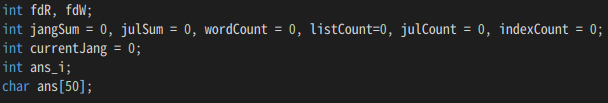
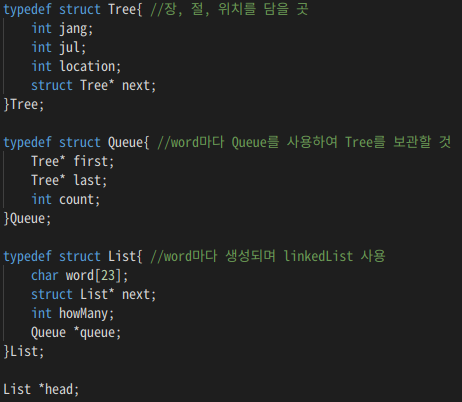


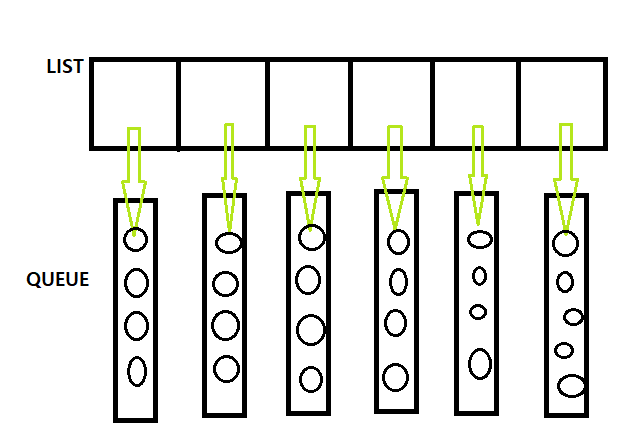
기존에 있던 헤더파일 외에는 다른 라이브러리를 참고하지 않았다. buf의 크기를 나타낼 BYTE\_SIZE의 크기를 512로 정의해주었다.



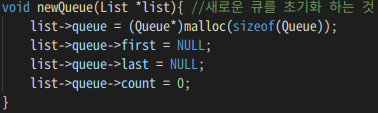
전역변수이다. 이번 프로그램에서 함수를 많이 사용하였는데, 각 함수에 필요한 변수들을 전역변수로 빼주게 되었다.



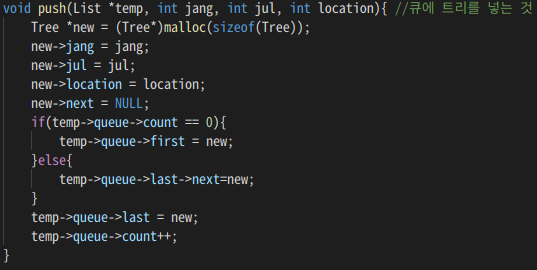
자료구조로써 사용한 구조체들이다. Tree는 Queue에 들어가며 Queue는 List에 들어간다.



그림과 같은 구조이다. Linked List 는 word마다 새로 생성되며, 각 list에는 Queue가 들어있다. 또 각 Queue에는 Tree들이 들어간다. 한 단어 마다의 인덱스를 저장하기 위한 방법으로써 고안하였다. 총 인덱스의 개수를 파악하기 위해 List에 howMany라는 멤버를 넣었고, Queue의 원소의 수를 파악하기 위해 Queue에도 count 멤버를 넣었다.

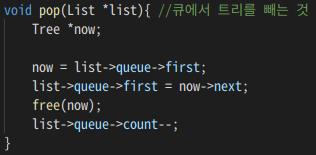


큐를 새로 초기화하는 함수이다. First와 last를 NULL로 초기화 해주고, count 또한 0으로 초기화해준다. malloc으로 메모리를 할당해준다.



