Data Structure Project

Project #1

담당교수 : 이기훈

제출일 : 2018. 10. 03.

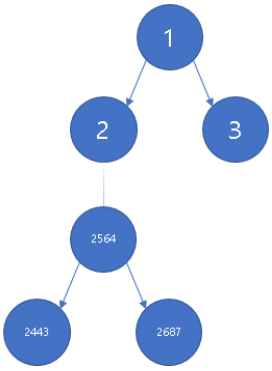
학과 : 컴퓨터정보학부

학번 : 2015722025

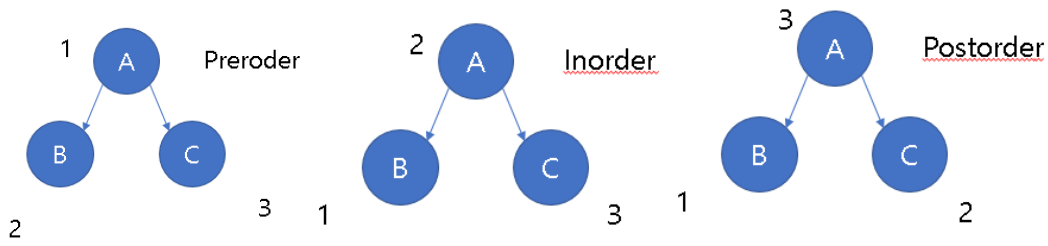
이름 : 정용훈

1. Introduction

이번 1차 프로젝트의 전체적인 내용은 세가지 자료구조 즉 linked list와 BST, Queue를 통하여 차량 보험 관련 정보를 관리하는 프로그램을 구현하는데 있습니다. 기존의 만들어진 SAVE파일 즉, 자료구조에 의해 만들어진 파일이 없다면 car.txt파일에서 새로운 정보들을 가져와 linked list를 구현합니다. 그리고 가져온 정보로부터 사고를 낸 사람들을 구분하여 다음 자료구조를 만들어 주어야 하는데 그에 대한 명령어가 MOVE입니다. MOVE는 인자로 자연수의 값을 가질 수 있습니다. Ex)MOVE 20 해당 숫자만큼 linked list에서 순차적으로 사고를 낸 사람들의 정보를 가져와 Binary Search Tree를 만들게 됩니다. BST의 구조로는 우선 number node가 구성되어 있으며 차량 맨 앞 번호를 비교하여 각 number node에서 각각의 BST를 구성하게 됩니다. 쉽게 말해 옆 이미지에서 보는 것과



같이 BST안에 BST가 또 구성이 되어있는 구조입니다. 다음으로는 Queue입니다. Queue는 사고를 낸 사람들에 한해서 보험처리를 완료해주어야 하는데 완료되는 명단이 바로 자료구조 queue에 들어가게 됩니다. Queue는 LIFO방식으로 자료가 구성되며 이는 마지막에 들어간 자료는 마지막에 나오는 구조입니다. 여기까지 설명한 부분이 자료를 구성하는데 가장 큰 역할을 하는 설명입니다. 또한 이번 프로그램에서는 BST에 대한 PRINT의 명령을 받아 명령을 받은 PRINT의 방법데로 결과를 출력하도록 되어있습니다. 출력하는 함수로는 크게 4가지로 나눌 수 있습니다. Preroder, Inorder, Postorder, levelorder로 나눌 수 있으며 각각의 프린트 방법은 아래와 같습니다.



각각의 order를 나타낸 이미지입니다. 방문 순서는 나와있는 숫자의 순서데로 방문하게 됩니다. B와 C에 sub트리가 있어도 방법은 같습니다. 각각의 오더들의 방문 순서를 정리하면 아래와 같습니다.

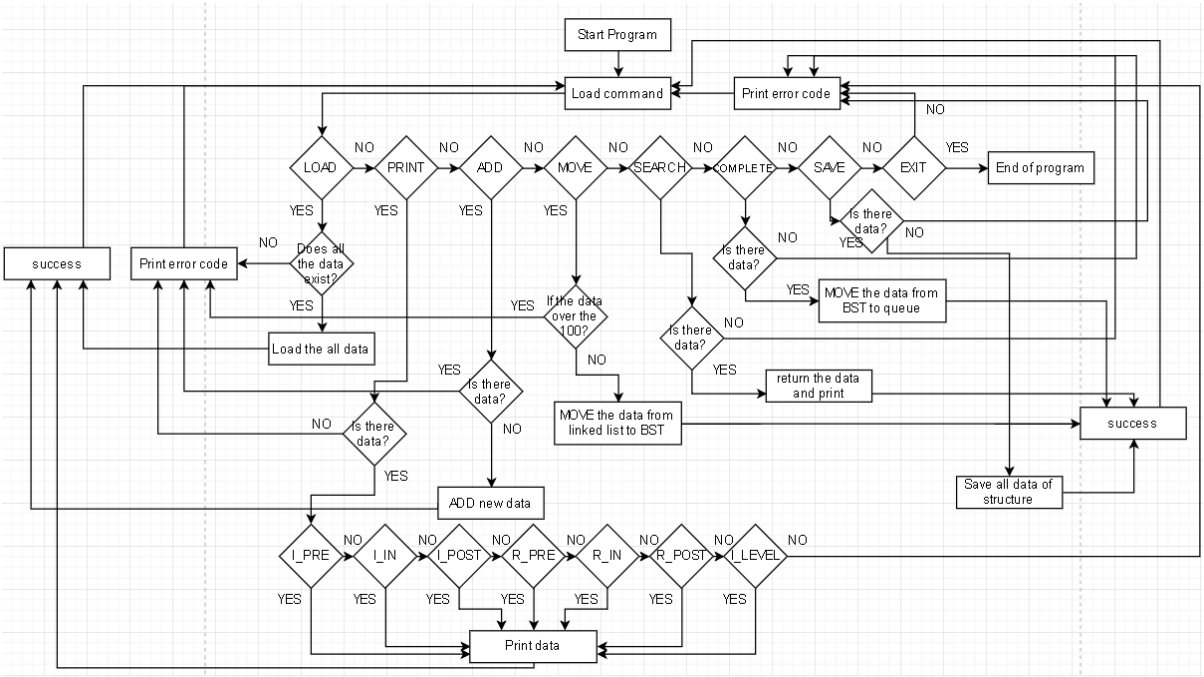
Preorder : 노드 방문 -> 왼쪽 서브 트리 -> 오른쪽 서브 트리

