Document: N2590=08-0100
Date: 2008-03-17
Reply to: Alisdair Meredith <alisdair.meredith@codegear.com>

Simplifying `swap` overloads

Basic Issue

Since rvalue references have been added to the language, the canonical form of `swap` in the standard library has changed. In 'classic' C++03 most library types provide a `swap` member function taking a (lvalue) reference where it can be implemented efficiently, and the `swap` algorithm is overloaded taking two (lvalue) references.

```
class X {
  void swap( X & );
};

void swap( X&, X& );
```

In C++0x the canonical form changes the member function to take an rvalue reference, and provides two further overloads (for each type) taking an lvalue reference and a rvalue reference. There is no fourth form taking two rvalue references in an attempt to alert the user of likely errors in their code. It is expected that where two rvalues are swapped the code has no effect, and removing this overload alerts the user to what is *probably* an error in their code.

```
class X {
  void swap( X && );
};

void swap( X&,  X&  );
void swap( X&&, X&  );
void swap( X&,  X&& );
void swap( X&&, X&& ); // NOT supplied
```

The Problem

The problem with this solution is that it is moving complexity from the user code into the library for the sake of diagnosing a rare error condition – and one that may not always be an error. This may sound like a small price to pay, but the rising number of defect reports on class templates where these overloads are missing suggests otherwise.

Note that this also pushes the problem at any 3rd party library vendors, who would now become expected to provide all three overloads for their own library types, if they offer one today.

It appears that the current cure may be worse than the problem it is intended to solve.

Proposed Solution

With the anticipated addition of concepts to the language, overload the `swap` algorithm for types that match a new concept of `MemberSwappable`. This allows a single overload to automatically pick up all types in the library that support extra-efficient `swap`.

Note that this would also automatically pick up member-swap for user defined container types. This is most likely what the user intended, although risks changed meaning for existing code. The risk should also be balanced against asking users to provide an additional two overloads for all their custom `swap` functions.

Note that the basic `swap` function template itself does not provide the triplicate of rvalue overloads, so cannot be used with temporary objects in the general case. This resolution would also extend that support.

```
class X {
  void swap( X && ); // Update any classes missing rvalue form
};

void swap( X&,  X&  );
void swap( X&&, X&  );
void swap( X&,  X&& );
void swap( X&&, X&& ); // NOT supplied
```

Fallback Solution

Replace all `swap` overload triples with a single swap function (template) taking two rvalue-reference arguments. Note that if the Evolution Working Group and Core accept paper n2584 these can be easily specified as `=default` in the synopsis, still removing the need for descriptive text.

```
class X {
  void swap( X && ); // Update any classes missing rvalue form
};

~~void swap( X&,  X&  );~~
~~void swap( X&&, X&  );~~
~~void swap( X&,  X&& );~~
void swap( X&&, X&& ); // Replace 3 overloads with one again
```

Defect Reports

This paper resolves a number of existing library issues, and a few unreported issues besides, it is recommended that a revised version reflecting the state of both concepts and LWG opinion be adopted. This paper aims for a simpler, self-consistent resolution than driving the issues list directly, and the author volunteers to update the paper accordingly.

The proposed wording below would resolve the following LWG issues as NAD Editorial

522 tuple swap (Open)
743 shared_ptr swap (Ready)
770 function swap (Ready)
774 member swap undefined for most containers (Open)

It also avoids opening similar issues for all the unordered containers and regex classes, and fixes a missing rvalue reference in std::set.

It does not attempt to resolve LWG issue 809 swap should be overloaded on array types (New)

Proposed Wording

Note : all wording has been written against working paper N2521, but assuming swap has moved to clause 20 as per the Core Concepts for Library paper N2502. It is likely that both these reference documents will be updated in this same mailing.

Add to <concepts> synopsis (n2502)

```
auto concept MemberSwappable<ClassType T> see below;

template< MemberSwappable T >
void swap( T&& a, T && b );
```

Add to **20.1.5 Copy and move [concept.copymove]** (n2502)

```
template< ClassType T >
auto concept MemberSwappable {
  void T::swap( T && );
};
```

Remark: Calling swap on an object shall exchange its value with that of the passed argument.

```
template< MemberSwappable T >
void swap( T&& a, T && b );
```

Effects: Calls `a.swap( b );`

Remove all overloads of the free-function `swap` overloads in all chapters, and update all member-swap functions to take an rvalue-reference argument. Provide member-swap functions where missing.

From **20.2 [utility]**, <utility> synopsis remove

~~template <class T1, class T2>~~
~~void swap(pair<T1,T2>&, pair<T1,T2>&);~~
~~template <class T1, class T2>~~
~~void swap(pair<T1,T2>&&, pair<T1,T2>&);~~
~~template <class T1, class T2>~~
~~void swap(pair<T1,T2>&, pair<T1,T2>&&);~~

After **20.2.3 p16 [pairs]** remove

~~template<class T1, class T2> void swap(pair<T1, T2>& x, pair<T1, T2>& y);~~
~~template<class T1, class T2> void swap(pair<T1, T2>&& x, pair<T1, T2>& y);~~
~~template<class T1, class T2> void swap(pair<T1, T2>& x, pair<T1, T2>&& y);~~
~~17 *Effects:* x.swap(y)~~

Add this signature to **20.3.1 [tuple.tuple]**

```
void swap(tuple&&);
```

Add the following section to the end of the tuple clause

20.3.1.7 tuple swap [tuple.swap]

```
void swap(tuple&& rhs);
```

Requires: Each type in `Types` shall be `Swappable`.

Effects: Calls `swap` for each element in `*this` and its corresponding element in `rhs`.

Throws: Nothing, unless one of the element-wise `swap` calls throw an exception.

From **20.5 p2 [function.objects]** <functional> synopsis remove

~~template<class R, class... ArgTypes>~~
~~void swap(function<R(ArgTypes...)>&, function<R(ArgTypes...)>&);~~

From **20.5.15.2 [func.wrap.func]** remove

~~template <class R, class... ArgTypes>~~
~~void swap(function<R(ArgTypes...)>&, function<R(ArgTypes...)>&);~~

Update member-swap signature in **20.5.15.2 [func.wrap.func]**

```
// 20.5.15.2.2, function modifiers:
void swap(function&&);
```

Update **20.5.15.2.2 function modifiers [func.wrap.func.mod]**

```
void swap(function&& other);
```

1 *Effects:* interchanges the targets of `*this` and `other`.

2 *Throws:* nothing.

