2019202005_DS_project1

수업 명 : 데이터구조설계
과제 이름 : DS project1
담당 교수님 : 이기훈 교수님
학 번 : 2019202005
이 름 : 남종식

Introduction

이번 과제는 data.txt에 저장되어 있는 data를 load해 Queue와 TermsList, TermsBST 그리고 NameBST를 구축하는 과제입니다. TermsList, TermsBST, NameBST는 Queue에서 pop을 통해 방출된 data로 통해 구성됩니다. 이때 data는 이름, 나이, 개인정보수집일자, 가입약관 종류로 이루어져 있고 가입약관 종류로는 A,B,C,D 총 4개가 있습니다.

TermsList는 가입약관 별로 이루어져 있으며 각 약관에 해당하는 인원수가 저장되어 있습니다. 각 가입약관 별로 TermsBST가 구성되는데 이는 이름, 나이, 개인정보수집일자, 만료일자가 저장되어 있습니다. 이때 개인정보 만료일자 비교를 통해 연결됩니다.

NameBST는 Queue에서 방출된 data를 사전적 이름 순서 비교를 통해 연결됩니다.

이때 각 BST에서 delete한 정보는 TermsList, TermsBST, NameBST에서 모두 삭제해줘야 합니다.

더 자세한 내용은 이후 알고리즘 동작을 설명 때 하겠습니다.

다음은 명령어 기능에 대한 간단한 설명입니다.

LOAD

LOAD명령어를 통해 data.txt파일을 불러올 수 있으며 이를 Member_Queue에 저장합니다. 오류가 없을 시에는 불러온 data를 출력합니다. 만약 텍스트 파일이 존재하지 않거나 자료구조에 이미 데이터가 들어가 있으면 에러 코드 100을 출력합니다.

ADD

ADD명령어를 통해 Member_Queue에 직접 데이터를 추가할 수 있습니다. 총 4개의 정보 중 하나라도 없으면 에러 코드 200을 출력합니다. 오류가 없을 시에는 추가하는 data를 출력합니다.

SEARCH

SEARCH명령어를 통해 Name_BST에 저장된 회원정보를 출력할 수 있습니다. 인자로 이름이 Name_BST에 존재하면 회원이름에 해당하는 회원정보를 출력하고 존재하지 않는다면 에로 코드 400을 출력합니다.

PRINT

PRINT명령어를 통해 Terms_BST와 Name_BST에 저장된 회원정보를 출력할 수 있습니다. 인자로 가입약관의 종류가 입력되면 해당 가입약관 Terms_BST의 저장된 데이터들을 출력하고 인자로 NAME이 입력되면 Name_BST의 저장된 데이터들을 출력합니다. 이때 중위 순회방식으로 탐색 후 출력하며 BST에 데이터가 없으면 에러 코드 500을 출력합니다.

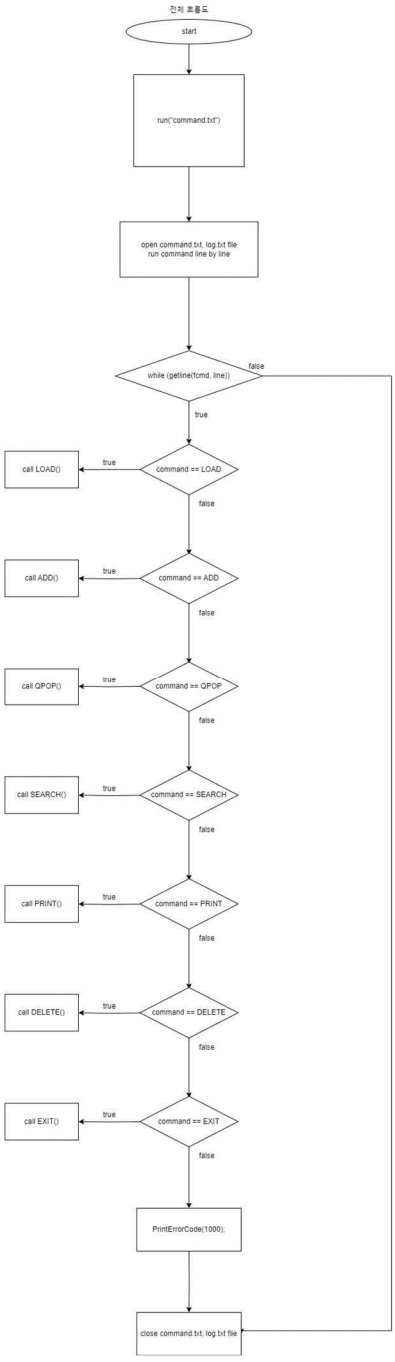
DELETE

DELETE명령어를 통해 저장된 데이터를 삭제할 수 있습니다. 인자로 DATE가 들어오면 해당 날짜보다 만료일자가 적은 데이터들은 모두 삭제하며 이때 삭제는 Terms_list, Terms_BST, Name_BST에서 모두 이루어집니다.
인자로 이름이 들어오면 해당 이름의 정보가 있는 데이터는 삭제하며 이 또한 Terms_list, Terms_BST, Name_BST에서 모두 이루어집니다.
삭제하고자 하는 회원 정보가 존재하지 않거나, 자료구조에 데이터가 존재하지 않을 시 에러 코드 600을 출력합니다.

EXIT

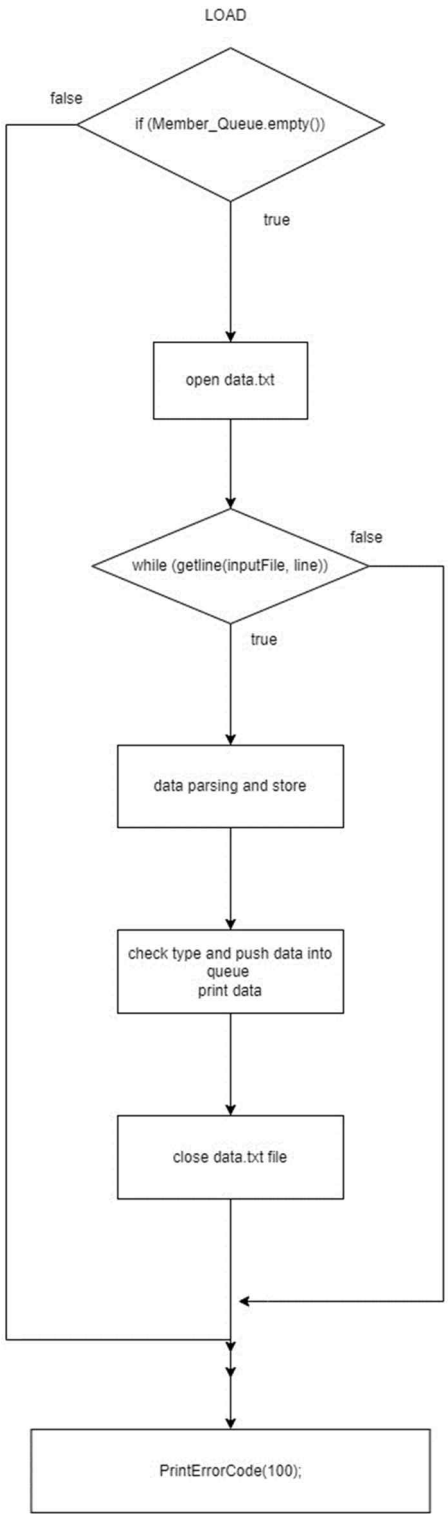
EXIT명령어를 통해 프로그램 상의 메모리를 해제하며, 프로그램을 종료합니다.
잘못된 명령어가 들어오면 에러 코드 1000을 출력합니다.