Inorder : 왼쪽 서브 트리 -> 노드 방문 -> 오른쪽 서브 트리

Postorder : 왼쪽 서브 트리 -> 오른쪽 서브 트리 -> 노드 방문

마지막으로는 levelorder가 있습니다. Levelorder는 맨 위 노드부터 같은 높이에 있는 노드들을 왼쪽에서부터 오른쪽으로 순서데로 방문하는 order입니다. 이렇게 프로그램에는 크게 4가지의 BST를 방문하는 order가 있습니다. 또한 이 order를 구성하는 방법으로는 여러가지가 있는데 특히 이번 설계에서는 levelorder를 제외한 나머지 3개의 order를 재귀함수(Recursive function)를 이용한 방법이 있고 반복 함수(iterator function)을 사용한 방법 이 두가지를 각각 구현하게 되었습니다. 그리고 각 자료구조를 저장하기 위한 SAVE명령어에 관련된 설명입니다. Linked list와 queue는 단순한 연결로 되어있기 때문에 저장하는데 어려움없이 앞에서부터 끝까지 순차적으로 저장하면 되지만 BST는 나중에 파일을 불러와야 하는 것을 생각해야하기 때문에 특정한 order를 사용하여 저장해주어야 합니다. 방법으로는 preorder를 사용하여 방문순서데로 데이터를 저장해주면 데이터를 불러와 BST를 다시 구성하는데 있어 원래 데이터구조데로 불러오기가 가능합니다. 마지막으로 데이터의 누수를 막기위하여 종료전에 모든 동적 할당된 데이터들을 지워주며 작업을 마무리해야 합니다. 이 또한 BST는 특정 order를 따라 delete를 시켜주어야 합니다. BST의 Leaf부터 해제를 시켜주면 되므로 Postorder를 사용하여 밑에서부터 데이터를 지워주면 데이터 누수를 막을 수 있습니다.

2. Flow Chart

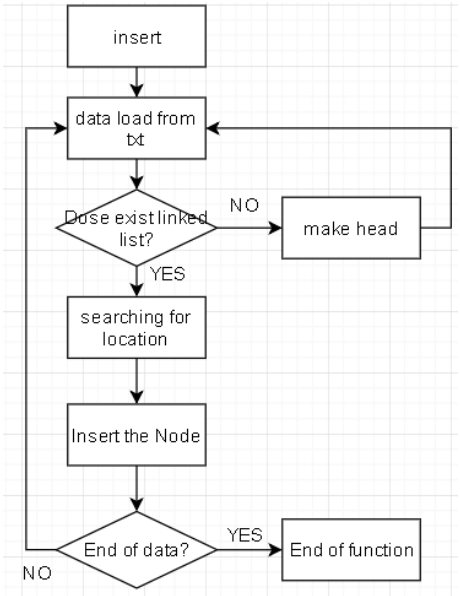


해당 flow chart는 프로젝트의 동작을 구현해 놓은 chart입니다. 해당하는 함수의 detail 한 동작은 숨겨져 있으며 명령어를 받았을 때 구분하는 알고리즘과 명령어가 알맞은 경우 어떠한 동작 또는 Print해야하는 order에 대한 종류와 동작이 그려져 있습니다. 마지막으로 해당 함수에 대한 데이터를 어떤 식으로 처리하는지에 대하여 쉽게 볼 수 있도록 그린 것입니다.

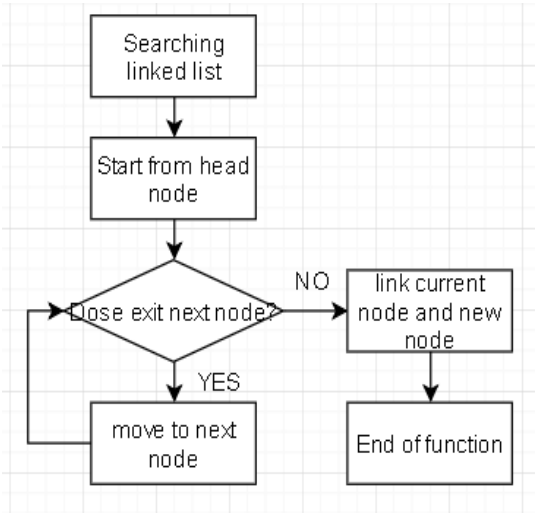
3. Algorithm

해당 프로젝트에서 가장 중요한 목적은 알맞은 자료구조를 구축하는데 있습니다. 자료를 구축하는데 있어 가장 중요한 것은 정보를 삽입하는 과정인 insert함수라고 생각됩니다. 아래 flow chart는 자료구조를 구성하는데 있어 가장 기본이 되는 insert함수의 알고리즘을 정리한 것입니다. 이번 프로젝트에서는 3가지의 자료구조인 linked list, Binary search Tree 그리고 Queue를 사용하였습니다. 각각의 자료구조의 세부적인 insert의 동작은 새로운 노드가 들어갈 위치를 찾는 것이 가장 중요합니다 제가 사용한 위치를 찾는 과정은 다음과 같습니다.

