Data Structure Project #1

학과: 컴퓨터정보공학부

학번: 2022202065

이름: 박나림

1. Introduction

개인정보 관리 프로그램을 구현하는 프로젝트로, 이진 탐색 트리(Binary Search Tree), 연결 리스트(Linked List), 큐(Queue) 자료구조를 이용한다. command.txt 파일에 저장되어 있는 명령어들을 차례대로 읽어오면서 그에 맞는 명령들을 수행하는 방식이다.

'LOAD'명령을 읽으면 data.txt 파일에 저장되어 있는 회원 정보를 읽어와 Member_Queue를 구축한다. 이때 회원 정보는 회원 이름, 나이, 개인정보 수집일자, 가입약관 종류(A: 6개월, B: 12개월, C: 24개월, D: 36개월의 개인정보 보관 유효기간)를 가진다.

'ADD'명령을 읽으면 해당 라인의 회원 정보를 마찬가지로 큐에 이어서 저장한다.

'QPOP'명령 시, 저장되어 있는 데이터를 순차적으로 방출하여 하나의 연결리스트와 두 개의 트리를 구축하는데, 가입약관종류를 기준으로 하는 Terms_List, Terms_BST와 회원이름을 기준으로 하는 Name_BST가 있다. Terms_List는 각 노드 별로 가입약관종류, 해당 가입약관의 회원 수, 해당 가입약관의 BST를 가리키는 포인터 정보를 가지게 된다. 입력된 순서대로 노드가 연결되며, 같은 가입약관을 가진 노드가 추가되는 경우 연결하지 않고 회원 수만 증가시키는 방식으로 한다. 이러한 연결리스트를 구축한 후, 각각의 가입약관 종류에 따른 트리를 구축한다. 이때 Terms_BST는 각 노드 별로 회원이름, 나이, 개인정보 수집일자, 개인정보 만료일자 정보를 가지며, 개인정보 만료일자를 기준으로 이전인 노드는 왼쪽, 이후인 노드는 오른쪽 서브 트리에 정렬된다. Name_BST에서는 각 노드 별로 회원이름, 나이, 개인정보 수집일자, 개인정보 만료일자, 가입약관종류 정보들을 가지며 회원이름을 기준으로 사전적 순서가 이전인 노드는 왼쪽, 이후인 노드는 오른쪽 서브 트리에 정렬된다.

'SEARCH'명령 시, 해당 라인의 이름 정보에 해당하는 노드를 Name_BST에서 탐색한 후 해당 회원의 정보를 출력한다.

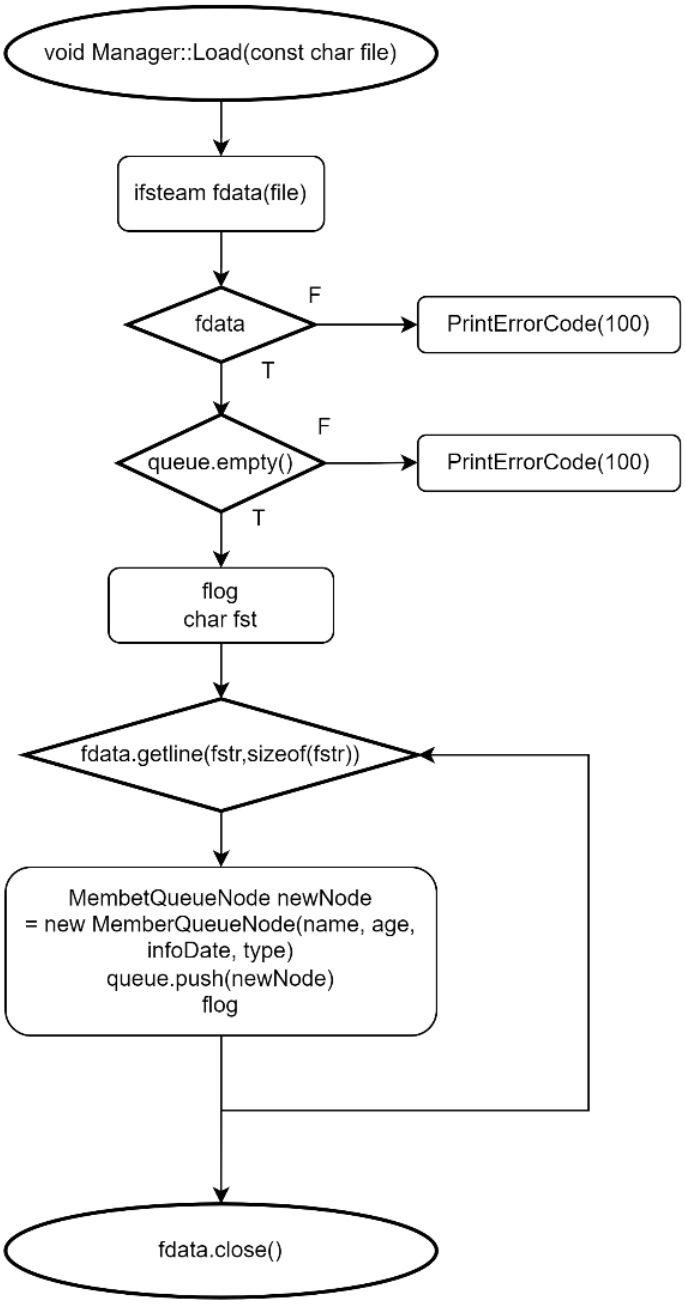
'PRITN'명령이 들어오면 해당 라인의 추가 정보를 기준으로 가입약관종류면 해당 가입약관의 Terms_BST의 정보들을, NAME이면 Name_BST의 정보들을 출력한다. 중위 순회(in-order)방식으로 탐색하여 순서대로 출력하는 방식으로 한다.

'DELETE'명령이면 해당 라인의 추가 정보들을 기준으로 연결리스트와 트리에 저장된 데이터들을 제거하도록 한다. 특정 일자가 들어오면 Terms_BST에서 해당하는 일자보다 개인정보 만료일자가 이전인 모든 노드들을 삭제한다. 이때 Terms_List에서 해당하는 가입약관 노드의 회원 수를 감소시키고, 0이 되는 노드는 마찬가지로 삭제한다. 또한 해당되는 노드들은 Name_BST에서도 삭제한다. 마찬가지로 특정 이름이 들어오면 Name_BST에서 해당 노드를 삭제하고, Terms_BST와 Terms_List에서도 삭제하는 연산을 수행시킨다.

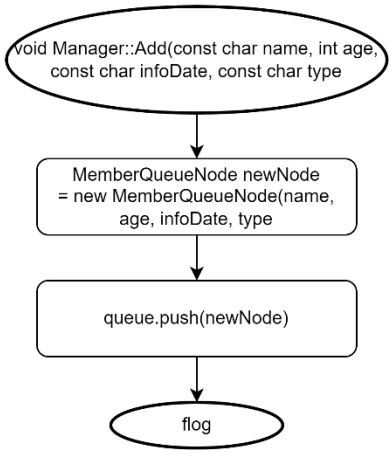
마지막으로 'EXIT'명령을 받아오면 프로그램 상의 메모리들을 해제시키며 프로그램을 종료하는 것으로 마친다. 모든 명령어들을 성공 시와 실패 시 안내 코드들을 출력한다.

