

# Chapter 13

## Optimize Memory Management

성무진

# Intro

- **메모리 관리자:** ‘동적 변수’에 메모리 할당을 감독하는 C++ 런타임 시스템의 함수와 자료구조의 집합
- 대부분의 C++ 프로그램에서, 메모리 관리자의 함수들은 상당히 **Hot**하다. 만약 메모리 관리자의 성능이 향상된다면, 프로그램의 전반적인 효과를 가져올 것이다.
- 큰 프로그램에서, 메모리 관리를 최적화해서 얻는 성능 향상은 무시할만한 수준에서 30% 사이이다.

# C++ Memory Management API Refresher:

## 동적 변수의 Life Cycle



# C++ Memory Management API Refresher:

## 동적 변수의 Life Cycle

- 1) 할당(Allocate)
- 2) 배치(Place)
- 3) 사용(Use)
- 4) 소멸(Destroy)
- 5) 해제(Free)

메모리 관리자가 프로그램에 메모리  
영역에 대한 포인터를 반환  
(malloc()이나 operator new()가  
이에 해당)

# C++ Memory Management API Refresher:

## 동적 변수의 Life Cycle

- 1) 할당(Allocate)
- 2) 배치(Place)
- 3) 사용(Use)
- 4) 소멸(Destroy)
- 5) 해제(Free)

동적 변수의 초기값을 할당된  
메모리에 집어 넣음  
(Constructor)

# C++ Memory Management API Refresher:

## 동적 변수의 Life Cycle

- 1) 할당(Allocate)
- 2) 배치(Place)
- 3) 사용(Use)
- 4) 소멸(Destroy)
- 5) 해제(Free)

프로그램이 동적변수의 값을 읽거나,  
멤버 함수를 호출하거나, 값을 씀

# C++ Memory Management API Refresher:

## 동적 변수의 Life Cycle

- 1) 할당(Allocate)
- 2) 배치(Place)
- 3) 사용(Use)
- 4) 소멸(Destroy)
- 5) 해제(Free)

시스템 리소스를 반환하거나,  
마지막 작업  
(Destructor)

# C++ Memory Management API Refresher:

## 동적 변수의 Life Cycle

- 1) 할당(Allocate)
- 2) 배치(Place)
- 3) 사용(Use)
- 4) 소멸(Destroy)
- 5) 해제(Free)

프로그램이 메모리를  
메모리 관리자에 반환  
(free()나 operator delete())



# C++ Memory Management API Refresher:

## 동적 변수의 Life Cycle (코드)

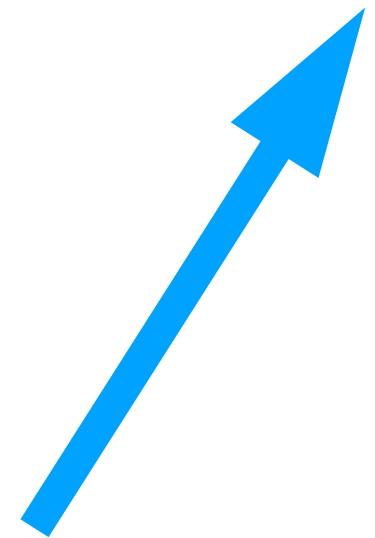
### New-Expression, Delete Expressions(3단계)

```
//! The Life Cycle of Dynamic Variables !//  
auto foo_p = new Foo(123); //1-New-Expression  
foo_p->printA();           //2-Use  
delete foo_p;              //3-Delete-Expression
```

Foo(int a) ctor  
A:123  
~Foo() dtor

### operator new(), operator delete() (5 단계)

```
//! The Life Cycle of Dynamic Variables !//  
auto foo_p = (Foo*)operator new(sizeof(Foo)); //1-Allocate  
new(foo_p) Foo(123);                          //2-Place  
foo_p->printA();                              //3-Use  
foo_p->~Foo();                                //4-Destroy  
operator delete(foo_p);                       //5-Free
```



# C++ Memory Management API Refresher :

## 메모리 관리 함수는 메모리 할당과 메모리 해제를 한다.

- **operator new()**  
동적 변수를 할당(Allocate)

