Data Structure Project #1 보고서

컴퓨터정보공학부

2024402049 정하영

1. Introduction

<개요>

이번 프로젝트는 이진 탐색 트리 (Binary Search Tree), 큐 (Queue), 순환 연결 리스트 (Circular Linked List) 자료구조를 활용하여 음악 플레이 리스트 관리 시스템을 구현한다. 사용자는 텍스트 파일에서 음악 데이터를 불러오고, 여러가지 자료구조에 저장하여 효율적으로 검색, 추가, 삭제할 수 있다.

<사용된 자료구조>

MusicQueue (Queue)

Music_List.txt 파일로부터 음악 데이터를 저장하는 FIFO(First-In_First-Out) 구조이다. 최대 100개의 음악 데이터를 저장할 수 있고, QPOP 명령어를 통해 데이터를 방출하여 BST를 만들 때 사용한다. 이중 연결 리스트로 구현돼서 양방향으로 접근이 가능하다.

ArtistBST (가수 기준 Binary Search Tree)

가수 이름을 key로 해서 사전순으로 정렬된 이진 탐색 트리이다. 각 노드는 해당 가수의 모든 곡 정보 (노래 제목, 재생 시간) 를 벡터 형태로 저장한다. 중위 순회 (In-order Traversal) 를 통해 가수 이름을 사전 순으로 출력이 가능하다.

TitleBST (노래 제목 기준 Binary Search Tree)

노래 제목을 key로 해서 사전순으로 정렬된 이진 탐색 트리이다. 각 노드는 같은 제목을 가진 곡을 여러 가수가 불렀을 경우, 곡 정보 (가수 이름, 재생 시간) 를 벡터 형태로 저장한다. 중위 순회 (In-order Traversal) 를 통해 제목 이름을 사전 순으로 출력이 가능하다.

PlayList (순환 연결 리스트 Circular Linked List)

BST에 저장된 음악 데이터를 가지고 사용자가 원하는 곡을 선택해서 구성하는 노래 재생 목록이다. Circular Linked List 구조로 구현된다. 최대 10곡까지 저장 가능하고, 곡이 추가되거나 삭제될 때마다 총 재생 시간이 자동으로 갱신된다.

<동작 명령어>

LOAD : Music_List.txt 파일에서 음악 데이터를 읽어서 MusicQueue에 저장한다.

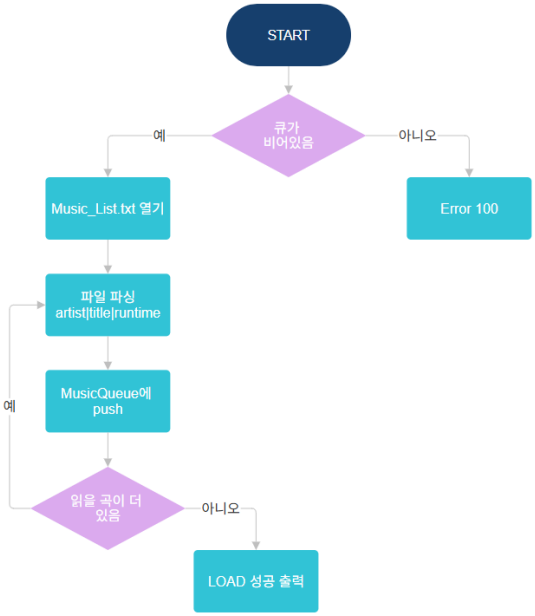
ADD : 사용자가 직접 음악 데이터를 MusicQueue에 추가한다.

QPOP : MusicQueue에서 데이터를 방출해서 ArtistBST와 TitleBST에 넣어준다.

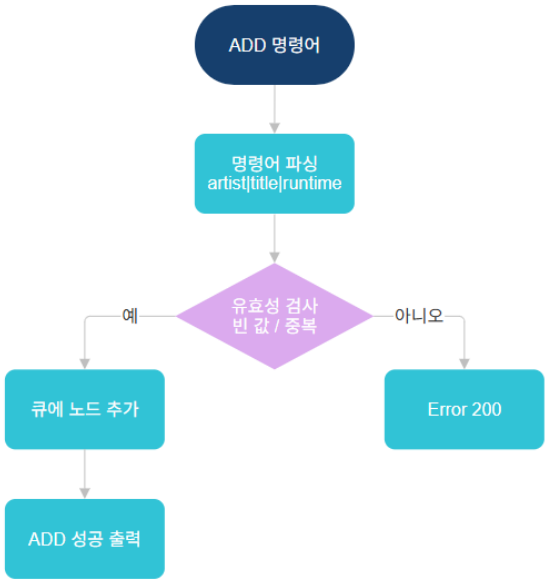
SEARCH : 가수 이름, 노래 제목, 또는 곡을 BST에서 검색한다.
MAKEPL : BST에 저장된 곡을 PlayList에 추가해서 플레이 리스트를 생성한다.
PRINT : ArtistBST, TitleBST, PlayList의 데이터를 출력한다.
DELETE : 가수, 제목, 또는 곡을 자료구조에서 삭제한다.

