데이터구조실습

Project1

담당교수 : 이형근 교수님

학 번 : 2021202058

성 명 : 송채영

1. Introduction

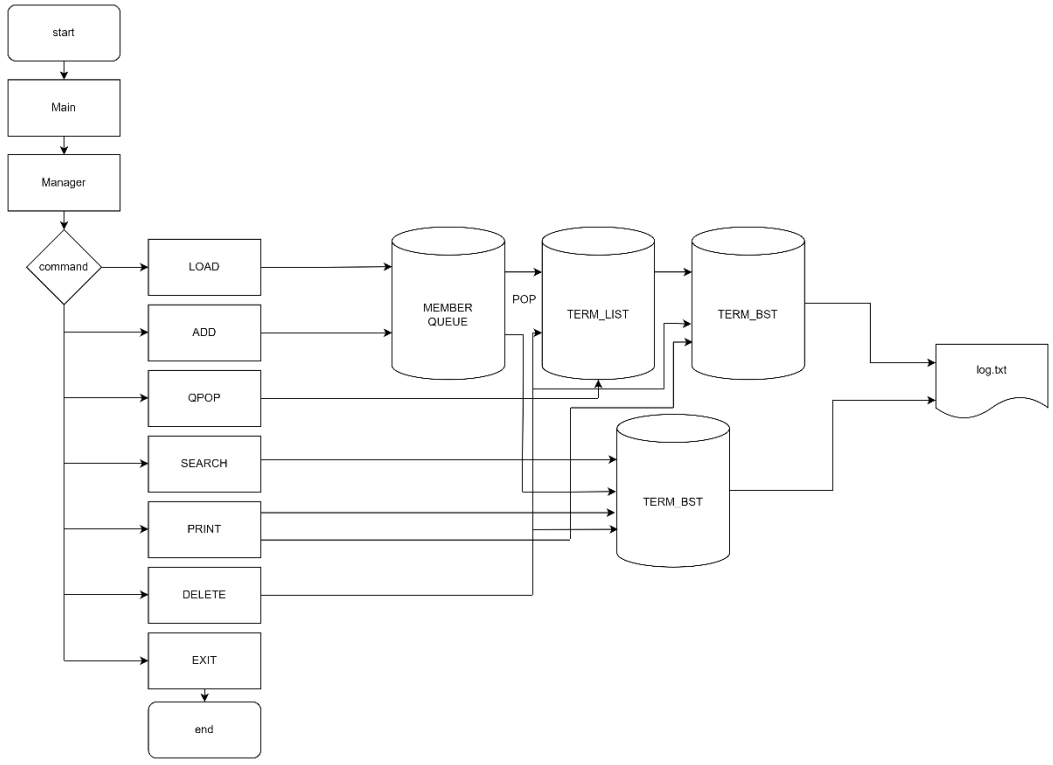
이번 프로젝트에서는 이진 탐색 트리(Binary Search Tree)와 연결 리스트(Linked List), 큐(Queue) 자료구조를 활용하여 개인정보 관리 프로그램을 구현하는 것이 목표이다. 데이터 파일로부터 회원의 이름, 나이, 개인정보 수집 일자, 가입약관 종류 정보를 읽어와 Member_Queue, 큐 자료구조에 구축한다. 이때 가입약관 종류는 A, B, C, D 이며 각각 6 개월, 12 개월, 24 개월, 36 개월에 해당한다. Member_Queue 에서 pop 명령어를 실행하면 데이터를 방출하고 가입약관 종류를 기준으로 정렬된 자료구조인 Term_List 와 Term_BST, 회원 이름을 기준으로 정렬된 자료구조인 Name_BST 에 저장한다. Term_List 는 가입약관 종류별로 노드가 구성되며 각 노드는 가입약관 종류, 해당 가입약관의 회원 수, 해당 가입약관의 BST 포인터 정보를 가지고 있다. 입력된 가입약관의 순서대로 노드가 생성되어 연결되며 이미 존재하는 가입약관과 정보가 입력될 경우 해당 노드의 회원수를 증가시킨다. List 의 정보가 입력된 후 해당 가입약관의 BST 에서 노드 구성을 수행한다. 이어서 Term_BST 는 가입약관 종류별로 구성되며 각 BST 의 노드는 회원 이름, 나이, 개인정보 수집일자, 개인정보 만료일자 정보를 가진다. 개인정보 만료 일자는 개인정보 수집일자에 가입약관 별 개인정보 유효기간이 더해져 계산되며 각 BST 는 개인정보 만료일자 정보를 기준으로 정렬된다. 이때 모든 날짜는 31 일이 있는 것으로 가정하고 프로젝트를 진행하였다. 마지막으로 Name_BST 는 회원이름을 기준으로 정렬된 자료구조로, 각 노드는 회원 이름, 나이, 개인정보 수집일자, 개인정보 만료일자, 가입약관 종류 정보를 가진다.