### operator new, operator new[]

Defined in header `<new>`

#### replaceable allocation functions

```
void* operator new ( std::size_t count ); (1)
```

```
void* operator new[]( std::size_t count ); (2)
```

```
void* operator new ( std::size_t count, const std::nothrow_t& tag ); (3)
```

```
void* operator new[]( std::size_t count, const std::nothrow_t& tag ); (4)
```

#### placement allocation functions

```
void* operator new ( std::size_t count, void* ptr ); (5)
```

```
void* operator new[]( std::size_t count, void* ptr ); (6)
```

```
void* operator new ( std::size_t count, user-defined-args... ); (7)
```

```
void* operator new[]( std::size_t count, user-defined-args... ); (8)
```

#### class-specific allocation functions

```
void* T::operator new ( std::size_t count ); (9)
```

```
void* T::operator new[]( std::size_t count ); (10)
```

```
void* T::operator new ( std::size_t count, user-defined-args... ); (11)
```

```
void* T::operator new[]( std::size_t count, user-defined-args... ); (12)
```

new 헤더는  
암시적으로 선언됨

# C++ Memory Management API Refresher :

메모리 관리 함수는 메모리 할당과 메모리 해제를  
한다.

- **operator new()**  
동적 변수를 할당(Allocate)

default new  
:실패시 bad\_alloc

operator new, operator new[]

Defined in header <new>

**replaceable allocation functions**

`void* operator new ( std::size_t count );` (1)

`void* operator new[]( std::size_t count );` (2)

`void* operator new ( std::size_t count, const std::nothrow_t& tag );` (3)

`void* operator new[]( std::size_t count, const std::nothrow_t& tag );` (4)

**placement allocation functions**

`void* operator new ( std::size_t count, void* ptr );` (5)

`void* operator new[]( std::size_t count, void* ptr );` (6)

`void* operator new ( std::size_t count, user-defined-args... );` (7)

`void* operator new[]( std::size_t count, user-defined-args... );` (8)

**class-specific allocation functions**

`void* T::operator new ( std::size_t count );` (9)

`void* T::operator new[]( std::size_t count );` (10)

`void* T::operator new ( std::size_t count, user-defined-args... );` (11)

`void* T::operator new[]( std::size_t count, user-defined-args... );` (12)

# C++ Memory Management API Refresher :

메모리 관리 함수는 메모리 할당과 메모리 해제를  
한다.

- **operator new()**  
동적 변수를 할당(Allocate)

operator new, operator new[]

Defined in header <new>

replaceable allocation functions

`void* operator new ( std::size_t count );` (1)

`void* operator new[]( std::size_t count );` (2)

`void* operator new ( std::size_t count, const std::nothrow_t& tag );` (3)

`void* operator new[]( std::size_t count, const std::nothrow_t& tag );` (4)

placement allocation functions

`void* operator new ( std::size_t count, void* ptr );` (5)

`void* operator new[]( std::size_t count, void* ptr );` (6)

`void* operator new ( std::size_t count, user-defined-args... );` (7)

`void* operator new[]( std::size_t count, user-defined-args... );` (8)

class-specific allocation functions

`void* T::operator new ( std::size_t count );` (9)

`void* T::operator new[]( std::size_t count );` (10)

`void* T::operator new ( std::size_t count, user-defined-args... );` (11)

`void* T::operator new[]( std::size_t count, user-defined-args... );` (12)

**non\_throwing  
new :**  
**실패시 nullptr**



# C++ Memory Management API Refresher :

메모리 관리 함수는 메모리 할당과 메모리 해제를  
한다.