2. Flowchart

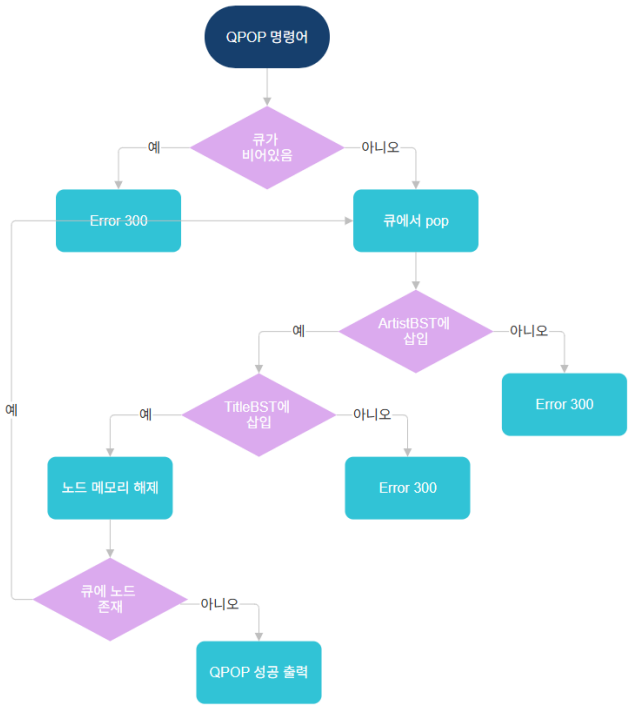
<LOAD 프로세스>



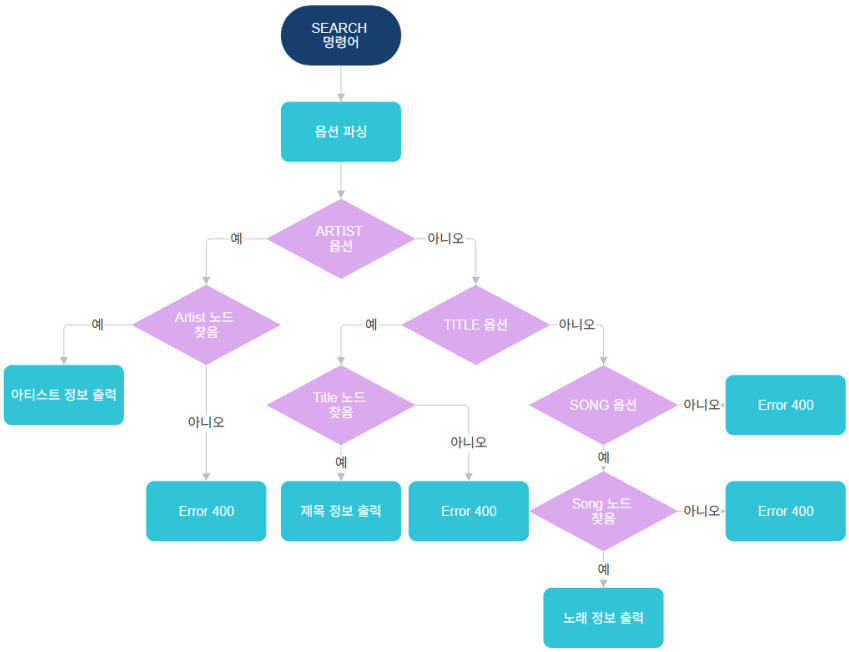
<ADD 프로세스>



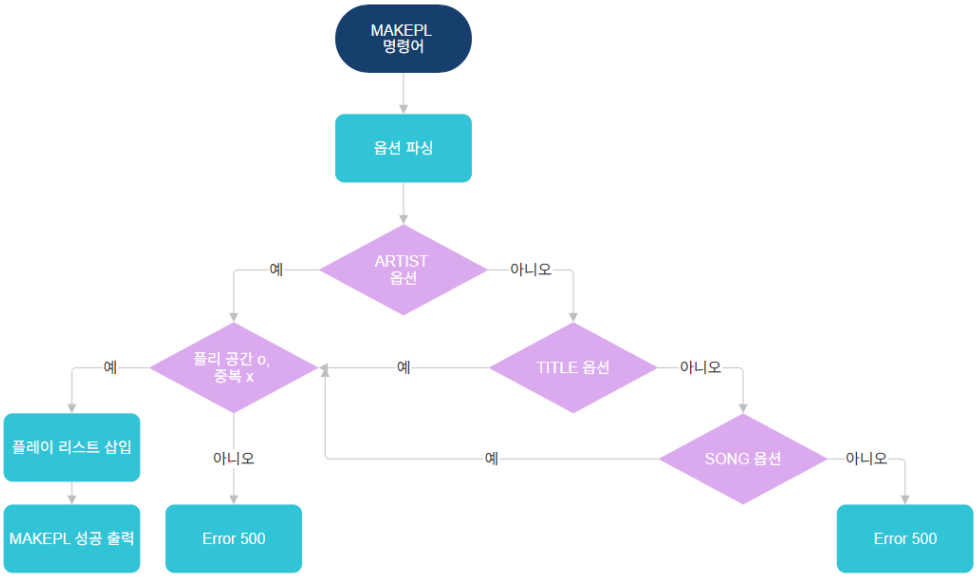
<QPOP 프로세스>



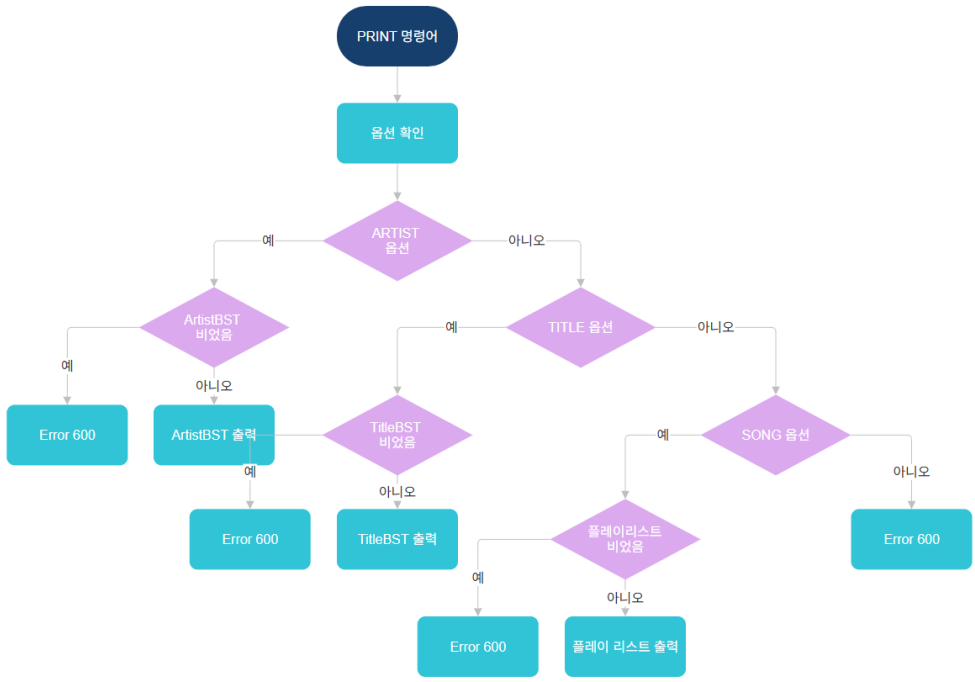
<SEARCH 프로세스>



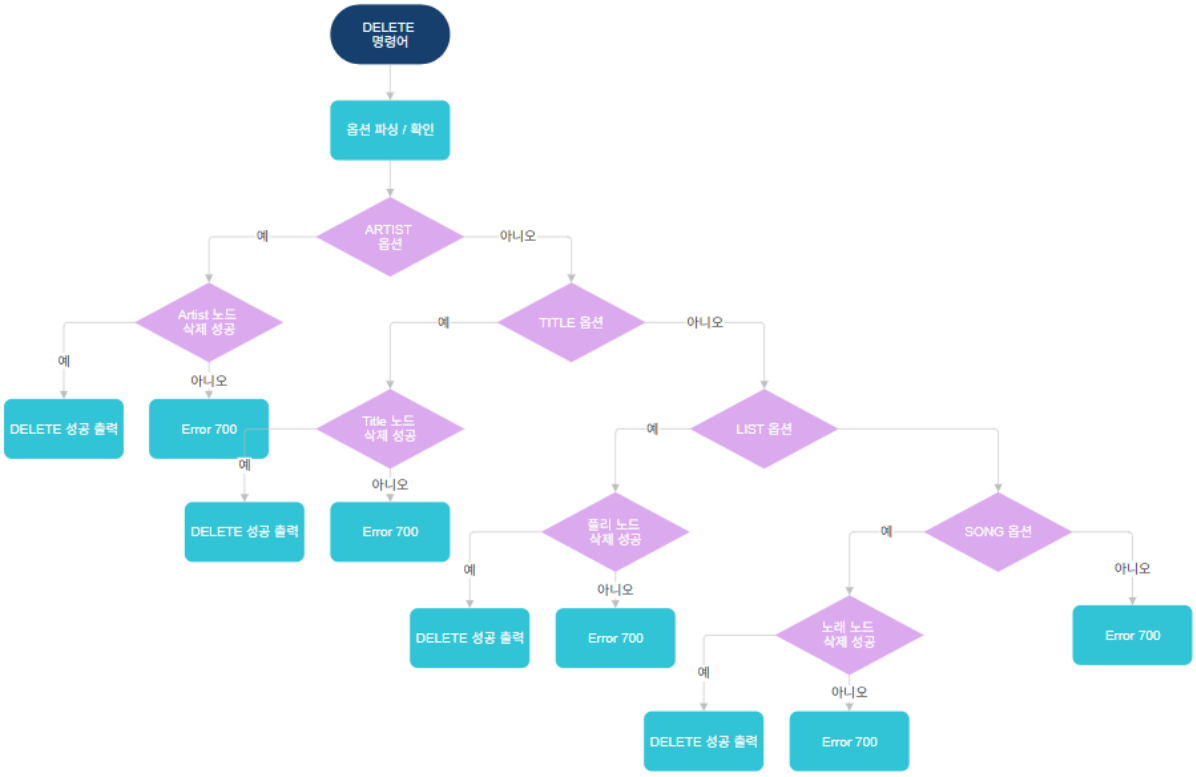
<MAKEPL 프로세스>



<PRINT 프로세스>



<DELETE 프로세스>



3. Algorithm

<전체 프로그램 구조>

Manager 클래스를 중심으로 MusicQueue, ArtistBST, TitleBST, PlayList의 4가지 주요 자료구조를 관리한다. command.txt 파일로부터 명령어를 읽어서 처리하고, 결과는 모두 log.txt에 저장한다.

Music_List.txt → MusicQueue (LOAD)

MusicQueue → ArtistBST & TitleBST (QPOP)

BST → PlayList (MAKEPL)

〈Manager 알고리즘〉

Manager.h, Manager.cpp

과제의 메인으로 제어하는 클래스이다. 명령어 파일을 입력 받아서 MusicQueue, ArtistBST, TitleBST, PlayList 자료구조를 관리한다. 각 명령 실행 결과와 오류 메시지를 log.txt 파일에 출력한다.

- convertToSeconds(run_time)
  mm : ss 형태의 문자열을 초 단위로 변환한다.
  ① :의 위치를 찾고, 찾지 못했을 경우, 0을 반환한다.
  ② stoi() 함수를 사용해서 분과 초를 추출한 후에, 초 형태로 반환한다.

  시간 복잡도 : O(1)

- run(const char* command)
  ① 외부 명령어 파일을 읽어서 한 줄씩 실행한다.
  ② 각 명령어에 맞는 기능 함수를 호출한다.
  ③ 결과 및 오류를 log.txt에 기록한다.

  시간 복잡도 : O(n), n = 명령어 개수

- LOAD() 텍스트 파일의 노래 정보를 불러오는 명령어
  ① Queue가 비어있지 않으면 에러를 출력하고 리턴한다.
  ② 파일에서 곡 정보를 읽어서 MusicQueue에 저장한다.

  시간 복잡도 : O(n)

