

Data Structure Lab Project 1

학 과: 컴퓨터정보공학부

담당교수: 최상호 교수님

강의시간: 화2, 목1

학 번: 2018202046

성 명: 이준휘

1. Introduction

해당 프로젝트는 Queue를 통해 가공된 1차적인 데이터들을 List와 BST로 저장하고 이를 Heap을 통해 분류할 수 있는 프로그램을 제작하는 것이다. 이 프로그램은 "command.txt"에 명령어가 저장되어 있고, "log.txt"를 통해 결과를 확인한다.

QLOAD를 통해 "data.txt"에서 이름 나이 아이디를 불러와 주어진 조건에 맞을 경우 이를 Queue형태로 저장한다.

ADD는 Queue에 데이터 하나를 직접 입력하는 명령어다.

QPOP은 Queue의 데이터를 지정된 개수만큼 List와 BST로 옮긴다.

SEARCH는 user나 id를 입력받아 이를 구분하여 각각 List, BST에서 탐색하여 해당 결과를 출력한다.

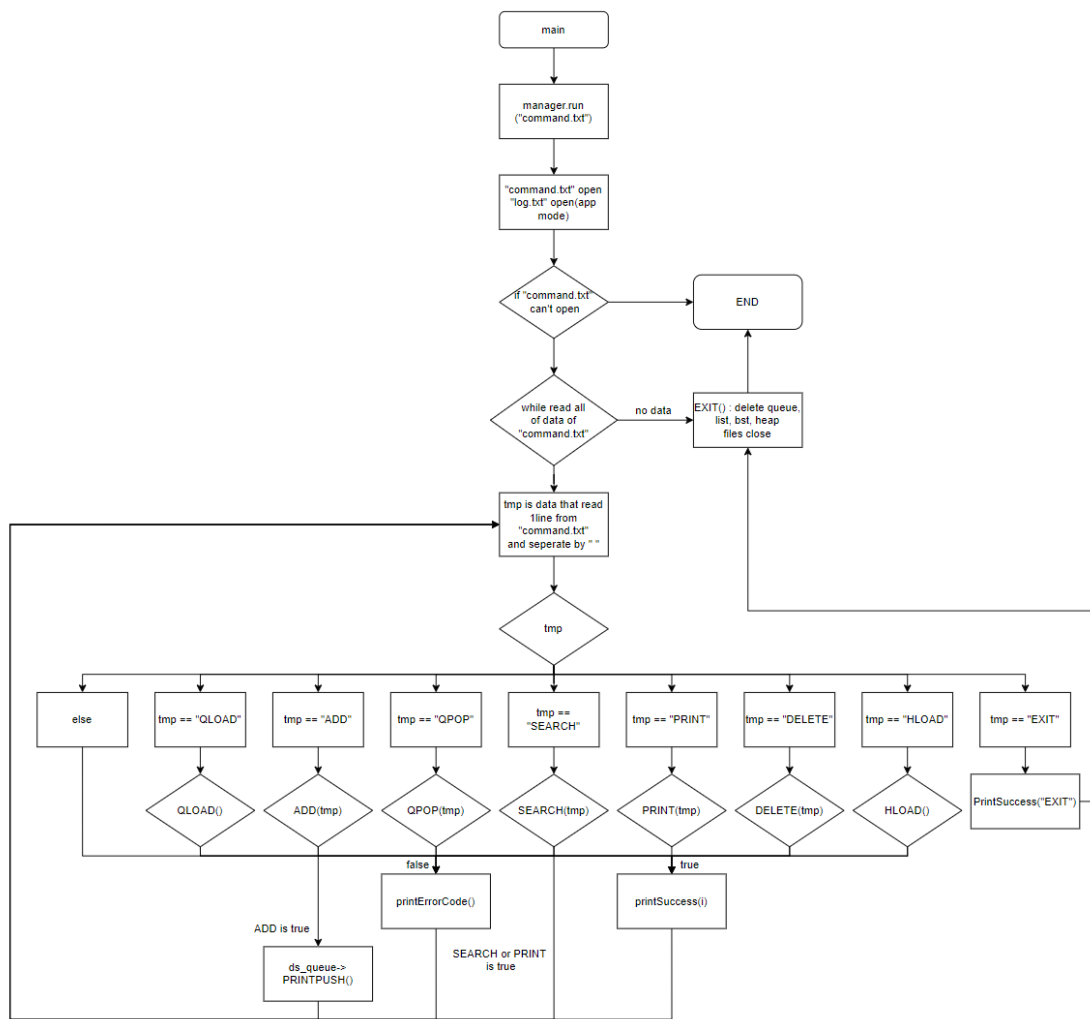
PRINT는 후속 인자로 L을 입력받은 경우 List의 데이터들을 출력한다. 후속 인자로 B를 입력 받은 경우에는 다음 인자로 PRE, IN, POST, LEVEL을 입력받아 각각 Pre-order, In-order, Post-order, Level-order의 순서로 값을 출력한다. 후속 인자가 H일 경우 Heap에 저장된 데이터를 Level-order로 출력한다.

DELETE의 경우 아이디를 입력받아 BST와 제거하고, 만약 해당 List의 보유 계정 수가 0일 시 해당 list 데이터도 제거한다.

HLOAD는 기존 Heap에 데이터가 존재할 경우 이를 제거하고 새롭게 데이터를 구성한다. 이 데이터는 User_List의 정보를 이용하여 연령대와 수를 저장한다.

2. Flowchart

flowchart of program



3. Algorithm

A. Manager class

- i. AccountQueue* ds_queue
Queue 방식의 데이터가 저장된 공간이다.
- ii. UserList* ds_list
Linked List 방식의 데이터가 저장된 공간이다.
- iii. AccountBST* ds_bst
이진 탐색 트리 방식은 데이터가 저장된 공간이다.

