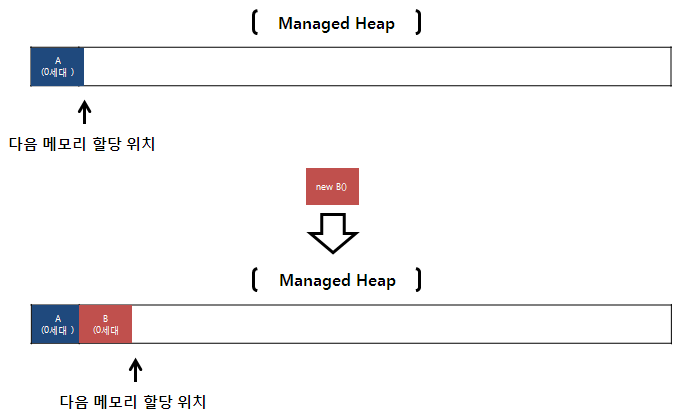
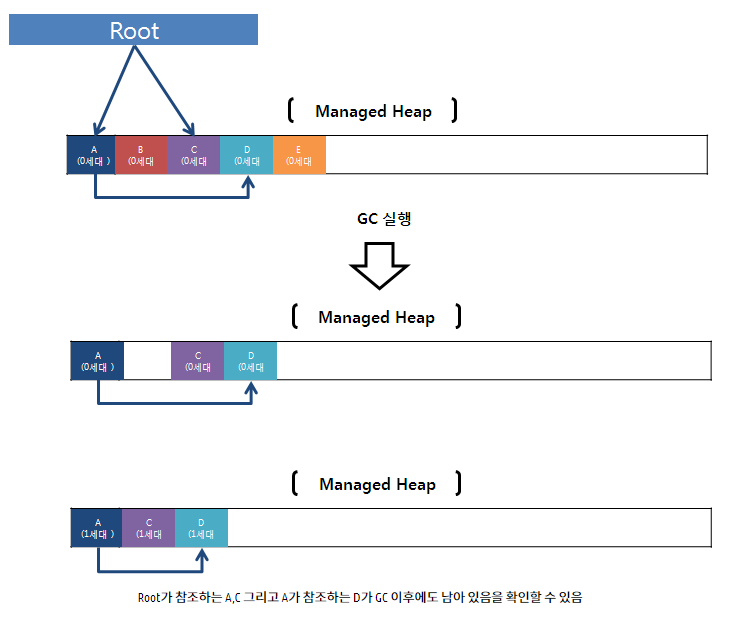
**Garbage Collection**

1. **가비지 컬렉션**- 메모리 관리 기법 중의 하나  
   - **프로그램이 동적으로 할당했던 메모리 영역 중에서 필요없게 된 영역을 해제하는 기법이다.**- 필요없게 된 영역이란, **어떤 변수도 가리키지 않게 된 영역**을 의미.   
   - 장점은 다음과 같은 버그를 줄이거나 완전히 막을 수 있다.  
   1) 유효하지 않은 포인터 접근 (이미 해제된 메모리에 접근하는 버그)  
   2) 이중 해제 (이미 해제된 메모리를 또다시 해제하는 버그)  
   3) 메모리 누수 (더 이상 필요하지 않은 메모리가 해제되지 않고 남아있는 버그)   
   - 단점  
   1) 어떤 메모리를 해제할지 결정하는 데 비용이 든다.  
   2) 쓰레기 수집이 일어나는 타이밍이나 점유 시간을 미리 예측하기 어렵다.  
   3) 할당된 메모리가 해제되는 시점을 알 수 없다.
2. **가비지 컬렉션 동작 원리  
   1) C# 메모리 할당**  
   - C#은 선형적인 Heap 메모리를 가지고 있으며, 메모리 할당을 위한 포인터 만을 갖고 있다.  
   - 메모리 할당이 필요한 경우 현재 필요한 메모리 크기만큼 증가시키고 0세대값으로 설정. 포인터는 증가된 크기만큼 이동 시킴  
   - 할당을 위한 충분한 메모리가 없을 경우, GC가 호출되고 GC가 호출됨에도 충분한 메모리가 없을 경우에는 Heap메모리가 확장된다. 대부분 Unity 플랫폼의 경우 힙의 크기를 2배로 늘림.  
   - 만일 사용가능한 메모리 공간이 부족해지는 경우 앱이 강제 종료  
     
   **2) C# 메모리 해제**  
   - GC는 시스템의 실제 메모리가 부족 하거나, Managed Heap에 할당된 개체에 사용되는 메모리가 허용되는 임계 값을 넘었을 때, GC.Collect 메서드가 호출될 때 발생  
   - GC는 힙 메모리 상에서 사용중인 객체들의 참조 그래프를 가지고 있으며, 이를 통해 사용중이지 않은 메모리를 해제하고 사용중인 메모리는 재배치를 한다.  
   \_ 참조 그래프의 Root는 Stack에 있는 로컬 변수, 정적 빌드, 전역 변수 등이 되고, 하위는 객체 간의 참조가 될 수 있다.  
     
   **3) SOH, LOH**  
   - C# 에서는 85KB를 기준으로 SOH(Small Object Heap), LOH(Large Object Heap)으로 나눔  
   - LOH는 SOH와 달리 할당과 동시에 2세대로 시작  
   - LOH는 메모리 재배치시 오버헤드가 크기 때문에 재배치를 하지 않는다. **- 세대별 가비지 컬렉션**  
   : 최근에 할당된 메모리부터 GC가 한번 불릴 때마다 살아 남은 메모리까지 0 ~ 2세대  
   : 최근에 생성될 객체일수록 메모리 해제가 발생가능성이 높아 0세대의 메모리부터 관리  
   : 0세대 GC에서 남은 메모리는 1세대로, 1세대의 GC가 발생하고 남은 1세대는 2세대로 변경  
   : 0세대의 메모리 관리 후에도 메모리가 부족해지면 1세대 -> 2세대 순으로 메모리를 관리.  
   