Flowchart



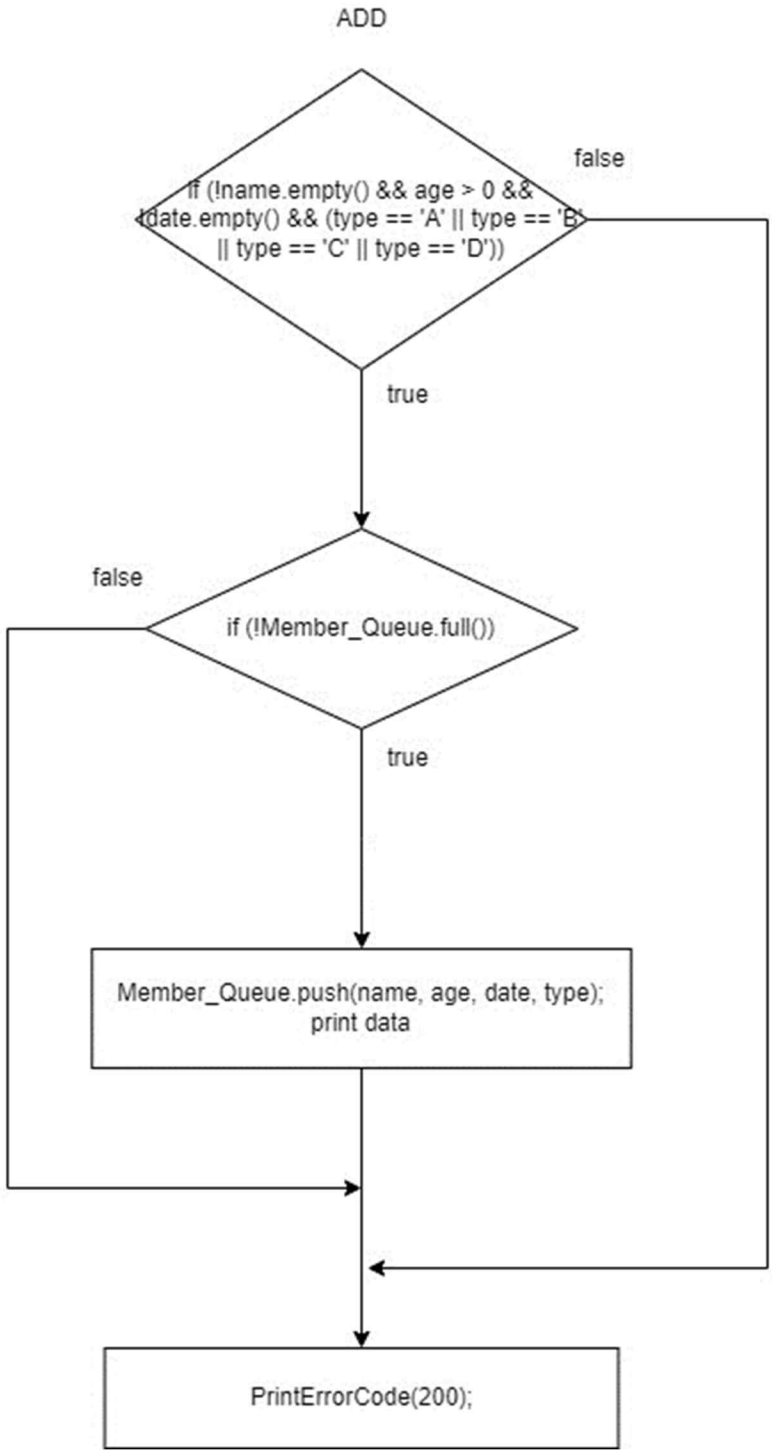
main.cpp 파일에서 제일 먼저 실행되는 Manager 클래스의 Run함수입니다. command.txt 파일을 인자로 전달하여 실행한 것을 flowchart에서 볼 수 있습니다. command.txt 파일에서 getline으로 한줄씩 읽어온 다음 이를 바탕으로 어떤 명령어가 실행될지 알 수 있습니다. EXIT함수로 while문이 끝날 때까지

command를 읽어와 명령어의 기능을 수행하게 됩니다.

