Document No: N3112 = 10-0102
Date: 2010-08-06
Project: Programming Language C++
References: WG21 N3092, SC 22 N4512: ISO/IEC FCD 14882
Reply to: Nicolai Josuttis
nico@josuttis.de

Proposed Resolution for CH 15: Double check copy and move semantics of classes due to new rules for default move constructors and assignment operators

Rationale

In Pittsburgh the default semantics of move constructors and move assignment operators changed. This might have the effect that accidentally we enable or disable move and/or copy semantics unintentionally.

In the library we found the following places where copy and move semantics are inconsistently specified and discussed them in the library group. The following table gives an overview of the result:

Red flags indicate places where we have to fix something.

Yellow flags indicate places where we don't *have to* fix something. But to clarify we might add explicit statements (usually with =delete) indicating that we disable copying/moving by intention. The proposed solution in this paper contains proposed wording for these places.

Green flags indicate places where we intentionally have different semantics and don't have to do anything.

Note that the necessary fix for pair<> in Section 20.3.5.2, will be covered by a different proposal by Daniel Krügler covering a couple of issues with pair and tuple.

Note that the necessary fix for atomic_future<> will be covered by the Concurrency group.

Note also that this proposed resolution adds missing constructor descriptions for stack<>.

Section	Class	C(const&)	C(&&)	op=(const&)	op=(&&)
20.3.5.2	pair<>	yes			yes
20.8.4	reference_wrapper<>	yes		yes	
20.9.6	scoped_allocator_adaptor<>	yes			
20.9.10.2	unique_ptr<>	delete	yes	delete	yes
20.9.10.3	unique_ptr<T[], D>	delete	yes	delete	yes
23.3.5.1	queue<>		yes		yes
23.3.5.2	priority_queue<>		yes		yes
23.3.5.3	stack<>		yes		yes
27.5.2	ios_base	delete		delete	
27.5.4	basic_ios<>	delete		delete	
27.7.1.1	basic_istream<>		protected		protected
27.7.1.5	basic_iostream<>		protected		protected
27.7.2.1	basic_ostream<>		protected		protected
27.7.2.4	basic_ostream<charT,traits>::sentry	delete		delete	
27.8.2	basic_istringstream<>		yes		yes
27.8.3	basic_ostringstream<>		yes		yes
27.8.4	basic_stringstream<>		yes		yes
27.9.1.6	basic_ifstream<>		yes		yes
27.9.1.10	basic_ofstream<>		yes		yes
27.9.1.14	basic_fstream<>		yes		yes
27.6.2	basic_streambuf<>	protected		protected	
27.8.1	basic_stringbuf<>		yes		yes
27.9.1.1	basic_filebuf<>		yes		yes
30.4	once_flag	delete		delete	
30.6.8	atomic_future<>	yes		yes	

Proposed Wording for pair<>

None (this part of the resolution will be covered by Daniel Krügler in a separate paper)

Proposed Wording for Scoped Allocator Adaptor

In **20.9.6 Scoped allocator adaptor** in the declaration of class scoped_allocator_adaptor after

```
scoped_allocator_adaptor(const scoped_allocator_adaptor& other);
```

add

```
scoped_allocator_adaptor(scoped_allocator_adaptor&& other);
```

In **20.9.6.2 Scoped allocator adaptor** constructors
after § 4 (copy constructor)
add:

```
scoped_allocator_adaptor(scoped_allocator_adaptor&& other);
```

Effects: Move constructs each allocator within the adaptor with the corresponding allocator from other.

Editorial comment:
In 20.9.6.2 §4 replace “intializes” by “initializes”.

Proposed Wording for Container Adaptors

In **23.3.5.1.1 queue definition**
strike

```
queue(queue&& q);
```

and **strike**

```
queue& operator=(queue&& q);
```

In **23.3.5.1.2 queue constructors**
strike:

```
queue(queue&& q);
```

3 *Effects:* Initializes c with std::move(q.c).

```
queue& operator=(queue&& q);
```

4 *Effects:* Assigns std::move(q.c) to c.

5 *Returns:* *this.

Editorial comment:
In 23.3.5.1.2 §1 replace “Initialzies” by “Initializes”.

In **23.3.5.2 Class template priority_queue**
strike
priority_queue(priority_queue&&);
and **strike**
priority_queue& operator=(priority_queue&&);

In **23.3.5.2.1 priority_queue constructors**
strike:

priority_queue(priority_queue&& q);

5 Effects: Initializes c with std::move(q.c) and initializes comp with std::move(q.comp).

priority_queue& operator=(priority_queue&& q);

6 Effects: Assigns std::move(q.c) to c and assigns std::move(q.comp) to comp.

7 Returns: *this.

In **23.3.5.3.1 stack definition**
strike
stack(stack&&s);
and **strike**
stack& operator=(stack&& s);

In **23.3.5.1.2 stack constructors**
strike:

stack(stack&& s);

Effects: Initializes c with std::move(s.c).

stack& operator=(stack&& s);

1 Effects: Assigns std::move(s.c) to c.

2 Returns: *this.

And **add**:

```
explicit stack(const Container& cont);
```

1 Effects: Initializes c with `cont`.

```
explicit stack(Container&& cont = Container());
```

2 Effects: Initializes c with `std::move(cont)`.

Proposed Wording for IO-Streams

In **27.5.2 Class ios_base**
after
```
    ios_base(const ios_base&) = delete;
```
add:
```
    ios_base(ios_base&&) = delete;
```
and after:
```
    ios_base& operator=(const ios_base&) = delete;
```
add:
```
    ios_base& operator=(ios_base&&) = delete;
```

