Document number:	P0198R0
Date:	2016-02-11
Project:	ISO/IEC JTC1 SC22 WG21 Programming Language C++
Audience:	Evolution Working Group / Library Evolution Working Group
Reply-to:	Vicente J. Botet Escriba <vicente.botet@wanadoo.fr>

Default Swap

Abstract

Defining `swap` as copy/move constructor/assignment operators, for simple classes is tedious, repetitive, slightly error-prone, and easily automated.

I propose to (implicitly) supply default version of this operation by the compiler, if needed. The meaning `swap` is to swap each member.

This paper contains only a working paper wording. This is a discussion paper to determine EWG/LEWG interest in the feature, and if there is interest to get direction for a follow-up paper with more detailed wording.

Table of Contents

1. Introduction
2. Motivation
3. Proposal
4. Design rationale
5. Working paper wording
6. Alternative solutions
7. Implementability
8. Open points
9. Acknowledgements
10. References

Introduction

Defining `swap` as copy/move constructor/assignment operators for simple classes is tedious, repetitive,

slightly error-prone, and easily automated.

I propose to (implicitly) supply default version of this operation by the compiler, if needed. The meaning `swap` is to swap each member.

If the simple defaults are unsuitable for a class, a programmer can, as ever, define more suitable ones or suppress the defaults. The proposal is to add the operations as an integral part of C++ (like =), rather than as a library feature.

The proposal follows the same approach as Default comparison as in N4475, that is, that having default generated code for these basic operations only when needed and possible would make the language simpler.

This paper contains only a working paper wording. This is a discussion paper to determine EWG interest in the feature, and if there is interest to get direction for a follow-up paper with more detailed wording.

Motivation

Some standard algorithms require that an argument type supply an overload for `swap`.

The standard `std::swap` algorithm works well for `Movable` types. However a compiler generated member-wise `swap` function could apply to more types, e.g. `std::array` is *Swappable* and not *Movable*, and be even more efficient. Writing such types can be tedious (and all tedious tasks are error prone).

For example

```
class Foo {
  int i;
  string str;
  bool b;
  //...

  friend void swap(const Foo& lhs, const Foo& rhs) {
    using std::swap;
    swap(lhs.i,rhs.i);
    swap(lhs.str, rhs.str);
    swap(lhs.b, rhs.b);
  }
};
```

One of the benefit for a default member-wise `swap` respect to `std::swap` move-based, beside the performances, is that the former can sometimes offer a better `noexcept` guarantee. E.g., for `std::vector`, `swap` can be always guaranteed to be `noexcept`, whereas its `move` operations

throw.

Even, when one of the member-wise `swap` can throw, it is worth generating the member-wise `swap` because otherwise the `std::swap` move-based will also throw, if a class has a `swap` throw it will also have a `move` that throws.

Proposal

I propose to generate a default version for `swap` for simple classes when needed. If this default is unsuitable for a type, `=delete` it. If non-default of this operation is needed, define them (as always). If `swap` is already declared, a default is not generated for it. This is exactly the way assignment and constructors work today and as comparison operators would work if N4475 is adopted.

What is the definition of `swap` ?

We propose a member-wise `swap` definition.

`noexcept` expression

`swap` will be generated regardless of `noexcept` of member `swap` function, and will have automatic `noexcept` specification that would depend on `is_nothrow_swapable` on each base class and non-static data member.

When it could be applied

The following restricts the generation of `swap`

- has that operation defined or deleted in the class namespace in any translation unit [Note: this requires link-time checking], or
- the move constructor and assignment operations are not generated by default or
- has a non-static data member for which the `swap` operation doesn't exist and cannot be generated

The generated implementation is not considered a function so it cannot have its address taken [Note: like the `=` operator. -- end].

[For example:

```
class S {
  int i;
  string str;
  bool b;
  //...
};

void f(S& x, S& y)
{
    swap(x,y);
}
```

Is equivalent to

```
class S {
  int i;
  string str;
  bool b;
  //...
};

void f(S& x, S& y)
{
    {
        using std::swap;
        swap(x.i,y.i);
        swap(x.str,y.str);
        swap(x.b,y.b);
    }
}
```

-- end]

Design Rationale

Rule of 6

I may be beneficial to have proper rule of six: i.e. non of following copy constructor/assignment, move constructor/assignment, `swap`, destructor is generated if at least on of them is defined. However this would be breaking change (as we may already have classes with swap, that uses compiler generated special functions), so we stick to making `swap` depended on other five.

`swap(T&, T&) = default`

As copy/move constructor/assignment and destructor can be declared `=default` by the author of the class, declaring `swap` as `=default`, should also be possible. The major case we have in mind is when the user add some traces in the destructor.

Working paper wording

This wording is very “drafty” and has not gone through expert review. It is intended to reflect the design decisions described above. It is inspired from the wording for Default comparison in N4532.

Add a "Swap expression" section in 5

Swap expression [expr.swap]

A *swap expression* is a particular case of a *function call expression* when the function name is `swap`.

