**항목 51. new 및 delete 를 작성할 때 따라야 할 기존의 관례를 잘 알아 두자.**

|  |
| --- |
| void\* operator new(std::size\_t size) throw(std::bad\_alloc){  using namespace std;  if (size == 0) { size = 1; }  while(true){  메모리 할당;  if (할당 성공) return (할당 메모리 포인터);  // 할당 실패 -> 현재 new\_handler 를 찾음  new\_handler globalHandler = set\_new\_handler(0);  set\_new\_handler(globalHandler);  if (globalHandler) (\*globalHandler)();  else throw std::bad\_alloc();  }  }  void operator delete(void\* rawMemory) throw() {  if (rawMemory == 0) return;  메모리 해제;  } |

- 반환 값이 제대로 되어 있어야 한다.

- 0바이트가 요구 되었을 때조차도 적법한 포인터를 반환해야 한다.

- 다중스레드 환경에서 사용되는 코드라면 스레드 안정 성이보장되어야 한다. 단일 스레드라면 스레드 안정성을 신경 안쓰는 것이 훨씬 빠르다.

- 무한 루프 안에서 구현을 하며 이 루프를 빠져나오는 유일한 조건은 메모리 할당에 성공하는 것이다.

- delete 작성시 nullptr에 대해서 안정성을 보장 하라.

특정 클래스 전용 할당자를 만들 때 주의해야 할 것은 그 파생 클래스에 대해서 정상동작 하지 않을 수 있다.

|  |
| --- |
| class Base {  public:  static void \*operator new(std::size\_t size) throw(std::bad\_alloc);  . . . .  };  class Derived : public Base { . . . . };  Derived \*p = new Derived; // Base::operator new 가 호출 됨. |

가장 좋은 해결 방법으로는 “틀린” 메모리 크기가 들어왔을 때는 시작 부분에서 확인한 후에 표준 operator new를 호출하는 쪽으로 살짝 비켜가게 만들어 주는 것으로 해결이 된다. Delete의 경우도 마찬가지이다.

|  |
| --- |
| class Base {  public:  static void \* operator new(std::size\_t size) throw(std::bad\_alloc);  static void operator delete(void \*rawMemory, std::size\_t size) throw();  ...  };  void \* Base::operator new(std::size\_t size) throw(std::bad\_alloc) {  if (size != sizeof(Base)) return ::operator new(size);  ...  }  void Base::operator delete(void\* rawMemory, std::size\_t size) throw() {  if (rawMemory == 0) return;  if (size != sizeof(Base)) {  ::operator delete(rawMemory);  return;  }  ...  return;  } |

**요약**

- 관례적으로, operator new 함수는 메모리 할당을 반복해서 시도하는 무한 루프를 가져야 하고, 메모리 할당 요구를 만족시킬 수 없을 때 new-handler를 호출해야 하며, 0바이트에 대한 대책도 있어야 합니다. 클래스 전용 버전은 자신이 할당하기로 예정된 크기보다 더 큰(틀린) 메모리 블록에 대한 요구도 처리해야 합니다.

- operator delete 함수는 널 포인터가 들어 왔을 때 아무 일도 하지 않아야 합니다. 클래스 전용 버전의 경우에는 예정 크기보다 더 큰 블록을 처리해야 합니다.