Basic insert algorithm

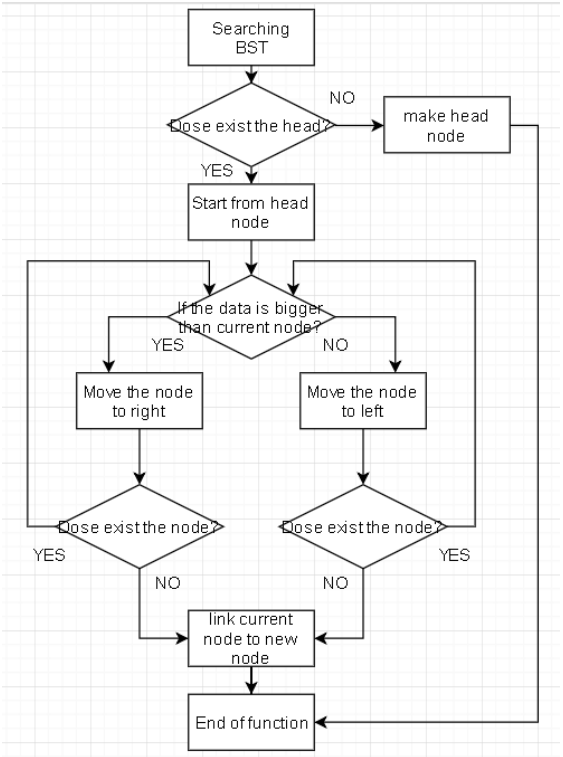


Searching location linked list & queue

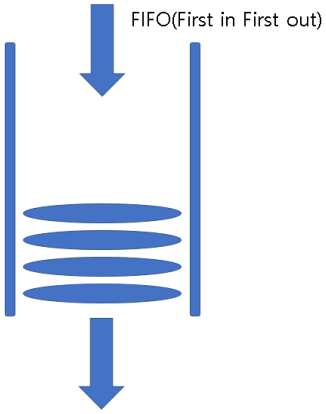


다음과 같이 linked list와 queue의 insert 알고리즘은 단순히 쭉 연결된 자료구조에서 가장 끝 노드를 찾아 새로 들어온 노드를 기존의 tail node와 연결해주면 간단하게 해결됩니다. 이번 프로젝트에서 가장 생각을 많이 해야 할 Binary Search Tree는 다음 과 같은 알고리즘을 사용하여 자료를 구축하게 됩니다.

Searching location BST



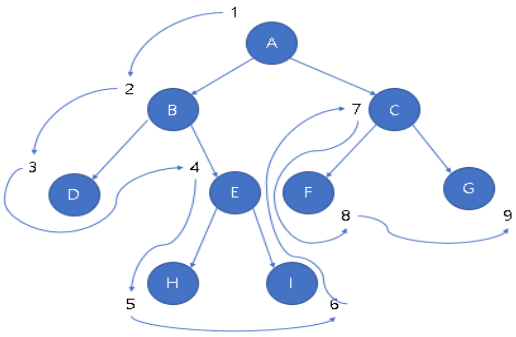
옆 이미지는 BST의 노드가 들어갈 자리를 찾는 알고리즘입니다. BST의 특징으로는 데이터의 특정 자료 즉, KEY의 값을 비교하여 비교대상이 되는 노드보다 크면 오른쪽으로 연결하며 작으면 왼쪽으로 연결하는 규칙을 갖게 됩니다. 다시 말해 새로운 노드가 들어오면 기존에 있던 BST의 노드들과 비교를 하며 current node의 방향이 왼쪽으로 갈지 오른쪽으로 갈지가 결정되며 새로운 노드가 들어갈 자리로는 마지막으로 비교 대상이 된 노드의 NULL이 가리키고 있는 방향으로 노드가 삽입되게 됩니다. 이러한 BST의 구조는 귀찮아 보일 수 있지만 자료를 찾는데 있어 일일이 하나씩 비교하는 것이 아니라 비교대상을 1/2씩 줄일 수 있는 장점이 있어 searching이 다른 구조에 비해 빠르다는 장점이 있습니다. 다음으로 Delete에 관련된 알고리즘입니다. Linked list에서는 삭제를 할 때 앞 또는 뒤에서부터 삭제를 시도해도 무관합니다. 하지만 자료구조 중에 queue가 포함이 되어있으므로 queue의 설명에 대해서는 짚고 넘어갈 필요가 있습니다. 아래 그림과 같이 자료구조 Queue는 FIFO방식의 자료 구조입니다.



FIFO의 뜻은 먼저 들어간 자료가 먼저 나오게 되는 방식으로 가장 먼저 Insert를 해준 node가 있다면 지우는 과정에서 가장 먼저 들어갔던 node가 가장 먼저 나와야 한다는 규칙이 있습니다. 지우는 함수를 Pop이라 하는데 pop이 실행이 되면 queue구조에서 head를 먼저 지운 후 다음 노드를 head로 선언해주는 방식을 반복적으로 하여 queue의 자료구조를 모두 지울 수 있습니다. BST의 삭제 알고리즘으로는 아래에서 설명한 preorder를 사용하여 leaf부터 방문하여 Delete 를 해주면 쉽게 메모리를 삭제할 수 있습니다.

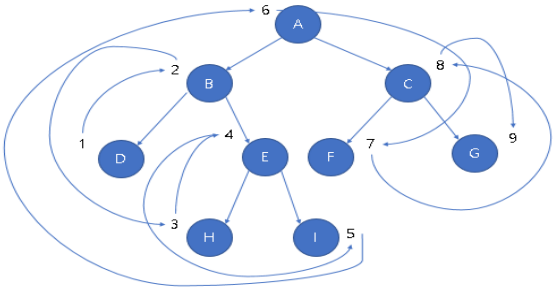
다음 소개할 알고리즘으로는 BST에서 상황에 따라 여러 가지 방법으로 사용할 수 3가지의 대표적인 order를 소개하겠습니다. 우선 원리에서 간단하게 설명하였지만 조금 더 구체적인 Tree를 사용하여 알아보겠습니다.