2. Flowchart

1) LOAD

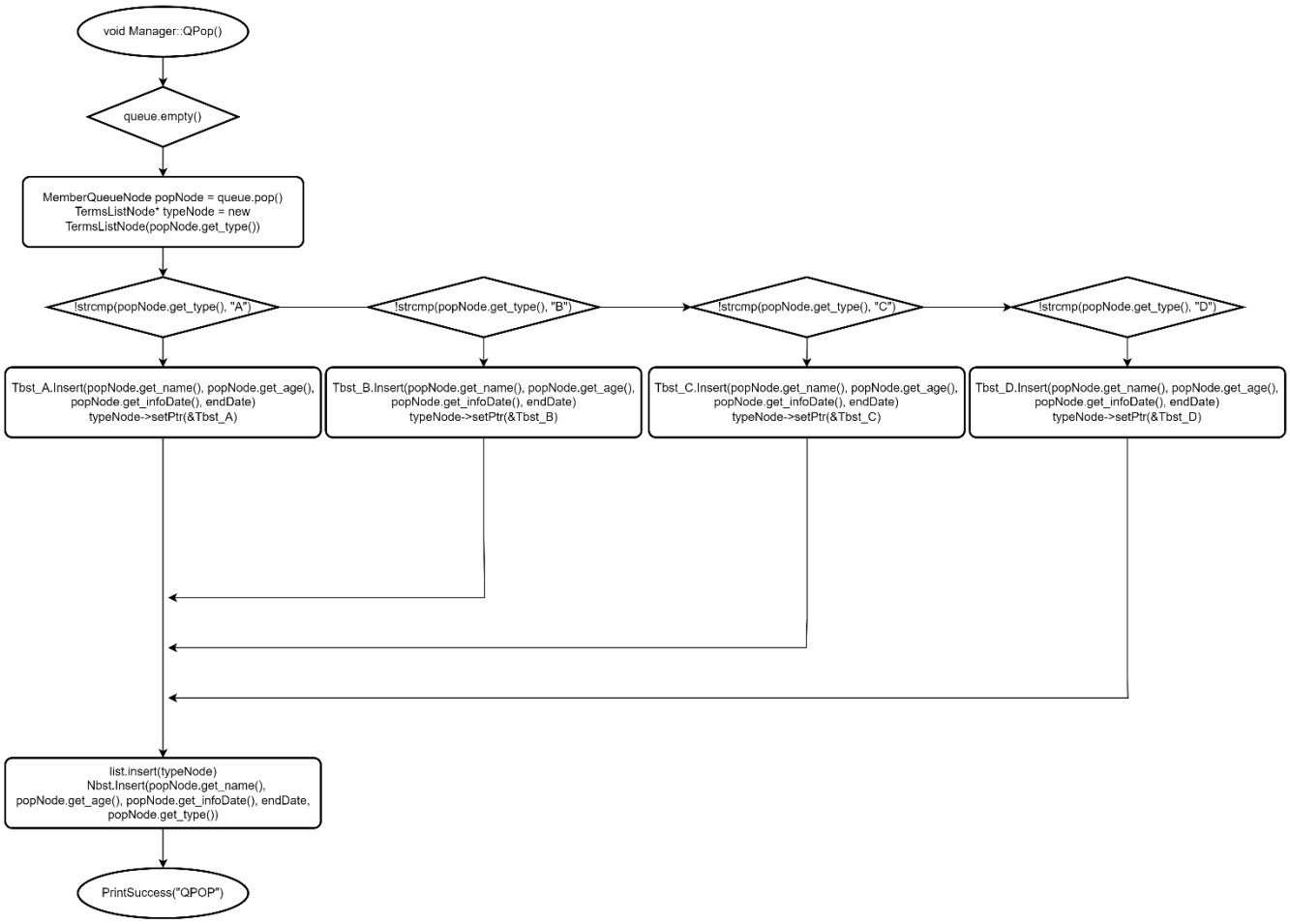


2) ADD

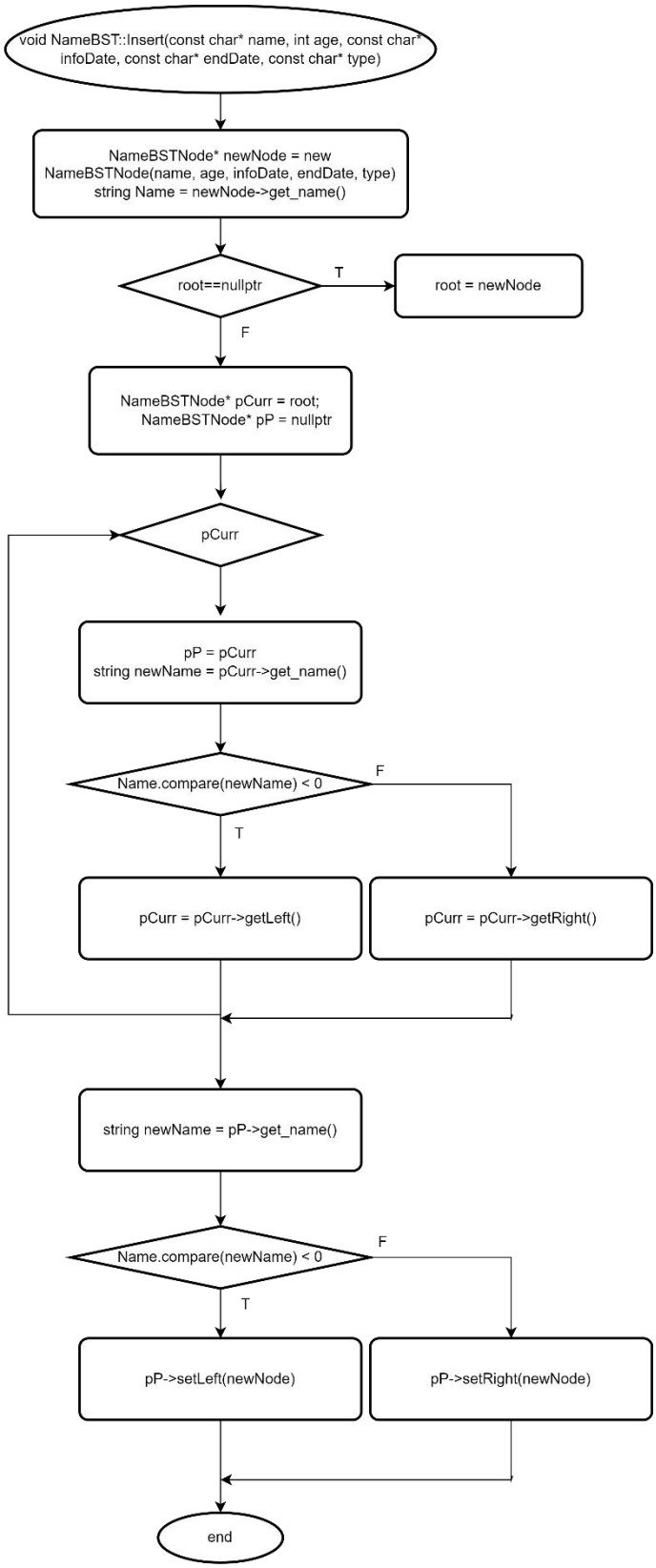


3) QPOP

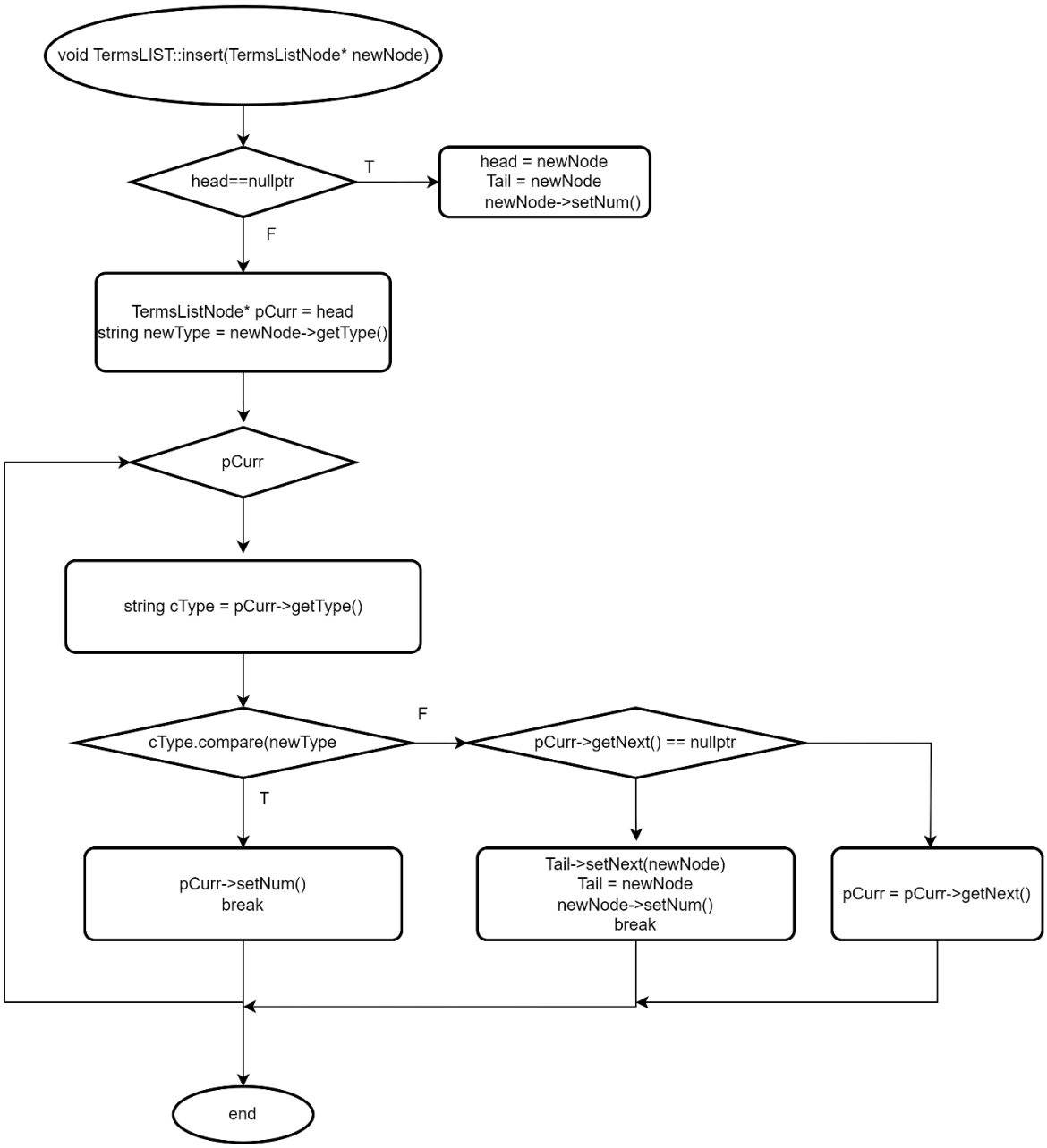
-manager



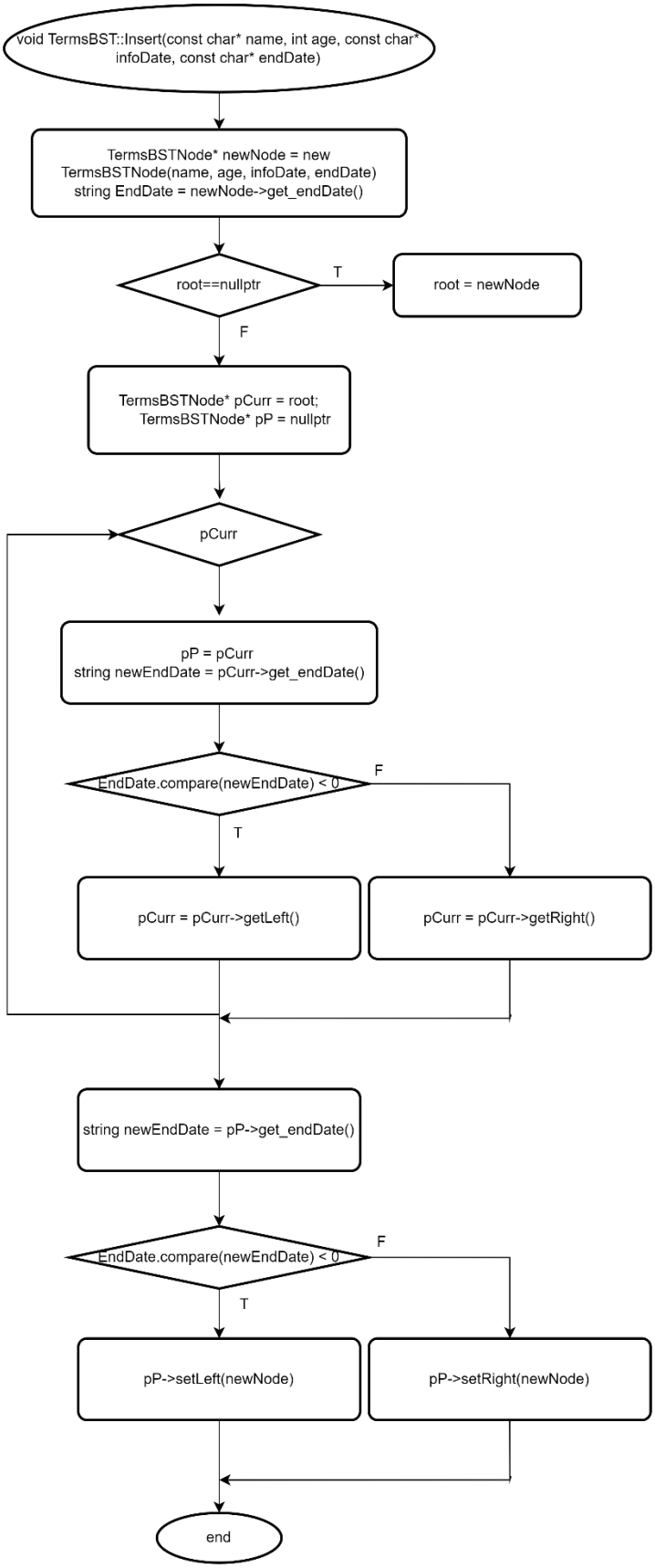
-Name_BST


void NameBST::Insert(const char* name, int age, const char* infoDate, const char* endDate, const char* type)
NameBSTNode* newNode = new NameBSTNode(name, age, infoDate, endDate, type)
string Name = newNode->get_name()
root==nullptr
T
root = newNode
F
NameBSTNode* pCurr = root;
NameBSTNode* pP = nullptr
pCurr
pP = pCurr
string newName = pCurr->get_name()
Name.compare(newName) < 0
F
T
pCurr = pCurr->getLeft()
pCurr = pCurr->getRight()
string newName = pP->get_name()
Name.compare(newName) < 0
F
T
pP->setLeft(newNode)
pP->setRight(newNode)
end