프로젝트에서 구현해야 하는 흐름을 간략하게 살펴보면 다음과 같다.
LOAD 명령어는 텍스트 파일 "data.txt"에서 데이터 정보를 읽어올 수 있으며 데이터 파일에 정보가 존재하는 경우, 해당 정보를 Member_Queue 자료구조에 저장한다. 이어서 ADD 명령어는 Member_Queue 에 데이터를 직접 추가하기 위한 명령어로 회원이름, 나이, 개인정보 수집일자, 가입약관종류를 추가로 입력한다. QPOP 명령어는 Member_Queue 에서 데이터를 pop 하여 Term_List 와 Term_BST 를 구성한다. SEARCH 명령어는 Name_BST 에 저장된 회원정보를 찾아 출력하는 명령어이다. PRINT 명령어는 BST 에 저장된 데이터들을 출력하는 명령어이며, DELETE 명령어는 List 와 BST 에 저장된 데이터를 제거한다. 마지막으로 EXIT 명령어는 프로그램 상의 메모리를 해제하며 프로그램을 종료한다.

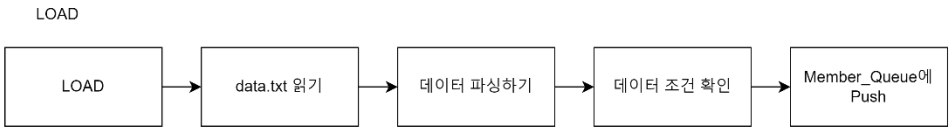
데이터 관리를 효과적으로 수행하고, 회원 정보를 다양한 기준으로 정렬 및 접근할 수 있도록 하는 개인정보 관리 프로그램을 구현하는 것을 목표로 하며, 전체적인 흐름과

명령어에 대한 추가적인 내용은 Flowchart 에서 설명할 것이며, Member_Queue, Term_List, Term_BST, Name_BST 에 대한 내용은 Algorithm 에서 설명하겠다.

2. Flowchart

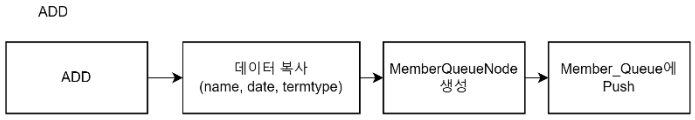


이번 프로젝트의 전체적인 흐름을 나타낸 Flowchart 이다. command.txt 를 통해 명령어를 입력 받고 이를 manager.h 에서 관리한다. 각각의 command 는 각 명령어에 맞는 동작을 취하며 위의 흐름도와 같이 동작한다. Manager.cpp 의 Run 함수에서 command.txt 파일을 파싱하여 command 를 추출한 후 command 와 명령어를 비교하여 각각에 맞는 동작을 한다. 모든 명령어는 성공 시 success 를 출력하며, 실패 시 Error code 를 log 에 출력한다. 명령어들은 동작을 마친 뒤 종료되지 않고 command 파일이 끝까지 읽힐 때까지 작동한다. 그에 대한 흐름은 log.txt 에서 직접 파악할 수 있으며 성공여부와 실패여부 역시 log 에 남는다.

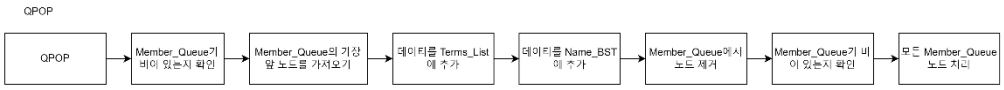


LOAD 명령어의 flow chart 이다. LOAD 는 data.txt 파일에 저장된 data 를 불러온다. 데이터 파일이 존재하지 않을 경우와 Member_Queue 에 데이터가 가득 찬 경우 에러메세지를 출력한다. 이후 data.txt 파일에서 한줄씩 읽어와 데이터를 추출한다. 이후 데이터의

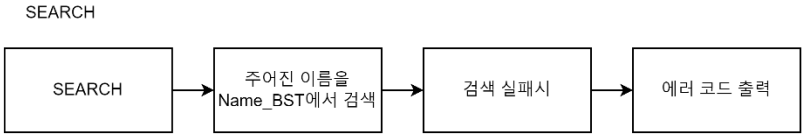
조건을 확인한다. 데이터의 조건은 다음과 같다. 회원 이름은 중복된 경우가 없으며, 공백문자 없이 소문자로 구성되고 20 자 이하의 길이를 가진다. 나이는 자연수 형태로 입력되며 10 에서 99 세의 범위를 가진다. 개인정보 수집일자는 xxxx-xx-xx 의 형식으로 입력된다. 가입약관 종류는 대문자 A , B, C, D 중 하나이다. 위의 데이터 조건을 모두 만족할 경우 Member_Queue 에 push 한다.



