Data Structure Lab. Project #1

Word Book

학　　과: 컴퓨터공학과

담당교수: 이기훈 교수님

수업시간: 금요일 2, 3

학　　번: 2015722094,

2015722093

성　　명: 안재원,

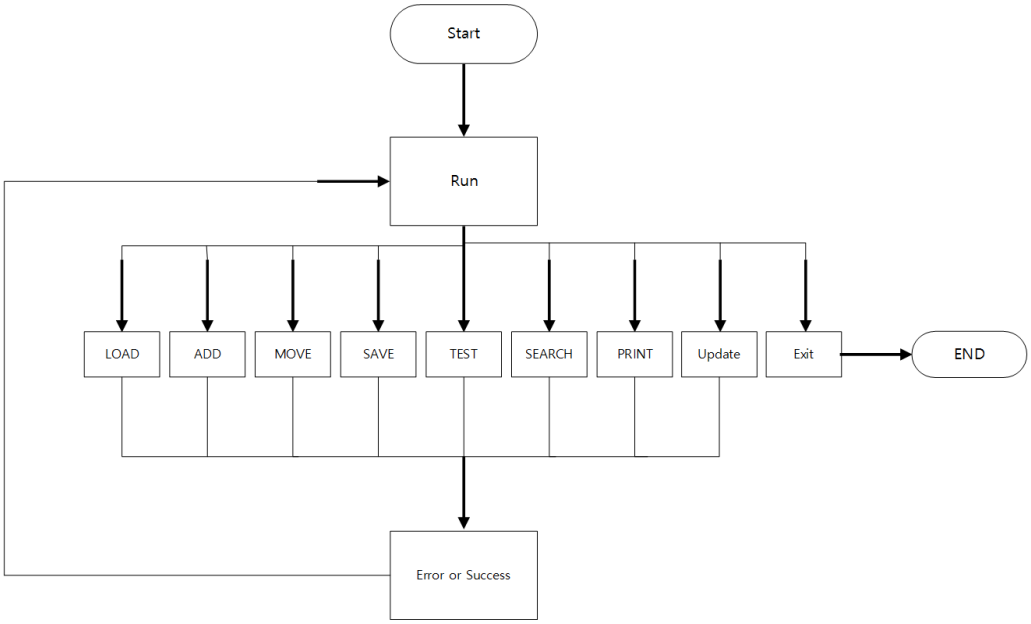
이종학

1. Introduction

- 본 프로젝트는 연결 리스트와 이진 탐색 트리와 환형 연결 리스트를 이용하여 영어

단어장을 만드는 프로젝트이다. 처음에 외워야 할 단어장인 TO_MEMORIZE에다가 연결 리스트를 이용하여 단어장을 구성해 주고 현재 외우고 있는 단어장인 MEMORIZING에 이진 탐색 트리를 이용하여 구성해 주고 다 외운 단어장인 MEMORIZED에 환형 연결 리스트를 이용하여 구성해 준다.

2. Flow Chart



-

3. Algorithm

- Queue.~queue

: 이 알고리즘은 소멸자로서 pHead를 따라서 pHead를 옮겨가면서 하나 하나씩 노드를 delete 해주도록 하여 노드를 없애주도록 하는 알고리즘이다.

- Queue.Push

: 이 알고리즘은 외워야 할 단어장인 Queue에다가 단어들을 넣어주어서 연결 리스트를 만들어주는 알고리즘이다. 하나도 없을 경우에만 첫 헤드에다가 노드를 넣어주고 그 다음부터는 먼저 넣은 순서대로 연결을 해 주도록 한다.

- Queue.Pop

: 이 알고리즘은 외워야 한 단어장인 Queue에서 단어를 빼주는 알고리즘이다. 먼저 넣은 것을 먼저 빼주도록 알고리즘을 구성해 주었다.

- Queue.Search

: 이 알고리즘은 queue에서 그 단어가 있는지 찾는 알고리즘이다. pWork를 이용하여 pHead에서부터 연결 리스트를 따라 그 단어와 입력 받은 단어를 비교하여 같은 것이 있는지 없는지 찾도록 한다. 찾을 경우 그 단어를 return하고 없을 경우에는 0을 return하도록 한다.

