

# Software Project

## Project #1

담당교수 : 신영주

제출일 : 2018. 10. 05.

팀: 6

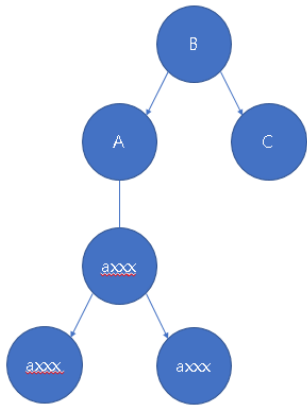
학과 : 컴퓨터정보공학부

학번 : 2015722025

이름 : 정용훈

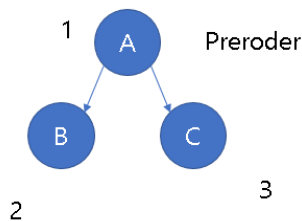
## 1. Introduction

해당 프로젝트는 java를 통하여 단어장을 만드는 것이 목표입니다. 구성으로는 외워야 할 단어를 불러와서 명령어를 통하여 외우고 있는 중인 단어로 바꾸어 주며 다른 명령어를 통해 다 외운 단어 상태인 크게 3가지로 단어의 구성이 나뉩니다. 이렇게 단어의 상태가 3세가지로 나뉘게 되며 해당 프로젝트의 가장 큰 목적은 3가지의 자료구조인 queue와 Binary Search Tree 그리고 circuit linked list를 구현하는데 있습니다. Java를 통하여 사전을 구현하는 것이며 외어야 할



단어는 queue고 외우고 있는(외우는 중) 자료구조는 Binary Search Tree로 구현합니다. 마지막으로 이전에 외운 단어장은 Circular Linked List로 구축합니다. 부가적으로 몇가지의 명령어를 통하여 단어의 자료구조끼리 옮겨가는 기능과 순회자를 통한 print기능 또한 파일을 저장할 수 있는 명령어가 있습니다. 특히 BST의 구성은 단어의 앞 알파벳을 구분해주는 알파벳 BST와 앞 알파벳이 같으며 단어의 정보를 가지고 있는 노드들이 구성 되어 있는 단어 노드 BST가 각 알파벳 노드에 포함되 있는 형태입니다. 다음 이미지를 참고하면 쉽게 이해할 수 있습니다.

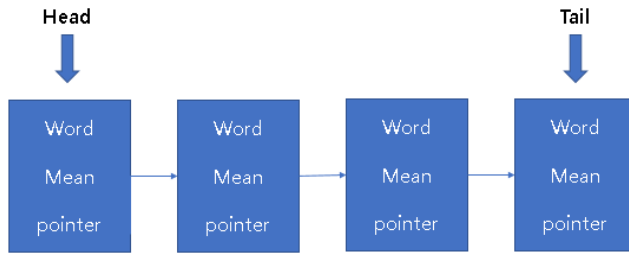
같은 출력하는데 있어 순회자는 Recursive Preorder를 사용하게 되며 재귀 함수를 사용한 전위 순회를 뜻합니다. 전위 순회는 아래와 같은 Tree의 방문 규칙을 가지고 있습니다.



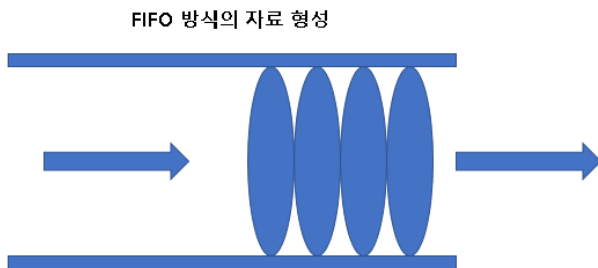
해당 오더는 먼저 기준이 되는 노드를 먼저 방문하고 왼쪽 서브 트리, 오른쪽 서브 트리로 이동하며 방문합니다. 해당 예시를 보면 출력의 순서가 A->B->C 순서로 출력되는 것을 알 수 있습니다. 이번 프로젝트에서는 사용되지 않지만 비슷한 순회자로서 Inorder와 Postorder가 있습니다. 이 ordering은 방문

의 순서가 다르며 특징으로는 Inorder를 print했을때는 해당 트리의 가장 작은 KEY값부터 순서대로 출력이 되는 규칙이 있고 Postorder의 경우 왼쪽 서브 트리 오른쪽 서브 트리 순으로 방문을 하기 때문에 java에서는 많이 쓰이지 않지만 data를 해제할 때 사용할 수 있습니다. 순회자의 또다른 응용으로는 BST를 저장하고 LOAD드를 할 때 원래 구현되어 있던 구조로 나와야 하기 때문에 SAVE를 실행할 때도 Preorder를 사용하여 저장하면 원래 있던 구조대로 불러올 수 있습니다. 위에서 언급하지 않은 자료구조로는 queue와 CC linked list가 있습니다. 각 자료구조는 아래와 같은 이미지로 자료가 형성되어 있습니다.