Remove

~~**20.5.15.2.7 specialized algorithms [func.wrap.func.alg]**~~
~~template<class R, class... ArgTypes>~~
~~void swap(function<R(ArgTypes...)>& f1, function<R(ArgTypes...)>& f2);~~
~~1 *Effects:* f1.swap(f2);~~

From **20.6 p1 [memory]** <memory> synopsis remove

~~// 20.6.6.2.9, shared_ptr specialized algorithms:~~
~~template<class T> void swap(shared_ptr<T>& a, shared_ptr<T>& b);~~

…

~~// 20.6.6.3.7, weak_ptr specialized algorithms:~~
~~template<class T> void swap(weak_ptr<T>& a, weak_ptr<T>& b);~~

From **20.6.5 [unique.ptr]** remove

~~template<class T, class D> void swap(unique_ptr<T, D>& x, unique_ptr<T, D>& y);~~
~~template<class T, class D> void swap(unique_ptr<T, D>&& x, unique_ptr<T, D>& y);~~
~~template<class T, class D> void swap(unique_ptr<T, D>& x, unique_ptr<T, D>&& y);~~

Remove

~~**20.6.5.5** `unique_ptr` **specialized algorithms [unique.ptr.special]**~~

```
template <class T, class D> void swap(unique_ptr<T, D>& x, unique_ptr<T, D>& y);
template <class T, class D> void swap(unique_ptr<T, D>&& x, unique_ptr<T, D>& y);
template <class T, class D> void swap(unique_ptr<T, D>& x, unique_ptr<T, D>& y);
```

~~1 *Effects:* Calls `x.swap(y)`.~~

Update member-swap signature in **20.6.6.2 [util.smartptr.shared]**

```
// 20.6.6.2.4, modifiers:
void swap(shared_ptr&& r)
```

From **20.6.6.2 [util.smartptr.shared]** remove

```
// 20.6.6.2.9, shared_ptr specialized algorithms:
template<class T> void swap(shared_ptr<T>& a, shared_ptr<T>& b);
```

Update **20.6.6.2.4** `shared_ptr` **modifiers [util.smartptr.shared.mod]**

```
void swap(shared_ptr&& r);
```

1 *Effects:* Exchanges the contents of `*this` and `r`.

2 *Throws:* nothing.

Remove

~~**20.6.6.2.9** `shared_ptr` **specialized algorithms [util.smartptr.shared.spec]**~~

```
template<class T> void swap(shared_ptr<T>& a, shared_ptr<T>& b);
```

~~1 *Effects:* Equivalent to `a.swap(b)`.~~

~~2 *Throws:* nothing.~~

From **20.6.6.3 [util.smartptr.weak]** remove

```
// specialized algorithms
template<class T> void swap(weak_ptr<T>& a, weak_ptr<T>& b);
```

Remove

~~**20.6.6.3.7** `weak_ptr` **specialized algorithms [util.smartptr.weak.spec]**~~

```
template<class T> void swap(weak_ptr<T>& a, weak_ptr<T>& b)
```

~~1 *Effects:* Equivalent to `a.swap(b)`.~~

~~2 *Throws:* nothing.~~

From **21.2 [string.classes]** remove

```
// 21.3.8.8: swap
template<class charT, class traits, class Allocator>
  void swap(basic_string<charT,traits,Allocator>& lhs,
            basic_string<charT,traits,Allocator>& rhs);
template<class charT, class traits, class Allocator>
  void swap(basic_string<charT,traits,Allocator>&& lhs,
            basic_string<charT,traits,Allocator>& rhs);
template<class charT, class traits, class Allocator>
  void swap(basic_string<charT,traits,Allocator>& lhs,
            basic_string<charT,traits,Allocator>&& rhs);
```

Remove

~~**21.3.8.8 swap [string.special]**~~

```
template<class charT, class traits, class Allocator>
  void swap(basic_string<charT,traits,Allocator>& lhs,
            basic_string<charT,traits,Allocator>& rhs);
template<class charT, class traits, class Allocator>
  void swap(basic_string<charT,traits,Allocator>&& lhs,
            basic_string<charT,traits,Allocator>& rhs);
template<class charT, class traits, class Allocator>
  void swap(basic_string<charT,traits,Allocator>& lhs,
            basic_string<charT,traits,Allocator>&& rhs);
```

~~1 *Effects:* `lhs.swap(rhs);`~~

Remove from 23.2 Header <array> synopsis

```
template <class T, size_t N >
```

~~void swap(array<T,N>& x, array<T,N>& y);~~

Remove from 23.2 Header <deque> synopsis

```
~~template <class T, class Allocator>~~
~~  void swap(deque<T,Allocator>& x, deque<T,Allocator>& y);~~
~~template <class T, class Allocator>~~
~~  void swap(deque<T,Allocator>&& x, deque<T,Allocator>& y);~~
~~template <class T, class Allocator>~~
~~  void swap(deque<T,Allocator>& x, deque<T,Allocator>&& y);~~
```

Remove from 23.2 Header <list> synopsis

```
~~template <class T, class Allocator>~~
~~  void swap(list<T,Allocator>& x, list<T,Allocator>& y);~~
~~template <class T, class Allocator>~~
~~  void swap(list<T,Allocator>&& x, list<T,Allocator>& y);~~
~~template <class T, class Allocator>~~
~~  void swap(list<T,Allocator>& x, list<T,Allocator>&& y);~~
```

Remove from 23.2 Header <queue> synopsis

```
~~template <class T, class Allocator>~~
~~  void swap(queue<T,Allocator>& x, queue<T,Allocator>& y);~~
~~template <class T, class Allocator>~~
~~  void swap(queue<T,Allocator>&& x, queue<T,Allocator>& y);~~
~~template <class T, class Allocator>~~
~~  void swap(queue<T,Allocator>& x, queue<T,Allocator>&& y);~~

…

~~template <class T, class Allocator>~~
~~  void swap(priority_queue<T,Allocator>& x, priority_queue<T,Allocator>& y);~~
~~template <class T, class Allocator>~~
~~  void swap(priority_queue<T,Allocator>&& x, priority_queue<T,Allocator>& y);~~
~~template <class T, class Allocator>~~
~~  void swap(priority_queue<T,Allocator>& x, priority_queue<T,Allocator>&& y);~~
```

Remove from 23.2 Header <stack> synopsis

```
~~template <class T, class Allocator>~~
~~  void swap(stack<T,Allocator>& x, stack<T,Allocator>& y);~~
~~template <class T, class Allocator>~~
~~  void swap(stack<T,Allocator>&& x, stack<T,Allocator>& y);~~
~~template <class T, class Allocator>~~
~~  void swap(stack<T,Allocator>& x, stack<T,Allocator>&& y);~~
```