In **27.5.4 Class template basic_ios**
after
```
    basic_ios(const basic_ios& ) = delete;
```
add:
```
    basic_ios(basic_ios&&) = delete;
```
and after:
```
    basic_ios& operator=(const basic_ios&) = delete;
```
add:
```
    basic_ios& operator=(basic_ios&&) = delete;
```

In **27.7.1.1 Class template basic_istream**
before
```
    basic_istream(basic_istream&& rhs);
```
add:
```
    basic_istream(const basic_istream& rhs) = delete;
```
and before:
```
    basic_istream& operator=(basic_istream&& rhs);
```
add:
```
    basic_istream& operator=(const basic_istream& rhs) = delete;
```

In **27.7.1.5 Class template basic_iostream**
before
```
    basic_iostream(basic_iostream&& rhs);
```
add:
```
    basic_iostream(const basic_iostream& rhs) = delete;
```
and before:
```
    basic_iostream& operator=(basic_iostream&& rhs);
```
add:
```
    basic_iostream& operator=(const basic_iostream& rhs) = delete;
```

In **27.7.2.1 Class template basic_ostream**
before
```
basic_ostream(basic_ostream&& rhs);
```
add:
```
basic_ostream(const basic_ostream& rhs) = delete;
```
and before:
```
basic_ostream& operator=(basic_ostream&& rhs);
```
add:
```
basic_ostream& operator=(const basic_ostream& rhs) = delete;
```

In **27.7.2.4 Class basic_ostream::sentry**
after
```
sentry(const sentry&) = delete;
```
add:
```
sentry(sentry&&) = delete;
```
and after:
```
sentry& operator=(const sentry&) = delete;
```
add:
```
sentry& operator=(sentry&&) = delete;
```

In **27.8.2 Class template basic_istringstream**
before
```
basic_istringstream(basic_istringstream&& rhs);
```
add:
```
basic_istringstream(const basic_istringstream& rhs) = delete;
```
and before:
```
basic_istringstream& operator=(basic_istringstream&& rhs);
```
add:
```
basic_istringstream& operator=(const basic_istringstream& rhs) = delete;
```

In **27.8.3 Class template basic_ostringstream**
before
```
basic_ostringstream(basic_ostringstream&& rhs);
```
add:
```
basic_ostringstream(const basic_ostringstream& rhs) = delete;
```
and before:
```
basic_ostringstream& operator=(basic_ostringstream&& rhs);
```
add:
```
basic_ostringstream& operator=(const basic_ostringstream& rhs) = delete;
```

In **27.8.4 Class template basic_stringstream**
before
```
basic_stringstream(basic_stringstream&& rhs);
```
add:
```
basic_stringstream(const basic_stringstream& rhs) = delete;
```
and before:
```
basic_stringstream& operator=(basic_stringstream&& rhs);
```
add:
```
basic_stringstream& operator=(const basic_stringstream& rhs) = delete;
```

In **27.9.1.6 Class template basic_ifstream**
before
```
basic_ifstream(basic_ifstream&& rhs);
```
add:
```
basic_ifstream(const basic_ifstream& rhs) = delete;
```
and before:
```
basic_ifstream& operator=(basic_ifstream&& rhs);
```
add:
```
basic_ifstream& operator=(const basic_ifstream& rhs) = delete;
```

In **27.9.1.10 Class template basic_ofstream**
before
```
basic_ofstream(basic_ofstream&& rhs);
```
add:
```
basic_ofstream(const basic_ofstream& rhs) = delete;
```
and before:
```
basic_ofstream& operator=(basic_ofstream&& rhs);
```
add:
```
basic_ofstream& operator=(const basic_ofstream& rhs) = delete;
```

In **27.9.1.14 Class template basic_fstream**
before
```
basic_fstream(basic_fstream&& rhs);
```
add:
```
basic_fstream(const basic_fstream& rhs) = delete;
```
and before:
```
basic_fstream& operator=(basic_fstream&& rhs);
```
add:
```
basic_fstream& operator=(const basic_fstream& rhs) = delete;
```

In **27.6.2 Class template basic_streambuf<charT,traits>**
after
```
  basic_streambuf(const basic_streambuf& rhs);
```
add:
```
  basic_streambuf(basic_streambuf&& rhs) = delete;
```
and after:
```
  basic_streambuf& operator=(const basic_streambuf& rhs);
```
add:
```
  basic_streambuf& operator=(basic_streambuf&& rhs) = delete;
```

In **27.8.1 Class template basic_stringbuf**
before
```
  basic_stringbuf(basic_stringbuf&& rhs);
```
add:
```
  basic_stringbuf(const basic_stringbuf& rhs) = delete;
```
and before:
```
  basic_stringbuf& operator=(basic_stringbuf&& rhs);
```
add:
```
  basic_stringbuf& operator=(const basic_stringbuf& rhs) = delete;
```

In **27.9.1.1 Class template basic_filebuf**
before
```
  basic_filebuf(basic_filebuf&& rhs);
```
add:
```
  basic_filebuf(const basic_filebuf& rhs) = delete;
```
and before:
```
  basic_filebuf& operator=(basic_filebuf&& rhs);
```
add:
```
  basic_filebuf& operator=(const basic_filebuf& rhs) = delete;
```

Proposed Wording for once_flag

In **30.4 Mutual exclusion**, header <mutex> synopsis, struct once_flag
after:

```
once_flag(const once_flag&) = delete;
```

add:

```
once_flag(once_flag&&) = delete;
```

and after:

```
once_flag& operator=(const once_flag&) = delete;
```

add:

```
once_flag& operator=(once_flag&&) = delete;
```

Proposed Wording for atomic_future<>

None (this part of the resolution will be covered by the Concurrency group in a separate paper)