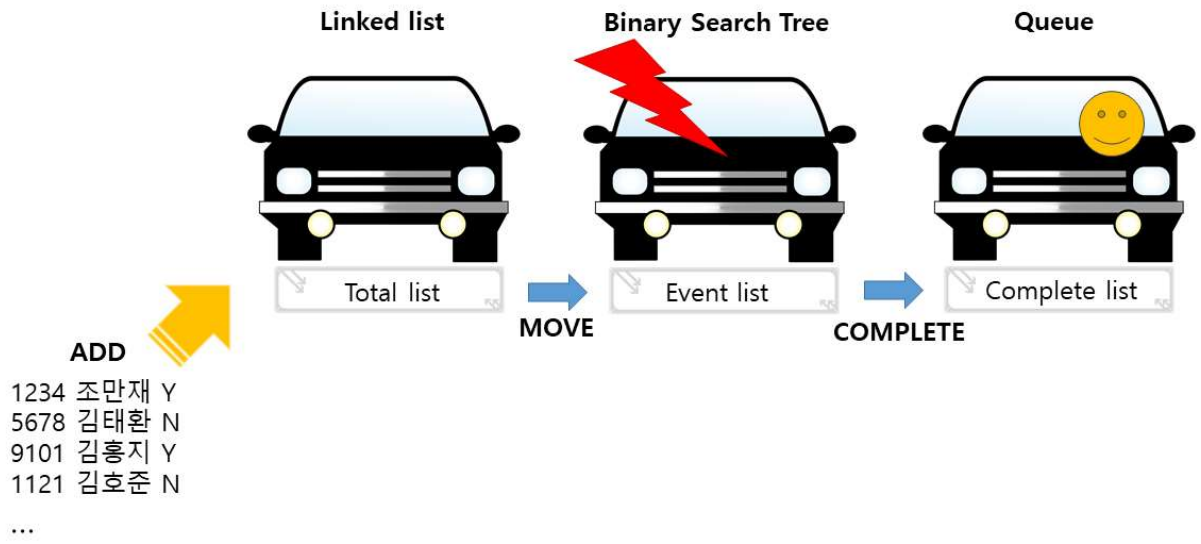
Data Structure Lab. Project #1

2018년 9월 5일

Due date: 2018년 10월 3일 수요일 23:59:59 까지

본 프로젝트에서는 이진 탐색 트리(Binary Search Tree, BST)와 연결 리스트(Linked List), Queue를 이용하여 차량 보험 관리 프로그램을 구현한다. 이 프로그램은 차량 보험 데이터(차량 번호, 차주, 사고유무)를 입력받아 TOTAL_LIST, EVENT_LIST, COMPLETE_LIST으로 구분하여 구축한다. TOTAL_LIST은 Linked List로, EVENT_LIST은 BST로, COMPLETE_LIST은 Queue로 구축한다.



□ Program implementation

프로그램에는 TOTAL_LIST, EVENT_LIST, COMPLETE_LIST가 존재하며 각각 Linked List, BST, Queue로 구현되어 있다. 그리고 각 자료구조의 단어 노드에는 차량번호, 차주와 사고유무가 데이터로 존재한다.

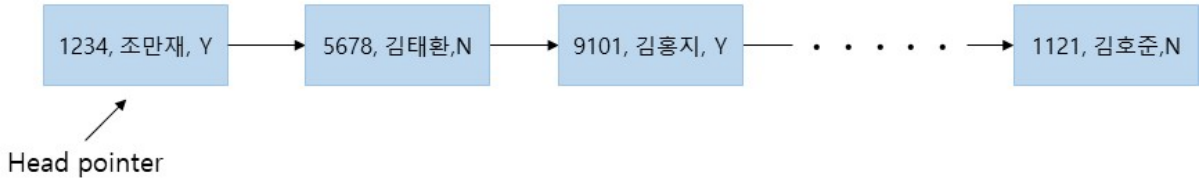


그림 2. TOTAL_LIST Linked List의 예

TOTAL_LIST의 Linked List는 기본적으로 새로운 데이터가 들어오면 Insert 동작을 하고 EVENT_LIST으로 데이터를 옮길 때 delete 동작을 한다. 프로그램에 데이터를 추가하게 되면 우선 TOTAL_LIST에 데이터가 모두 들어가게 되며 그 후에 MOVE 명령어를 통해 TOTAL_LIST의 데이터를 EVENT_LIST으로 옮길 수 있다. EVENT_LIST에는 최대 100개의 데이터만 존재해야 한다.