Remove from 23.2 Header <vector> synopsis

```
~~template <class T, class Allocator>~~
~~  void swap(vector<T,Allocator>& x, vector<T,Allocator>& y);~~
~~template <class T, class Allocator>~~
~~  void swap(vector<T,Allocator>&& x, vector<T,Allocator>& y);~~
~~template <class T, class Allocator>~~
~~  void swap(vector<T,Allocator>& x, vector<T,Allocator>&& y);~~
```

Update member-swap signature in **23.2.1p3 [array]**

```
void swap(array<T, N> &&);
```

Update **23.2.1.2 array specialized algorithms [array.special]**

```
template <class T, size_t N> void array::swap(array<T,N>&& x~~, array<T,N>& y~~);
```

1 *Effects:* `swap_ranges(~~x.~~begin(), ~~x.~~end(), ~~y~~x.begin() );`

Remove from **23.2.2 [deque]** synopsis

```
~~// specialized algorithms:~~
~~template <class T, class Allocator>~~
~~  void swap(deque<T,Allocator>& x, deque<T,Allocator>& y);~~
~~template <class T, class Allocator>~~
```

~~void swap(deque<T,Allocator>&& x, deque<T,Allocator>& y);~~
~~template <class T, class Allocator>~~
~~void swap(deque<T,Allocator>& x, deque<T,Allocator>&& y);~~

Append to **23.2.2.3 [deque.modifiers]**

`void swap( deque< T, Allocator > && x )`
Effects: Exchanges the contents of `*this` with that of `x`.
Complexity: Constant time.

Remove

~~**23.2.2.4 deque specialized algorithms [deque.special]**~~
~~template <class T, class Allocator>~~
~~void swap(deque<T,Allocator>& x, deque<T,Allocator>& y);~~
~~template <class T, class Allocator>~~
~~void swap(deque<T,Allocator>&& x, deque<T,Allocator>& y);~~
~~template <class T, class Allocator>~~
~~void swap(deque<T,Allocator>& x, deque<T,Allocator>&& y);~~
~~1 *Effects:* `x.swap(y);`~~

Remove from **23.2.3 [list]** synopsis

~~*// specialized algorithms:*~~
~~template <class T, class Allocator>~~
~~void swap(list<T,Allocator>& x, list<T,Allocator>& y);~~
~~template <class T, class Allocator>~~
~~void swap(list<T,Allocator>&& x, list<T,Allocator>& y);~~
~~template <class T, class Allocator>~~
~~void swap(list<T,Allocator>& x, list<T,Allocator>&& y);~~

Append to **23.2.3.3 [list.modifiers]**

`void swap( list< T, Allocator > && x )`
Effects: Exchanges the contents of `*this` with that of `x`.
Complexity: Constant time.

Remove

~~**23.2.3.5 list specialized algorithms [list.special]**~~
~~template <class T, class Allocator>~~
~~void swap(list<T,Allocator>& x, list<T,Allocator>& y);~~
~~template <class T, class Allocator>~~
~~void swap(list<T,Allocator>&& x, list<T,Allocator>& y);~~
~~template <class T, class Allocator>~~
~~void swap(list<T,Allocator>& x, list<T,Allocator>&& y);~~
~~1 *Effects:* `x.swap(y);`~~

Remove from **23.2.4.1.1 [queue.defn]** synopsis

~~*// specialized algorithms:*~~
~~template <class T, class Container>~~
~~void swap(queue<T,Container>& x, queue<T,Container>& y);~~
~~template <class T, class Container>~~
~~void swap(queue<T,Container>&& x, queue<T,Container>& y);~~
~~template <class T, class Container>~~
~~void swap(queue<T,Container>& x, queue<T,Container>&& y);~~

Remove

~~**23.2.4.1.3 queue specialized algorithms [queue.special]**~~
~~template <class T, class Container>~~
~~void swap(queue<T,Container>& x, queue<T,Container>& y);~~
~~template <class T, class Container>~~
~~void swap(queue<T,Container>&& x, queue<T,Container>& y);~~
~~template <class T, class Container>~~
~~void swap(queue<T,Container>& x, queue<T,Container>&& y);~~
~~1 *Effects:* `x.swap(y);`~~

Update member-swap specification

`void swap(priority_queue && q)~~;~~ { c.swap(q.c); }`

Remove from **23.2.4.2 [priority.queue]** synopsis

~~*// specialized algorithms:*~~
~~template <class T, class Container>~~
~~void swap(priority_queue<T,Container>& x, priority_queue<T,Container>& y);~~
~~template <class T, class Container>~~
~~void swap(priority_queue<T,Container>&& x, priority_queue<T,Container>& y);~~
~~template <class T, class Container>~~
~~void swap(priority_queue<T,Container>& x, priority_queue<T,Container>&& y);~~

Remove

~~**23.2.4.2.3 priority_queue specialized algorithms [priqueue.special]**~~
~~template <class T, class Container>~~
~~void swap(priority_queue<T,Container>& x, priority_queue<T,Container>& y);~~
~~template <class T, class Container>~~
~~void swap(priority_queue<T,Container>&& x, priority_queue<T,Container>& y);~~
~~template <class T, class Container>~~
~~void swap(priority_queue<T,Container>& x, priority_queue<T,Container>&& y);~~
~~1 *Effects:* x.swap(y);~~

Remove from **23.2.4.3.1 [stack.defn]** synopsis

~~*// specialized algorithms:*~~
~~template <class T, class Container>~~
~~void swap(stack<T,Container>& x, stack<T,Container>& y);~~
~~template <class T, class Container>~~
~~void swap(stack<T,Container>&& x, stack<T,Container>& y);~~
~~template <class T, class Container>~~
~~void swap(stack<T,Container>& x, stack<T,Container>&& y);~~

Remove

~~**23.2.4.3.3 stack specialized algorithms [stack.special]**~~
~~template <class T, class Container>~~
~~void swap(stack<T,Container>& x, stack<T,Container>& y);~~
~~template <class T, class Container>~~
~~void swap(stack<T,Container>&& x, stack<T,Container>& y);~~
~~template <class T, class Container>~~
~~void swap(stack<T,Container>& x, stack<T,Container>&& y);~~
~~1 *Effects:* x.swap(y);~~

Remove from **23.2.5p2 [vector]** synopsis

~~*// specialized algorithms:*~~
~~template <class T, class Allocator>~~
~~void swap(vector<T,Allocator>& x, vector<T,Allocator>& y);~~
~~template <class T, class Allocator>~~
~~void swap(vector<T,Allocator>&& x, vector<T,Allocator>& y);~~
~~template <class T, class Allocator>~~
~~void swap(vector<T,Allocator>& x, vector<T,Allocator>&& y);~~