- ADD(const string& line) 데이터 추가 명령어
  ① '|'를 기준으로 명령어 뒷부분을 파싱한다.
  ② 가수, 제목, 재생 시간이 비었거나 queue에 이미 존재하는 경우 에러를 출력하고, 리턴한다.
  ③ Queue에 새로운 노래 newNode를 만들어 푸시한다.

  시간 복잡도 : O(1)

- QPOP() 데이터 POP 명령어
  ① Queue가 비어있으면 에러를 출력하고 리턴한다.
  ② 모든 노래를 MusicQueueNode::pop() 함수를 호출하여 pop해준다.
  ③ 각 BST에 삽입해주고, 실패하면 에러를 출력한다.

  시간 복잡도 : O(n)

- SEARCH(const string& line) 노래 검색 명령어
  ① 탐색하는 기준의 string에 따라 케이스를 분류한다.
  ② 각 노드에 따라 탐색해주고, 존재하지 않으면 에러를 출력한다.

  시간 복잡도 : O(log n)

- MAKEPL(const string& line) 플레이 리스트 추가 명령어
  ① 플레이 리스트에 추가할 기준의 string에 따라 케이스를 분류한다.
  ② 각 노드에서 노래가 존재하지 않거나 공간이 부족하면 에러를 출력한다.

  시간 복잡도 : O(k), k = 추가되는 노래 개수

- PRINT(const string& line) 출력 명령어
  ① 출력 대상의 기준 string에 따라 케이스를 분류한다.
  ② 출력 대상의 노드가 비어있으면 에러를 출력한다.

  시간 복잡도 : O(n)

- DELETE(const string& line) 삭제 명령어
  ① 삭제 대상의 기준 string에 따라 케이스를 분류한다.
  ② 각 노드에서 노래가 존재하지 않으면 에러를 출력한다.
  ③ 모든 BST와 플레이 리스트에서 해당 곡들을 삭제한다.
  ④ 해당 노드에서 삭제 실패 시 에러를 출력한다.

  시간 복잡도 : O(log n + k)

- EXIT() 종료 명령어
  프로그램 정상 종료 메시지를 출력하고, 로그 파일을 닫는다.
  시간 복잡도 : O(1)

<MusicQueue 알고리즘>

MusicQueueNode.h

Music Queue에서 각 노래 정보를 저장하고, 양방향 연결 리스트의 노드로 사용되는 클래스이다. 각 노드는 이전 노드 (prev)와 다음 노드 (next)를 가리키며, 가수 이름(artist), 노래 제목(title), 재생 시간(run_time) 정보를 포함한다.