-Terms_List

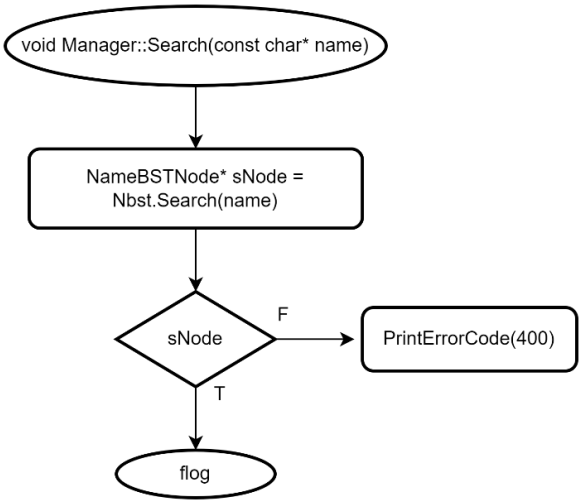

void TermsLIST::insert(TermsListNode* newNode)
head==nullptr
T
head = newNode
Tail = newNode
newNode->setNum()
F
TermsListNode* pCurr = head
string newType = newNode->getType()
pCurr
string cType = pCurr->getType()
cType.compare(newType
F
pCurr->getNext() == nullptr
T
pCurr->setNum()
break
Tail->setNext(newNode)
Tail = newNode
newNode->setNum()
break
pCurr = pCurr->getNext()
end

-Terms_BST

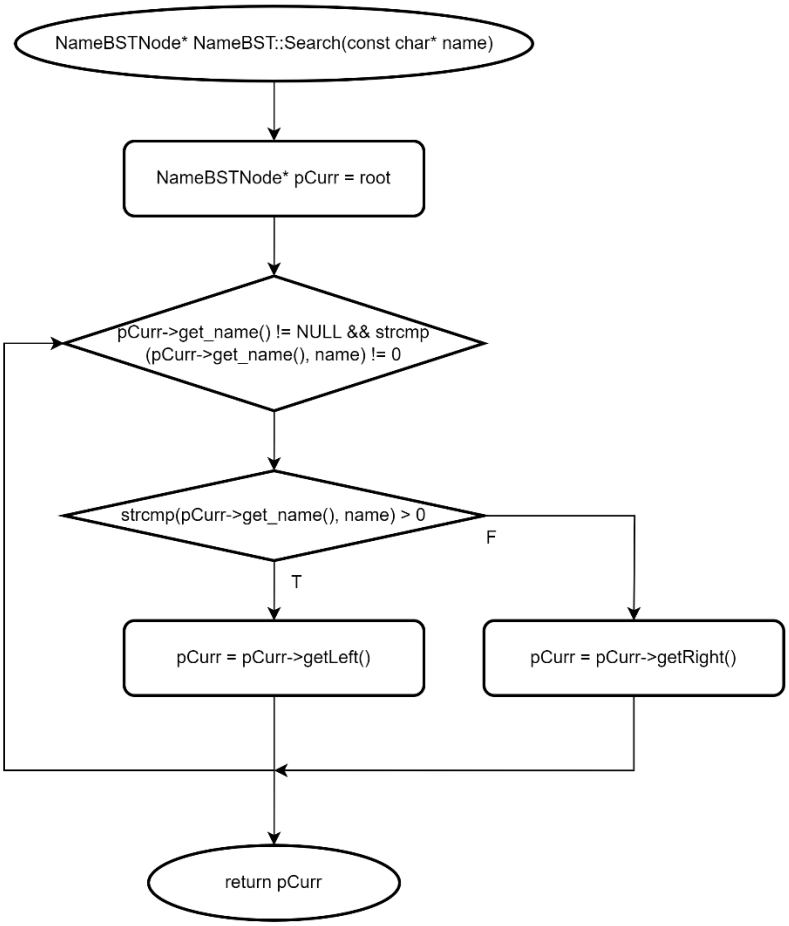

void TermsBST::Insert(const char* name, int age, const char* infoDate, const char* endDate)
TermsBSTNode* newNode = new TermsBSTNode(name, age, infoDate, endDate)
string EndDate = newNode->get_endDate()
root==nullptr
T
root = newNode
F
TermsBSTNode* pCurr = root;
TermsBSTNode* pP = nullptr
pCurr
pP = pCurr
string newEndDate = pCurr->get_endDate()
EndDate.compare(newEndDate) < 0
F
T
pCurr = pCurr->getLeft()
pCurr = pCurr->getRight()
string newEndDate = pP->get_endDate()
EndDate.compare(newEndDate) < 0
F
T
pP->setLeft(newNode)
pP->setRight(newNode)
end