## Queue

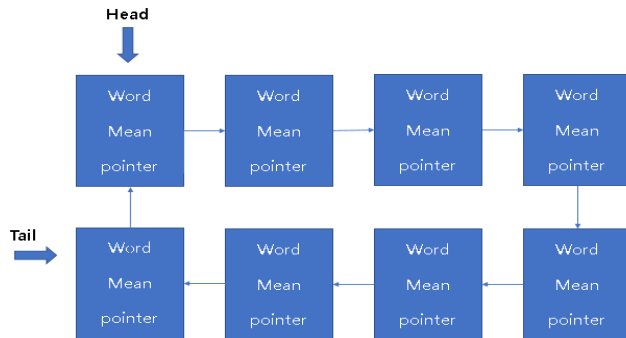


현재 구현한 프로그램에서 Queue의 구현은 이렇게 되어있습니다. Tail은 따로 정해주지 않았지만 노드들의 연결이 단순 선형으로 되어있으며 자료나 정보를 찾을 때도 앞에서 순차적으로 노드를 접근하면서 비교하며 찾게 되어있습니다. 이는 BST와의 차이점으로 BST는 정보를 찾을 때 Key의 값을 기준으로 노드의 이동이 왼쪽이나 오른쪽으로 정해지기 때문에 Search에 있어 Linked list보다 평균적으로 훨씬 빠르게 정보를 찾을 수 있습니다. 또한 위와 같은 자료구조를 Linked List라고 부르지만 이번 프로젝트에서 Queue라고 지칭하는 이유는 아래와 같은 그림을 예로 들어 설명할 수 있습니다.



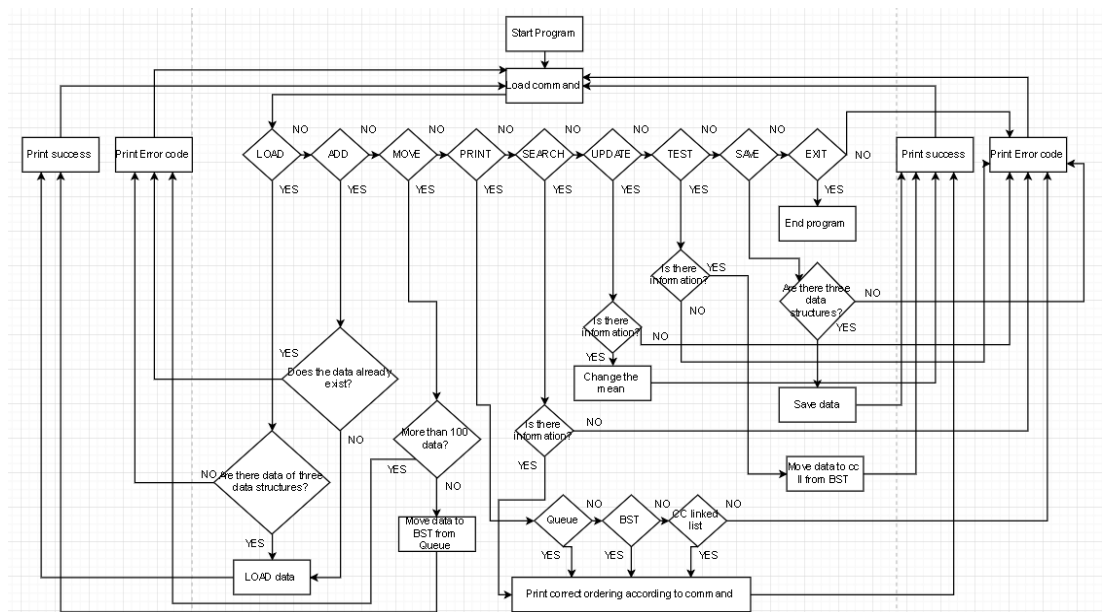
다음 그림은 Queue의 정보를 입력하고 출력하는 순서를 나타낸 이미지입니다. First In First Out 방식으로 가장 먼저 들어간 정보는 삭제가 실행되면 가장 먼저 지워지는 것을 원칙으로 합니다. 즉 구현한 Linked list에서는 삽입이 끝 노드로 이루어지기 때문에 Pop함수를 사용하면 Head부터 지워주면서 다음 노드를 Head로 정의해주면 됩니다. 비슷한 자료구조로는 Stack이 있으며 Last In First Out 방식을 사용하며 이번 프로젝트에서는 다루지 않은 내용입니다. 다음으로는 3번째 자료구조인 Circular Linked List입니다.