Remove

~~**23.2.5.5 vector specialized algorithms [vector.special]**~~
~~template <class T, class Allocator>~~
~~void swap(vector<T,Allocator>& x, vector<T,Allocator>& y);~~
~~template <class T, class Allocator>~~
~~void swap(vector<T,Allocator>&& x, vector<T,Allocator>& y);~~
~~template <class T, class Allocator>~~
~~void swap(vector<T,Allocator>& x, vector<T,Allocator>&& y);~~
~~1 *Effects:* x.swap(y);~~

Remove from **23.3 [associative]** <map> synopsis

~~template <class Key, class T, class Compare, class Allocator>~~
~~void swap(map<Key,T,Compare,Allocator>& x,~~
~~map<Key,T,Compare,Allocator>& y);~~
~~template <class Key, class T, class Compare, class Allocator>~~
~~void swap(map<Key,T,Compare,Allocator&& x,~~
~~map<Key,T,Compare,Allocator>& y);~~
~~template <class Key, class T, class Compare, class Allocator>~~

~~void swap(map<Key,T,Compare,Allocator& x,~~
~~map<Key,T,Compare,Allocator>&& y);~~

…

~~template <class Key, class T, class Compare, class Allocator>~~
~~void swap(multimap<Key,T,Compare,Allocator>& x,~~
~~multimap<Key,T,Compare,Allocator>& y);~~
~~template <class Key, class T, class Compare, class Allocator>~~
~~void swap(multimap<Key,T,Compare,Allocator&& x,~~
~~multimap<Key,T,Compare,Allocator>& y);~~
~~template <class Key, class T, class Compare, class Allocator>~~
~~void swap(multimap<Key,T,Compare,Allocator& x,~~
~~multimap<Key,T,Compare,Allocator>&& y);~~

Remove from **23.3 [associative]** <set> synopsis

~~template <class Key, class Compare, class Allocator>~~
~~void swap(set<Key,Compare,Allocator>& x,~~
~~set<Key,Compare,Allocator>& y);~~
~~template <class Key, class T, class Compare, class Allocator>~~
~~void swap(set<Key,T,Compare,Allocator&& x,~~
~~set<Key,T,Compare,Allocator>& y);~~
~~template <class Key, class T, class Compare, class Allocator>~~
~~void swap(set<Key,T,Compare,Allocator& x,~~
~~set<Key,T,Compare,Allocator>&& y);~~

…

~~template <class Key, class Compare, class Allocator>~~
~~void swap(multiset<Key,Compare,Allocator>& x,~~
~~multiset<Key,Compare,Allocator>& y);~~
~~template <class Key, class T, class Compare, class Allocator>~~
~~void swap(multiset<Key,T,Compare,Allocator&& x,~~
~~multiset<Key,T,Compare,Allocator>& y);~~
~~template <class Key, class T, class Compare, class Allocator>~~
~~void swap(multiset<Key,T,Compare,Allocator& x,~~
~~multiset<Key,T,Compare,Allocator>&& y);~~

Remove from **23.3.1p2 [map]** synopsis

~~template <class Key, class T, class Compare, class Allocator>~~
~~void swap(map<Key,T,Compare,Allocator>& x,~~
~~map<Key,T,Compare,Allocator>& y);~~
~~template <class Key, class T, class Compare, class Allocator>~~
~~void swap(map<Key,T,Compare,Allocator&& x,~~
~~map<Key,T,Compare,Allocator>& y);~~
~~template <class Key, class T, class Compare, class Allocator>~~
~~void swap(map<Key,T,Compare,Allocator& x,~~
~~map<Key,T,Compare,Allocator>&& y);~~

Append to **23.3.1.3 [map.modifiers]**

`void swap(map<Key,T,Compare,Allocator>&& x)`
Effects: Exchanges the contents of `*this` with that of `x`.
Complexity: Constant time.

Remove

~~23.3.1.5 `map` specialized algorithms [map.special]~~
~~template <class Key, class T, class Compare, class Allocator>~~
~~void swap(map<Key,T,Compare,Allocator>& x,~~
~~map<Key,T,Compare,Allocator>& y);~~
~~template <class Key, class T, class Compare, class Allocator>~~
~~void swap(map<Key,T,Compare,Allocator&& x,~~
~~map<Key,T,Compare,Allocator>& y);~~
~~template <class Key, class T, class Compare, class Allocator>~~
~~void swap(map<Key,T,Compare,Allocator& x,~~

~~map<Key,T,Compare,Allocator>&& y);~~
~~1 *Effects:* x.swap(y);~~

Remove from **23.3.2p2 [multimap]** synopsis

~~template <class Key, class T, class Compare, class Allocator>~~
~~void swap(multimap<Key,T,Compare,Allocator>& x,~~
~~multimap<Key,T,Compare,Allocator>& y);~~
~~template <class Key, class T, class Compare, class Allocator>~~
~~void swap(multimap<Key,T,Compare,Allocator&& x,~~
~~multimap<Key,T,Compare,Allocator>& y);~~
~~template <class Key, class T, class Compare, class Allocator>~~
~~void swap(multimap<Key,T,Compare,Allocator& x,~~
~~multimap<Key,T,Compare,Allocator>&& y);~~

Append to **23.3.2.2 [multimap.modifiers]**

`void swap(multimap<Key,T,Compare,Allocator>&& x)`
Effects: Exchanges the contents of `*this` with that of `x`.
Complexity: Constant time.

Remove

~~23.3.2.4 `multimap` specialized algorithms [multimap.special]~~
~~template <class Key, class T, class Compare, class Allocator>~~
~~void swap(multimap<Key,T,Compare,Allocator>& x,~~
~~multimap<Key,T,Compare,Allocator>& y);~~
~~template <class Key, class T, class Compare, class Allocator>~~
~~void swap(multimap<Key,T,Compare,Allocator&& x,~~
~~multimap<Key,T,Compare,Allocator>& y);~~
~~template <class Key, class T, class Compare, class Allocator>~~
~~void swap(multimap<Key,T,Compare,Allocator& x,~~
~~multimap<Key,T,Compare,Allocator>&& y);~~
~~1 *Effects:* x.swap(y);~~

Update member-swap declaration in **23.3.3p2 [set]** synopsis

`void swap(set<Key,Compare,Allocator>&&);`

Remove from **23.3.3p2 [set]** synopsis

~~template <class Key, class Compare, class Allocator>~~
~~void swap(set<Key,Compare,Allocator>& x,~~
~~set<Key,Compare,Allocator>& y);~~
~~template <class Key, class Compare, class Allocator>~~
~~void swap(set<Key,Compare,Allocator&& x,~~
~~set<Key,Compare,Allocator>& y);~~
~~template <class Key, class Compare, class Allocator>~~
~~void swap(set<Key,Compare,Allocator& x,~~
~~set<Key,Compare,Allocator>&& y);~~