- iv. UserHeap* ds_heap
Heap 구조 방식의 데이터가 저장된 공간이다.
- v. Constructor
ds_queue, ds_list, ds_bst, ds_heap을 new를 통해 메모리 공간을 할당시켜준다.
- vi. run(const char* command)
해당 함수는 프로그램이 구동되는 실질적인 부분이다.

파일명을 입력으로 받으며, 해당 파일을 읽기로, log.txt는 이어 쓰기로 가져온다.
그리고 읽는 파일의 종료지점까지 아래 내용을 반복한다.

- 파일로부터 한 줄의 데이터를 읽어온다. 만약 해당 줄이 공백일 경우 다음 줄을 읽는다.
- 해당 줄을 공백을 기준으로 자른다.
- 만약 읽은 데이터가 QLOAD인 경우 QLOAD()함수를 수행한다. 해당 QLOAD()함수의 반환값이 true일 시 ds_queue->PRINT()함수를 통해 queue를 출력하고, 거짓일 경우 PrintErrorCode(100)를 통해 에러코드 100을 log.txt에 출력한다.
- 만약 읽은 데이터가 ADD인 경우 tmp에 저장된 다음 데이터를 ADD()함수의 인자로 넣어 수행한다. 해당 ADD(tmp) 함수의 반환값이 true일 시 ds_queue->PRINTPUSH()함수를 통해 마지막으로 입력된 데이터를 출력하고, 거짓일 경우 PrintErrorCode(200)를 통해 에러코드 200을 log.txt에 출력한다.
- QPOP을 만날 경우 tmp의 저장된 다음 데이터를 QPOP() 함수의 인자로 넣어 수행한다. QPOP(tmp)가 성공할 경우 PrintSuccess()를 통해 QPOP이 성공함을 알리고, 실패할 경우 PrintErrorCode(300)을 통해 에러코드를 출력한다.
- 읽은 데이터가 SEARCH인 경우, tmp의 다음 데이터와 함께 SEARCH(tmp)를 수행하고 이에 실패할 경우 PrintErrorCode(400)를 통해 에러코드를 출력한다.
- 읽은 데이터가 PRINT인 경우, tmp의 다음 데이터와 함께 PRINT(tmp)를 수행하고, 이에 실패할 경우 PrintErrorCode(500)을 통해 에러를 출력한다.
- 읽은 데이터가 DELETE인 경우, tmp의 다음 데이터와 함께 DELETE(tmp)를 수행한다. 삭제에 성공할 경우 PrintSuccess()를 통해 DELETE에 성공함을 알리고, 실패할 경우 PrintErrorCode(600)을 통해 에러코드를 출력한다.

- 읽은 데이터가 HLOAD인 경우 HLOAD()함수를 수행하고 만약 반환값이 참일 경우 PrintSuccess()를 통해 성공함을 출력한다. 만약 반환값이 false일 경우 PrintErrorCode(700)을 통해 에러코드를 출력한다.
 - 읽은 데이터가 EXIT인 경우 PrintSuccess()를 통해 성공함을 출력하고 루프를 탈출한다.
 - 이 외의 입력을 받을 경우 PrintErrorCode(800)을 출력하여 커멘드가 잘못됨을 알린다.
- 위의 루프에서 탈출할 경우 log.txt와 command.txt 파일을 닫고 EXIT()을 통해 메모리 할당을 해제한 뒤 함수를 종료한다.

vii. bool QLOAD()

해당 함수는 QLOAD를 통해 파일로부터 ds_queue에 데이터를 저장하는 함수다. data.txt로부터 fdata를 통해 데이터를 읽어온다. 만약 파일이 없을 경우 false를 반환한다.

fdata에서 읽을 내용이 없을 때까지 아래의 내용을 반복한다.

- fdata에서 한 줄을 읽어와 이를 공백을 기준으로 구별하여, 각각 name, charAge, id로 저장한다.
 - 만약 이 중 하나라도 NULL일 경우 파일을 닫고 false를 반환한다.
 - isalpha()를 통해 이름에 알파벳만이 들어갔는지를 확인하고 알파벳 외의 문자가 있을 경우 파일을 닫고 false를 반환한다.
 - isdigit()을 통해 나이에 숫자만 있는지 확인하고 숫자 외의 문자가 있을 경우 파일을 닫고 false를 반환한다.
 - 정상적인 숫자일 경우 이를 int형으로 변환한다. 만약 이 숫자가 10~69범위 외의 숫자일 경우 파일을 닫고 false를 반환한다.
 - isalnum()을 통해 아이디에 알파벳이나 숫자만 있는지를 확인하고, 이외의 문자가 포함된 경우 파일을 닫고 false를 반환한다.
 - AccountQueueNode* data를 새로 할당하고 여기에 이름, 나이, 계정 정보를 넣는다.
 - ds_queue->PUSH(data)를 통해 데이터를 queue에 넣는다. 만약 이에 실패할 경우 파일을 닫고 false를 반환한다.
- 위의 반복문을 정상적으로 끝낸 경우 파일을 닫고 true를 반환한다.

viii. bool ADD(char* input)

input에 있는 데이터를 가지고 노드 한 개를 넣는다.

fdata에서 한 줄을 읽어와 이를 공백을 기준으로 구별하여, 각각 name, charAge, id로 저장한다.

만약 이 중 하나라도 NULL일 경우 파일을 닫고 false를 반환한다.

isalpha()를 통해 이름에 알파벳만이 들어갔는지를 확인하고 알파벳 외의 문자가 있을 경우 파일을 닫고 false를 반환한다.

isdigit()을 통해 나이에 숫자만 있는지 확인하고 숫자 외의 문자가 있을 경우 파일을 닫고 false를 반환한다.