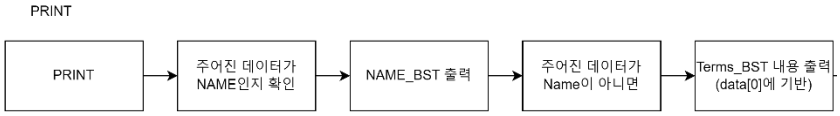
ADD 명령어의 flow chart 이다. 4 개의 인자를 Member_Queue 에 추가로 입력하는 명령어이며 이때 4 개의 인자는 각각 회원이름, 나이 개인정보수집일자, 가입약관종류에 해당한다. 이때 데이터가 하나라도 존재하지 않을 시 에러코드를 출력해주었다. 지역변수를 선언해서 name, date, termtype 데이터를 복사해주었다. 이때 age 를 복사하지 않은 이유는 age 는 int type 으로 선언되어 매개변수로 전달될 때 복사가 필요하지 않기 때문이다. 이후 MemberQueueNode 를 생성한 후 Member_Queue 에 push 한다.



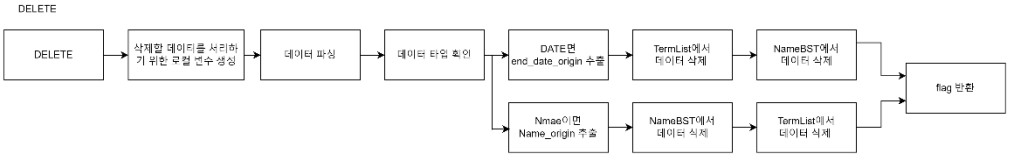
QPOP 명령어의 flow chart 이다. Member_Queue 의 front 부터 모든 노드를 POP 하여 해당 노드의 데이터로 자료구조를 구축하는 명령어이이다. 우선 MemberQueue 에 데이터가 존재하지 않을 시 에러코드를 출력해주었다. 이후 Member_Queue 의 모든 노드가 비어질 때 까지 다음의 과정을 반복한다. Member_Queue 의 가장 앞 노드를 가져와 데이터를 Term_List 와 Nambe_BST 에 추가한 후 Member_queue 에서 노드를 제거한다.



SEARCH 명령어의 flow chart 이다. 인자로 탐색하고자 하는 회원이름을 입력 받으며 Name_BST 에서 입력 받은 회원이름의 노드를 찾아 정보를 출력하는 명령어이다. 우선 찾고자 하는 회원정보를 Name_BST 에서 검색한다. 이때 찾고자 하는 회원정보가 존재하지 않으면 에러코드를 출력한다.



PRINT 명령어의 flow chart 이다. 인자로 Name 이 입력되는 경우 Name_BST 에 저장된 데이터를 출력하며, 인자로 가입약관종류 중 하나가 입력되는 경우 해당 가입약관 Terms_BST 의 저장된 데이터를 출력한다. 중위 순회, (in-order) 방식으로 BST 를 탐색하여 데이터를 출력하며 BST 에 데이터가 존재하지 않을 시 에러코드를 출력하였다.



DELETE 명령어의 flow chart 이다. 인자로 DATE 특정 일자가 입력되는 경우 모든 Term_BST 에서 해당 일자보다 개인정보만료일자가 이전인 모든 노드들을 제거해야 하므로 Term_List 에서 해당하는 가입약관 노드들을 삭제한 후 Name_BST 에서 데이터를 삭제하였다. 인자가 NAME 특정 회원이름으로 입력되는 경우에도 마찬가지로 Name_BST 에서 해당하는 이름을 지운 후 Term_List 에서도 데이터를 삭제하였다. 또한 삭제하고자 하는 회원 정보가 존재하지 않거나, 자료구조에 데이터가 존재하지 않을 시 에러 코드를 출력하였다.

3. Algorithm

- Member_Queue

우선 queue 는 선입선출(FIFO)의 데이터 구조로 요소를 추가할 때는 큐의 뒤에, 요소를 제거할 때는 큐의 앞에서 처리하는 자료구조이다. 구현한 동작에 대해 자세히 설명하겠다. 우선 큐 객체가 생성될 때 호출되며 크기가 100 인 동적 배열을 생성하여 큐를 초기화 하였으며 소멸자에서 큐 객체가 소멸될 때 동적으로 할당된 큐 배열 메모리를 해제하였다. 또한 frontIndex 와 rearIndex 를 -1 로 초기화하여 queue 가 비어 있음을 나타내었다. empty 함수는 큐가 비어있는지 확인하며 frontIndex 가 -1 이면 큐가 비었음을 의미한다. Full 함수는 큐가 가득 차있는지 확인한다. 큐의 시작과 끝이 연결된 원형 큐로 구성되어 있으며, (rearIndex + 1) % size == frontIndex 조건으로 큐가 가득 차 있는지 확인하였다. 이어서 push 함수는 큐에 새로운 요소를 추가하며, full 함수로 큐가 가득 차 있는지 확인하고 가득 차 있지 않다면 새 요소를 rearIndex 에 추가하고 realIndex 를 업데이트 하였다. Pop 함수는 큐에서 가장 앞에 있는 요소를 제거하고 반환한다. Empty 함수로 큐가 비어있는지 확인하고 비어있지 않다면 frontIndex 에서

