데이터구조설계

DS_Project1 보고서

이름: 양서은

학번: 2022202061

학과: 컴퓨터정보공학부

교수님: 최상호 교수님

1. Introduction

이진 탐색 트리, 연결 리스트, 큐 자료구조를 사용하여 개인정보를 관리하는 프로그램을 구현한다. 개인정보엔 회원의 이름, 나이, 개인정보 수집 날짜, 가입 약관이 있다. 수집된 개인정보를 토대로 자료구조를 구축하여 개인정보 추가 및 삭제, 탐색 기능을 구현한다.
가입 약관엔 A, B, C, D가 있으며 개인정보 보관 유효기간을 나타낸다. A부터 각각 6개월, 1년, 2년, 3년을 나타내며 개인정보 수집 날짜로부터 약관 유효기간을 더해 개인정보 만료 일자를 계산한다. 개인정보 관리 프로그램엔 텍스트 파일에 저장되어 있는 개인정보를 저장한 MemberQueue, 이름을 기준으로 정렬된 NameBST, 가입 약관과 마감 날짜로 정렬된 TermsList/TermsBST가 있다.

1) MemberQueue

텍스트 파일로부터 개인정보를 불러와 자료구조 Queue에 저장한다. 일반적인 queue와 동일하게 동작하지만 저장하는 정보의 개수가 이름, 나이, 개인정보 수집 날짜, 약관, 총 4개가 묶음으로 한 MemberQueueNode에 저장된다. LOAD 명령과 ADD 명령을 통해 MemberQueue에 저장하고 QPOP 명령이 실행되었을 때 Node가 한 개씩 queue에서 빠져나오면서 NameBST와 TermsList, TermsBST에 저장된다.

2) NameBST

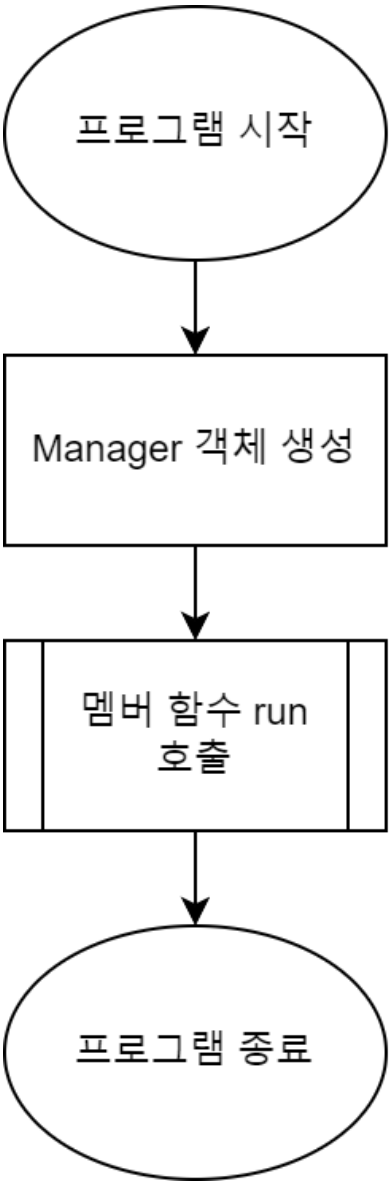
Tree 구조로 이름을 기준으로 정렬된다. 각 Node는 최대 2개의 subtree를 가질 수 있으며 각 Node를 기준으로 왼쪽엔 Node보다 사전적으로 먼저 오는 이름들이, 오른쪽으로는 Node보다 사전적으로 뒤에 오는 이름들이 저장된다. NameBST의 Node는 회원의 이름, 나이, 개인정보수집일자, 만료일자, 가입 약관을 저장한다. 2개의 자식 Node가 존재하는 Node가 삭제될 경우, 오른쪽 자식 Node에서 가장 작은 Node가 삭제되는 Node 위치로 이동한다.

3) TermsList/TermsBST

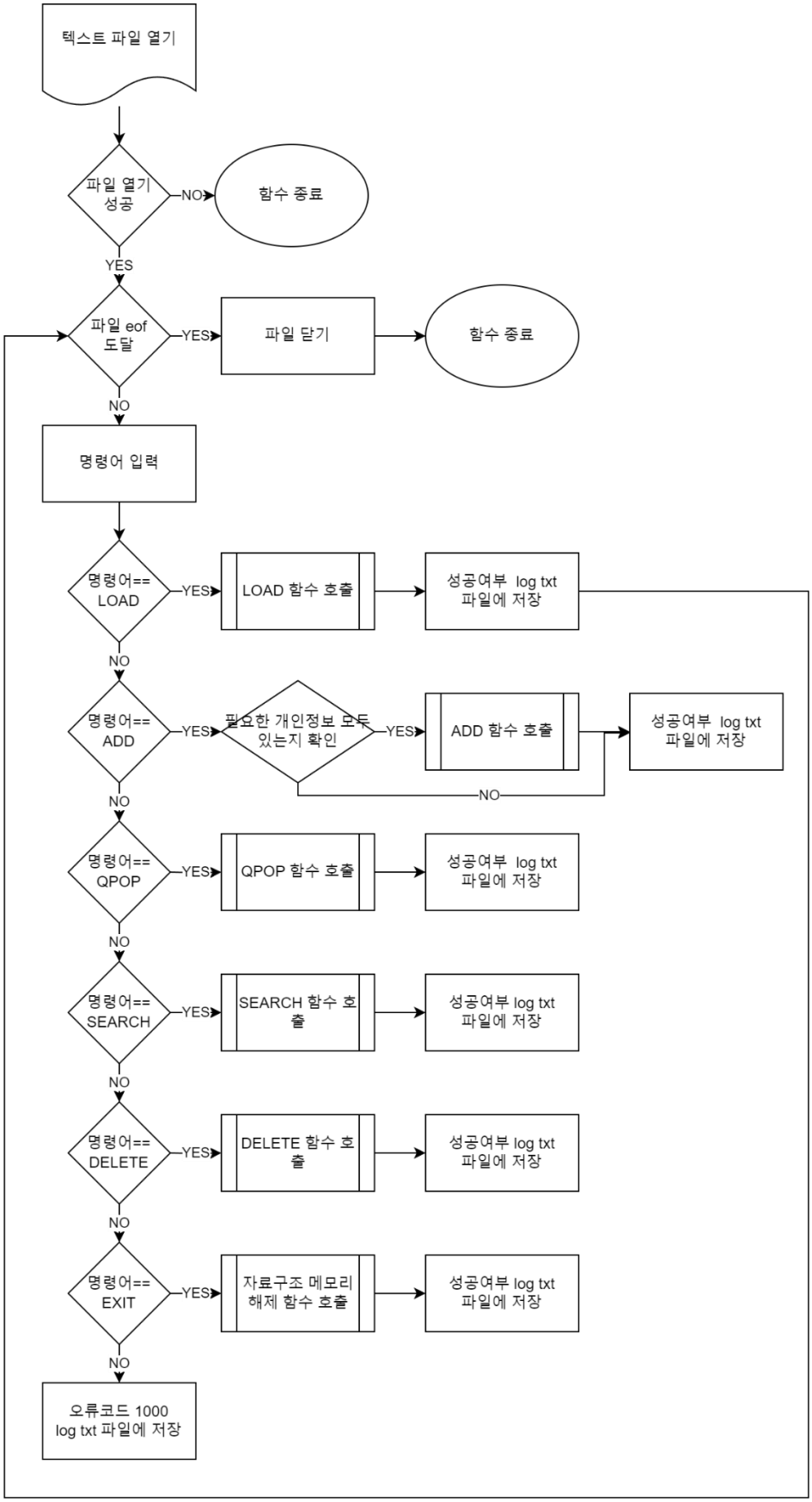
QPOP을 통해 방출되는 개인정보를 토대로 생성된다. 가입 약관을 기준으로 TermsListNode가 생성되는데 TermsList에 저장되어 있는 가입 약관이 아니라면 TermsListNode가 생성되고 그에 따라 TermsBST가 생성되고 개인정보가 저장된다. 만약 이미 TermsList에 저장되어 있던 가입약관이라면 해당 가입약관으로 가입한 회원수를 증가시키고 생성되어 있던 TermsBST에 새로운 개인정보를 추가한다.
각 TermsListNode엔 대응되는 TermsBST가 존재하며 해당 BST엔 같은 가입 약관을 가지는 개인정보가 저장되어있다. TermsBST는 개인정보 만료 일자를 기준으로 정렬된다. 즉 각 Node의 왼쪽으론 해당 Node보다 만료일자가 빠른 개인정보가, 오른쪽으론 해당 Node보다 만료일자가 느린 개인정보가 저장된다.