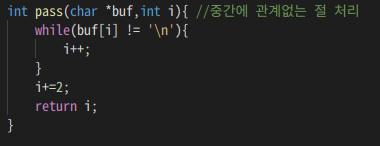
Queue에 Tree를 넣는 push함수이다. 일반적인 자료구조인 queue의 구조를 생각하였다.

Push를 할 때 마다 count가 하나 씩 증가하게 된다.

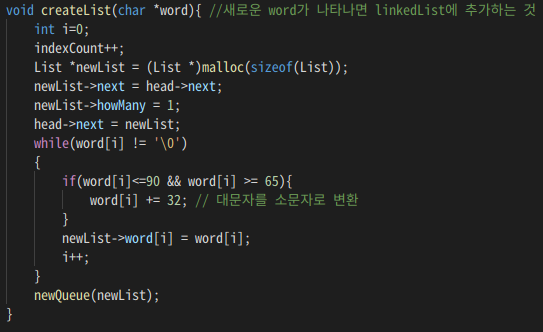


Queue의 front를 제거하는 pop함수이다. 일반적인 자료구조인 queue의 구조를 생각하였다.

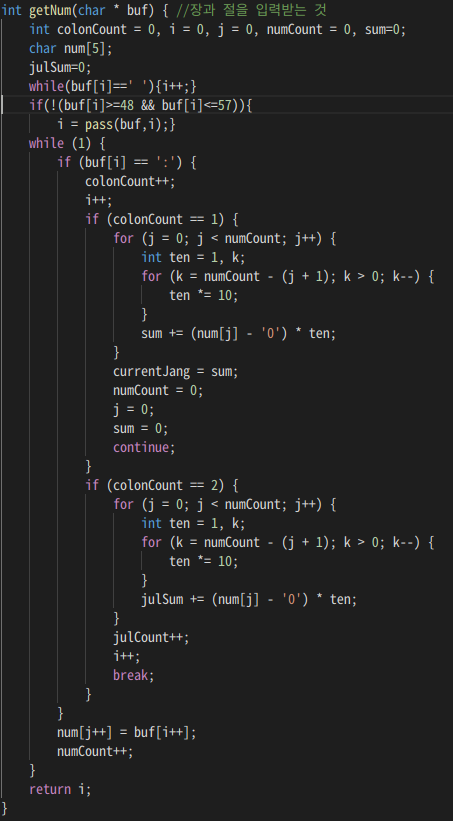
Pop을 할 때 마다 count가 하나 씩 감소하게 된다.



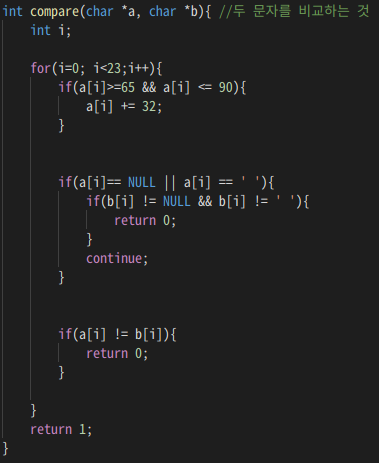
이사야서와 같은 경우에는 중간에 관계없는 내용이 나오게 된다. 이 경우를 처리하기 위하여 도입한 함수이다.