## CC Linked list



해당 이미지는 Circular Linked list입니다. 단순히 생각하면 Linked List의 Head와 Tail을 연결하여 Linked list가 원형을 이루도록 만들어 놓은 자료 구조입니다. 특징으로는 Head는 물론 Tail의 대한 위치정보를 꼭 기억하고 선언해 주어야합니다. 특히 명령어를 사용할 때 조건문으로 Tail의 대한 선언을 해주어야 무한 loop에 빠지지 않고 프로그램이 정상적으로 돌아가는 것을 확인 할 수 있습니다. 또한 자료를 삽입할 때 기존의 있던 Tail과 Head의 연결을 끊고 새로운 노드를 Tail로 선언해주며 다시 Head와 연결해주어야 한다는 과정이 추가됩니다. 또는 모든 자료를 먼저 넣고 나중에 연결해주는 방식을 사용해도 해당 자료구조를 구축할 수 있습니다.

## 2. Flow Chart

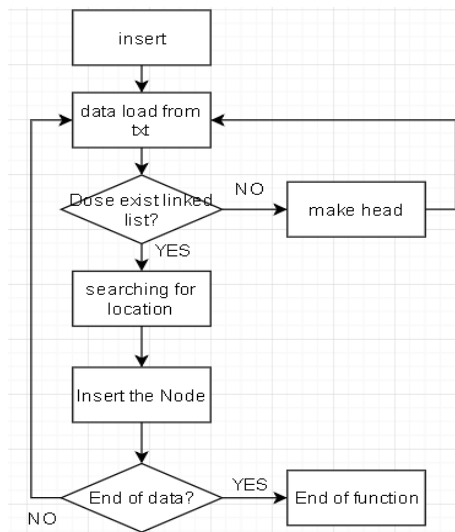


각 명령어의 대한 실행명령과 에러코드의 출력 유무를 확인하는 flow chart입니다. 함수 자체의 디테일한 알고리즘은 숨겨져 있으며 명령어를 넣었을 경우 해당 명령어를 통하여 데이터들의 상태에 따라 동작해야 하는 기능을 chart로 표현했습니다. 예를 들면 program이 시작하고 Print명령을 받았다고 했을 때 프린트해야 하는 자료구조를 판단한 후 알맞은 자료구조에 대한 Print기능 하는 함수로 이동하며 이에 맞지 않는 명령어 이거나 자료가 없으면 error코드가 나오는 동작을 chart를 통해 쉽게 이해할 수 있습니다. Chart를 통하여 전체적인 프로그램의 기능을 알 수 있습니다.

## Algorithm

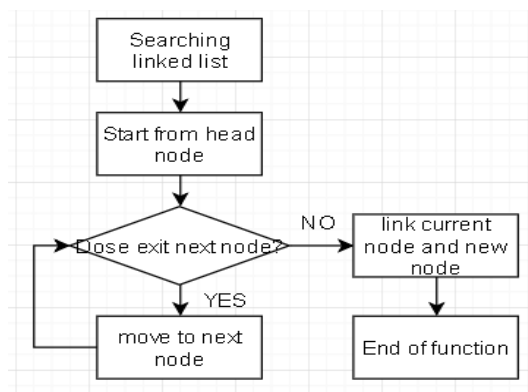
이번 프로젝트에서 다룬 자료구조는 Queue, Binary Search Tree, Circular linked list가 있습니다. 자료구조를 구축하는데 가장 중요한 함수는 insert와 delete함수 마지막으로 해당 자료구조에서 정보를 찾는 함수라고 생각합니다. 또한 insert를 하기위해선 특히 Binary Search Tree에서 노드가 들어가기 위한 위치를 찾는 것이 중요합니다. 다음 이미지는 insert함수의 가장 기본적인 구조를 chart로 나타낸 이미지입니다.