2. Flowchart

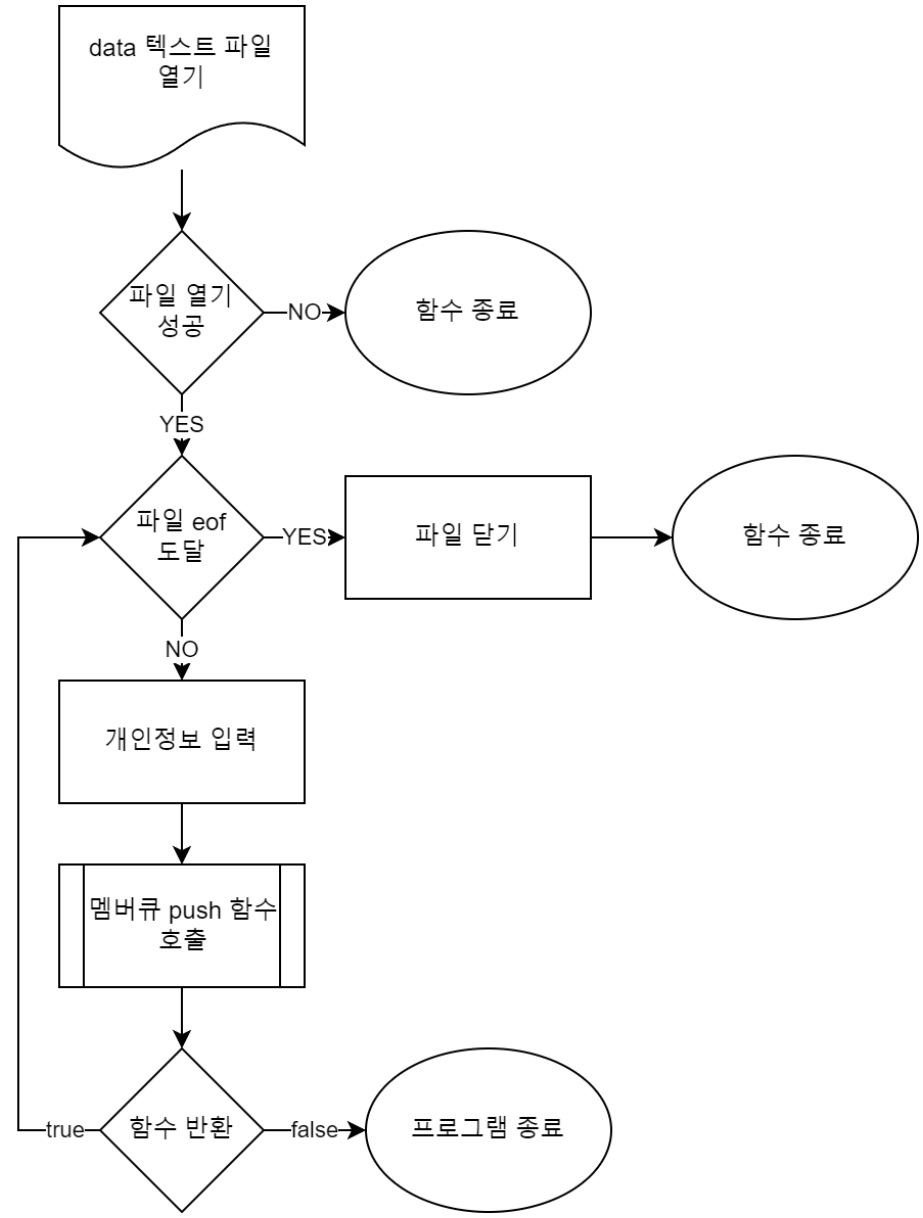
main function



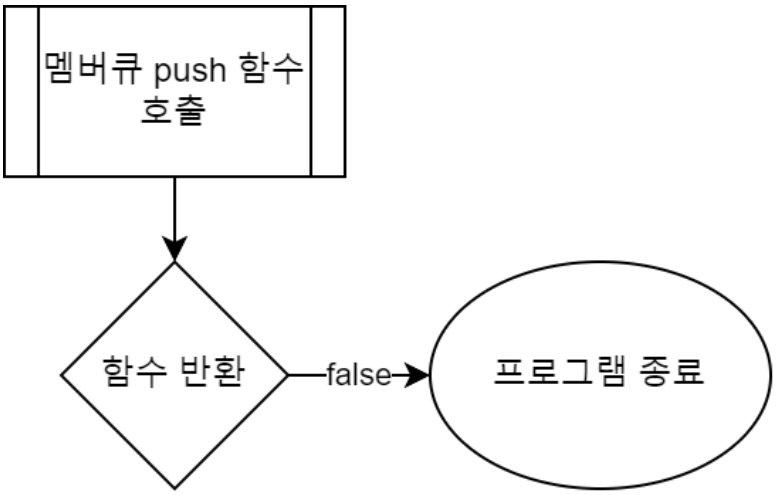
run function