정상적인 숫자일 경우 이를 int형으로 변환한다. 만약 이 숫자가 10~69범위 외의 숫자일 경우 파일을 닫고 false를 반환한다.

isalnum()을 통해 아이디에 알파벳이나 숫자만 있는지를 확인하고, 이외의 문자가 포함된 경우 파일을 닫고 false를 반환한다.

AccountQueueNode* data를 새로 할당하고 여기에 이름, 나이, 계정 정보를 넣는다.

ds_queue->PUSH(data)를 통해 데이터를 입력하고 해당 결과를 반환한다.

ix. bool QPOP(char* input)

input으로 입력한 데이터만큼의 개수를 Queue에서 POP하고, 해당 데이터를 List와 BST에 저장한다.

isdigit()을 통해 입력받은 데이터가 숫자로만 이루어졌는지 확인하고, 아닐 경우 false를 반환한다.

만약 숫자만을 입력받은 경우 이를 정수로 변환한다.

해당 숫자가 ds_queue의 size보다 클 경우 false를 반환한다.

해당 숫자만큼 queue를 POP()을 수행하고 나온 데이터를 ds_bst->Insert(ds_list->Insert(temp))를 통해 데이터를 넣고 이에 실패할 경우 false를 반환한다. 만약 성공한 경우에는 temp를 삭제하고 반복을 다시 수행한다.

위의 과정을 모두 수행한 경우 true를 반환한다.

x. bool SEARCH(char* input)

input에 따라 이름이나 계정을 검색하는 함수다.

input을 통해 해당 mode와 find를 입력받는다.

만약 mode나 find 중 하나라도 NULL일 경우 false를 반환한다.

만약 mode로 user를 입력받을 경우 ds_list->Search(find)를 통해 리스트에서 이

름을 탐색한다.

만약 mode로 id를 입력받은 경우 ds_bst->Search(find)를 통해 bst에서 아이디를 찾아서 출력한다.

만약 위의 두 탐색에서 결과가 없을 경우 false를 반환한다.

xi. bool PRINT(char* input)

input에 따라 데이터를 출력하는 함수다.

mode를 입력받아 해당 모드가 L이고 List에 데이터가 존재할 경우 ds_list->Print_L을 통해 list를 출력한다. 이외의 경우에는 false를 반환한다.

mode가 B이고 BST에 데이터가 존재할 경우 mode2를 추가로 입력받아 mode2에 따라 PRINT_PRE(), PRINT_IN(), PRINT_POST, PRINT_LEVEL을 수행한다. 이외의 경우에는 false를 반환한다.

mode가 H일 경우 ds_heap에 데이터가 존재하면, ds_heap을 Level order로 출력한다.

xii. bool DELETE(char* input)

해당 함수는 BST에서 계정을 삭제하는 함수다.

input을 통해 삭제할 아이디를 알 수 있다.

삭제하고자 하는 아이디를 사용하는 사람을 FindNameFromId(input)을 통해 알아낸다.

그리고 만약 찾을 사람이 없을 경우 false를 반환한다.

ds_list->Delete_Account(findname, input)을 통해 리스트에서 데이터를 삭제하고, 삭제가 되었을 경우 ds_bst->Delete(input)을 통해 리스트에서도 삭제한다.

만약 둘 중 하나라도 삭제되지 않을 경우 false를 반환한다.

xiii. bool HLOAD()

HLOAD는 Heap을 만들어 힙에 List의 정보를 바탕으로 데이터를 입력하는 함수다.

만약 Heap의 사이즈가 1보다 클 경우 힙을 삭제하고 다시 생성한다.

temp는 List의 첫 주소를 가져온다.

temp가 마지막 주소를 가리킬 때까지 temp의 값에서 나이를 가져와 agegroup을 일의 자리 버림을 통해 만들고 ds_heap->Insert(agegroup) 함수를 통해 입력한다. 만약 false를 출력한 경우 false를 반환한다. 그 후 temp의 주소를 다음으로 가리키고 위의 작업을 반복한다.

- xiv. `bool EXIT()`
해당 함수는 프로그램 종료 직전 모든 메모리 공간을 할당 해제하는 함수다.
각각의 class 객체가 존재하는지 if문으로 확인한 다음 있을 경우 delete를 해준다.
그 후 프로그램을 종료한다.
- xv. `void PrintErrorCode(int num)`
해당 함수는 에러코드 num을 log.txt.에 출력하는 함수다.
- xvi. `void PrintSuccess(char* act)`
해당 함수는 어떠한 명령어가 성공했는지를 log.txt.에 출력하는 함수다.

B. AccountQueue Class

- i. `int queue_size`
해당 인자는 Queue의 크기를 저장하는 변수다.
- ii. `AccountQueueNode* Front`
해당 인자는 Queue의 Front 위치를 가리키는 포인터다.
- iii. Constructor
queue_size를 0, Front를 NULL로 초기화한다.
- iv. Destructor
Front가 NULL이 될 때까지 Front를 지우고 Front가 가리키던 다음 주소에서 동일한 동작을 실행한다.
- v. `AccountQueueNode* POP()`
해당 함수는 Front의 데이터를 내보내는 함수다.
만약 front가 비어있을 경우 NULL을 반환한다.
이외의 경우에는 Front의 주소를 다음 주소로 바꾸고, queue_size를 1 줄인 후, 기존 주소를 반환한다.
- vi. `bool PUSH(AccountQueueNode* node)`

해당 함수는 Queue의 end 위치에 노드를 저장하는 메서드다.

만약 Front가 비어있을 경우 Front가 노드를 가리키도록 하고 true를 반환한다.

이외의 경우에는 temp를 통해 Front의 주소부터 끝까지 이동하면서 같은 데이터가 있는지 확인한다. 만약 같은 데이터가 있을 경우 node를 삭제한다.

