What is "placement new" and why would I use it?

There are many uses of placement new. The simplest use is to place an object at a particular location in memory. This is done by supplying the place as a pointer parameter to the new part of a new expression:

```
#include <new>       // Must #include this to use "placement new"
#include "Fred.h"    // Declaration of class Fred

void someCode()
{
  char memory[sizeof(Fred)];     // Line #1
  void* place = memory;          // Line #2

  Fred* f = new(place) Fred();   // Line #3 (see "DANGER" below)
  // The pointers f and place will be equal

  ...
}
```

Line #1 creates an array of sizeof(Fred) bytes of memory, which is big enough to hold a Fred object. Line #2 creates a pointer place that points to the first byte of this memory (experienced C programmers will note that this step was unnecessary; it's there only to make the code more obvious). Line #3 essentially just calls the constructor Fred::Fred(). The this pointer in the Fred constructor will be equal to place. The returned pointer f will therefore be equal to place.

ADVICE: Don't use this "placement new" syntax unless you have to. Use it only when you really care that an object is placed at a particular location in memory. For example, when your hardware has a memory-mapped I/O timer device, and you want to place a Clock object at that memory location.

DANGER: You are taking sole responsibility that the pointer you pass to the "placement new" operator points to a region of memory that is big enough and is properly aligned for the object type that you're creating. Neither the compiler nor the run-time system make any attempt to check whether you did this right. If your Fred class needs to be aligned on a 4 byte boundary but you supplied a location that isn't properly aligned, you can have a serious disaster on your hands (if you don't know what "alignment" means, please don't use the placement new syntax). You have been warned.

You are also solely responsible for destructing the placed object. This is done by explicitly calling the destructor:

```
void someCode()What is "placement new" and why would I use it?
```

There are many uses of placement `new`. The simplest use is to place an object at a particular location in memory. This is done by supplying the place as a pointer parameter to the `new` part of a `new` expression:

```
#include <new>          // Must #include this to use "placement new"
#include "Fred.h"       // Declaration of class Fred

void someCode()
```

```
{
  char memory[sizeof(Fred)];      // Line #1
  void* place = memory;           // Line #2

  Fred* f = new(place) Fred();    // Line #3 (see "DANGER" below)
  // The pointers f and place will be equal

  ...
}
```

Line #1 creates an array of `sizeof(Fred)` bytes of memory, which is big enough to hold a `Fred` object. Line #2 creates a pointer `place` that points to the first byte of this memory (experienced C programmers will note that this step was unnecessary; it's there only to make the code more obvious). Line #3 essentially just calls the constructor `Fred::Fred()`. The `this` pointer in the `Fred` constructor will be equal to `place`. The returned pointer `f` will therefore be equal to `place`.

ADVICE: Don't use this "placement `new`" syntax unless you have to. Use it only when you really care that an object is placed at a particular location in memory. For example, when your hardware has a memory-mapped I/O timer device, and you want to place a `Clock` object at that memory location.

DANGER: You are taking *sole* responsibility that the pointer you pass to the "placement `new`" `operator` points to a region of memory that is big enough and is properly aligned for the object type that you're creating. Neither the compiler nor the run-time system make any attempt to check whether you did this right. If your `Fred` class needs to be aligned on a 4 byte boundary but you supplied a location that isn't properly aligned, you can have a serious disaster on your hands (if you don't know what "alignment" means, *please* don't use the placement `new` syntax). You have been warned.

You are also solely responsible for destructing the placed object. This is done by explicitly calling the destructor:

```
void someCode()
{
  char memory[sizeof(Fred)];
  void* p = memory;
  Fred* f = new(p) Fred();
  ...
  f->~Fred();      // Explicitly call the destructor for the placed object
}
```

This is about the only time you ever explicitly call a destructor.


```
{
  char memory[sizeof(Fred)];
  void* p = memory;
  Fred* f = new(p) Fred();
  ...
  f->~Fred();   // Explicitly call the destructor for the placed object
}
```

This is about the only time you ever explicitly call a destructor.