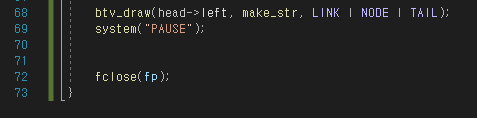
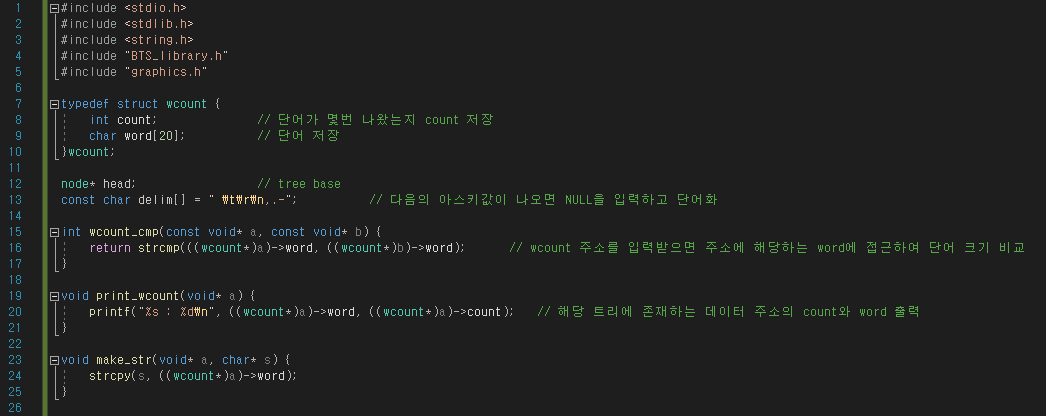
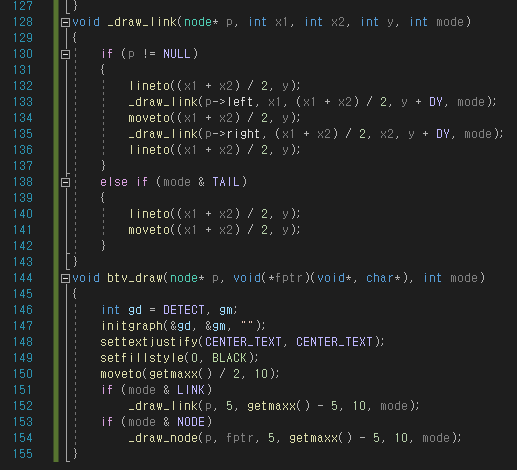
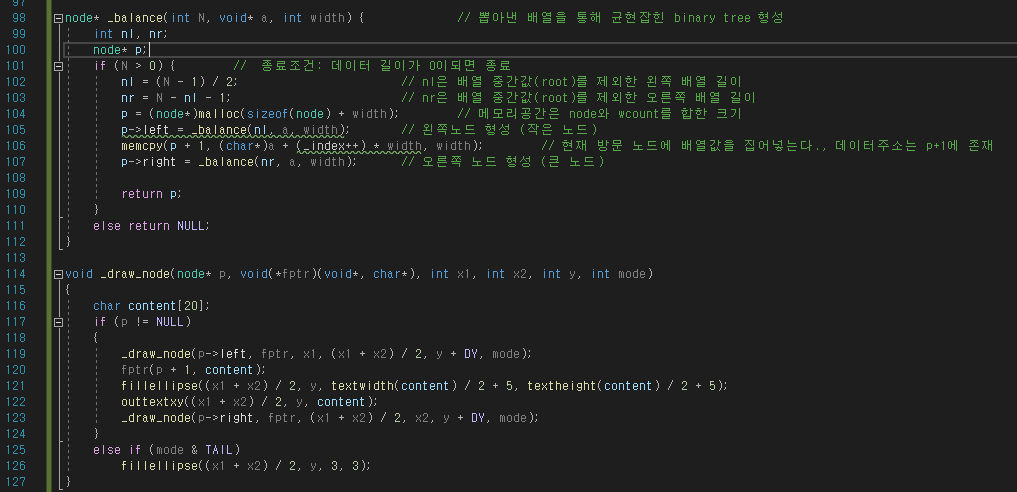