### Basic insert algorithm



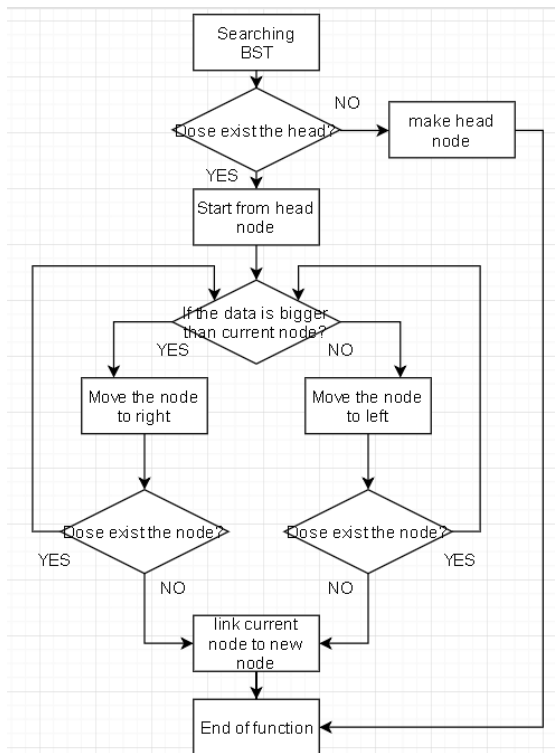
다음 그림은 모든 자료구조에서 사용되는 기본적인 insert함수입니다. 각 자료구조의 자세한 insert함수를 알아보기 위해서는 Searching for location이라는 block을 좀 더 세부적으로 알아보아야 합니다. 또한 위치를 찾는 부분의 알고리즘만 차이가 있고 모든 자료구조는 위와 같은 insert알고리즘을 기본으로 합니다. 우선 Queue와 Circular linked list의 Searching알고리즘을 찾아보면 다음과 같은 이미지로 나타낼 수 있습니다.

### Searching location queue & Circular linked list

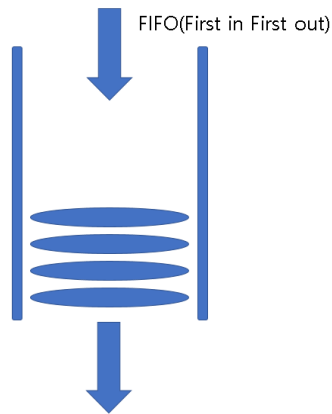


다음 이미지는 Queue와 linked list의 새로운

Node가 들어가는 알고리즘을 간단하게 이미지로 나타낸 것입니다. 위치를 찾기 시작하면 Head Node부터 pointer가 시작되며 head가 없다면 새로운 노드는 Head가 됩니다. 또한 Queue의 경우 tail node의 다음이 NULL을 가리키고있다면 새로운 노드는 NULL의 자리로 삽입됩니다. Circular linked list는 가리키는 노드가 tail Node일 때 tail과 head의 연결을 끊고 새로운 노드를 삽입 후 새로운 노드에 기존의 tail과 head를 연결시켜주면 두 자료구조를 간단하게 구축할 수 있습니다. 다음으로는 Binary Search Tree의 새로운 노드를 추가하기 위한 Searching알고리즘을 알아보겠습니다.

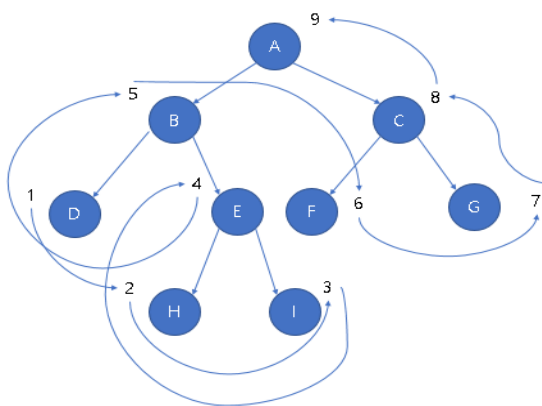


다음 이미지는 Binary Search Tree에 새로운 노드를 삽입하는 명령을 받았을 때 새로운 노드를 삽입하기 위하여 들어갈 위치를 찾아주는 알고리즘입니다. 먼저 데이터의 존재 유무를 확인 후 Head에서 시작하는 것은 동일합니다. 하지만 들어온 Key값과 원래 구축되어 있던 Node의 Key값을 비교하여 삽입될 노드의 Key값이 작으면 포인터가 왼쪽으로 이동하게 되며 더 크면 오른쪽으로 이동하게 됩니다. 그리하여 최종적으로 leaf노드의 조건에 맞는 null을 가리키게 되는데 새로운 노드는 null을 대신하여 삽입되면 최종적으로 Binary Search Tree를 구축할 수 있게 됩니다. 또한 Insert에 이어 delete도 중요한 함수입니다. Queue와 Circular linked list인 경우 단순 선형 자료구조이기 때문에 Head에서부터 순차적으로 정보를 비교하며 삭제할 Node를 찾아 원래 있던 연결을 끊고 새롭게 앞뒤 노드를 다시 연결시켜주면 됩니다. 또한 모든 데이터를 지운다고 했을 때 Queue의 경우는 아래 그림과 같은 FIFO방식의 자료 입출력을 사용하기 때문에 다음 설명처럼 삭제와 삽입을 실행해야 합니다.