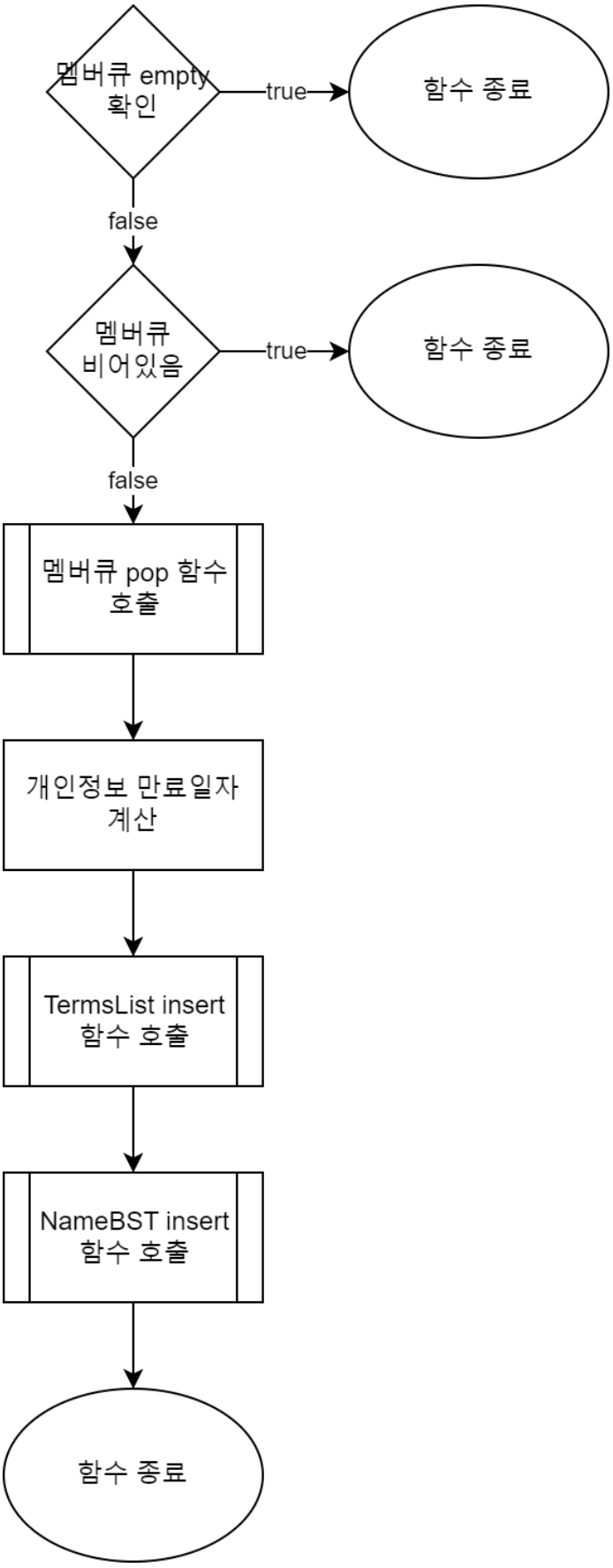
LOAD function



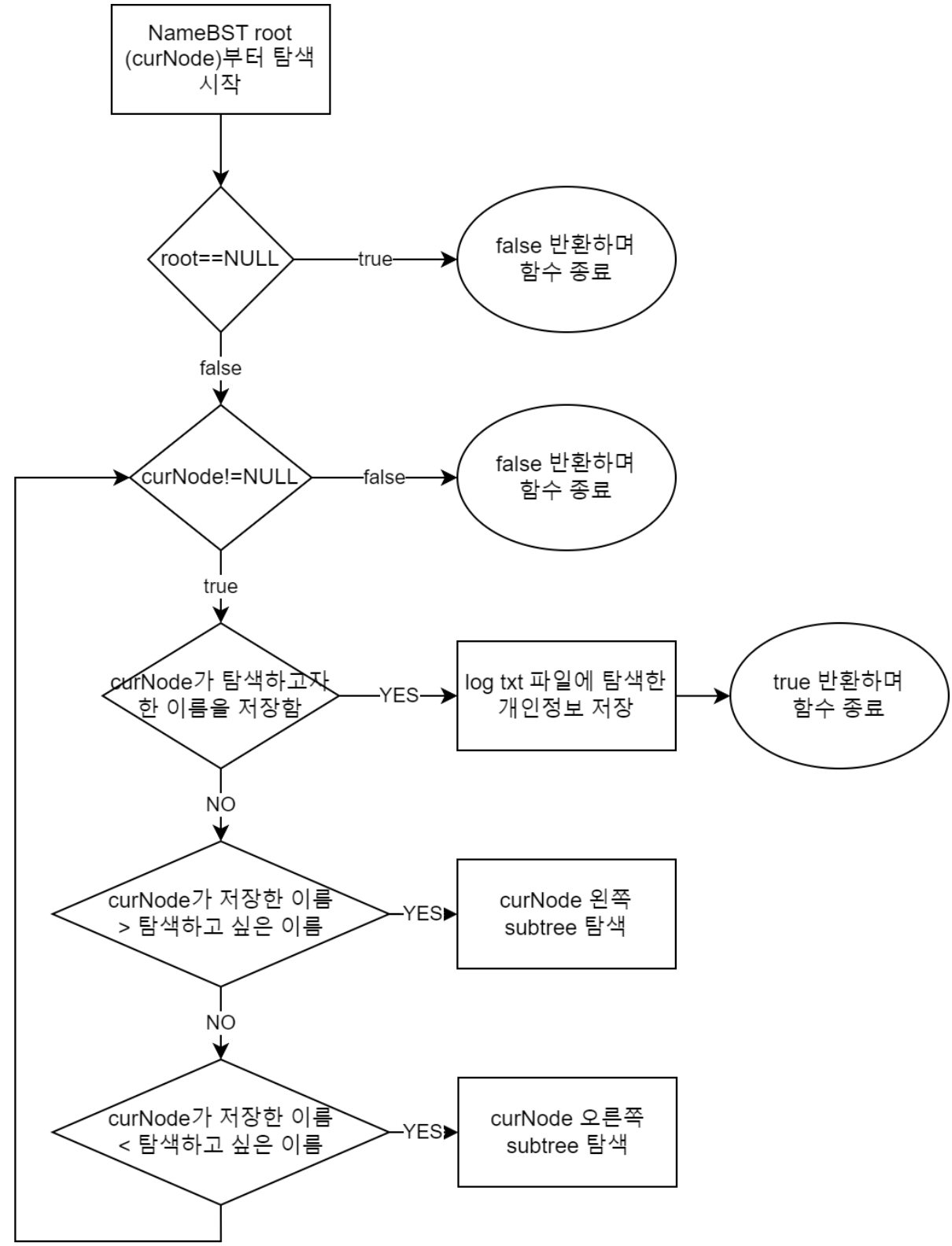
ADD function



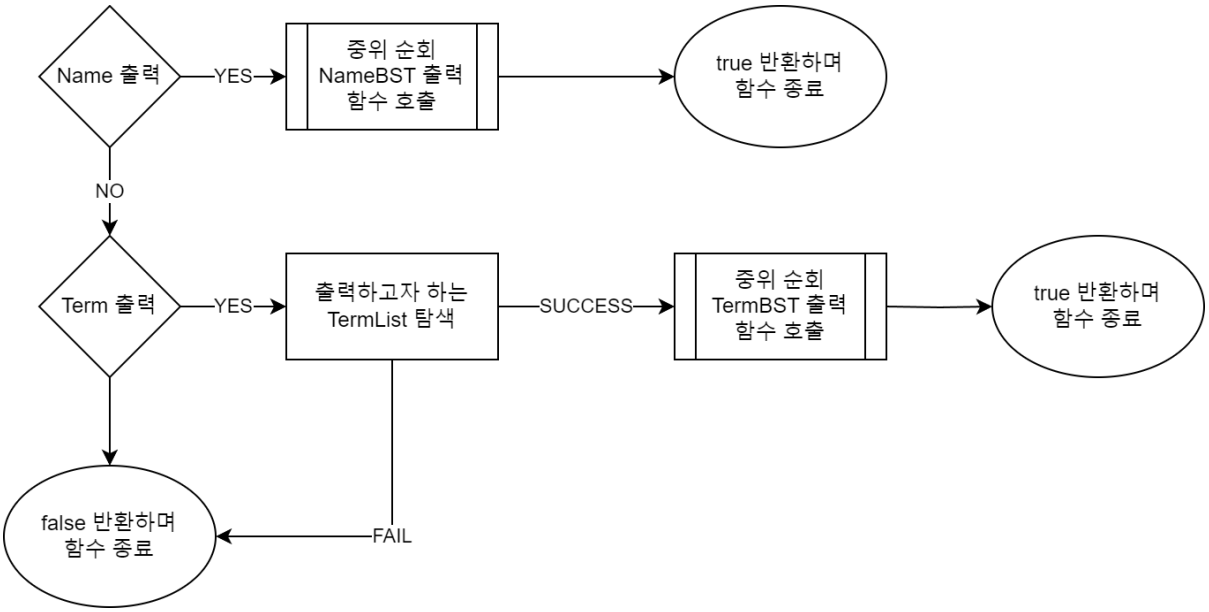