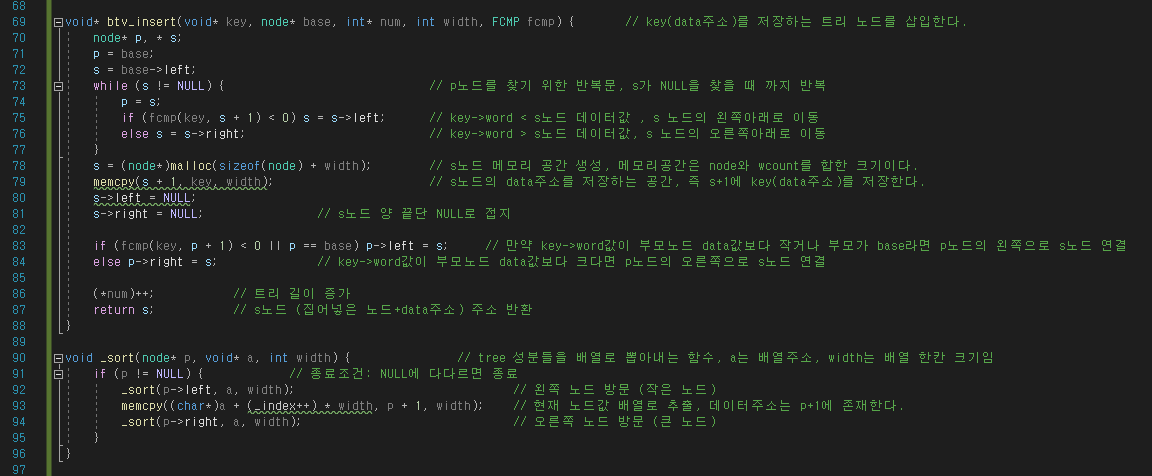
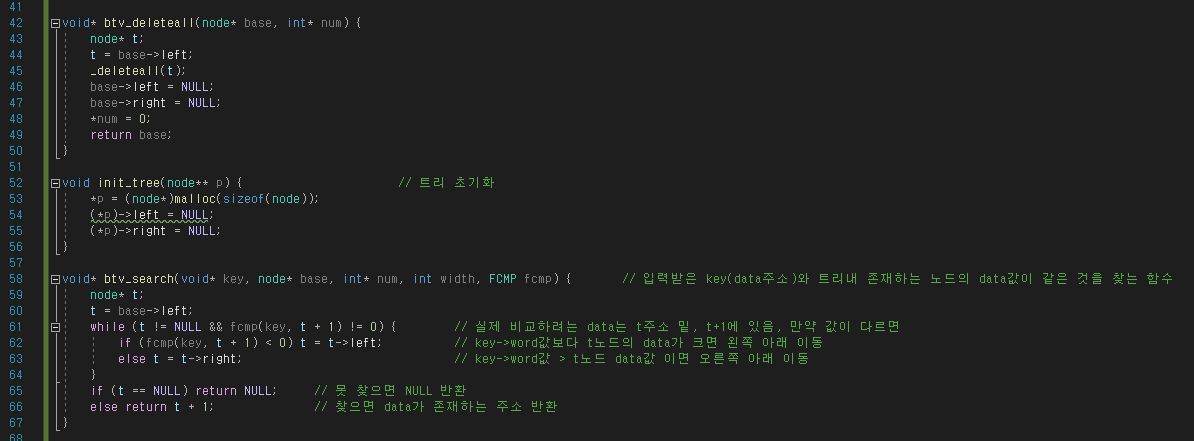
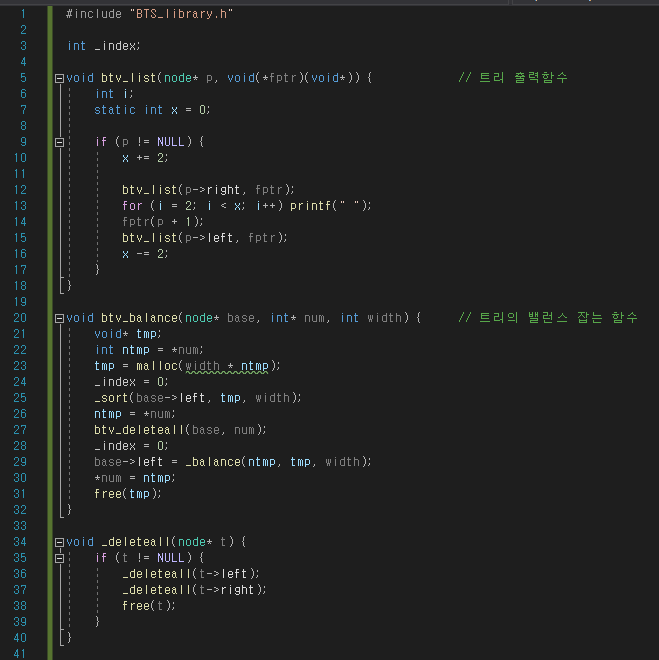
Wordcount