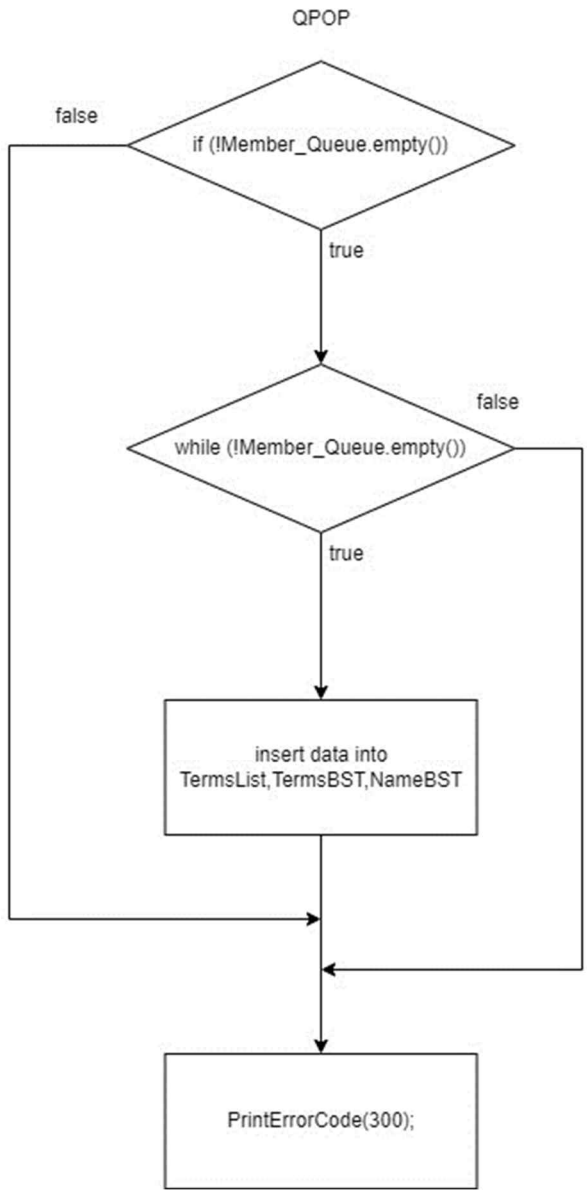


다음으로 LOAD command일 때 실행되는 순서도입니다. 만약 텍스트 파일이 존재하지 않거나 큐에 데이터가 들어가 있는 상태라면 오류코드 100을

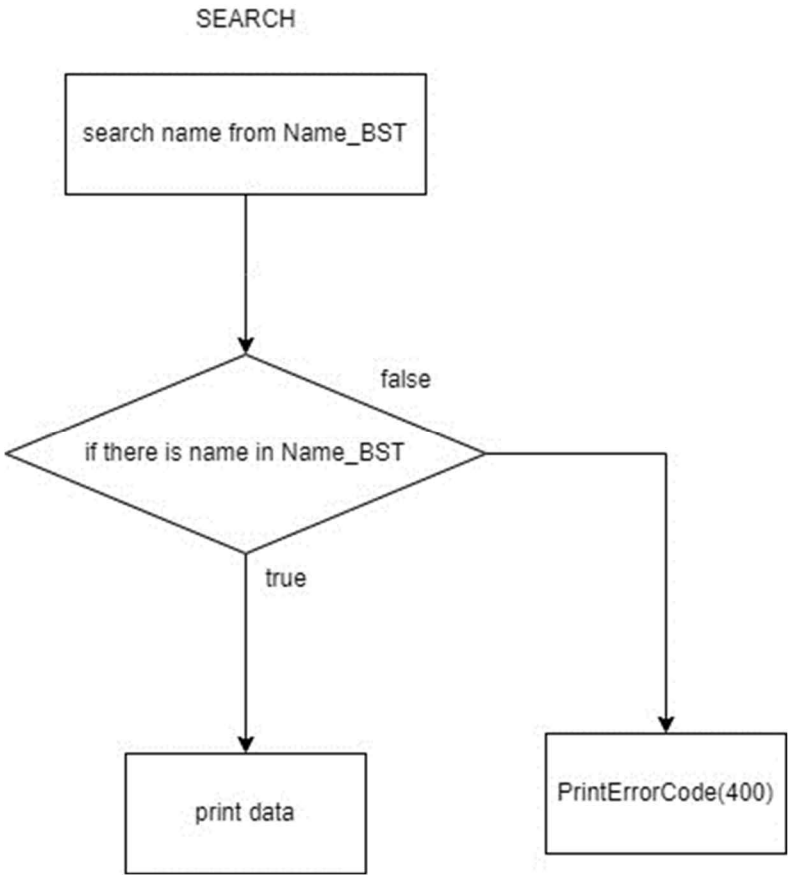
출력합니다.



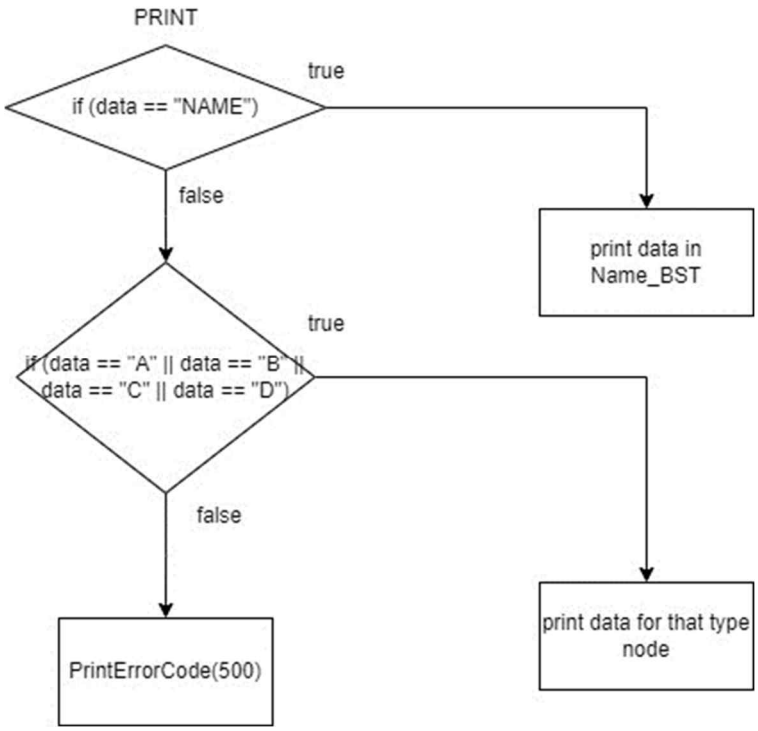
다음으로 ADD command일 때 실행되는 순서도입니다. 먼저 에러 조건을 체크를 해 에러 발생시 에러 코드 200을 출력합니다. 에러가 없다면 큐가 비어 있는지에 대해 확인하고 데이터를 추가해줍니다.