QPOP function



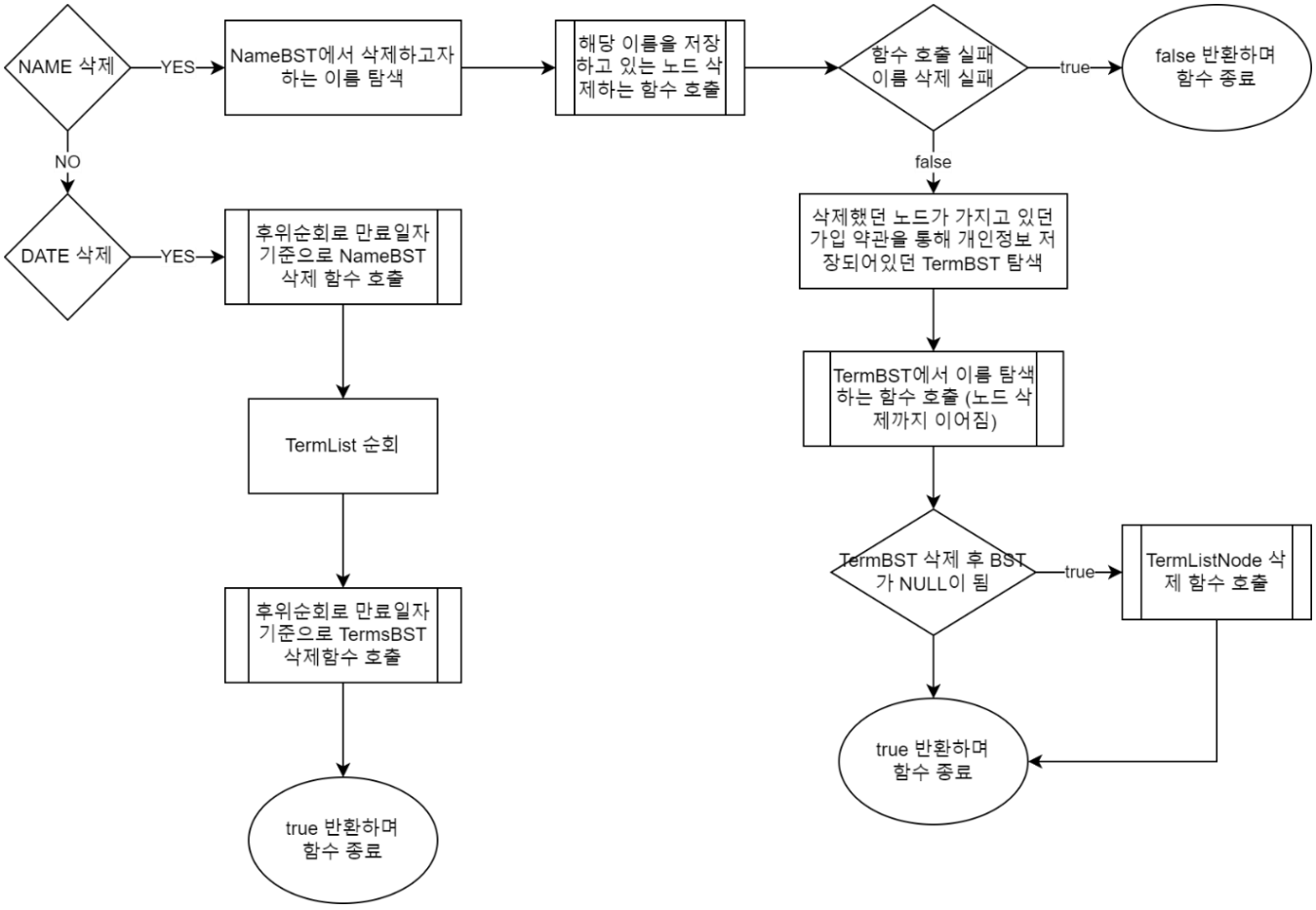
SEARCH function



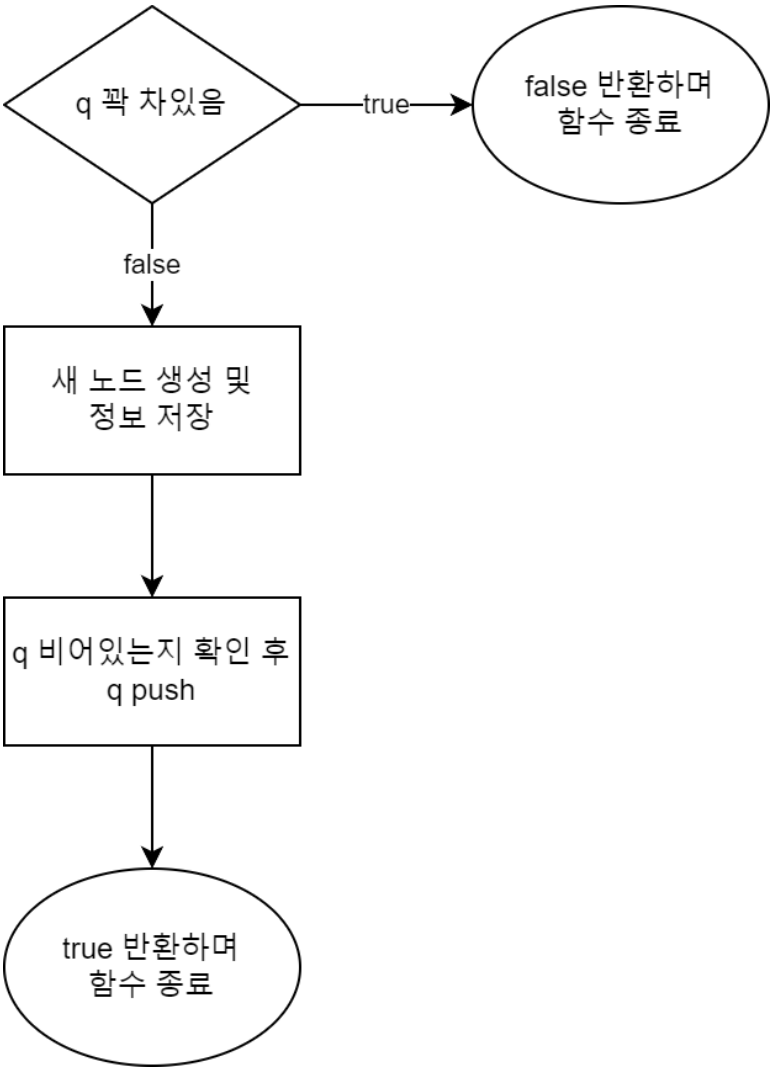
PRINT function



DELETE function