Preorder



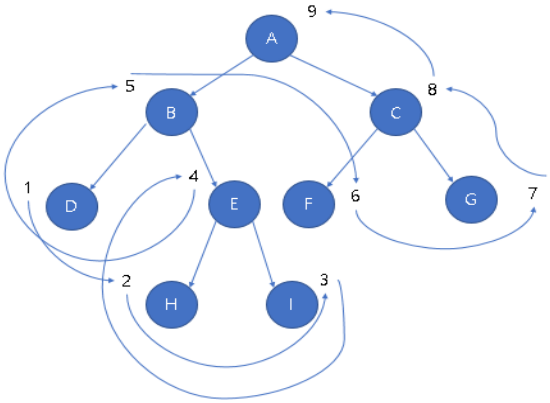
첫번째 preorder즉 전위 순회입니다. 전위 순회는 가운데 노드를 먼저 방문하고 순차적으로 왼쪽 서브 트리 오른쪽 서브 트리 순으로 방문을 하는 order를 뜻합니다. 재귀 함수를 사용한다면 가운데 노드에서 먼저 출력 후 왼쪽 노드를 인자로 함수 호출 다음으로 오른쪽 노드를 인자로 함수 호출을 하면 preorder를 따라 순회를 하게 됩니다.

Inorder



두번째는 Inorder즉 중위 순회입니다. 방문 순서로는 왼쪽 서브 트리 가운데 노드 오른쪽 서브 트리 순으로 방문을 하는 order입니다. 재귀 함수를 사용한다면 왼쪽 노드를 인자로 하는 함수 호출 후 가운데 노드를 출력하고 오른쪽 노드를 인자로 하는 함수를 호출하면 Inorder에 따라 순회를 하게 됩니다.

Postorder



마지막으로는 Postorder즉 후위 순회입니다. 방문순서는 왼쪽 서브 트리 오른쪽 서브 트리를 방문하고 가운데 노드를 방문하는 방식입니다. 재귀 함수를 사용한다면 왼쪽 노드를 인자로 하는 함수를 호출하고 오른 쪽 노드를 인자로 하는 함수를 호출한 후 가운데 노드를 출력하면 postorder에 따라 순회하게 됩니다.

특히 Postorder는 서브 트리를 먼저 지우고 root가 되는 노드를 가장 마지막에 지우기 때문에 BST를 Delete할 때 사용하면 좋다는 장점이 있습니다.

- 프로젝트에서 사용한 알고리즘의 동작을 설명

4. Result Screen

첫번째 실행 명령

```
LOAD
PRINT COMPLETE_LIST
ADD
PRINT TOTAL_LIST
MOVE 60
SEARCH 3654
PRINT EVENT_LIST R_IN
PRINT EVENT_LIST I_PRE
COMPLETE 3654
COMPLETE 1546
COMPLETE 9048
COMPLETE 9999
PRINT COMPLETE_LIST
LOAD
SAVE
EXIT
```

```
======== ERROR ========
100
======================== 첫번째 실행하는 명령어로써 LOAD할 txt파일이 없으므로 에러코드가 나오는 것을 확인할 수 있다.

======== ERROR ========
600
======================== 두번째 실행하는 명령어로써 현재 데이터 구조가 구축이 되지 않았으므로 Print할수있는 정보가 없어 에러코드가 나온다.

======== ADD ========
Success
======================== 세번째 실행하는 명령어로써 데이터가 없으므로 최초로 car.txt에서 정보를 가져오는 명령어로 함수 실행에 성공하여 성공 문구를 확인할 수 있다.
```