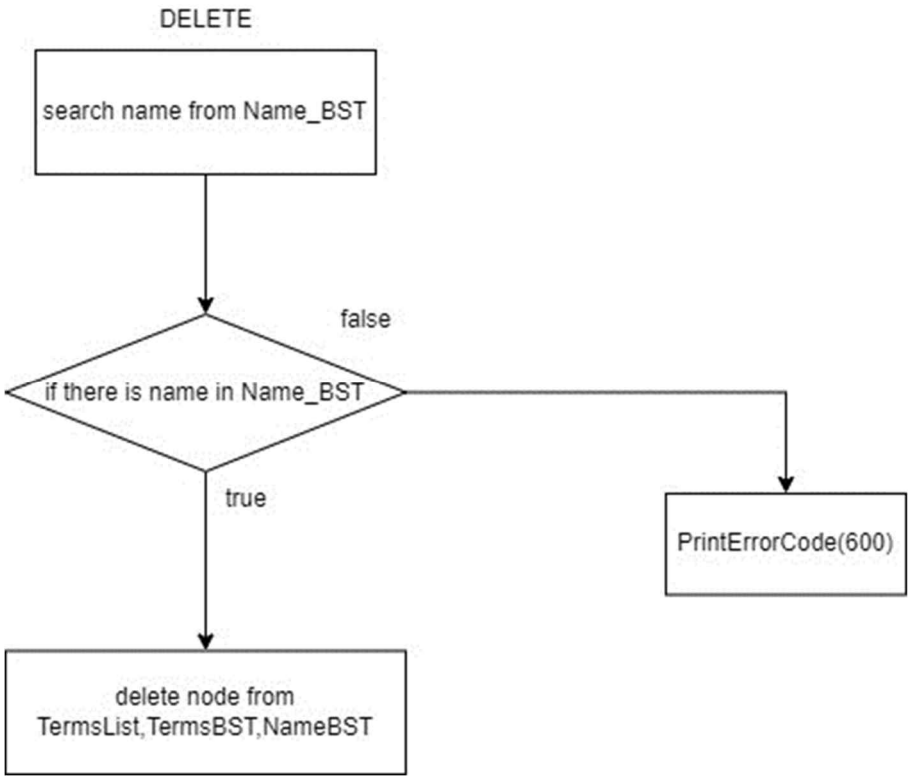
다음은 QPOP command일 때 실행되는 순서도입니다. 동일하게 에러 조건을 먼저 체크하고 만약에 큐가 비어 있다면 에러 코드 200을 출력합니다. 에러 조건을 통과하면 큐에 데이터가 없을 때까지 pop을 진행한 후 이 pop된 데이터들을 자료구조에 insert하게 됩니다.



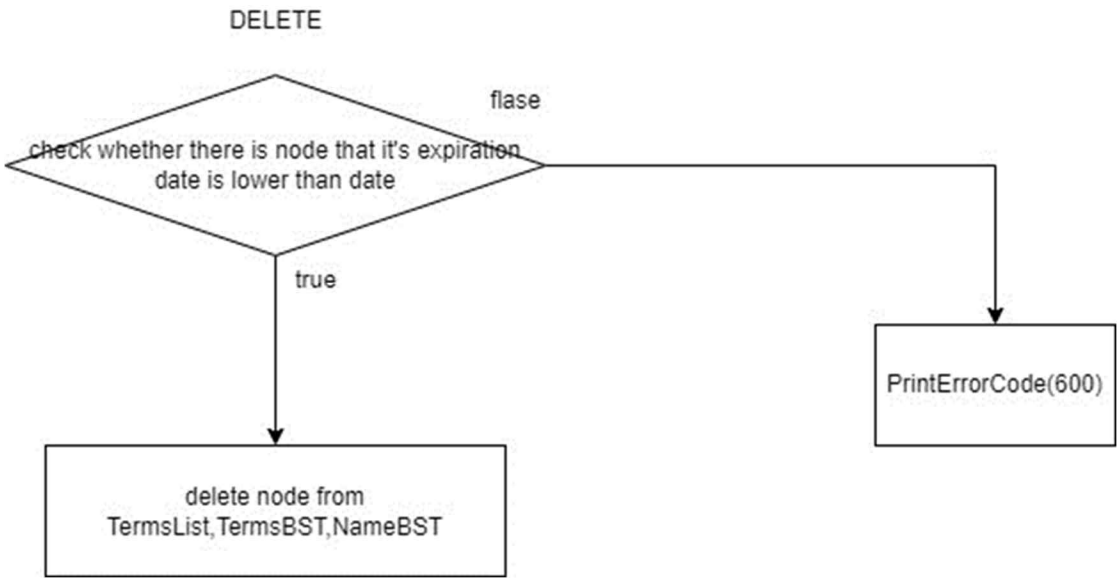
다음은 SEARCH command일 때 실행되는 순서도입니다. 먼저 Name_BST에 정의되어 있는 searchNode를 통해 해당 이름 데이터가 있는 노드가 존재하는지 찾고 존재한다면 해당 노드의 데이터를 출력하도 존재하지 않는다면 에러 코드 400을 출력합니다.



PRINT command일 때 실행되는 순서도입니다. 일단 인자로 data가 전달되는데 이때 data가 NAME이라면 Name_BST에 저장 되어있는 모든 노드의 데이터를 출력해줍니다. 인자로 type중 하나가 전달된다면 해당 타입의 Terms_BST에 저장되어있는 모든 노드의 데이터를 출력합니다. 이때 출력되는 순서는 inorder방식입니다. 그 외의 인자가 전달된다면 에러 코드 500을 출력합니다.



DELETE command일 때 실행되는 순서도입니다. 먼저 DELETE command의 인자로 이름이 전달될 때부터 보겠습니다. 이름을 통해 Name_BST에 존재하는 노드인지 확인하고 존재한다면 해당 노드를 삭제해줍니다. 만약 해당 노드가 존재하지 않는다면 에러 코드 600을 출력합니다.