다음 그림은 Queue의 대한 FIFO방식의 삽입과 삭제를 간단하게 나타낸 사진입니다. 그림 처럼 먼저 들어간 자료는 가장 먼저 나오는 방식을 사용하므로 자료구조 Queue를 삭제 하게 되면 가장 먼저 들어왔던 자료, 즉 Head부터 자료를 순차적으로 삭제해 주어야 해당 자료구조의 목적에 맞게 실행이 되는 것을 볼 수 있습니다. 다음으로 Binary Search Tree의 Delete입니다. Key값을 비교여 특정 Node 하나만 지운다고 가정할 때 나눠야 줘야 할 case가 여러 개가 있습니다. 삭제 지시를 받은 Node가 왼쪽 자식 노드가 없는 경우, 왼쪽 child는 존재하고 오른쪽은 존재하지 않는 경우, 왼쪽은 가지고 오른쪽은 가지고 있지 않은 경우, 두 child를 모두 가지고 있는 경우 4가지로 나눠서 Delete를 실행하게 됩니다. 또한 전체 메모리를 해제한다고 하였을 때 순회를 통하여 간단하게 Binary Search Tree를 모두 해제할 수 있습니다. Post order를 사용한 방법입니다. 방문 순서가 왼쪽 서브 트리 -> 오른쪽 서브 트리 -> 가운데 노드 이기 때문에 모든 데이터를 지운 후 최종적으로 root를 지울 수 있는 구조라서 모든 데이터를 깔끔하게 지울 수 있습니다. Post order의 구체적인 순회는 아래와 같습니다.

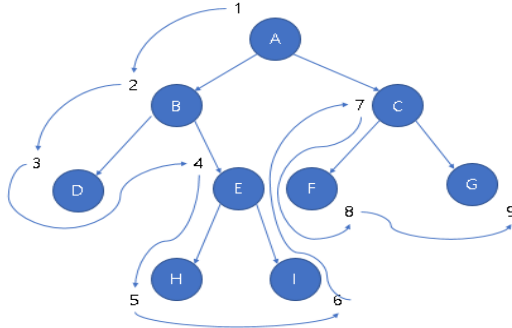
## Postorder



그림에서 보는 것과 같이 아래의 노드부터 마지막으로 root를 순회하기 때문에 EXIT함수를 실행하기전에 메모리 해제를 할 때 많이 쓰입니다. 하지만 이번 프로젝트에서는 java환경에서 구현을 하기 때문에 쓰인 알고리즘은 아닙니다.

다음으로는 Print를 위한 Preoder의 순회 순서를 소개하겠습니다.

## Preorder

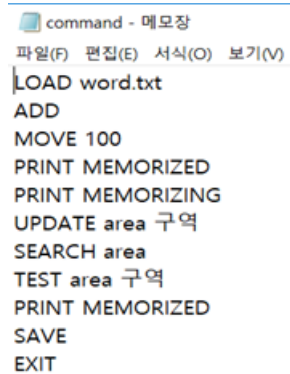


다음 그림은 Print와 Save에 사용하게 될 Preorder입니다. Print에서는 단순히 출력을 해주는 것이므로 의문을 갖지 않지만 Save에 사용하게 된다면 의문을 가질 수 있습니다. 해당 순회자를 Save에 사용하는 이유는 LOAD를 통하여 자료를 가지고 올 때 원래 구축되어 있던 Binary Search Tree로 가져와야 하기 때문에 자료를 원래 구축되어 있던 것처럼 구축하기 위하여 Preorder를 사용하여 다시 정보가 들어오게 되어도 원래대로 구축할 수 있도록 도와줍니다. 순회순서가 삽입이 되는 순서이기 때문에 가능한 알고리즘입니다. 순회자로는 Inorder와 Level order가 더 있지만 이번 프로젝트 구축에서는 사용하지 않고 있습니다. 간단하게 설명하면 Inorder는 왼쪽 서브 트리 -> 가운데 노드 -> 오른쪽 서브 트리를 순서로 순회하며 Level order는 level대로 왼쪽에서 오른쪽으로 순회가 진행됩니다. 또한 각 ordering들은 재귀함수를 사용한 방법과 반복문을 사용하여 구현한 함수가 따로 나눌 수 있습니다.



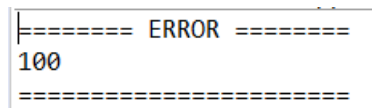
### 3. Result Screen

다음은 결과 화면에 대한 설명입니다. 각각의 명령어를 하나씩 수행하였으며 이번 프로젝트에서는 LOAD에 관련된 명령어도 수행해야 하므로 같은 명령어로 두 번 실행하여 결과 화면을 도출했습니다.



