2021년 1학기

**프로그래밍과 문제해결**

Assignment #4

담당 교수 : 윤은영

학번 : 20210084

학과 : 무은재학부

이름 : 김지민

POVIS ID : kjm1672

================명예서약(Honor code)===============

“나는 이 프로그래밍 과제를 다른 사람의 부적절한 도움 없이 완수하였습니다.”

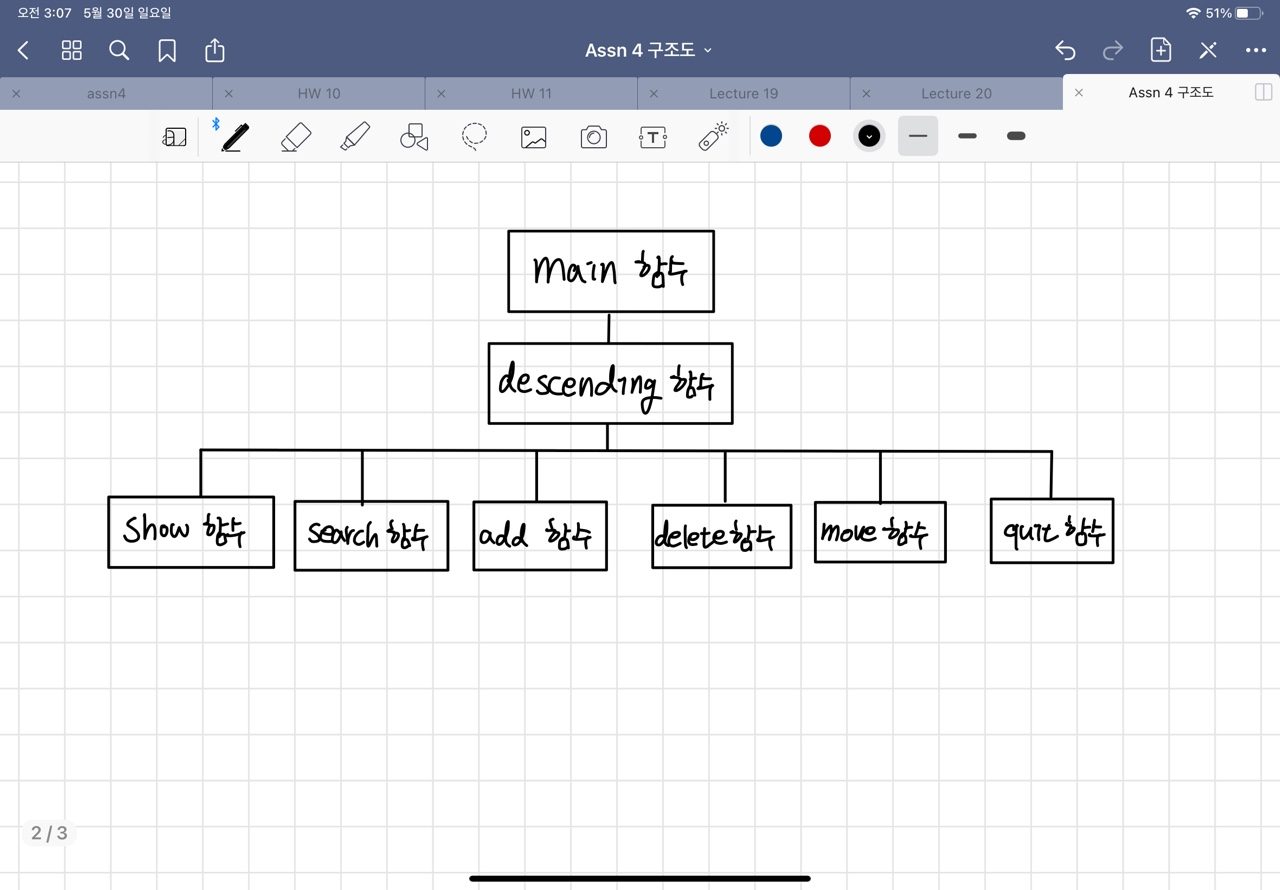
==============================================

1. 문제의 개요

본 프로그램을 간략히 소개하면 다음과 같다.

