

Doc. no. WG21/N2112=/06-0182
Date: 2006-11-03
Reply-To: Gabriel Dos Reis
gdr@cs.tamu.edu

Templates Aliases

Gabriel Dos Reis

Bjarne Stroustrup

Revision 2

This note is an expansion on the proposal [2] to add template aliases to C++. It further discusses the fundamental issue, the solution suggested in [2] and proposed several generalizations to non-template settings.

1 Syntax and semantics

1.1 Non-parameterized type alias

Grammar modification. Change the production *block-declaration* in §7/1 to read

1 [...]

block-declaration:
simple-declaration
asm-definition
namespace-alias-definition
using-declaration
using-directive
alias-declaration

alias-declaration:
using *identifier* = *type-id*

Paragraph extension. Add to paragraph §7.1.3/1 (“The `typedef` specifier”), the following paragraph:

1 [...]

A *typedef-name* can also be introduced by an *alias-declaration*. The *identifier* following the `using` keyword becomes a *typedef-name*. It has the same semantics as if it were introduced by the `typedef` specifier. In particular, it does not define a new type and it cannot appear in the *type-id*.
[Example:

```

using handler_t = void (*)(int);
extern handler_t ignore;           // #1
extern void (*ignore)(int);       // redeclare #1
using cell = pair<void*, cell*>;  // ill-formed

```

—end example]

1.2 Parameterized type alias

Paragraph modification. Change paragraph §14/1 from

- 1 A *template* defines a family of classes or functions.

[Grammar elided]

The *declaration* in a *template-declaration* shall

- declare or define a function or a class, or
- define a member function, a member class or a static data member of a class template or of a class nested within a class template, or
- define a member template of a class or class template.

to

- 1 A *template* defines a family of classes, or functions, **or an alias for a family of types.**

[Grammar elided]

The *declaration* in a *template-declaration* shall

- declare or define a function or a class, or
- define a member function, a member class or a static data member of a class template or of a class nested within a class template, or
- define a member template of a class or class template, or
- **be an *alias-declaration*.**

New paragraph. Add a new paragraph §14/10

- 10 A *template-declaration* that declares a template alias (14.5.6) shall not be **exported**.

New paragraph. Add a new paragraph §14.2/7

- 7 A *template-id* that names a template alias specialization is a *type-name*.

Paragraph modification. Change §14.3.3/1 from

- 1 A *template-argument* for a template *template-parameter* shall be the name of a class template, expressed as an *id-expression*. Only primary class templates are considered when matching the template template argument with the corresponding, [...]

to

- 1 A *template-argument* for a template *template-parameter* shall be the name of a class template **or a template alias**, expressed as an *id-expression*. **When the *template-argument* names a class template**, only primary class templates are considered when matching the template template argument with the corresponding, [...] [...]

Paragraph modification. Change paragraph §14.4/1 from

- 1 Two *template-ids* refer to the same class or function if their template names are identical, they refer to the same template, their type *template-arguments* are the same type, their non-type *template-arguments* of integral or enumeration type have identical values, their non-type *template-arguments* of pointer type or reference type refer to the same external object of unction, and their template *template-arguments* refer to the same template. [Example:

```
template<class E, int size> class buffer { /* ... */ };
buffer<char,2*512> x;
buffer<char,1024> y;
```

declares *x* and *y* to be of the same type, and

```
template<class T, void(*err_fct)()> class list> { /* ... */ };
list<int,&error_handler1> x1;
list<int,&error_handler2> x2;
list<int,&error_handler2> x3;
list<char,&error_handler1> x4;
```

declares *x2* and *x3* to be of the same type. Their type differs from the types of *x1* and *x4*. —end example]

to

- 1 Two *template-ids* refer to the same **class type** or function if
 - **their respective** template names are identical **and** they refer to the same template; **or at least one of the template names refers to a template alias, in which case the *template-id* is replaced (for the purpose of determining equivalence) by the corresponding *type-id* (14.5.6) and the current rules are applied with the replacements; and**
 - their type *template-arguments* are the same type, **and**
 - their non-type *template-arguments* of integral or enumeration type have identical values, **and**
 - their non-type *template-arguments* of pointer type or reference type refer to the same external object or function, **and**

— and their template *template-arguments* refer to the same template.

[Example:

```
template<class E, int size> class buffer { /* ... */ };
buffer<char,2*512> x;
buffer<char,1024> y;
```

declares `x` and `y` to be of the same type, and

```
template<class T, void(*err_fct) ()> class list> { /* ... */ };
list<int,&error_handler1> x1;
list<int,&error_handler2> x2;
list<int,&error_handler2> x3;
list<char,&error_handler1> x4;
```

declares `x2` and `x3` to be of the same type. Their type differs from the types of `x1` and `x4`.

```
template<template<class> class TT> struct X { };
template<class> struct Y { };
template<class T> using Z = Y<T>;
X<Y> y;
X<Z> z;
```

declares `y` and `z` to be of the same type. —end example]

New paragraph. Add a new paragraph §14.5/3

- 3 Because an *alias-declaration* cannot declare a *template-id*, it is not possible to partially or explicitly specialize a template alias.

New subsection. Add a new subsection §14.5.6 titled “Template aliases”

- 1 A *template alias* declares a name for a family of types. The name of the template alias is a *template-name*.
- 2 When a *template-id* refers to the specialization of a template alias, it is syntactically equivalent to the associated type obtained by substitution of its *template-arguments* for the *template-parameters* in the *type-id* of the template alias. [Example:

```
template<class T> struct Alloc { /* ... */ };
template<class T>
    using Vec = vector<T, Alloc<T>>;
Vec<int> v;    // same as vector<int, Alloc<int>> v;
```

```
template<class T>
    void process (Vec<T>& v)
    { /* ... */ }
```

```
template<class T>
    void process (vector<T, Alloc<T> >& w)
    { /* ... */ }           // error: redefinition
```

—end example]

References

- [1] Walter E. Brown, *A Case for Template Aliasing*, document no. WG21/N1451=J16/03-0034.
- [2] Gabriel Dos Reis and Mat Marcus, *Proposal to add template aliases to C++*, document no. WG21/N1449=J16/03-0032.
- [3] Bjarne Stroustrup and Gabriel Dos Reis, *Template aliases for C++*, document no. nxxxx=03-yyyy.
- [4] Herb Sutter, *Typedef templates*, document no. WG21/1406.