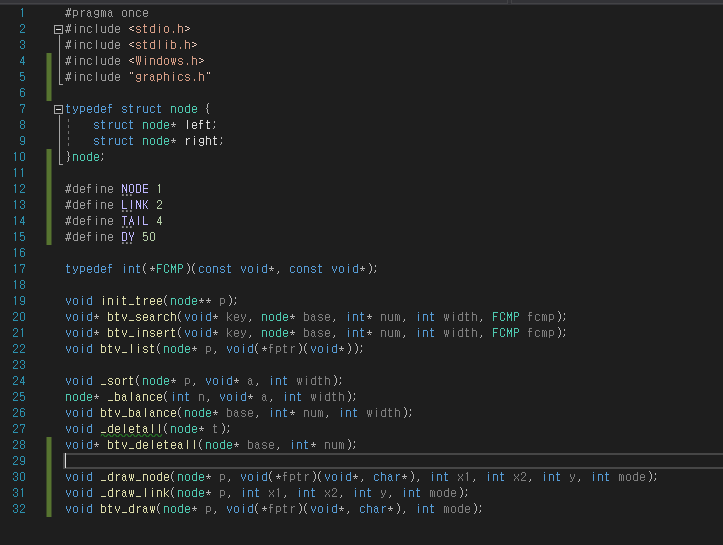
main 함수

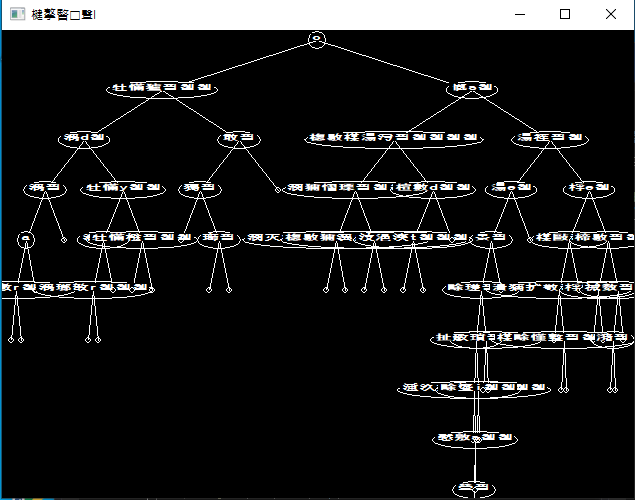
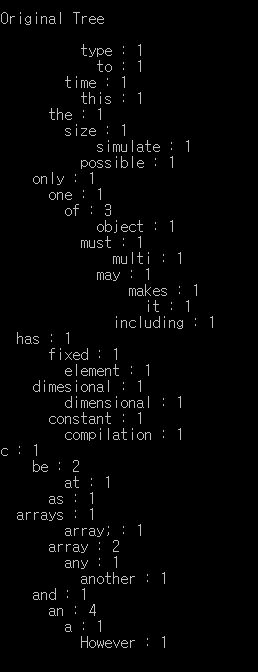