======== PRINT ========
3654 / 조만재 / Y
6873 / 김태환 / Y
8925 / 김홍지 / N
7931 / 김호준 / N
8695 / 신희민 / Y
5703 / 전선경 / Y
2754 / 권주영 / N
8971 / 박채원 / N
9705 / 조은별 / Y
8014 / 남궁민 / Y
3041 / 이지은 / Y
5084 / 유한욱 / N
8605 / 정현우 / Y
7635 / 한성민 / N
6053 / 이한솔 / N
2549 / 전예은 / N
2763 / 전보은 / Y
9048 / 이미연 / N
4301 / 김은지 / Y
7382 / 백현은 / N
9138 / 남윤창 / Y
5028 / 김성훈 / N
9314 / 이지연 / N
2748 / 김종훈 / N
4623 / 박지현 / Y
7435 / 김동영 / Y
2645 / 최지은 / N
1639 / 박영수 / Y
1074 / 김성호 / N
6128 / 정상훈 / Y
2317 / 박미경 / Y
2980 / 조정연 / Y
4268 / 박혜미 / N
9874 / 안주연 / N
9178 / 서예지 / Y
8126 / 차성우 / Y
9254 / 한준희 / Y
2635 / 김준석 / N
7954 / 전성원 / N
8063 / 홍현기 / Y
9720 / 김준동 / Y
8495 / 신진한 / N
2879 / 김윤기 / N
8547 / 심재현 / Y
1506 / 고성현 / Y
1497 / 송민준 / Y
2361 / 문요한 / N
6123 / 김효빈 / Y
6538 / 권혁신 / N
8326 / 강예정 / N
7965 / 배득현 / N
8534 / 신호준 / Y
3629 / 김도엽 / N
9743 / 김기선 / Y
4629 / 주재홍 / N
1068 / 김지용 / Y
3165 / 이명진 / N
8432 / 나승학 / N
6725 / 이지훈 / N
4860 / 김재준 / Y
8694 / 김주형 / Y
4236 / 이상효 / N
1467 / 서진석 / Y
6251 / 박현 / N
2916 / 손성주 / Y
4173 / 정동찬 / Y
6103 / 정의헌 / Y
3580 / 장윤석 / N
2934 / 이성민 / N
3459 / 손상렬 / Y
2880 / 강창희 / Y
8905 / 박지훈 / Y
3160 / 이병민 / N
8470 / 임연수 / N
3452 / 김동현 / Y
9217 / 우동혁 / Y
2935 / 정병찬 / N
1892 / 정윤석 / N
9318 / 정석호 / Y
7189 / 신기환 / Y
1739 / 양승균 / Y
4673 / 박수빈 / N
7493 / 이태빈 / Y
8406 / 주정인 / N
5083 / 민세기 / N
9542 / 김동욱 / N
2795 / 최지영 / Y
9378 / 임언석 / N
9685 / 김태연 / Y
4576 / 정민호 / N
7935 / 옥준영 / Y
7329 / 고재현 / N
4592 / 황성영 / N
3124 / 김성희 / N
7053 / 박일영 / Y
2439 / 김백호 / Y
8273 / 송민규 / N
2648 / 강희승 / Y
7843 / 박상준 / N
1958 / 변주형 / Y
4976 / 이기명 / Y
7826 / 임동준 / Y
6794 / 정우식 / N
5714 / 강윤구 / N
4783 / 김유성 / Y
3029 / 이호영 / Y
6980 / 고정원 / Y
3856 / 김시훈 / N
2519 / 남건욱 / N
5147 / 김동현 / Y
6853 / 김태영 / Y
1602 / 조원호 / N
9865 / 박민규 / N
6572 / 박종현 / Y
1689 / 김동민 / Y
9031 / 이형민 / Y
2764 / 김봉효 / N
6840 / 최재은 / Y
7248 / 양효령 / N
6072 / 최종현 / N
6917 / 박현아 / N
2520 / 김현아 / Y
9684 / 이민영 / N
8796 / 김혜리 / Y
2096 / 강희정 / N
7301 / 온희라 / Y
4691 / 김자영 / N
3140 / 신진호 / N
1798 / 조만희 / N
7385 / 정현우 / Y
4198 / 김재윤 / Y
3607 / 이승호 / N
6082 / 고은나래 / Y
1725 / 서나리 / N
9485 / 김소희 / Y
9736 / 한혜인 / Y
7682 / 김홍지 / Y
9307 / 유한욱 / N
6153 / 이경진 / N
8465 / 이소영 / Y
1745 / 강민호 / Y
7148 / 함하늘 / Y
1603 / 장한별 / N
1392 / 박재훈 / N
4953 / 윤민규 / Y
2341 / 최영진 / Y
7325 / 이창욱 / N
4052 / 이남규 / N
1459 / 이전제 / Y
9157 / 조종현 / Y

```
3016 / 권영상 / Y      7486 / 이소정 / Y
7518 / 김병휘 / N      9807 / 김민경 / Y
7018 / 허준 / Y        4632 / 임소리 / Y
8539 / 오재원 / N      1483 / 김유리 / N
7025 / 김석호 / N      1275 / 오지현 / N
2305 / 최용락 / N      1785 / 문초이 / Y
2860 / 고재혁 / Y      8563 / 박세영 / Y
2180 / 김민우 / N      7591 / 신정민 / Y
3094 / 김민섭 / Y      5870 / 홍도이 / N
8372 / 권우창 / N      6134 / 이유진 / N
6345 / 김동우 / Y      7350 / 유재연 / Y
8935 / 방호상 / N      9236 / 김용수 / Y
9730 / 김건우 / N      8476 / 최연희 / N    9641 / 송예찬 / N
4380 / 김종현 / N      3549 / 조재용 / N    6701 / 정하영 / N
5862 / 홍명오 / Y      2367 / 고유진 / Y    7968 / 박민춘 / N
6359 / 김태원 / Y      5189 / 허금조 / Y    2946 / 정창우 / Y
7613 / 남해울 / N      4327 / 최유미 / Y    8069 / 홍진혁 / N
4918 / 송광현 / Y      5907 / 양근보 / N    4905 / 유현근 / Y
7829 / 장다빈 / N      8532 / 공혜준 / Y    3408 / 오윤제 / N
4376 / 홍순우 / Y      7945 / 서지현 / N    6721 / 최진호 / Y
4150 / 이순형 / Y      1762 / 박지은 / N    1903 / 오성수 / N
2485 / 최경찬 / Y      4652 / 김지영 / N    3896 / 김형인 / N
4298 / 이강풍 / N      1729 / 김준모 / Y    7659 / 송태근민 / N
3426 / 송인철 / N      3691 / 고경아 / N    1532 / 한성민 / Y
9624 / 유민선 / Y      6091 / 홍유경 / Y    7302 / 백하담 / Y
4630 / 권민 / Y        8596 / 안예림 / N    7349 / 허준우 / N
3125 / 음승태 / Y      1024 / 이승희 / Y    5640 / 박동현 / Y
8710 / 강동수 / N      2416 / 박시연 / N    4501 / 이유진 / N
1823 / 이세영 / N      5038 / 배홍섭 / N    3849 / 이두호 / Y
4608 / 김태진 / Y      3947 / 이진호 / N    6520 / 김병기 / Y
3249 / 최영진 / Y      7560 / 이승우 / Y    7326 / 김영태 / Y
6543 / 임형빈 / N      8904 / 홍석우 / Y    9751 / 노혜리 / N
9132 / 고준영 / N      8036 / 최재현 / N    4567 / 이재승 / N
3275 / 장성록 / Y      6852 / 박승진 / Y    2036 / 이준혁 / Y
4612 / 오남훈 / Y      8091 / 곽태웅 / N    9247 / 이다연 / Y
5406 / 엄태현 / Y      2745 / 임치헌 / Y    2687 / 곽민우 / Y
3469 / 노재현 / N      7021 / 신현욱 / Y    2481 / 남건우 / N
6129 / 이동준 / Y      7365 / 김진범 / Y    7632 / 정준영 / N
1095 / 김남고 / N      9183 / 이정호 / N    3742 / 박유림 / Y
8760 / 우성윤 / N      1238 / 조두현 / N    2913 / 노선영 / Y
2458 / 박재우 / N      7683 / 윤상민 / Y    7918 / 최윤혜 / N
9012 / 안홍경 / Y      5381 / 김광호 / Y    9842 / 이한수 / N
7504 / 강민주 / N      3501 / 안승기 / Y    7263 / 김재원 / Y
4375 / 오영헌 / Y      3261 / 김효준 / N    2985 / 김태훈 / Y
6319 / 정찬호 / N      3807 / 권영서 / N    1057 / 김대망 / Y
9403 / 박덕영 / Y      9241 / 주재훈 / Y    5018 / 전효희 / N
6307 / 심민섭 / N      2968 / 김현준 / Y    6795 / 방소윤 / Y
3497 / 조건웅 / N      5478 / 최현구 / N    8593 / 전영광 / N
7485 / 최은진 / N      7281 / 이진수 / N    6793 / 임경택 / N
5471 / 정유나 / Y      4985 / 오형근 / Y    7215 / 마예린 / N
9015 / 이영서 / Y      4638 / 임유섭 / Y    4023 / 강연구 / Y
3854 / 김지홍 / N      7589 / 박규민 / Y    7149 / 이근영 / N
3076 / 윤지혜 / Y      8260 / 정용훈 / N    1395 / 김수진 / Y
7835 / 이혜연 / N      1674 / 손현식 / Y    ========================
```