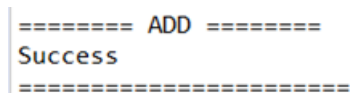
```
command - 메모장
파일(F) 편집(E) 서식(O) 보기(V)
LOAD word.txt
ADD
MOVE 100
PRINT MEMORIZED
PRINT MEMORIZING
UPDATE area 구역
SEARCH area
TEST area 구역
PRINT MEMORIZED
SAVE
EXIT
```

해당 그림은 command입니다. 수행하게 될 명령어를 나열한 것이며 프로그램은 command.txt 파일을 통해서만 명령을 받고 프로그램을 수행합니다.



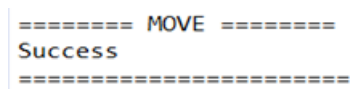
```
===== ERROR =====
100
=====
```

다음은 LOAD 명령에 대한 수행 에러입니다. 이유는 프로그램이 처음 동작하였으며 아직 구축된 자료가 없기 때문에 LOAD해야 할 정보가 없어서 Error code를 출력하고 수행을 종료합니다.



```
===== ADD =====
Success
=====
```

다음으로는 ADD입니다. 처음 프로그램을 동작할 때 수행되는 명령어이며 새로운 단어의 정보들을 불러오는 명령어입니다. Queue자료 구조를 이용하여 구축하며 데이터가 이미 있으면 해당 명령어는 수행되지 않습니다. 이미지는 수행 후 정상적으로 동작하여 성공한 모습을 볼 수 있습니다.



```
===== MOVE =====
Success
=====
```

다음은 MOVE명령입니다. Queue에 있던 자료들을 받은 인자, 즉 parameter의 수만큼 Binary Search Tree로 만들어주는 명령을 뜻합니다. 기존에 Queue의 연결은 끊어주며 따로 자료구조를 하나 더 만듭니다. 인자를 100개 이하로 받았기 때문에 자료 구축에 성공한 모습을 볼 수 있습니다.

```

===== ERROR =====
700
=====

```

해당 명령어는 Print와 관련된 명령어입니다. MEMORIZED는 TEST명령어를 통해 Binary Search Tree에서 Circular linked list로 넘어간 단어들을 Print하는 명령어입니다. 해당 명령어 전에 TEST된 단어가 없기 때문에 해당 자료구조는 NULL인 상태이며 정보가 없기 때문에 Error code를 출력하고 명령어 수행에 실패한 모습입니다.

```

===== PRINT =====
problem 문제
place 장소
person 사람
part 부분
paper 종이
piece 조각
pet 애완동물
plan 계획
plant 식물
poor 가난한
poem 시
hand 손
habit 습관
history 역사
hot 뜨거운
different 다른
bird 새
back 뒤
bottle 병
activity 활동
autumn 가을
advertisement 광고
always 언제나
area 지역
art 미술
clothes 옷
clean 깨끗한
course 강좌
cook 요리사
culture 문화
fire 불
face 얼굴
famous 유명한
fast 빨리
fun 재미
fly 날다
forest 숲
earth 지구
each 각각
early 일찍
example 예
enjoy 즐기다
easy 쉬운
end 끝
group 집단
gene 유전자
life 인생
letter 편지
learn 배우다
leaf 잎
lot 많이
listen 듣다
light 빛
line 선
job 직업
just 단지
important 중요한
information 정보
island 섬
name 이름
newspaper 신문
nature 자연
neighbor 이웃
make 만들다
machine 기계
movie 영화
mind 마음
office 사무실
often 자주
trip 여행
restaurant 식당
remember 기억하다
rock 바위
story 이야기
same 같은
spring 봄
space 공간
science 과학
sound 소리
soldier 군인
send 보내다
sometimes 때때로
special 특별한
start 시작
store 가게
state 상태
summer 여름
street 거리
subject 주제
vacation 휴가
volunteer 자원봉사자
visit 방문
village 마을
vegetable 채소
work 일
winter 겨울
watch 시계
war 전쟁
win 이기다
wear 입다
=====

```

Print MEMORIZING에 대한 명령어입니다. MOVE를 통하여 Binary Search Tree를 형성한 자료구조에 대한 Print입니다. 출력된 오더로는 Preorder를 사용하여 출력합니다. 명령어 이전에 TEST명령어가 실행되지 않았기 때문에 MOVE의 인자로 받은 100개가 그대로 들어 있는 상태입니다.

```

===== UPDATE =====
area 지역 -> 구역
=====

```

UPDATE명령어입니다. 해당 단어를 찾아 뜻을 바꿔주는 명령어입니다. 모든 자료구에서 검색을 실행하게 되며 일치하는 자료가 없으면 Error를 출력하게 됩니다. 이번 경우는 3개의 어떤 자료 중 하나에서 단어를 찾게 되고 뜻을 성공적으로 바꾸게 되어 명령에 성공한 모습입니다.

```
===== SEARCH =====  
area 구역  
=====
```