- Queue.Print

: 이 알고리즘은 외워야 할 단어인 Queue에 있는 단어를 모두 출력시켜주는 알고리즘이다. pWork를 이용하여 pHead부터 시작해서 연결 리스트를 따라 하나씩 모두 출력 시켜주도록 한다. 모두 출력시켜 주었을 경우에는 1을 return 해주고 queue가 비어있을 경우에는 0을 출력 시켜주도록 한다.

- Queue.Save

: 이 알고리즘은 외워야 할 단어를 queue에 넣어준 것을 to_memorize.txt 파일에 저장해주도록 하는 알고리즘이다. 먼저 쓰기용 파일인 to_memorize_word.txt. 파일을 열어서 하나씩 텍스트 파일에 저장해주고 파일을 닫아주도록 하였다. 파일이 열리지 않을 경우에는 0값을 return 했고 파일을 저장에 성공했을 경우에는 1값을 return 해주었다.

- CircularLinkedList.~circularlinkedlist

: 이 알고리즘은 소멸자로서 환형 연결리스트라 그 다음 값이 NULL값이 없기 때문에 pHead의 다음 부분을 pWork로 해주고 하나하나씩 지워서 pWork가 pHead까지 돌아올때 까지 delete를 해준다. 만약 노드가 하나도 없을 경우 return을 해준다.

- CircularLinkedList.Insert

: 이 알고리즘은 환형 연결 리스트에서 노드 하나씩 삽입 해주는 알고리즘이다. 연결 리스트와는 달리 환형 연결 리스트여서 마지막이 가리키고 있는 노드는 맨 처음 pHead가 되도록 리스트를 만들어 준다.

- CircularLinkedList.Search

: 이 알고리즘은 다 외운 단어장에서 단어를 찾는 알고리즘이다. 환형 연결 리스트이므로 pWork를 pHead로부터 시작해서 하나하나씩 찾아나가고 다시 pHead가 될 때까지 찾도록 한다. 만약 다 외운 단어장에 아무것도 없을 경우에는 Error인 0을 return 해 주도록 한다. 그리고 단어를 못 찾은 경우에도 0 값을 return 해준다.

- CircularLinkedList.Print

: 이 알고리즘은 다 외운 단어장을 출력 시켜주는 알고리즘이다. 환형 연결 리스트이므로 pWork를 pHead부터 시작해서 환형 연결 리스트를 따라 하나하나씩 출력 시켜주도록 한다. pWork가 다시 pHead일 때까지 출력시켜주고 다 출력이 되면 1 값을 return 시켜준다. 만약 다 외운 단어장에 아무것도 없을 경우에는 0 값을 return 시켜주도록 한다.

- CircularLinkedList.Save

: 이 알고리즘은 다 외운 단어장에 있는 리스트를 "memorized_word.txt" 파일에 저장시켜 주는 알고리즘이다. 먼저 쓰기 전용 파일인 memorized_word 텍스트 파일을 열은 다음에 하나하나씩 저장시켜 주도록 한다. 만약 파일이 열리지 않을 경우 0 값을 return 해주고 다 저장해주었을 경우에는 1 값을 return 해주고 파일을 닫아주도록 한다.

- WordBST

단어 노드를 트리구조로 저장하는 클래스로서 INSERT 함수를 이용하여 메모리할당이 완료된 노드의 포인터를 받아 그 노드의 단어를 기준으로 Binary Search Tree를 구현하였다.

Search문에서는 Search할 단어를 받아와 root로부터 시작하여 노드와 찾을 단어의 크기를 비교하여 노드의 단어가 크면 왼쪽자식으로 방향을 옮기고, 노드의 단어가 작으면 오른쪽 자식으로 방향을 옮기는 방식으로 단어를 찾아 단어를 찾았으면 그 단어의 노드를, 찾지 못했을 경우 0을 반환하는 함수이다.