네번째 실행 명령어로 linked list에 저장된 정보를 모두 print하는 명령어이다.

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

다섯 번째 실행 명령어로 60을 인자로 주었고 100명이 넘지 않았기 때문에 에러코드는 나오지 않는다.

```
======== SEARCH ========
3654 조만재 Y
=======================
```

여섯 번째 실행 명령어로 3654라는 KEY 값으로 해당 정보를 찾아 성공한 모습이다.

```
======== PRINT R_IN ========
0
1
1068 김지웅 Y
1467 서진석 Y
1497 송민준 Y
1506 고성협 Y
1639 박영수 Y
1689 김동민 Y
1739 양승균 Y
1958 변주형 Y
2
2317 박미경 Y
2439 김백호 Y
2648 강희승 Y
2763 전보은 Y
2795 최지영 Y
2880 강창희 Y
2916 손성주 Y
2980 조정연 Y
3
3029 이호영 Y
3041 이지은 Y
3452 김동현 Y
3459 손상렬 Y
3654 조만재 Y
4
4173 정동찬 Y
4301 김은지 Y
4623 박지현 Y
4783 김유성 Y
4860 김재준 Y
4976 이기명 Y
5
5147 김동현 Y
5703 전선경 Y
6
6103 정의헌 Y
6123 김효빈 Y
6128 정상훈 Y
6572 박종현 Y
6853 김태영 Y
6873 김태환 Y
6980 고정원 Y
7
7053 박일영 Y
7189 신기환 Y
7435 김동영 Y
7493 이태빈 Y
7826 임동준 Y
7935 옥준영 Y
8
8014 남궁민 Y
8063 홍현기 Y
8126 차성우 Y
8534 신호준 Y
8547 심재현 Y
8605 정현우 Y
8694 김주형 Y
8695 신희민 Y
8905 박지훈 Y
9
9138 남윤창 Y
9178 서예지 Y
9217 우동혁 Y
9254 한준희 Y
9318 정석호 Y
9685 김태연 Y
9705 조은별 Y
9720 김준동 Y
9743 김기선 Y
========================
```

재귀함수를 사용한 Inorder 방식의 print를 명령한 결과 화면이다.

```
======== PRINT I_PRE ========
7
7435 김동영 Y
7189 신기환 Y
7053 박일영 Y
7493 이태빈 Y
7935 옥준영 Y
7826 임동준 Y
3
3654 조만재 Y
3041 이지은 Y
3029 이호영 Y
3459 손상철 Y
3452 김동현 Y
1
1639 박영수 Y
1506 고성협 Y
1497 송민준 Y
1068 김지웅 Y
1467 서진석 Y
1739 양승균 Y
1689 김동민 Y
1958 변주형 Y
0
2
2763 전보은 Y
2317 박미경 Y
2439 김백호 Y
2648 강희승 Y
2980 조정연 Y
2916 손성주 Y
2880 강창희 Y
2795 최지영 Y
5
5703 전선경 Y
5147 김동현 Y
4
4301 김은지 Y
4173 정동찬 Y
4623 박지현 Y
4860 김재준 Y
4783 김유성 Y
4976 이기명 Y
6
6873 김태환 Y
6128 정상훈 Y
6123 김효빈 Y
6103 정의헌 Y
6853 김태영 Y
6572 박종현 Y
6980 고정원 Y
                    9
                    9705 조은별 Y
                    9138 남윤창 Y
                    9178 서예지 Y
                    9254 한준희 Y
                    9217 우동혁 Y
                    9318 정석호 Y
                    9685 김태연 Y
                    9720 김준동 Y
                    9743 김기선 Y
                    8
                    8695 신희민 Y
                    8014 남궁민 Y
                    8605 정현우 Y
                    8126 차성우 Y
                    8063 홍현기 Y
                    8547 심재현 Y
                    8534 신호준 Y
                    8694 김주형 Y
                    8905 박지훈 Y
                    ========================
```

반복 함수를 사용한 preorder의 print 명령을 실행한 모습이다.

```
======== COMPLETE========
3654 / 조만재 / C
=======================

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

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

======== ERROR ========
700
=======================
```