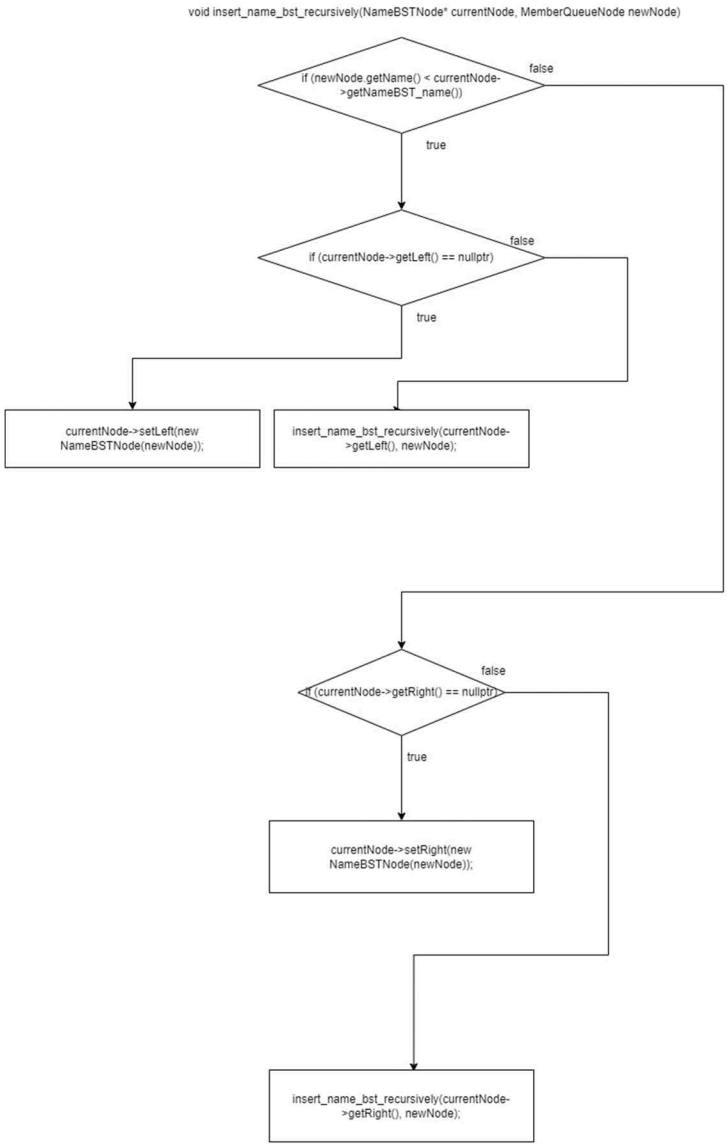


다음으로 DELETE command의 인자로 날짜가 전달되는 상황입니다. 먼저

Name_BST에서 해당 날짜보다 만기날짜가 더 이전인 노드가 존재한다면 해당 노드들을 모든 자료구조에서 삭제해줍니다. 하지만 모든 노드의 만기날짜가 해당 날짜 이후라면 삭제를 진행할 노드가 없으므로 에러 코드 600을 출력합니다.

다음으로 각 command에서 사용되는 중요한 함수들에 대해서 알아보겠습니다.

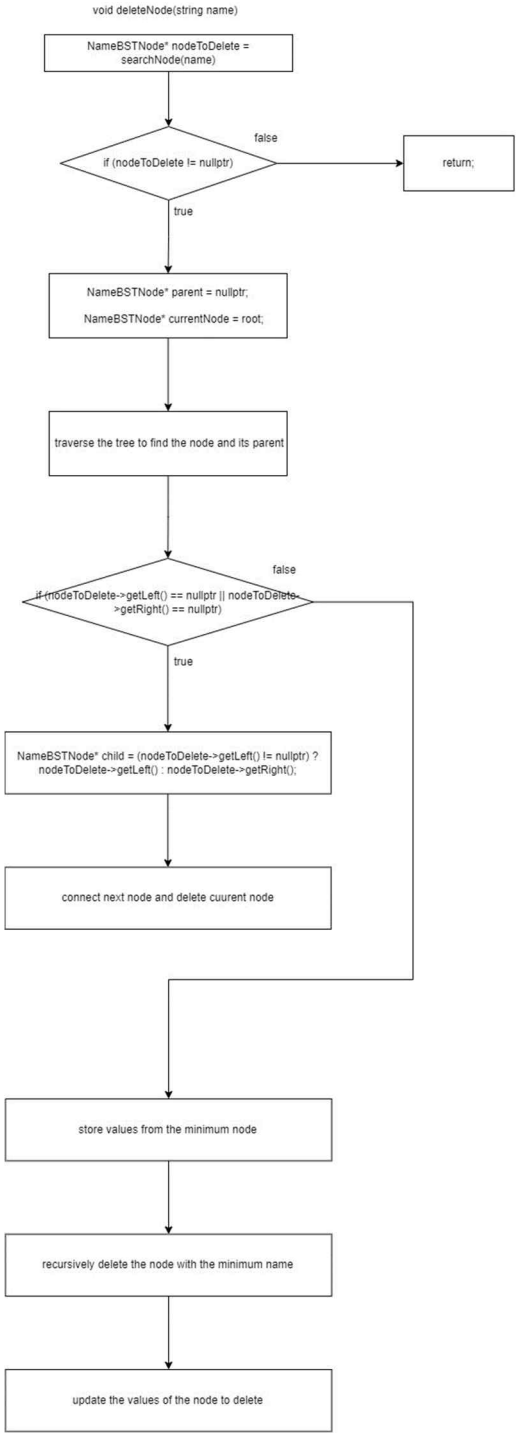
void NameBST::insert_name_bst_recursively(NameBSTNode* currentNode, MemberQueueNode newNode)



NameBST를 구축하기 위해서 사용되는 insert관련 함수입니다. 이 함수를 재귀적으로 호출하고 만기날짜를 계산하는 과정을 통해 NameBST를 구축할 수 있습니다. 이름을 통해 사전적 순서를 비교하고 루트보다 작으면 왼쪽 크면

오른쪽 서브트리로 이동하는 방법으로 BST는 구축됩니다.

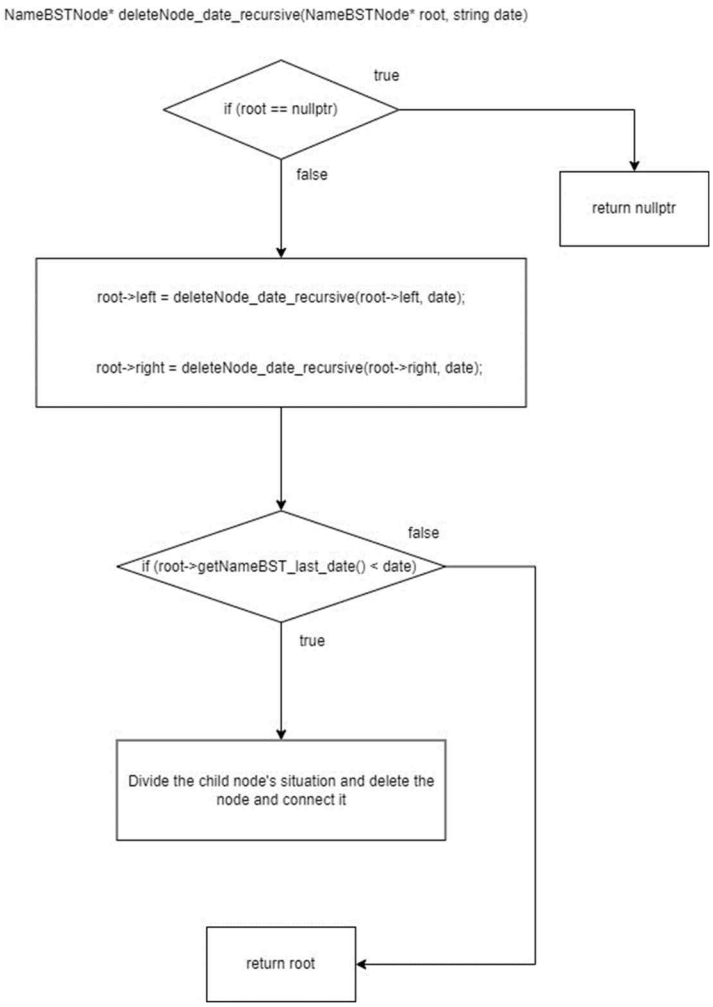
void NameBST::deleteNode(string name)



