Document Number: N3955
Date: 2014-02-25
Project: Programming Language C++ Evolution Working Group
Reply to: Andrew Tomazos andrewtomazos@gmail.com

Group Member Specifiers

0.1 Proposal [proposal]

0.1.1 Informal Summary [proposal.summary]

We propose allowing a space-separated sequence of specifiers to be placed after an access specifier in a class definition. The specifiers so placed are applied to each member declaration in the group following the access-specifier.

0.1.2 Motivation and Example [proposal.motivation]

1 In class definitions, many times there are contiguous sequences of member declarations that have common specifiers. Under the proposal these common specifiers can be specified once at the start of a group of member declarations, rather than repeated for each member declaration. This can in some cases lead to cleaner, safer, easier-to-write, easier-to-read and easier-to-maintain code.

So instead of:

```
class foo : public bar
{
public:
    explicit foo(int);
    explicit foo(float);
    explicit foo(double);

protected:
    virtual T f() const override;
    virtual U g() const override;
    virtual V h() const override;
    virtual W i() const override;

private:
    static constexpr int X = 123;
    static constexpr float Y = 42.0f;
    static constexpr double Z = 123.0;
};
```

You could with group member specifiers equivalently write:

```
class foo : public bar
{
public explicit:
    foo(int);
    foo(float);
```

```
    foo(double);

protected virtual const override:
    T f();
    U g();
    V h();
    W i();

private static constexpr:
    int X = 123;
    float Y = 42.0f;
    double Z = 123.0;
};
```

0.1.3 Existing Practice [proposal.practice]

1 Access-specifier is already used in C++ with the group member specifier syntax, so the proposal simply extends that way to the other kinds of specifiers.

2 Qt uses group member specifier syntax to allow groups of members to be specified as signals or slots.

3 The D programming language has a similar group member specifier feature that allows groups of declarations to be specified as a group.

0.1.4 Design Decisions [proposal.design]

1 The proposal is mostly straight-forward, but there were a couple of design decisions that were considered.

2 The idea of also conversely providing access-specifier in the non-group position was considered (like in Java). It was decided to leave this out of this proposal as a number of corner cases arose, and the access-specifier provides a nice introducer token for efficient parsing of the member specifiers anyway. The proposal is forward-compatible with a potential future proposal that includes access-specifier in decl-specifier-seq and subsumed access-specifier into member-specifier.

3 As the proposal relates to specifiers, discussion about it also wandered into revisiting decisions about the syntax of specifiers themselves. For example, would have abstract been a better pure-specifier than =0 in C++? Why do some specifiers appear at the end of a declaration in C++, and some at the beginning? And so on. It was decided that any changes or additions to specifiers in C++ should come from a different proposal. This proposal does not introduce any new specifiers, and the tokens used for specifiers are identical whether used in normal position or in group member position. It was felt that this is a valuable property of the proposal, and that this makes the syntax easy to understand and learn.

4 What to do with the cv-qualifiers was also considered. In a declaration there are numerous places where const can be placed. For example:

```
struct C
{
    const int* f();
    int* const f();
    int* f() const;

public const:
    int x;    // means const int x
    int* p;   // means int* const p;
    int* f(); // means int* f() const
};
```

Because of this ambiguity, the idea of not including the cv-qualifiers as possible group member specifiers was considered, but it was decided to define them as applying to the outermost part of the declarations. That is, for functions it applies to the implicit object parameter, and for non-functions it applies in the same way as through a typedef T that is declared const T (like via constexpr). This is because of the compelling

and common use case of declaring the query/accessor functions of a class together, which are all const. Also, the use case of immutable class types in which all the data members are const at the outermost level.

5 We considered what to do if a specifier is not applicable to a declaration, or if the declaration already has the specifier. It was decided to leave all these cases ill-formed. If we find in practice that some of these ill-formed use cases are desired we can always add them later. The reverse is not so, as it would break backward-compatibility. For example:

```
struct C
{
public explicit:
    explicit C(); // ill-formed: multiple explicit specifiers
    int x;        // ill-formed: explicit not allowed on variable
    using X = T; // ill-formed: alias declarations, static asserts and using declarations cannot have group member specifiers
};
```

6 Finally we considered a more elaborate syntax where member specifiers could be specified over a brace-enclosed list of member declarations, and so member specifiers could be combined and nested into a tree. It was felt that for ease of reading, having the member declarations grouped into a simple partitioning was sufficient. Under the proposal, and as for the existing access-specifer notation, one only has to read up to the nearest access-specifier line above the declaration, and this contains the complete set of specifiers that are applied. The concrete use-cases for a tree of specifiers are not really convincing, and do not justify the extra complexity.

0.2 Technical Specifications [proposal.techspecs]

0.2.1 Grammar Additions [proposal.grammar]

member-specification:
    member-declaration member-specification$_{opt}$
    access-specifier $\underline{\textit{member-specifier-seq}_{opt}}$: *member-specification*$_{opt}$

member-specifier-seq:
    member-specifier member-specifier-seq$_{opt}$

member-specifier:
    cv-qualifier
    storage-class-specifier
    function-specifier
    virt-specifier
    pure-specifier
    `friend`
    `typedef`
    `constexpr`
    `= delete`
    `= default`

0.2.2 Member Specifiers [class.memspec]

1 Each *member-specifier* is applied to each target *member-declaration* up until the next *access-specifier* or the end of the class. Each target declaration shall be one that can have a *decl-specifier-seq*. For *cv-qualifiers*, target declarations of function type shall be as if the function is declarared with the qualifier in the suffix of its *parameters-and-qualifiers*, and for non-functions it shall be applied to the top level declarator of the type. For *virt-specifier*, *pure-specifier*, *= delete* and *= default* the target declaration shall of function type and the member specifier is inserted at the appropriate place. For the remaining member-specifiers, they are inserted into the *decl-specifier-seq* of the target declaration. If a member specifier is already present in a target declaration, the program is ill-formed.