4) SEARCH

-manager

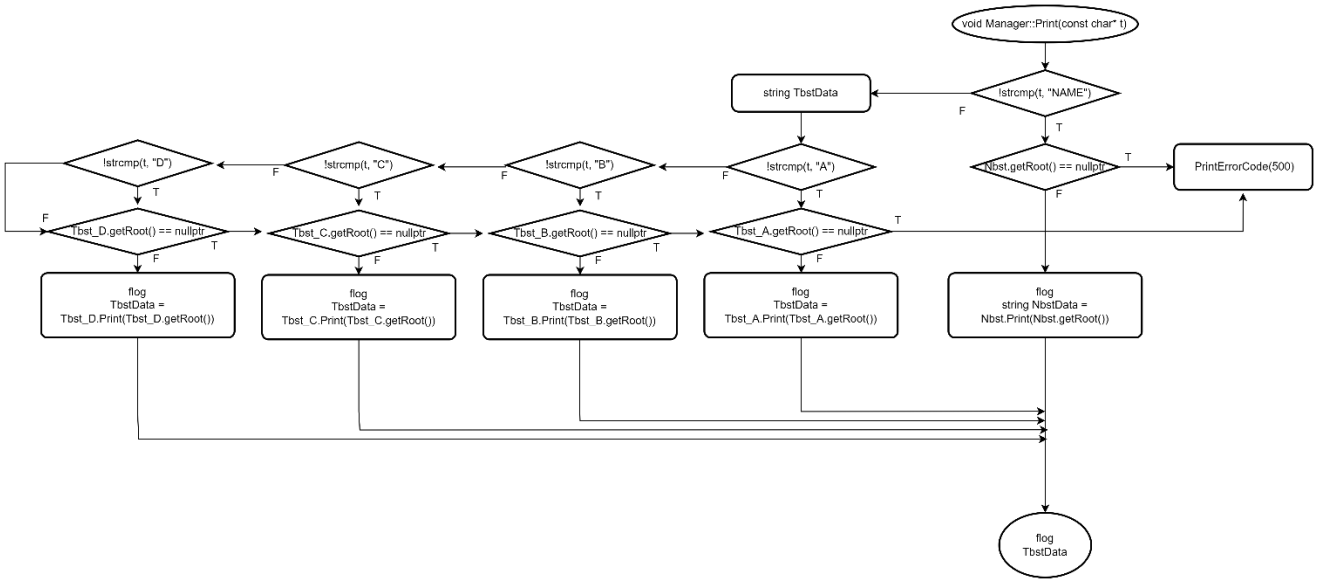


-Name_BST

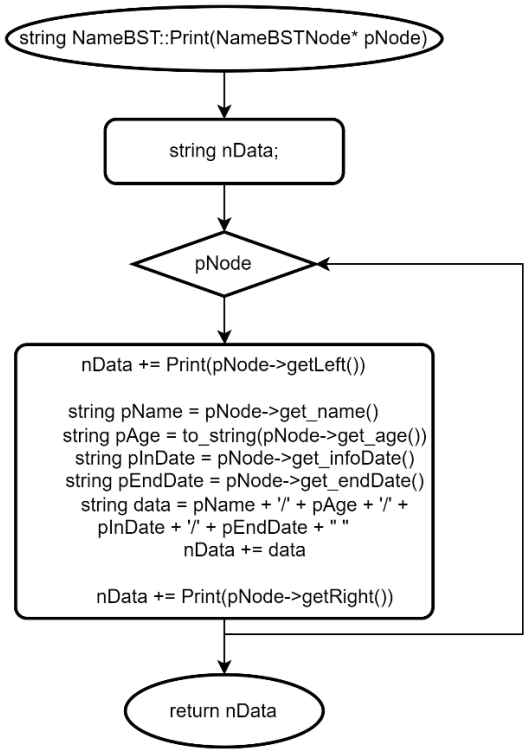


5) PRINT

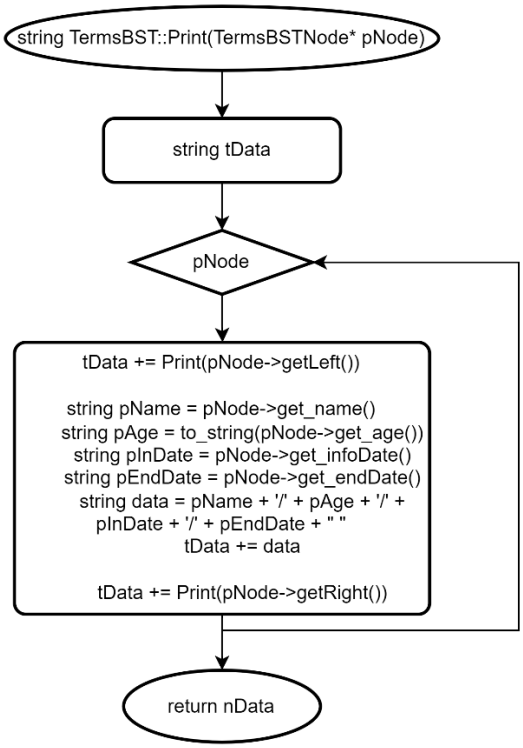
-manager



-Name_BST

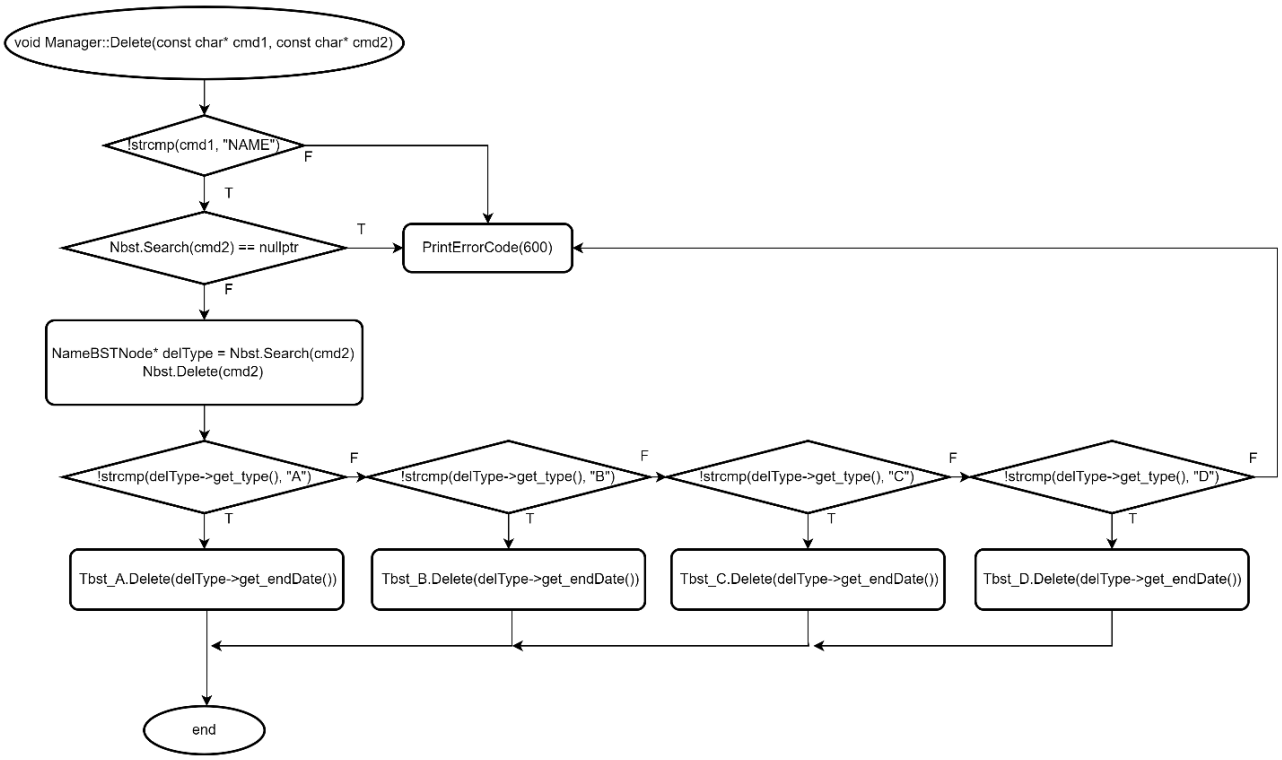


-Terms_BST

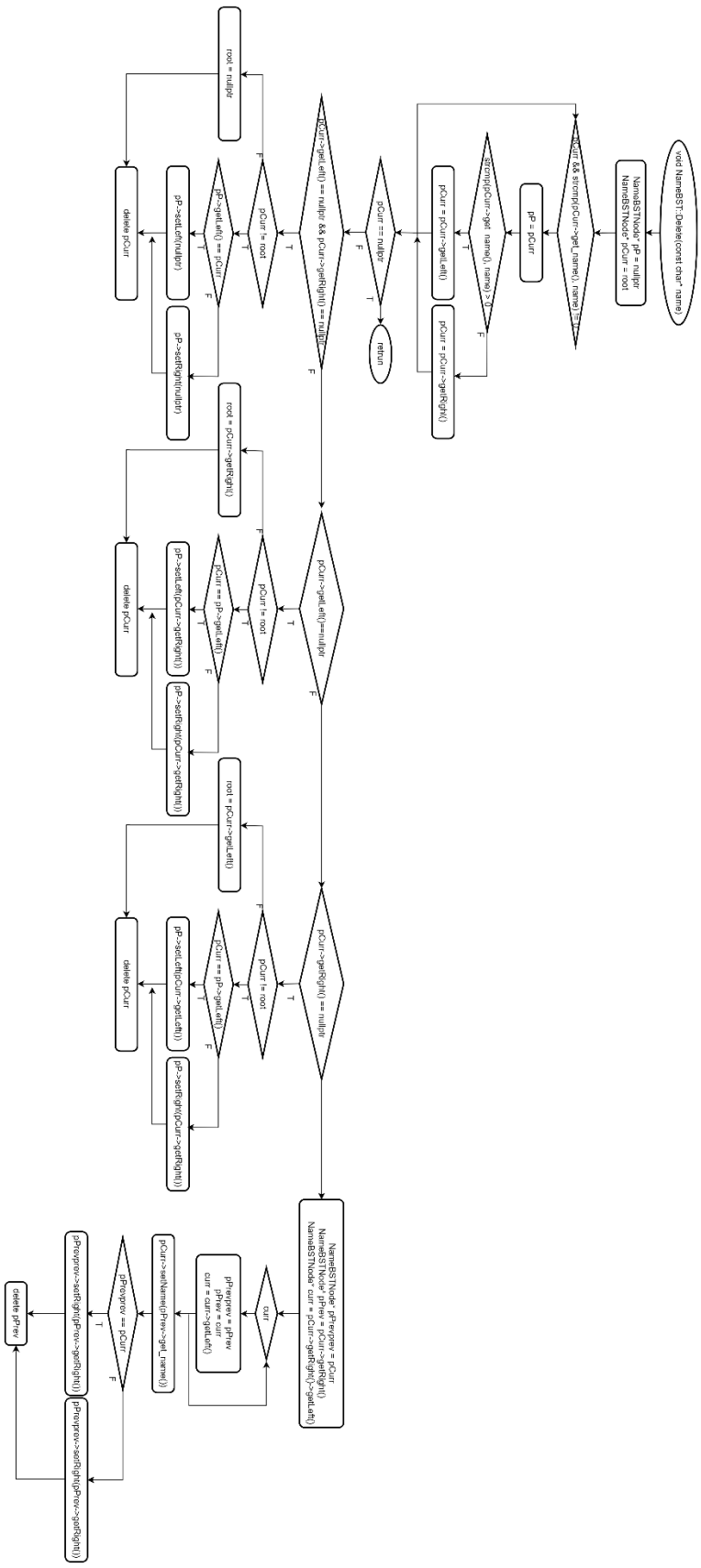


6) DELETE

-manager



-Name_BST


void NameBST::Delete(const char* name)
NameBSTNode* pP = nullptr
NameBSTNode* pCurr = root
pCurr && strcmp(pCurr->getL_name(), name) != 0
pP = pCurr
strcmp(pCurr->get_name(), name) < 0
pCurr = pCurr->getLeft()
pCurr = pCurr->getRight()
pCurr == nullptr
return
pCurr->getLeft() == nullptr && pCurr->getRight() == nullptr
pCurr != root
pP->getLeft() == pCurr
pP->setLeft(nullptr)
pP->setRight(nullptr)
delete pCurr
root = nullptr
pCurr->getLeft()==nullptr
pCurr == pP->getLeft()
pP->setLeft(pCurr->getRight())
pP->setRight(pCurr->getRight())
root = pCurr->getRight()
pCurr->getRight() == nullptr
pP->setLeft(pCurr->getLeft())
pP->setRight(pCurr->getLeft())
root = pCurr->getLeft()
NameBSTNode* pPrevprev = pCurr
NameBSTNode* curr = pCurr->getRight()->getLeft()
curr
pPrevprev = pPrev
curr = curr->getLeft()
pCurr->setName(pPrev->getL_name())
pPrevprev == pCurr
pPrevprev->setRight(pPrevprev->getRight()->getRight())
pPrevprev->setLeft(pPrevprev->getRight()->getRight())
delete pPrev

3. Algorithm

1) Insert

BST에서 이용한 삽입 알고리즘으로, Insert함수에서 회원정보들을 인자로 받아 함수 안에서 해당 BST의 newNode를 생성하는 것으로 시작한다. NameBST인 경우에는 newNode의 get_name()함수를 이용하여 이름 정보만 따로 string형으로 저장하여 순서를 비교하였고, TermsBST면 개인정보 만료일자만 따로 저장하여 비교하였다.