Search명령입니다. 앞서 설명한 UPDATE와 유사한 기능을 가지고 있습니다. 다만 검색된 단어만 보여줄 뿐 그에 대한 정보 변환이나 다른 실행기능은 가지고 있지 않습니다. 검색에 성공하여 출력된 모습입니다.

```
===== TEST =====  
Pass  
=====
```

TEST명령입니다. Binary Search Tree에서 TEST를 통하여 이미 외운 단어로 처리해주는 명령어입니다. 결과적으로는 Circular linked list를 형성하게 되며 Pass의 뜻은 명령에 성공했다는 것을 의미합니다.

```
===== PRINT =====  
area 구역  
=====
```

TEST명령 이후에 Print에 성공한 모습입니다. 앞서 말한 것과 같이 Circular linked list의 정보들을 출력해준 모습이며 TEST를 한번 거쳤기 때문에 하나의 정보가 들어간 것을 확인할 수 있습니다.

```
===== SAVE =====  
Success  
=====
```

Save 명령입니다. 앞선 명령어를 통해 모든 자료구조가 구축되었기 때문에 SAVE에 성공한 모습입니다. 결과적으로 각각의 자료구조 정보가 담긴 txt파일이 생성되며 이는 다음 프로그램을 실행할 때 LOAD에 쓰이는 재료가 됩니다.

```
===== EXIT =====  
Success  
=====
```

EXIT함수입니다. 프로그램의 동작을 끝내는 함수이며 성공적으로 프로그램을 끝낸 모습을 볼 수 있습니다.

**다음 실행화면은 같은 명령어로 프로그램을 한번 더 실행한 결과입니다.**

```
===== LOAD =====  
Success  
=====
```

전에 실행하면서 자료를 구축하여 txt파일을 만들었기 때문에 LOAD를 통한 명령을 성공하여 자료를 불러온 모습입니다.

```
===== ERROR =====  
200  
=====
```

이미 자료가 구성 되어있기 때문에 ADD의 명령이 실패한 모습입니다.

```
===== ERROR =====  
300  
=====
```

처음 프로그램을 돌리면서 100개의 단어를 이동시키고 TEST를 통하여 하나를 없애며 99개의 자료를 저장하고있었습니다. 다시 MOVE를 통해 100개를 넣으려 했으나 Binary Search Tree는 100개의 단어가 한계이기 때문에 1개만 더 추가로 받고 동작은 멈추며 Error코드를 출력하게 됩니다.

```
===== PRINT =====  
area 구역  
=====
```

LOAD를 통해 자료를 불러왔으므로 Circular linked list에 있는 정보를 출력한 모습입니다.

```

===== PRINT =====
problem 문제
place 장소
person 사람
part 부분
paper 종이
piece 조각
pet 애완동물
plan 계획
plant 식물
poor 가난한
poem 시
hand 손
habit 습관
history 역사
hot 뜨거운
different 다른
bird 새
back 뒤
bottle 병
activity 활동
autumn 가을
advertisement 광고
always 언제나
art 미술
clothes 옷
clean 깨끗한
course 강좌
cook 요리사
culture 문화
fire 불
face 얼굴
famous 유명한
fact 사실
fast 빨리
fun 재미
fly 날다
forest 숲
earth 지구

each 각각
early 일찍
example 예
enjoy 즐기다
easy 쉬운
end 끝
group 집단
gene 유전자
life 인생
letter 편지
learn 배우다
leaf 잎
lot 많이
listen 듣다
light 빛
line 선
job 직업
just 단지
important 중요한
information 정보
island 섬
name 이름
newspaper 신문
nature 자연
neighbor 이웃
make 만들다
machine 기계
movie 영화
mind 마음
office 사무실
often 자주
trip 여행
restaurant 식당
remember 기억하다
rock 바위
story 이야기
same 같은
spring 봄
space 공간

science 과학
sound 소리
soldier 군인
send 보내다
sometimes 때때로
special 특별한
start 시작
store 가게
state 상태
summer 여름
street 거리
subject 주제
vacation 휴가
volunteer 자원봉사자
visit 방문
village 마을
vegetable 채소
work 일
winter 겨울
watch 시계
war 전쟁
win 이기다
wear 입다
=====

```

Binary Search Tree에 구축되어 있는 단어들을 Preorder를 통하여 출력한 모습입니다.

```

===== UPDATE =====
area 구역 -> 구역
=====

```

Area를 찾아 UPDATE한 모습입니다. 일단 area라는 key값이 있기 때문에 같은 뜻으로 변경하더라도 Error가 나오지 않는 모습입니다.

```

===== SEARCH =====
area 구역
=====

```

앞서 말한 SEARCH와 동일한 내용입니다.

```

===== ERROR =====
500
=====

```

TEST명령입니다. 이미 기억이 된 단어로 판단하여 linked list로 넘어가 있기 때문에 Binary Search Tree에는 정보가 없기 때문에 정보를 찾지못해 Error가 출력된 모습을 볼 수 있습니다.

```
===== PRINT =====  
area 구역  
=====
```

TEST가 된 단어가 추가적으로 없기 때문에 area만 나온 모습입니다.

```
===== SAVE =====  
Success  
=====
```

세개의 자료구조가 동일하게 존재하므로 해당 자료구조를 저장합니다.

```
===== EXIT =====  
Success  
=====
```

프로그램의 종료를 뜻합니다.