If an operand is of class type and no suitable function is found in the class namespace, the implicitly-declared `swap` non-member operation as described in over.generate_swap is used.

Add a "Special non-member swap operation" section after 13.6

Special non-member `swap` operation [over.generate_swap]

Implicitly-declared `swap` non-member operation

If no user-defined `swap` operation is provided for a class type `T` (`struct`, `class` or `union`), and all of the following is true:

- there are no user-declared copy constructors;
- there are no user-declared copy assignment operators;
- there are no user-declared move constructors;
- there are no user-declared move assignment operators;
- there are no user-declared destructors;

then the compiler will declare a `friend` `swap` operation with the signature

```
friend void swap(T&, T&) noexcept(see below);
```

The generated implementation is not considered a function so it cannot have its address taken [Note: like the = operator.].

Explicitly defaulted `swap` non-member operation

The user may still force the generation of the implicitly declared `swap` operation declaring it as a friend with the keyword `default`.

The generated implementation is not considered a function so it cannot have its address taken [Note: like the = operator.].

Deleted implicitly-declared `swap` non-member operation

The implicitly-declared or defaulted `swap` operation for class `T` is defined as deleted in any of the following is true:

- `T` has non-static data members that cannot be swapped;
- `T` has direct or virtual base class that cannot be swapped;
- `T` implicit-generated move constructor is deleted;
- `T` implicit-generated move assignment is deleted;
- `T` implicit-generated destructor is deleted;

The deleted implicitly-declared `swap` operation is ignored by overload resolution.

Implicitly-defined `swap` non-member operation

If the implicitly-declared `swap` operation is not deleted, it is defined (that is, a function body is generated and compiled) by the compiler if odr-used.

- For `union` types, the implicitly-defined `swap` operation do as the `std::swap`.
- For non-union class types (`class` and `struct`), the `swap` non-member operation performs full member-wise swap of the object's bases and non-static members, in their initialization order.

Alternative solution

Reflection, generic `swap` and friend

Based on a future reflection library e.g. N4428 or N4451, we could define a generic `swap` function instead of generating it. However, to the author knowledge, this would need to declare a friend function, which is much more intrusive than the compiler generated solution.

Next follows how the generic overload could be defined making use of some reflection

```
namespace std {
namespace experimental {namespace reflect { inline namespace v1 {

    template <class C>
    struct is_swap_generation_enabled;

    template <class C>
    struct is_nothrow_swapable_generated_swap;

}}}

// However this swap overload would need a friend declaration
template <class C>
enable_if<reflect::is_swap_generation_enabled<C>{}> swap(C & x, C & y)
    noexcept(is_nothrow_swapable_generated_swap<C>{})
{
    using std::swap
    // swap the base classes (needs friend access and reflection)
    ...
    // swap the non-static data-member (needs friend access and reflection)
    ...
}

} // namespace std
```

Note that the `swap` overload would need to be declared as friend on the class.

```
namespace MyNS {

class MyC {

    template <class C>
        friend
        enable_if<std::experimental::reflect::is_swap_generation_enabled_t<C>{}>
        std::swap(C & x, C & y)
            noexcept(std::experimental::reflect::is_nothrow_swapable_generated_swap_
    // ...
};

} //namespace
```

This would be almost a showstopper and one of the reasons, that even with reflection, a compiler generated version is a better and less intrusive choice.

The user could also add the

```
namespace MyNS {

void swap(MyC & x, MyC & y)
         noexcept(std::swap(x,y))
{
    return std::swap(x,y);
}

} //namespace
```

so that there is no more need to introduce `std::swap`.

```
MyNs::MyC a, b;
MyNs::swap(a,b)
```

Implementability

This proposal needs some compiler magic, either by generating directly the `swap` function or by providing the reflection traits as e.g. in N4428 or N4451.

Open Questions

The authors would like to have an answer to the following points if there is any interest at all in this proposal:

- Do we want a default or a reflection solution?
- Quiz of generating also the `swap` member function.

Summary

Defaulting `swap` operations is simple, removes a common annoyance. It is completely compatible. In particular, the existing facilities for defining and suppressing those operations are untouched.

Acknowledgments

Thanks to Tomasz Kamiński for its clear identification of the types that are subject to this kind of default generation.

Thanks to all those that have commented the idea on the std-proposals ML helping to the proposal in

general, in particular Andrzej Krzemieński.

References

- N3746 Proposing a C++1Y Swap Operator, v2

  http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2013/n3746.pdf

- N4428 Type Property Queries (rev 4)

  http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2015/n4428.pdf

- N4451 Static reflection

  http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2015/n4451.pdf

- N4475 Default comparisons (R2)

  http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2015/n4475.pdf

- N4511 "Adding [nothrow-]swappable traits, revision 1

  http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2015/n4511.html

- N4532 Proposed wording for default comparisons

  http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2015/n4532.html

- P0017R0 Extension to aggregate initialization

  http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2015/p0017r0.html