Replace

~~23.3.3.2 `set` specialized algorithms [set.special]~~
~~template <class Key, class T, class Compare, class Allocator>~~
~~void swap(set<Key,T,Compare,Allocator>& x,~~
~~set<Key,T,Compare,Allocator>& y);~~
~~template <class Key, class T, class Compare, class Allocator>~~
~~void swap(set<Key,T,Compare,Allocator&& x,~~
~~set<Key,T,Compare,Allocator>& y);~~
~~template <class Key, class T, class Compare, class Allocator>~~
~~void swap(set<Key,T,Compare,Allocator& x,~~
~~set<Key,T,Compare,Allocator>&& y);~~
~~1 *Effects:* x.swap(y);~~

with

23.3.3.2 `set` modifiers [set.modifiers]

`void swap(set<Key,Compare,Allocator>&& x)`
Effects: Exchanges the contents of `*this` with that of `x`.
Complexity: Constant time.

Remove from **23.3.4p2 [multiset]** synopsis

```
template <class Key, class Compare, class Allocator>
  void swap(multiset<Key,Compare,Allocator>& x,
            multiset<Key,Compare,Allocator>& y);
template <class Key, class Compare, class Allocator>
  void swap(multiset<Key,Compare,Allocator&& x,
            multiset<Key,Compare,Allocator>& y);
template <class Key, class Compare, class Allocator>
  void swap(multiset<Key,Compare,Allocator& x,
            multiset<Key,Compare,Allocator>&& y);
```

Replace

~~23.3.4.2 `multiset` specialized algorithms [multiset.special]~~

```
template <class Key, class T, class Compare, class Allocator>
  void swap(multiset<Key,T,Compare,Allocator>& x,
            multiset<Key,T,Compare,Allocator>& y);
template <class Key, class T, class Compare, class Allocator>
  void swap(multiset<Key,T,Compare,Allocator&& x,
            multiset<Key,T,Compare,Allocator>& y);
template <class Key, class T, class Compare, class Allocator>
  void swap(multiset<Key,T,Compare,Allocator& x,
            multiset<Key,T,Compare,Allocator>&& y);
```

~~1 *Effects:* `x.swap(y);`~~

with

23.3.4.2 `multiset` modifiers [multiset.modifiers]

```
void swap(multiset<Key,Compare,Allocator>&& x)
```

Effects: Exchanges the contents of `*this` with that of `x`.

Complexity: Constant time.

Remove from **23.4 [unord]** `<unordered_map>` synopsis

```
template <class Key, class T, class Hash, class Pred, class Alloc>
  void swap(unordered_map<Key, T, Hash, Pred, Alloc>& x,
            unordered_map<Key, T, Hash, Pred, Alloc>& y);

template <class Key, class T, class Hash, class Pred, class Alloc>
  void swap(unordered_multimap<Key, T, Hash, Pred, Alloc>& x,
            unordered_multimap<Key, T, Hash, Pred, Alloc>& y);
```

Remove from **23.4 [unord]** `<unordered_set>` synopsis

```
template <class Value, class Hash, class Pred, class Alloc>
  void swap(unordered_set<Value, Hash, Pred, Alloc>& x,
            unordered_set<Value, Hash, Pred, Alloc>& y);

template <class Value, class Hash, class Pred, class Alloc>
  void swap(unordered_multiset<Value, Hash, Pred, Alloc>& x,
            unordered_multiset<Value, Hash, Pred, Alloc>& y);
```

Update member-swap declaration in **23.4.1p3 [unord.map]** synopsis

```
void swap(unordered_map&&);
```

Remove from **23.4.1p3 [unord.map]** synopsis

```
template <class Key, class T, class Hash, class Pred, class Alloc>
  void swap(unordered_map<Key, T, Hash, Pred, Alloc>& x,
            unordered_map<Key, T, Hash, Pred, Alloc>& y);
```

Replace

~~**23.4.1.3 `unordered_map` swap [unord.map.swap]**~~

```
template <class Key, class T, class Hash, class Pred, class Alloc>
  void swap(unordered_map<Key, T, Hash, Pred, Alloc>& x,
            unordered_map<Key, T, Hash, Pred, Alloc>& y);
```

~~1 *Effects:* `x.swap(y).`~~

with

23.4.1.3 `unordered_map` swap [unord.map.swap]

```
void swap(unordered_map&& x)
```

Effects: Exchanges the contents of `*this` with that of `x`.
Complexity: Constant time.

Update member-swap declaration in **23.4.2p3 [unord.multimap]** synopsis

```
void swap(unordered_multimap&&);
```

Remove from **23.4.2p3 [unord.multimap]** synopsis

~~template <class Key, class T, class Hash, class Pred, class Alloc>~~
~~void swap(unordered_multimap<Key, T, Hash, Pred, Alloc>& x,~~
~~unordered_multimap<Key, T, Hash, Pred, Alloc>& y);~~

Replace

~~**23.4.2.2 unordered_multimap swap [unord.multimap.swap]**~~

~~template <class Key, class T, class Hash, class Pred, class Alloc>~~
~~void swap(unordered_multimap<Key, T, Hash, Pred, Alloc>& x,~~
~~unordered_multimap<Key, T, Hash, Pred, Alloc>& y);~~

~~1 *Effects:* x.swap(y).~~

with

23.4.2.2 `unordered_multimap` swap [unord.multimap.swap]

`void swap(unordered_multimap&& x)`
Effects: Exchanges the contents of `*this` with that of `x`.
Complexity: Constant time.

Update member-swap declaration in **23.4.3p3 [unord.set]** synopsis

```
void swap(unordered_set&&);
```

Remove from **23.4.3p3 [unord.set]** synopsis

~~template <class Key, class T, class Hash, class Pred, class Alloc>~~
~~void swap(unordered_set<Key, T, Hash, Pred, Alloc>& x,~~
~~unordered_set<Key, T, Hash, Pred, Alloc>& y);~~

Replace

~~**23.4.3.2 unordered_set swap [unord.set.swap]**~~

~~template <class Key, class T, class Hash, class Pred, class Alloc>~~
~~void swap(unordered_set<Key, T, Hash, Pred, Alloc>& x,~~
~~unordered_set<Key, T, Hash, Pred, Alloc>& y);~~

~~1 *Effects:* x.swap(y).~~

with

23.4.3.2 `unordered_set` swap [unord.set.swap]

`void swap(unordered_set&& x)`
Effects: Exchanges the contents of `*this` with that of `x`.
Complexity: Constant time.

