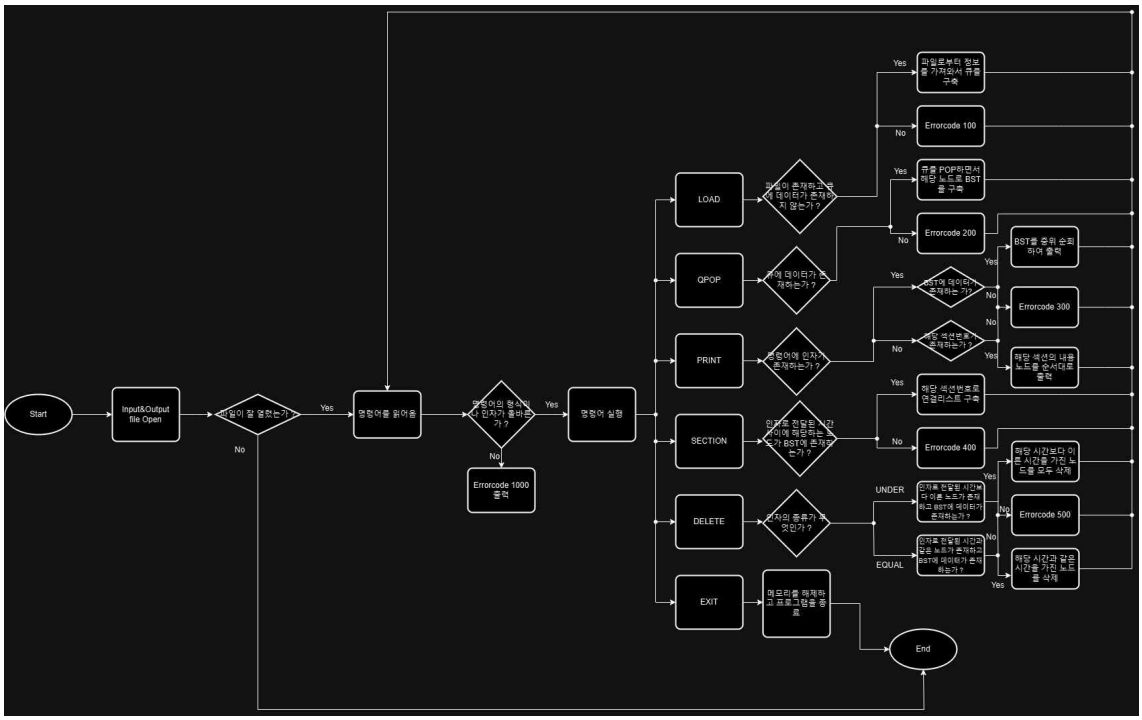
데이터 구조 설계 1차 프로젝트 보고서

2021202085 전한아솔

1. Introduction-프로젝트 소개

이번 프로젝트는 큐, 이진 탐색 트리, 연결 리스트의 자료 구조를 이용하여 자막을 관리하는 프로그램을 구현하는 프로젝트이다. LOAD 명령어를 실행하면 파일로부터 정보를 읽어서 QUeue를 구축하고, QPOP 명령어를 실행하면 Queue에서 pop 연산을 진행하여 해당 데이터를 바탕으로 BST를 구축한다. SECTION 명령어를 실행하면 같이 인자로 전달되는 특정 시간 구간에 해당하는 노드를 탐색하고 이를 바탕으로 연결 리스트를 구축한다. 이 외에도 BST나 List를 출력하는 PRINT, 인자로 시간을 입력받아서 해당 시간에 맞는 노드를 BST에서 삭제하거나 해당 시간보다 이른 시간을 가지는 노드를 BST에서 모두 삭제하는 DELETE 명령어도 존재한다. EXIT라는 메모리를 해제하고 프로그램을 종료하는 명령어까지 총 6개의 명령어에 동작하는 프로그램을 3가지 자료 구조를 이용하여 만드는 프로젝트이다. 입력과 출력은 파일 입·출력으로 진행하며, 각 명령어에 맞는 동작을 시행하면 된다. 명령어에 인자가 모자라거나 넘치는 경우 에러를 출력하며, 모든 예외 상황에 맞는 에러 코드를 출력하도록 구현한다.

2. Flowchart-프로젝트의 전반적인 흐름과 동작을 설명



먼저 프로그램을 실행하면 Input으로 사용할 명령어가 적힌 파일과 Output으로 사용할

파일을 연다. 이때, 파일이 제대로 열리지 않으면 프로그램을 종료하도록 구성되어 있고, 제대로 열렸다면 Input 파일로부터 명령어를 읽어와서 차례대로 명령어를 실행한다. 만약 해당 명령어가 소문자로 적혀있거나 인자의 수가 제대로 맞지 않거나 명령어의 형식이 틀린 경우에는 에러 코드를 출력하고 다음 명령어를 읽는다.

LOAD)
먼저 LOAD 명령어는 자막 데이터 파일로부터 자막 시간과 내용을 가져와 해당 정보로 Queue를 구축한다. 성공적으로 데이터를 불러오면 “시간 - 내용”의 형태로 데이터를 출력하고 데이터 파일이 존재하지 않거나 이미 큐가 구축되어 있다면 에러 코드를 출력하는 방식으로 구성된다.

QPOP)
QPOP 명령어는 자막 데이터로 구축한 Queue에서 데이터를 Pop하여 Binary Search Tree(BST)를 구축한다. BST의 노드도 Queue와 같이 자막 시간과 내용을 정보로 가진다. Queue에 데이터가 존재하지 않을 때는 에러 코드를 출력하는 방식으로 구성된다.

PRINT)
PRINT 명령어는 BST나 연결 리스트에 저장된 데이터를 출력한다. 명령어에 인자가 입력되지 않는다면 중위 순회의 방식으로 BST를 출력하며 인자로 특정 숫자가 입력된다면 해당 섹션 번호를 갖는 리스트의 내용 노드를 순차적으로 출력한다. 이때 존재하지 않는 섹션 번호가 인자로 입력되면 에러 코드를 출력하는 방식으로 구성된다.

SECTION)
SECTION 명령어는 섹션 번호, 탐색 시작 시각, 탐색 종료 시각을 받아서 인자로 전달받아서 BST를 탐색하여 해당 시간에 포함되는 시간을 가진 노드를 하나의 섹션 번호로 연결 리스트로 구축한다. 리스트는 헤더 노드와 내용 노드로 구성하는데, 헤더 노드는 섹션 번호와 다음 헤더 노드의 주소를 가지고 있고 뒤에 내용 노드들이 자막 시간과 내용의 정보를 가지고 연결되는 방식으로 구성된다. 따라서 2차원의 형태를 가진다. 이때 BST에 탐색 된 노드가 없다면 에러 코드를 출력하는 방식으로 구현한다.

DELETE)
DELETE 명령어는 인자로 EQUAL이나 UNDER을 전달받는다. 먼저, EQUAL을 전달받은 경우, 뒤에 같이 전달받은 자막 시간과 같은 자막 시간을 가지는 노드를 BST에서 제거한다. UNDER은 전달받은 자막 시간보다 이른 자막 시간을 가지는 노드를 BST에서 모두 제거한다. 두 경우 모두 삭제할 노드가 존재하지 않거나 BST에 데이터가 존재하지 않는다면 에러 코드를 출력하는 방식으로 구현한다.

EXIT)
EXIT 명령어는 프로그램의 메모리를 해제하고 프로그램을 종료하는 명령어이다.

파일로부터 명령어를 읽고 수행한 결과가 에러 코드를 출력한 결과이거나 해당 명령어의

제대로 된 동작을 수행한 결과 모두, 한 번의 명령어가 끝나면 파일에서 다음 명령어를 읽어와서 수행하는 방식으로 구성된다. 따라서 프로그램이 종료되는 상황이 EXIT의 명령어를 입력하여 메모리를 해제하고 프로그램을 종료하는 경우 말고도 2가지 상황이 더 발생할 수 있다. 제안서를 보면 Queue가 비어있을 때 POP 연산을 진행하는 상황에서 프로그램을 종료한다고 작성되어 있다. 따라서 LOAD 명령어 실행 후 QPOP까지 실행하여 Queue가 비워진 상황에서 QPOP 명령어를 통해서 Queue를 Pop 하려고 하면 큐가 비어있다는 메시지를 출력하면서 프로그램을 종료되는 상황이 발생한다.

다음으로는 Queue가 전부 차 있을 때 Push 연산을 진행하는 상황이다. Queue는 100의 크기로 초기화되고 이후 크기가 변하지 않으므로 Input file에서 100개가 넘는 정보를 받아와서 Queue를 구축하는 경우, Queue가 전부 차 있는 상황에서 Push 연산을 수행하므로 큐가 전부 차 있다는 메시지를 출력하면서 프로그램을 종료하는 상황이 발생한다.

Flowchart에는 프로그램을 비정상적으로 종료하는 경우가 file이 제대로 열리지 않은 경우만 작성되어 있지만 앞서 설명한 2가지 경우도 프로그램이 비정상적으로 종료되므로 보고서에 글로 작성해서 설명했다.

3. Algorithm-프로젝트에서 사용한 알고리즘의 세부적인 동작과 아이디어를 설명

알고리즘 부분을 설명할 때 클래스별로 나누어서 설명할지, 파일별로 나누어서 설명할지 고민했다. 클래스별로 나누어서 설명하게 되면 각 클래스의 동작을 명확히 나누어서 설명할 수 있기에, 파일별로 나누어서 설명하는 것보다 명확한 설명이 가능하다고 판단하여 클래스별로 나누어서 설명했다. 또한 클래스에 존재하는 함수에 대한 설명도 진행하는데, 모든 함수를 하나하나 설명하기보단 해당 클래스에 메인이 되는 함수를 여러 개 설명하고 그 함수에서 사용한 여러 알고리즘, 구현 방법, 다른 함수 등등을 설명하는 방식으로 보고서의 알고리즘 부분을 구성했다.

<Manager>

프로그램을 실행하면 수행되는 main 함수는 manager 객체를 선언하고 해당 객체의 함수(Run)를 실행하는 코드로 단 2줄이면 끝나게 된다. 따라서 Manager 클래스는 이번 프로젝트에서 사용되는 3가지 자료 구조를 명령어에 맞게 어떤 방식으로 작동시킬 것인가를 나타내는 일종의 컨트롤러 같은 역할을 한다. 따라서 이 Manager 클래스의 멤버 변수로 다른 객체를 선언하여 자료 구조들을 전체적으로 조정한다. 또한 ifstream과 ofstream 객체를 멤버 변수로 가지고 있어서 처음 파일을 열고 해당 파일에 작성해야 하는 부분이 있다면 ofstream 객체를 참조로 전달하여 파일에 작성할 수 있도록 구현했다.

-void Manager::Run(const char* command)

Run 함수는 인자로 전달된 command(.txt)를 input으로 열고 파일에 적혀있는 명령어에 맞는 동작을 수행하는 다른 함수를 호출하는 함수이다.

먼저 command를 ifstream 객체로 열고 log.txt라는 파일을 ofstream 객체로 연다. 각 파일은 input, output의 용도로 사용된다. 만약 command에 전달된 파일을 제대로 열지 못한다면 오류 문구와 함께 -1을 반환하며 프로그램을 종료한다. 제대로 파일을 열었다면 string 변수(temp)를 선언하여 getline을 통해서 파일로부터 한 줄을 읽어와 string 변수에 저장한다. 그리고 if문을 통하여 해당 명령에 맞는 동작을 수행한다.

과제 제안서에 “명령어에 인자가 모자라거나 필요 이상으로 입력받을 경우”와 “예외 처리”에 대해서는 반드시 에러 코드를 출력해야 한다고 적혀있다. 따라서 명령어가 제대로 입력되지 않은 경우에 대한 모든 예외 처리를 해당 함수에서 수행했다.
LOAD와 QPOP은 인자가 따로 존재하지 않으므로 temp==LOAD or temp==QPOP인 경우에 명령어를 실행하도록 했다.