* 플레이리스트와 같이 음악의 정보를 저장하고 관리하는 것을 목적으로 한다.
* 사용자로부터 파일 이름을 입력 받는다.
* 텍스트 파일로 저장된 음악 정보를 읽고 플레이리스트에 저장한다. 이때, 내림차순으로 정렬한다.
* 사용자로부터 명령어를 입력 받는다.
* 출력, 검색, 추가, 삭제, 순서 이동 등의 기능을 구현한다.
* “quit”을 입력 받으면, 현재 플레이리스트를 파일로 출력하고 프로그램을 종료한다.

이때 구조 차트 (structure chart)는 아래와 같이 표현될 수 있다.

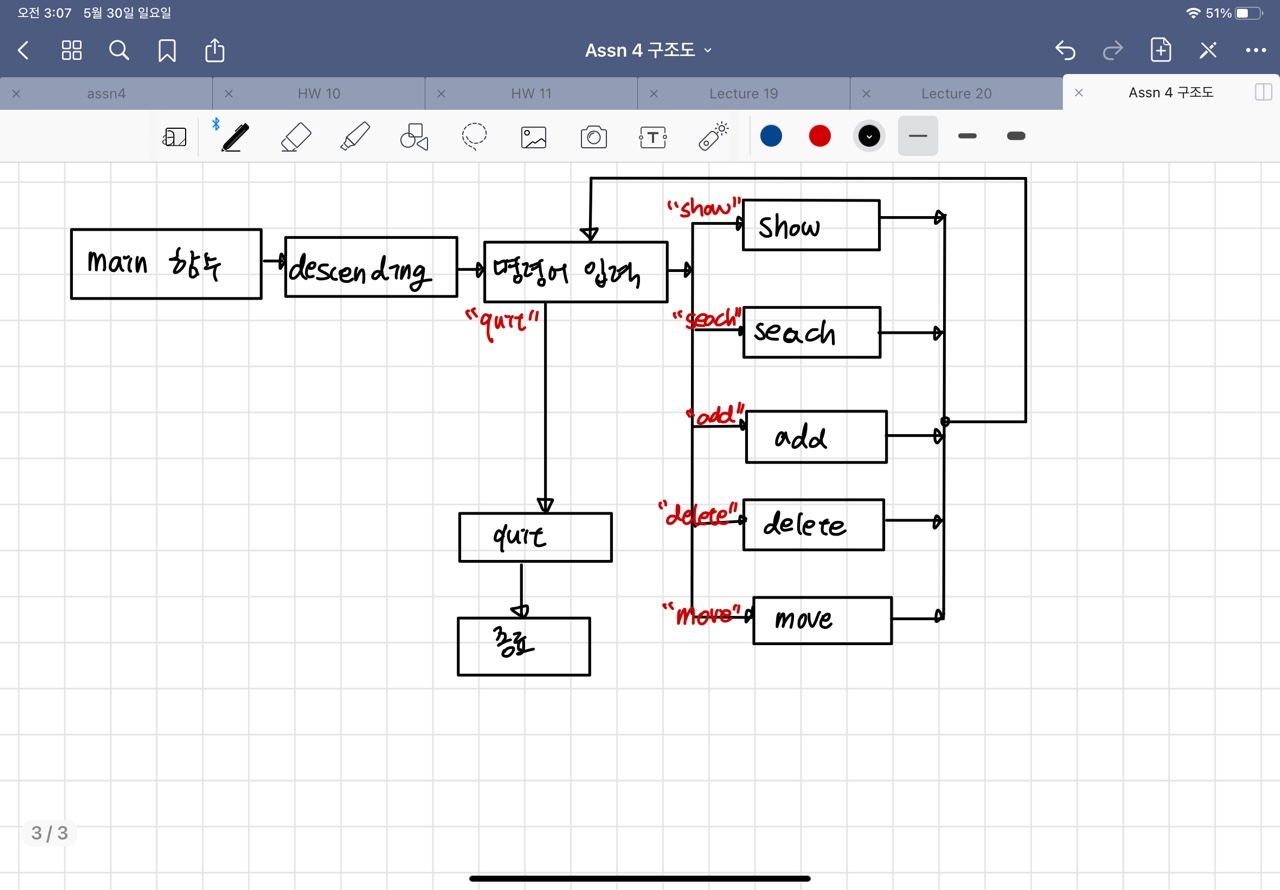


1. 알고리즘

본 프로그램 작성을 위한 알고리즘을 Presudo 코드 형태로 나타내면 다음과 같다.

|  |
| --- |
| Presudo-algorithm |
| 1. 헤더파일 <stdio.h>, <stdlib.h>, <string.h>를 포함시킵니다. 2. 사용자로부터 파일 이름을 입력 받는다. 파일 이름에는 공백이 없으며 30자를 넘지 않는다고 가정한다.   - 존재하지 않는 파일 이름을 입력 시, 에러 메시지를 출력하고 다시 입력 받습니다.  3. 파일의 각 줄에는 하나의 음악정보가 들어가며 제목, 아티스트, 재생 시간, 재생 횟수로 구성 된다. MUSIC 데이터 구조체에 하나의 음악정보를 저장하도록 한다. 그리고 NODE 데이터 구조체 안에 저장 후, 각 NODE 데이터 구조체를 연결 리스트 (PLAYLIST)로 연결한다. 차례대로 연결한 후, descending 함수를 호출하여 리스트를 내림차순으로 정렬한다.  4. 플레이리스트에서 상위 3개의 음악 정보를 출력한 후 사용자로부터 명령어를 입력 받는다.  - “show”, “search”, “add”, “delete”, “move”, “quit”이외의 명령어를 입력 시, 에러 메시지를 출력 후 다시 입력 받는다.  5. 