해당 명령어는 COMPLETE 명령어를 사용한 것으로 첫번째 명령만 성공한 모습이다. 두번째와 마지막 번호는 존재하지 않는 번호이고 세번째 번호는 존재하지만 사고의 이력이 없으므로 BST에 들어가지 못해 에러코드가 나온 것을 확인 할 수 있다.

```
======== PRINT ========
3654 / 조만재 / C
=======================

======== ERROR ========
100
=======================

======== SAVE ========
Success
=======================

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

COMPLETE LIST를 출력한 모습과 다시 LOAD를 하였지만 txt 파일이 존재하지 않고 이미 정보가 구축되어 있어 에러코드가 나온 모습이다. 세 자료구조가 모두 정보를 가지고있으므로 SAVE가 완료된 것을 볼 수 있고 메모리 해제를 하며 EXIT실행이 성공된 것을 확인 할 수 있다.

두번째 실행 명령

```
LOAD
ADD
MOVE 60
SEARCH 3654
COMPLETE 3654
COMPLETE 7053
COMPLETE 1639
COMPLETE 5703
PRINT COMPLETE_LIST
LOAD
SAVE
EXIT
```

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

첫번째 실행 명령에서 SAVE에 성공하였으므로 SAVE된 파일을 불러와 해당 동작을 성공한 모습이다.

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

이미 자료구조가 구축되어 있는 상태로 ADD의 명령에서 에러 코드가 나온 것을 확인할 수 있다.

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

MOVE 60이 실행된 모습으로 첫번째 실행에서 60명 이번 실행에서 60명으로 100을 초과하는 명령이므로 100명까지만 채우고 에러코드가 실행된 모습이다.

```
======== SEARCH========
3654 / 조만재 / C
======================

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

======== COMPLETE========
7053 / 박일영 / C
======================

======== COMPLETE========
1639 / 박영수 / C
======================

======== COMPLETE========
5703 / 전선경 / C
======================
```

Search로 인하여 해당 정보를 complete list에서 찾은 결과이다. 또한 이미 보험처리가 된 대상을 complete 실행한 결과 에러코드를 발생하며 BST에 있는 사람들을 complete한 결과 모두 성공한 모습이다.

```
======== PRINT ========
3654 / 조만재 / C
7053 / 박일영 / C
1639 / 박영수 / C
5703 / 전선경 / C
======================
```

complete된 정보들을 Print하는 명령어이다.

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

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

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

LOAD명령을 한번 더 한 결과 이미 자료구조가 구축이 되어 있어 에러코드가 나온 모습이고 두번째 실행으로 다시 구축된 자료구조를 SAVE에 성공한 모습이다. 또한 모든 메모리를 해제하는 EXIT함수가 정상적으로 실행된 모습이다.

5. Question

- 가상 함수가 무엇인가?

Derived class가 Base class를 상속할 때 Derived class의 객체는 자기 자신이 아니라 Base class에 맞춰지게 된다. 즉, 포인터가 Derived class로 맞춰진다는 뜻이다. 이 뜻은 다음과 같은 문제를 낳을 수 있다. 동적할당을 받아 각각의 클래스에서 Print를 하기 원하여 포인터로 Derived class type으로 동적할당을 받아 Derived class에서 Print를 호출하고 해당 변수를 delete 후 Base class type으로 다시 동적 할당을 받아Base class에서도 Print를 호출하게 된다면 두 함수 모두 Base class에서 호출이 되어 실행되는 것을 볼 수 있다. 이러한 문제점을 해결하기 위한 함수가 가상 함수이다. 가상함수는 함수를 정의하는데 있어 virtual이라는 키워드가 앞에 들어가게 되며 Derived class에 정의가 된다면 Base class에서 반드시 재정의를 해주어야 한다. 아래 코드의 예시를 보면 쉽게 이해할 수 있다.

```
class Animal {
public:
    void  move(void) {
        cout << "This animal moves in some way" << endl;
    }
    virtual void eat(void) {
        cout << "some animal eat food!" << endl;
    }
};


class dog : public Animal {
public:
    void eat(void) {
        cout << "dog eat apple!" << endl;
    }
};
```

```
int main()
{
    Animal * objptr = new Animal();
    objptr->eat();
    delete objptr;

    objptr = new dog();
    objptr->eat();
    return 0;
}
```

```
some animal eat food!
some animal eat food!
```

```
some animal eat food!
dog eat apple!
```

첫번째 결과 화면은 eat함수를 virtual로 선언하지 않은 결과이고 두번째 결과 화면은 virtual로 선언하여 실행 한 결과이다.

- 순수 가상 함수가 무엇인가?

순수 가상 함수란 함수의 구현이 없는 가상 함수를 뜻합니다. 아래 이미지와 같이 함수를 0으로 초기화 해주면 해당 함수는 순수 가상 함수가 됩니다.