PRINT의 경우 temp[0]이 P일 때 temp가 PRINT라면 인자가 없으므로 bst를 출력하도록 했고, 아니라면 리스트를 출력하거나 에러 중 하나를 출력한다. 0~4의 인덱스를 저장하여 해당 문자열이 PRINT이고, 5~끝의 인덱스를 저장하여 해당 문자열이 숫자라면 제대로 된 인자를 전달받은 PRINT 명령어이므로 해당 섹션 번호를 가지는 리스트를 출력하고 이 외에 경우는 PRINT의 에러 코드인 300을 출력한다. 여기서 사용된 문자열이 숫자인지 판단하는 함수는 bool isNum(string num)의 형태로, 인자로 전달된 문자열을 순회하면서 숫자인지 확인하는데, 아스키 값으로 48 이상, 57 이하의 값이 숫자이다. 따라서 한 문자라도 아스키 값이 48 미만 or 57 초과라면 false를 반환하고 아니라면 true를 반환한다.

인자로 전달되는 값이 숫자라는 조건이 없어서 숫자가 아닌 예외에 대한 처리를 추가했다.

DELETE의 경우 PRINT와 비슷하게 temp[0]이 D이면 0~5까지의 문자열을 파싱하여 해당 문자열이 DELETE인가를 확인한다. DELETE가 맞다면 7~11까지의 문자열을 파싱하여 해당 문자열이 EQUAL인가, UNDER를 확인한다. 둘 다 5글자이므로 7~11의 문자열만 추출하도록 했다. if문으로 각 인자일 때의 조건으로 들어가면 13부터 문자열의 끝을 추출한다. 이 부분은 인자로 전달되는 자막 시간인데, 자막 시간의 형식이 맞는가를 판단하는 isCont_time이라는 함수를 호출하여 자막 시간의 형식이 제대로 되었는가를 판단한다.

isNum와 비슷하게 bool isCont_time(string num)으로, 자막 시간의 형식은 xx:xx:xx이므로 길이가 8이다. 따라서 길이가 8이 아니면 false를 반환한다. 길이가 8이라면 2, 5 인덱스는 ‘:’이고 나머지 인덱스의 아스키 값을 비교하여 숫자인가를 판단하여 하나라도 어긋난다면 false, 모두 정확하다면 true를 반환하는 함수를 구현하여 사용했다. isCont_time함수를 통해서 제대로 된 자막 시간임을 확인했으면 명령어에 맞는 함수를 호출하여 동작시키고 아니라면 DELETE의 에러 코드인 500을 출력한다.

SECTION도 마찬가지로 temp[0]이 S라면 0~6의 문자열을 추출하여 해당 문자열이 SECTION이라면 8~빈칸까지 문자열을 추출하여 제대로 된 자막 시간인지 확인하고 인덱스를 하나 증가하여 빈칸은 건너뛰고 해당 인덱스부터 문자열의 끝까지 추출하여 2개의 인자 모두 제대로 된 자막 시간인지 확인한다. 맞다면 2개의 인자로 SECTION 명령어를 수행하는

함수를 호출하고 아니라면 SECTION의 에러 코드인 400을 출력한다.

문자열을 추출하는 과정에서 해당 문자가 공백이 아니고 해당 인덱스가 문자열의 길이보다 작다 라는 조건을 && 연산자를 사용해서 if문의 조건으로 사용하게 되면 해당 문자가 공백이 아님을 체크할 때 인덱스의 범위를 넘을 수 있으므로 문자열의 길이보다 작고 해당 문자가 공백이 아니라는 조건으로 순서를 바꾸어서 구현했다.

EXIT는 입력하면 return을 통하여 프로그램을 종료하면 객체의 소멸자를 자동으로 호출하여 메모리를 해제하고 프로그램을 종료하므로 다른 코드나 예외 처리는 작성하지 않았다.

이 외에 명령어들은 모두 잘못된 명령어를 의미하는 에러 코드 1000을 출력한다.

-void Manager::load()
LOAD 명령어가 입력되면 수행하는 함수로 텍스트 파일로부터 데이터 정보를 받아와서 Queue에 모두 저장하는 명령어이다. 자막 시간과 내용이 적혀있는 “subtitle.txt”파일을 istream 객체로 연다. 텍스트 파일이 존재하지 않거나 자료 구조에 이미 데이터가 들어가 있으면 에러 코드를 출력하라고 되어있으므로 파일이 제대로 열렸고, Queue가 비어있으며 BST의 root가 nullptr이고 list의 head가 nullptr이라면 파일로부터 getline을 통해서 한 줄씩 string에 저장하여 0~7은 자막 시간 부분으로 추출하고 0~8은 내용 부분으로 추출하여 2개의 string 변수로 저장하여 Queue에 push 함수의 인자로 전달한다. Queue의 push 함수는 SubtitleQueue 클래스에서 설명하겠다. 하나라도 부합하지 않는 조건이 있다면 LOAD의 에러 코드인 100을 출력한다.

-void Manager::qpop()
QPOP 명령어는 Queue에서 데이터를 Pop 연산하여 BST를 구축하는 명령어이다. 제안서의 Queue 자료 구조 부분을 보면 Queue가 비어있는 상황에서 Pop 연산을 진행하면 프로그램이 종료하도록 해야 한다. 이는 아무것도 없는 상황에서 QPOP을 하거나 이미 QPOP을 한 상황에서 QPOP 명령어를 수행하는 경우이다. 따라서 Queue가 비어있다면 QPOP의 에러 코드인 200을 출력하고 큐가 비어있다는 메시지와 함께 -1을 반환하며 프로그램을 종료한다. 비어있지 않다면 큐가 빌 때까지 while을 통하여 BST를 구축한다. Queue의 Pop 함수를 설명하는 부분에서 추가 적으로 설명하겠지만, Pop 함수 내에서 메모리를 해제하면 해당 노드의 데이터를 가져올 수 없다고 판단하여 반환형을 SubtitleQueueNode*로 구현했고 해당 노드의 주소를 반환하여 Manager 클래스의 qpop 함수에서 전달받은 노드의 주소로 데이터를 가져오고 해당 노드는 삭제하는 방식으로 Queue에 Pop 연산을 구현했다.

-void Manager::print(string num)
PRINT 명령어는 인자가 존재하지 않으면 BST를 출력하고 인자가 존재하면 해당 인자에 맞는 리스트를 출력하는 명령어이다. 따라서 인자로 전달되는 num이 빈 문자열이라면 BST가 비어있는지 확인 후, 비어있다면 에러 코드를 출력하고 아니라면 BST를 중위 순회로 출력하는 BST 클래스의 함수를 호출한다. 이는 BST 클래스를 설명하면서 추가로 설명하겠다.

num이 빈 문자열이 아니라면 리스트 클래스의 출력 함수를 호출한다. 이 함수는 인자로 섹션 번호와 bool형 변수를 전달받아서 섹션 번호를 가지는 헤더 노드가 존재하면 해당 노드의 내용 노드를 출력하고, 해당 섹션 번호를 가지는 노드가 존재하지 않는다면 인자로 같이 전달된 bool형 변수(참조로 전달)를 false로 바꾸어서 Manager 클래스의 print 함수에서 에러 코드를 출력하도록 동작한다. 리스트 클래스의 print 함수는 해당 클래스를 설명하면서 추가로 설명하겠다.

-void Manager::section(int num, string first_time, string second_time)
SECTION 명령어는 인자로 숫자와 두 개의 자막 시간을 전달받아서 해당 시간 사이에 존재하는 노드를 BST에서 탐색하여 데이터를 가져와서 인자로 전달받은 숫자를 섹션 번호로 가지는 리스트를 구축하는 명령어이다. 이 함수에서는 BST 객체의 search 함수로 숫자와 두 개의 자막 시간을 모두 전달하는 역할만 한다. print 함수와 비슷하게 bool형 flag를 추가로 전달하여 해당 자막 시간 사이에 탐색 된 노드가 없다면 에러 코드를 이 함수에서 출력하도록 구현했다. BST 객체의 search 함수만 호출하는 이유는 해당 함수를 재귀로 구현하여 중위 순회 방식으로 노드를 탐색하고 리스트를 구축하도록 하기 위함이다. 이에 대한 추가 설명은 BST 클래스에서 더 설명하겠다.

-void Manager::delete_equal(string time)
DELETE 명령어는 인자로 UNDER과 EQUAL 중 하나를 골라서 자막 시간과 같은 노드를 삭제할지 자막 시간보다 이른 시간을 가지는 노드를 삭제할지 선택한다. 따라서 DELETE의 동작은 2가지이므로 함수를 나누어서 구현했고 BST 클래스 또한 2가지의 삭제를 동작할 수 있도록 구현했다. 이 함수는 인자로 전달받은 자막 시간과 같은 시간을 가지는 노드를 BST에서 삭제하는 함수이다.

인자로 선언한 bool형 flag와 자막 시간을 BST 객체의 delete_equal 함수로 전달한다. 해당 함수는 인자로 전달된 시간과 동일한 노드를 삭제하고 동일한 노드가 존재하지 않으면 flag를 true로 바꾼다. 따라서 Manager 클래스의 delete_equal 함수 내에서 에러 코드를 출력하도록 구현했다.

-void Manager::delete_under(string time)
인자로 전달된 시간보다 이른 시간을 가지는 노드를 BST에서 삭제하는 동작을 수행한다. BST 클래스에서 후위 순회를 통하여 노드를 삭제하고 다시 탐색하는 방식으로 구현하려 했으나 삭제 과정에서 노드가 바뀌며 노드를 건너뛰고 탐색하는 상황이 발생하여 LIFO의 구조를 가지는 Stack을 활용하여 중위 순회를 통해서 조건에 맞는 노드를 스택에 집어넣으면 리프 노드가 스택의 윗부분에 위치할 것이므로 스택이 빌 때까지 반복하여 스택에 들어있는 노드의 정보를 BST 클래스의 delete_equal 함수로 전달하여 해당 노드를 삭제하고 스택의 원소를 Pop하는 방식으로 구현했다. 따라서 해당 노드의 직접적인 삭제는 BST 클래스의 delete_equal 함수에서 진행하고 Manager 클래스의 delete_under 함수는 노드를 탐색하여 전달하는 역할만 한다.

<SubtitleQueueNode>

SubtitleQueue의 노드를 의미하는 SubtitleQueueNode 클래스이다. 멤버 변수로는 자막 시간과 자막 내용을 의미하는 string 변수와 다음 노드를 가리키는 SubtitleQueueNode*형 변수가 존재한다. 생성자는 string 변수 2개를 인자로 받으며 생성자가 호출되면 해당 인자로 자막 시간과 내용을 초기화하고 다음 노드는 nullptr로 초기화한다. 메모리를 동적으로 할당한 내용이 없으므로 소멸자는 따로 작성하지 않았다. 멤버가 private으로 선언되어 있으므로 멤버의 정보를 가져오고 멤버에 저장하기 위해서 set, get 함수를 정의해주었다.

<SubtitleQueue>

프로젝트에서 구현해야 하는 3가지 자료 구조 중, Queue를 의미하는 SubtitleQueue 클래스이다. SubtitleQueueNode 클래스의 객체를 활용하여 연결 리스트로 큐의 자료 구조를 구현했다. 해당 클래스의 멤버 변수로는 큐의 맨 앞을 의미하는 front, rear가 있고 이는 SubtitleQueueNode* 형이다. 또한 큐의 사이즈를 의미하는 int형 변수도 포함한다. 생성자에서는 front와 rear를 nullptr로 초기화하며, size는 0으로 초기화한다.

소멸자에서는 SubtitleQueueNode* 객체를 선언하여 해당 객체의 front의 다음 노드를 저장하고 front의 메모리를 해제, front는 아까 저장한 노드(front의 다음 노드)로 이동하는 방식으로 front가 nullptr이 될 때까지 반복하며 메모리를 해제하도록 했다.