각 명령어에 따른 작업을 수행한다. 이때 각 명령어 별로 함수를 정의하여 사용한다.  - “show”: 플레이리스트의 내용을 재생횟수의 오름차순으로 출력한다. 음악의 총 개수와 총 재생 시간 또한 출력한다.  - “search”: 사용자로부터 아티스트의 이름을 입력 받고 그 아티스트의 음악 정보를 출력한다. 출력된 음악의 총 개수와 총 재생 시간 또한 출력한다. 만약 없는 아티스트의 이름을 입력 시, 다시 명령어를 입력 받는 과정으로 돌아간다.  - “add”: 사용자가 원하는 순서에 음악을 추가한다. 사용자로부터 추가할 순서, 추가할 음악의 제목, 아티스트, 재생시간, 재생 횟수를 입력 받는다. 이때 입력 받은 순서가 1보다 작거나 (현재 플레이리스트의 총 음악 개수 + 1)보다 클 경우 에러 메시지를 출력 후 다시 입력 받는다.  - “delete”: 사용자가 원하는 순서에 있는 음악 정보를 삭제한다. 이때 입력 받은 순서가 1보다 작거나 (현재 플레이리스트의 총 음악 개수)보다 클 경우 에러 메시지를 출력 후 다시 입력 받는다. 삭제할 음악의 정보를 출력 후 해당 음악 정보를 저장한 메모리를 할당 해제한다.  - “move”: 사용자가 원하는 순서로 음악 정보를 이동시킨다. 이때 입력 받은 순서가 1보다 작거나 (현재 플레이리스트의 총 음악 개수)보다 클 경우 에러 메시지를 출력 후 다시 입력 받는다. Swap(두 음악 순서를 바꾸는 것)이 아니며, 특정 음악의 순서가 변경됨에 따라 다른 음악의 순서는 밀린다.  - “quit”: 사용자로부터 원하는 파일명을 입력 받고, 현재까지 편집한 플레이리스트의 내용을 파일로 출력한다. 동적 할당 받은 메모리를 모두 할당 해제시킨 후, 프로그램을 종료시킨다. 이때 현재 플레이리스트에 내용이 없을 경우, 파일을 따로 출력하지 않는다. |

위 알고리즘을 flowchart 로 표현하면 아래와 같다.