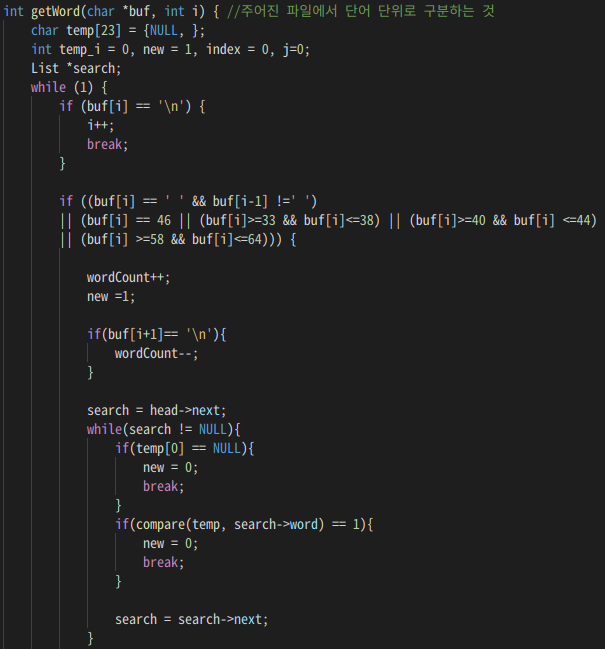
Linked list 에 새로운 word가 나왔을 때, list를 추가하는 함수이다. 대문자를 소문자로 변환하는 부분이 있는데, 이는 index 생성 시, 대소문자를 구별하지 않기 위함이다. 따라서 모두 소문자로 처리를 하였다.



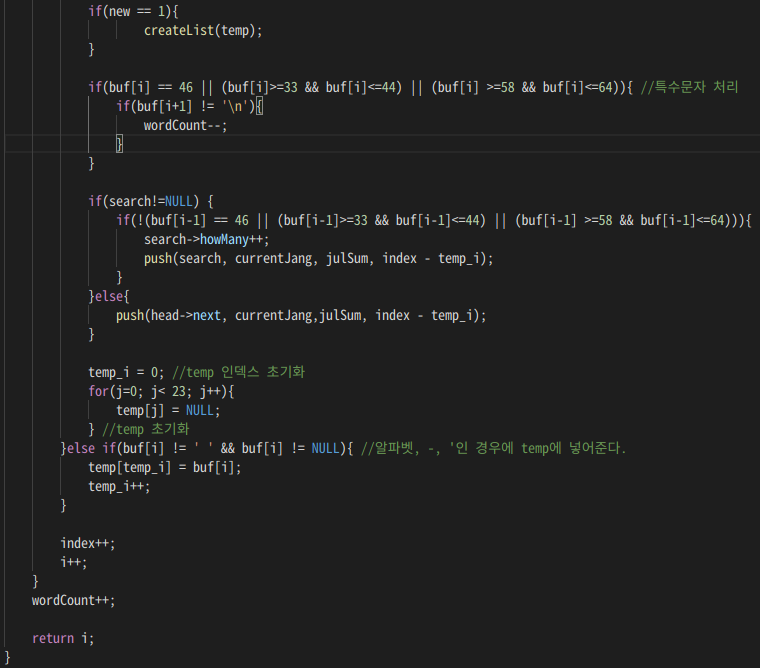
장과 절을 입력받는 함수이다. Colon의 개수에 따라 장인지 절인지를 구별하였다. Colon이 한 개가 나왔을 때에는 장에 관련된 변수에 저장을 하였다. currentJang은 최후에 write함수에 쓰일 변수이다. 현재 몇장인지를 파악하는 것이다. 절의 개수와 달리 장의 개수는 마지막 장과 같기 때문이다. 장과 절 모두 문자로 입력받았기 때문에 –‘0’ 과정이 포함되어 있다. 절까지 파악하고 나면 break로 반복문을 탈출하게 된다.



두 문자를 비교하는 compare함수이다. 아까 대소문자 변환과 마찬가지로 대문자인 경우에는 소문자로 변환하여 비교를 하였다. 이미 인덱스에 저장된 것들은 소문자이기에 따로 변환과정을 거치지 않아도 되었다.

