**Pool**

new delete, malloc free는 HeapAlloc() 시스템 함수와 1대1 대응(내부적으로 호출하나? 리눅스에서 fopen이 open을 호출하는 것처럼) 이 과정에서 유저모드에서 커널모드로 ContextSwitching 이 일어남.

1. 프로그램 속도에 영향을 미침(지속적으로 많은 양의 동적할당과 해제를 할 경우).
2. 미리 할당된 메모리 블록을 사용하는게 좋음(new delete로 많은 양의 메모리를 할당한 후 그 메모리를 갖다가 쓴다.)
3. Boost의 pool이라는 memory pool이 있음. (memory leaks 없는 것을 보장.)
4. Boost pool\_allocator는 지속적인 다수의 메모리 블록 할당에 효율적.
5. Boost fast\_pool\_allocator는 한번에 1개의 메모리 블록 할당에 효율적. 다수의 메모리 블록 할당 요청이 가능하지만 pool\_allocator만큼은 효율적이지 않음. 자료구조 std::list, std::vector를 사용하면 퍼포먼스 향상을 기대할 수 있다.

pool : 정렬된 메모리 블럭, null return, 생성자 소멸자 호출하지 않음. 할당해야할 사이즈를 모름.

object\_pool : template 클래스 메모리 블럭, null return, 생성자, 소멸자 호출 가능.

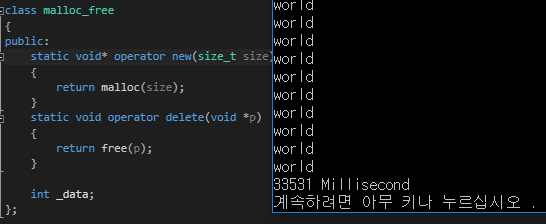
singleton\_pool : object\_pool singleton 버전인듯.. thread safely

pool\_allocator : singleton\_pool with Exceptions. stl과 같이 사용 가능.

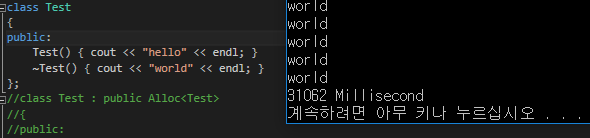
boost 빌드 방법: <http://jacking.tistory.com/1068>

new, delete, malloc free vs boost pool

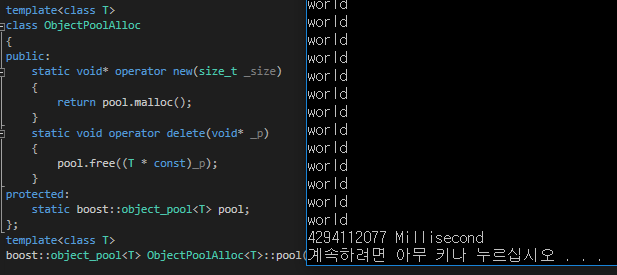
1. malloc free 10만번 할당 해제 시간: 33.531초



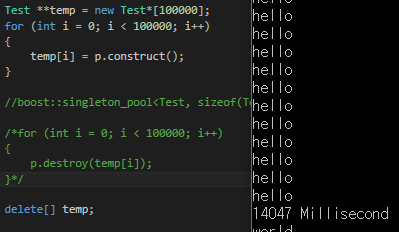
1. new delete 10만번 할당 해제 시간: 31.062초



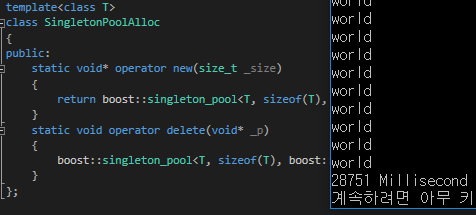
1. object\_pool 10만번 할당 해제 (ObjectPool 은 프로그램 종료시 알아서 리소스 해제 해줌.)
   1. 직접 해제 했을 때: 4294112.077초 엄청 느림



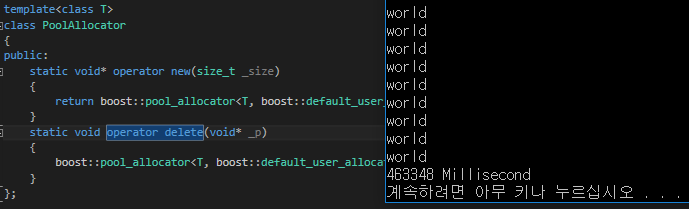
* 1. 직접 해제 안했을 때: 14.047초, 해제하는 시간 합하면 약 28초



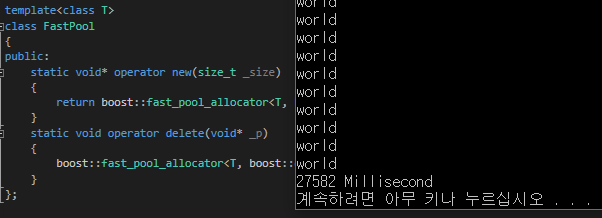
1. Singleton\_pool 10만번 할당 해제 시간: 28.751초
   1. 안탑깝게도 free해주지 않고 프로그램 종료시 purge\_memory로 해제 하면 소멸자를 호출 안함.



1. Pool\_allocator 10만번 할당 해제: 463.348초
   1. 내부적으로 singleton\_pool로 구현되어 있음. sngleton\_pool과 마찬가지로 프로그램 종료시 해제 함수를 호출 해야 함. Free로 하나씩 해제 하지 않으면 소멸자 호출 X. 속도는 빠름.
   2. Boost::singleton\_pool<boost::pool\_allocator\_tag, sizeof(int)::release\_memory();



1. Fast\_pool\_allocator



pool\_allocator는 정렬된 할당/해제를 사용, fast\_pool\_allocator는 그냥 할당/해제.. pool\_allocator는 vector와 같이 연속된 메모리를 사용할 때 적합. fast\_pool\_allocator는 list, map, set과 같은 연속된 메모리를 사용하지 않는 자료구조에 적합.

Boost pool and shared\_ptr



자원관리자 클래스

1. 프로그램 시작할 때 할당, 프로그램 종료 때 해제
2. 쓰레드 사용 여부, 소멸자 호출 여부에 따라 결정

Scene

1. 프로그램 시작될 때 할당. 종료 때 해제

GameObject

1. 자주 할당/해제 될 수 있음.
2. 자료구조에 담아 둘 것. Scene 안에 있을 예정

Component

1. 자주 할당/해제 될 수 있음.
2. 자료구조에 담아 둘 것. Scene 안에 있을 예정.

구현

1. 각 리소스풀 선언 후 static 함수로 할당, 해제 (템플릿)함수 만들기
   1. 장단점
      1. 나만의 함수 이용 가능
      2. 자료형만으로 리소스 풀 찾기 어려움. Pool의 전역 함수를 사용하는 것 고려해야 함.
2. 템플릿 클래스 정의, static operator new, delete, static pool 선언 후 상속 해서 사용.
   1. 장단점
      1. New, delete 키워드 그대로 사용
      2. 해당 pool 객체 찾을 필요 없음.

2번 방식으로 테스트 결과 new delete, malloc free, pool 방식에서(할당 해제 10만번 테스트) 속도 차이는 거의 없었음. 최근 C++ 은 new delete 기능 향상이 되어서 그런 것 같다. 그래도 boost pool 사용에 의의를 두고, boost pool은 메모리 누수 방지 염려가 없고 정렬된 할당기능도 제공 되기 때문에 boost를 사용할 예정.