요소를 제거하고 frontIndex 를 업데이트 하였다. 마지막으로 front 함수는 큐의 가장 앞에 있는 요소를 반환하며 empty 함수로 큐가 비어있는지 확인하고 비어 있지 않다면, frontIndex 위치의 요소를 반환하였다.

- Terms_List

링크드 리스트(Linked list)는 데이터 요소가 순서대로 연결된 선형 자료 구조로, 각 요소는 데이터와 다음 요소를 가리키는 포인터로 구성된다. 구현한 동작에 대해 자세히 설명하겠다. 우선 생성자에서 head 포인터를 초기화한다. 이 포인터는 가입약관 data 를 저장하는 링크드 리스트의 첫번째 노드를 가리킨다. 소멸자에서는 링크드리스트의 모든 노드와 해당 노드에서 사용된 BST(이진 검색 트리) 객체를 메모리에서 해제하였다. 각 노드에 연결된 BST 객체가 있다면 해당 객체를 먼저 삭제한 후 노드를 삭제하였다. 모든 노드를 삭제한 후 head 포인터를 nullptr 로 설정하였다. getHead 함수는 head 포인터를 반환하여 링크드 리스트의 첫번째 노드에 대한 접근을 제공한다. Insert 함수는 새로운 가입약관을 삽입하는 함수로 리스트를 검색하여 같은 용어가 있는지 확인하였다. 이때 같은 가입약관이 있다면 해당 가입약관의 BST 에 데이터를 삽입하며 회원수를 증가시키며, 같은 가입약관이 없을 경우 새로운 노드를 생성해서 다음 링크드 리스트의 끝에 추가하였다. Searchprint 함수는 특정 가입약관의 데이터를 검색하고 출력하는 함수이다. 해당하는 가입약관을 링크드 리스트에서 검색하고 해당 노드의 BST 데이터를 출력하였다. 마지막으로 delete 부분이다. Deletelist 함수는 특정 날짜와 이름에 해당하는 항목을 삭제하는 함수이며 링크드 리스트를 순회하며 해당 용어가 있는 BST 를 삭제하고 회원 수를 감소시켰다. 만약 해당 노드의 BST 가 더 이상 항목을 가지고 있지 않다면 해당 노드를 삭제하였다. 이어서 deletelistname 함수는 delete 명령어에서 NAME 과 특정 회원 이름이 들어올 때 따로 delete 를 진행하기 위해 만든 함수로, deletelist 함수와 동작이 유사하며 하나 다른 점은, flag 를 사용하여 삭제 여부를 추적하도록 한점이다.

- Term_BST

이진 탐색 트리(Binary Search Tree)는 데이터를 저장하는 계층적인 이진 구조로 왼쪽 서브트리의 모든 노드는 현재 노드보다 작은 값을 가지고, 오른쪽 서브트리의 모든 노드는 현재 노드보다 큰 값을 가지는 구조이다. 구현한 동작에 대해 자세히 설명하겠다. 우선 Insert 함수는 새로운 memberQueue 데이터를 BST 에 삽입한다. 이때 날짜를 기반으로 데이터를 정렬하도록 하였으며, 삽입할 노드를 생성하고 BST 에 추가하였다. insertNode 함수는 BST 에 노드를 삽입할 때 필요한 함수로 재귀적으로 호출되도록 구현하였다. 이 함수는 날짜에 따라 노드를 적절한 위치에 삽입한다. inorderPrint 함수는 BST 를 중위 순회하며 데이터를 출력하는 함수이다. 여기서 in-order 는 BST 의 순회 방법