멤버 함수로는 큐가 비어있는지를 판단하는 isEmpty, 큐가 꽉차있는지를 판단하는 isFull, 큐에 원소를 삽입하는 push, 큐의 원소를 삭제하는 pop, 큐의 맨 앞 원소를 반환하는 front, 큐를 출력하는 print로 구성된다.

-bool SubtitleQueue::isEmpty()

큐의 front가 nullptr이라면 큐가 비어있으므로 true를 반환한다.

-bool SubtitleQueue::isFull()

제안서를 보면 큐의 크기는 100으로 초기화되고 변하지 않으므로 size가 100이 된다면 큐가 꽉 찼음을 의미한다. 따라서 size가 100이라면 true를 반환한다.

void SubtitleQueue::Push(string time, string content, ofstream& flog)

큐에 원소를 추가하는 Push 연산을 수행하는 함수이다. 큐는 FIFO의 구조를 가지므로 먼저 삽입된 원소가 밀린다. 제안서의 Queue 부분에 큐가 가득 차 있는 경우에 Push를 하게 되면 프로그램을 종료하도록 구현하라고 되어있다. 따라서 isFull 함수를 이용하여 큐가 가득 찼다면, 큐가 가득 찼다는 메시지를 출력하고 -1을 반환하며 프로그램을 종료한다. 아니라면 인자로 전달받은 자막 시간과 내용으로 new를 통해서 새로운 노드를 생성한다. 큐가 비어있다면 해당 노드는 front이자 rear가 될 것이고 아니라면 rear의 다음 노드로 새로운 노드를 저장하고 rear를 새로운 노드로 옮겨서 리스트의 맨 뒤에 새로운 노드를 추가하는 방식으로 큐의 Push 연산을 구현했다. Push가 일어나면 큐의 크기가 하나 증가한다.

SubtitleQueueNode* SubtitleQueue::Pop(ofstream& flog)

큐에서 원소를 삭제하는 Pop 연산을 수행하는 함수이다. 제안서에서 큐가 비어있는 경우에 Pop을 하게 되면 프로그램을 종료하도록 구현하라고 했으므로 isEmpty 함수를 이용하여 큐

가 비어있다면, 큐가 비어있다는 메시지를 출력하고 -1을 반환하며 프로그램을 종료한다. FIFO의 구조를 가지므로 먼저 삽입된 원소가 먼저 삭제된다. 따라서 리스트의 맨 앞 노드를 삭제하는 방식으로 큐의 Pop 연산을 구현했다. 큐가 비어있지 않다면 front 노드를 저장하고 front를 다음 노드로 옮긴 후, front 노드를 반환한다. 이렇게 반환된 노드는 Manager 클래스의 QPOP 함수에서 해당 노드의 데이터(자막 시간과 자막 내용)를 저장한 후 메모리가 해제되도록 구현하기 위해서 함수의 반환형을 SubtitleQeueuNode*로 택했다.

SubtitleQueueNode* SubtitleQueue::Front()
큐의 맨 앞 노드를 반환하는 함수로 front를 반환한다.

SubtitleQueueNode* SubtitleQueue::print(ofstream& flog)
큐에 저장된 원소를 모두 출력하는 함수로 front부터 반복하여 nullptr이 아닐 때까지 자막시간 - 자막내용의 형식으로 한 줄씩 출력하고 다음 노드로 이동한다.

<SubtitleBSTNode>
Queue에서 Pop한 노드를 가지고 구축한 BST의 노드를 의미하는 SubtitleBSTNode 클래스이다. Queue와 같은 데이터를 가져야 하므로 멤버 변수도 똑같이 자막 시간과 자막 내용을 의미하는 string형 변수가 존재하고, binary search tree의 구조이므로 왼쪽, 오른쪽 자식을 의미하는 SubtitleBSTNode*형 변수도 선언했다.

생성자에서는 이니셜라이저를 이용하여 왼쪽, 오른쪽 자식을 nullptr로 초기화하고 자막 시간과 내용을 빈문자열로 초기화한다. 메모리를 동적으로 할당한 내용은 없으므로 소멸자는 따로 코드를 작성하지 않았다. SubtitleQueueNode와 마찬가지로 멤버가 private으로 선언되어 있다. 따라서 멤버에 접근하기 위해서 get, set 함수를 정의했다.

<SubtitleBST>
프로젝트의 자료 구조 중 BST를 의미하는 클래스이다. 멤버 변수로는 Root 노드를 의미하는 SubtitleBSTNode*형 변수만 존재한다. 해당 멤버도 private으로 선언되어 있으므로 루트 노드를 반환하는 getRoot, 노드를 삽입하는 insert, 중위 순회 방식으로 트리를 출력하는 print, 인자로 전달된 2개의 자막 시간 사이에 해당하는 노드를 탐색하고 해당 노드로 리스트를 구축하기 위한 search, 인자로 전달된 자막 시간과 동일한 노드를 삭제하는 delete_equal, delete_equal을 이용하여 인자로 전달된 시간보다 이른 시간을 가지는 노드를 모두 삭제하는 delete_under 함수가 존재한다. 후위 순회 방식으로 트리를 순회하며 메모리를 삭제하기 위한 delete_bst 함수도 존재한다. 생성자는 루트 노드를 nullptr로 초기화한다. 메모리를 해제하기 위해서 트리를 후위 순회를 통해서 리프 노드부터 메모리를 해제해야 하므로 재귀함수를 이용해야 한다. 따라서 소멸자는 delete_bst의 인자로 root를 전달하는 역할뿐이다.

void SubtitleBST::delete_bst(SubtitleBSTNode* node)
재귀를 통해서 메모리를 해제하는 함수로, 인자가 nullptr이라면 함수를 종료하고 아니라면 왼쪽 자식을 전달하여 자기 자신을 호출, 오른쪽 자식을 전달하여 자기 자신을 호출, 현재 노드를 delete하는 코드로 구성된다. 이렇게 되면 트리에서 리프 노드부터 메모리를 해제할 수

있어서 에러가 발생하지 않는다.

void SubtitleBST::insert(string time, string cont)
BST에 새로운 노드를 삽입하는 함수이다. 먼저, 인자로 전달받은 데이터로 새로운 SubtitleBSTNODE 객체를 초기화한다. 루트 노드가 비어있다면 해당 객체가 루트 노드가 되고 왼쪽, 오른쪽 자식은 모두 nullptr이다. 루트 노드가 비어있지 않다면 BST가 되기 위해서 자막 시간이 이르면 왼쪽, 느리면 오른쪽 노드로 이동해야 한다. 루트 노드부터 반복하며 새로운 노드의 자막 시간이 현재 노드의 자막 시간보다 이르다면 왼쪽 자식 노드로 이동, 느리다면 오른쪽 자식 노드로 이동하여 비교를 반복한다. 이렇게 되면 리프 노드에 도달하고 마지막으로 리프 노드와 자막 시간을 비교하여 해당 리프 노드의 왼쪽 또는 오른쪽 자식으로 새로운 노드를 저장한다.

void SuvtitleBST::print(SubtitleBSTNode* node, ofstream& flog)
BST를 중위 순회하여 출력하는 print 함수이다. 아까 메모리를 해제할 때와 비슷하게 재귀를 이용하여 인자로 전달된 노드가 nullptr이라면 함수를 종료하고 아니라면 왼쪽 자식 노드를 전달하여 자기 자신을 호출, 현재 노드의 정보를 자막 시간-자막 내용의 형식으로 출력, 오른쪽 자식 노드를 전달하여 자기 자신을 호출하는 방식으로 중위 순회를 구현했다. 이렇게 출력하면 BST의 노드는 자막 시간이 오름차순으로 정렬되어 출력된다.

void SubtitleBST::search(SubtitleBSTNode* node, int num, string first_time, string escond_time, bool& flag)
BST에서 탐색은 재귀의 방식을 이용해서 반환형이 void인데, search 함수를 다른 함수에 이용하여 노드를 탐색하고 해당 노드에서 데이터를 가져오도록 구현하기 어렵다고 판단하여 search 함수는 인자로 2개의 자막 시간을 받아서 중위 순회를 통해 해당 노드가 두 자막 시간에 포함된다면 list 클래스의 insert_head 함수로 숫자를 전달하여 섹션 번호를 가지는 헤더 노드를 생성하고 insert_content 함수로 해당 노드의 자막 시간과 내용을 전달하여 내용 노드를 생성하는 방식으로 구현했다. 중위 순회를 해야하므로 node가 nullptr이라면 함수를 종료, 아니라면 왼쪽 자식 노드를 전달하여 자기 자신을 호출, 2개의 자막 시간 사이에 현재 노드가 포함되는지 판단하고 맞다면 리스트를 구축, 오른쪽 자식 노드를 전달하여 자기 자신을 호출하는 순서로 코드를 작성했다.

void SubtitleBST::delete_equal(string time, bool& flag)
인자로 전달된 자막 시간과 일치하는 노드를 삭제하는 함수이다. 현재 노드를 의미하는 p, p의 부모 노드를 의미하는 q를 선언하고 p는 root, q는 nullptr로 초기화한다. 자막 시간과 일치하는 노드를 찾아야 하므로 p가 nullptr이 아니고 p의 자막 시간이 인자의 시간과 일치하지 않는 동안 반복하여 p를 q에 저장하고 인자의 시간보다 p의 시간이 크다면 p는 왼쪽 자식 노드로 이동, 작다면 오른쪽 자식 노드를 이동함을 반복한다. 이렇게 반복한 후, p가 nullptr이라면 BST에는 인자의 자막 시간을 가진 노드가 존재하지 않으므로 flag를 true로 초기화 후 함수를 종료한다. p가 nullptr이라면 p에는 내가 삭제하고 싶은 노드가 저장되고 q는 p의 부모 노드가 된다.

1) p가 리프 노드인 경우
p의 왼쪽, 오른쪽 자식이 모두 없다면 p는 리프 노드이다. q가 nulltptr 이라면 p는 루트라는 뜻이므로 root를 nullptr로 초기화한다. q의 왼쪽 자식이 p라면 q의 왼쪽 자식을 nullptr로 초기화하고 q의 오른쪽 자식이 p라면 q의 오른쪽 자식을 nullptr로 초기화하여 q와 p의 연결을 끊는다. 이후 p를 delete하여 메모리까지 삭제해서 p를 BST에서 삭제한다.

2) p가 오른쪽 자식 노드만 가지는 경우
p의 왼쪽 자식이 nullptr이라면 p는 오른쪽 자식 노드만 가진다. 마찬가지로 q가 nullptr 이라면 p는 루트 노드이므로 새로운 루트 노드는 p의 오른쪽 자식이 된다. q의 왼쪽 자식 노드가 p인 경우에는 q의 새로운 왼쪽 자식 노드를 p의 오른쪽 자식 노드로 설정하고 q의 오른쪽 자식 노드가 p인 경우는 q의 새로운 오른쪽 자식 노드를 p의 오른쪽 자식 노드로 설정한다. 이렇게 되면 중간에 끼인 p가 삭제되어도 남아있는 노드를 이어 붙여서 BST가 될 수 있다.
마지막으로 p의 메모리를 삭제한다.

3) p가 왼쪽 자식 노드만 가지는 경우
2번과 똑같은 상황인데 달라지는 것은 p가 루트 노드인 경우, 새로운 루트 노드가 p의 왼쪽 자식이 된다는 것과 q의 왼쪽 자식 노드가 p인 경우, q의 새로운 왼쪽 자식 노드를 p의 왼쪽 자식 노드로 설정하고 q의 오른쪽 자식 노드가 p인 경우, q의 새로운 오른쪽 자식 노드가 p의 왼쪽 자식 노드가 된다는 것이 달라진다. 즉, 2번의 상황에서 오른쪽 노드 대신 왼쪽 노드를 이어 붙이는 셈이다.