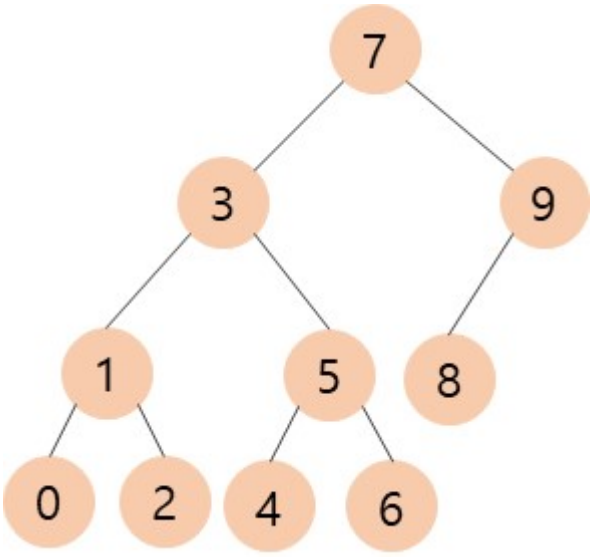


그림 3. EVENT_LIST NumberBST의 예

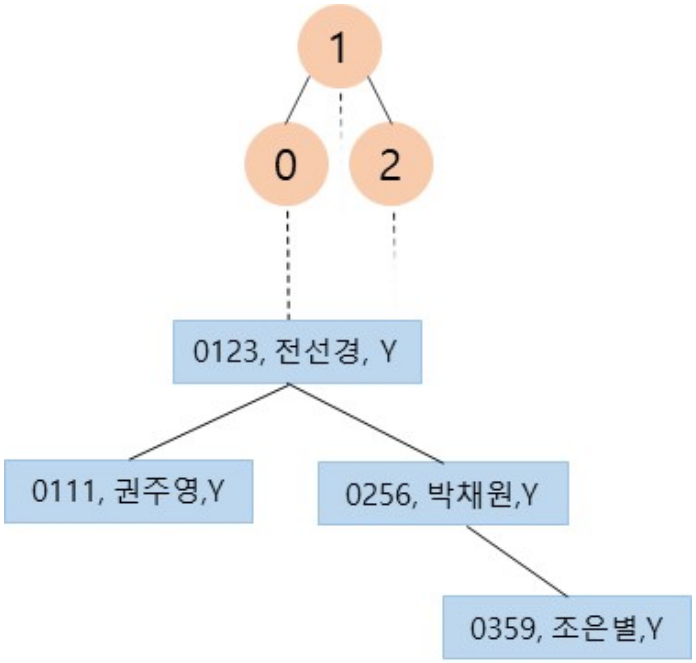


그림 4. EVENT_LIST CarBST의 예

EVENT_LIST에는 번호 BST와 차량 정보 BST가 존재해야 한다. 번호 BST는 그림 3과 같이 숫자 노드로 이루어져있으며 반드시 7, 3, 9, 1, 0, 2, 5, 4, 6, 8 순으로 숫자를 추가하여 위와 같은 번호 BST를 구축한다. 번호 BST는 프로그램이 시작되면 바로 구축한다. 각각의 숫자 노드는 그림 4와 같은 차량 정보 BST를 가지고 있으며 MOVE 명령어를 통해 TOTAL_LIST로부터 사고유무가 Y인 차량 노드 n개를 입력받아 차량 번호 BST를 구축할 수 있다. 여기서 EVENT_LIST에 존재하는 데이터 노드는 최대 100개까지만 존재할 수 있음을 명심한다. COMPLETE 명령어를 통해 현재 사고 차량의 보험 처리가 완료된 것으로 설정하면, EVENT_LIST의 해당 데이터를 COMPLETE_LIST로 옮긴다. 그리고 해당 차량의 사고유무를 Y->C 로 수정한다.