Delete 문에서는 지울 단어를 받아와 Search문과 같은 알고리즘을 이용하여 지울 단어를 찾은 후 지울 단어가 leaf면 바로 지우고, 자식이 한 개이면 단어의 부모의 다리와 단어의 자식을 연결하여 단어를 삭제하고, 자식이 두 개일 경우 왼쪽 자식의 subtree중 가장 큰 노드를 지울 단어의 위치로 옮기는 방법으로 단어를 삭제하고, Delete함수가 성공 시 그 단어의 노드를 반환하고, 찾을 단어가 없을 경우 0을 반환하는 함수이다.

Print문에서는 tree를 프린트할 order를 받아와서 그 order에 맞는 방식으로 tree를 순회하여 프린트하는 함수이다. Order에는 pre-order, in-order, post-order, level-order가 있고, 각 order에는 재귀함수를 이용한 방법과 반복문을 이용한 방법으로 나누어서 프린트를 한다.

Save 문에서는 tree의 노드들을 다음에 load했을 때 다시 기존의 tree와 같은 모양으로 출력할 수 있도록 저장하기 위해서 pre-order 방식을 이용하여 tree의 노드들을 "memorizing_word.txt"에 저장한다.

Pre-order의 경우 work를 root에서부터 시작하여 root를 거치고, 왼쪽 자식을 거친 후, 오른쪽 자식의 노드로 이동하는 방법의 알고리즘을 각 반복식에 적용하여 tree를 순회하는 order이다

In-order의 경우 work를 root에서부터 시작하여 왼쪽 자식을 거친 후, 자기 자신의 노드에 수행할 동작을 수행한 후, 오른쪽 자식의 노드로 이동하는 방법의 알고리즘을 각 반복식에 적용하

여 tree를 순회하는 order이다.

Post-order의 경우 work를 root에서부터 시작하여 왼쪽 자식을 거친 후, 오른쪽 자식으로 이동하고, 수행할 동작을 마지막에 수행하는 방법의 알고리즘을 각 반복식에 적용하는 tree 순회 order이다.

Level-order의 경우 Queue를 이용하여 work를 root에서부터 시작하여 왼쪽 자식과 오른쪽 자식을 각각 순서대로 push를 한 후, 동작을 수행하고, work를 Queue에서 Pop하여 같은 방법의 알고리즘을 반복하는 트리 순회오더이다.

- AlphabetBST

이 클래스는 프로그램이 시작되면 바로 26자 각각의 Alphabet으로 이루어진 Binary Search Tree를 구현하고, 각각의 노드들이 WordBST를 갖도록 하는 Tree이다. AlphabetBST는 단어들이 하나의 WordBST에 저장될 경우 잘못하면 skewed tree가 생성될 수도 있기 때문에 단어들의 첫 글자들을 따로 구분하여 BST를 구현한다. 따라서 단어들을 Insert 할 때 AlphabetBST를 거쳐서 WordBST에 들어가도록 구현하고, Insert의 알고리즘은 WordBST와 같도록 구현한다.

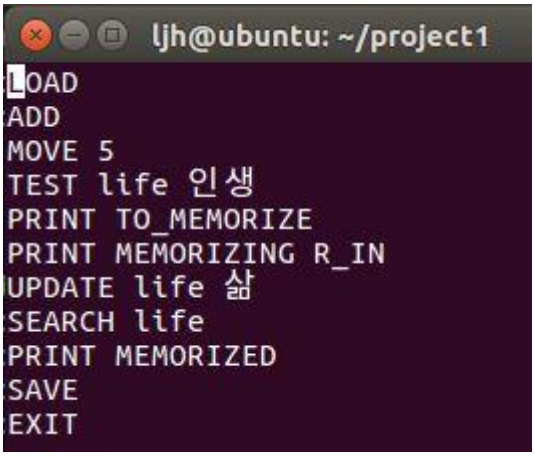
Print 문에서도 WordBST의 알고리즘과 같도록 구현하고, order를 받아 order대로 BST를 순회하고, 그 후, WordBST의 Print문을 호출하여 order을 전달하여 Print한다.

Search 또한 WordBST와 같은 알고리즘을 이용하여 단어의 첫 글자에 맞는 노드를 찾아서 그 노드의 WordBST에 Search를 호출하여 실행한다.

