



Key Issues

- "new" error error handler
 - allows application code to be called in the event of a memory allocation failure during new.
 - eliminates need to constantly check for failed memory allocations if program termination is acceptable.
- override new/delete
 - provides the capability to develop custom global dynamic memory allocation.
- placement new
 - provides the capability to custom-locate new objects
- coordinate new, delete, new[] and delete[]
 - run-time errors can occur when mixing single element and array news/deletes.
- overload global new
 - allow for custom usages of new (virtual constructors)
- overload new and delete for an individual class
 - provides ability to customize allocation on a per class basis

new Error Handler

- allows application code to be called in the event of a memory allocation failure during `new`.
- eliminates need to constantly check for failed memory allocations if program termination is acceptable.

```
void outOfHeapSpace() {
    static int firstTime = 0;
    if (!firstTime){
        //try to recover
    }
    else {
        //do some cleanup
        exit(1);
    }
}

void main() {
    set_new_handler(outOfHeapSpace);
    do {
        allocations += 1;
        cerr << ".";
    } while(new char[alot]);
    cout << "done\n";
}
```

Construction and Memory Allocation

■ new operator

```
String *s = new String("Hello World");
```

- Allocates enough memory for object by calling operator new()

```
void* rawMemory = operator new(sizeof(String));
```

- Calls constructor function to initialize object

```
String::String(rawMemory, "Hello World") //theory
```

- Changes the type pointer of the memory

```
String *s = static_cast<String*>(rawMemory);
```

- Cannot be overridden

■ operator new() function

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

- Allocates amount of memory requested

- Calls no constructors

- Can be overridden at global and class scope

- Can be overloaded with size_t being first parameter

```
void* operator new(size_t size, int fill);
```

```
String *s = new ('A') String("Hello World");
```

```
//passes size=sizeof(String) and fill='A' to op-new
```

Placement new

- Users cannot call constructors directly
- Placement new allows object to be constructed within a pre-allocated buffer

```
#include <new.h>    //defines ...  
    //void* operator new(size_t, void* loc) { return loc; }
```

```
String *s3 = new (rawMemory) String("Hello World");
```

- Useful for share memory or memory mapped I/O implementations

Deletion and Memory Deallocation

- delete operator

```
delete s;
```

- Calls destructor function to shutdown object

```
s->~String();
```

- Deallocates memory assigned to object by calling operator delete

```
operator delete(s);
```

- operator delete

```
void operator delete(void* location);
```

- Deallocates memory location requests

- Calls no destructors

- Can be overridden at global and class scope

- SHOULD BE PAIRED WITH operator new() IMPLEMENTATION!!!

- Cannot be overloaded

Allocating Arrays

- Behaves differently from allocating single object

```
String *sa = new String[2];
```

- Memory allocated using operator new[]() - size overhead added

```
void* rawBuffer = operator new[](
    sizeof(size_t) + numElements*sizeof(String) );
```

- Size information stored at head of array

```
*(int*)rawBuffer = numElements;
```

- Constructors are called

```
for(int i=0;i<numElements;i++)
    String::String(rawBuffer+sizeof(size_t)); //theory
```

- Changes type pointer of memory

```
sa = static_cast<String*>(rawBuffer+sizeof(size_t))
```

- operator new[]() function

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

- Allocates amount of memory requested - size overhead has been added
- Calls no constructors
- Can be overridden at global and class scope
- Can be overloaded with size_t being first parameter