가장 마지막까지 왔을 경우 해당 위치의 다음을 node로 설정하고 queue_size를 1 늘린 후 true를 반환한다.

vii. `bool EMPTY()`

해당 queue의 사이즈를 출력하는 함수다.

queue_size가 0일 경우 true을, 아닐 경우 false 반환한다.

viii. `int SIZE()`

queue의 사이즈를 반환하는 함수다.

ix. `void PRINT()`

queue를 출력하는 함수다.

flog를 통해 log.txt를 이어쓰기 모드로 연다.

그리고 Front부터 끝까지의 데이터를 출력 형식에 맞게 출력하고 flog를 닫고 함수를 종료한다.

x. `void PRINTPUSH()`

가장 마지막에 푸시한 데이터를 출력하는 함수다.

flog를 통해 log.txt를 이어쓰기 모드로 연다.

그리고 temp를 Front의 pNext들이 가리키는 가장 끝 주소로 가리키게 만든다.

그 후 해당 주소의 내용을 출력하고, flog를 닫고 함수를 종료한다.

C. UserList Class

i. `UserListNode* Root`

Linked List의 가장 첫 주소를 가리키는 노드다.

ii. `Constructor`

Root를 NULL로 초기화한다.

iii. Destructor

Root가 NULL을 가리킬 때까지 Root를 지우고, 기존의 Root가 가리키던 주소로 Root를 바꾼다.

iv. UserListNode* GetRoot()

Root를 반환한다.

v. AccountBSTNode* Insert(AccountQueueNode* node)

해당 함수는 node의 정보를 토대로 List에 데이터를 추가하는 함수다.

AccountBSTNode* insertNode에 node의 데이터를 옮겨온다.

만약 Root가 NULL일 경우 Root에 새로 만드는 insertList의 주소를 넣어두고 insertList의 이름과 나이를 설정한 후 insertList->InsertAccount(insertNode)를 통해 BST 노드를 pNext에 연결한다. 그리고 insertNode를 반환한다.

만약 Root가 NULL이 아닐 경우, temp에 Root의 주소를 부여하고 temp의 정보가 node의 정보와 같지 않을 동안 아래 내용을 반복한다.

- 만약 temp의 다음 위치가 NULL일 경우 insertList를 만들고 데이터를 저장한다. 그 후 이를 temp 다음 위치에 연결한다. 그 후 insertNode를 반환한다.
- 이외의 경우에는 temp를 다음 주소로 변경한다.

만약 같은 이름의 데이터를 찾을 경우 해당 데이터의 AccNum이 3을 넘지 않는지 확인한다. 넘지 않을 경우 temp->InsertAccount(insertNode)를 통해 아이디 노드를 연결하고 insertNode를 반환한다.

이외의 경우 insertNode를 삭제하고 NULL을 반환한다.

vi. bool Search(char* name)

해당 메서드는 name을 찾아 출력하는 메서드다.

만약 Root가 NULL일 경우 false를 반환한다.

이외의 경우 temp를 Root값으로 설정 후 temp의 주소가 NULL을 가리킬 때까지 다음 주소로 이동해가며 해당 주소의 이름과 입력받은 name이 같은지 확인한다.

만약 같을 경우 flog를 통해 log.txt를 이어쓰기 모드로 열어 해당 listnode의 데이터를 출력한 후 true를 반환한다.

찾지 못한 경우 false를 반환한다.

vii. bool Delete_Account(char* name, char* id)

해당 메서드는 이름과 아이디를 입력받아 이를 토대로 리스트의 데이터를 지우는 동작을 수행한다.

만약 Root가 NULL일 경우 false를 반환한다.

cur은 Root의 주소를, Prev는 NULL의 주소를 가리키게 한 후 cur이 NULL을 가리키지 않고 cur의 이름이 입력받은 이름과 같지 않을 동안 prev의 주소는 cur로, cur의 주소는 다음 주소로 이동한다.

만약 cur이 NULL일 경우 같은 데이터를 찾지 못한 것으로 false를 출력한다.

tempBST는 cur의 pHead를 가리키게 한 후 tempBST를 이용하여 해당 아이디가 존재하는지 탐색한다.

만약 같은 아이디를 찾지 못할 경우 false를 반환한다.

같은 아이디를 찾을 경우 cur->Delete_Accout(id)를 통해 해당 아이디를 지운다.

그리고 만약 해당 리스트에서 AccNum이 0일 경우 해당 리스트 또한 지우고 true를 반환한다.

viii. void Print_L(UserListNode* node)

해당 메서드는 list를 출력하는 메서드다.

만약 출력하는 node가 없을 경우 메서드를 종료한다.

flog를 통해 log.txt를 이어쓰기 모드로 연 후 현재 노드의 데이터를 출력한다.

그 후 다음 노드의 데이터를 인자로 하는 Print_L(node->GetNext())을 재귀적 방법으로 호출한다.

D. AccountBST Class

i. AccountBSTNode* Root

해당 인자는 BST의 첫 번째 주소를 가리킨다.

ii. Constructor

Root의 값을 NULL로 초기화한다.

iii. Destructor

queue <AccountBSTNode*> deleteQueue를 사용하여 level-order의 순으로 데이터를 삭제한다.

deleteQueue에 Root 값을 push하고, deleteQueue가 빌 때까지 front를 읽어와

left와 right가 존재할 경우 push하고, pop을 통해 front를 삭제한 후 데이터를 삭제한다.

iv. AccountBSTNode* GetRoot()

Root의 주소를 반환하는 메서드다.

v. bool Insert(AccountBSTNode* node)

node를 BST에 넣는 메서드다.

만약 넣는 node가 NULL일 경우 false를 반환한다.