이렇게 정보들을 저장한 뒤, 크게 if else문으로 하여 먼저 해당 트리의 root가 nullptr인지 조사한다. 처음 생성된 트리일 경우 조건에 맞으므로 root에 newNode를 저장한다. 두번째 이상부터는 else문으로 넘어가서 현재 노드를 가리키는 pCurr을 생성하여 root를 저장하고, 부모 노드를 가리키는 pP를 생성하여 nullptr로 초기화 시킨다. 그 후 while문으로 pCurr이 nullptr을 가리키지 않는 동안 반복하며 트리에 들어갈 자리를 찾는다. pP에 pCurr을 저장하고, pCurr 노드의 회원 정보를 string으로 저장하여 만약 새 노드의 정보가 현재 노드의 정보보다 앞서는 경우, pCurr은 왼쪽 자식으로 이동하며, 뒤쪽인 경우엔 오른쪽 자식 노드로 이동한다. 자리를 찾아 while문이 종료된 경우, pP는 현재 찾은 자리의 부모 노드를 가리키고 있는 상태이므로 pP의 정보와 비교하여 앞서면 pP의 왼쪽 자식으로 newNode를 set하고 뒤쪽이면 오른쪽 자식 노드로 set하며 끝낸다.

List에서 이용한 삽입 알고리즘에서는, 단일 연결리스트로 노드를 추가하여 연결하는 방식으로 진행한다. 이때 head와 tail 포인터를 이용하여 tail을 옮겨가며 노드 추가를 한다. 처음으로 노드가 들어오는 경우에는 head와 tail 둘 다 newNode로 설정하며 회원 수를 증가시키기 위해 newNode의 setNum()함수를 이용하여 해당 가입약관의 회원 수를 증가시킨다. 두번째 이상부터는 pCurr을 생성하여 head로 초기화 시킨 뒤, 가입약관을 string형으로 순서를 비교하며 while문으로 들어갈 자리를 찾는다. 만약 이미 같은 종류의 노드가 존재한다면, 회원 수만 증가시키고 노드를 연결시키지 않는다. 새로운 노드라면 tail 포인터가 이동하면서 추가하고 회원 수를 증가시킨다.

2) Search

NameBST의 Search함수에서 인자로 회원 이름 정보를 받고, 함수 내에서 마찬가지로 현재 노드를 가리키는 pCurr을 생성하여 root값으로 초기화를 시킨다. 그 다음 while문으로 트리를 탐색하며 맞는 노드를 찾아낸다. 이때 while문의 조건은 pCurr의 get_name()이 NULL을 가리키지 않으며 get_name()이 검색할 이름과 일치하지 않을 동안 반복하는 것으로 한다. 또한 반복하는 동안 if else문으로 2가지 경우로 나누어서 탐색한다. pCurr의 이름보다 검색할 이름의 사전 순이 앞서는 경우 왼쪽 자식으로 이동하며, 뒤쪽인 경우에 오른쪽 자식으로 이동한다. while문이 종료되면 pCurr을 반환한다. 이런 방식으로 Search

를 하여, Manager에서 pCurr을 받으면 pCurr의 값이 nullptr일 때 이름을 찾지 못했다는 뜻이 되고 아니면 이름을 찾았다는 뜻이 된다.

3) Print

해당 트리의 root 노드를 인자로 Print함수에 전달하는 것으로 시작하여, 우선 string형으로 데이터를 저장할 변수를 생성한다. 그 후 if문으로 root 노드가 nullptr이 아닌 경우에만 중위 순회 방식으로 트리를 탐색하여 출력한다. 중위 순회 방식을 이용하면 트리에 이름이나 날짜 순서대로 저장한 것이 그대로 출력 된다. 그러기 위해서는 먼저 왼쪽 끝까지 내려갔다가 노드를 방문 처리하고, 끝까지 간 경우 그 부모 노드로 되돌아 가서 방문하고 오른쪽 자식으로 내려가서 방문하고, 다시 되돌아가서 방문하는 형식으로 출력해야 한다. 이 순서대로 진행하기 위해 먼저 왼쪽 자식으로 내려갈 때마다 재귀적으로 반복하며 Data에 저장하고, 중간에 회원 정보들을 출력 포맷을 따라 Data에 string으로 저장한 다음 마지막으로 오른쪽 자식으로 재귀적으로 내려가며 Data에 이어서 저장하는 식으로 진행한다. 끝까지 저장하면 총 Data를 반환하고, Manager에서 Data를 다시 출력 포맷에 맞게 변환하여 출력하면 성공적으로 나올 수 있게 된다.