1. 프로그램 구조 및 설명
2. 프로그램 시작

* struct 선언

1. MUSIC : 음악 정보를 저장한다. int count, time (재생 횟수 및 재생 시간), char title [], artist[](제목, 아티스트)로 구성되어 있다.
2. NODE : 각 음악의 순서를 저장할 int num과 음악정보를 저장할 MUSIC data, NODE형 포인터변수 next로 구성된다.
3. PLAYLIST: 총 음악 개수를 저장할 int num과 첫 번째 NODE를 가리킬 NODE형 포인터변수 head로 구성된다.

* main 함수

1. 프로그램을 시작하면 사용자로부터 음악정보를 불러올 파일명을 입력 받는다.
2. NODE 포인터변수 temp를 선언하여 음악 정보를 저장하고 PLAYLIST 포인터변수 plist와 순서대로 연결한다. 이때 뒤에서부터 차례대로 연결한다.
3. 각 음악 정보는 tab키로 구분되어 있으므로, 입력 받을 때 “%[^\t]s”로 받음으로써 공백이 있는 제목 및 아티스트의 정보도 받을 수 있도록 한다.
4. descending함수를 호출하여 plist와 연결된 리스트를 내림차순으로 정렬한다.
5. 상위 3개의 음악 정보를 출력한다.
6. while문 안에 명령어에 따른 기능을 수행하는 코드를 삽입함으로써 계속 사용자로부터 명령을 받을 수 있도록 한다.
7. strcmp를 이용하여 사용자로부터 입력 받은 명령어가 6개의 명령어 중 무엇과 일치하는지 검사하고, 해당 명령에 따른 함수를 호출한다. quit을 입력 받을 시, quit함수 호출 후 break를 걸어 while문 밖으로 나와 프로그램이 종료될 수 있도록 한다.
8. 플레이리스트 관리

* show 함수

1. NODE형 포인터변수 temp를 선언하고, temp = plist->head로 초기화 한다. for문을 활용하여 조건문 부분에 temp != NULL, 증감문 부분엔 temp = temp ->next를 넣어 temp가 NULL이 될 때 즉, 리스트의 끝을 넘어서면 종료하도록 한다.
2. 한 줄에 음악 순서, 제목, 아티스트, 재생 시간, 재생 횟수 순으로 출력되도록 한다. 재생 시간의 합을 구하기 위해 time\_sum 변수에 각 음악의 재생 시간을 더한다.
3. 반복문 종료 후, plist->num에 저장된 총 음악 개수와 time\_sum에 저장된 총 재생 시간을 출력한다.

* search 함수

1. 사용자로부터 어떤 아티스트의 음악 정보를 출력할지 입력 받는다.
2. for문과 if문을 이용하여 입력 받은 아티스트가 플레이리스트 내에 존재하는지 검사한다. 없으면 에러 메시지를 출력하고 다시 main 함수의 명령문 입력 부분으로 돌아간다.
3. for문과 if 문, strcmp을 이용하여 리스트의 처음부터 해당 아티스트가 존재하는지 검사하고 일치하면 그 음악 정보를 출력한다. 이때 time\_sum, total변수에 해당 아티스트의 모든 음악의 재생 시간의 합과 개수를 저장한다.
4. 반복문 종료 후, total에 저장된 해당 아티스트의 음악 개수와 time\_sum에 저장된 총 재생 시간을 출력한다.