만약 Root가 비어있을 경우 Root를 node로 설정한 후 true를 반환한다.

이외의 경우에는 temp에 Root의 값을 할당하고 아래 내용을 반복한다.

- CompareName(char*, char*)를 통해 두 문자열의 우선순위를 비교하여 같을 경우, false를 반환한다.
- node가 가리키는 id가 temp의 아이디보다 먼저 와야 할 경우 left가 없으면 왼쪽에 저장하고, left가 있으면 pLeft로 temp를 바꾼다. 그리고 true를 반환한다.
- 만약 나중에 와야 할 경우 temp의 right가 없으면 오른쪽에 저장하고, right가 있으면 pRight로 temp를 바꾼다. 그 후 true를 반환한다.

vi. bool Search_Id(char* id)

해당 메서드는 입력한 id와 같은 id를 가진 노드를 찾는 메서드다.

만약 Root가 NULL일 경우 false를 반환한다.

이외의 경우 temp에 Root의 주소를 갖도록 하고, 아래 내용을 반복한다.

- 만약 입력한 id와 temp의 id가 같을 경우 해당 노드를 flog를 통해 log.txt에 출력하고 true를 반환한다.
- 입력한 id가 먼저 오는 경우 좌측 노드가 없으면 false를 반환하고, 있을 경우 temp의 주소를 pLeft로 옮긴다.
- 입력한 id가 나중에 오는 경우 우측 노드가 없으면 false를 반환하고, 있을 경우 temp의 주소를 pRight로 옮긴다.

vii. bool Delete(char* id)

해당 메서드는 입력받은 id를 BST에서 제거하는 동작을 한다.

만약 기존의 데이터가 없을 경우 false를 반환한다.

temp에 Root의 주소를 할당하고, parent에 NULL을 할당한다.

그 후 temp가 NULL을 가리키지 않고 입력받은 아이디가 temp의 아이디와 같지 않을 동안 parent의 값을 temp를, temp의 값은 CompareName()에 따라 먼저 올 경우 좌측을, 나중에 올 경우 우측을 가리키도록 한다. 만약 temp가 NULL을 가리킬 경우 삭제할 노드가 없다는 것으로 false를 반환한다.

만약 temp의 좌우가 다 비어있을 경우, temp가 Root이면 Root를 지우고, 이외의 경우 temp를 parent의 자식 노드에서 지운다.

만약 temp가 왼쪽 또는 오른쪽 노드 하나만 가질 경우, temp가 Root일 경우 Root를 temp의 자식으로 설정하고, 이외의 경우 parent의 자식 노드 중 temp를 temp의 자식으로 변경한다. 그 후 temp를 삭제한다.

만약 temp가 두 자식을 모두 갖는 경우 아래의 작업을 실행한다.

- temp를 가리키는 prevprev, temp의 우측 자식을 가리키는 prev, prev의 좌측 자식을 가리키는 cur를 선언하고, temp의 자식들을 Ltemp와 Rtemp에 저장한다.
- cur이 NULL이 아닐 동안 prevprev는 prev로, prev는 cur로, cur은 cur의 좌측 노드로 이동한다.
- prev의 우측 자식을 Rprev로 저장한다.
- 만약 temp가 Root일 경우 Root를 prev로 설정한다. prev가 temp의 우측 자식이었던 경우 prev의 좌측 자식을 Ltemp로 변경하고, 이외의 경우에는 prev의 좌측 자식은 Ltemp로, 우측 자식은 Rtemp로 변경하고 prevprev의 좌측 자식을 Rprev로 변경한다.
- 만약 temp가 Root가 아닐 경우 prev의 좌우측을 Ltemp와 Rtemp로 설정하고, parent의 자식이었던 temp를 prev로 변경한다.
- temp의 우측 노드가 바꿀 노드일 경우 prev의 우측을 Rprev로 변경하고, 아닐 경우 prevprev의 좌측 노드를 Rprev로 변경한다.

위의 작업을 마치고 temp를 메모리 할당 해제 후 true를 반환한다.

viii. void Print_PRE(AccountBSTNode* node)

해당 함수는 pre-order의 순으로 BST를 출력한다.

node가 NULL이 아닐 경우 flog를 통해 log.txt를 열어 해당 위치의 데이터를 출력하고, Print_PRE(node->GetLeft())와 Print_PRE(node->GetRight())를 통해 재귀 호출하여 작업을 실행한다.

ix. void Print_IN(AccountBSTNode* node)

해당 함수는 in-order의 순으로 BST를 출력한다.

node가 NULL이 아닐 경우 Print_IN(node->GetLeft())를 통해 좌측의 노드를 재귀 호출하고, flog를 통해 log.txt를 열어 해당 위치의 데이터를 출력한다.
그 후 Print_IN(node->GetRight())를 통해 재귀 호출하여 우측 작업을 실행한다.

x. void Print_POST(AccountBSTNode* node)

해당 함수는 post-order의 순으로 BST를 출력한다.

node가 NULL이 아닐 경우 Print_PRE(node->GetLeft())와 Print_PRE(node->GetRight())를 통해 재귀 호출하여 작업을 실행한다.

flog를 통해 log.txt를 열어 해당 위치의 데이터를 출력한다

xi. void Print_LEVEL()

해당 메서드는 queue를 사용하여 Level-order로 BST를 출력한다.

출력에 사용할 queue <AccountBSTNode*> levelQueue를 선언하고 Root를 push한다. 그리고 flog를 통해 log.txt를 이어쓰기 모드로 연다. 그리고 levelQueue가 빌 때까지 front를 통해 데이터를 읽어오고 pop으로 front를 지운 후 읽어온 데이터를 출력 후, 읽어온 노드의 좌우측 노드가 존재할 경우 해당 노드들을 push해준다.

xii. int CompareName(char* cm1, char* cm2)

해당 메서드는 두 문자열을 isupper()과 tolower() 함수를 통해 대소문자 구분 없이 비교하여 cm1이 먼저 올 경우 -1, 같을 경우 0, cm2가 먼저 올 경우 1을 반환하는 함수다.

xiii. char* FindNameFromId(char* id)

해당 함수는 id를 사용하는 사람의 이름을 찾아주는 함수다.