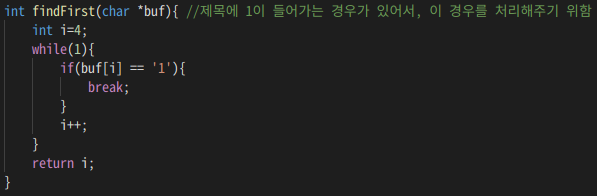


단어를 받는 함수이다. Temp[23]으로 설정한 이유는, 가장 긴 성경 단어를 찾아보았을 때에 22byte가 나왔기 때문이다. 조건문이 길게 되어있는 까닭은, -와 ‘을 제외한 문자들에 대해서도 인덱스에 같은 단어로 포함시켜주어야 하기 때문이다. Search로 linked list를 돌면서 compare를 통해 같은 단어가 있는지를 파악하였다.

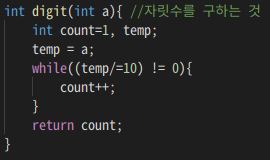


같은 단어가 없는 경우에는 new = 1로 유지되어 createList로 가게된다. 그 후 특수문자인 경우에는 문자가 중복되어 세어지는 것을 방지하기 위해 wordCount를 하나 내리게 된다.

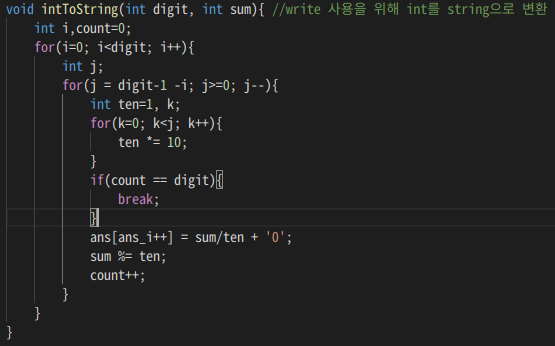
끝까지 search 하지 않고 존재하는 index를 찾았다면 그 list의 큐에 Tree를 넣게된다. 그렇지 않은 경우에는 새롭게 만들어진 list에 넣게 된다. 단어를 넣으면 temp를 초기화한다. Temp는 문자를 저장해주는 임시 문자열이다. 함수 종료시에는 현재 buf의 위치인 i를 반환한다. 어디까지가 절의 끝인지를 나타내주기 위함이다.



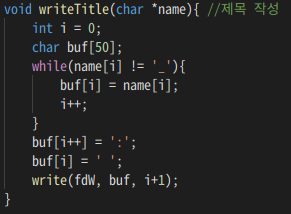
이 것은 1corinthians과 같은 경우, 맨 첫 단어가 1로 시작하여서, 이 경우를 제외해주기위해 i=4라는 적당한 위치부터 시작하였다. 그리고 1장의 위치를 찾을 때 까지 i를 증가시키는 것이다.



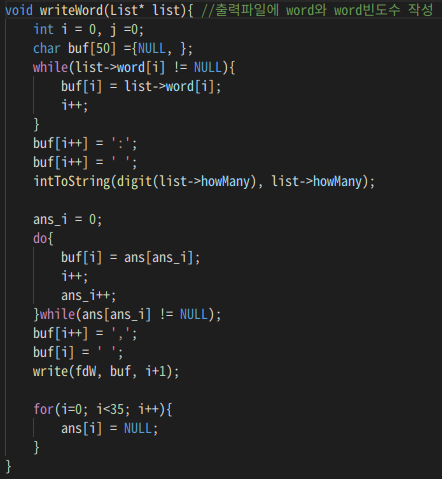
결과들은 int로 나오지만 write를 사용하려면 문자열로 반환하여야 하는데, 이를 위하여 우선 각 수의 자릿수를 구해주는 것이다.