4) p가 2개의 자식 노드를 가지는 경우
p가 양쪽 자식을 모두 가지는 경우, p를 대체할 노드는 p의 왼쪽 서브 트리에서 가장 큰 노드, p의 오른쪽 서브 트리에서 가장 작은 노드이다. 왜냐하면 BST는 왼쪽 자식 노드<현재 노드<오른쪽 자식 노드를 만족해야 한다. 왼쪽 서브 트리의 최댓값보다 큰 값은 현재 노드의 왼쪽 서브 트리에 존재하지 않으므로 BST의 속성을 깨지 않고, 오른쪽 서브 트리의 최솟값보다 작은 값은 현재 노드의 오른쪽 서브 트리에 존재하지 않으므로 내가 삭제할 노드를 대체해도 BST의 속성을 깨지 않는다. 따라서 두 방법 중, 하나를 선택하여 삭제할 노드를 대체하는 방식으로 노드를 삭제하도록 구현했고, 이번 프로젝트에서는 후자의 방법을 택했다.

prev의 부모를 의미하는 prevprev, 삭제할 노드를 대체할 오른쪽 서브 트리에서 가장 큰 노드를 의미하는 prev, prev의 왼쪽 자식 노드인 curr을 선언하고 prevprev는 p, prev는 p의 오른쪽 자식 노드, curr은 p의 오른쪽 자식 노드의 왼쪽 자식 노드로 초기화한다. 이제 curr이 nullptr이 아닐 때까지 반복하며 prevprev에는 prev를, prev에는 curr을 저장하고 curr은 curr의 왼쪽 자식 노드로 이동한다. curr이 nullptr이 되면 prev는 p의 오른쪽 서브 트리의 가장 왼쪽 노드가 된다. 따라서 이는 p를 대체할 수 있는 오른쪽 서브트리에서 가장 작은 값을 가지는 노드가 된다. 따라서 p의 자막 시간과 내용을 prev의 자막 시간과 내용으로 변경한다. 이렇게 변경하고 난 후에 2가지 경우가 있는데, 하나는 prevprev가 p와 같은 노드를 가리키는 경우이다. 이 경우에는 prevprev(p)의 오른쪽 노드가 prev이므로 prevprev의 오른쪽 자식 노드를 prev의 오른쪽 자식 노드로 설정하여 prev를 트리에서 없앤다. 해당 경우가 아니라면 p에서부터 오른쪽 자식으로 이동 후 쭉 왼쪽 자식으로 이동했으므로 prevprev는 p

의 오른쪽 서브 트리에 존재하고 prev는 prevprev의 왼쪽 자식 노드가 된다. 따라서 prevprev의 왼쪽 자식 노드를 prev의 오른쪽 자식 노드로 설정하여 prev를 트리에서 없앤다. 마지막으로 prev의 메모리를 해제하면 2개의 자식 노드를 가지는 노드의 삭제가 완료된다.

void SubtitleBST::delete_under(SubtitleBSTNode* node, string time, bool& flag, stack<SubtitleBSTNode*>& stk)

DELETE UNDER 명령어를 수행하기 위해서 위에서 설명한 delete_under 함수를 이용하여 인자로 전달받은 시간보다 이른 시간을 가지는 노드를 모두 삭제하는 함수이다. 재귀를 사용하여 먼저 왼쪽 자식 노드를 전달하여 자기 자신을 호출하고 현재 노드의 자막 시간이 인자로 전달받은 시간보다 이르다면 참조로 전달받은 SubtitleBSTNode*형 스택에 Push 한다. 참조로 전달을 받았으므로 Manager 클래스에 선언된 스택에도 영향을 미칠 수 있다. 따라서 Manager 클래스의 스택에도 해당 노드가 Push 되고, 이후에 Manager 클래스의 delete_under 함수에서 BST 클래스의 delete_equal 함수에 노드의 정보를 전달하여 삭제하는 방식으로 구현했다.

<SectionListNode>

Section_List는 헤더 노드와 내용 노드로 구성되는데, 헤더 노드는 SectionListNode 객체, 내용 노드는 SubtitleListNode로 구현된다. 따라서 해당 클래스는 리스트의 헤더 노드를 의미하는 클래스이다. 헤더 노드는 섹션 번호를 가지므로 int형 변수, 다음 헤더 노드를 가리킬 SectionListNode*형 변수, 해당 섹션의 내용 노드를 가리킬 SubtitleListNode*형 변수를 멤버 변수로 가진다. 헤더는 헤더를 가리키고 내용 노드를 가리키므로 Section_List는 2차원의 형태를 가진다고 생각할 수 있다. 해당 클래스로 리스트를 구성하므로 멤버의 데이터 얻고 멤버를 세팅하기 위해서 get, set 함수도 정의했다.

<SectionList>

Section_List를 의미하는 클래스이다. 리스트는 Head를 바탕으로 2차원으로 뻗어가는 형태로, Head가 왼쪽 맨 위에 존재한다면 세로로는 헤더 노드, 가로로는 내용 노드로 구성된다고 생각할 수 있다. SectionListNode 객체는 헤더, 내용 노드 모두를 가리킬 수 있으므로 멤버 변수는 Head를 의미하는 SectionListNode*형 변수 하나만 선언했다.

멤버 함수로는 헤더 노드를 추가하는 insert_head, 내용 노드를 추가하는 insert_content, 인자로 숫자를 받아서 해당 섹션 번호를 가지는 헤더 노드를 반환하는 search, 인자로 숫자를 받아서 해당 섹션 번호를 가지는 헤더 노드의 내용 노드를 출력하는 print 함수가 있다.

생성자는 이니셜라이저를 이용하여 head를 nullptr로 초기화한다. 소멸자는 리스트가 2차원이므로 현재 헤더 노드, 다음 헤더 노드를 의미하는 변수와 현재 내용 노드, 다음 내용 노드를 의미하는 변수 총 4개를 선언한 후, 현재 헤더 노드를 head로 지정한다. 현재 헤더 노드가 nullptr이 아닐 때까지 반복하는데 현재 헤더 노드의 다음 헤더 노드와 현재 헤더 노드의

다음 내용 노드(즉 현재 내용 노드)를 새로 저장한 후 현재 헤더 노드의 메모리를 해제한다. 이제 현재 내용 노드가 nullptr이 아닐 때까지 반복하며 현재 내용 노드의 다음 내용 노드를 저장하고 현재 내용 노드의 메모리를 해제한다. 내용 노드의 메모리를 모두 해제하면 다음 헤더 노드로 넘어가 해당 헤더 노드의 메모리를 해제하고 이어진 내용 노드의 메모리를 해제하는 방식으로 메모리를 해제했다.

void SectionList::insert_head(int num)
헤더 노드를 삽입하는 함수로, search 함수를 통하여 인자로 받은 숫자를 섹션 번호로 가지는 헤더를 탐색한다. 만약 nullptr이 아니라면 해당 섹션 번호를 이미 가지는 헤더 노드가 존재한다는 뜻이므로 함수를 종료한다. 이미 가지고 있는 헤더 노드가 없다면 인자로 받은 숫자를 섹션 번호로 하는 헤더 노드를 새롭게 생성한다. Head가 nullptr 이라면 리스트가 비어있으므로 head는 새롭게 생성한 헤더 노드가 된다. Head가 nullptr이 아니라면 이미 리스트에 여러 헤더가 존재하므로 Head부터 nullptr이 아닐 때까지 이동하며 마지막 헤더 노드로 이동한 후, 마지막 헤더 노드의 다음 헤더 노드로 새롭게 생성한 헤더 노드를 연결한다.

void SectionList::insert_content(int num, string sub_time, string sub_cont)
섹션 번호에 맞는 리스트에 내용 노드를 삽입하는 함수로, 인자로 섹션 번호, 자막 시간, 자막 내용을 전달받는다. 전달받은 자막 시간과 내용으로 새로운 내용 노드를 하나 생성한 후, search 함수를 통하여 num과 일치하는 헤더 노드를 저장한다. 리스트를 구축할 때 insert_head와 insert_content 함수에 같은 인자를 전달하여 노드를 구축하기에 존재하지 않는 섹션 번호에 내용 노드를 삽입할 수 있는 상황은 존재하지 않지만, 해당 헤더 노드가 nullptr이라면 num과 일치하는 헤더가 존재하지 않으므로 함수를 종료하도록 구현은 해두었다. 헤더 노드의 바로 다음 내용 노드가 nullptr이라면 헤더 노드의 바로 다음 내용 노드로 새로운 내용 노드를 저장하고 아니라면 헤더 노드의 바로 다음 내용 노드부터 nullptr 까지 순회하며 다음 노드로 이동하여 마지막 내용 노드의 다음 내용 노드로 새로운 내용 노드를 저장했다.

inline SectionListNode* SectionList::search(int num)
인자로 전달된 숫자를 섹션 번호로 가지는 헤더 노드를 반환하는 함수로, head부터 시작하여 nullptr까지 헤더 노드를 순회하여 현재 헤더 노드의 섹션 번호가 num과 일치하면 해당 헤더 노드를 반환하고 아니라면 다음 헤더 노드로 이동한다. while문이 끝났는데도 노드를 반환하지 못하고 nullptr에 도달하면 해당 리스트에는 num과 같은 섹션 번호를 가지는 헤더 노드가 없다는 뜻이므로 nullptr을 반환한다. 해당 함수가 SectionList 클래스에서 자주 호출될 것으로 판단하여 inline 함수로 정의했다.

void SectionList::print(int num, bool& flag, ofstream& flag)
num과 같은 섹션 번호를 가지는 헤더 노드와 연결된 내용 노드를 차례대로 출력하는 함수이다. search 함수를 통해서 num과 같은 섹션 번호를 가지는 헤더 노드가 존재하는지 판단하고 없다면 flag를 false로 초기화하고 함수를 종료한다. 이렇게 되면 Manager 클래스의 리스

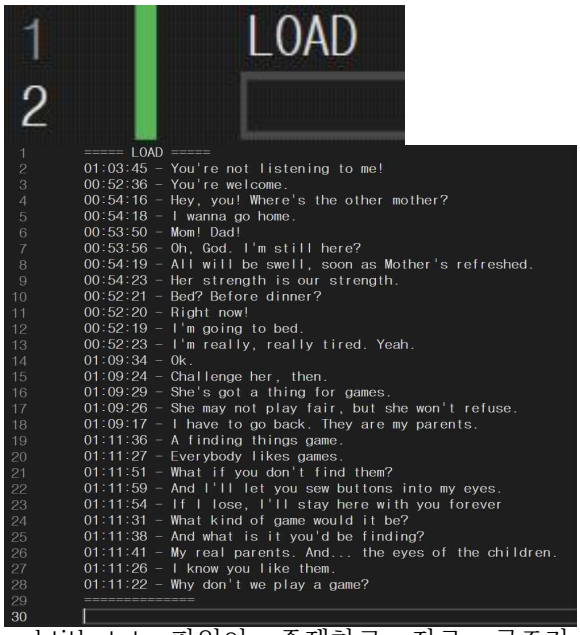
트를 출력하는 함수에서 flag를 인식하여 에러 코드를 출력한다. 번호가 맞는 헤더 노드가 존재하면 해당 헤더 노드의 바로 다음 내용 노드부터 nullptr이 아닐 때까지 순회하며 자막 시간- 자막 내용의 양식으로 출력 후, 다음 내용 노드로 이동하도록 구현했다.

BST의 print 함수와 다르게 해당 번호에 맞는 헤더 노드가 있는지 확인하고 존재한다면 출력하기 때문에, Manager 클래스의 print 함수에서 출력이 가능한지 아닌지를 미리 확인할 수 없다. 따라서 PRINT 명령어 수행 시에 나타나는 출력 양식과 섹션 번호를 출력하는 코드를 Manager 클래스의 print 함수가 아닌, 리스트 클래스의 print에 작성했다.