BTS\_library.c

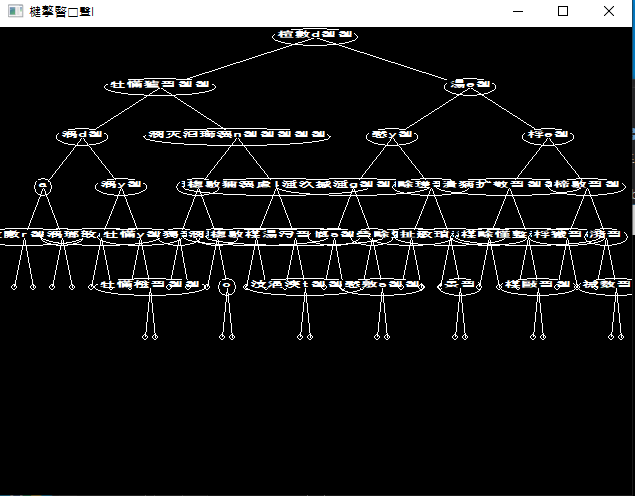
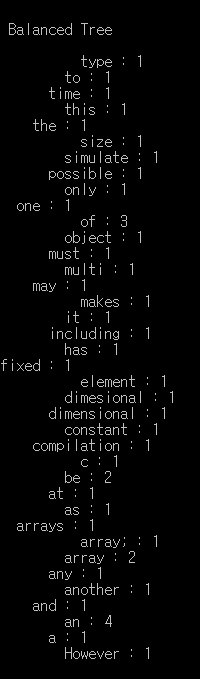


BTS\_library.h





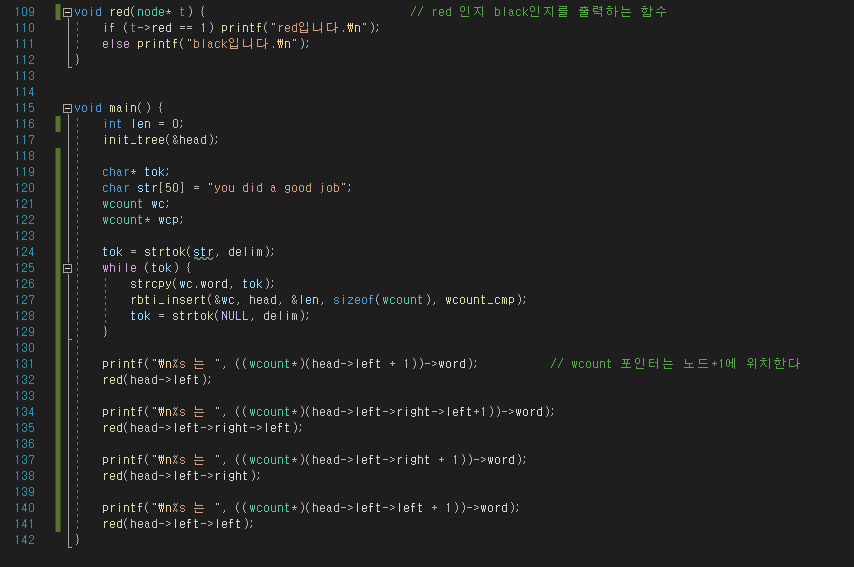
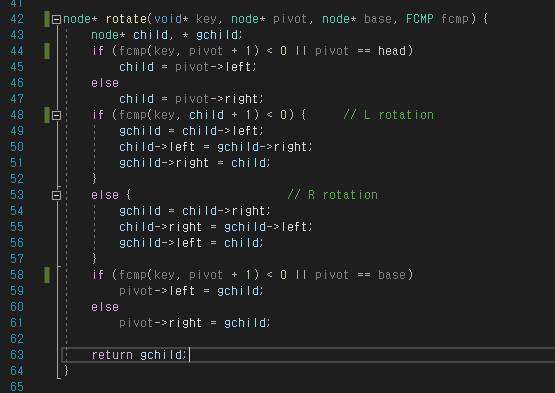
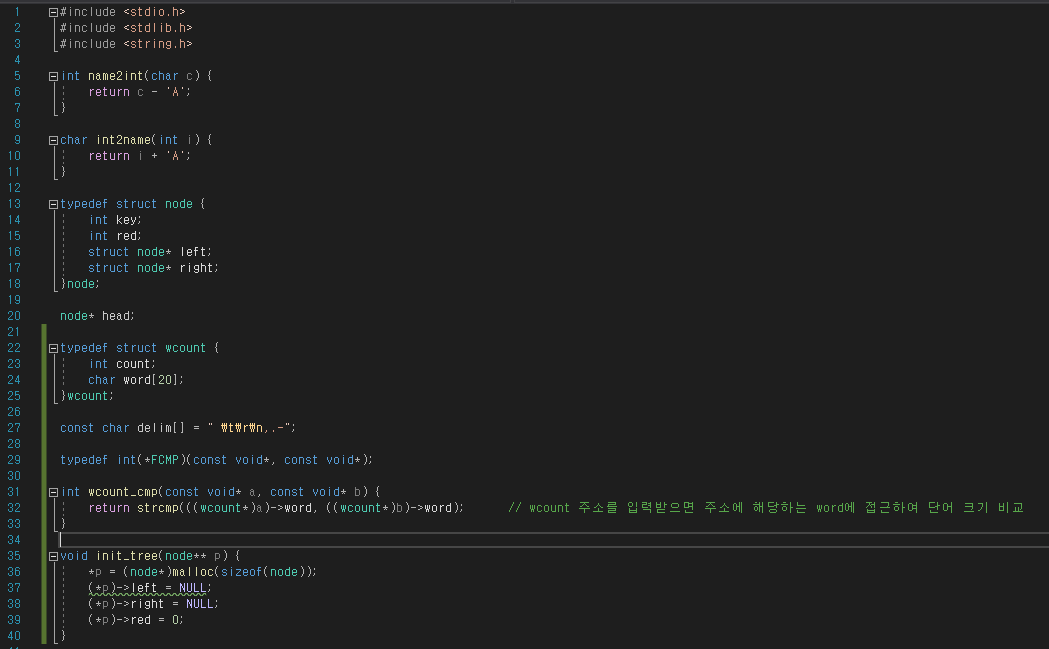
origianl tree 의 결과물