```
class Animal {
public:
    virtual void move() = 0;
    virtual void eat() = 0;
    virtual void sleep() = 0;
};
```

이 동시에 순수 가상 함수를 포함한 클래스는 아래서 다룰 추상 클래스(Abstract Class)로 지정됩니다.

- 추상 클래스가 무엇인가?

  추상 클래스란 순수 가상 함수를 포함하고 있는 클래스입니다. 이 클래스는 인스턴스를 만들 수 없게 됩니다. 쉽게 말해서 구체적인 함수의 구현은 자식 클래스에게 모두 맡기게 되는 것입니다. 자식클래스에서 모두 재정의가 된다면 자식클래스로부터 인스턴스를 받을 수 있는 함수를 뜻합니다. 즉, 추상 클래스란 자식 클래스가 어떠한 함수를 재정의를 할지 '추상적인 형태' 만 제공하는 클래스를 뜻합니다.

- Polymorphism이 무엇인가?

  C++에서의 Polymorphism이란 어떤 클래스의 포인터 변수에 그 클래스의 하위 클래스 객체의 주소도 할당할 수 있도록 허용된다. 즉, 클래스의 포인터 변수에는 다양한 자료형의 주소를 할당할 수 있다는 말이다. 예를 들어, 도형 클래스가 부모 클래스이고 삼각형 클래스가 자식 클래스라면 Cshape* 형 변수에 CTriangle 객체의 주소를 할당할 수 있다는 뜻이므로 CShape* 형 변수에는 CShape 형뿐만 아니라 그 하위 클래스 객체의 주소도 할당할 수 있게 된다. 이로 인해서 객체를 가리키는 포인터 변수에는 여러가지 형의 객체를 가리킬 수 있고 객체의 런타임 형에 따라서 함수의 기능에 차이를 보이는 성질을 다형성이라 한다. 결론적으로 C++에서 객체들이 다형성을 갖게 하는 방법은 가상함수를 오버라이드하는 것이다. 쉽게 정리하자면 클래스를 상속받는 조건이 첫번째이고 base class에서 가상함수를 쓰는 것이 두번째 마지막으로 포인터 변수를 사용해야 한다.

```cpp
class Animal {
public:
    virtual void move() = 0;
    virtual void eat() = 0;
    virtual void sleep() = 0;
};


class dog : public Animal {
public:
    void eat() {
        cout << "dog eat apple!" << endl;
    }
    void move() {
        cout << "move!" << endl;
    }
    void sleep() {
        cout << "sleep!" << endl;
    }
};
```

```cpp
int main()
{
    Animal * objptr = new Animal();
    objptr->eat();
    delete objptr;

    objptr = new dog();
    objptr->eat();
    return 0;
}
```

이와 같이 위에서 가상함수를 설명하면서 사용하였던 코드가 다형성을 사용한 대표적인 예가 될 수 있다.

- 소멸자 앞에 virtual을 왜 붙여야 하는가?

```
#include <iostream>
using namespace std;
class classA
{
public:
    classA();
     ~classA();
};
class classB : public classA
{
public:
    classB();
    ~classB();
};
classA::classA()
{
    cout << "A" << endl;
}
classA::~classA()
{
    cout << "~A" << endl;
}
classB::classB()
{
    cout << "B" << endl;
}
classB::~classB()
{
    cout << "~B" << endl;
}
int main()
{
    cout << "TEST" << endl;
    classB *B = new classB;
    classA *A = B;
    delete A;
    return 0;
}
```

```
TEST
A
B
~A
```

```
#include <iostream>
using namespace std;
class classA
{
public:
    classA();
    virtual ~classA();
};
class classB : public classA
{
public:
    classB();
    ~classB();
};
classA::classA()
{
    cout << "A" << endl;
}
classA::~classA()
{
    cout << "~A" << endl;
}
classB::classB()
{
    cout << "B" << endl;
}
classB::~classB()
{
    cout << "~B" << endl;
}
int main()
{
    cout << "TEST" << endl;
    classB *B = new classB;
    classA *A = B;
    delete A;
    return 0;
}
```

```
TEST
A
B
~B
~A
```

앞서 가상함수를 설명하면서 봤던 것처럼 상속을 받고 자식클래스의 함수가 가상함수로 재정의가 되지 않고 부모 클래스이 포인터로부터 자식클래스를 호출하게 된다면 부모클래스의 함수만 실행되는 것을 확인 할 수 있었습니다. 이러한 문제점으로 자식 소멸자에 virtual을 붙여주지 않는다면 왼쪽그림과 같이 자식 클래스의 소멸자가 실행되지 않는 모습을 볼 수 있습니다. 오른쪽은 virtual을 붙여 주었을 경우입니다.

6. Consideration

해당과제의 가장 큰 목적은 세가지의 자료구조를 구축하는데 있었습니다. 우선 단일 linked list는 1학년때부터 다루던 자료구조 이므로 구축하며 몇 가지 명령어를 실행하는데 큰 어려움이 없었습니다. 문제는 BST에 관련된 구축과 명령어였습니다. 프로젝트 전 미리 과제로 제출한 homework에서 많은 도움을 받았으며 거의 대부분의 알고리즘을 미리 했던 homework에서 사용하며 insert나 search함수를 쉽게 해결할 수 있었습니다. 가장 난해했던 문제로는 Number BST에 포함되어 있는 각각의 노드 안에 Car BST가 포함이 되어 있는 형태입니다. 처음 프로젝트를 봤을 때는 바로 구축 되어있는 상태가 묘사되지 않았지만 프로그램을 작성하면서 그에 대한 개념을 좀더 잡을 수 있던 계기가 되었습니다. 특히 지금까지 써왔던 linked list와는 달리 Search에서 빠르다는 장점이 있다는 것을 알게 되었습니다. 이유는 자료를 계속 비교하면서 정보의 위치를 왼쪽 오른쪽으로 결정해주며 비교대상이 될 노드를 계속 줄일 수 있기 때문에 완전형 트리를 가정한다면 순차적으로 node를 계속 방문하여 비교하는 linked list에 비해 평균적으로 search의 속도가 빠른 것을 알 수 있었습니다. 특히 이 효과는 자료가 커질수록 더 효과적일 것이라고 예상합니다. 다음으로는 queue를 구축하였습니다. 구축자체는 linked list와 다르지 않지만 삭제를 하는데 있어 먼저 들어온 것을 지워주어야 하기 때문에 pop함수를 실행하는데 있어 조금 신경 썼던 자료 구조입니다. 특히 queue나 stack같은 자료구조는 BST의 순회함수를 설계할 때도 사용함으로써 이에 대한 응용은 굉장히 많을 것이라고 예측됩니다.