- **operator new()**  
동적 변수를 할당(Allocate)

operator new, operator

Defined in header <new>

**replaceable allocation functions**

```
void* operator new ( std::size_t count );  
void* operator new[]( std::size_t count );  
void* operator new ( std::size_t count, const std::nothrow_t& tag ); (3)  
void* operator new[]( std::size_t count, const std::nothrow_t& tag ); (4)
```

**placement allocation functions**

```
void* operator new ( std::size_t count, void* ptr ); (5)  
void* operator new[]( std::size_t count, void* ptr ); (6)  
void* operator new ( std::size_t count, user-defined-args... ); (7)  
void* operator new[]( std::size_t count, user-defined-args... ); (8)
```

**class-specific allocation functions**

```
void* T::operator new ( std::size_t count ); (9)  
void* T::operator new[]( std::size_t count ); (10)  
void* T::operator new ( std::size_t count, user-defined-args... ); (11)  
void* T::operator new[]( std::size_t count, user-defined-args... ); (12)
```

**placement new :**  
메모리를 할당하지 않고, 배치만!

# C++ Memory Management API Refresher :

메모리 관리 함수는 메모리 할당과 메모리 해제를  
한다.

- **operator delete()**  
할당된 메모리를 해제(Free)

**delete:**  
메모리를 해제 및 반환

operator **delete**, operator **delete[]**

Defined in header `<new>`

---

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

(1)

---

```
void operator delete[]( void* ptr );
```

(2)

---

```
void operator delete ( void* ptr, const std::nothrow_t& );
```

(3)

---

```
void operator delete[]( void* ptr, const std::nothrow_t& );
```

(4)

---

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

(5)

---

```
void operator delete[]( void* ptr, void* );
```

(6)

---

# C++ Memory Management API Refresher :

메모리 관리 함수는 메모리 할당과 메모리 해제를  
한다.

- C 라이브러리의 메모리 관리 함수

## malloc

Defined in header <stdlib.h>

```
void* malloc( size_t size );
```

**malloc:**  
**size만큼 메모리 할당 후**  
**포인터 반환**

## free

Defined in header <stdlib.h>

```
void free( void* ptr );
```

# C++ Memory Management API Refresher :

메모리 관리 함수는 메모리 할당과 메모리 해제를  
한다.

- C 라이브러리의 메모리 관리 함수

## malloc

Defined in header <stdlib.h>

```
void* malloc( size_t size );
```

## free

Defined in header <stdlib.h>

```
void free( void* ptr );
```

**free:**  
pointer의 메모리를  
메모리 관리자에 반환



# C++ Memory Management API Refresher :

메모리 관리 함수는 메모리 할당과 메모리 해제를  
한다.

- C 라이브러리의 메모리 관리 함수

## calloc

Defined in header <stdlib.h>

```
void* calloc( size_t num, size_t size );
```

**calloc:**  
**malloc의 Array 버전**

## realloc

Defined in header <stdlib.h>

```
void *realloc( void *ptr, size_t new_size );
```

**realloc:**  
**메모리 블록의 사이즈 재정의**  
**old block copy**

# C++ Memory Management API Refresher:

## New-Expression은 동적 변수를 생성한다.

### Syntax

---

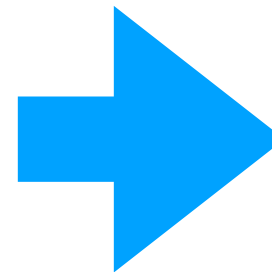
```
::(optional) new (placement_params)(optional) ( type ) initializer(optional) (1)
```

---

```
::(optional) new (placement_params)(optional) type initializer(optional) (2)
```

---

```
//! The Life Cycle of Dynamic Variables3 !//  
auto foo_p = (Foo*)operator new(sizeof(Foo));  
auto a = ::new(foo_p) (Foo)(123);  
auto b = new(foo_p) Foo(123);  
auto c = new Foo(123);  
  
cout<<"Address of foo_p: "<<foo_p<<endl;  
cout<<"Address of a: "<<a<<endl;  
cout<<"Address of b: "<<b<<endl;  
cout<<"Address of c: "<<c<<endl;
```



```
Address of foo_p: 0x7fbe7c500000  
Address of a: 0x7fbe7c500000  
Address of b: 0x7fbe7c500000  
Address of c: 0x7fbe7c500010
```