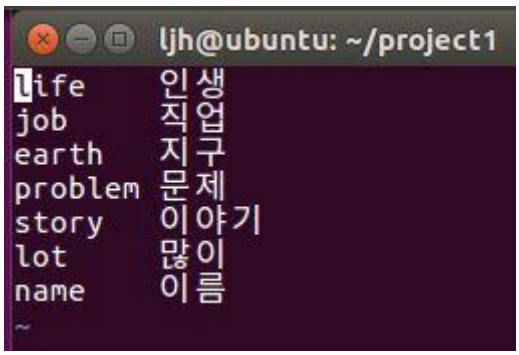
Save 또한 WordBST와 같은 방법을 이용하여 pre-order로 tree를 순회하고, WordBST의 Save를 호출하여 저장한다.

4. Result Screen

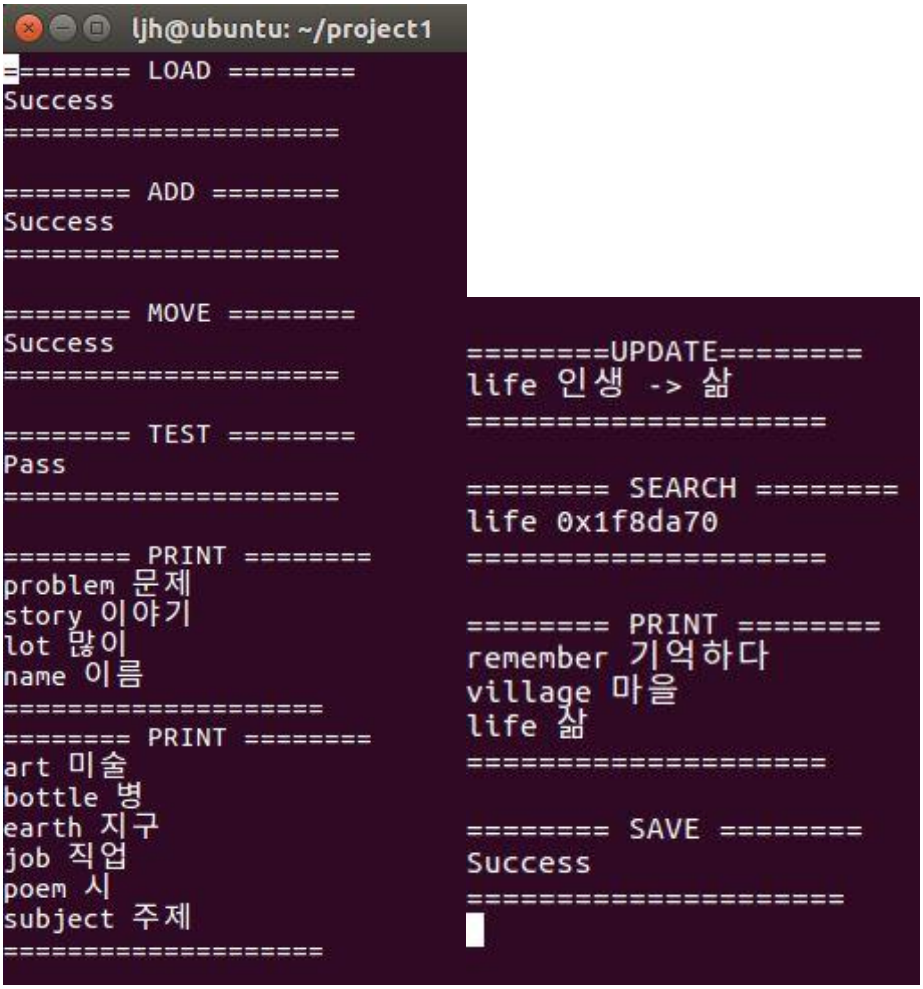
- Command.txt



- word.txt

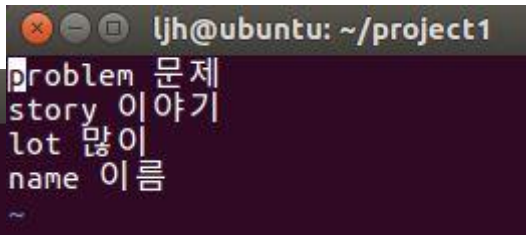


- log.txt

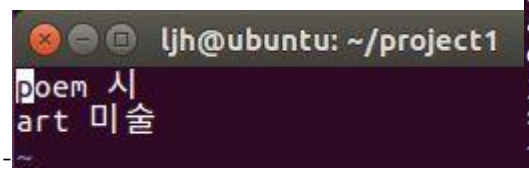


- to memorizing (before)

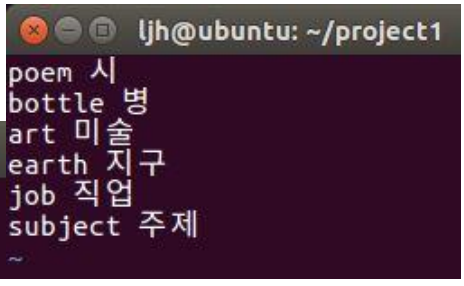
- to memorizing (after)



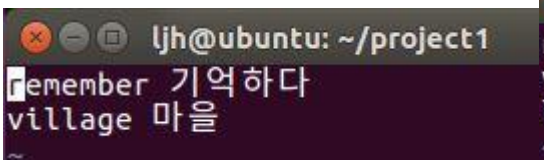
-memorizing(before)



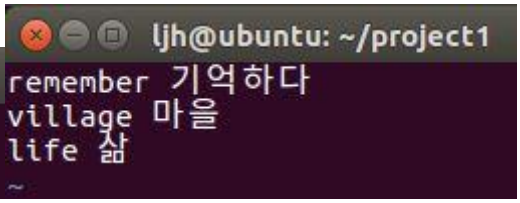
-memorizing(after)



- memorized(before)



- memorized(after)



- Manager.run

: 이 알고리즘은 전체적인 매니저를 관리하는 알고리즘이다. "Command txt" 파일을 읽어와서 Command에서 명령을 받아와 그 명령에 해당하는 알고리즘을 실행시켜주는 알고리즘이다. 각 알고리즘을 실행해주고 그 알고리즘이 성공 할 경우 각 상황에 맞게 출력시켜주고 알고리즘이 실패할 경우에 Error와 그 Error에 맞는 코드를 출력시켜주도록 한다. 또한 맨 처음에 AlphabetNode를 넣어주어서 AlphabetBST를 만들어주도록 하였다.