- insert() 노드 삽입 함수
  현재 노드의 다음 위치에 새로운 노드를 삽입한다.
  ① 새 노드(newNode)의 next 포인터를 현재 노드의 next로 연결한다.
  ② 새 노드의 prev 포인터를 현재 노드 (this)로 연결한다.
  ③ 현재 노드의 다음 노드가 존재한다면, 그 노드의 prev를 새 노드로 변경한다.
  ④ 현재 노드의 next를 새 노드로 연결한다.

  시간 복잡도 : O(1) 단순 포인터 교환만 수행

- exist() 노드 탐색 함수
  Queue 내에 특정 가수와 노래 제목이 일치하는 노드를 탐색한다.
  ① 현재 노드(this)부터 next를 통해서 탐색을 한다.
  ② 각 노드에서 artist와 title이 모드 일치하면 true를 리턴한다.
  ③ 끝까지 탐색했지만 찾지 못한 경우, false를 리턴한다.

  시간 복잡도 : O(n) 리스트 전체를 순회

- Getter 함수 (getArtist(), getTitle(), getRunTime())
  각 노드가 가진 음악 정보를 반환한다.
  시간 복잡도 : O(1)

MusicQueue.h, MusicQueue.cpp
MusicQueue 클래스는 음악 재생을 위한 Queue를 구현한 자료구조이다. 곡을 FIFO(First-In-First-Out) 순서롤 관리한다. 각 노드는 MusicQueueNode 객체로 구성되고, 이중 연결 리스트 구조를 사용한다.

- MusicQueue() 생성자
  head, rear를 nullptr로 초기화 하고, size = 0으로 초기화한다.
  시간 복잡도 : O(1)
- ~MusicQueue() 소멸자
  큐가 비어있지 않으면 pop()을 반복해서 모든 노드의 메모리를 해제한다.
  시간 복잡도 : O(n) 노드의 수가 n개
- empty() 큐가 비었는지 확인 함수
  size가 0이면 true, 아니면 false를 반환한다.
  시간 복잡도 : O(1)
- exist() 노드 존재 여부 확인 함수
  Queue 내에 특정 가수와 노래 제목이 일치하는 노드를 탐색한다.
  ① head부터 탐색해서 MusicQueueNode::exist()로 확인한다.
  ② 노드가 존재하면 true, 아니면 false를 반환한다

  시간 복잡도 : O(n)

- push() 노드 추가 함수
  ① exist()로 중복 검사 후, 이미 존재하면 노드 삭제 후 종료
  ② 큐가 비어있으면 head = rear = newNode
  ③ 큐가 비어있지 않으면 rear의 next 포인터를 newNode로 연결한다.
  ④ newNode의 prev를 rear로 연결한다.
  ⑤ rear을 newNode로 지정한다.
  ⑥ size를 +1 해준다.

  시간 복잡도 : O(n) (exist 검사 포함), O(1) (삽입 자체)

- pop() 노드 제거 함수
  ① empty()로 비어있는지 확인 후, 비어있으면 nullptr 반환
  ② 임시 노드를 head 노드로 가리킨다.
  ③ head를 head의 next 노드로 가리킨다.
  ④ head가 존재하면 head의 prev는 nullptr을 가리키고,
  head가 존재하지 않으면 rear가 nullptr을 가리킨다.
  ⑤ 임시 node를 연결 해제해준다.
  ⑥ size를 -1 해주고, 임시 노드를 반환한다.

  시간 복잡도 : O(1)

- front() 맨 앞 노드 조회
  ① empty()로 비어있는지 확인 후, 비어있으면 nullptr 반환
  ② 비어있지 않으면 head 반환

  시간 복잡도 : O(1)

<ArtistBST 알고리즘>
ArtistBSTNode.h
가수의 노래 정보를 저장하는 이진 탐색 트리(BST) 노드이다. 각 노드는 가수 이름(artist), 노래 제목 목록(titles), 노래 재생 시간 목록(run_times), 노래 개수(count)를 가진다. left와 right는 자식 노드를 가리키며 BST 구조를 형성한다.

- set() 노래 추가 함수
  ① searchTitle()로 이미 존재하는지 확인 후, 존재하는 제목이면 false 반환
  ② 존재하지 않으면 titles와 run_times에 추가, count++, true 반환

  시간 복잡도 : O(m), m = 현재 가수의 노래 개수

- 출력 함수

printElement(title) 특정 곡 출력 함수

① 순회하며 제목과 일치하는 노래만 출력한다.

시간 복잡도 : O(m)

print() 모든 곡 출력 함수
① 모든 titles, run_times를 순회해서 출력한다.

시간 복잡도 : O(m)

- 노래 검색 함수
  search(node, artistSearch) 노드 검색 함수
  artistBST구조에서 재귀 탐색을 한다.
  ① artist < node.artist일 경우, 왼쪽을 탐색한다.
  ② artist > node.artist일 경우, 오른쪽을 탐색한다.
  ③ artist == node.artist일 경우. 반환을 한다.

  시간 복잡도 : O(log n) 균형 BST일 경우

  searchTitle(titleSearch) 노래 검색 함수

  titles 벡터를 순회하며 일치하는 제목을 찾을 경우, true를 반환한다.

  시간 복잡도 : O(m)

- deleteTitle() 노래 삭제 함수
  ① 제목이 존재하면 벡터에서 제거 후, count--, true를 반환한다.
  ② 제목이 존재하지 않으면 false를 반환한다.

  시간 복잡도 : O(m)

- Getter / Setter 함수 (getArtist(), setArtist(newArtist), getTitle(index), getRunTime(index), getCount())
  Getter : 각 정보를 반환한다.
  Setter : 각 정보를 변경한다.
  시간 복잡도 : O(1)

ArtistBST.h, ArtistBST.cpp

가수의 노래 정보를 저장하고 관리하는 이진 탐색 트리(BST)를 구현한다. 노드 단위로 가수 이름을 저장하고, 가수의 이름 기준으로 정렬해서 탐색, 삽입, 삭제, 출력 기능을 수행한다.

- ArtistBST() 생성자
  root, parent를 nullptr로 초기화한다.
  시간 복잡도 : O(1)
- ~ArtistBST() 소멸자
  deleteTree(root) 함수를 호출해서 모든 노드의 메모리를 재귀적으로 해제한다.
  +deleteTree() 모든 노드 메모리 해제 보조 함수
  ① 노드가 존재하지 않는다면, 리턴한다.
  ② 존재한다면, 왼쪽 트리부터 오른쪽 트리를 재귀적으로 삭제한다.
  시간 복잡도 : O(n), n = 가수 이름 수
- insert() 가수 삽입 함수
  ① insertFunc()을 통해 root를 기준으로 재귀적인 삽입을 수행한다.
  ② 삽입 성공 여부를 반환한다.
  +insertFunc() 가수 삽입 보조 함수
  ① 가수가 존재하지 않으면 새로운 노드를 생성한다.
  ② 가수가 이미 존재하면 artistBST에서 해당 가수의 위치를 찾아서 ArtistBSTNode::set()을 호출해서 노래를 추가한다.