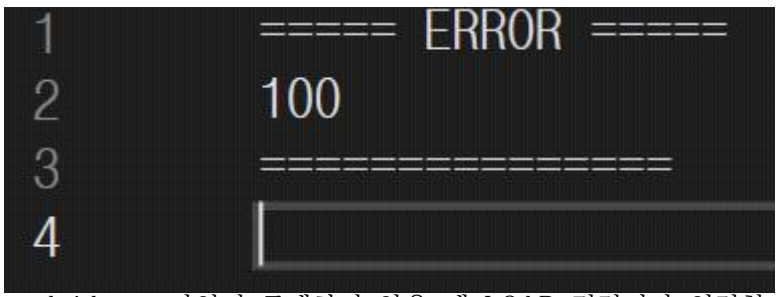
4. Result Screen-모든 명령어의 결과화면과 예외 처리의 동작을 설명

각 명령어마다 입력과 결과화면의 출력을 첨부하고 설명하는 방식으로 작성했다.

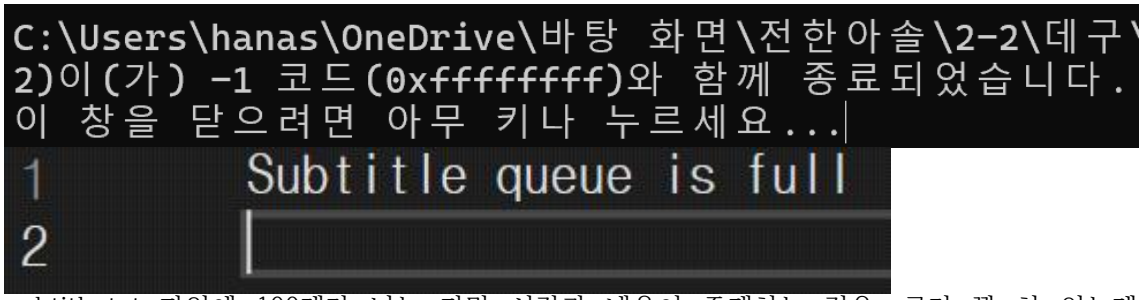
1. LOAD



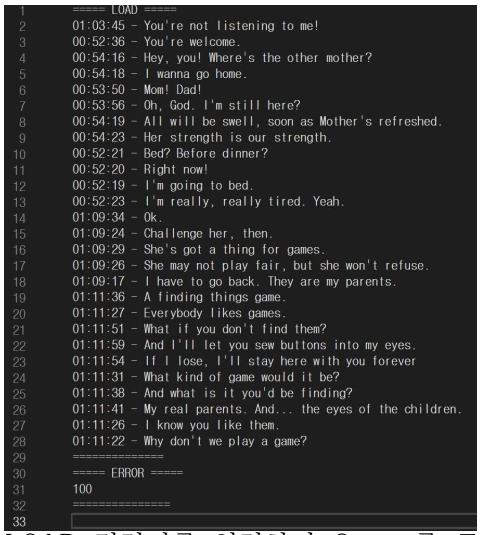
subtitle.txt 파일이 존재하고 자료 구조가 비어있을 때, LOAD 명령어를 입력하면 subtitle.txt에 존재하는 자막 내용과 시간이 잘 출력되는 결과를 볼 수 있다.



subtitle.txt 파일이 존재하지 않을 때 LOAD 명령어만 입력한 결과로, LOAD의 에러 코드인 100이 출력되는 결과가 나타난다.



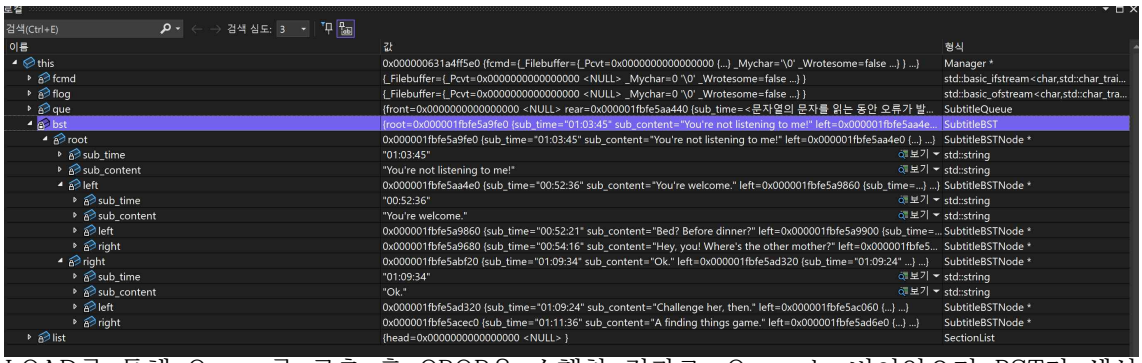
subtitle.txt 파일에 100개가 넘는 자막 시간과 내용이 존재하는 경우, 큐가 꽉 차 있는데 Push 연산을 시행했으므로 큐가 꽉 찼다는 메시지를 출력하며 -1을 반환하고 프로그램을 종료하는 결과가 나타난다.



LOAD 명령어를 입력하여 Queue를 구축한 상태에서 LOAD를 입력한 결과로, 이미 자료 구조에 데이터가 들어가 있으므로 에러 코드가 출력되는 결과를 볼 수 있다.

2. QPOP

```
1   ===== LOAD =====
2   01:03:45 - You're not listening to me!
3   00:52:36 - You're welcome.
4   00:54:16 - Hey, you! Where's the other mother?
5   00:54:18 - I wanna go home.
6   00:53:50 - Mom! Dad!
7   00:53:56 - Oh, God. I'm still here?
8   00:54:19 - All will be swell, soon as Mother's refreshed.
9   00:54:23 - Her strength is our strength.
10  00:52:21 - Bed? Before dinner?
11  00:52:20 - Right now!
12  00:52:19 - I'm going to bed.
13  00:52:23 - I'm really, really tired. Yeah.
14  01:09:34 - Ok.
15  01:09:24 - Challenge her, then.
16  01:09:29 - She's got a thing for games.
17  01:09:26 - She may not play fair, but she won't refuse.
18  01:09:17 - I have to go back. They are my parents.
19  01:11:36 - A finding things game.
20  01:11:27 - Everybody likes games.
21  01:11:51 - What if you don't find them?
22  01:11:59 - And I'll let you sew buttons into my eyes.
23  01:11:54 - If I lose, I'll stay here with you forever
24  01:11:31 - What kind of game would it be?
25  01:11:38 - And what is it you'd be finding?
26  01:11:41 - My real parents. And... the eyes of the children.
27  01:11:26 - I know you like them.
28  01:11:22 - Why don't we play a game?
29  ==============
30  ===== QPOP =====
31  Success
32  ==============
```



LOAD를 통해 Queue를 구축 후 QPOP을 수행한 결과로, Queue는 비어있으며 BST가 생성된 모습을 볼 수 있다.

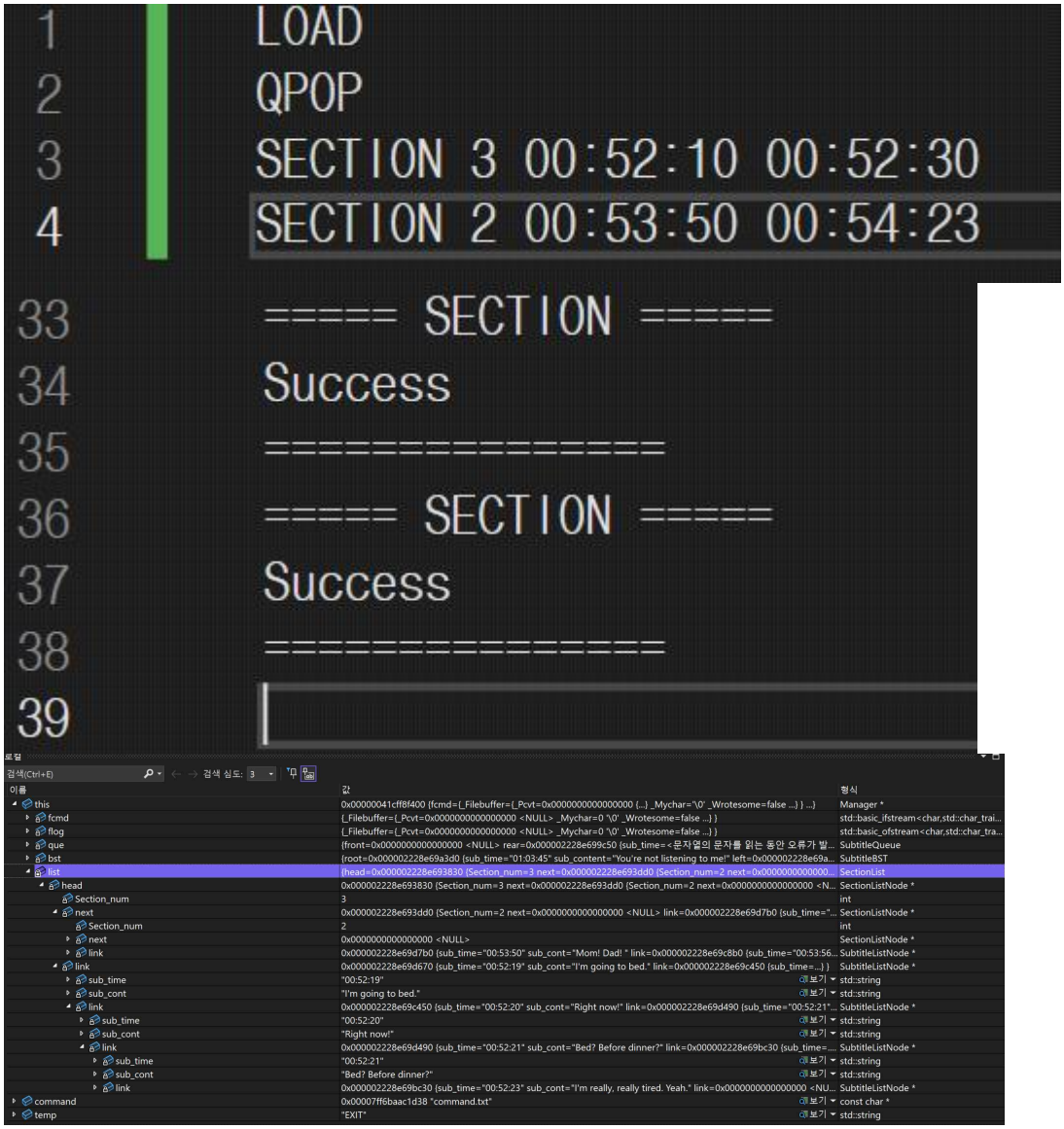
```
1   QPOP
```

```
C:\Users\hanas\OneDrive\바탕 화면\전한아솔\2-2\데구
6)이(가) -1 코드(0xffffffff)와 함께 종료되었습니다.
이 창을 닫으려면 아무 키나 누르세요...
```

