Doc No: SC22/WG21/N1467
J16/03-0050
Date: 01-Apr-2003
Project: JTC1.22.32
Reply to: Daniel Gutson
danielgutson@hotmail.com

NON DEFAULT CONSTRUCTORS FOR ARRAYS

1. The problem

When declaring arrays, their elements are initialized with the default constructor (for objects), or with garbage (for PODs). There is no way of specifying other type of construction or initialization mechanism.
This issue has two consequences:

1- for non constant objects: a) requires a second step (post construction) to iterate over recently created objects for initializing them which represents additional code, and probably not as efficient as the compiler would automatically implement it. The size and complexity of the additional required code grows with the number of dimensions in the array, when it could be made by the compiler.
b) requires that objects to be allocated in this way (as arrays) have to provide an interface (i.e. projectors) for the post-construction initialization; however, references and const members cannot be used if they must be initialized thru parameters of constructors in the initializer list. Moreover, if the object does not provide a visible/available default constr., it cannot be "arrayed".

2- for constant objects: can rarely be used as arrays as far as they will not be modified after construction, mandatoryly requiring initializing information from constr-parameters. Even for PODs when a constant bunch of elements must occupy a space with a fixed value.

- How are people addressing, or working around, the problem today? For constructing non constant arrays, there are two common ways:

    1) do thru "for" statements (as many nested as dimensions the array has), and "manually" initialize individually each element (in the 'for' body).
    2) For PODs [when all the elements must have the same value], it's also common to initialize them using a memory function (i.e. memset) when possible.

- Why is the problem important?
    a) Because, when default-constructor is available for the purpose, requires additional workarounds (see above), impacting in performance, productivity, and effort. Also, might impact in readability of classes interface when special accessing mechanisms

are required when the information should come in the (non-default) ctor.

b) And arrays of constant objects are important –for example- for performance when preprocess phase (and not time-critical) could perform calculus as a pre-processed data repository for a "later" time-critical phase. This still can be done from design, but conceptually such data should be "self-calculated" (in ctor) and remain constant for the rest of the program (even for the pre-process phase). This concept cannot be exactly reflected in code.

Categories:

* improve support for systems programming (performance, coding overhead).
* improve support for library building (releases the user from additional tasks mentioned above, requiring additional knowledge of the object's interfaces while the work could be performed by the library implementation in the [non-def.] ctors).
* remove embarrassments (code can be cleaner as far as initialization can still be performed in ctors, instead of the instantiation/use place).

2. The proposal

Enable non-default constructors for arrays, placing the parameters after the brackets

2.1. Basic cases:

```
int mat [10] [10] (0); //invokes int (int) for each element

BlocksInfo blocks [10] (database); //invokes BlocksInfo (database&);
```

2.2. Advanced cases:

Dynamic memory, dynamic-state of constructor parameters:

```
const X *x = new X [length](++position);          //see 3.2
      Y   y [10](f());        // see 3.2
```

3. Interactions and Implementability

3.1. Interactions

- Non-default constructors usage for array elements
- Proposed syntax not currently supported, therefore no incompatibility issues presented
- Non-default constructor for array members from the initializer list: the syntax proposed for this is the array member name, followed for as many empty brackets as dimensions it has, and then the constructor invocation:

```
MyClass: My Class ( ):
```

_myArray [][] (data) // contructs “_my Array” elements with “data”

3.2. Implementability

- when constructing arrays of PODs, and constant (or non-volatile parameters are used), the compiler might allocate the code first and then stamp the value over.
  But, when one of the following situations occur, the compiler should invoke the construction statement for each element (even if a dynamic length was specified thru “new” operator):
- the parameters vary their state for each ctor-call (i.e. functions or increment/decrement operators as the examples of 2.2)
- the constructor modifies the state of any component of the program

In general, we could resume these points saying “when the individual construction statement modifies the state of any component of the system”.