- Manager.LOAD

: 이 알고리즘은 원래 외워야 할 단어장(to_memorize.txt)이나 외우고 있는 단어장(memorizing.txt)이나 다 외운 단어장(memorized.txt)에 전에 들어있던 단어들을 불러오는 단어장이다. 만약 세 개의 텍스트 파일 중에 한 가지라도 열리지 않을 경우 에러가 뜨게 하였다. 각각의 파일을 단어별로 불러와서 외워야 할 단어는 queue->push에 넣어서 연결 리스트를 만들어 주었고 외우고 있는 단어는 bst->insert를 통하여 맨 처음 알파벳을 찾은 다음에 그 alphabetNode를 통하여 wordBST에다가 넣어서 이진 탐색 트리로 이어주었다. 마지막으로 외운 단어는 cll->Insert를 통해서 환형 연결 리스트를 통해 구성해 주었다. 그렇게 해서 다 LOAD가 되면 Success를 출력하도록 해주었다.

- Manager.ADD

: 이 알고리즘은 "word.txt", 단어장 텍스트 파일을 불러와서 외워야 할 단어장인 queue에 넣어주도록 하는 알고리즘이다. 파일이 열리지 않을 경우에 에러를 출력하도록 하였다. 단어장 텍스트 파일을 불러올 때 단어 별로 불러와서 queue에다가 넣어줘서 연결 리스트를 만들어 주도록 하였다, 그러나 만약에 외워야 할 단어장이나 외우고 있는 단어장이나 외운 단어장에 추가할 단어랑 똑같은 단어가 있을 경우에 그 단어는 Error를 출력시키지 않고 그냥 넘어가도록 알고리즘을 작성하였다. 또한 파일에 아무 단어도 없을 경우에 Error를 출력하도록 하였다.