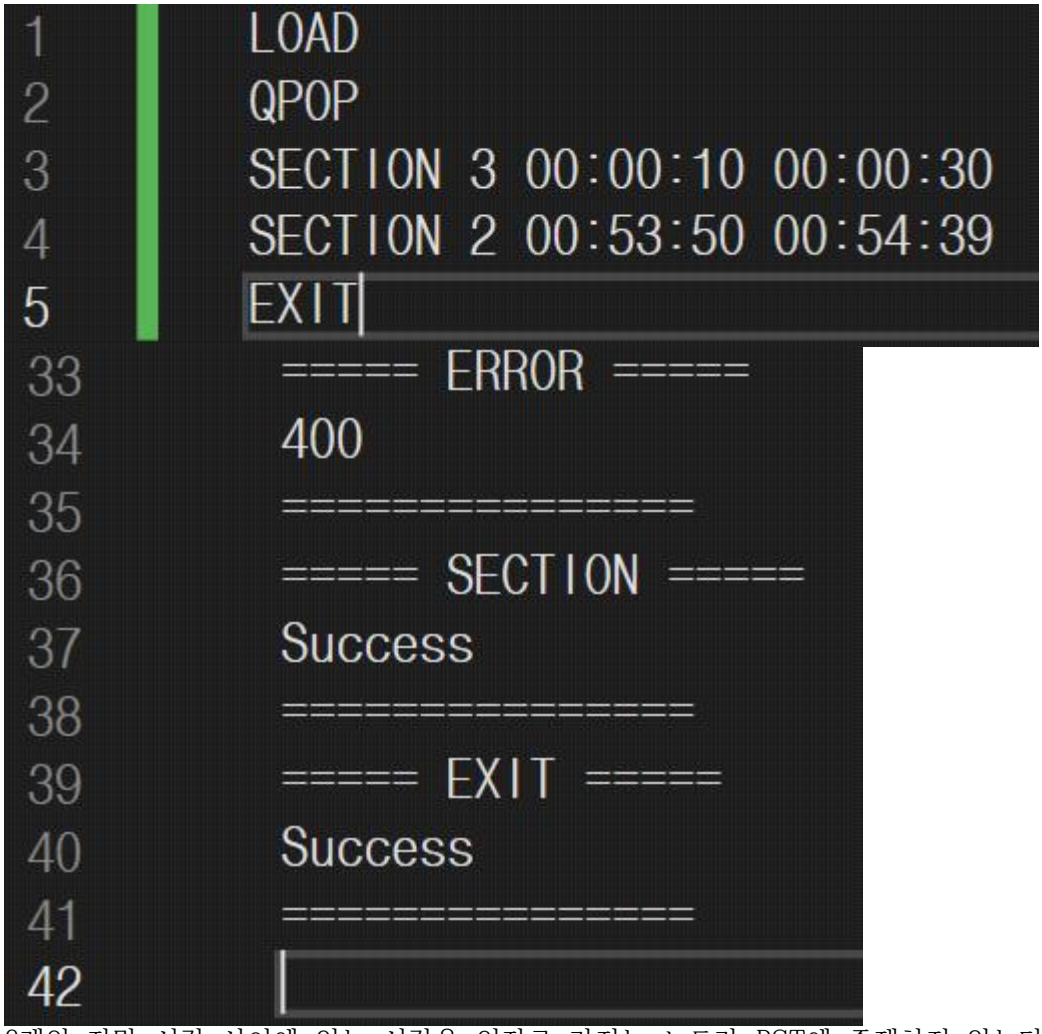
```
1   ===== ERROR =====
2   200
3   ================
4   Subtitle queue is empty
5
```

처음부터 QPOP 명령어를 입력한 결과로 Queue가 비어있는 상태에서 POP 연산을 진행했으므로 -1을 반환하며 프로그램이 종료되고 큐가 비어있는 상태에서 QPOP 연산이 진행되었으므로 에러 코드를 출력한 결과를 볼 수 있다.

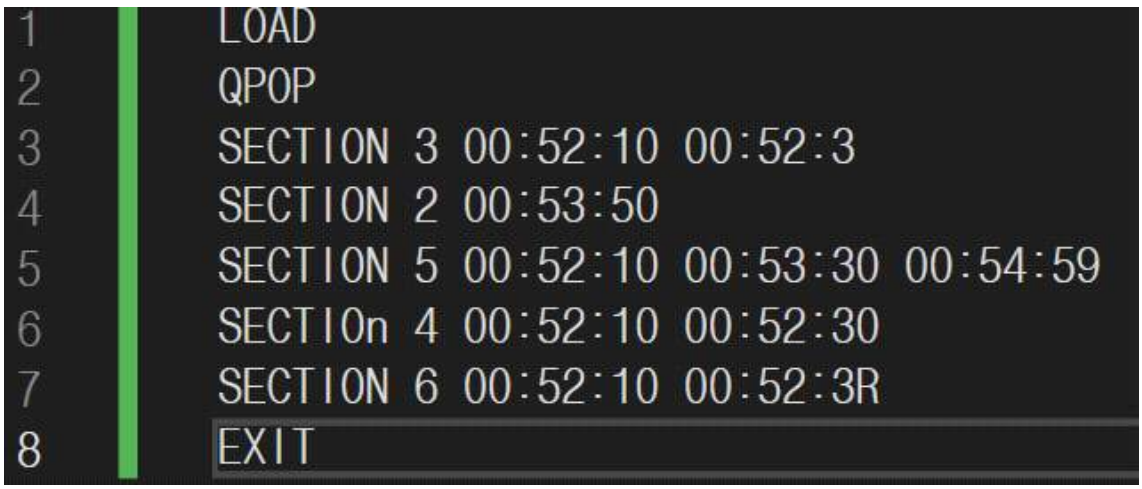
3. SECTION

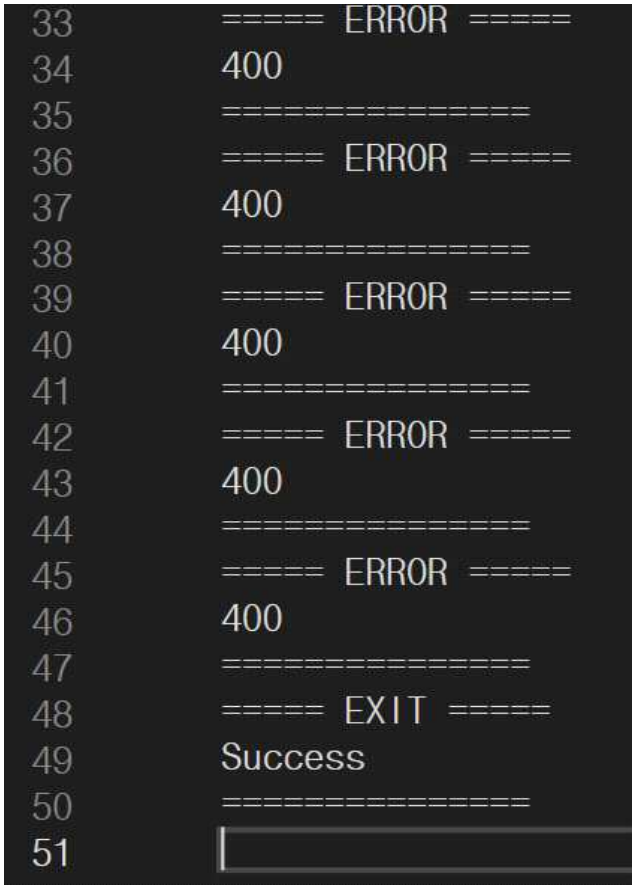


00:52:10~00:52:30의 시간에 해당하는 노드를 섹션 번호 3으로 구축하고 00:53:50~00:54:23의 시간에 해당하는 노드를 섹션 번호 2로 구축한 결과이다. 3을 먼저 구축했으므로 3번 섹션의 헤더 노드가 head가 되고 해당 해더는 섹션 번호 2의 헤더를 next로 가리키며 link로 00:52:10~00:52:30의 시간을 가지는 내용 노드를 가리키는 2차원 연결 리스트의 형태를 구성한 결과를 볼 수 있다.

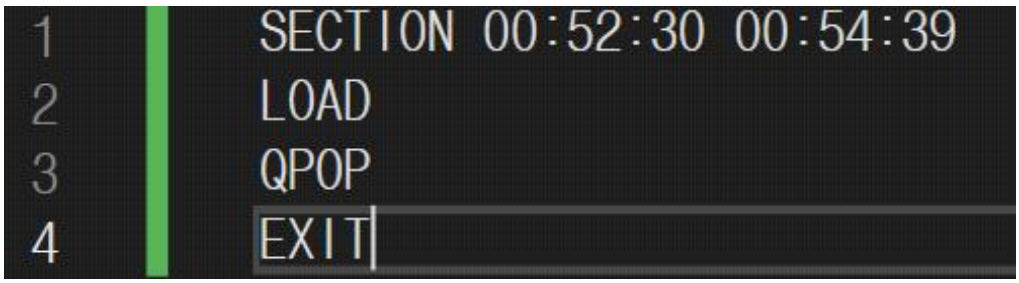


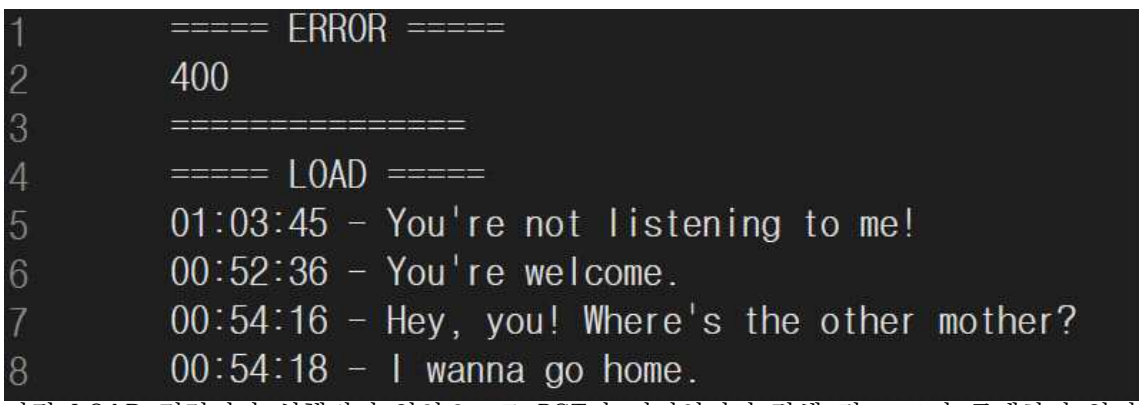
2개의 자막 시간 사이에 있는 시간을 인자로 가지는 노드가 BST에 존재하지 않는다면 에러 코드를 출력하는 결과를 볼 수 있다.





차례대로 자막 시간의 형식이 안 맞는 경우, 인자가 부족한 경우, 인자가 과다한 경우, 명령어가 대문자가 아닌 경우, 자막 시간이 숫자가 아닌 경우 모두 에러 코드를 출력하는 결과를 볼 수 있다.

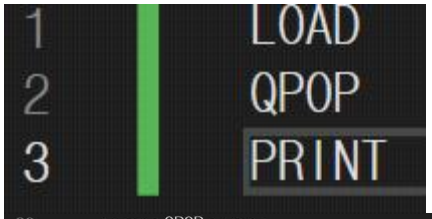




아직 LOAD 명령어가 실행되지 않았으므로 BST가 비어있어서 탐색 된 노드가 존재하지 않기

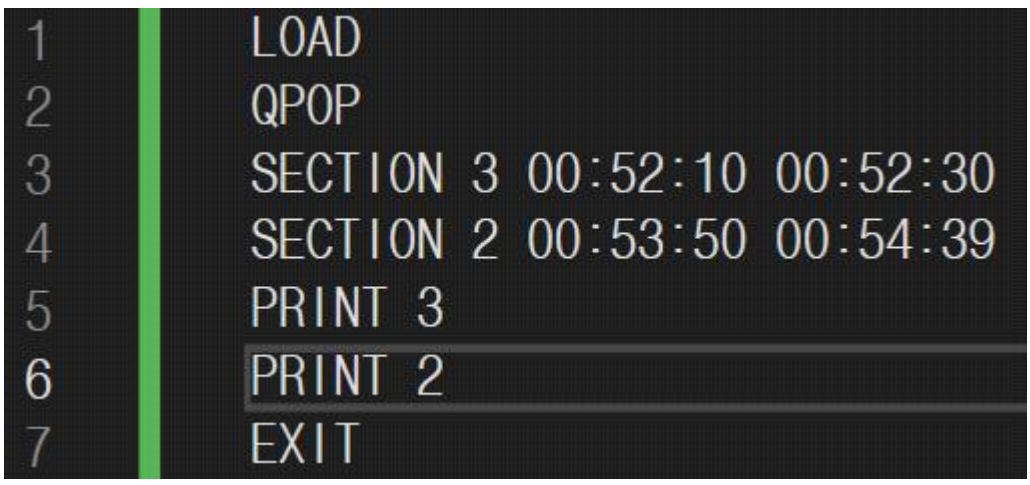
때문에 에러가 나타나는 결과를 볼 수 있다.