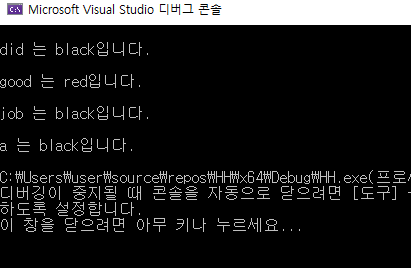


Balance 작업을 수행한 Tree의 결과물

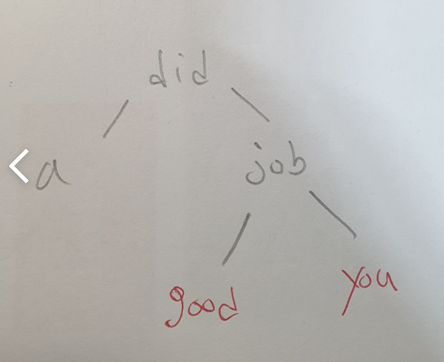
단어들을 그냥 순서대로 읽어들여 만든 tree 보다 Balance 작업을 수행한 tree의 경우 더욱 탐색속도나 동작속도가 빨라질것임을 그림을 통해 확인할 수 있다. 기존의 tree는 최하단의 자식의 경우 9번의 탐색동작을 수행해야 접근할 수 있는데 반면 Balanced tree의 경우 5번의 탐색동작을 통해 최하단의 자식에 접근할 수 있다.

RB tree 만들기 - “you did a good job”





RB tree의 ground rule을 따르면 you did a good job을 넣었을 때 다음 그림과 같이 저장된다.



스트링을 통해 해당 노드의 (정확히는 노드 밑에 존재하는 wcount포인터를 통해 word에 접근) word값과 해당 노드가 red 인지 black인지 판별하는 red( ) 함수를 통해 RB tree 가 제대로 만들어졌는지 확인할 수 있다.