Doc No: SC22/WG21/N1584 = 04-0024

Project: JTC1.22.32

Date: Thursday, February 12, 2004

Author: Francis Glassborow

email: francis@robinton.demon.co.uk

Regularizing Initialization Syntax

1 The Problem

C++ has two distinct syntactic forms for initialization. However the two 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.

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

2 The Proposal

Change the grammar so that in as far as possible, wherever one syntactic form is valid the alternative form is also and has exactly the same semantic result. The following is a list of the changes that need to be considered:

1) Make function-style and assignment-style initialization in variable declarations equivalent in all cases so that:

```
mytype mt = expr;
```

and

```
mytype mt(expr);
```

are equivalent in all cases where the latter cannot parse as a function declaration. I.e. we remove the implicit conversion followed by copy construction from the former case. The only remaining 'irregularity is that the former syntax is never a function declaration.

(note this does not resolve the ambiguity of variable versus function declaration)

2) Allow assignment-style initialization in constructor initialiser lists so:

```
mytype::mytype(int val): i(val), j(val) {}
```

can be written as:

```
mytype::mytype(int val): i = val, j = val {}
```

3) Allow function-style initialization in conditionals so:

```
if(mytype mt = expr) dosomething;
```

can be written as:

```
if(mytype mt(expr)) dosomething;
```

3 Discussion

1) Above is the problem case because there are some subtle cases. The most notable of these is that in:

```
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.

Possibly we might also consider the syntax for default arguments but that is a distinct concept and requiring assignment syntax seems reasonable in this case.

There is very little that we can do about the potentially ambiguous parse of initialized variable versus function declaration from a Standards perspective because there does not seem to be even a transitional path from where we are now to anywhere else that is useful. Possibly we might consider allowing use of 'auto' to disambiguate in favor of an initialized variable so that:

```
auto mytype mt();
```

defines `mt` as a default initialized `mytype` object. I would like to strongly encourage implementors to default to issuing warnings whenever a function appears to be declared at block scope as I think it is rare for modern code to declare a function at other than at namespace or class scope.

Another issue that has been raised by members of the BSI C++ Panel is that there is actually a third initialization form; brace initialization of aggregates. It would be nice to consider how we might bring that in from the cold and allow it in constructor initializer lists. I think it is doable, and doing so would certainly be appreciated by those wishing to initialize containers at the point of definition.

Changes to the Working Paper

These are not provided at this point. It seems more important to agree the mechanism.