EVENT_LIST에 사용되는 BST를 연결하는 규칙은 다음과 같다.

※ BST 연결 규칙

① 부모 노드보다 차량 번호의 사전적 순서가 작은 노드는 왼쪽, 큰 노드는 오른쪽 서브 트리에 반드시 위치한다.

② 노드를 제거할 때, **양쪽 자식 노드가 모두 존재할 경우에는 왼쪽 자식 노드 중 가장 큰 노드를 제거되는 노드 위치로 이동**시키며, 왼쪽 자식노드가 존재하지 않을 경우에는 오른쪽 자식 노드 중 가장 작은 노드를 제거되는 노드 위치로 이동시킨다.

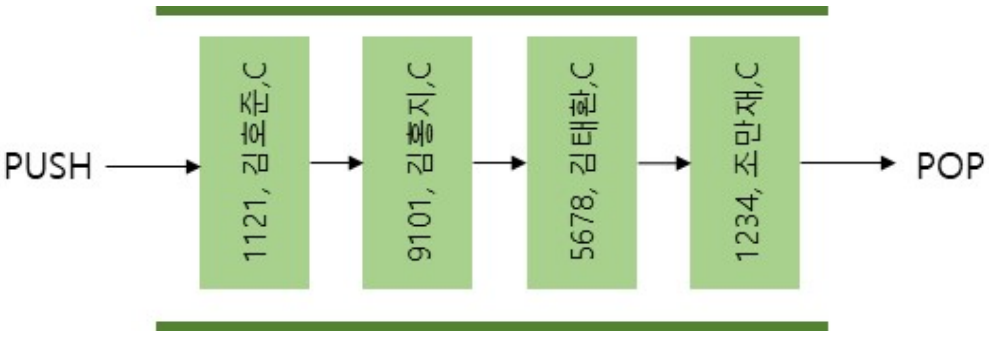


그림 5. COMPLETE_LIST의 Queue 예

COMPLETE_LIST의 Queue는 그림5와 같이 구성된다. 처음 데이터가 들어오면 Push 동작을 하고 최종적으로 처리완료를 할 때, Queue가 빌 때까지 Pop 동작을 하며 출력한다.

□ Functional Requirements

표 1. 명령어 사용 예 및 기능 설명

명령어	명령어 사용 예 및 기능
LOAD	사용 예) LOAD 기존의 데이터 정보를 불러오는 명령어로, 텍스트 파일에 데이터 정보가 존재할 경우 텍스트 파일을 읽어 TOTAL_LIST Linked List, EVENT_LIST BST, COMPLETE_LIST Queue 자료구조에 이전과 동일한 연결 순서를 가지도록 저장한다. 만약 3개의 텍스트 파일이 존재하지 않거나 자료구조에 이미 데이터가 들어가 있을 경우에 알맞은 에러 코드를 출력한다. 사용되는 텍스트 파일의 이름은 아래와 같으며 파일 이름을 수정하지 않는다. TOTAL_LIST 텍스트 파일: total_list_car.txt EVENT_LIST 텍스트 파일: event_list_car.txt COMPLETE_LIST 텍스트 파일: complete_list_car.txt
ADD	사용 예) ADD 데이터 텍스트 파일에 있는 데이터 정보를 읽어오는 명령어로, 데이터 텍스트 파일에 존재하는 단어들을 TOTAL_LIST에 모두 저장한다. 데이터 텍스트 파일이 존재하지 않거나 파일에 정보가 존재하지 않을 경우 알맞은 에러 코드를 출력한다. 사용되는 데이터 텍스트 파일의 이름은 아래와 같으며, 파일의 이름은 절대 수정하지 않는다. 단어 텍스트 파일: car.txt
MOVE	사용 예) MOVE 10 사용자가 입력한 수만큼 TOTAL_LIST의 단어들을 EVENT_LIST으로 옮기는 명령어로, 1~100 사이의 정수를 입력받고 입력받은 수와 EVENT_LIST 데이터 수의 합이 100을 넘어가지 않도록 한다. 이때 옮겨지는 데이터는 사고유무가 Y인 데이터들 이다(순차적으로 Y인 것들만 넘김). TOTAL_LIST에 데이터가 없는 경우, 입력받은 수와 EVENT_LIST의 데이터 수의 합이 100을 넘어가게 되는 경우, 사용자가 입력한 수만큼 데이터가 존재하지 않을 경우 등이 발생하면 에