* add 함수

1. NODE형 포인터변수 pnew를 선언하고 pnew->num에 원하는 음악 순서를 입력 받는다. 이때 1보다 작거나 (현재 음악 개수 + 1)보다 크면 다시 입력 받는다.
2. pnew->data부분을 사용자로부터 입력 받는다(추가할 음악의 제목, 아티스트, 재생 시간, 재생 횟수 순으로 차례대로).
3. if문으로 plist -> head == NULL인 경우 (음악이 하나도 없는 경우)와 그렇지 않은 경우(음악이 하나라도 있는 경우)를 구분한다. plist -> head == NULL인 경우, 새로 추가한 음악 정보가 첫 번째 순서가 되어야 하므로 다음과 같이 코딩한다.

if (plist->head == NULL)

{ pnew->next = plist->head; plist->head = pnew; }

1. 그렇지 않은 경우: NODE형 포인터변수 pre와 temp를 이용한다. pre는 temp의 이전 NODE를 가리킨다. 처음에 삽입하는 경우, 끝에 삽입하는 경우, 중간에 삽입하는 경우로 나누어 진행한다.

for (pre = plist->head, temp = pre; temp != NULL; pre = pre->next, temp = pre->next)

{

temp = pre->next;

if (pnew->num == 1)

{

pnew->next = plist->head;

plist->head = pnew;

break;

}

else if (temp->num == (plist->num) - 1 && pnew->num == (plist->num))

{

temp->next = pnew; break;

}

else if (temp->num == pnew->num)

{

pnew->next = pre->next;

pre->next = pnew;

break;

}

}

즉, 다음과 같이 코드를 구성할 수 있다.

1. 반복문을 이용하여 각 NODE의 num을 바꿈으로써 각 음악 순서를 재정의한다.

* delete 함수

1. plist->head == NULL인 경우 “플레이리스트에 삭제할 음악이 없습니다.”라는 메시지를 출력하고 다시 main 함수로 돌아간다.
2. NODE형 포인터변수 pnew를 선언하고 pnew->num에 원하는 음악 순서를 입력 받는다. 이때 1보다 작거나 (현재 음악 개수)보다 크면 다시 입력 받는다.
3. 첫 번째 NODE를 삭제하는 경우와 그렇지 않은 경우를 구분한다. NODE형 포인터변수 pre와 temp를 이용한다. temp는 pre의 다음 NODE를 가리키며, pre= plist->head로 초기화한다.

* 처음을 삭제 : pre가 가리키는 NODE의 음악 정보를 출력하고 plist->head가 pre의 다음 NODE를 가리키도록 즉, plist->head = temp로 바꾼다. 그런 다음 pre가 가리키는 첫 번째 노드를 동적 할당 해제하고 반복문 밖으로 빠져나온다.
* 중간이나 끝을 삭제 : pnew->num와 일치하는 값을 갖고 있는 temp->num을 조사하고 temp가 가리키는 노드의 음악 정보를 출력한다. 그런 다음 temp의 이전 노드를 가리키는 pre의 next가 temp->next값을 갖도록 즉, pre->next = temp->next로 바꾼다. 이후 temp가 가리키는 노드를 동적 할당 해제하고 반복문 밖으로 빠져나온다.

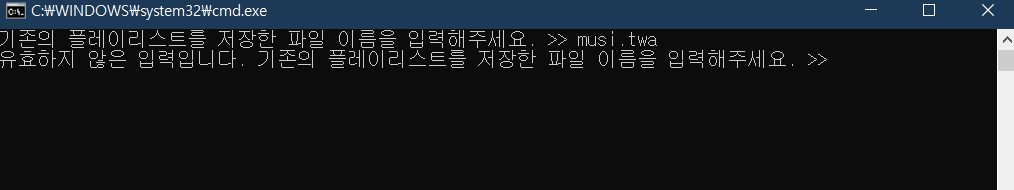
1. 반복문을 이용하여 각 NODE의 num을 바꿈으로써 각 음악 순서를 재정의한다.