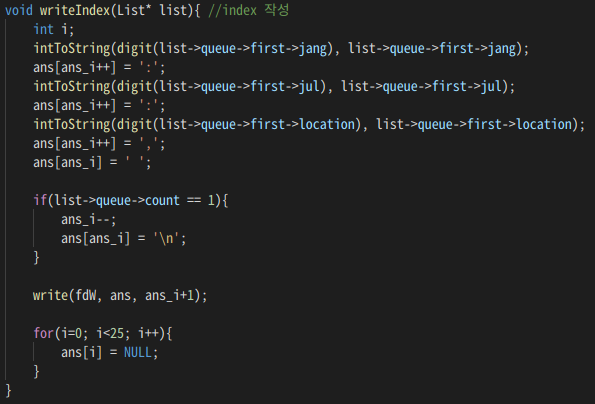
그 후, 이 함수를 통해 각각의 수를 각각의 digit, 자릿수로 나누어서 문자열에 저장해주게 된다.



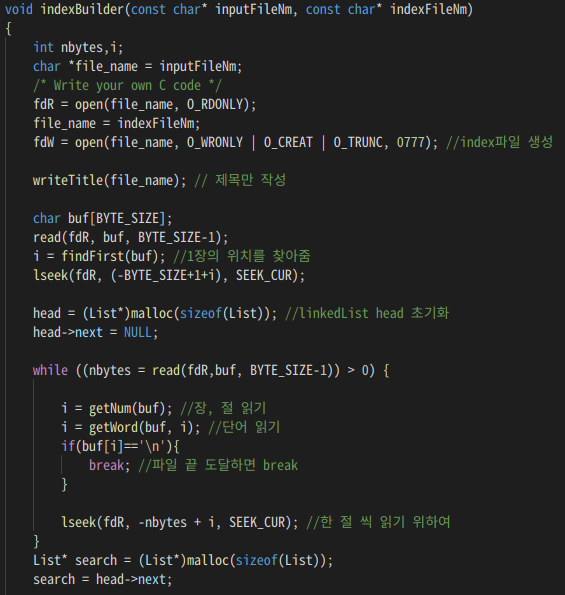
파일의 처음 부분, .txt를 제외한 부분을 출력하여야 하는데, 이를 위한 함수이다. genesis: 까지 출력하게 된다.



Index가 어떤 단어에 대한 것인 지와 그 단어가 몇 번 나왔는지를 파일에 출력해주는 함수이다.

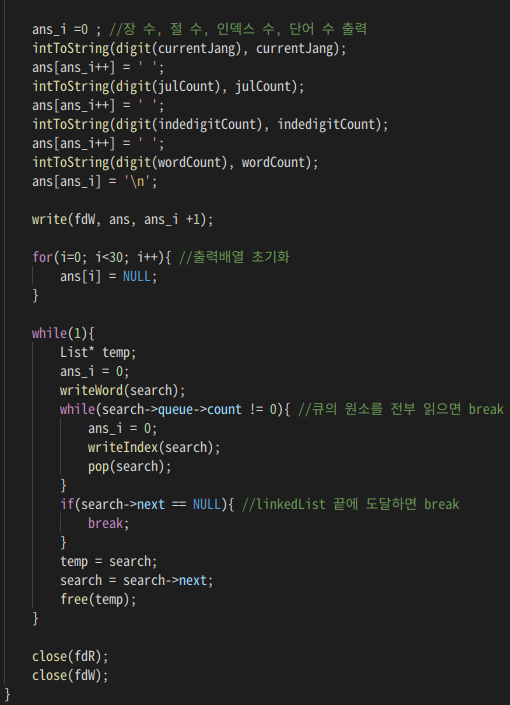


단어가 나타난 곳의 장, 절, 위치를 출력해주는 함수이다. Queue에 넣었던 tree들에 저장했던 값을 꺼내온다. 마지막의 경우에는 쉼표를 지우고 new line 값을 넣어준다.