4) Delete

삭제할 회원의 특정 정보를 인자로 함수에 전달하고, 함수 내에서 해당 트리의 노드로 pCurr과 pP를 생성하여 각각 root, nullptr값으로 초기화 시킨다. 그 다음 while문으로 pCurr이 nullptr이 아니고 삭제하고자 하는 회원 정보가 일치하지 않을 동안 반복하여 삭제 노드의 위치를 찾아낸다. 이는 앞서 한 Insert 알고리즘과 유사하게 진행된다. pP에 pCurr을 저장하고, if else문으로 순서를 따져서 왼쪽 자식으로 내려가거나 오른쪽 자식으로 내려간다. while문이 종료되고 나서 만약 pCurr이 nullptr이라면 탐색에 실패했다는 뜻이므로 return으로 종료시킨다.

pCurr이 nullptr이 아닌 경우, 4가지 경우로 나누어서 삭제를 진행한다. 먼저 첫번째로 pCurr이 자식이 없는 leaf 노드인 경우, pCurr이 root라면 root를 nullptr로 만들고 pCurr을 삭제시킨다. root가 아니라면 부모 노드인 pP의 왼쪽 자식이 pCurr인 경우 pP의 왼쪽 자식을 nullptr로 set시키며, 반대인 경우는 오른쪽 자식을 nullptr로 set시키며 pCurr을 삭제시킨다.

두번째로 pCurr이 오른쪽 자식만 가지고 있는 경우, 마찬가지로 root인지부터 따져서 맞다면 오른쪽 자식을 root로 바꾸고 pCurr을 삭제한다. 아니라면 pP의 왼쪽 자식이 pCurr인 경우 pP의 왼쪽 자식을 pCurr의 오른쪽 자식으로 대체하고 pCurr을 삭제한다. 반대인 경우엔 pP의 오른쪽 자식을 pCurr의 오른쪽 자식으로 대체하고 pCurr을 삭제한다.

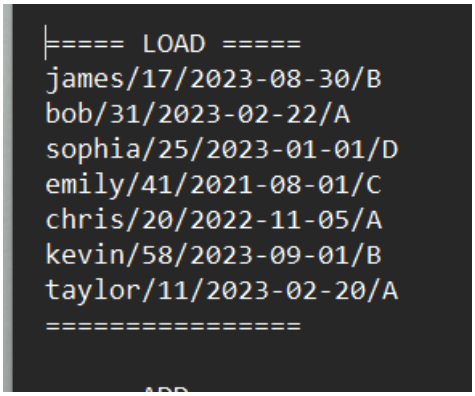
세번째로 pCurr이 왼쪽 자식만 가지고 있는 경우, 위와 마찬가지로 진행하는데 pP에서 대체할 때 pCurr의 왼쪽 자식으로 대체한다는 점만 달라진다.

마지막으로 pCurr이 왼쪽 오른쪽 자식 둘 다 가지고 있는 경우, 가리킬 노드가 더 필요하다. pPrevPrev로 PCurr을, pPrev로 pCurr의 오른쪽 자식을, 새로운 curr로 pCurr의 오른쪽 자식의 왼쪽 자식을 가리키도록 한다. 그 다음 while문으로 curr이 존재할 동안 한 단계씩 포인터들이 내려가는 식으로 저장하여 트리의 모형을 파악한다. while문이 종료되면 pCurr의 정보를 pPrev의 정보로 대체한다. pCurr이 지울 노드이기 때문에 복제시키는 것이다. 그리고나서 if else문으로 만약 pPrevprev가 pCurr과 같다면 curr이 존재하지 않아 while문을 돌지 않은 경우를 뜻한다. 이때 pPrevprev의 오른쪽 자식을 pPrev의 오른쪽 자식으로 대체시키면서 pPrev를 삭제한다. 같지 않은 경우엔 curr이 존재하여 while문을 돌면서 달라졌다는 의미이므로, 왼쪽 자식을 대체시키는 것으로 진행한다. 이렇게 하면 모든 경우에 대해 성공적으로 delete를 진행할 수 있게 된다.

4. Result Screen

1) LOAD

data.txt에 저장된 회원 정보들을 출력 형식에 맞게 log.txt에 순서대로 출력한 모습이다. 이때 회원 정보들은 MemberQueue에 차례대로 추가된다.



2) ADD

command.txt에 입력된 회원 정보들을 읽어와 같은 MemberQueue에 저장한다. 이후 정상적으로 완료되었음을 log.txt에 이어서 출력한다.

```
===== ADD =====
tom/50/2020-07-21/D
===============

===== ADD =====
bella/94/2023-08-31/B
===============

===== ADD =====
harry/77/2024-02-03/B
===============

      QPOP
```

3) QPOP

MemberQueue에 저장된 회원 정보들을 차례대로 pop한 걸 바탕으로 TermsList, TermsBST, NameBST를 구축한다. 성공적으로 완료되었을 경우 마찬가지로 이어서 안내문을 출력한다.

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

      SEARCH
```

4) SEARCH

command.txt에 입력된 회원 이름 정보를 NameBST에서 탐색하여 노드가 존재할 경우, 해당 노드의 회원 정보들을 이어서 출력한 모습이다. 이때 가입 약관 종류에 따라 개인 정보 만료일자도 계산된 것을 확인할 수 있다.

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

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

      PRINT
```

5) PRINT

Command.txt에 입력된 추가 인자에 따라, NAME인 경우 NameBST에 저장된 내용들을 중위 순회 방식으로 탐색하여 순서대로 출력한다. 가입 약관 종류가 추가 인자로 전달되면 해당 종류 별로 각각 저장된 TermsBST의 내용들을 같은 중위 순회 방식으로 이어서 출력한다. 이때 중위 순회 방식으로 출력해서 트리에 사전 순, 날짜 순으로 저장된 걸 그대로 확인할 수 있다.

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

6) DELETE

command.txt 에서 NAME으로 'emily'를 삭제했을 때, 다시 PRINT 해 보면 NameBST에서 해당 회원 정보가 삭제된 것을 볼 수 있다. 또한 가입약관종류로 'C'를 PRINT 했을 때, 해당 TermsBST에는 emily밖에 없었으므로, 삭제된 이후에는 해당 트리는 노드가 비게 되어 프린트 오류 메시지가 뜨게 된다.

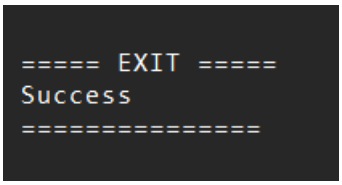
```
=================

===== 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/41/2021-08-01/2023-08-01
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
=================

===== ERROR =====
500
=================


      PRINT
```

그 다음으로 특정 날짜가 입력되었을 때 이전 날짜들까지 모두 삭제하는 것은 구현하지 못하였다. 따라서 다시 종류별로 PRINT를 했을 때 같은 결과들이 나온다.

```
===== 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/41/2021-08-01/2023-08-01
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 =====
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 D
tom/50/2020-07-21/2023-07-21
sophia/25/2023-01-01/2026-01-01
=================


      EXIT
```

7) EXIT

마지막으로 EXIT가 입력되면 프로그램 상의 메모리들을 해제하고 프로그램을 종료시키게 된다. 이때 정상적으로 종료된다는 알림 메시지를 출력한다.