# C++ Memory Management API Refresher: New-Expressions은 동적 변수를 생성한다.

- 메모리 할당이 실패할 경우

throw std::bad\_alloc --> try-catch로 잡아줌

```
try{
    while(1)
    {
        p = new char [0x7FFFFFFF];

        cout << "Succeeded!\n";
        //delete[] p;
    }
}
catch(std::bad_alloc e)
{
    cout<<"New() Failed! --> std::bad_alloc\n";
}
```

# C++ Memory Management API Refresher:

## New-Expressions은 동적 변수를 생성한다.

- 메모리 할당이 실패할 경우  
throw std::bad\_alloc -> try-catch로 잡아줌

```
Succeeded!  
Succeeded!  
Succeeded!  
newexpression(32173,0x7fffd62253c0) malloc: *** mach_vm_map(size=134217  
728) failed (error code=3)  
*** error: can't allocate region  
*** set a breakpoint in malloc error_break to debug  
New() Failed! --> std::bad_alloc
```

# C++ Memory Management API Refresher:

## New-Expressions은 동적 변수를 생성한다.

- **Non-throwing new**  
std::bad\_alloc 대신 nullptr를 반환한다.

```
while(fail!=true)
{
    p = new (std::nothrow) char [0x7FFFFFFF];

    if (p == nullptr) {
        cout << "New() Failed!\n";
        fail = true;
    }
    else {
        cout << "Succeeded!\n";
        //delete[] p;
    }
}
```

# C++ Memory Management API Refresher:

## New-Expressions은 동적 변수를 생성한다.

- **Non-throwing new**  
std::bad\_alloc 대신 nullptr를 반환한다.

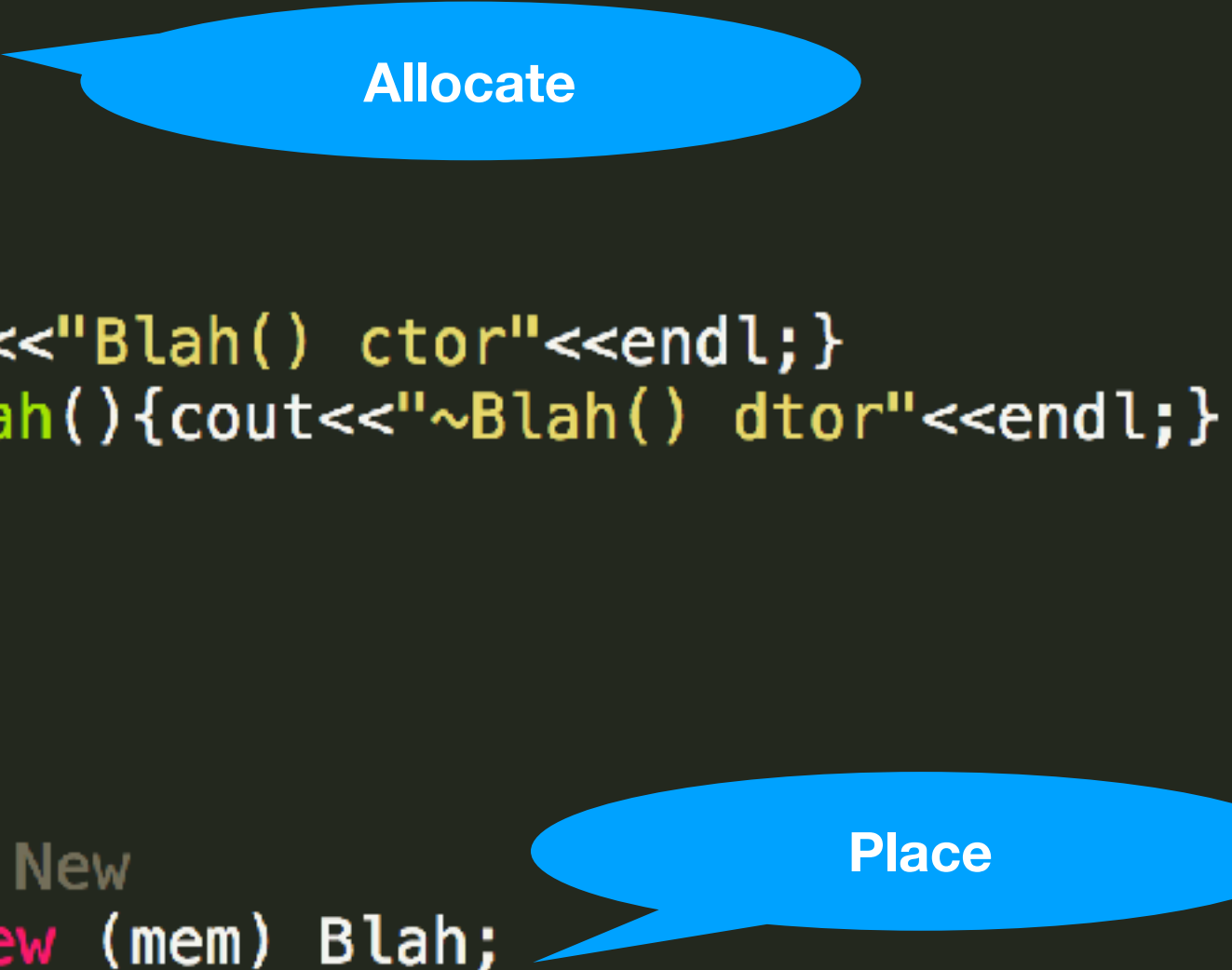
```
Succeeded!  
Succeeded!  
Succeeded!  
newexpression(31824,0x7fffd62253c0) malloc: *** mach_vm_map(size=134217  
728) failed (error code=3)  
*** error: can't allocate region  
*** set a breakpoint in malloc_error_break to debug  
New() Failed! -> p is nullptr
```