	러 코드를 출력한다.
SAVE	사용 예) SAVE 현재 단어장 정보를 저장하는 명령어로, 각각 Linked List, BST, Queue에 저장된 단어들을 TOTAL_LIST, EVENT_LIST, COMPLETE_LIST 텍스트 파일에 저장한다. 이때 EVENT_LIST 텍스트 파일에 데이터를 차량번호순으로 저장하면 LOAD할 때 BST의 Child Node가 한쪽으로 쏠려 Skewed Binary Tree가 되므로 주의하도록 한다(Hint: pre-order 사용). 사용되는 텍스트 파일의 이름은 LOAD 명령어와 동일하며 데이터 정보가 존재하지 않을 경우 알맞은 에러 코드를 출력한다.
SEARCH	사용 예) SEARCH 1234 데이터의 정보를 찾아 출력하는 명령어로, TOTAL_LIST, EVENT_LIST, COMPLETE_LIST에 입력한 데이터가 존재할 경우, 차량번호, 차주, 사고유무를 출력해준다. 만약 입력한 데이터가 없을 경우 알맞은 에러 코드를 출력한다.
PRINT	사용 예) PRINT TOTAL_LIST 사용 예) PRINT EVENT_LIST I_PRE 입력한 리스트에 있는 데이터들을 출력하는 명령어로, TOTAL_LIST를 입력할 경우 TOTAL_LIST Linked List에 있는 단어들을 Header부터 순서대로 전부 출력한다. EVENT_LIST의 경우 추가로 “R_PRE”, “I_PRE”, “R_IN”, “I_IN”, “R_POST”, “I_POST”, “I_LEVEL”을 입력받아 아래의 의미를 참조하여 각각의 트리순회(Tree Traversal) 방법에 따른 알맞은 순서대로 데이터들을 전부 출력한다. COMPLETE_LIST를 입력할 경우 COMPLETE_LIST Queue의 처리 완료된 데이터들을 순서대로 전부 출력한다. 만약 입력한 데이터 정보가 존재하지 않을 경우 알맞은 에러 코드를 출력한다. R_PRE: Recursive pre-order(재귀함수를 이용한 pre-order) I_PRE: Iterative pre-order(반복문을 이용한 pre-order, 재귀함수 사용금지) R_IN: Recursive in-order(재귀함수를 이용한 in-order) I_IN: Iterative in-order(반복문을 이용한 in-order, 재귀함수 사용금지) R_POST: Recursive post-order(재귀함수를 이용한 post-order) I_POST: Iterative post-order(반복문을 이용한 post-order, 재귀함수 사용금지) I_LEVEL: Iterative level-order(반복문을 이용한 level-order)
COMPLETE	사용 예) COMPLETE 1234 차량 번호의 사고 상태를 처리하는 명령어로 EVENT_LIST에 입력한 차량 번호가 존재할 경우 차량 번호의 사고 상태를 보험 처리 완료 상태(COMPLETE_LIST로 옮긴 후, 사고유무를 ‘C’로 변경) 로 변경한 뒤 결과를

	출력해준다. 만약 차량 번호가 존재하지 않거나 차량 정보가 존재하지 않을 경우 알맞은 에러 코드를 출력한다.
EXIT	사용 예) EXIT 프로그램 상의 메모리를 해제하며, 프로그램을 종료한다.