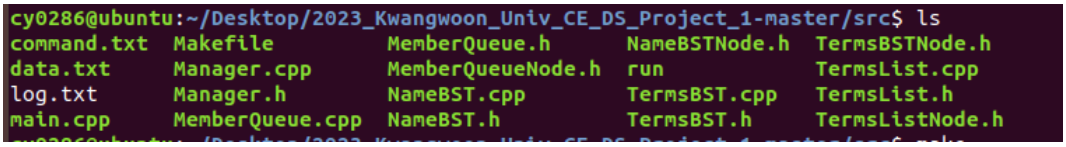
중 하나로 노드를 왼쪽 서브트리를 먼저 순회한 후 현재 노드를 방문하며 마지막으로 오른쪽 서브트리를 순회하는 순서를 의미한다. 따라서 데이터를 정렬된 순서대로 출력한다. Print 함수는 출력할 파일을 ofstream log 를 지정하여 BST 를 중위 순회하여 데이터를 파일에 출력한다. 마지막으로 delete 부분이다. Deletebst 함수는 end_date 와 Name 을 기준으로 노드를 삭제하는 함수이며, 노드를 찾고 해당 노드를 삭제한 후, 필요한 경우 부모 노드의 링크를 업데이트 하도록 하였다. 이때 부모노드보다 end_date 가 이전인 노드는 왼쪽, 같거나 이후인 노드는 오른쪽 서브트리에 위치하며 Delete 조건이 해당 end_date 보다 작은 data 를 모두 삭제하는 것이었기 때문에 가장 왼쪽으로 가서 날짜를 비교하는 식으로 코드를 구현하였다. 이어서 deleteBSTname 함수는 위에서도 언급했듯이, Date 와 특정 일자가 들어왔을 때 이름을 delete 하기 위해 만든 함수로, 이름을 기반으로 노드를 찾아 삭제하고, 필요한 경우 부모 노드의 링크를 업데이트 하였다. 자세한 내용은 NameBST 의 delete 부분과 유사하므로 그 부분에서 추가적으로 설명하겠다. 마지막으로 findMinValueNode 함수는 주어진 서브트리에서 최솟값을 가진 노드를 찾는 함수로 BST 의 왼쪽 하위 트리에서 최솟값을 찾도록 하였다.

- Name_BST

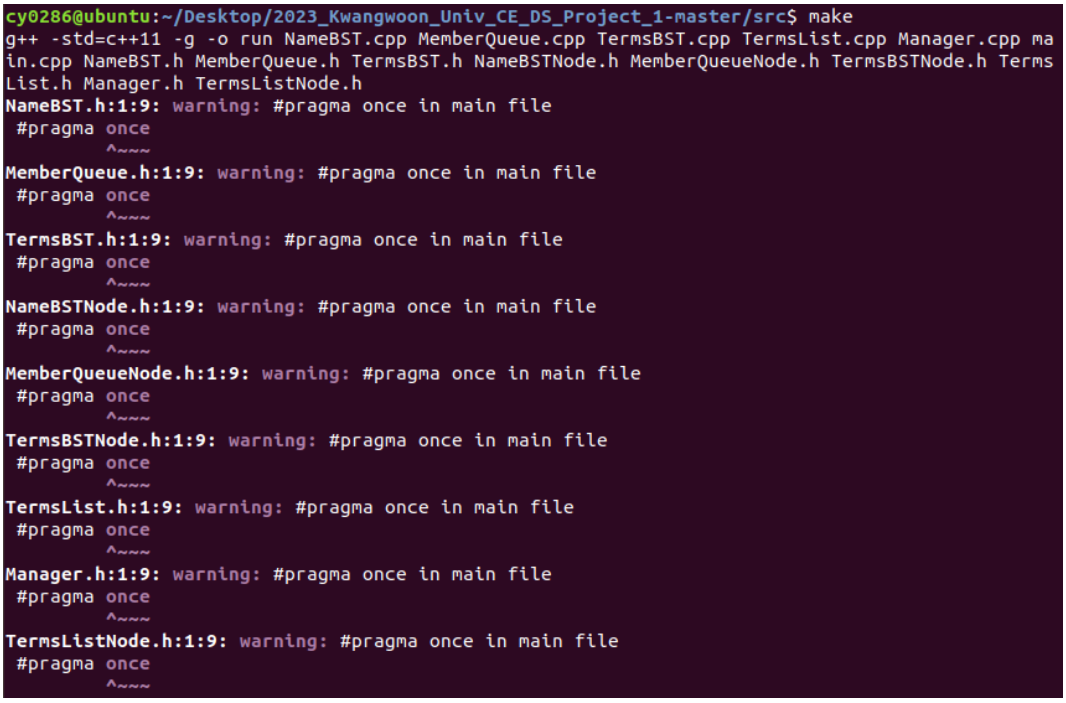
우선 insert 함수는 새로운 memberQueueNode 데이터를 BST 에 삽입한다. 이때 데이터는 이름을 기준으로 정렬되며 BST 를 순회하며 적절한 위치를 찾아 노드를 삽입하였다. 이때 Name_BST 의 insert 함수에 중 트리를 순회하며 노드의 삽입 위치를 찾는 부분에 대해 더 자세히 설명하겠다. 우선 현재 노드를 나타내는 임시변수와 루트 노드를 초기화한 후 임시변수와 삽입할 노드의 이름을 비교하며 다음의 동작을 수행하였다. 만약 삽입할 노드의 이름이 임시변수의 이름보다 작거나 같다면 왼쪽 서브트리로, 크다면 오른쪽 서브트리로 이동한 후 비교하여 임시변수가 NULL 이면 현재 위치에 새로운 노드를 삽입하고 함수 실행을 종료한다. 이때 임시변수가 NULL 이 아니라면 다음 노드로 이동하여 다시 비교를 시작한다. 이렇게 반복하다가 올바른 위치가 결정되면, 해당 위치에 새로운 노드를 삽입하며 이때 왼쪽 또는 오른쪽 자식 노드로 설정되도록 하였다. 이어서 search 함수는 주어진 data 와 일치하는 노드를 찾아 해당 노드 정보를 출력하도록 구현하였으며 중위 순회를 하며 일치하는 이름을 찾아주었다. inorderPrint 함수와 print 함수는 Term_BST 의 동작 방식과 똑같다. Delete 함수에 대해 자세히 설명해보겠다.우선 deletename 함수는 재귀적으로 호출되며 최초 호출은 BST 의 루트 노드부터 시작된다. 현재 노드의 이름과 지우려는 이름을 비교하여 비교 결과에 따라 다음을 수행하도록 하였다. 만약 0 보다 작다면 name 이 현재 노드의 이름보다 작음을 의미하므로 왼쪽 서브트리로 이동해 name 을 검색하고, 0 보다 크다면