먼저 searchNode함수를 호출해 name을 통해 해당하는 노드를 찾습니다. 이때 searchNode함수는 모든 노드를 순회합니다. 일단 해당 노드가 자식을 2개 가지는지, 하나 또는 없는 지를 기준으로 조건을 나눕니다. 부모 노드가 null일 때

해당 노드를 지우고 자식 노드를 root로 업데이트 합니다. 부모 노드가 null이 아닐 때는 해당 노드가 부모의 왼쪽에 있으면 해당 노드의 자식 노드를 왼쪽 노드로 할당하고 해당 노드는 지워 줍니다. 오른쪽 서브 트리도 같은 방식으로 진행합니다. 두개의 자식 노드를 가지고 있을 때는 오른쪽 서브 트리에서 최솟값을 가진 노드를 찾아 이 노드를 해당 노드의 자리로 대체하고 해당 노드는 지워줍니다.
인자가 날짜로 들어올 때 또한 비슷하게 진행합니다. 이름 대신 날짜를 비교하여 인자로 들어온 날짜보다 만기날짜가 더 이전인 모든 노드들을 삭제하고 남아있는 노드들을 연결합니다.

NameBSTNode* NameBST::deleteNode_date_recursive(NameBSTNode* root, string date)



인자로 들어온 날짜보다 만기날짜가 더 이전인 노드를 삭제하기 위해서 일단

재귀적으로 순회하고 날짜 비교를 진행합니다. 만약 이때 삭제를 진행해야 하는 노드라면 다음 3가지 경우의 수를 따릅니다.
Case 1: 삭제할 노드가 왼쪽 자식을 가지지 않는 경우에는 삭제할 노드를 임시 변수 temp에 저장하고, 해당 노드를 삭제한 후 오른쪽 자식을 반환합니다.
Case 2: 삭제할 노드가 오른쪽 자식을 가지지 않는 경우에는 마찬가지로 삭제할 노드를 temp에 저장하고 해당 노드를 삭제한 후 왼쪽 자식을 반환합니다.
Case 3: 삭제할 노드가 양쪽 자식을 모두 가지는 경우에는 오른쪽 서브트리에서 최솟값을 가진 노드를 찾습니다. 그 노드의 값들을 현재 노드에 복사하고, 오른쪽 서브 트리에서 최솟값을 삭제합니다. 마지막으로 현재 노드를 반환하면 수정된 BST를 확인할 수 있습니다.

Algorithm

파일 읽기 후 명령어 및 인자들을 구분하는 Algorithm

일단 파일로부터 getline을 통해 데이터를 한 줄 씩 읽어서 string로 저장하고 명령어와 인자들을 구분하기 위해서는 stringstream을 사용하였습니다.

LOAD와 ADD를 통한 큐에 데이터 삽입하는 Algorithm

LOAD명령어를 통해 data.txt에 읽는 데이터를 읽어와 큐 자료구조에 삽입할 수 있습니다. 이때 또한 인자들을 파싱 해서 이름을 name, 나이를 age, 개인정보 수집일자를 date, 약관 종류를 type에 저장 후 큐에 삽입합니다. 이때, 큐에는 데이터가 존재하지 않아야 하며 만약 존재한다면 에러 코드 100을 출력합니다. ADD명령어 또한 거의 동일합니다. 이 또한 큐에 데이터를 추가하는 명령어이지만 LOAD와 다르게 직접 큐에 추가할 수 있습니다. 4개의 인자 중 하나라도 존재하지 않으면 에러 코드 200을 출력합니다. 두 명령어 모두 MemberQueue.cpp에 구현되어 있는 push함수를 통해 큐에 데이터 삽입이 이루어집니다.

QPOP을 통한 큐에서 데이터를 POP하여 Terms_List와 Terms_BST와 Name_BST를 구성하는 Algorithm

QPOP명령어는 LOAD와 ADD를 통해 만들어진 큐에서 pop을 해서 TermsList와 TermsBST와 NameBST를 만들 수 있습니다. 이 명령어가 실행되면 각 자료구조에 구현되어 있는 insertNode함수를 호출하여 자료구조를 구축합니다.

TermsList에서는 pop된 데이터들의 약관 종류 별로 총 4개의 리스트를 구축하며 이때 각각의 리스트는 TermsBST를 구축하게 됩니다. TermsBST에서는 개인정보 수집일자에 가입약관 별로 개인정보 보관 유효기간이 더해져 계산되어 이를 기준으로 BST를 구축합니다. 부모 노드보다 개인정보만료일자가 이전인 노드는 왼쪽, 같거나 이후인 노드는 오른쪽 서브 트리에 위치합니다.
또한, 큐에서 pop된 데이터는 NameBST를 구축하게 되는데 이때는 회원 이름의 사전적 순서에 따라서 BST가 구축됩니다. 부모 노드보다 회원이름의 사전적 순서가 이전인 노드는 왼쪽, 같거나 이후인 노드는 오른쪽 서브 트리에 위치합니다. BST를 구성할 때 모두 재귀적으로 함수 호출을 통해 구성하였으며 ExpirationDate라는 만료일자를 계산하는 함수를 작성하여 수집일자를 통해 계산 후 노드를 구성할 때 만료일자 정보도 추가해주었습니다.
큐에 데이터가 존재하지 않을 때 명령어가 실행되면 에러 코드 300을 출력합니다.

SEARCH를 통한 NameBST에 저장된 회원정보를 출력하는 Algorithm

SEARCH명령어가 실행된다면 NameBST에 구현되어 있는 searchNode함수가 호출되며 이때 인자에 해당하는 이름과 모든 노드에 저장되어 있는 이름을 비교해 이름에 해당하는 회원정보를 찾아 출력합니다. 이때 인자로 들어온 이름이 노드에 저장되어 있는 이름보다 사전적 순서가 더 이전이면 왼쪽 서브트리로 이동해 탐색하고 이후라면 오른쪽 서브트리로 이동해 탐색합니다. 찾고자 하는 회원정보가 Name_BST에 존재하지 않는 경우 에러 코드 400을 출력합니다.