4. PRINT




```
30      ===== QPOP =====
31      Success
32      ===============
33      ===== PRINT =====
34      00:52:19 - I'm going to bed.
35      00:52:20 - Right now!
36      00:52:21 - Bed? Before dinner?
37      00:52:23 - I'm really, really tired. Yeah.
38      00:52:36 - You're welcome.
39      00:53:50 - Mom! Dad!
40      00:53:56 - Oh, God. I'm still here?
41      00:54:16 - Hey, you! Where's the other mother?
42      00:54:18 - I wanna go home.
43      00:54:19 - All will be swell, soon as Mother's refreshed.
44      00:54:23 - Her strength is our strength.
45      01:03:45 - You're not listening to me!
46      01:09:17 - I have to go back. They are my parents.
47      01:09:24 - Challenge her, then.
48      01:09:26 - She may not play fair, but she won't refuse.
49      01:09:29 - She's got a thing for games.
50      01:09:34 - Ok.
51      01:11:22 - Why don't we play a game?
52      01:11:26 - I know you like them.
53      01:11:27 - Everybody likes games.
54      01:11:31 - What kind of game would it be?
55      01:11:36 - A finding things game.
56      01:11:38 - And what is it you'd be finding?
57      01:11:41 - My real parents. And... the eyes of the children.
58      01:11:51 - What if you don't find them?
59      01:11:54 - If I lose, I'll stay here with you forever
60      01:11:59 - And I'll let you sew buttons into my eyes.
61      ===============
62
```

인자로 아무것도 전달되지 않은 PIRNT를 실행한 결과로, QPOP을 통해서 구축한 BST를 중위 순회하여 출력하므로 시간이 오름차순으로 정렬되어 출력된 결과를 볼 수 있다.



```
39      ===== PRINT =====
40      Section 3
41      00:52:19 - I'm going to bed.
42      00:52:20 - Right now!
43      00:52:21 - Bed? Before dinner?
44      00:52:23 - I'm really, really tired. Yeah.
45      ===============
46      ===== PRINT =====
47      Section 2
48      00:53:50 - Mom! Dad!
49      00:53:56 - Oh, God. I'm still here?
50      00:54:16 - Hey, you! Where's the other mother?
51      00:54:18 - I wanna go home.
52      00:54:19 - All will be swell, soon as Mother's refreshed.
53      00:54:23 - Her strength is our strength.
54      ===============
55      ===== EXIT =====
56      Success
57      ===============
58
```

SECTION의 인자로 3과 2를 전달하여 각 섹션 번호를 가지는 리스트를 구성했고 PRINT의 인자로 3과 2를 전달하여 3의 섹션 번호를 가지는 헤더의 내용 노드와 2의 섹션 번호를 가지는 헤더의 내용 노드들을 차례로 출력한 결과를 볼 수 있다.

```
1       PRINT
2       PRINT 3
3       LOAD
4       QPOP
5       SECTION 3 00:52:10 00:52:30
6       SECTION 2 00:53:50 00:54:39
7       PRINT 4
8       PRINt
9       PRINT X
10      EXIT
```

```
1       ===== ERROR =====
2       300
3       ===============
4       ===== ERROR =====
5       300
6       ===============
7       ===== LOAD =====
8       01:03:45 - You're not listening to me!
9       00:52:36 - You're welcome.
10      00:54:16 - Hey, you! Where's the other mother?
11      00:54:18 - I wanna go home.
12      00:53:50 - Mom! Dad!
```

```
43      Success
44      ==============
45      ===== ERROR =====
46      300
47      ==============
48      ===== ERROR =====
49      300
50      ==============
51      ===== ERROR =====
52      300
53      ==============
54      ===== EXIT =====
55      Success
56      ==============
57
```

자료구조에 아무것도 존재하지 않을 때 BST와 리스트를 출력한 결과, PRINT의 에러 코드인 300이 출력되는 결과가 나타났다. 또한 헤더 노드가 존재하지 않는 섹션 번호를 입력한 경우, 대문자가 아닌 경우, 섹션 번호로 전달한 인자가 숫자가 아닌 경우 모두 에러 코드가 출력되는 결과를 볼 수 있다.

5. DELETE

```
1       LOAD
2       QPOP
3       PRINT
4       DELETE EQUAL 00:52:36
5       PRINT
6       DELETE UNDER 00:53:50
7       PRINT
8       EXIT
```

```
===== PRINT =====
00:52:19 - I'm going to bed.
00:52:20 - Right now!
00:52:21 - Bed? Before dinner?
00:52:23 - I'm really, really tired. Yeah.
00:52:36 - You're welcome.
00:53:50 - Mom! Dad!
00:53:56 - Oh, God. I'm still here?
00:54:16 - Hey, you! Where's the other mother?
00:54:18 - I wanna go home.
00:54:19 - All will be swell, soon as Mother's refreshed.
00:54:23 - Her strength is our strength.
01:03:45 - You're not listening to me!
01:09:17 - I have to go back. They are my parents.
01:09:24 - Challenge her, then.
01:09:26 - She may not play fair, but she won't refuse.
01:09:29 - She's got a thing for games.
01:09:34 - Ok.
01:11:22 - Why don't we play a game?
01:11:26 - I know you like them.
01:11:27 - Everybody likes games.
01:11:31 - What kind of game would it be?
01:11:36 - A finding things game.
01:11:38 - And what is it you'd be finding?
01:11:41 - My real parents. And... the eyes of the children.
01:11:51 - What if you don't find them?
01:11:54 - If I lose, I'll stay here with you forever
01:11:59 - And I'll let you sew buttons into my eyes.
===============
```

BST에 저장된 데이터

```
62      ===== DELETE =====
63      Success
64      ===============
65      ===== PRINT =====
66      00:52:19 - I'm going to bed.
67      00:52:20 - Right now!
68      00:52:21 - Bed? Before dinner?
69      00:52:23 - I'm really, really tired. Yeah.
70      00:53:50 - Mom! Dad!
71      00:53:56 - Oh, God. I'm still here?
72      00:54:16 - Hey, you! Where's the other mother?
73      00:54:18 - I wanna go home.
74      00:54:19 - All will be swell, soon as Mother's refreshed.
75      00:54:23 - Her strength is our strength.
76      01:03:45 - You're not listening to me!
77      01:09:17 - I have to go back. They are my parents.
78      01:09:24 - Challenge her, then.
79      01:09:26 - She may not play fair, but she won't refuse.
80      01:09:29 - She's got a thing for games.
81      01:09:34 - Ok.
82      01:11:22 - Why don't we play a game?
83      01:11:26 - I know you like them.
84      01:11:27 - Everybody likes games.
85      01:11:31 - What kind of game would it be?
86      01:11:36 - A finding things game.
87      01:11:38 - And what is it you'd be finding?
88      01:11:41 - My real parents. And... the eyes of the children.
89      01:11:51 - What if you don't find them?
90      01:11:54 - If I lose, I'll stay here with you forever
91      01:11:59 - And I'll let you sew buttons into my eyes.
```

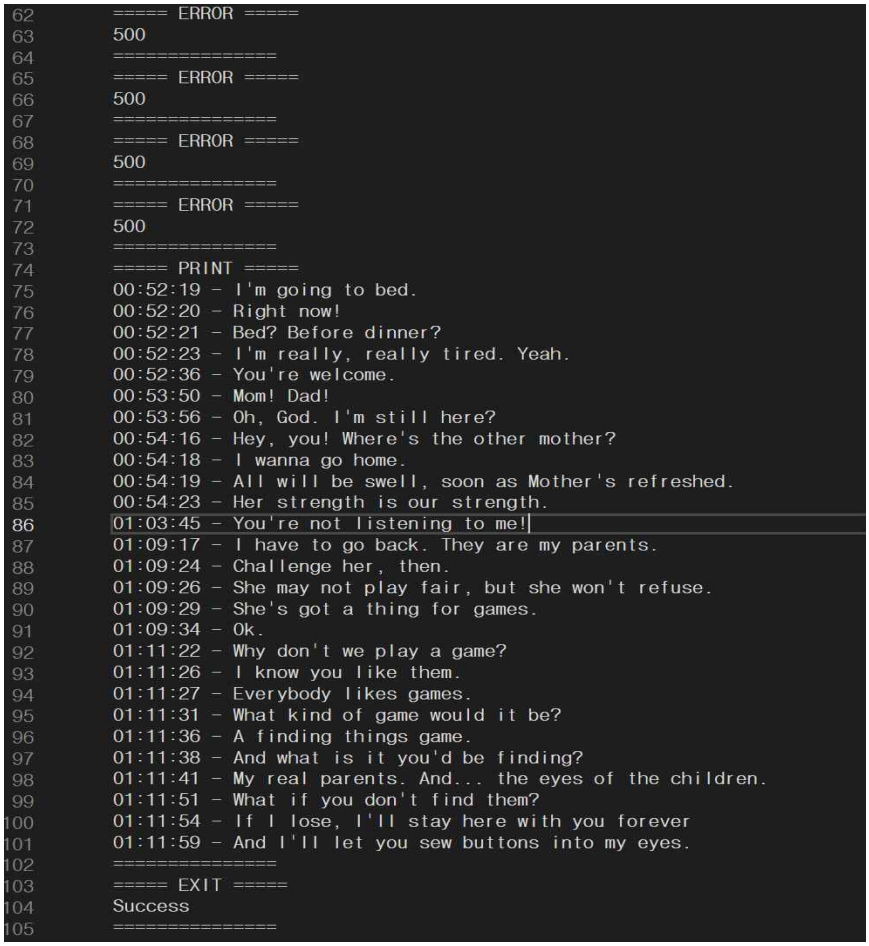
첫 번째 삭제(DELETE EQUAL 00:52:36)를 진행한 후, BST에 저장된 데이터를 출력한 결과,

00:52:36의 자막 시간을 가지는 내용이 삭제된 결과가 나타났다.

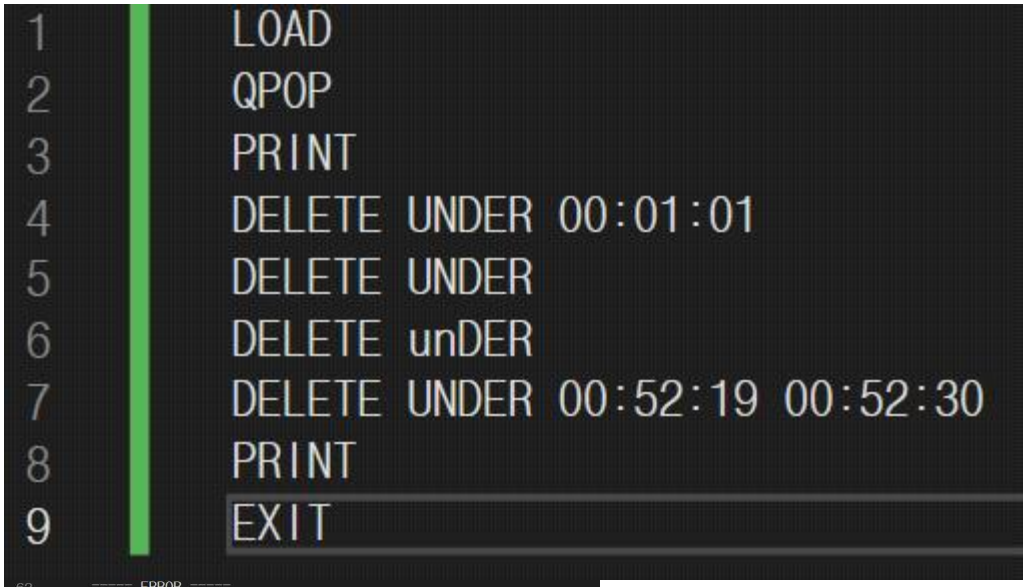
```
93      ===== DELETE =====
94      Success
95      ==============
96      ===== PRINT =====
97      00:53:50 - Mom! Dad!
98      00:53:56 - Oh, God. I'm still here?
99      00:54:16 - Hey, you! Where's the other mother?
100     00:54:18 - I wanna go home.
101     00:54:19 - All will be swell, soon as Mother's refreshed.
102     00:54:23 - Her strength is our strength.
103     01:03:45 - You're not listening to me!
104     01:09:17 - I have to go back. They are my parents.
105     01:09:24 - Challenge her, then.
106     01:09:26 - She may not play fair, but she won't refuse.
107     01:09:29 - She's got a thing for games.
108     01:09:34 - Ok.
109     01:11:22 - Why don't we play a game?
110     01:11:26 - I know you like them.
111     01:11:27 - Everybody likes games.
112     01:11:31 - What kind of game would it be?
113     01:11:36 - A finding things game.
114     01:11:38 - And what is it you'd be finding?
115     01:11:41 - My real parents. And... the eyes of the children.
116     01:11:51 - What if you don't find them?
117     01:11:54 - If I lose, I'll stay here with you forever
118     01:11:59 - And I'll let you sew buttons into my eyes.
119     ==============
120     ===== EXIT =====
121     Success
122     ==============
```