name 이 현재 노드의 이름보다 크다는 것을 의미하므로 오른쪽 서브트리로 이동해 name 을 검색하였다. 만약 0 이면 name 과 현재 노드의 이름이 일치하므로 해당 노드가 삭제 대상이 되도록 하였다. 이어서 일치할 경우 3 가지 경우로 나누었는데, 먼저 삭제할 노드가 자식 노드를 가지지 않는 경우, 해당 노드를 삭제하고 NULL 을 반환하여 부모 노드의 링크를 갱신하였다. 이렇게 되면 삭제된 노드의 부모 노드의 왼쪽 또는 오른쪽 자식 링크를 NULL 로 설정해야 한다. 두번째 경우는 삭제할 노드가 오른쪽 자식 노드만 가지는 경우이다. 이경우엔 해당 노드를 삭제하고 오른쪽 자식 노드로 대체하고 삭제된 노드의 부모 노드와 대체된 노드를 연결하였다. 마지막 경우는 삭제할 노드가 두개의 자식을 가진 경우이다. 이경우에는 삭제할 노드를 대체할 노드, 삭제할 노드의 오른쪽 서브트리에서 가장 작은 값을 가지는 노드를 findMinvalueNode 함수를 이용해 찾았다. 이후 대체할 노드의 데이터를 삭제할 노드로 복사한 후 대체할 노드를 재귀적으로 생산하고 삭제 대상의 오른쪽 서브트리를 새로운 서브트리로 갱신하였다. 이렇게 되면 삭제된 노드의 대체로 대체된 노드를 반환하여 부모 노드의 링크를 업데이트 하여 특정 이름을 가진 노드를 삭제할 수 있다.

4. Result Screen



프로젝트 파일에서 ls 명령어로 프로젝트 폴더 안에 있는 파일들을 확인하였다.



Make 명령어를 통해 컴파일을 한 사진이다. 스켈레톤 코드 안에 #pragma once 가 있어 warning 이 뜨는 것 이외에는 코드가 잘 컴파일 된 것을 확인할 수 있다.

```
cy0286@ubuntu:~/Desktop/2023_Kwangwoon_Univ_CE_DS_Project_1-master/src$ cat command.txt
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
EXITcy0286@ubuntu:~/Desktop/2023_Kwangwoon_Univ_CE_DS_Project_1-master/src$
```

Cat 명령어를 사용해 주어진 command.txt 파일에 있는 명령어를 출력해보았다.

```
cy0286@ubuntu:~/Desktop/2023_Kwangwoon_Univ_CE_DS_Project_1-master/src$ cat data.txt
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 Acy0286@ubuntu:~/Desktop/2023_Kwangwoon_Univ_CE_DS_Project_1-master/src$ S
```

Cat 명령어를 사용해 data.txt 파일에 있는 data 를 출력해보았다.

```
cy0286@ubuntu:~/Desktop/2023_Kwangwoon_Univ_CE_DS_Project_1-master/src$ ./run
Queue is empty.
```

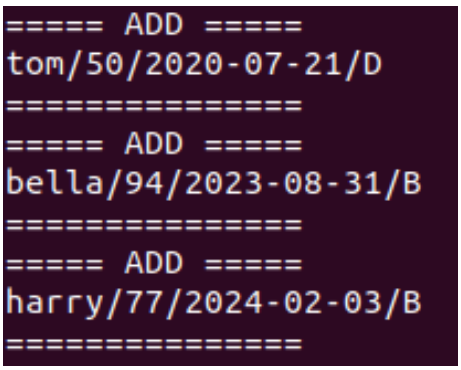
실행하였을 때 결과화면이다. Queue 가 비어 있는지 확인하는 출력문이 있어 queue is empty. 가 출력되는 것을 볼 수 있다.

```
cy0286@ubuntu:~/Desktop/2023_Kwangwoon_Univ_CE_DS_Project_1-master/src$ cat log.txt
```