MemberQueue push function



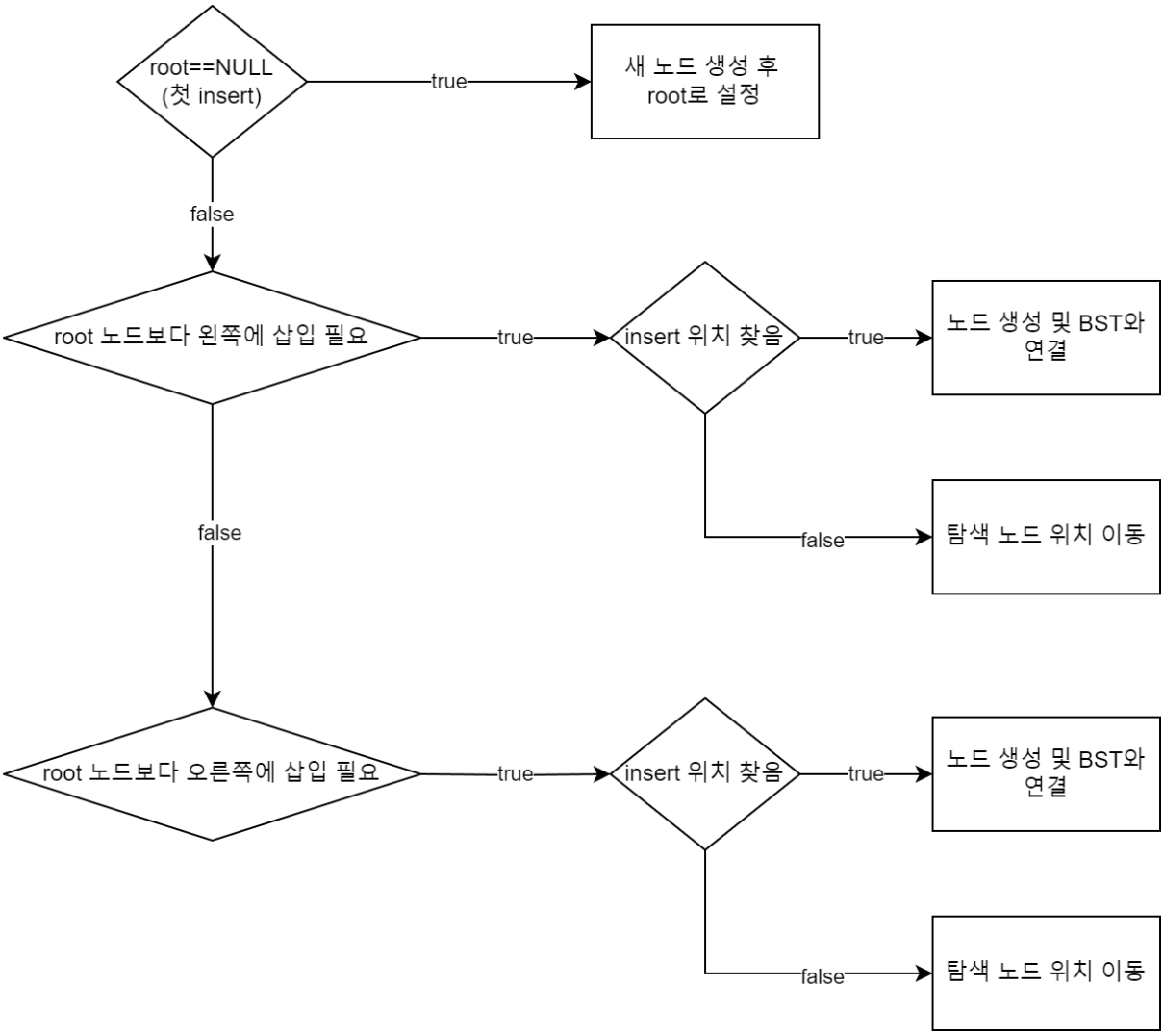
- Push function

  Queue의 사이즈를 확인한 후 push가 가능할 때만 node insert를 진행한다.