  시간 복잡도 : O(log n) (BST가 균형적일 때), O(n) (편향트리)

- search() 가수 탐색 함수
  ① ArtistBSTNode::search()를 호출해서 root부터 탐색한다.
  ② 찾으면 해당 노드 반환, 없으면 nullptr를 반환한다.

  시간 복잡도 : O(logn) (BST가 균형적일 때), O(n) (편항트리)

- print() 가수 기준 출력 함수
  ① printFunc()을 통해 중위 순회로 재귀적인 출력을 한다.

  +printFunc() 가수 기준 출력 보조 함수

  ① 노드가 존재하지 않으면 리턴한다.
  ② 노드가 존재한다면 왼쪽 트리부터 오른쪽 트리 순으로 중위 순회한다.
- 삭제 함수
  delete_node() 가수 노드 삭제 함수
  ① ArtistBSTNode::search() 함수를 통해 가수 이름을 가진 노드가 존재하는지 탐색한다.
  ② 삭제하고자 하는 가수 이름을 가진 노드가 존재하지 않는다면 false를 반환한다.

③ 존재한다면, deleteFunc()을 호출해서 삭제 후, true를 반환한다.

+deleteFunc() 가수 노드 삭제 보조 함수

① 노드가 존재하지 않는 경우, 리턴한다.
② 자식이 없는 경우, 바로 삭제한다.
③ 자식이 왼쪽 혹은 오른쪽에 하나 있는 경우, 자식으로 연결한다.
④ 자식이 둘 다 있는 경우, 오른쪽 서브트리에서 최솟값을 찾아서 교체한 후 삭제한다.

시간 복잡도 : O(log n) (BST가 균형적일 때), O(n) (편향트리)

delete_song() 노래 삭제 함수

① ArtistBSTNode::search() 함수를 통해 가수 이름을 가진 노드가 존재하는지 탐색한다.
② 삭제하고자 하는 가수 이름을 가진 노드가 존재하지 않는다면 false를 반환한다.
③ 존재한다면, ArtistBSTNode::deleteTitle() 함수를 통해 가수의 노드에서 지우고자 하는 제목을 가진 노래를 삭제한다.
④ 가수 노드의 노래가 0개일 때, deleteFunc()을 호출해서 가수 노드를 삭제한다.

시간 복잡도 : O(log n + m)

가수 탐색 : O(log n) + 노래 삭제 : O(m), m = 해당 가수의 노래 개수

- empty() artistBST가 비었는지 확인 함수
  root == nullptr이면 true를 반환한다.
  시간 복잡도 : O(1)

<TitleBST 알고리즘>
TitleBSTNode.h
해당 제목을 가진 노래 정보를 저장하는 이진 탐색 트리(BST) 노드이다. 각 노드는 가수 이름 목록(artists), 노래 제목(title), 노래 재생 시간 목록(run_times), 노래 개수(count)를 가진다. left와 right는 자식 노드를 가리키며 BST 구조를 형성한다.

- set() 노래 추가 함수
  ③ searchArtist()로 이미 존재하는지 확인 후, 존재하는 가수면 false 반환
  ④ 존재하지 않으면 artists와 run_times에 추가, count++, true 반환

시간 복잡도 : O(m), m = 해당 제목의 노래 개수

- 출력 함수

  printElement(artist) 특정 곡 출력 함수

  ② 순회하며 가수와 일치하는 노래만 출력한다.

  시간 복잡도 : O(m)

  print() 모든 곡 출력 함수
  ② 모든 artists, run_times를 순회해서 출력한다.

  시간 복잡도 : O(m)

- 노래 검색 함수
  search(node, titleSearch) 노드 검색 함수
  titleBST구조에서 재귀 탐색을 한다.
  ④ title < node.title일 경우, 왼쪽을 탐색한다.
  ⑤ title > node.title일 경우, 오른쪽을 탐색한다.
  ⑥ title == node.title일 경우. 반환을 한다.

  시간 복잡도 : O(log n) 균형 BST일 경우

  searchArtist(artistSearch) 노래 검색 함수

  artists 벡터를 순회하며 일치하는 가수를 찾을 경우, true를 반환한다.

  시간 복잡도 : O(m)

- deleteArtist() 노래 삭제 함수
  ③ 가수가 존재하면 벡터에서 제거 후, count--, true를 반환한다.
  ④ 가수가 존재하지 않으면 false를 반환한다.

  시간 복잡도 : O(m)

- Getter / Setter 함수 (getTitle(), setTitle(newTitle), getArtist(index), getRunTime(index), getCount())
  Getter : 각 정보를 반환한다.
  Setter : 각 정보를 변경한다.
  시간 복잡도 : O(1)

TitleBST.h, TitleBST.cpp

해당 제목을 가지고 있는 노래 정보를 저장하고 관리하는 이진 탐색 트리(BST)를

구현한다. 노드 단위로 노래 제목 이름을 저장하고, 노래 제목을 기준으로 정렬해서 탐색, 삽입, 삭제, 출력 기능을 수행한다.

- TitleBST() 생성자
  root, parent를 nullptr로 초기화한다.
  시간 복잡도 : O(1)
- ~TitleBST() 소멸자
  deleteTree(root) 함수를 호출해서 모든 노드의 메모리를 재귀적으로 해제한다.
  +deleteTree() 모든 노드 메모리 해제 보조 함수
  ③ 노드가 존재하지 않는다면, 리턴한다.
  ④ 존재한다면, 왼쪽 트리부터 오른쪽 트리를 재귀적으로 삭제한다.
  시간 복잡도 : O(n), n = 노래 제목 이름 수
- insert() 제목 삽입 함수
  ③ insertFunc()을 통해 root를 기준으로 재귀적인 삽입을 수행한다.
  ④ 삽입 성공 여부를 반환한다.
  +insertFunc() 제목 삽입 보조 함수
  ③ 제목이 존재하지 않으면 새로운 노드를 생성한다.
  ④ 제목이 이미 존재하면 titleBST에서 해당 제목의 위치를 찾아서 TitleBSTNode::set()을 호출해서 노래를 추가한다.