두 번째 삭제(DELETE UNDER 00:53:50)를 진행한 후, BST에 저장된 데이터를 출력한 결과, 00:53:50의 자막 시간보다 이른 시간을 가진 내용이 모두 삭제된 결과를 볼 수 있다.
두 번의 삭제를 진행한 후, BST에 저장된 데이터를 보았을 때, 각각 EQUAL과 UNDER의 명령어가 잘 작동되는 결과를 볼 수 있다.

```
1   LOAD
2   QPOP
3   PRINT
4   DELETE EQUAL 00:00:00
5   DELETE EQUAL
6   DELETE Equal
7   DELETE EQUAL 00:52:19 00:52:30
8   PRINT
9   EXIT
```

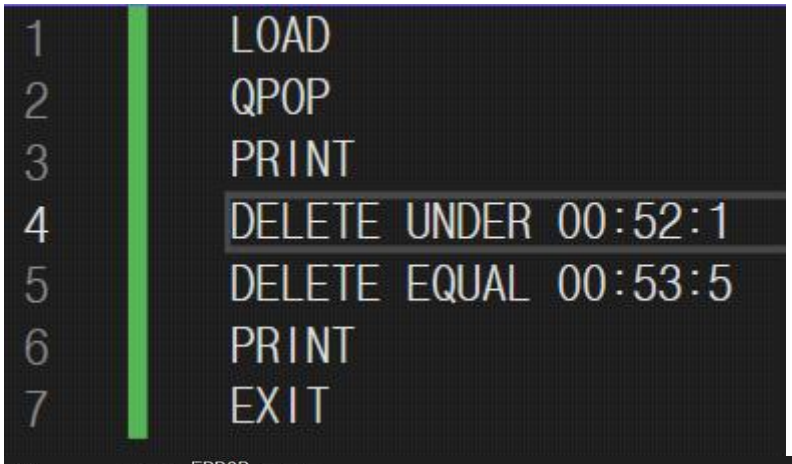


차례대로 인자로 전달된 자막 시간과 일치하는 노드가 존재하지 않는 경우, 인자가 부족한 경우, 명령어가 대문자가 아닌 경우, 인자가 과도한 경우 모두 에러 코드를 출력하는 결과를 볼 수 있다. 또한 BST에서 아무 노드도 삭제되지 않고 이전에 출력한 결과와 동일한 출력이 나타난다.




```
62      ===== ERROR =====
63      500
64      ==============
65      ===== ERROR =====
66      500
67      ==============
68      ===== ERROR =====
69      500
70      ==============
71      ===== ERROR =====
72      500
73      ==============
74      ===== PRINT =====
75      00:52:19 - I'm going to bed.
76      00:52:20 - Right now!
77      00:52:21 - Bed? Before dinner?
78      00:52:23 - I'm really, really tired. Yeah.
79      00:52:36 - You're welcome.
80      00:53:50 - Mom! Dad!
81      00:53:56 - Oh, God. I'm still here?
82      00:54:16 - Hey, you! Where's the other mother?
83      00:54:18 - I wanna go home.
84      00:54:19 - All will be swell, soon as Mother's refreshed.
85      00:54:23 - Her strength is our strength.
86      01:03:45 - You're not listening to me!
87      01:09:17 - I have to go back. They are my parents.
88      01:09:24 - Challenge her, then.
89      01:09:26 - She may not play fair, but she won't refuse.
90      01:09:29 - She's got a thing for games.
91      01:09:34 - Ok.
92      01:11:22 - Why don't we play a game?
93      01:11:26 - I know you like them.
94      01:11:27 - Everybody likes games.
95      01:11:31 - What kind of game would it be?
96      01:11:36 - A finding things game.
97      01:11:38 - And what is it you'd be finding?
98      01:11:41 - My real parents. And... the eyes of the children.
99      01:11:51 - What if you don't find them?
100     01:11:54 - If I lose, I'll stay here with you forever
101     01:11:59 - And I'll let you sew buttons into my eyes.
102     ==============
103     ===== EXIT =====
104     Success
105     ==============
106
```

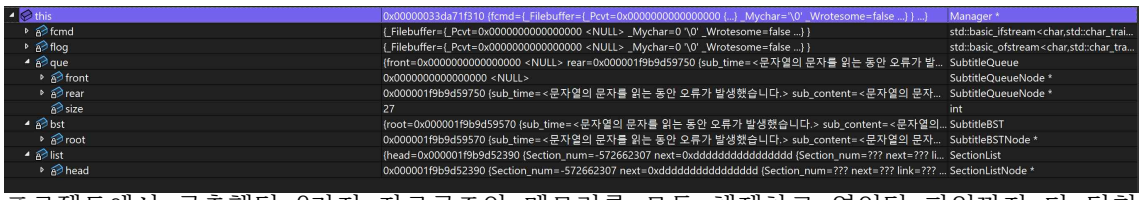
EQUAL과 마찬가지로 DELETE UNDER 또한 해당되는 자막 시간을 가진 노드가 존재하지 않는 경우, 인자가 부족한 경우, 명령어가 대문자가 아닌 경우, 인자가 과도한 경우 모두 에러 코드를 출력하고 BST 아무 노드도 삭제되지 않고 이전의 결과와 동일한 출력이 나타난 결과를 볼 수 있다.




```
62  ===== ERROR =====
63  500
64  ===============
65  ===== ERROR =====
66  500
67  ===============
68  ===== PRINT =====
69  00:52:19 - I'm going to bed.
70  00:52:20 - Right now!
71  00:52:21 - Bed? Before dinner?
72  00:52:23 - I'm really, really tired. Yeah.
73  00:52:36 - You're welcome.
74  00:53:50 - Mom! Dad!
75  00:53:56 - Oh, God. I'm still here?
76  00:54:16 - Hey, you! Where's the other mother?
77  00:54:18 - I wanna go home.
78  00:54:19 - All will be swell, soon as Mother's refreshed.
79  00:54:23 - Her strength is our strength.
80  01:03:45 - You're not listening to me!
81  01:09:17 - I have to go back. They are my parents.
82  01:09:24 - Challenge her, then.
83  01:09:26 - She may not play fair, but she won't refuse.
84  01:09:29 - She's got a thing for games.
85  01:09:34 - Ok.
86  01:11:22 - Why don't we play a game?
87  01:11:26 - I know you like them.
88  01:11:27 - Everybody likes games.
89  01:11:31 - What kind of game would it be?
90  01:11:36 - A finding things game.
91  01:11:38 - And what is it you'd be finding?
92  01:11:41 - My real parents. And... the eyes of the children.
93  01:11:51 - What if you don't find them?
94  01:11:54 - If I lose, I'll stay here with you forever
95  01:11:59 - And I'll let you sew buttons into my eyes.
96  ===============
97  ===== EXIT =====
98  Success
99  ===============
```

DELTE UNDER과 EQUAL 모두 인자로 전달되는 자막 시간의 형식이 맞지 않으면 에러가 출력되고 아무런 노드도 삭제되지 않는 결과를 볼 수 있다.

6. EXIT



프로젝트에서 구축했던 3가지 자료구조의 메모리를 모두 해제하고 열었던 파일까지 다 닫힌 후에 프로그램을 종료하는 결과를 볼 수 있다.

6. Consideration-고찰

1. 이번 프로젝트를 통해서 BST의 삭제 연산을 처음 구현해 보아서 구현 과정에서 많은 어려움이 있었다. 노드를 삭제하고 해당 노드의 부모 노드에 자식 노드를 nullptr로 초기화하여 삭제를 완성해야 하는데, 노드만 삭제하고 다음 코드를 작성하여 BST의 삭제를 수행할 때 프로그램이 비정상적으로 종료되는 상황이 있었다. 디버깅을 통해 삭제할 노드의 부모 노드에 자식 노드에 접근할 때, 에러가 나는 것을 보고 해당 코드를 수정할 수 있었다.

2. 자막 시간을 string형 변수에 저장하여 비교 연산자를 통해서 비교하는 방식으로 이번 프로젝트를 구현했는데, 비교 연산자로 비교하는 과정에서 분명 제대로 된 시간을 string 변수에 저장했는데, 에러가 발생하는 상황이 있었다. 디버깅을 통해서 해당 string을 보니 파일에 있던 공백이 같이 string에 저장되어서 제대로 된 비교가 이루어지지 않음을 발견했다. command 파일에 존재하는 명령어의 맨 마지막은 개행이 되도록 공백을 모두 삭제하니 제대로 된 비교가 이루어졌다.

3. Algorithm 부분에서도 설명했지만 SECTION 명령어는 BST에서 탐색 된 노드가 없으면 리스트를 생성하지 않고 에러 코드를 출력하는데, 인자로 전달된 섹션 번호로 먼저 헤더 노드를 만들고 BST에서 탐색을 진행하는 방식으로 코드를 작성하여 탐색 된 노드가 존재하지 않아도 헤더 노드가 생성되어 리스트가 출력되는 상황이 있었다. 따라서 Manager 클래스가 아닌 BST 클래스에서 노드를 탐색하고 탐색 된 노드가 존재하면 헤더 노드와 내용 노드를 구축하는 방식으로 구현했다. 이렇게 구현하면 헤더 노드를 계속 구축할 수 있으므로 헤더 노드를 구축하는 함수에서 중복을 확인하여 해당 섹션 번호를 가지는 헤더 노드가 존재하지 않을 때만 생성하도록 구현하여 해당 문제를 해결했다.

4. 리스트로 Queue를 구현 해하고 제안서를 확인하던 중, Queue의 크기가 100으로 초기화되고 이후에 변하지 않는다는 말이 Queue를 배열로 구현해야 한다는 뜻이라는 생각이 들어서 해당 Queue의 구현 방식을 배열로 바꿔야 하나 라고 고민했다. 하지만 Github의 issue를 통해서 질문한 결과, 구현 방식은 상관없고 Queue 클래스의 isFull 함수를 통해서 100개의 원소만 수용할 수 있도록 구현했으므로 이는 크기가 100인 Queue와 다름없이 동작한다고 생각한다.

5. 각 자료구조의 노드를 의미하는 클래스가 분리되어 있고 해당 멤버 변수는 private으로 선

언되어 있어서 각 자료구조 클래스에서 해당 노드의 멤버 변수에 접근하지 못하고 pubilc으로 선언된 멤버 함수를 통하여 데이터를 얻어오고 멤버 변수를 바꿀 수 있다. 해당 상황이 불편하여 강의자료에 나와 있는 friend 키워드를 사용하여 private 멤버에 직접 접근할 수 있도록 구현할지 고민하던 와중, 이는 OOP의 특성인 Encapsulation을 위배하는, 적합하지 않은 방법이라고 생각하여 friend 키워드를 사용하지 않고 public으로 get, set 함수들을 정의하여 멤버의 값을 얻거나 멤버 변수를 바꿀 수 있도록 구현했다.