# C++ Memory Management API Refresher:

## New-Expressions은 동적 변수를 생성한다.

- Placement new는 할당(Allocate)없이 배치(Place)를 한다.

```
char mem[1000];  
  
class Blah{  
public:  
    Blah(){cout<<"Blah() ctor"<<endl;}  
    virtual ~Blah(){cout<<"~Blah() dtor"<<endl;}  
};  
  
int main()  
{  
    //Placement New  
    Blah* b = new (mem) Blah;
```



# C++ Memory Management API Refresher:

## New-Expressions은 동적 변수를 생성한다.

- Class-Specific한 operator new()는 할당(Allocate)에 대해 정교한 컨트롤을 할 수 있다.



# C++ Memory Management API Refresher:

## Delete-Expressions는 동적 변수를 제거한다

### Syntax

---

<code>::(optional)</code>	<code>delete</code>	<code>expression</code>	(1)
---------------------------	---------------------	-------------------------	-----

---

<code>::(optional)</code>	<code>delete []</code>	<code>expression</code>	(2)
---------------------------	------------------------	-------------------------	-----

---

1) Destroys one non-array object created by a `new-expression`

2) Destroys an array created by a `new[]-expression`

```
delete foo_p;
```

# C++ Memory Management API Refresher:

명시적 소멸자의 호출은 동적 변수를 소멸한다.

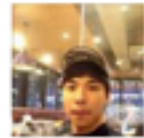
- 소멸자이름은 클래스이름 앞에 ~를 붙이면 된다.

```
class Blah{  
public:  
    Blah(){cout<<"Blah() ctor"<<endl;}  
    virtual ~Blah(){cout<<"~Blah() dtor"<<endl;}  
};
```

```
b->~Blah(); //Destructor Explicit Call
```

# C++ Memory Management API Refresher:

## 명시적 소멸자의 호출은 동적 변수를 소멸한다.



성무진

23시간 전

분명 어려운 개념은 아닌 것같은데, 왜 컴파일 에러가 뜰까요... 삼십분째 삽질 중인데, 도저히 컴파일이 안되는 이유가 이해가 안되네요ㅠ

```
5 public:
6     Foo(int a):m_a(a){
7         cout<<"Foo()"<<endl;
8     }
9
10    virtual ~Foo(){
11        cout<<"~Foo()"<<endl;
12    }
13
14    void printA(){
15        cout<<"A:"<<m_a<<endl;
16    }
```

```
it:(master) ✗ g++ -std=c++14 -o newexpression newexpressi
o:26:14: error: cannot refer to type member 'Foo' in 'Foo
oo::Foo(123);
```

```
^
o:4:7: note: member 'Foo' declared here
```

d.