#### 4. 본인이 구현한 부분 설명

이번 프로젝트를 구현하면 도입부를 담당하게 되었습니다. 파일 입출력을 다루게 되었고 자료구조 두개를 구축하게 되었습니다. 우선 파일 입출력에 관련된 설명입니다. 평소 프로그래밍 언어로는 C언어와 C++만 다뤄보았기 때문에 아예 다른 함수들을 접하게 되었습니다. C++와는 다르게 파일을 읽는 함수가 따로 있고 그에 대한 정보를 버퍼를 통해 출력하는 방식으로 되어있어서 강의에서는 배운 내용이 아니었기 때문에 해당 함수와 사용법을 찾는데 시간이 좀 걸린 편이었습니다. 또한 명령어의 인자를 나눠 주어야 하기 때문에 C++에서 사용한 Token을 대신한 함수를 찾게 되었고 관련 여러가지 함수를 사용하고 parameter를 구분하여 받을 수 있게 했습니다. 또한 C++에는 없는 String에 대한 다양한 여러가지 함수가 많았습니다. 예를 들면 String의 문자들이 대문자 일 때 Low case 함수를 사용하여 String안에있는 문자들을 모두 소문자로 바꿀 수 있는 함수도 알게 되었습니다. 특히 이 함수를 통하여 명령어 parameter의 대소문자를 구분하지 않는 조건을 만족시켰습니다. 그리고 equals, compare to와 같은 문자열들을 비교하는 함수도 다양하게 존재하였습니다. 다음으로는 Queue의 형성입니다. 즉 함수에서 ADD부분을 담당하게 되었습니다. 앞서 설명한 파일 입출력을 통하여 txt파일에서 정보를 가져와 Queue자료구조를 구축하게 되었습니다. 1학년때부터 Linked list에 대한 구현을 많이 했기 때문에 delete 함수만 신경 써서 해준다면 Queue를 만드는 것은 크게 어렵지 않게 하였습니다. 또한 Queue에 대한 Print를 구현하게 되었으며 방법으로는 단순히 Head의 포인터부터 current의 상태가 다음 노드를 NULL을 가리키게 될 때까지 함수를 반복시키며 Print하게 하였습니다. 평소 많이 구현해본 함수로 어렵지 않게 구현할 수 있었습니다. 다음으로 Binary Search Tree를 구축한 것입니다. 즉, 함수로 따지면 MOVE 함수를 구현한 것입니다. 구현 방법으로는 인자로 삽입할 노드의 개수를 인자로 받게 되며 최대 100개의 Node를 받아야 하므로 count를 따로 BST에 저장하여 개수를 판단 해 줍니다. 사용한 알고리즘으로는 1문항인 프로그램 소개에 자세하게 써 놓았습니다. 함수의 세부적인 기능과 세부적인 에러에 대한 것은 다음 팀원들이 구현하였으며 전체적인 에러와 마무리는 제출 전 모여서 토의를 통하여 마무리하며 프로그램을 다듬었습니다.

## 5. 팀원 참여도 평가

- 조원 이름: 홍진혁
  - i. 참여도: 33.3
  - ii. 평가의 근거: 프로그램의 기본적인 동작을 담당하게 되었으며 마지막으로 세번째 자료구조인 Circular linked list 구축했습니다. UPDATE, SEARCH, TEST, SAVE를 중점으로 구현하였으며 node의 이동이나 삭제에 관련된 함수를 구현했습니다.
- 조원 이름: 오윤제
  - i. 참여도: 33.3
  - ii. 평가의 근거: 프로그램의 결과화면인 log.txt와 에러부분을 담당하였습니다. 세부적인 error를 담당해주었으며 모든 Print에 접근하여 파일 writer를 사용하여 log 파일을 구현하였습니다.

프로그램의 전체적인 부분을 균등하게 나누어 프로그램 구현을 일찍 시작하여 같이 한번에 프로그램을 작성한 것이 아니라 여유를 두고 하루 단위로 한 명씩 프로그램을 이어서 코딩하게 되었습니다.