PRINT를 통한 중위 순회로 노드의 정보들을 출력하는 Algorithm

PRINT명령어를 통해 BST에 저장되어 있는 노드들을 중위 순회로 출력할 수 있습니다. 이때, 인자로 약관 종류가 들어온다면 해당 약관의 TermsBST에 저장되어 있는 모든 데이터를 출력해야 하고 인자로 NAME이 들어온다면 NameBST에 저장되어 있는 모든 데이터를 출력해야 합니다. 이때 출력 방식은 중위 순회 방식으로 재귀함수를 이용하게 됩니다. 따라서 재귀함수 호출 – 출력 – 재귀함수 호출 이런 형태로 알고리즘을 구성했습니다.
만약 BST에 데이터가 존재하지 않는다면 에러 코드 500을 출력합니다.

DELETE를 통한 List와 BST에 저장된 데이터를 삭제하는 Algorithm

먼저 DELETE명령어의 인자로 이름이 들어온 경우에는 이름에 해당하는 회원 정보를 searchNode를 통해 찾습니다. 찾고 난 이후에 이 노드가 만약에 자식이 없는 경우에는 그냥 삭제를 진행해주고 하나가 있는 경우에 만약 루트 노드라면 삭제 후 자식 노드를 루트로 설정해주고 루트 노드가 아니라면 본인의 부모 노드와 자식 노드를 연결해줍니다. 그 후 본인 노드는 삭제합니다. 그리고 양쪽 자식 노드가 모두 존재할 경우에는 오른쪽 자식 노드 중 가장 작은 노드를 제거되는 노드 위치로 이동합니다. 이때는 오른쪽 자식 노드 중 가장 작은 노드를 찾는 findMin함수를 이용합니다. 이 모든 과정 또한 재귀적으로 이루어지게 됩니다. DELETE명령어가 실행되면 TermsList와 TermsBST에서도 삭제가 진행되어야 하는데 TermsList에서는 해당 회원의 약관 종류에 해당하는 노드의 개수를 하나 감소시키면 됩니다. TermsBST에서는 만기날짜를 기준으로 BST가 구성되어 있기 때문에 인자로 받은 이름을 가지고 searchNode를 통해 날짜를 반환합니다. 이를 통해 날짜 정보를 얻게 되면 NameBST와 동일한 과정으로 해당 노드를 재귀적으로 찾아 삭제합니다.

다음으로 DELETE명령어의 인자가 날짜가 들어온 경우에는 해당 날짜보다 만기날짜가 더 이전인 노드들은 모두 삭제해야 합니다. TermsList에서는 위와 같은 방법으로 객수를 줄여주며 NameBST에서는 각 노드에 저장되어 있는 만기날짜와 인자로 들어온 date를 비교하여 위와 같은 방법으로 재귀적으로 삭제합니다. TermsBST에서는 인자로 들어온 날짜를 가지고 가입 약관 별로 구성되어 있는 총4개의 BST에서 모든 재귀적으로 비교를 통해 삭제하게 됩니다.

각각의 BST를 구축할 때는 어차피 하나의 BST안에서는 약관 종류가 같으므로 각 노드의 개인정보 수집일자를 기준으로 insert하였으며 이때 만기날짜를 계산하는 함수를 호출하여 각 노드를 추가할 때 만기날짜 계산하였으며 삭제 시에는 만기날짜를 기준으로 삭제하였습니다.

이때 삭제하고자 하는 회원 정보가 존재하지 않는 경우와 주어진 날짜보다 만기날짜가 이전인 노드가 없는 경우에는 에러 코드 600을 출력합니다.

가입일자를 기준으로 만기날짜를 계산하는 Algorithm

과제에서는 가입일자를 알려주지만 노드에 데이터를 삽입하거나 출력할 때는 만기날짜 또한 포함되어 있기 때문에 이를 계산하는 함수를 따로 구현했습니다. 먼저 가입이자를 인자로 받으면 이를 파싱해서 년, 월, 일로 나누어 변수에

저장해줍니다. 그 후, 타입을 서로 비교해 A타입은 +6개월, B타입은 +1년, C타입은 +2년, D타입은 +3년으로 계산합니다. tm structure를 이용하여 날짜를 정한 후 형식을 맞추어 만료날짜를 저장합니다. tm_mon은 0부터 시작하기 때문에 1을 빼서 저장합니다. 이 과정을 재귀적으로 반복합니다.

Result Screen

```
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
```

위 사진은 command.txt이며 여기에 작성된 순서대로 명령어에 대해 결과화면을 캡쳐하고 동작을 설명하겠습니다.

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
```

```
log.txt
===== 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
================
```

LOAD명령어를 통해 첫번째 사진에서 볼 수 있는 data.txt의 데이터들을 가져와 출력한 모습을 확인할 수 있습니다. 출력 포맷에 맞게 각 정보들은 /를 통해 구분되어 출력됩니다. 그리고 이 데이터들은 큐에 저장됩니다.

ADD

```
===== 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

위 command를 진행한 결과이며 ADD명령어를 통해 큐에 직접 데이터를 추가하는 동작을 하고 추가된 데이터의 정보가 잘 출력되는 모습을 확인할 수 있습니다. 하지만 큐에 추가되었는 지에 대해서는 아직 확인할 수 없으므로 다음 PRINT문에서 확인하겠습니다.

QPOP

```
===== QPOP =====
Success
===============
```

QPOP명령어가 성공적으로 이루어지면 큐에서 pop이 진행되고 pop된 데이터들을 바탕으로 TermsList, TermsBST, NameBST를 구축한 후 success

메시지를 출력합니다. 모든 자료구조에 데이터들이 성공적으로 insert된 모습을 확인하기 위해서는 다음 SEARCH와 PRINT를 참고하면 됩니다.

SEARCH

```
===== SEARCH =====
bob/31/2023-02-22/2023-08-22
===============

===== SEARCH =====
tom/50/2020-07-21/2023-07-21
===============
```

SEARCH bob

SEARCH tom

위 command를 진행한 결과이며 SEARCH명령와 뒤에 인자 이름을 통해 Name_BST에서 해당 이름의 노드 정보를 출력하는 모습을 확인할 수 있습니다. 이를 통해 BST에 데이터들이 잘 insert된 것 또한 알 수 있습니다.