  시간 복잡도 : O(log n) (BST가 균형적일 때), O(n) (편향트리)

- search() 제목 탐색 함수
  ③ TitleBSTNode::search()를 호출해서 root부터 탐색한다.
  ④ 찾으면 해당 노드 반환, 없으면 nullptr를 반환한다.

  시간 복잡도 : O(logn) (BST가 균형적일 때), O(n) (편항트리)

- print() 제목 기준 출력 함수
  ② printFunc()을 통해 중위 순회로 재귀적인 출력을 한다.

  +printFunc() 제목 기준 출력 보조 함수

  ③ 노드가 존재하지 않으면 리턴한다.
  ④ 노드가 존재한다면 왼쪽 트리부터 오른쪽 트리 순으로 중위 순회한다.
- 삭제 함수
  delete_node() 제목 노드 삭제 함수
  ④ TitleBSTNode::search() 함수를 통해 해당 제목을 가진 노드가 존재하는지 탐색한다.

⑤ 삭제하고자 하는 노래 제목을 가진 노드가 존재하지 않는다면 false를 반환한다.
⑥ 존재한다면, deleteFunc()을 호출해서 삭제 후, true를 반환한다.

+deleteFunc() 제목 노드 삭제 보조 함수

⑤ 노드가 존재하지 않는 경우, 리턴한다.
⑥ 자식이 없는 경우, 바로 삭제한다.
⑦ 자식이 왼쪽 혹은 오른쪽에 하나 있는 경우, 자식으로 연결한다.
⑧ 자식이 둘 다 있는 경우, 오른쪽 서브트리에서 최솟값을 찾아서 교체한 후 삭제한다.

시간 복잡도 : O(log n) (BST가 균형적일 때), O(n) (편향트리)

delete_song() 노래 삭제 함수

⑤ TitleBSTNode::search() 함수를 통해 해당 제목을 가진 노드가 존재하는지 탐색한다.
⑥ 삭제하고자 하는 제목 이름을 가진 노드가 존재하지 않는다면 false를 반환한다.
⑦ 존재한다면, TitleBSTNode::deleteArtist() 함수를 통해 해당 제목의 노드에서 지우고자 하는 가수를 삭제한다.
⑧ 해당 제목 노드의 노래가 0개일 때, deleteFunc()을 호출해서 해당 제목 노드를 삭제한다.

시간 복잡도 : O(log n + m)

제목 탐색 : O(log n) + 제목 삭제 : O(m), m = 해당 제목을 가진 노래 개수

- empty() titleBST가 비었는지 확인 함수
  root == nullptr이면 true를 반환한다.

시간 복잡도 : O(1)

<PlayList 알고리즘>

PlayListNode.h

플레이리스트의 개별적인 노래를 표현하는 이중 연결 리스트의 노드 클래스이다. 각 노드는 하나의 곡 (artist, title, runtime)을 저장하고, 이전 노드(prev)와 다음 노드(next)를 가리켜서 양방향 순회가 가능한 리스트 구조를 형성한다.

- PlayListNode() 생성자
  Artist, title, runtime_sec를 초기화하고, prev 및 next를 nullptr로 설정해서 독

립적인 노드로 만든다.

시간 복잡도 : O(1)

- set() 노드 삽입 함수
  ① newNode의 next는 현재 노드의 다음 노드를 가리킨다.
  ② newNode의 prev는 현재 노드를 가리킨다.
  ③ 기존의 next 노드가 존재하면, 그 노드의 prev를 newNode로 변경한다.
  ④ 현재 노드의 next를 newNode로 연결한다.
- Getter 함수 (getArtist(), getTitle(), getRunTimeSec())
  각 노드가 가진 음악 정보를 반환한다.
  시간 복잡도 : O(1)

PlayList.h, PlayList.cpp

PlayListNode의 객체들을 연결해서 원형 이중 연결 리스트 형태로 플레이 리스트를 관리한다. 노드의 삽입, 삭제, 탐색, 출력, 및 전체 재생 시간 계산 등을 수행한다.

- PlayList() 생성자
  head를 nullptr, count와 time을 0으로 초기화 한다.
  시간 복잡도 : O(1)
- ~PlayList() 소멸자
  원형 리스트를 순회하면서 모든 노드를 메모리에서 해제한다.
  시간 복잡도 : O(n)
- insert_node(artist, title, runtime_sec) 노래 삽입 함수
  ① full() 함수를 활용하여 플레이 리스트가 가득 차있는지 확인하고, exist(artist, title) 함수를 활용하여 노래가 플레이 리스트에 존재하는지 확인한다.
  ② 플레이 리스트가 가득 차있거나, 해당 노래가 이미 존재한다면 false를 반환한다.
  ③ 새로운 노드를 만들어 해당 노래의 가수 이름, 제목, 재생 시간을 저장한다.
  ④ 플레이 리스트가 비어있으면 head에 새로운 노드를 넣어주고, head의 next와 prev가 head를 가리키게 연결한다.
  ⑤ 비어있지 않다면, head의 prev에 newNode를 위치시킨다.
  ⑥ count++, 총 재생시간에 해당 노래의 시간을 더해주고, true를 반환한다.

  시간 복잡도 : O(1)

- delete_node(artist, title) 노래 삭제 함수
  ① 플레이 리스트가 비어있으면 false를 반환한다.

② head부터 순회하며 특정 가수와 제목이 일치하는 삭제 대상 노래를 탐색 성공하면, count—와 총 재생시간에서 해당 노래 시간만큼 빼준다.
③ 리스트에 해당 노래만 존재할 때, 삭제 후, head를 nullptr로 설정한다.
④ head에 위치한 노래를 삭제할 때, head를 head->next로 이동시킨다.
⑤ 링크를 업데이트 시킨 후, 해당 노드를 삭제하고, true를 반환한다.
⑥ 해당 노래를 찾지 못한 경우, false를 반환한다.

시간 복잡도 : O(n)

- empty() 플레이 리스트가 비었는지 확인 함수
  count가 0인지 확인한다.
  시간 복잡도 : O(1)
- full() 플레이 리스트가 가득 찼는지 확인 함수
  count가 10인지 확인한다.
  시간 복잡도 : O(1)
- exist(artist, title) 플레이 리스트 탐색 함수
  ① 플레이 리스트가 비어있으면 false를 반환한다.
  ② head부터 순회하여 특정 가수와 제목을 가진 노래가 있는지 확인한다.

  시간 복잡도 : O(n)

- print() 플레이 리스트 출력 함수
  ① empty() 함수를 호출하여 플레이 리스트가 비어있다면 리턴한다.
  ② head부터 순회하여 가수 / 제목 / 재생시간 형식으로 출력한다.
  ③ 마지막에는 총 노래 개수와 재생 시간도 함께 출력한다.