만약 기존에 입력된 데이터가 없을 경우 false를 반환한다.

기존의 데이터가 있을 경우 temp에 Root를 할당하고 아래의 반복을 수행한다.

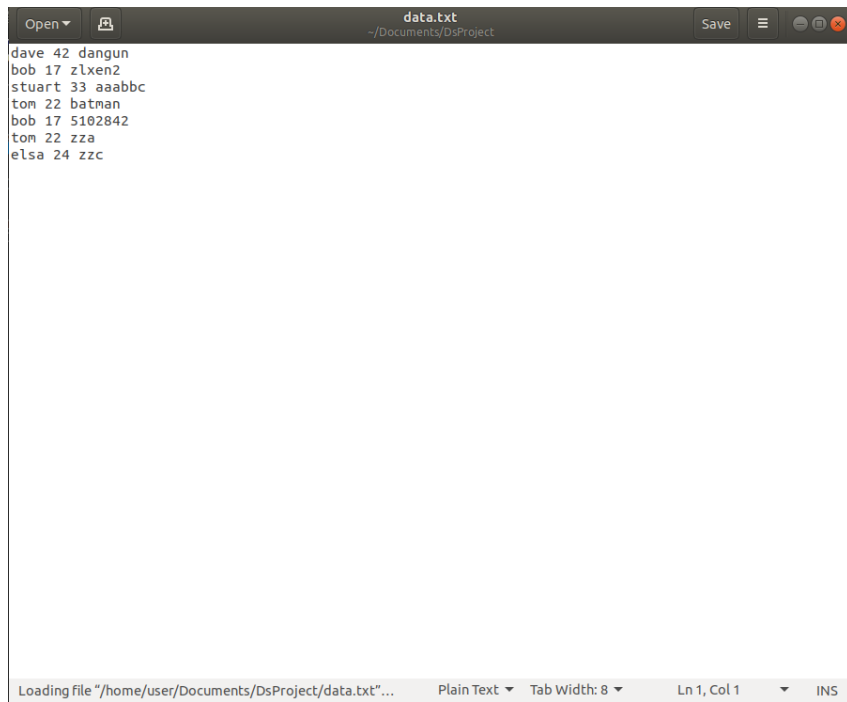
- 만약 id와 temp의 id가 같을 경우 해당 temp의 name를 반환한다.
- 만약 입력한 id가 temp보다 먼저 올 경우 temp의 좌측이 없을 경우 NULL을 반환하고, 있을 경우 temp를 temp의 좌측으로 옮긴다.
- 만약 temp의 id가 입력한 id보다 먼저 올 경우, temp의 우측이 없으면 NULL을 반환하고, 있을 경우 temp를 temp의 우측으로 옮긴다.

E. UserHeap Class

- i. `vector<UserHeapNode*> Heap`
해당 인자는 `UserHeadNode*`를 갖는 벡터다.
- ii. Constructor
벡터에 처음 첫번째 인자를 `NULL`로 넣는다.
- iii. Destructor
- iv. 벡터의 처음 주소부터 크기까지 만약 데이터가 있을 경우 해당 데이터를 삭제한다.
- v. `int GetSize()`
해당 `Heap.size()`를 출력하는 메서드다.
- vi. `bool Insert(int agegroup)`
agegroup 데이터를 heap에 추가하는 메서드다.
우선 Heap의 같은 나이대를 가리키거나 마지막 위치를 가리키지 않을 동안 `i`를 1부터 증가시킨다.
만약 `i`가 마지막 위치까지 왔을 경우 새로운 `UserHeapNode*`인 `temp`를 할당해 주고 해당 데이터에 agegroup과 유저의 수인 1을 입력해준다.
그 후 `Heap.push_back(temp)`를 통해 가장 마지막 위치에 데이터를 추가해준다.
그 후 부모 노드인 `i/2`를 `i`가 1이 아닐 때까지 비교하며 만약 부모가 자식보다 작을 경우 두 데이터를 바꿔준 후 모든 반복을 마치면 `true`를 반환한다.
만약 `i`가 같은 나이대를 가리키는 위치에서 멈췄을 경우 해당 노드의 `NumUser`를 1 늘린 후 `true`를 반환한다.
- vii. `void Print()`
해당 메서드는 heap을 level-order에 따라 출력하는 메서드다.
flog를 통해 log.txt파일을 이어쓰기 모드로 열고, 1번째 벡터값부터 마지막 벡터값까지를 읽어 이를 출력한다.
출력을 마친 후 flog를 닫고 메서드를 마친다.

4. Result Screen

A. data.txt



A screenshot of a text editor window titled "data.txt" with the path "~/Documents/DsProject". The window contains the following text:

```
dave 42 dangun
bob 17 zlxen2
stuart 33 aaabbc
tom 22 batnan
bob 17 5102842
tom 22 zza
elsa 24 zzc
```

The status bar at the bottom indicates "Loading file "/home/user/Documents/DsProject/data.txt"...", "Plain Text", "Tab Width: 8", "Ln 1, Col 1", and "INS".