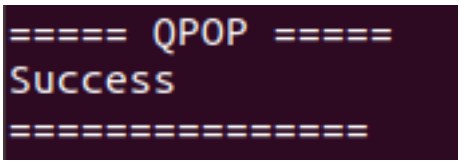
Cat 명령어를 사용해 log.txt 파일에 출력된 결과를 확인해보았다.

```
===== 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
===============
```

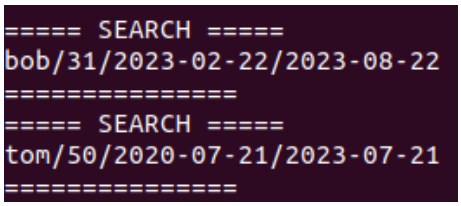
LOAD 명령어 부분이다. Data.txt 파일의 데이터들이 잘 load 된 것을 확인할 수 있다.



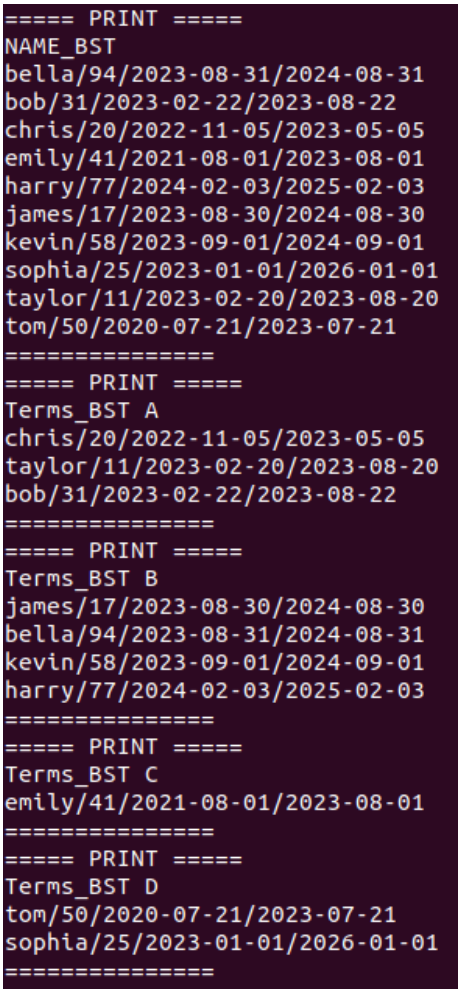
ADD 명령어 부분이다. Command.txt 파일을 바탕으로 성공적으로 데이터가 추가된 것을 확인할 수 있다.



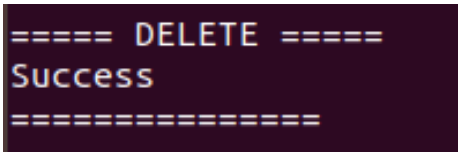
QPOP 명령어 부분이다. Success 가 뜬 것을 통해 Member_Queue 에서 데이터를 POP 하여 Term_List 와 Term_BST 와 Name_BST 의 해당 노드의 데이터로 성공적으로 구축된 것을 알 수 있다.



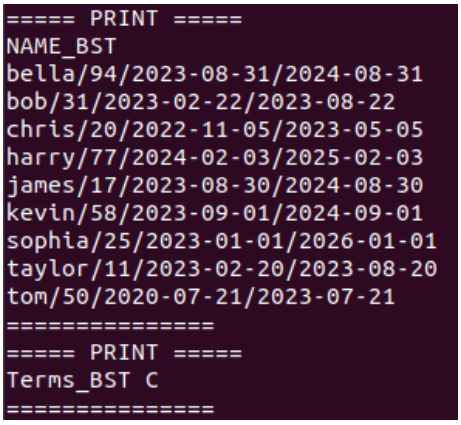
SEARCH 명령어 부분이다. Command.txt 파일을 바탕으로 Name_BST 에 저장된 회원정보를 성공적으로 찾아 출력한 것을 확인할 수 있다.



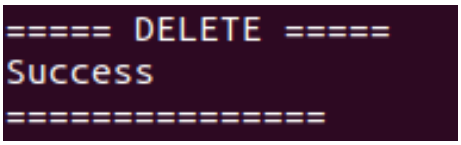
PRINT 부분이다. 명령어와 함께 NAME 이 입력되는 경우 저장된 이름들이 사전적 순서로 출력된 것을 확인할 수 있다. 이어서 가입약관종류 중 하나가 입력되는 경우 해당하는 가입약관부분이 가입약관만료일자를 기준으로 정렬되어 출력되는 것을 확인할 수 있다.