  시간 복잡도 : O(n)

- Getter 함수 (getRunTime(), getCount())
  각 노드가 가진 음악 정보를 반환한다.
  시간 복잡도 : O(1)

4. Result Screen
   LOAD 모든 데이터 로드

log.txt

```
========LOAD========
N.Flying/blue moon/3:36
N.Flying/moonshot/3:26
N.Flying/songbird/3:46
N.Flying/rooftop/3:19
N.Flying/oh really./3:10
PSY/gangnam style/3:25
PSY/gentleman/3:14
PSY/daddy/3:32
PSY/new face/3:16
PSY/that that/3:05
BTS/dynamite/3:19
IU/blueming/3:37
Michael Jackson/beat it/4:18
Maroon 5/sugar/3:55
Westlife/my love/3:51
Buzz/my love/4:05
Hyorin/blue moon/3:24
====================
```

ADD

ADD KDA|pop stars|3:23 새로운 데이터를 추가

ADD N.Flying|songbird|3:46 이미 Queue에 존재하는 곡을 추가해서 에러 발생

```
========ADD========
KDA/pop stars/3:23
====================
========ERROR========
200
======================
```

QPOP BST 구조 형성 성공

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

SEARCH

SEARCH ARTIST N.Flying 가수 이름으로 노래 검색

SEARCH TITLE blue moon 노래 제목을 기준으로 검색

SEARCH TITLE my love 노래 제목을 기준으로 검색

SEARCH SONG N.Flying|blue moon 하나의 노래 검색

```
29  ========SEARCH========
30  N.Flying/blue moon/3:36
31  N.Flying/moonshot/3:26
32  N.Flying/songbird/3:46
33  N.Flying/rooftop/3:19
34  N.Flying/oh really./3:10
35  ====================
36  ========SEARCH========
37  N.Flying/blue moon/3:36
38  Hyorin/blue moon/3:24
39  ====================
40  ========SEARCH========
41  Westlife/my love/3:51
42  Buzz/my love/4:05
43  ====================
44  ========SEARCH========
45  N.Flying/blue moon/3:36
46  ====================
```

MAKEPL

MAKEPL ARTIST N.Flying 해당 가수의 모든 노래를 플레이 리스트에 넣기

MAKEPL TITLE blue moon 이미 존재하는 노래를 플레이 리스트에 넣기→에러

MAKEPL TITLE my love 해당 제목을 가진 모든 노래를 플레이 리스트에 넣기

MAKEPL SONG Maroon5|sugar 한 곡을 플레이 리스트에 넣기

MAKEPL ARTIST PSY 넣으려는 노래 개수가 플레이 리스트의 여유 공간보다 많을 경우→에러 출력

```
47  ========MAKEPL========
48  N.Flying/blue moon/3:36
49  N.Flying/moonshot/3:26
50  N.Flying/songbird/3:46
51  N.Flying/rooftop/3:19
52  N.Flying/oh really./3:10
53  Count : 5 / 10
54  Time : 17min 17sec
55  ======================
56  ========ERROR========
57  500
58  ====================
59  ========MAKEPL========
60  N.Flying/blue moon/3:36
61  N.Flying/moonshot/3:26
62  N.Flying/songbird/3:46
63  N.Flying/rooftop/3:19
64  N.Flying/oh really./3:10
65  Westlife/my love/3:51
66  Buzz/my love/4:05
67  Count : 7 / 10
68  Time : 25min 13sec
69  ======================
```

```
70  ========MAKEPL========
71  N.Flying/blue moon/3:36
72  N.Flying/moonshot/3:26
73  N.Flying/songbird/3:46
74  N.Flying/rooftop/3:19
75  N.Flying/oh really./3:10
76  Westlife/my love/3:51
77  Buzz/my love/4:05
78  Maroon 5/sugar/3:55
79  Count : 8 / 10
80  Time : 29min 8sec
81  ======================
82  ========ERROR========
83  500
84  ====================
```

PRINT

PRINT ARTIST ArtistBST 전체 출력

PRINT TITLE TitleBST 전체 출력

PRINT LIST 플레이 리스트 전체 출력

```
========PRINT========
ArtistBST
BTS/dynamite/3:19
Buzz/my love/4:05
Hyorin/blue moon/3:24
IU/blueming/3:37
KDA/pop stars/3:23
Maroon 5/sugar/3:55
Michael Jackson/beat it/4:18
N.Flying/blue moon/3:36
N.Flying/moonshot/3:26
N.Flying/songbird/3:46
N.Flying/rooftop/3:19
N.Flying/oh really./3:10
PSY/gangnam style/3:25
PSY/gentleman/3:14
PSY/daddy/3:32
PSY/new face/3:16
PSY/that that/3:05
Westlife/my love/3:51
====================
```

```
========PRINT========
TitleBST
Michael Jackson/beat it/4:18
N.Flying/blue moon/3:36
Hyorin/blue moon/3:24
IU/blueming/3:37
PSY/daddy/3:32
BTS/dynamite/3:19
PSY/gangnam style/3:25
PSY/gentleman/3:14
N.Flying/moonshot/3:26
Westlife/my love/3:51
Buzz/my love/4:05
PSY/new face/3:16
N.Flying/oh really./3:10
KDA/pop stars/3:23
N.Flying/rooftop/3:19
N.Flying/songbird/3:46
Maroon 5/sugar/3:55
PSY/that that/3:05
====================
```

```
========PRINT========
N.Flying/blue moon/3:36
N.Flying/moonshot/3:26
N.Flying/songbird/3:46
N.Flying/rooftop/3:19
N.Flying/oh really./3:10
Westlife/my love/3:51
Buzz/my love/4:05
Maroon 5/sugar/3:55
Count : 8 / 10
Time : 29min 8sec
====================
```

DELETE

DELETE ARTIST PSY 해당 가수의 노래 전부 삭제 (BST, 플레이 리스트)

DELETE ARTIST PSY 존재하지 않는 데이터를 삭제하고자 함→에러 출력

DELETE TITLE blue moon 해당 제목을 가지고 있는 노래 전부 삭제

DELETE TITLE blue moon 존재하지 않는 데이터를 삭제하고자 함→에러 출력

DELETE LIST N.Flying|songbird 플레이 리스트에서 해당 노래 삭제

DELETE LIST N.Flying|songbird 존재하지 않는 데이터를 삭제하고자 함→에러

DELETE SONG N.Flying|rooftop 해당 노래를 전부 삭제 (BST, 플레이 리스트)
DELETE SONG N.Flying|rooftop 존재하지 않는 데이터를 삭제하고자 함→에러

```
========DELETE========
Success
====================
========ERROR========
700
====================
========DELETE========
Success
====================
========ERROR========
700
====================
========DELETE========
Success
====================
========ERROR========
700
====================
========DELETE========
Success
====================
========ERROR========
700
====================
```