B. command.txt

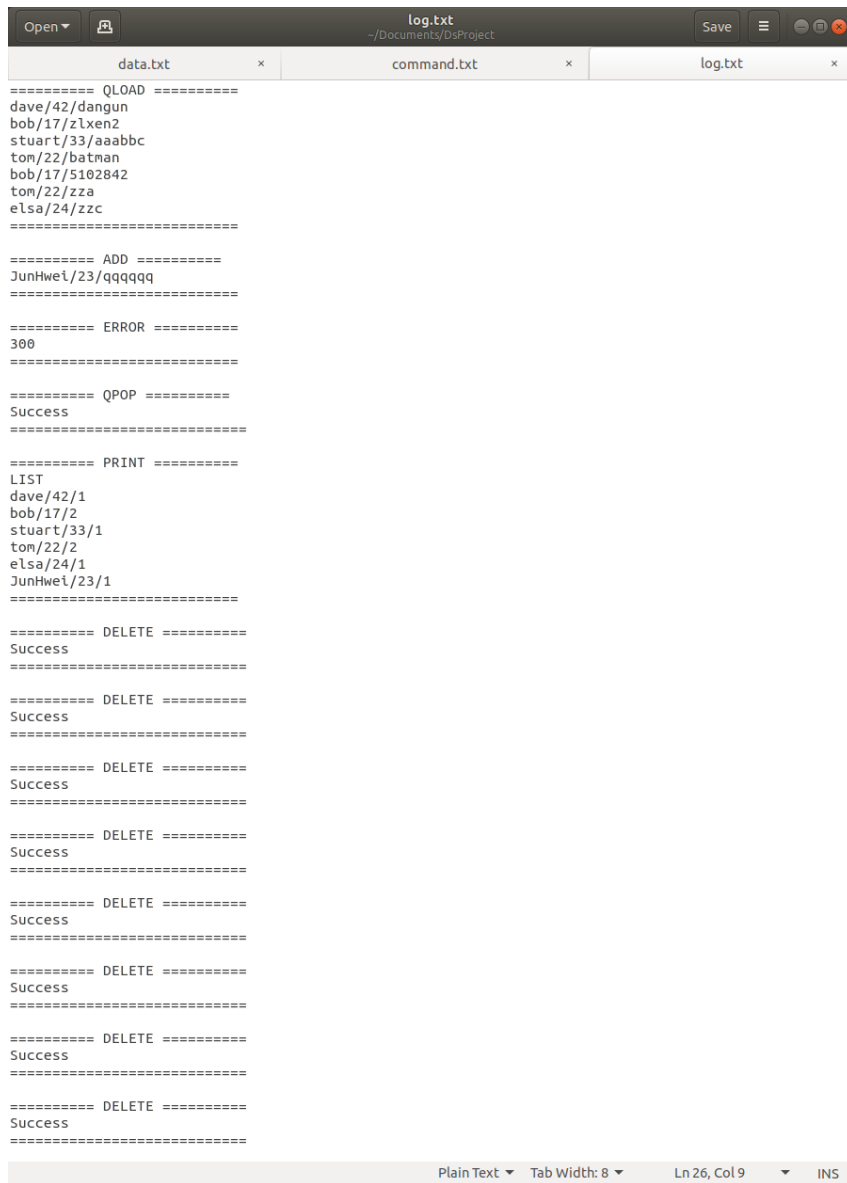


A screenshot of a text editor window titled "command.txt" with the path "~/Documents/DsProject". The window shows two tabs: "data.txt" and "command.txt". The "command.txt" tab is active and contains the following commands:

```
QLOAD
ADD JunHwei 23 qqqqqq
QPOP 9
QPOP 8
PRINT L
DELETE dangun
DELETE zlxen2
DELETE aaabbc
DELETE batnan
DELETE 5102842
DELETE zza
DELETE zzc
DELETE qqqqqq
PRINT L
PRINT B level
ADD JunHwei 23 qqqqqq
QLOAD
QPOP 8
PRINT L
PRINT B PRE
PRINT B IN
PRINT B POST
PRINT B LEVEL
HLOAD
PRINT H
SEARCH JunHwei
SEARCH user bob
SEARCH id qqqqqq
EXIT
```

The status bar at the bottom indicates "Plain Text", "Tab Width: 8", "Ln 26, Col 15", and "INS".

C. log.txt



The screenshot shows a text editor window titled "log.txt" with a menu bar (Open, Save, icons) and a toolbar. The editor has three tabs: "data.txt", "command.txt", and "log.txt". The "log.txt" tab is active and displays the following text:

```
===== QLOAD =====
dave/42/dangun
bob/17/zlxen2
stuart/33/aaabbc
tom/22/batman
bob/17/5102842
tom/22/zza
elsa/24/zzc
=====

===== ADD =====
JunHwe1/23/qqqqq
=====

===== ERROR =====
300
=====

===== QPOP =====
Success
=====

===== PRINT =====
LIST
dave/42/1
bob/17/2
stuart/33/1
tom/22/2
elsa/24/1
JunHwe1/23/1
=====

===== DELETE =====
Success
=====

===== DELETE =====
Success
=====

===== DELETE =====
Success
=====

===== DELETE =====
Success
=====

===== DELETE =====
Success
=====

===== DELETE =====
Success
=====

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

The status bar at the bottom indicates "Plain Text", "Tab Width: 8", "Ln 26, Col 9", and "INS".