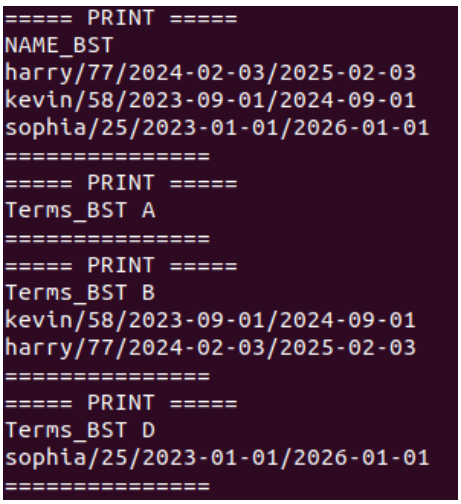
DELETE 부분이다. Command.txt 를 바탕으로 NAME emily 를 지워야 하며, Success 가 뜨는 것을 통해 DELETE 가 성공적으로 된 것을 알 수 있다.



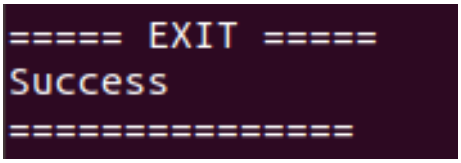
DELETE 후 PRINT 로 잘 지워졌는지 확인하는 부분이다. Emily 를 지웠으므로 Name 을 출력했을 때 emily 가 없는 것을 확인할 수 있으며, emily 는 가입약관 C 에 해당하므로 Term_BST C 를 출력했을 때 해당 부분에서도 emily 가 잘 지워진 것을 확인할 수 있다.



DELETE 부분이다. Command.txt 를 바탕으로 DATE 2024-09-01 보다 만료일자가 적은 모든 date 를 지워야 하며, Success 가 뜨는 것을 통해 DELETE 가 성공적으로 된 것을 알 수 있다.



DELETE 후 PRINT 로 잘 지워졌는지 확인하는 부분이다. 만료일자가 2024-09-01 보다 작은 모든 date 가 지워졌으며 Term_BST 와 Name_BST 둘 다 모두에서 잘 지워진 것을 알 수 있다.



마지막으로 EXIT 부분이다. 프로그램이 성공적으로 종료되었음을 알 수 있다.

5. Consideration

우선 이번 프로젝트를 진행하면서 작년에 진행하였던 1 차 프로젝트를 할 때 보다 성장하였다는 걸 느꼈다. 그 당시에는 명령어 구현에 있어 Load, Add Print 를 겨우 구현하여 냈었던 것 같은데, 이번 프로젝트에서는 끝까지 구현도 하고 결과도 성공적으로 나오는 것을 확인하며 뿌듯했다. 개념적으로 BST, Linked List, Queue 를 알고 있었지만, 직접 구현하려고 해보니 생각보다 헷갈리는 부분이 많아 그림도 여러 개 그려보고 경우의 수도 생각해보며 코드를 구현하려고 했던 것 같다. 이 과정에 있어서 교수님께서 코드를 다 구현해봐야 실력도 늘고 이해도 된다고 말씀하셨던 것이 이해가 됐다.

프로젝트 설계에 있어 힘들었던 점은 delete 할 때 Term_List 와 Term_BST, Name_BST 세 곳에서 모두 삭제되게 구현해야 한다는 점이었다. Name_BST 에서는 잘 삭제가 되었지만, Term 부분을 Print 해보면 emily 가 계속 남아있어 애를 먹었던 것 같다. 이를 해결하기 위해 name 을 delete 하는 코드를 추가하였다. Manager.cpp 파일에서 origin_Name 과 origin_end_date 변수를 설정하여 인자로 넘겨준 후 data 를 삭제하게끔 구현하여 해결할 수 있었다.

BST 부분에서 재귀적으로 함수를 구현하였지만, 재귀적으로 구현했다 보니 제한되는 점도 많았고, 동작 과정을 생각해내는 것에서 계속 실수가 있어 시간을 많이 뺏겼다. 코드가 여러 cpp 파일로 나누어져 있어 흐름을 그려 나가고 생각해내는 것도 조금 어려웠다. 또한 flag 를 설정해주었는데 *flag++; 라고 해준 부분에서 계속 core aborted 오류가 떴다. 이부분을 *flag = 1 로 설정해주니 오류는 해결됐지만 이러한 자잘한 동작 방식이 이해 안 되는 부분도 많았던 것 같다.

원래는 github 에 코드를 올려두어 코드가 날아가는 것을 막았지만, 운영체제 수업을 같이 듣다 보니 스냅샷 기능을 사용하여 코드를 관리하였다.