Doc No: SC22/WG21/ N1701 04-0141

Project: JTC1.22.32

Date: Friday, September 10, 2004

Author: Francis Glassborow

email: francis@robinton.demon.co.uk

Regularizing Initialization Syntax (revision 1)

1 The Problem

C++ has three distinct syntactic forms for initialization. However the forms are not interchangeable, there are cases where only assignment style syntax is allowed (in conditionals such as `if`, `while` and the second part of `for`) and there are places where only function style is allowed (constructor initializer lists) and places where the two forms have different semantics (conversion initialization). And finally we have the case where a potential ambiguity is resolved in favor of a function declaration.

Currently I am aware of the following different cases:

Case	assignment style	function style	brace initializer
Variable definition	yes	yes	yes
conditional	yes	no	no
array definition	no	no	yes
constructor init list	no	yes	no
new expression	no	yes	no
new[] expression	no	no	no
Function value parameters init by call	as if	no	no
Function return init	as if	no	no

Note the last two, because these are defined to be copy initialization even if the type of the argument exactly matches the type of the parameter. Copy initialization is only defined for the assignment style. Function style is always direct initialization.

We also have a limited range of places where brace initialization is allowed, for example the following are all equivalent:

```
int i(0);
int i = 0;
int i = {1};
```

However the equivalence of the following is dependent on the definition of `mytype`:

```
mytype mt(0);
```

```
mytype mt = 0;
```

```
mytype mt = {0};
```

And the possibility of:

```
mytype mt_array[] = {1, 2, 3, 4};
```

depends on `mytype` being a POD.

We lack any way to initialize a dynamic array, even where that is an array of POD type.

This makes C++ harder to teach and seems to give us nothing in exchange.

2 The Proposal

a) Change the grammar so that brace initialization of the form:

```
mytype mt{<constructor args>};
```

becomes a correct way to create a single object of `mytype`.

Allow that form (or whatever syntax we select) everywhere and define all other initialization syntax in terms of it.

b) Provide the following syntax for initializing an array:

```
mytype mt[]{   {<ctor args>},
               {<ctor args>},
// with as many brace intializers as required.
                };
```

In the event of any of the constructors throwing an exception, any already constructed elements will be destroyed (exactly as is currently the case if a default ctor throws during the creation of an array)

c) Allow the invoking of different constructors for different elements of an array as determined by the types of the arguments provided in a specific brace initializer.

d) Allow explicit specification of array size with default initialization of all elements not explicitly provided with the above direct brace initializer syntax.

e) Remove the concept of copy initialization as a special process and just provide it as a form of direct initialization using a copy constructor.

3) Discussion

If the direct brace initialization syntax is accepted I am not sure that we need to require both assignment and function style initialization be interchangeable in all contexts, only that brace initialization be always acceptable. However the removal of the distinction between direct and copy initialization does have impact on some potentially existing code, in particular whether an explicit qualified constructor is considered or not (currently 'yes' for direct initialization and 'no' for copy initialization). For example:

```
class X {
```

```
public:
    X(short s){}
    explicit X(long l) {}
};

int main(){
    X x(3); //A
    X y=3; //B
}
```

Line A is ambiguous because we do not prefer conversion of an `int` to `long` as better than conversion to `short`. Line B is surprising because it is not ambiguous (`explicit` constructors are not considered for implicit conversions) but selects a narrowing conversion. This provides a problem because it means that the proposed change potentially impacts on existing code. It also strengthens my feeling that we should teach newcomers to use the function style syntax or even better have a new syntax that is universally usable.

There are several major advantages to the new syntax. It provides a way to remove the problem with confusion between function declarations and object definitions:

```
mytype mt{};
```

will be a definition of a default initialised object while:

```
mytype mt();
```

will continue to be a declaration of a function.

The new syntax allows the creation of initialized dynamic arrays:

```
mytype * mt_ptr = new mytype[]{< list of brace initializers providing ctor args>};
```

It would also allow the creation of temporary arrays. Given:

```
void foo(mytype const * const array, size_t array_size);
```

It would be nice to explore the possibility of writing

```
foo(mytype[10]{<ctor args list>});
```

as equivalent to:

```
mytype temp[10]{<ctor args list>};

foo(temp, 10);
```

instead of having to write:

```
foo(mytype[10]{<ctor args list>}, 10)
```

It might be that we could use some form of template such as `pair`, to provide this functionality via the library rather than try to produce a syntax for inclusion in the core of the language.

Given these syntax extensions and possible library support we could, for example, provide a constructor for `std::vector` so that the following creates an initialized vector:

```
std::vector<int> v(int[]{1, 2, 3, 4});
```

We should consider the extent to which we should import the C99 designated initializer syntax (ISO/IEC 9899:1999, 6.7.8)

We should also consider of modification to the for-statement syntax so that iteration over the values of a temporary array can be supported possibly by something like:

```
for(T item : []{2, 3, 5, 7}){
     // process the T objects in the anonymous array
     // using item to refer to the current instance
}
```

Conclusion

I think that this proposal addresses several distinct issues in a beneficial way. It will require considerable work to incorporate it into the WP, however some of that work (such as removing the need to distinguish between direct and copy initialization) will reduce the overall size of the WP while providing a more powerful and uniformly applicable initialization syntax with clear semantics, and one that is specific to defining objects as opposed to declaring functions.

Changes to the Working Paper

These are not provided at this point. The volume of changes to the WP means that the work needs to be agreed in principle before the specific changes are drafted. The following will need attention:

5.3.4 paragraphs 16 & 17

8.5 paragraph 1 add to grammar and extend example in paragraph 2. The remainder of 8.5, 8.5.1 (aggregates), 8.5.2(character arrays) and 8.5.3 (references) will need review and substantial alteration.

12.6.1 will need minor modification to accommodate the new syntax as an option

12.6.2 will require modification to accommodate initialization of array members.

5.2 The grammar may need adjustment to accommodate temporary arrays used as arguments in function calls.

5.2.2 (function call) will need attention

12.2 (temporary objects) example in paragraph 5 may need extension to include alternative syntax.

6.4 (selection statements) The grammar of condition needs attention

6.5 (iteration statements) will need review to ensure that changes elsewhere are sufficient.