```
log.txt
~/Documents/DsProject

data.txt x command.txt x log.txt x

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

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

===== ADD =====
JunHwei/23/qqqqqq
=====

===== QLOAD =====
JunHwei/23/qqqqqq
dave/42/dangun
bob/17/zlxen2
stuart/33/aaabbc
tom/22/batman
bob/17/5102842
tom/22/zza
elsa/24/zzc
=====

===== QPOP =====
Success
=====

===== PRINT =====
LIST
JunHwei/23/1
dave/42/1
bob/17/2
stuart/33/1
tom/22/2
elsa/24/1
=====

===== PRINT =====
BST PRE
qqqqqq/JunHwei
dangun/dave
aaabbc/stuart
5102842/bob
batman/tom
zlxen2/bob
zza/tom
zzc/elsa
=====

===== PRINT =====
BST IN
5102842/bob
aaabbc/stuart
batman/tom
dangun/dave
qqqqqq/JunHwei
zlxen2/bob
zza/tom
zzc/elsa
=====

===== PRINT =====
BST POST
=====

Plain Text Tab Width: 8 Ln 62, Col 5 INS
```

```
log.txt
~/Documents/DsProject

data.txt x command.txt x log.txt x

batman/tom
dangun/dave
qqqqqq/JunHwe1
zlxen2/bob
zza/tom
zxc/elsa
=====

===== PRINT =====
BST POST
5102842/bob
batman/tom
aaabbc/stuart
dangun/dave
zxc/elsa
zza/tom
zlxen2/bob
qqqqqq/JunHwe1
=====

===== PRINT =====
BST LEVEL
qqqqqq/JunHwe1
dangun/dave
zlxen2/bob
aaabbc/stuart
zza/tom
5102842/bob
batman/tom
zxc/elsa
=====

===== HLOAD =====
Success
=====

===== PRINT =====
Heap
1/40
1/30
1/10
3/20
=====

===== ERROR =====
400
=====

===== SEARCH =====
User
bob/17
zlxen2
5102842
=====

===== SEARCH =====
ID
qqqqqq/JunHwe1
=====

===== EXIT =====
Success
=====

Plain Text Tab Width: 8 Ln 62, Col 5 INS
```

D. explanation

- i. QLOAD를 통해 데이터를 불러오는 것에 성공하였다.
- ii. ADD를 통해 데이터를 1개 입력하는 것에 성공하였다.
- iii. QPOP을 통해 현재 데이터의 개수인 8보다 더 많이 POP하려 하자 오류를 출력하였다.
- iv. QPOP을 통해 Queue에 있는 데이터를 모두 List와 BST에 옮겨왔다.
- v. PRINT L을 통해 List를 출력하였다.
- vi. DELETE를 통해 dangun를 제거하는데 성공하였다.
- vii. DELETE를 통해 zlxen2를 제거하는데 성공하였다.
- viii. DELETE를 통해 aaabbc를 제거하는데 성공하였다.
- ix. DELETE를 통해 batman을 제거하는데 성공하였다.
- x. DELETE를 통해 5102842를 제거하는데 성공하였다.

- xi. DELETE를 통해 zza를 제거하는데 성공하였다.
- xii. DELETE를 통해 zzc를 제거하는데 성공하였다.
- xiii. DELETE를 통해 qqqqqq를 제거하는데 성공하였다.
- xiv. PRINT L을 하였지만 List가 비어있기 때문에 오류를 출력하였다.
- xv. PRINT B level을 하였지만 BST가 비어있기 때문에 오류를 출력하였다.
- xvi. ADD를 통해 데이터를 1개 입력하는 것에 성공하였다.
- xvii. QLOAD를 통해 데이터를 불러오는 것에 성공하였다.
- xxviii. QPOP을 통해 데이터를 List와 BST로 로드하는 것에 성공했다.
- xix. PRINT L을 통해 List를 출력하였다.
- xx. PRINT B PRE를 통해 Pre-order의 형식으로 BST를 출력하였다.
- xxi. PRINT B IN를 통해 in-order의 형식으로 BST를 출력하였다.
- xxii. PRINT B POST를 통해 Post-order의 형식으로 BST를 출력하였다.
- xxiii. PRINT B LEVEL를 통해 Level-order의 형식으로 BST를 출력하였다.
- xxiv. HLOAD를 통해 Heap에 데이터를 구성하는데 성공했다.
- xxv. PRINT H를 통해 Heap을 Level-order로 출력하는데 성공했다.
- xxvi. SEARCH를 하였지만 user 또는 id를 입력을 받지 못하여 에러를 출력하였다.
- xxvii. SEARCH를 하여 user bob을 List에서 검색하는데 성공하였다.
- xxviii. SEARCH를 하여 id qqqqqq를 BST에서 검색하는데 성공하였다.
- xxix. EXIT을 통해 프로그램을 종료하는데 성공하였다.

5. Consideration

해당 과제를 진행하면서 한 번에 4가지의 데이터 구조를 만들어 연결하는 것에 처음에는 어려움을 느꼈다. 아무래도 모든 데이터 구조의 원리를 정확히 숙지해야 하기 때문에 이를 공부하는 데에 시간이 지체되었다. 또한 처음으로 Linux 환경에서 코드를 짜고 실행하는 과정을 거쳤기 때문에 visual studio에 익숙한 나에게는 이 또한 적응되지 않았다. 특히 코드를 완성한 후 검증을 할 때, 한 줄씩 읽어가며 검증하고 싶었으나 하는 방법을 몰랐다. 하지만 코드를 짤 때의 어려움은 visual studio code를 설치하여 코드를 작성함으로써 어느정도 익숙하게 작업을 수행할 수 있었다. 다만 검증을 하는 과정은 아직 익숙하지 않아 해당 코드를 윈도우로 옮겨 visual studio 상에서 검증을 하여 정상적으로 고치고 다시 옮기는 작업을 수행하였다. 또한 필자는 mac을 추가로 사용하고 있어 OS를 바꿔가며 사용하는 일이 많았다. 하지만 데이터를 gitHub상의 개인 레퍼지토리에 올려 관리함으로써 OS가 바뀌는 환경에서도 코드의 최신화를 계속 유지할 수 있었다. 또한 과제를 수행하면서 Linux상의 terminal에서 파일을 만들고 이를 실행하는 과정을 연습하여, 리눅스 Terminal 명령어에 더욱 익숙해질 수 있는 기회가 생겼다.