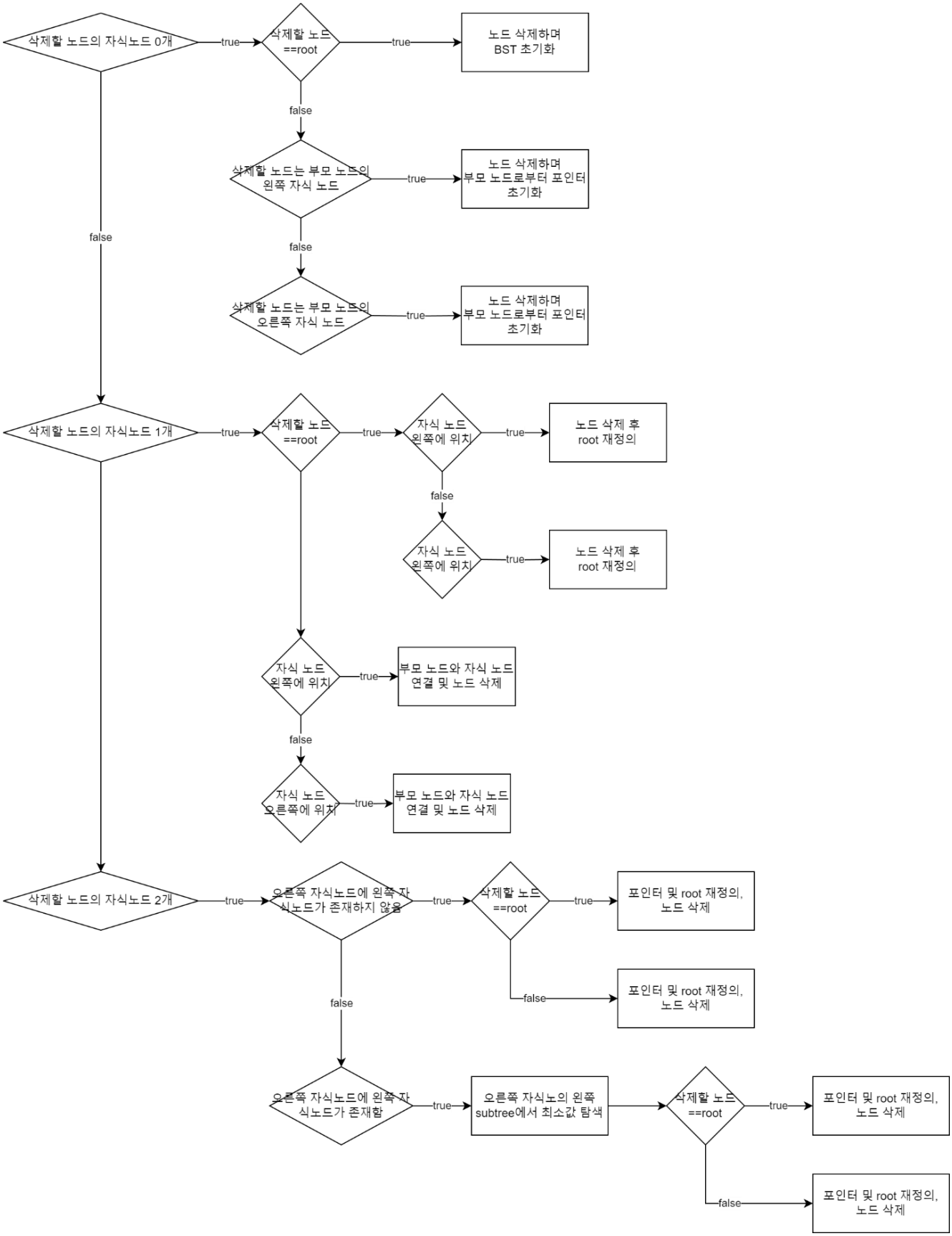
- Pop function

  rear부터 차례대로 node를 방출하며 마지막 node를 방출할 때 front와 rear을 초기화하여 오류를 방지한다.

BST insert function



BST delete function



3. Algorithm

- MemberQueue

  1) Push

  일반적인 자료구조 queue와 동일한 형태로 동작한다. 하지만 데이터를 여러 개를 묶어서 한 노드에 저장한다는 차이점이 있다. 또한 저장할 수 있는 최대 개인정보가 100개로 한정되어 있다. 따라서 push를 할 때 MemeberQueue가 비어있는지, 꽉 차 있는지 확인한 후 push를 수행해야 된다.
  만약 MemberQueue에 개인정보 100개가 저장되어 있다면, 최대 개수를 채웠으므로 더 이상의 push가 불가능하다. 따라서 false를 반환하며 함수를 종료한다.
  push를 할 수 있는 상황이라면, queue에 처음으로 들어가는 노드인지 아닌지 판별해야 한다. 자료구조 queue는 FIFO(First In First Out)으로 제일 먼저 삽입된 노드를 가리키는 front와 마지막에 삽입된 노드를 가리키는 rear가 있다. 첫 push일 경우, front와 rear가 push된 노드를 가리킨다. 그 다음 push부턴 front만 push되는 노드를 가리키게 하여 MemberQueue를 업데이트한다.

  2) Pop

  제일 먼저 삽입된 front가 가리키는 노드부터 차례대로 방출이 일어난다. 만약 Queue에 저장되어 있는 마지막 노드를 삭제하게 된다면 front와 rear이 같은 노드를 가리키고 있는 상태이다. 이 때 노드를 삭제하고 front와 rear을 모두 NULL로 초기화 시켜줘야 한다. 그렇지 않으면 모든 노드를 삭제한 후 다시 queue를 불러왔을 때 쓰레기 값을 가리키고 있기 때문에 오류가 발생할 수 있다.