0.3 More Examples [proposal.examples]

```
// EXAMPLE 1: WITHOUT MEMBER SPECIFIERS

class IXMLProcessor
{
public:
    virtual void startDocument() = 0;
    virtual void endDocument() = 0;

    virtual void startElement(const QName *name) = 0;
    virtual void endElement() = 0;
    virtual void attribute(const QName *name, const String &text) = 0;

    virtual void text(const String &text) = 0;
    virtual void comment(const String &text) = 0;

    virtual ~IXMLProcessor() {}
};

class MyXMLProcessor : public IXMLProcessor
{
public:
    virtual void startDocument() override;
    virtual void endDocument() override;

    virtual void startElement(const QName *name) override;
    virtual void endElement() override;
    virtual void attribute(const QName *name, const String &text) override;

    virtual void text(const String &text) override;
    virtual void comment(const String &text) override;
};

// EXAMPLE 1: WITH MEMBER SPECIFIERS

class IXMLProcessor
{
public virtual = 0:
    void startDocument();
    void endDocument();

    void startElement(const QName *name);
    void endElement();
    void attribute(const QName *name, const String &text);

    void text(const String &text);
    void comment(const String &text);

public:
    virtual ~IXMLProcessor() {}
};

class MyXMLProcessor
{
public virtual override:
    void startDocument();
```

```
    void endDocument();

    void startElement(const QName *name);
    void endElement();
    void attribute(const QName *name, const String &text);

    void text(const String &text);
    void comment(const String &text);
};

// EXAMPLE 2: WITHOUT MEMBER SPECIFIERS

template<class T>
class numeric_limits
{
  public:
    static constexpr bool is_specialized = false;
    static constexpr T min();
    static constexpr T max();
    static constexpr T lowest();
    static constexpr int digits = 0;
    static constexpr int digits10 = 0;
    static constexpr bool is_signed = false;
    static constexpr bool is_integer = false;
    static constexpr bool is_exact = false;
    static constexpr int radix = 0;
    static constexpr T epsilon();
    static constexpr T round_error();
    static constexpr int min_exponent = 0;
    static constexpr int min_exponent10 = 0;
    static constexpr int max_exponent = 0;
    static constexpr int max_exponent10 = 0;
    static constexpr bool has_infinity = false;
    static constexpr bool has_quiet_NaN = false;
    static constexpr bool has_signaling_NaN = false;
    static constexpr float_denorm_style has_denorm = denorm_absent;
    static constexpr bool has_denorm_loss = false;
    static constexpr T infinity();
    static constexpr T quiet_NaN();
    static constexpr T signaling_NaN();
    static constexpr T denorm_min();
    static constexpr bool is_iec559 = false;
    static constexpr bool is_bounded = false;
    static constexpr bool is_modulo = false;
    static constexpr bool traps = false;
    static constexpr bool tinyness_before = false;
    static constexpr float_round_style round_style = round_toward_zero;
};

// EXAMPLE 2: WITH MEMBER SPECIFIERS

template<class T>
class numeric_limits
{
  public static constexpr:
```

```
    bool is_specialized = false;
    T min();
    T max();
    T lowest();
    int digits = 0;
    int digits10 = 0;
    bool is_signed = false;
    bool is_integer = false;
    bool is_exact = false;
    int radix = 0;
    T epsilon();
    T round_error();
    int min_exponent = 0;
    int min_exponent10 = 0;
    int max_exponent = 0;
    int max_exponent10 = 0;
    bool has_infinity = false;
    bool has_quiet_NaN = false;
    bool has_signaling_NaN = false;
    float_denorm_style has_denorm = denorm_absent;
    bool has_denorm_loss = false;
    T infinity();
    T quiet_NaN();
    T signaling_NaN();
    T denorm_min();
    bool is_iec559 = false;
    bool is_bounded = false;
    bool is_modulo = false;
    bool traps = false;
    bool tinyness_before = false;
    float_round_style round_style = round_toward_zero;
};
```

// EXAMPLE 3: WITHOUT MEMBER SPECIFIERS

```
template<typename CharT, typename Traits, typename Alloc>
class basic_string
{
public:
      typedef Traits traits_type;
      typedef typename Traits::char_type value_type;
      typedef Alloc allocator_type;
      typedef cat::size_type size_type;
      typedef cat::difference_type difference_type;
      typedef cat::reference reference;
      typedef cat::const_reference const_reference;
      typedef cat::pointer pointer;
      typedef cat::const_pointer const_pointer;
      typedef normal_iterator<pointer, basic_string> iterator;
      typedef normal_iterator<const_pointer, basic_string> const_iterator;
      typedef std::reverse_iterator<const_iterator> const_reverse_iterator;
      typedef std::reverse_iterator<iterator> reverse_iterator;
};
```

// EXAMPLE 3: WITH MEMBER SPECIFIERS

```
template<typename CharT, typename Traits, typename Alloc>
class basic_string
{
public typedef:
      Traits traits_type;
      typename Traits::char_type value_type;
      Alloc allocator_type;
      cat::size_type size_type;
      cat::difference_type difference_type;
      cat::reference reference;
      cat::const_reference const_reference;
      cat::pointer pointer;
      cat::const_pointer const_pointer;
      normal_iterator<pointer, basic_string> iterator;
      normal_iterator<const_pointer, basic_string> const_iterator;
      std::reverse_iterator<const_iterator> const_reverse_iterator;
      std::reverse_iterator<iterator> reverse_iterator;
};
```