# C++ Memory Management API Refresher:

명시적 소멸자의 호출은 동적 변수를 소멸한다.

- **명시적인 생성자 호출은 없다 ... 있나?**

C++ 표준에 따르면 “생성자는 이름이 없다”. 따라서 프로그램이 직접 생성자를 호출할 수 없다.

만약, Placement New Syntax를 사용하면 생성자 호출을 명시적으로 할 수는 있다.

근데!! Visual C++에서는 명시적 호출이 된다?

GCC는 더 표준적이어서 명시적 생성자 호출은 에러메시지를 띄운다.

# C++ Memory Management API Refresher:

명시적 소멸자의 호출은 동적 변수를 소멸한다.

```
class Blah{
public:
    Blah(){cout<<"Blah() ctor"<<endl;}
    virtual ~Blah(){cout<<"~Blah() dtor"<<endl;}
};

int main()
{
    //Heap Allocation
    Blah* b = (Blah*) new char[sizeof(Blah)];
    new(b) Blah;
    b->~Blah();

    cout<<endl;

    //Stack Allocation
    Blah myBlah;
    new(&myBlah) Blah;
    myBlah.~Blah();
}
```

2) 소멸자 명시적 호출

1) Placement new syntax로 생성자 명시적 호출

# High-Performance Memory Managers

## 고성능 메모리 관리자들

- 디폴트 C++ 메모리 관리자는
  - **효율적으로** 실행되어야 한다. 가장 hot하다.
  - 멀티쓰레드에서 잘 돌아가야 한다. 기본 메모리 관리자에서의 데이터구조 접근은 직렬화(Serialized)되어야 한다.
  - (리스트의 노드 같이) 같은 크기의 많은 객체를 **효율적으로** 할당해야 한다.
  - (String 같이) 다른 크기의 많은 객체를 **효율적으로** 할당해야 한다.

# High-Performance Memory Managers

## 고성능 메모리 관리자들

- 디폴트 C++ 메모리 관리자는
  - 아주 큰 자료구조(I/O 버퍼, int의 백만개의 배열)에서 작은 자료구조(단순 포인터)까지 다 할당이 가능해야 한다.
  - 효율성을 최대화 하기 위해, 포인터의 alignment boundary와 cache lines, virtual memory page는 알고 있어야 한다.
  - 런타임 성능은 시간에 따라 저하 되어서는 안된다.
  - 반환된 메모리를 효율적으로 재사용해야 한다.

# High-Performance Memory Managers

## 고성능 메모리 관리자들

- 메모리 관리자를 최적화 할수 있는 여지
  - C++ 메모리 관리자의 이러한 많은 요구들을 충족시키는 것은 아직도 학술연구와 컴파일러 Vendor에게도 도전이다.
  - 어떤 경우, 메모리 관리자는 모든 요구조건을 충족할 필요는 없다.



# High-Performance Memory Managers

## 고성능 메모리 관리자들

- 최신 메모리 관리자
  - 몇몇 독립적인 메모리 관리자 라이브러리들은 디폴트 메모리 메니저보다 상당한 성능 향상이 있다고 주장한다.

# High-Performance Memory Managers

## 고성능 메모리 관리자들

- 최신 메모리 관리자. 그러나?
  - 최신의 메모리 관리자들이 기본 malloc()보다 더 나은 성능향상을 보여주고 있음에도 불구하고, 정확한 기준치는 없다.  
Linux3.70이나 Windows7부터는 최신의 메모리 관리자를 채택하여 사용하고 있다는 소문이..
  - 더 빠른 메모리 관리자는 동적변수의 할당(Allocate)과 해제(Free)가 프로그램의 런타임에 대부분일 때만 성능에 도움을 준다.
  - 메모리 관리자의 호출 횟수를 줄이는 것이 성능 향상에 더 도움을 준다.

감사합니다