1. Coding Conventions에 따라 개발되었는가?
(예: 변수 네이밍, 들여쓰기, 그리고 괄호 스타일 등)

2. 변수 정의가 적절히 주석처리가 되어 있는가?
주석은 이름이 설명되지 않은(이름이 적절하지 않은) 변수들을 위해 필요하다.
각 전역 변수는 변수의 목적과 전역으로 선언한 이유을 나타내야 한다.

3. 숫자 데이터의 단위가 명확히 명시 되어 있는가?
숫자 또는 시간 데이터의 단위를 주석화 해야한다. (예: cm, m, sec, min 등)

4. 주석화된 코드가 하나의 설명으로 되어 있는가?
주석화된 어떤 코드를 위한 하나의 설명이 있어야 한다.
(예: /* Dead Code */ , 임시로 작성되었다면 그것을 알려야 한다)

5. 코드안의 빠진 기능 또는 해결되지 않은 이슈에 대해 주석화 되어 있는가?
주석은 완벽히 구현되지 않은 모든 코드를 위해 필요하다.
주석은 무엇이 구현되어 있는지 또는 빠졌는지를 설명해야 한다.
또한 당신이 나중에 찾을 수 있도록 확실히 구별되는 표시(marker)를 사용해야 한다. (예: /* TODO:francis*/)

6. 에러에 대한 함수 리턴이 적절히 다루어 지고 있는가?
에러가 루틴의 나머지 부분이 실행 되는데 영향을 끼친다면 에러를 검출해 적절히 다루어야(적절히 변경해야) 한다.
예를 들어, 자원 할당이 실패했다면 이것은 자원을 사용하는 루틴의 나머지 부분에 영향을 미친다.
이것을 추출해 적절히 행동하도록 해주어야 한다. 어떤 경우에 "적절한 행동"은 에러를 기록하는 것(로그에 남기는것) 처럼 간단할 수 도 있다.

7. 자원과 메모리가 모든 에러 경로안에서 **free**되어 있는가?
모든 자원과 메모리 할당이 에러 경로로 부터 free되어 있다는 것을 확신해야 한다.

8. 에러 핸들링 코드가 테스트 되었는가?
에러 핸들링 코드를 실행하는 테스트 케이스를 작성하는 것은 중요하다.

9. 이미 할당된 메모리**(non-garbage collected)**가 사용되지 않고 있진 않나?
모든 할당된 메모리는 더 이상 필요 없을 때 할당해제 해 줄 필요가 있다.
메모리가 어떠한 코드 경로에서도 사용되지 않고 있다는 것을 확인하라.
특히 에러 코드 경로에서.

10. 같은 객체가 두 번 이상 해제되고 있는가?
객체가 두 번 이상 해제되는 코드가 없음을 확인하라. 에러 코드 경로를 체크하라.

11. 모든 전역 변수들은 쓰레드에 안전한가?
전역 변수가 두 개 이상의 쓰레드에 의해 접근될 수 있다면, 전역 변수를 mutex 같은 동기화 메카니즘을 사용해서 에워싸도록 코드를 고쳐야 한다.

12. 동시에 락을 얻었을 때 락은 해제 되는가?
데드락 현상을 피하기 위해 동시에 얻은 락을 해제하는 것은 중요하다. 에러 코드 경로를 확인하라.

13. 데드락 또는 락 경쟁이 발생할 가능성이 있는가?
다른 순서로 락의 집합 (mutex, semaphores 등등)을 얻을 가능성이 전혀 없다는 것을 확인하라.
예를 들면 만약 '쓰레드A'가 'Lock#1'을 얻은 후 'Lock#2'를 얻으면, '쓰레드B'는 'Lock#2' 와 'Lock#1'을 얻을 수 없어야 한다.

14. 코드가 무한 루프에서 자유로운가?
무한 루프를 일으킬 수 있는 코드 경로를 확인하라. 루프가 끝나는 상태가 별도로 주석화 되지 않는 한 확인 해봐야 할 것이다.

15. 재귀 함수는 스택 영역을 충분히 확보한 상태에서 실행되는가?
재귀 함수는 충분한 스택 영역을 확보한 상태에서 실행되어야 한다. 일반적으로 순환하는 함수 코드가 더 낫다.

16. 전체 객체들이 단지 참조가 필요해서 중복되고 있진 않은가?
이것은 객체가 단지 참조가 필요할 때 값을 담기위해 사용될 때 일어난다.
이것은 메모리에 많은 복사를 유발하는 알고리즘을 적용하게 된다.
객체 중복의 수를 최소하화한 알고리즘을 사용하는 것을 고려해야 한다.
메모리에서 이동되어질 필요가 있는 데이터는 감소시키는게 바람직하다.

17. 코드가 크기**,** 속도**,** 또는 메모리 사용에 효과적인가?
코드를 최적화 시킬 수 있는가? 예를 들어, 덩치가 큰 자료 구조를 사용한다면, 자료의 크기를 감소시킬 필요가 있을 것이다.

18. 배열에서 인덱스가 범위 밖인지를 체크 하는가?
인덱스가 범위 밖 (out-of-bound)이면 화면에 표시된다는 것을 확인해야 한다.

19. 함수가 스택에 정의된 객체에 대한 참조를 리턴하고 있는가?
스택에 정의된 객체에 대한 참조를 리턴하지 마라. 힙에 만들어진 객체를 리턴해야 한다.

20. 변수들이 사용되기 전에 초기화되어 있는가?
코드 내에서 변수를 초기화 하기 전에 사용한 곳이 없다는 것을 확인하라.
한 객체가 다수의 쓰레드에 의해 사용되었다면 그 객체를 당신이 소멸시킬 때 또 다른 쓰레드에 의해 사용되지 않는다는 것을 확인하라.
만약 한 객체가 함수 호출에 의해 생성되었다면, 그 객체를 사용하기 전에 이미 생성되었다는 것을 확인하라.

21. 코드에서 존재하는 **API**를 사용해서 새로운 기능을 만들었는가?
불필요한 일을 하지 마라. 새로운 코드는 가능한 많이 이미 존재하는 기능을 사용해야 한다.
프로젝트 내에 이미 존재하는 소스코드를 재작성 하지 마라. 하나의 함수보다 많이 복사된 코드는 쉬운 유지보수를 위해 헬퍼로 지정되어야 한다.

22. 버그가 발견되면 모두 올바르게 수정하는가?
당신이 리뷰한 코드에서 버그를 고치고 있다면, 버그가 발견된 모든 곳을 수정해야 한다.