- Manager.MOVE

: 이 알고리즘은 외워야 할 단어들을 외우고 있는 단어장으로 옮겨주는 알고리즘이다. 외우고 있는 단어장인 BST(memorizing.txt)에는 100개보다 많은 단어가 들어가면 안되므로 AlphabetBST에

서 세준 개수와 queue에서 넘겨 줄 단어 수의 합이 100보다 커지면 Error를 출력하게 해주었다. 또한 만약에 MOVE를 시켜줄 개수보다 외워야 할 단어장에 없을 경우 아얘 넣어주지 않고 바로 Error가 뜨게 하도록 하였다. Queue-pop을 통하여 단어를 하나씩 빼서 BST에 Insert하도록 알고리즘을 만들어 주었다. 다 성공하였을 경우 Success를 출력하도록 해주었다.

- Manager.SAVE

: 이 알고리즘은 지금의 queue와 BST와 cll에 들어가 있는 단어들을 저장해 주는 단어이다. 외워야 할 단어들은 queue에 있는 save 알고리즘에서 하나씩 찾아서 to_memorize.txt 파일에 저장을 하도록 해주었다. 외우고 있는 단어들은 bst에 있는 save 알고리즘에서 하나씩 찾아서 memorizing.txt. 파일에 저장을 하도록 해주었고 다 외운 단어는 cll에 있는 save 알고리즘에서 하나씩 찾아서 memorized.txt 파일에 저장을 하도록 해주었다. 다 성공하면 Success를 출력하도록 해주었고 저장에 실패할 경우 Error를 출력하도록 해주었다.

- Manager.TEST

: 이 알고리즘은 외우고 있는 단어장에 있는 단어를 command.txt 파일에서 써준 단어와 뜻이 맞는지 틀린지 TEST하는 알고리즘이다. 입력 받은 단어가 일단 binary search tree에 있는지 찾도록 한다. 찾지 못했을 경우에 Error를 출력하도록 해주고 찾았을 경우 입력 받은 뜻과 찾은 단어의 뜻이 같은지 같지 않은지 판별해 주도록 한다.
만약 단어의 뜻이 같을 경우에 Pass를 출력하도록 해주고 다를 경우에는 Error를 뜨게 해주도록 한다.

- Manager.SEARCH

: 이 알고리즘은 입력 받은 단어가 외워야 할 단어장이나 외우고 있는 단어장이나 다 외운 단어장에 있는지 찾아보는 알고리즘이다. 만약 외워야 할 단어장에도 없고 외우고 있는 단어장에도 없고 다 외운 단어장에도 없는 경우에는 찾지 못한 경우이므로 Error를 출력하도록 해준다. 그러나 외워야 할 단어장에서나 외우고 있는 단어장에서나 다 외운 단어장 중에서 하나라도 찾은 경우에는 Success를 출력하도록 해준다.

- Manager.PRINT

: 이 알고리즘은 command에서 입력 받은 단어장에 따라 그 단어장을 출력시켜주는 알고리즘이다. 만약 command에서 TO_MEMORIZE를 입력 받았을 경우 외워야할 단어장인 queue->print를 출력하도록 해준다. 만약 출력을 하지 못했을 경우에 Error를 출력하도록 해준다. 출력을 하였을 경우에는 print된 목록을 출력하도록 해준다. MEMORIZING을 입력 받았을 경우에는 외우고 있는

중인 단어장에다가 출력을 해주어야 하므로 bst->print를 출력하도록 해준다. 마찬가지로 출력을 하지 못했을 경우에는 Error를 출력하도록 한다. 마지막으로 command에서 MEMORIZED를 입력 받았을 경우에는 다 외운 단어장인 cll->print를 이용하여 출력을 해주고 출력을 하지 못했을 경우에는 Error를 출력해주도록 한다.