Update member-swap declaration in **23.4.4p3 [unord.multiset]** synopsis

```
void swap(unordered_multiset&&);
```

Remove from **23.4.4p3 [unord.multiset]** synopsis

~~template <class Key, class T, class Hash, class Pred, class Alloc>~~
~~void swap(unordered_multiset<Key, T, Hash, Pred, Alloc>& x,~~
~~unordered_multiset<Key, T, Hash, Pred, Alloc>& y);~~

Replace

~~**23.4.4.2 unordered_multiset swap [unord.multiset.swap]**~~

~~template <class Key, class T, class Hash, class Pred, class Alloc>~~
~~void swap(unordered_multiset<Key, T, Hash, Pred, Alloc>& x,~~
~~unordered_multiset<Key, T, Hash, Pred, Alloc>& y);~~

~~1 *Effects:* x.swap(y).~~

with

23.4.4.2 `unordered_multiset` swap [unord.multiset.swap]

`void swap(unordered_multiset&& x)`
Effects: Exchanges the contents of `*this` with that of `x`.
Complexity: Constant time.

Remove from **26.5.1 Header <valarray> synopsis [valarray.synopsis]**

~~template<class T> void swap(valarray<T>&, valarray<T>&);~~
~~template<class T> void swap(valarray<T>&&, valarray<T>&);~~
~~template<class T> void swap(valarray<T>&, valarray<T>&&);~~

Remove

~~**26.5.3.4 valarray specialized algorithms [valarray.special]**~~
~~template <class T> void swap(valarray<T>& x, valarray<T>& y);~~
~~template <class T> void swap(valarray<T>&& x, valarray<T>& y);~~
~~template <class T> void swap(valarray<T>& x, valarray<T>&& y);~~
~~1 *Effects:* x.swap(y).~~

From **27.6.1.1 Class template basic_istream [istream]** remove

~~*// swap:*~~
~~template <class charT, class traits>~~
~~void swap(basic_istream<charT, traits>& x, basic_istream<charT, traits>& y);~~
~~template <class charT, class traits>~~
~~void swap(basic_istream<charT, traits>&& x, basic_istream<charT, traits>& y);~~
~~template <class charT, class traits>~~
~~void swap(basic_istream<charT, traits>& x, basic_istream<charT, traits>&& y);~~

From **27.6.1.1.2 Class basic_istream assign and swap [istream.assign]** remove

~~template <class charT, class traits>~~
~~void swap(basic_istream<charT, traits>& x, basic_istream<charT, traits>& y);~~
~~template <class charT, class traits>~~
~~void swap(basic_istream<charT, traits>&& x, basic_istream<charT, traits>& y);~~
~~template <class charT, class traits>~~
~~void swap(basic_istream<charT, traits>& x, basic_istream<charT, traits>&& y);~~
~~4 *Effects:* x.swap(y).~~

From **27.6.1.5 Class template basic_iostream [iostreamclass]** remove

~~template <class charT, class traits>~~
~~void swap(basic_iostream<charT, traits>& x, basic_iostream<charT, traits>& y);~~
~~template <class charT, class traits>~~
~~void swap(basic_iostream<charT, traits>&& x, basic_iostream<charT, traits>& y);~~
~~template <class charT, class traits>~~
~~void swap(basic_iostream<charT, traits>& x, basic_iostream<charT, traits>&& y);~~

From **27.6.1.5.3 basic_iostream assign and swap [iostream.assign]** remove

~~template <class charT, class traits>~~
~~void swap(basic_iostream<charT, traits>& x, basic_iostream<charT, traits>& y);~~
~~template <class charT, class traits>~~
~~void swap(basic_iostream<charT, traits>&& x, basic_iostream<charT, traits>& y);~~
~~template <class charT, class traits>~~
~~void swap(basic_iostream<charT, traits>& x, basic_iostream<charT, traits>&& y);~~
~~3 *Effects:* x.swap(y).~~

From **27.6.2.1 Class template basic_ostream [ostream]** remove

~~*// swap:*~~
~~template <class charT, class traits>~~
~~void swap(basic_ostream<charT, traits>& x, basic_ostream<charT, traits>& y);~~
~~template <class charT, class traits>~~
~~void swap(basic_ostream<charT, traits>&& x, basic_ostream<charT, traits>& y);~~
~~template <class charT, class traits>~~
~~void swap(basic_ostream<charT, traits>& x, basic_ostream<charT, traits>&& y);~~

From **27.6.2.3 Class basic_ostream assign and swap [ostream.assign]** remove

~~template <class charT, class traits>~~
~~void swap(basic_ostream<charT, traits>& x, basic_ostream<charT, traits>& y);~~
~~template <class charT, class traits>~~
~~void swap(basic_ostream<charT, traits>&& x, basic_ostream<charT, traits>& y);~~
~~template <class charT, class traits>~~
~~void swap(basic_ostream<charT, traits>& x, basic_ostream<charT, traits>&& y);~~
~~4 *Effects:* x.swap(y).~~

From **27.7.1 Class template basic_stringbuf [stringbuf]** remove

~~template <class charT, class traits, class Allocator>~~
~~void swap(basic_stringbuf<charT, traits, Allocator>& x,~~
~~basic_stringbuf<charT, traits, Allocator>& y);~~
~~template <class charT, class traits, class Allocator>~~
~~void swap(basic_stringbuf<charT, traits, Allocator>&& x,~~
~~basic_stringbuf<charT, traits, Allocator>& y);~~
~~template <class charT, class traits, class Allocator>~~
~~void swap(basic_stringbuf<charT, traits, Allocator>& x,~~
~~basic_stringbuf<charT, traits, Allocator>&& y);~~

From **27.7.1.2 Assign and swap [stringbuf.assign]** remove

~~template <class charT, class traits, class Allocator>~~
~~void swap(basic_stringbuf<charT, traits, Allocator>& x,~~
~~basic_stringbuf<charT, traits, Allocator>& y);~~
~~template <class charT, class traits, class Allocator>~~
~~void swap(basic_stringbuf<charT, traits, Allocator>&& x,~~
~~basic_stringbuf<charT, traits, Allocator>& y);~~
~~template <class charT, class traits, class Allocator>~~
~~void swap(basic_stringbuf<charT, traits, Allocator>& x,~~
~~basic_stringbuf<charT, traits, Allocator>&& y);~~
~~4 *Effects:* x.swap(y).~~