PRINT ARTIST했을 때, 지우고자 했던 PSY의 노래들, blue moon 노래들, N.Flying의 rooftop 노래가 삭제된 것을 확인할 수 있다.

```
========PRINT========
ArtistBST
BTS/dynamite/3:19
Buzz/my love/4:05
IU/blueming/3:37
KDA/pop stars/3:23
Maroon 5/sugar/3:55
Michael Jackson/beat it/4:18
N.Flying/moonshot/3:26
N.Flying/songbird/3:46
N.Flying/oh really./3:10
Westlife/my love/3:51
====================
========DELETE========
Success
====================
```

DELETE SONG N.Flying|songbird 해당 노래를 전부 삭제 (BST, 플레이 리스트)
다음 결과 화면에서 ArtistBST를 출력하여 N.Flying의 songbird 노래가 위의 ArtistBST 출력에는 존재했지만, 아래의 ArtistBST에는 존재하지 않는 것을 확인할 수 있다.

여기에서 DELETE 이후의 전체 결과를 확인할 수 있다.

```
179  ========PRINT========
180  ArtistBST
181  BTS/dynamite/3:19
182  Buzz/my love/4:05
183  IU/blueming/3:37
184  KDA/pop stars/3:23
185  Maroon 5/sugar/3:55
186  Michael Jackson/beat it/4:18
187  N.Flying/moonshot/3:26
188  N.Flying/oh really./3:10
189  Westlife/my love/3:51
190  ====================
191  ========PRINT========
192  TitleBST
193  Michael Jackson/beat it/4:18
194  IU/blueming/3:37
195  BTS/dynamite/3:19
196  N.Flying/moonshot/3:26
197  Westlife/my love/3:51
198  Buzz/my love/4:05
199  N.Flying/oh really./3:10
200  KDA/pop stars/3:23
201  Maroon 5/sugar/3:55
202  ====================
203  ========PRINT========
204  N.Flying/moonshot/3:26
205  N.Flying/oh really./3:10
206  Westlife/my love/3:51
207  Buzz/my love/4:05
208  Maroon 5/sugar/3:55
209  Count : 5 / 10
210  Time : 18min 27sec
211  ====================
```

EXIT 모든 메모리 할당 해제하며 프로그램 종료

```
212  ========EXIT========
213  Success
214  ====================
```

5. Consideration

이번 프로젝트는 큐, 이진 탐색 트리, 원형 이중 연결 리스트 등의 여러가지 자료구조들을 활용해서 음악 데이터를 관리하는 시스템을 구현했다. 사용자는 명령어 파일을 통해서 음악을 로드, 추가, 검색, 플리 생성, 삭제 등을 수행할 수 있다. 모든 결과는 log.txt 라는 파일에 기록이 된다.

<구현 과정 어려움 & 해결 방법>

- 메모리 관리 복잡성

  QPOP 명령을 구현할 때 큐에서 pop한 노드를 ArtistBST와 TitleBST에 삽입한 후, 원래 노드의 메모리를 해제해야 했다. 그런데 BST에 삽입할 때 새로운 노드를 생성해야해서 큐에 있는 노드를 그대로 사용할 수 없이 데이터를 복사만 해야했다.

  →MusicQueueNode에서 artist, title, run_time 정보를 getter 함수로 꺼내서

이 저보들을 BST 노드 생성자의 매개변수로 전달하고자 했다. 큐에서 pop한 노드는 데이터를 전달 하고 바로 delete로 메모리를 해제해서 메모리 누수를 효과적으로 막을 수 있게 했다. 그리고 소멸자에서 재귀적으로 자식 노드를 삭제하는 deleteTree 함수를 만들어서 프로그램을 종료할 때, 모든 메모리가 제대로 정리되도록 구현했다.

- BST가 두개여서 동기화에 문제
  하나의 곡 데이터를 ArtistBST와 TitleBST 양쪽에 각각 저장을 해야 했다. 그런데 두 트리의 구조가 다르기 때문에 삽입을 하거나 삭제를 할 때 동기화에 좀 어려움을 겪었다. 그중에서도 DELETE SONG 명령에서 한쪽 트리만 삭제되거나, 가수나 제목 노드에 여러 곡이 있을 때 특정 노래만 삭제해야 하는 경우, 코드가 너무 길어지고 복잡해졌다.
  →각 BST 노드에 vector라는 것을 사용해서 하나의 가수 혹은 제목에 여러 노래를 저장할 수 있도록 구현했다. deleteTitle과 deleteArtist 함수를 구현해서 vector에서 해당 노래만 제거하고, count가 0이 되면 노드 자체를 삭제하여 삭제 로직을 2단계로 구성했다. DELETE 명령에서는 두 개의 BST와 플리에서 모두 삭제를 수행하도록 해야했다.
- 원형 이중 연결 리스트의 무한루프
  플리를 원형 이중 연결 리스트로 구현하면서 순회의 종료 조건을 잘못 설정해서 무한 루프에 빠지는 문제가 발생했다. 노드를 삭제할 때, head와 tail을 제대로 처리하지 않아서 플리의 구조가 이상해질 때가 있었다.
  →순회할 때 do-while문을 사용해서 current != head 조건으로 한바퀴를 돌게 했다. 하나의 노드일 경우 current == head && current->next == head로 해서 head를 nullptr로 저장했고, head 노드를 삭제하는 경우엔 head = head->next로 head를 이동시킨 후에 연결을 끊도록 순서를 확실히 했다. 삭제 전에는 count와 time을 먼저 업데이트 해서 삭제 후에는 노드 정보에 접근하지 않도록 확실하게 했다.
- 문자열 파싱
  명령어와 데이터가 되게 다양하고 복잡한 형태로 파일에 저장되어 있었다. | 와 DELETE SONG등으로 각각 다른 구분자, 형식을 사용했다. 그리고 런타임 형식을 초 단위로 변환해야하는 것처럼 파싱 로직이 좀 복잡했던 것 같다.
  →substr와 find 함수를 조합해서 파싱 함수를 구현했다. 공백을 기준으로 옵션이라는 변수와 데이터를 분리했고, |를 기준으로 artist, title, runtime을 추출하였다. convertToSeconds 라는 보조 함수를 구현해서 분:초 형식을 초로 변환할 수 있었다. 파싱할 때 잘못된 형식이나 비어있다면 에러를 로그에 찍도록 해서 제대로 확인할 수 있었다.