- Manager.UPDATE

: 이 알고리즘은 원래 단어장에 있는 단어의 뜻을 바꿔주는 알고리즘이다. 먼저 외워야 할 단어장, 외우고 있는 중인 단어장, 다 외운 단어장에 이 단어가 있는지 찾도록 한다. 못 찾았을 경우에는 Error를 출력하도록 하고 찾았을 경우에는 그 단어의 원래의 뜻을 입력 받은 뜻으로 바꾸어 주도록 한다. 만약 바꿔주는 데 성공했을 경우에 단어의 뜻이 어떻게 바뀌었는지 출력하도록 한다.

5. Consideration

-조원(2015722094, 안재원)

: 이번 프로젝트에서 모든 Node와 Queue, CircularLinkedlist, Manager를 구현하는 역할을 했다. 이번 프로젝트에서 bst를 중간에 나누기 힘들어서 이렇게 역할을 분담하였고 조장인 이종학님이 열심히 하셔서 그 것에 걸맞게 열심히 하도록 노력하였고 모르는 것에 대해 많이 물어보고 친절히 알려주어서 많은 도움이 되었다고 생각한다. 그래서 내 스스로 생각하는 점수는 10점이라고 생각하고 조장인 이종학님도 10점이라고 생각한다. 이번 프로젝트는 여태까지 했던 링크드 리스트와 환형 링크드 리스트 그리고 이번에 새로 배운 트리에 대한 종합적인 프로젝트 였다고 생각한다. 또한 리눅스를 처음 사용하는 것이어서 아직 리눅스에 대해 익숙하지 않은 점도 있었고 비쥬얼 스튜디오와 리눅스가 다른 점도 있어서 고치는데 문제가 있었다. 이번 팀프로젝트에서 팀별로 하기 때문에 팀원한테 도움도 받을 수 있고 서로 모르는 부분에 대해 도움을 주면서 나눠서 하기 때문에 해야할 분량이 주는 장점도 있었지만 서로 생각하는 부분도 다르고 나눠서 하기 때문에 남의 코드를 이해해야지 할 수 있는 단점도 있었다. 그래도 많은 도움이 되었다고 생각하는 팀 프로젝트였다.

-조장(2015722093, 이종학)

처음 과제를 시작할 때는 평소에 사용해오던 툴인 비쥬얼 스튜디오가 아닌 리눅스로 코드를 완성하여 제출하여야 된다는 것 때문에 디버깅을 어떻게 해야 하는지 몰랐기 때문에 코드를 완성하기 전에 오류를 찾지 못 할 것 같아서 난해하였지만, 비주얼로 코드를 짜고 리눅스로 옮겨 에러가 뜨는 부분을 고치는 방법을 사용하여 처음에 생각하였던 난해함을 해결하여 무난히 프로젝트를 진행할 수 있었습니다. 프로젝트에서 주로 BST 클래스의 코드를 작성하는 역할을 수행하였고, 수업에서 배운 재귀함수를 이용한 order는 무난히 수행하였으나, 반복문을 이용하는 order의 경우

처음 알고리즘을 생각할 때 고민을 좀 해야 하였기 때문에 난이도가 적절하였다고 생각합니다.

프로젝트를 진행하면서 같은 팀원인 안재원님과 서로 평소에 코딩을 하는 방법과 능력에 따라 역할을 분담하여 저는 데이터 구조 수업에서 배운 BST 부분과 비주얼 툴에서 리눅스 툴로 코드를 변환시키며 오류를 찾고 해결하는 부분의 역할을 수행하였고, 팀원인 안재원님은 node 클래스들과 queue, circularLinkedList 클래스를 구현하였으며, Manager 파일을 생성하며 저와 함께 상의하며 도움을 받아 Manager 파일생성까지의 역할을 수행하였습니다. 이번 팀 프로젝트를 진행하며 서로의 역할분담이 적절히 진행되었고, 서로 상의가 잘되어 서로에게 도움이 잘 되어 괜찮은 팀 프로젝트가 되었다고 생각합니다. 본인이 생각하는 본인의 점수는 10점이라고 생각하며 팀원의 점수도 10점을 받을 가치가 있다고 생각합니다.