- BST (Binary Search Tree)

  1) Concept

  BST, 이진 탐색 트리는 정렬된 이진 트리로 자료의 검색, 삭제, 삽입, 정렬 등을 효율적으로 수행할 수 있는 트리 자료구조이다. 이진 탐색 트리의 특징으론 이진 탐색을 쉽고 효과적으로 구현할 수 있다는 것에 있다.
  여기서 이진 탐색이란, 검색 범위를 줄여 나가면서 원하는 데이터를 검색하는 알고리즘이다. 예를 들어 어떠한 방식으로 정렬되어 있는 리스트에서 중간값을 확인하고 탐색한 값과 비교, 같다면 반환, 다르다면 대소를 비교해 탐색 범위를 줄여 나갈 수 있다.
  삽입과 삭제가 이루어질 때 root부터 대소 관계를 파악해 올바른 위치를 빠르게 찾을 수 있다. 재귀나 반복문을 통해 탐색하는 노드를 변경하여 위치를 탐색한다.

특정 데이터를 탐색할 때도 root노드부터 subtree를 확인하며 원하는 값을 갖는 노드를 찾아나간다. 따라서 시간복잡도는 평균적으로 O(log n)이 걸린다.

이진 탐색 트리의 구조는 다음과 같다. 최상단에 root 노드가 있고 최대 2개의 자식 노드를 가질 수 있다. 모든 노드의 키 값은 서로 달라야 한다. 트리에 삽입이 일어날 때 root 노드부터 탐색한다. 만약 root 노드가 없다면 삽입하는 노드가 root 노드가 된다. root 노드가 존재한다면 root 노드의 키 값과 삽입할 키 값을 비교해 탐색할 위치를 특정한다. 만약 root 노드의 키 값이 더 크다면 왼쪽으로, 아니라면 오른쪽으로 이동해 다시 탐색을 시작한다. 만약 삽입하기에 올바른 위치를 찾았다면 노드를 삽입하고 트리와 연결해준다. 삭제할 때도 삭제할 노드가 root인지 아닌지 판단하고, 삭제할 노드의 자식 노드의 개수를 판단하여 적절하게 삭제를 수행한다. 삭제에 대한 자세한 내용은 아래의 Delete를 참고하길 바란다.

다만 이 프로젝트에서 TermsBST는 완벽한 이진 탐색 트리라고 말할 수 없다. 왜냐하면 TermsBST는 개인정보 만료일자를 기준으로 트리가 정렬되는데 두 명 이상의 사람들의 개인정보 만료일자가 동일할 수 있기 때문이다. 따라서 TermsBST는 이진 탐색 트리의 성질은 가지지만 완벽하게 이진 탐색 트리라고 정의할 수 없다.

2) Insert

반복문을 돌면서 삽입할 위치를 찾고 삽입한 후 반복문을 빠져나오는 구조이다. 만약 root가 NULL이라면 트리 자체에 노드가 없다는 것이고 삽입하는 노드가 트리의 root가 된다.

root가 NULL이 아니라면 노드를 삽입할 위치를 찾는다. 트리를 이루는 기준을 통해 root 노드의 왼쪽에 위치할 것인지, 오른쪽에 위치할 것인지 판단한다. 만약 탐색하는 노드의 자리가 비어 있다면 그곳이 바로 새로운 노드를 삽입할 자리이다. 노드를 생성하고 데이터를 저장한 후 부모 노드와 포인터로 연결해준다.

3) Delete

트리에서 노드 삭제의 경우, 삭제하고 싶은 노드의 자식 노드 개수, root 노드인지의 여부 등이 영향을 미친다. 삭제하고자 하는 노드를 curNode, curNode의 부모 노드를 parent라고 정의하고 삭제 방법에 대해 설명하겠다.

자식 노드의 개수가 0일 때,

curNode가 root라면, 트리에 마지막 노드가 남아있었고 해당 노드를 삭제하는 것이기 때문에 root를 삭제하고 초기화해준다. root가 아니라면, curNode가 parent의 어느 방향 자식 노드인지 판단하고 노드 삭제 후 parent의 포인터를 NULL로 초기화해준다. 예를 들어 curNode가 parent의 오른쪽 자식 노드였다면, curNode를 삭제한 후 parent 노드의 right 포인터를 NULL로 설정한다.

자식 노드의 개수가 1일 때,

총 6가지 경우의 수가 존재하는데 curNode의 자식 노드 위치와 curNode가 root 인지, curNode가 parent의 어느 방향 자식 노드인지 등에 따라 달라진다. curNode가 root라면 노드를 삭제하고 curNode의 자식노드를 새로운 root로 설정해준다. curNode가 root가 아니라면 curNode와 연결되어 있던 parent의 포인터와 curNode의 자식노드를 연결해주고 curNode를 삭제한다. 예를 들어 curNode는 parent의 오른쪽 자식 노드이고 curNode에게 왼쪽 자식 노드가 존재한다면 parent->setRight(curNode->getLeft())를 통해 curNode의 위아래로 연결해준다. 그 후 curNode를 삭제해 BST가 온전하게 남아있도록 한다.

자식 노드의 개수가 2일 때,

이번 프로젝트에선 삭제 노드의 오른쪽 subtree에서 가장 작은 값을 대체하여 트리를 재구성하는 방향으로 구현해야 한다.

curNode의 오른쪽 자식 노드가 중요하다. 해당 노드를 rsmall이라고 정의하겠다.

rsmall에게 왼쪽 자식 노드가 존재하지 않을 경우,

curNode의 오른쪽 subtree에서 rsmall이 가장 작은 노드가 된다. curNode가 root라면 rsmall이 새로운 root가 되고 curNode의 왼쪽 subtree를 rsmall의 왼쪽 자식 노드로 연결해준다. curNode가 root가 아니라면 parent와 rsmall을 연결하고 curNode의 왼쪽 subtree와 rsmall을 연결해준다.

rsmall에게 왼쪽 자식 노드가 존재할 경우,

rsmall의 왼쪽 subtree에서 가장 작은 값을 찾아 새로운 rsmall로 지정해준다. rsmall과 rsmall의 부모 노드와의 연결을 끊고 rsmall이 curNode를 완벽하게 대체할 수 있도록 parent와 자식 노드들을 연결한다. 그 후 curNode를 삭제한다.

