Memory Allocation: Proposed Working Paper Changes

Gregory Colvin
Information Management Research
gregor@netcom.com

This paper reviews the Standard C++ library memory allocation facilities for inconsistencies and unspecified behavior. It is organized by Section number (pre-Valley Forge draft) and symbolic name. I propose that:

- *operator new* be specified to report failure by throwing *bad_alloc*, and never to return a null pointer;
- a special *operator new (const nothrow&)* be specified to report failure by returning a null pointer;

18.4.1.1.1 operator new [lib.op.new]

This *operator new* function is only partially specified. The A.R.M. specifies that *operator new* will return zero if no memory can be allocated and no *new_handler* is set, but the working paper leaves this behavior implementation defined. I have proposed already (94-0167/N0554) that this function be specified to throw *bad_alloc*. Thus I recommend the following wording:

```
void* operator new(size_t size) throw(bad_alloc);
```

The *allocation function* (3.6.3.1) called by a *new-expression* (5.3.4) to allocate *size* bytes of storage suitably aligned to represent any object of that size.

Replaceable: a C++ program may define a function with this function signature that displaces the default version defined by the Standard C++ library.

Required behavior: return a pointer to dynamically allocated storage (3.6.3) or else throw a *bad_alloc* exception.

Default behavior:

- executes a loop. Within the loop, the function first attempts to allocate the requested storage. Whether the attempt involves a call to the Standard C library function *malloc* is unspecified.
- Returns a pointer to the allocated storage if the attempt is successful. Otherwise, if the last argument to *set_new_handler()* was a null pointer, throw *bad_alloc*.
- Otherwise, the function calls the current *new_handler* (_lib.new.handler_). If the called function returns, the loop repeats.
- The loop terminates when an attempt to allocate the requested storage is succesful or when a called *new_handler* does not return.

The working paper, in footnote 92, states that "A common extension when *new_handler* is a null pointer is for *operator new(size_t)* to return a null pointer, in accordance with many earlier implementations of C++". This footnote is intended to provide a transistion path for older code, but instead just leaves it uncertain whether or not a *operator new* may yield a null pointer, and whether or not *set-new-handler(0)* is defined. This footnote should be removed, as the above changes ensure that *operator new(size_t)* may not return a null pointer. The intent of providing a transition path for old code can be satisfied by specifying a special operator:

```
class nothrow {};
void* operator new(size_t size,const nothrow&) throw();
```

Allocate *size* bytes of storage suitably aligned to represent any object of that size.

Replaceable: a C++ program may define a function with this function signature that displaces the default version defined by the Standard C++ library.

Required behavior: return a pointer to dynamically allocated storage (3.6.3) or else return a null pointer.

Default behavior:

- executes a loop. Within the loop, the function first attempts to allocate the requested storage. Whether the attempt involves a call to the Standard C library function *malloc* is unspecified.
- Returns a pointer to the allocated storage if the attempt is successful. Otherwise, if the last argument to *set_new_handler()* was a null pointer, return a null pointer.
- Otherwise, the function calls the current *new_handler* (_lib.new.handler_). If the called function returns, the loop repeats.
- The loop terminates when an attempt to allocate the requested storage is succesful or when a called *new_handler* does not return. If the called *new_handler* terminates by throwing a *bad_alloc* exception the function returns a null pointer.

18.4.1.1.2 operator delete [lib.op.delete]

In accordance with the above changes to *operator new()*, the *operator delete()* function should be specified as:

```
void operator delete(void* ptr) throw();
```

The *deallocation function* (3.6.3.2) called by a *delete-expression* to render the value of *ptr* invalid.

Replaceable: a C++ program may define a function with this function signature that displaces the default version defined by the Standard C++ library.

Required behavior: accept a value of *ptr* that is null or that was returned by an earlier call to *operator new()*.

Default behavior:

- For a null value of *ptr*, do nothing.
- Any other value of *ptr* shall be a value returned by an earlier call to a default *operator new()* function. For such a non-null value of *ptr*, reclaims storage allocated by the earlier call to *operator new()*.

It is unspeficied under what conditions part or all of such reclaimed storage is allocated by a subsequent call to *operator new()* or any of *malloc*, *calloc*, or *realloc*, declared in *<cstdlib>*.

18.4.1.3 operator new[] **[lib.op.new.array]**

18.4.1.4 operator delete[] **[lib.op.delete.array]**

These functions need throw specifications:

```
void* operator new[] (size_t size) throw(bad_alloc);
void operator delete[] (void* ptr) throw();
```

Also, a *nothrow* version of array new is needed:

```
void* operator new[] (size_t size,const nothrow&) throw();
```

18.4.1.5.1 Placement operator new **[lib.placement.op.new]**

18.4.1.5.2 Placement operator new[] **[lib.placement.op.new.array]**

These functions need throw specifications:

```
void* operator new (size_t size, void* ptr) throw();
void* operator new[] (size_t size, void* ptr) throw();
```

18.4.2.2 Type new_handler **[lib.new.handler]**

Given the changes to *operator new()* no default *new-handler* is needed, so paragraph 3, reading "Default behavior: ..." can be removed.

20.3.1 The default allocator **[lib.default.allocator]**

It is unspecified how this class obtains and invalidates memory. I recommend that the *allocator::allocate()* function allocate memory by calling *operator new(size_t)* and that the *allocator::deallocate()* function reclaim memory by calling *operator delete().* This template is still changing, so I will not attempt to give exact wording.

20.3.3.1 allocate **[lib.allocate]**

20.3.3.2 deallocate **[lib.deallocate]**

It is unspecified how these functions obtain and invalidate memory. If they are to be retained at all I recommend that the *allocate()* function allocate memory by calling *operator new(size_t)* and that the *deallocate()* function reclaim memory by calling *operator delete().*