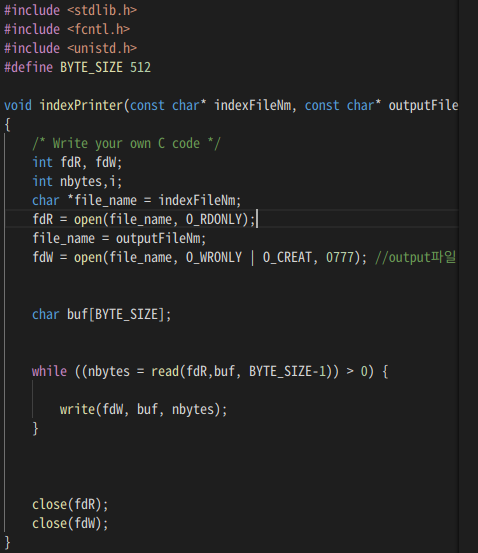


indexBuilder의 메인함수이다. 파일을 넣어주고, 파일 이름들을 설정해준 후, 제목을 적고 1장의 위치를 찾는다. Linkedlist의 head를 초기화하고 while문으로 장,절,단어들을 읽게된다.

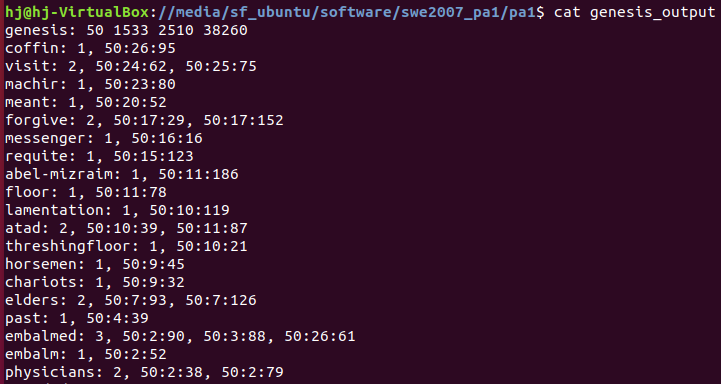
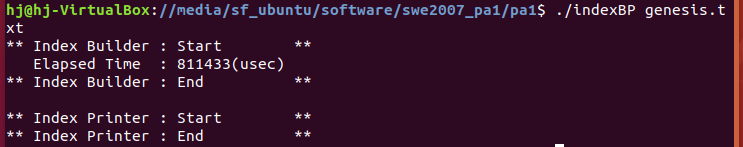
그 후에 linked list 처음부터 인덱스를 출력하기 위해 search를 할당해준다.



파일에 장수 , 절수, 인덱스 수 , 단어 수를 출력하고 난 후에, 각각의 단어와 인덱스를 출력해주게 된다. Linked List의 끝에 도달할 때 까지, queue의 원소를 전부 pop할 때 까지 반복문이 돌게 된다. 전부 구해주고 난 후에 close로 파일을 닫아준다.



printer파일에서는 index파일을 그대로 읽고 출력하는 식으로 간단하게 구현하였다. 복잡한 과정을 index에서 전부 처리하였기에 가능한 일이었다.



내 컴퓨터의 가상환경, 우분투 리눅스에서 실행한 화면이다. 정상적으로 출력이 되는 것을 확인할 수 있다.

Opinion :

결과는 전부 출력이 잘 되지만 이번 과제는 다소 불만족스럽다. 속도가 매우 느린 자료구조가 되었기 때문이다. genesis.txt에 대해 실행을 하면 예제에 적힌 96043(usec)의 거의 10배 가까이 되는 속도가 나오게 된다. LinkedList에 각각 queue를 넣는 아이디어를 낸 것에 대해서는 만족스럽다. 여러 자료구조들을 합해서 또 새로운 자료구조를 만들 수 있다는 것을 알게 되었다. 다음 번에도 비슷한 문제를 푼다면, hash map 방법을 도입 해보고싶다. word별로 일정 규칙을 주어 hash를 한 후에, 이를 key로 하여 좀 더 빠르게 접근을 하는 것이다. 할 때에는 굉장히 힘들었지만, 결과물을 보았을 때에 전율이 돋는, 실력향상을 느끼게 해준 뿌듯한 과제였다.