□ Requirements in implementation

- ✓ 모든 명령어는 command.txt에 저장하여 순차적으로 읽고 처리한다.
- ✓ 모든 명령어는 반드시 대문자로 입력한다.
- ✓ 명령어에 인자(Parameter)가 모자라거나 필요 이상으로 입력받을 경우 에러 코드를 출력한다.
- ✓ 번호 노드 클래스(NumberNode Class)는 반드시 차량 번호, 차주와 사고유무를 저장한다.
- ✓ 차량 관리 프로그램에는 중복된 차량 번호가 존재하지 않아야한다.
  - ADD할 때 중복된 차량 번호가 존재할 경우 에러를 발생시키지 않고 중복된 차량 번호를 제외한 나머지 단어들을 TOTAL_LIST에 저장한다.
- ✓ EVENT_LIST에는 100개 이하의 차량 정보가 존재해야한다.
- ✓ 예외처리에 대해 반드시 에러 코드를 출력한다.
- ✓ 출력은 “출력 포맷”을 반드시 따라한다.
- ✓ log.txt 파일에 출력 결과를 반드시 저장한다.
  - log.txt가 이미 존재할 경우 텍스트 파일 가장 뒤에 이어서 추가로 저장한다.
- ✓ 읽어야할 텍스트 파일이 존재하지 않을 경우 해당 텍스트 파일을 생성한 뒤 진행하도록 한다.
- ✓ 프로그램 종료 시 메모리를 모두 해제할 수 있도록 Queue, CarBST, NumberBST, LinkedList Class의 소멸자를 반드시 작성한다.
- ✓ PRINT의 Level-order 기능은 위에서부터 level 순으로, 왼쪽에서부터 오른쪽으로 순서대로 출력한다.

□ 동작별 에러 코드

동작	에러 코드
LOAD	100
ADD	200
MOVE	300
SAVE	400
SEARCH	500
PRINT	600
COMPLETE	700
EXIT	0

□ 출력 포맷

기능	출력 포맷	설명
에러(ERROR)	======== ERROR ======== 100 ======================	에러 발생 시 출력 포맷에 맞춰 에러 코드를 출력
LOAD	======== LOAD ======== Success ======================	차량 번호 정보를 읽어오는데 성공 시 출력 포맷에 맞춰 결과를 출력 텍스트 파일 정보가 없을 시 에러 출력
ADD	======== ADD ======== Success ======================	차량 번호 추가 성공 시 출력 포맷에 맞춰 결과를 출력 텍스트 파일이 존재하지 않거나 텍스트 파일에 데이터가 없을 경우 에러 출력
MOVE	======== MOVE ======== Success ======================	차량 번호 이동 성공 시 출력 포맷에 맞춰 결과를 출력 TOTAL_LIST에 데이터가 없거나 EVENT_LIST에 데이터 수가 100을 초과하게 될 경우 에러 출력

SAVE	======== SAVE ======== Success ======================	차량 번호 정보 저장 성공 시 출력 포맷에 맞춰 결과를 출력 저장할 차량 번호 정보가 없을 시 에러 출력
SEARCH	======== SEARCH ======== 1234 조만재 Y ========================	차량 번호 탐색에 성공 시 출력 포맷에 맞춰 결과를 출력 차량 번호가 없을 경우 에러 출력
PRINT	======== PRINT ======== 1234 조만재 Y 5678 김태환 N 9101 김홍지 Y ... ======================	해당 목록 정보가 존재할 시 출력 포맷에 맞춰 결과를 출력 해당 목록 정보가 없을 경우 에러 출력
COMPLETE	======== COMPLETE ======== 1234 조만재 C ========================	처리 완료할 차량이 존재할 경우 출력 포맷에 맞춰 결과를 출력 처리 완료할 차량이 존재하지 않을 경우 에러 출력

□ 동작 예시

command.txt

```
LOAD
PRINT COMPLETE_LIST
ADD
MOVE 20
SEARCH 3654
PRINT EVENT_LIST R_IN
COMPLETE 3654
PRINT COMPLETE_LIST
LOAD
SAVE
EXIT
```