PRINT

```
===== 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 NAME을 진행한 결과이며 이를 통해 Name_BST가 잘 구축된 모습을 확인할 수 있습니다. 이 PRINT를 통하여 이전에 LOAD후 ADD한 모든 정보들이 QPOP을 통하여 BST를 잘 구축했음을 알 수 있습니다. 이때 이름 비교를 통해 BST를 그려봤을 때 중위 순회로 출력된 순서가 위에 log에 출력되는 순서와 동일한 것을 알 수 있습니다. 만기날짜 또한 잘 계산되어 삽입 후 출력까지 되는 모습을 확인할 수 있습니다.

```
===== PRINT =====
TermsBST 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 =====
TermsBST 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 =====
TermsBST C
emily/41/2021-08-01/2023-08-01
==============

===== PRINT =====
TermsBST D
tom/50/2020-07-21/2023-07-21
sophia/25/2023-01-01/2026-01-01
==============
```

PRINT A

PRINT B

PRINT C

PRINT D

위 command를 실행한 화면이며 약관 종류 별로 총 4개의 BST가 잘 구축되어 출력되는 모습을 확인할 수 있습니다. 이 또한 중위순회로 출력된 것이며 만기날짜 또한 잘 계산되어 삽입 후 출력까지 이어지는 모습입니다. 이는 각 타입 별로 BST를 그려봤을 때 중위순회로 출력된 결과가 위에 log에 출력되는 순서가 똑같다는 것을 알 수 있습니다.

수집날짜와 만기날짜가 서로 TermsBST A는 6개월, TermsBST B 1년, TemsBST C는 2년, TermsBST D는 3년 차이나는 모습을 볼 수 있습니다.

DELETE

```
===== DELETE =====
Success
===============
```

DELETE NAME emily를 진행한 화면이며 오류 코드 없이 success가 잘 출력되는

모습입니다. 즉, 삭제가 잘 이루어졌다는 점을 알 수 있으며 이에 대한 검증은 다음 PRINT문에서 할 수 있습니다.

```
===== 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 NAME을 진행한 결과 화면입니다. 삭제를 진행하지 않았을 때와 비교했을 때 DELETE NAME emily이후로 emily의 이름 정보가 있는 노드가 삭제되어 출력되지 않는 모습을 확인할 수 있습니다. 이를 통해 해당 노드의 삭제가 잘 이루어진 모습을 확인할 수 있습니다.

```
===== ERROR =====
500
=================
```

DELETE NAME emily를 진행했을 때 TermsBST에서도 해당 노드가 삭제되었는지에 대해 확인하기 위해서 PRINT C를 진행한 결과 화면입니다. emily밖에 없었던 TermsBST이므로 삭제 이후에는 노드에 데이터가 없어 오류 코드 500을 출력하는 모습입니다. TermBST에서도 해당 노드의 삭제가 잘 이루어진 점을 확인할 수 있습니다.

```
===== DELETE =====
Success
===============
```

DELETE DATE 2024-09-01를 진행한 결과 화면이며 오류 코드 없이 success를 보아 삭제가 잘 이루어진 것을 알 수 있습니다. 이에 대한 검증은 다음 PRINT문을 통해 할 수 있습니다.

```
===== 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 NAME을 진행한 결과 화면이며 만기날짜가 2024-09-01 이전의 모든 노드들은 삭제되고 당일이거나 그 이후인 노드들은 삭제되지 않고 잘 출력되는 모습을 확인할 수 있습니다.

```
===== ERROR =====
500
=================

===== PRINT =====
TermsBST B
kevin/58/2023-09-01/2024-09-01
harry/77/2024-02-03/2025-02-03
==============

===== PRINT =====
TermsBST D
sophia/25/2023-01-01/2026-01-01
==============
```

PRINT A

PRINT B

PRINT D

위 command 진행 후 결과 화면이며 TermsBST A의 모든 노드들은 만기날짜가 2024-09-01보다 이전이어서 삭제가 진행된 후 해당 BST에 저장된 데이터가 없어 오류 코드 500을 출력하는 모습입니다. 다음 TermsBST B와 TermsBST D는 2024-09-01이후인 노드들만 출력되고 있습니다.

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

마지막으로 프로그램이 성공적으로 종료된 모습을 확인할 수 있습니다.

Consideration

일단 처음에 각 자료구조들을 구현하고 그 안의 함수들을 확인할 때 마다 임의로 main함수를 수정하며 진행하였는데 이 방법이 처음에 자료구조가 하나일 때는 편했는데 자료구조들이 많아질수록 많이 헷갈리고 굉장히 불편했습니다. 처음부터 manager파일에서 함수들의 동작을 확인했으면 보다 덜 헷갈리고 좀 더 빨리 진행할 수 있었을 것 같습니다. 그리고 이번 프로젝트를 진행할 때 윈도우 환경에서 작성 후 비주얼 스튜디오를 통해 동작을 확인했는데 다음 프로젝트부터는 바로 리눅스 환경에서 진행을 해야겠다고 생각했습니다. 처음에 윈도우에서 작업한 이유는 디버깅을 통해 어디 부분에서 오류가 나는 지 확인하기 위함 이였는데 다음부터는 리눅스 환경에서 gdb사용법을 익혀서 바로 진행할 것입니다. 윈도우에서 리눅스로 파일을 옮길 때 혹시 모르는 오류가 발생할 수도 있다고 생각했습니다. 이번 과제에서는 크게 문제가 발생하지 않았지만 혹시라도 발생하면 과제제출이 다가온 시점에서 멘탈이 많이 흔들릴 것 같았습니다. 그리고 과제를 진행하면서 각 명령어에 따른 데이터 출력 결과는 log.txt에 저장해야 하는데 계속 로그 파일을 확인하는 것 보다는 터미널에서 출력되는 지 빠르게 확인하는 것이 나을 것 같아서 log에 기록하는 부분마다 cout을 통해 터미널에서도 확인할 수 있게 했던 점이 좋았던 것 같습니다. 마지막으로 이번 과제를 통해 많이 들어만 보고 직접 구현은 하지 않았었던 자료구조들에 대해서 좀 더 자세히 알게 되었고 전체 프로그램이 여러 개의 파일로 어떻게 이어지는 지 또한 알게 되었습니다. 프로그램의 구조면에서 헷갈리지 않게 node.h파일, .h파일, .cpp파일 순으로 각 자료구조를 만들어갔으며 .cpp에서 부족한 함수가 있으면 .h파일에 추가하는 형식으로 프로그램 구성을 할 수 있었습니다. 그리고 운영체제 강의에서 배운 snapshot을 통해 혹시 모를 오류 및 우분투 에러 상황을 대비할 수 있었습니다.