5. Consideration

본 프로젝트를 진행할 때, 우선 클래스 객체를 이용함에 있어서 시행착오를 겪었었다. 처음에 queue 객체를 이용하기 위해 Manager 클래스의 멤버 변수로 선언해봤지만, this가 nullptr이었다는 오류가 떴었다. 그래서 멤버 변수로 하지 않고 전역 변수로 선언하니 정상적으로 이용할 수 있었다. 그런데 BST에 관해서도 마찬가지로 전역 변수로 선언하니, 실행했을 때 컴파일이 끝나지 않는 오류가 발생했었다. 이에 단계 별로 주석처리를 해가며 원인을 찾아본 결과, 이쪽이 문제가 아니라 BST 함수 내부에서 while문이 제대로 탈출되지 않아 무한 로딩이 걸렸었던 것이었다. 탈출 조건을 더 추가하여 수정하여 돌리니 무사히 돌아갈 수 있었다.

개인정보 만료일자를 계산하는 방법도 여러가지 고민했었는데, 처음에 떠오른 건 개인정보 수집일자를 문자열 배열로 저장했으니 해당 인덱스 칸 부분만 아스키코드를 업데이트하여 수정하는 방식이었다. 하지만 이는 제대로 계산하기도 어려울뿐더러 비효율적인 것 같아서 다른 방안을 생각해보다가, 맨처음에 command.txt를 읽으면서 문자열을 strtok으로 자른 후 종류 별로 저장했던 방식이 떠올라 다시 코드를 수정하였다. 그래서 각각 잘라 년, 월, 일로 atoi()함수를 이용하여 int형으로 바꾼 다음 가입약관종류에 따라 수를 계산하고, 다시 문자열로 재변환하여 하나로 합치는 걸로 완성하였다.

TermsBST를 진행할 때 처음에는 무작정 하나의 트리로 만들었다가, 가입약관종류별로 TermsList에서 각각의 BST 포인터를 가져야 하니 개별적으로 트리가 생성되어야 한다는 걸 깨달아서 다시 수정하였다. 그래서 Manager에서 가입약관종류에 따라 if문으로 나누어서 트리 객체를 종류 별로 생성하였다.

PRINT 명령어에 대한 함수를 설계할 때도 여러 시행착오를 겪었었는데, 먼저 문제가 발생했던 건 NameBST나 TermsBST나 해당 .cpp 파일에서는 Manager의 flog객체를 사용할 수 없다는 점이었다. 그래서 트리 파일의 print 함수에서 바로 출력을 못하니, 중위 순회로써 재귀적으로 반복해야 하는데 다른 함수에 전달했다 하는 것도 오류가 나서 진행에

어려움을 겪었었다. Print 함수에 인자로 ofstream& flog식으로 객체를 전달하는 방법도 떠올랐긴 했으나, 아직 그 부분은 제대로 공부가 안된 상태여서 실행해봤을 때 계속 오류가 났었다. 그래서 계속 여러 방법을 시도해보다가, 아예 방문한 노드의 정보들을 하나의 string문자열로 계속 이어서 합치는 방식을 떠올렸다. 이 점에 대해서도 처음에는 char형 2차원 배열 형식으로 동적 할당하여 저장하는 방법을 시도했다가 메모리 위반 오류로 string형으로 수정하였다. String도 배열로 하면 선언에 있어 문제가 있었기 때문에, +=연산자로 하나로 합치기로 하였다. 이때도 여러 오류들이 발생하였는데, 주로 예기치 않은 토큰 문제나 구문 오류 같은 것들이 떴었다. 문법 상 ';'을 안 붙인 부분은 없었지만 그런 오류들이 뜨길래 이러한 오류들의 원인에 대해서도 찾아서 공부하였다. 그 후 이러한 오류들은 보통 진짜 문법 문제이기 보다 헤더 파일의 중복이나 몇 가지 선언을 하지 않아서 발생하는 문제라는 점을 깨달았다. 여기서는 해당 .cpp파일에서 처음에 using namespace std; 선언을 하지 않아서 발생했던 문제여서 바로 수정하니 정상적으로 돌아갈 수 있었다.

Delete 명령어는 아직 미완성인 부분인데, NAME 인자가 전달된 경우 해당 회원만 삭제하는 데엔 성공하였으나 특정 일자가 전달된 경우 이전 날짜까지 모두 삭제하는 데에 어려움을 겪었다. 가입약관종류 별로 각각 객체가 있으니, 전부 탐색을 하여 삭제를 해야될 것 같다는 생각은 들었다. 하지만 직접 코드를 짜면서 해보니 적절한 방법이 떠오르지 않았다. 이에 TermsBST에서도 Search함수를 만들어서 이용하려 했으나, BST마다 저장하고 있는 회원 정보의 종류도 다 달라서 잘 사용할 수가 없었다. 이러한 부분에 대해서는 더 고민이 필요할 것 같다.

마지막으로, 이 코드들을 처음엔 전부 윈도우에서 Visual studio로 작성하였는데 이때까진 컴파일 할 때마다 무사히 성공적으로 결과가 출력되었으나, 리눅스(우분투 18.04)에서 VScode로 옮겨서 실행하니 여러 에러가 뜨면서 실패하였다. 이에 전체적으로 주석처리를 하면서 오류를 찾아보기도 하고, 리눅스의 컴파일 방법에 대해서 여러가지 공부도 하였다. 결국 VScode 자체에서 컴파일 하지 않고 터미널에서 명령어로 컴파일 하니 결과가 나온다는 것을 깨달았다. 또한 에러들도 자세한 내용들이 떴었는데, Visual studio에서는 따로 헤더파일을 안 불러와도 사용할 수 있었던 함수들이 여기서는 꼭 불러와야 쓸 수 있다는 점을 알 수 있었다. 또한 원래는 strcpy_s() 함수를 이용하여 코드를 설계하였었는데, 이쪽에서는 strcpy_s나 strncpy같은 함수들이 사용이 불가능했었다. 이 부분에 대해서는 더 알아보고 싶지만, 일단 전부 strcpy로 고치는 방식으로 진행하였더니 오류들이 사라졌었다. 하지만 제일 여러 시행착오를 겪게 한 오류는 터미널에서 실행하였을 때 'Segmentation fault(core dumped)'라고 뜨면서 실행이 되지 않던 문제였다. 해당 오류에 대해 찾아보니 주로 null에 접근하는 등 잘못된 메모리에 접근했을 때 발생하는 문제였다. 마찬가지로 주석처리하며 하나씩 원인을 따져본 결과, 몇 가지 클래스 멤버 변수를 생성자에서 초기화하지 않아서 발생했던 것이라는 걸 깨달았다. 바로 수정을 하니 무사

히 모든 명령어에 대해 전부 log.txt 파일에 출력할 수 있었다.

이런 식의 오류들을 확인하니 확실히 VScode쪽이 더 확실하게 코딩을 해야 정상적으로 돌아간다는 점을 깨달을 수 있었다. 또한 이번 프로젝트를 진행하면서 기본적으로 연결 리스트, 큐, 트리에 대한 자료구조 알고리즘을 공부하는 것 뿐만 아니라 여러 오류들에 대한 해결방안, 리눅스 사용법 등을 공부해볼 수 있었다.