4. Result Screen

전체 command txt 파일

```
LOAD
ADD tom 50 2020-07-21 D
ADD bella 94 2023-08-31 B
ADD harry 77 2024-02-03 B
QPOP
SEARCH bob
SEARCH tom
PRINT NAME
PRINT A
PRINT B
PRINT C
PRINT D
DELETE NAME emily
PRINT NAME
PRINT C
DELETE DATE 2024-09-01
PRINT NAME
PRINT A
PRINT B
PRINT D
EXIT
```

data txt 파일

```
james 17 2023-08-30 B
bob 31 2023-02-22 A
sophia 25 2023-01-01 D
emily 41 2021-08-01 C
chris 20 2022-11-05 A
kevin 58 2023-09-01 B
taylor 11 2023-02-20 A
```

command	log txt file
LOAD	===== LOAD ===== james/17/2023-08-30/B bob/31/2023-02-22/A sophia/25/2023-01-01/D emily/41/2021-08-01/C chris/20/2022-11-05/A kevin/58/2023-09-01/B taylor/11/2023-02-20/A ========== data txt 파일로부터 개인정보 불러와 MemberQueue에 저장 완료

ADD tom 50 2020-07-21 D ADD bella 94 2023-08-31 B ADD harry 77 2024-02-03 B	===== ADD ===== tom/50/2020-07-21/D ========== ===== ADD ===== bella/94/2023-08-31/B ========== ===== ADD ===== harry/77/2024-02-03/B ==========
QPOP	===== QPOP ===== Success ==========
SEARCH bob SEARCH tom	===== SEARCH ===== bob/31/2023-02-22/2023-08-22 ========== ===== SEARCH ===== tom/50/2020-07-21/2023-07-21 ==========

PRINT NAME	===== PRINT ===== Name_BST bella/94/2023-08-31/2024-08-31 bob/31/2023-02-22/2023-08-22 chris/20/2022-11-05/2023-05-05 emily/41/2021-08-01/2023-08-01 harry/77/2024-02-03/2025-02-03 james/17/2023-08-30/2024-08-30 kevin/58/2023-09-01/2024-09-01 sophia/25/2023-01-01/2026-01-01 taylor/11/2023-02-20/2023-08-20 tom/50/2020-07-21/2023-07-21 ==========
PRINT A PRINT B PRINT C PRINT D	===== PRINT ===== Terms_BST A chris/20/2022-11-05/2023-05-05 taylor/11/2023-02-20/2023-08-20 bob/31/2023-02-22/2023-08-22 ========== ===== PRINT ===== Terms_BST B james/17/2023-08-30/2024-08-30 bella/94/2023-08-31/2024-08-31 kevin/58/2023-09-01/2024-09-01 harry/77/2024-02-03/2025-02-03 ========== ===== PRINT ===== Terms_BST C emily/41/2021-08-01/2023-08-01 ========== ===== PRINT ===== Terms_BST D tom/50/2020-07-21/2023-07-21 sophia/25/2023-01-01/2026-01-01 ==========

DELETE NAME emily	===== DELETE ===== Success ================
PRINT NAME	===== PRINT ===== Name_BST bella/94/2023-08-31/2024-08-31 bob/31/2023-02-22/2023-08-22 chris/20/2022-11-05/2023-05-05 harry/77/2024-02-03/2025-02-03 james/17/2023-08-30/2024-08-30 kevin/58/2023-09-01/2024-09-01 sophia/25/2023-01-01/2026-01-01 taylor/11/2023-02-20/2023-08-20 tom/50/2020-07-21/2023-07-21 ================
PRINT C	===== ERROR ===== 500 ================ Emily 삭제 후 가입 약관 C를 가지는 회원 없음. 따라서 출력 오류 발생
DELETE DATE 2024-09-01	===== DELETE ===== Success ================
PRINT NAME	===== PRINT ===== Name_BST harry/77/2024-02-03/2025-02-03 kevin/58/2023-09-01/2024-09-01 sophia/25/2023-01-01/2026-01-01 ================

PRINT A PRINT B PRINT D	===== ERROR ===== 500 ========== ===== PRINT ===== Terms_BST B kevin/58/2023-09-01/2024-09-01 harry/77/2024-02-03/2025-02-03 ========== ===== PRINT ===== Terms_BST D sophia/25/2023-01-01/2026-01-01 ========== 가입 약관 A로 설정했던 회원 모두 만료 일자에 의해 삭제됨. 따라서 출력 오류 발생
EXIT	===== EXIT ===== Success ==========

5. Consideration

MemberQueue와 BST insert까진 어렵지 않게 구현할 수 있었다. 하지만 DELETE 명령이 실행되었을 때 적절한 방법을 사용하여 BST 노드를 삭제해야 했는데 그 과정이 가장 번거롭고 오류가 많이 발생했던 부분이었다. 삭제하고 싶은 노드의 자식 노드 개수를 파악하고, 삭제하고 싶은 노드가 root 노드인지 확인해야 했으며 자식 노드의 개수가 2개일 땐 생각해야 할 경우의 수가 너무 많았다. 또한 DELETE가 이름을 기준으로 개인정보 하나를 삭제하는 건지, 만료일자를 기준으로 다수의 정보를 삭제해야 하는지 또한 고려해야 했기 때문에 효율적으로 구현하기 어려웠다. 처음으로 구현했을 때 예시로 제공된 command txt 파일은 정상적으로 작동했지만 임의로 만든 command 명령이 제대로 수행되지 않는 것을 확인, 모든 경우의 수를 고려했는지 재확인하기 위해 DELETE 관련 함수들을 모두 다시 만들었다. 우선 각 BST class에서 삭제할 노드를 매개변수로 하는 delete 함수를 작성했다. 그리고 특히 DELETE DATE의 경우 모든 BST를 후위순회를 돌며 조건에 따라 노드를 삭제하도록 구현했다. 삭제하고자 하는 노드가 root인지 아닌지를 파악하는 것이 중요했으며 root라면 꼭 root를 재정의시켜 BST가 끊겨 개인정보가 사라지는 오류

가 발생하지 않도록 조치하는 것이 중요하다는 것을 깨달았다.
본인은 리눅스에 대한 이해도가 부족해 먼저 visual studio를 사용해 구현하고 리눅스로 옮겨 정상적으로 동작하는지 확인하면서 프로젝트를 진행했다. command txt 파일과 data txt 파일을 열고 안에서 데이터를 불러올 때 string stream을 사용하여 불러왔었다. 하지만 리눅스에선 지원하지 않음을 확인했고 txt 파일을 열고 닫을 때 선언했던 fstream을 통해 입력받을 수 있는 방법을 알아봤고 그에 맞게 수정 후 리눅스에서 정상적으로 동작함을 확인했다.