From **27.7.2 Class template basic_istringstream [istringstream]** remove

~~template <class charT, class traits, class Allocator>~~
~~void swap(basic_istringstream<charT, traits, Allocator>& x,~~
~~basic_istringstream<charT, traits, Allocator>& y);~~
~~template <class charT, class traits, class Allocator>~~
~~void swap(basic_istringstream<charT, traits, Allocator>&& x,~~
~~basic_istringstream<charT, traits, Allocator>& y);~~
~~template <class charT, class traits, class Allocator>~~
~~void swap(basic_istringstream<charT, traits, Allocator>& x,~~
~~basic_istringstream<charT, traits, Allocator>&& y);~~

From **27.7.2.2 Assign and swap [istringstream.assign]** remove

~~template <class charT, class traits, class Allocator>~~
~~void swap(basic_istringstream<charT, traits, Allocator>& x,~~
~~basic_istringstream<charT, traits, Allocator>& y);~~
~~template <class charT, class traits, class Allocator>~~
~~void swap(basic_istringstream<charT, traits, Allocator>&& x,~~
~~basic_istringstream<charT, traits, Allocator>& y);~~
~~template <class charT, class traits, class Allocator>~~
~~void swap(basic_istringstream<charT, traits, Allocator>& x,~~
~~basic_istringstream<charT, traits, Allocator>&& y);~~
~~4 *Effects:* x.swap(y).~~

From **27.7.3 Class basic_ostringstream [ostringstream]** remove

~~template <class charT, class traits, class Allocator>~~
~~void swap(basic_ostringstream<charT, traits, Allocator>& x,~~
~~basic_ostringstream<charT, traits, Allocator>& y);~~
~~template <class charT, class traits, class Allocator>~~
~~void swap(basic_ostringstream<charT, traits, Allocator>&& x,~~
~~basic_ostringstream<charT, traits, Allocator>& y);~~
~~template <class charT, class traits, class Allocator>~~
~~void swap(basic_ostringstream<charT, traits, Allocator>& x,~~
~~basic_ostringstream<charT, traits, Allocator>&& y);~~

From **27.7.3.2 Assign and swap [ostringstream.assign]** remove

~~template <class charT, class traits, class Allocator>~~
~~void swap(basic_ostringstream<charT, traits, Allocator>& x,~~
~~basic_ostringstream<charT, traits, Allocator>& y);~~
~~template <class charT, class traits, class Allocator>~~
~~void swap(basic_ostringstream<charT, traits, Allocator>&& x,~~
~~basic_ostringstream<charT, traits, Allocator>& y);~~
~~template <class charT, class traits, class Allocator>~~

~~void swap(basic_ostringstream<charT, traits, Allocator>& x,~~
~~basic_ostringstream<charT, traits, Allocator>&& y);~~
~~4 *Effects:* x.swap(y).~~

From **27.7.4 Class basic_stringstream [stringstream]** remove

~~template <class charT, class traits, class Allocator>~~
~~void swap(basic_stringstream<charT, traits, Allocator>& x,~~
~~basic_stringstream<charT, traits, Allocator>& y);~~
~~template <class charT, class traits, class Allocator>~~
~~void swap(basic_stringstream<charT, traits, Allocator>&& x,~~
~~basic_stringstream<charT, traits, Allocator>& y);~~
~~template <class charT, class traits, class Allocator>~~
~~void swap(basic_stringstream<charT, traits, Allocator>& x,~~
~~basic_stringstream<charT, traits, Allocator>&& y);~~

From **27.7.5.1 Assign and swap [stringstream.assign]** remove

~~template <class charT, class traits, class Allocator>~~
~~void swap(basic_stringstream<charT, traits, Allocator>& x,~~
~~basic_stringstream<charT, traits, Allocator>& y);~~
~~template <class charT, class traits, class Allocator>~~
~~void swap(basic_stringstream<charT, traits, Allocator>&& x,~~
~~basic_stringstream<charT, traits, Allocator>& y);~~
~~template <class charT, class traits, class Allocator>~~
~~void swap(basic_stringstream<charT, traits, Allocator>& x,~~
~~basic_stringstream<charT, traits, Allocator>&& y);~~
~~4 *Effects:* x.swap(y).~~

From **27.8.1.1 Class template basic_filebuf [filebuf]** remove

~~template <class charT, class traits>~~
~~void swap(basic_filebuf<charT, traits>& x,~~
~~basic_filebuf<charT, traits>& y);~~
~~template <class charT, class traits>~~
~~void swap(basic_filebuf<charT, traits>&& x,~~
~~basic_filebuf<charT, traits>& y);~~
~~template <class charT, class traits>~~
~~void swap(basic_filebuf<charT, traits>& x,~~
~~basic_filebuf<charT, traits>&& y);~~

From **27.8.1.3 Assign and swap [filebuf.assign]** remove

~~template <class charT, class traits>~~
~~void swap(basic_filebuf<charT, traits>& x,~~
~~basic_filebuf<charT, traits>& y);~~
~~template <class charT, class traits>~~
~~void swap(basic_filebuf<charT, traits>&& x,~~
~~basic_filebuf<charT, traits>& y);~~
~~template <class charT, class traits>~~
~~void swap(basic_filebuf<charT, traits>& x,~~
~~basic_filebuf<charT, traits>&& y);~~
~~4 *Effects:* x.swap(y).~~

From **27.8.1.6 Class template basic_ifstream [ifstream]** remove

~~template <class charT, class traits, class Allocator>~~
~~void swap(basic_ifstream<charT, traits, Allocator>& x,~~
~~basic_ifstream<charT, traits, Allocator>& y);~~
~~template <class charT, class traits, class Allocator>}~~
~~void swap(basic_ifstream<charT, traits, Allocator>&& x,~~
~~basic_ifstream<charT, traits, Allocator>& y);~~
~~template <class charT, class traits, class Allocator>~~
~~void swap(basic_ifstream<charT, traits, Allocator>& x,~~
~~basic_ifstream<charT, traits, Allocator>&& y);~~

From **27.8.1.8 Assign and swap [ifstream.assign]** remove

~~template <class charT, class traits, class Allocator>~~
~~void swap(basic_ifstream<charT, traits, Allocator>& x,~~
~~basic_ifstream<charT, traits, Allocator>& y);~~

~~template <class charT, class traits, class Allocator>~~
~~void swap(basic_ifstream<charT, traits, Allocator>&& x,~~
~~basic_ifstream<charT, traits, Allocator>& y);~~
~~template <class charT, class traits, class Allocator>~~
~~void swap(basic_ifstream<charT, traits, Allocator>& x,~~
~~basic_ifstream<charT, traits, Allocator>&& y);~~
~~4 *Effects:* x.swap(y).~~