실행 결과 화면 및 log.txt

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

```
=======ERROR=======
600
====================

=======ADD=========
Success
====================

=======MOVE========
Success
====================

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

======= PRINT =======
1639 박영수 Y
2317 박미경 Y
2763 전보은 Y
2980 조정연 Y
3041 이지은 Y
3654 조만재 Y
4301 김은지 Y
4623 박지현 Y
5703 전선경 Y
6128 정상훈 Y
6873 김태환 Y
7435 김동영 Y
8014 남궁민 Y
8126 차성우 Y
8605 정현우 Y
8695 신희민 Y
9138 남윤창 Y
9178 서예지 Y
9254 한준희 Y
9705 조은별 Y
====================

======= PRINT =======
3654 조만재 C
====================
=======ERROR=======
```

```
100
==================

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

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

□ 구현 시 반드시 정의해야하는 Class

- ✓ NumberNode - 번호 노드 클래스
- ✓ NumberBST - 번호 BST 클래스
- ✓ CarNode - 차량 번호 노드 클래스
- ✓ CarBST - 차량 정보 BST 클래스
- ✓ LinkedList - 연결 리스트 클래스(Linked List)
- ✓ Queue - 큐 클래스(Queue)
- ✓ Manager - Manager 클래스
  - 다른 클래스들의 동작을 관리하여 프로그램을 전체적으로 조정하는 역할을 수행

□ Files

- ✓ car.txt : 프로그램에 추가할 차량 정보들이 저장되어 있는 파일
- ✓ command.txt : 프로그램을 동작시키는 명령어들을 저장하고 있는 파일
- ✓ total_list_car.txt : TOTAL_LIST의 단어들을 저장하고 있는 파일
- ✓ event_list_car.txt : EVENT_LIST의 단어들을 저장하고 있는 파일
- ✓ complete_list_car.txt : COMPLETLE_LIST의 단어들을 저장하고 있는 파일
- ✓ log.txt : 프로그램 출력 결과를 모두 저장하고 있는 파일

□ 제한사항 및 구현 시 유의사항

- ✓ 반드시 제공되는 압축파일(project1.tar.gz)를 이용하여 구현하며 작성된 소스 파일의 이름과 클래스와 함수 이름 및 형태를 절대 임의로 변경하지 않는다.
- ✓ 클래스의 함수 및 변수는 자유롭게 추가 구현이 가능하다.
- ✓ 제시된 Class를 각 기능에 알맞게 모두 사용한다.
- ✓ 프로그램 구조에 대한 디자인이 최대한 간결하도록 고려하여 설계한다.
- ✓ 채점 시 코드를 수정해야 하는 일이 없도록 한다.
- ✓ 주석은 반드시 영어로 작성한다. (한글로 작성하거나 없으면 감점)
- ✓ 프로그램은 반드시 리눅스(Ubuntu 16.04)에서 동작해야한다. (컴파일 에러 발생 시 감점)
  - 제공되는 Makefile을 사용하여 테스트 하도록 한다.

□ 보고서 작성

(필수 추가작성 항목)
아래 설명하고 소스코드 예를 보일 것
✓ 가상 함수가 무엇인가?/ 왜 필요한가?
✓ 순수 가상 함수가 무엇인가?
✓ 추상 클래스가 무엇인가?
✓ Polymorphism이 무엇인가?
✓ 소멸자 앞에 virtual을 왜 붙여야 하는가?

□ 제출기한 및 제출방법

✓ 제출기한
- 2018년 10월 3일 수요일 23:59:59 까지 제출

✓ 제출방법
- 소스코드(Makefile과 텍스트 파일 제외)와 보고서 파일(pdf)을 함께 압축하여 제출
  • 확장자가 .cpp, .h, .pdf가 아닌 파일은 제출하지 않음
  • 보고서 파일 확장자가 pdf가 아닐 시 감점
- 제출 사이트는 추후 공지

✓ 제출 형식
- 학번_DS_project1.tar.gz

✓ 보고서 작성 형식 및 제출 방법
- Introduction : 프로젝트 내용에 대한 설명
- Flowchart : 설계한 프로젝트의 플로우차트를 그리고 설명
- Algorithm : 프로젝트에서 사용한 알고리즘의 동작을 설명
- Result Screen : 모든 명령어에 대해 결과화면을 캡처하고 동작을 설명
- Question : 5개의 추가 질문에 대한 소스코드 예와 개념 설명
- Consideration : 고찰 작성
  • 위의 각 항목을 모두 포함하여 작성
  • 보고서 내용은 한글로 작성
  • 보고서에는 소스코드를 포함하지 않음

- 하드카피는 제출하지 않음