelist<~~T~~>**)/ (uncapitalize "L" and delete "T")

25.3 Tuple Algorithms

(Page 590, tuples/tuplesorttest)

s/ // t2 is Tuple<int, long, **std::string>**/ // t2 is Tuple<int, long, **std::string, std::complex<double>**>/

25.5 Optimizing Tuple

(Page 598, line 1)

s/ but **it's** recursive implementation **requires**…/ but **its** recursive implementation **requires**…/ (remove ')

26 Discriminated Unions

(Page 606, 3 lines before title 26.2)

s/ getBufferAs(), **it** sufficient for.../ getBufferAs(), **it's** sufficient for.../ (missing the verb 's)

Appendix C.2

(Page 687, code snippet)

Change the struct name "**Value**" to "**X**" to match the definition of g(X&& x). (3 places.)

Appendix D.3

(Page 715, last line before std::rank<T>::value)

code snippet/ // **yield1**/ // **yield 1**/ (missing a blank)

(Page 717, std::result_of)

1. s/ result_of<**T, Args…**>/ result_of<**T(Args…)**>/
2. 6th bullet/ provides... **such the** easier syntax/ provides... **such as an** easier syntax/

Appendix D.4

(Page 729, std::make_**sign**ed)
In 3rd bullet, **remove** the **unrelated** sentence: "whereas a non-**const** pointer…is not **const-qualified.**"

Appendix D.6

(Page 735, one line before the last code snippet)

s/ neither classes **not** unions/ neither classes **nor** unions/

Best regards,

Yung-Hsiang Huang