From **27.8.1.10 Class template basic_ofstream [ofstream]** remove

~~template <class charT, class traits, class Allocator>~~
~~void swap(basic_ofstream<charT, traits, Allocator>& x,~~
~~basic_ofstream<charT, traits, Allocator>& y);~~
~~template <class charT, class traits, class Allocator>~~
~~void swap(basic_ofstream<charT, traits, Allocator>&& x,~~
~~basic_ofstream<charT, traits, Allocator>& y);~~
~~template <class charT, class traits, class Allocator>~~
~~void swap(basic_ofstream<charT, traits, Allocator>& x,~~
~~basic_ofstream<charT, traits, Allocator>&& y);~~

From **27.8.1.12 Assign and swap [ofstream.assign]** remove

~~template <class charT, class traits, class Allocator>~~
~~void swap(basic_ofstream<charT, traits, Allocator>& x,~~
~~basic_ofstream<charT, traits, Allocator>& y);~~
~~template <class charT, class traits, class Allocator>~~
~~void swap(basic_ofstream<charT, traits, Allocator>&& x,~~
~~basic_ofstream<charT, traits, Allocator>& y);~~
~~template <class charT, class traits, class Allocator>~~
~~void swap(basic_ofstream<charT, traits, Allocator>& x,~~
~~basic_ofstream<charT, traits, Allocator>&& y);~~
~~4 *Effects:* x.swap(y).~~

From **27.8.1.14 Class template basic_fstream [fstream]** remove

~~template <class charT, class traits, class Allocator>~~
~~void swap(basic_fstream<charT, traits, Allocator>& x,~~
~~basic_fstream<charT, traits, Allocator>& y);~~
~~template <class charT, class traits, class Allocator>~~
~~void swap(basic_fstream<charT, traits, Allocator>&& x,~~
~~basic_fstream<charT, traits, Allocator>& y);~~
~~template <class charT, class traits, class Allocator>~~
~~void swap(basic_fstream<charT, traits, Allocator>& x,~~
~~basic_fstream<charT, traits, Allocator>&& y);~~

From **27.8.1.16 Assign and swap [fstream.assign]** remove

~~template <class charT, class traits, class Allocator>~~
~~void swap(basic_fstream<charT, traits, Allocator>& x,~~
~~basic_fstream<charT, traits, Allocator>& y);~~
~~template <class charT, class traits, class Allocator>~~
~~void swap(basic_fstream<charT, traits, Allocator>&& x,~~
~~basic_fstream<charT, traits, Allocator>& y);~~
~~template <class charT, class traits, class Allocator>~~
~~void swap(basic_fstream<charT, traits, Allocator>& x,~~
~~basic_fstream<charT, traits, Allocator>&& y);~~
~~4 *Effects:* x.swap(y).~~

From **28.4 Header <regex> synopsis [re.syn]** remove

~~*// 28.8.6, basic_regex swap:*~~
~~template <class charT, class traits>~~
~~void swap(basic_regex<charT, traits>& e1, basic_regex<charT, traits>& e2);~~

…

~~*// 28.10.6, match_results swap:*~~
~~template <class BidirectionalIterator, class Allocator>~~
~~void swap(match_results<BidirectionalIterator, Allocator>& m1,~~
~~match_results<BidirectionalIterator, Allocator>& m2);~~

Update the member-swap declaration in **28.8 Class template basic_regex [re.regex]**

```
// 28.8.6, swap:
void swap(basic_regex&&);
```

Update **28.8.6 basic_regex swap [re.regex.swap]**

```
void swap(basic_regex&& e);
```

1 *Effects:* Swaps the contents of the two regular expressions.

Remove

~~**28.8.7 basic_regex non-member functions [re.regex.nonmemb]**~~

~~**28.8.7.1 basic_regex non-member swap [re.regex.nmswap]**~~

```
template <class charT, class traits>
void swap(basic_regex<charT, traits>& lhs, basic_regex<charT, traits>& rhs);
```

1 *Effects:* Calls `lhs.swap(rhs)`.

Update the member-swap declaration in **28.10 Class template match_results [re.results]**

```
// 28.10.6, swap:
void swap(match_results&& that);
```

Update **28.10.6 `match_results` swap [re.results.swap]**

```
void swap(match_results&& that);
```

1 *Effects:* Swaps the contents of the two sequences.

2 *Postcondition:* `*this` contains the sequence of matched sub-expressions that were in `that`, `that` contains the sequence of matched sub-expressions that were in `*this`.

3 *Complexity:* constant time.

```
template <class BidirectionalIterator, class Allocator>
void swap(match_results<BidirectionalIterator, Allocator>& m1,
          match_results<BidirectionalIterator, Allocator>& m2);
```

4 *Effects:* `m1.swap(m2)`.

From **30.2 Threads [thread.threads]** remove

```
void swap(thread& x, thread& y);
void swap(thread&& x, thread& y);
void swap(thread& x, thread&& y);
```

Remove

~~**30.2.1.7 thread specialized algorithms [thread.thread.algorithm]**~~

```
void swap(thread& x, thread& y);
void swap(thread&& x, thread& y);
void swap(thread& x, thread&& y);
```

1 *Effects:* `x.swap(y)`

From **30.3 Mutual exclusion [thread.mutex]** remove

```
template <class Mutex>
  void swap(unique_lock<Mutex>& x, unique_lock<Mutex>& y);
template <class Mutex>
  void swap(unique_lock<Mutex>&& x, unique_lock<Mutex>& y);
template <class Mutex>
  void swap(unique_lock<Mutex>& x, unique_lock<Mutex>&& y);
```

From **30.3.3.2 Class template unique_lock [thread.lock.unique]** remove

```
template <class Mutex>
  void swap(unique_lock<Mutex>& x, unique_lock<Mutex>& y);
template <class Mutex>
  void swap(unique_lock<Mutex>&& x, unique_lock<Mutex>& y);
template <class Mutex>
  void swap(unique_lock<Mutex>& x, unique_lock<Mutex>&& y);
```

From **30.3.3.2.3 unique_lock modifiers [thread.lock.unique.mod]** remove

```
template <class Mutex>
```

~~void swap(unique_lock<Mutex>& x, unique_lock<Mutex>& y);~~
~~template <class Mutex>~~
~~void swap(unique_lock<Mutex>&& x, unique_lock<Mutex>& y);~~
~~template <class Mutex>~~
~~void swap(unique_lock<Mutex>& x, unique_lock<Mutex>&& y);~~
~~6 *Effects:* x.swap(y)~~
~~7 *Throws:* Nothing.~~