* move 함수

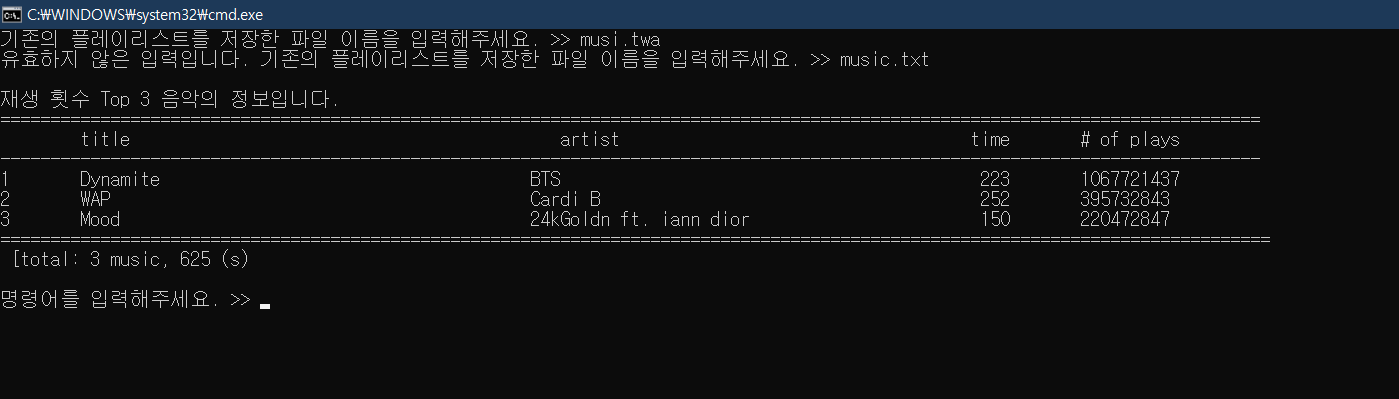
1. 총 음악 개수가 1인 경우 “플레이리스트에 순서를 이동할 음악이 없습니다.”를 출력한 후 다시 main 함수로 돌아간다.
2. 사용자로부터 몇 번째 순서의 음악 위치를 바꿀 것인지, 몇 번째 순서로 이동시킬 것인지 입력 받아 order1, order2 변수에 각각 저장한다.
3. order1과 같은 음악 순서를 가진 노드를 찾아 pnew로 가리키게 하고, 리스트에서 제외시킨다. (연결을 끊는다.)
4. order1이 order2보다 작은 경우 : order2와 같은 음악 순서를 가진 노드를 찾아 그 뒤에 pnew가 가리키는 노드를 연결시킨다.
5. order1이 order2보다 큰 경우 : order2와 같은 음악 순서를 가진 노드를 찾아 그 앞에 pnew가 가리키는 노드를 연결시킨다.
6. 반복문을 이용하여 각 NODE의 num을 바꿈으로써 각 음악 순서를 재정의한다.

* quit 함수

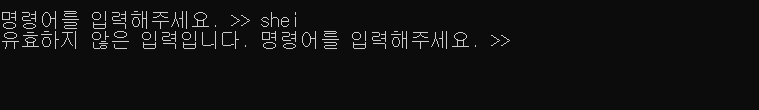
1. 만약 plist->head ==NULL인 경우 파일 출력을 진행하지 않는다.
2. 그렇지 않은 경우, 사용자로부터 저장할 파일명을 입력 받고, 제목, 아티스트, 재생 시간, 재생 횟수 순으로 플레이리스트를 파일로 출력한다.
3. for문을 이용하여 plist와 연결된 노드를 동적 할당 해제시키고, plist 또한 해제시키고 main 함수로 돌아간다. 이후 break로 인해 main 함수 반복문에서 빠져나오게 되고 프로그램이 종료하게 된다.
4. 프로그램 실행방법 및 예제



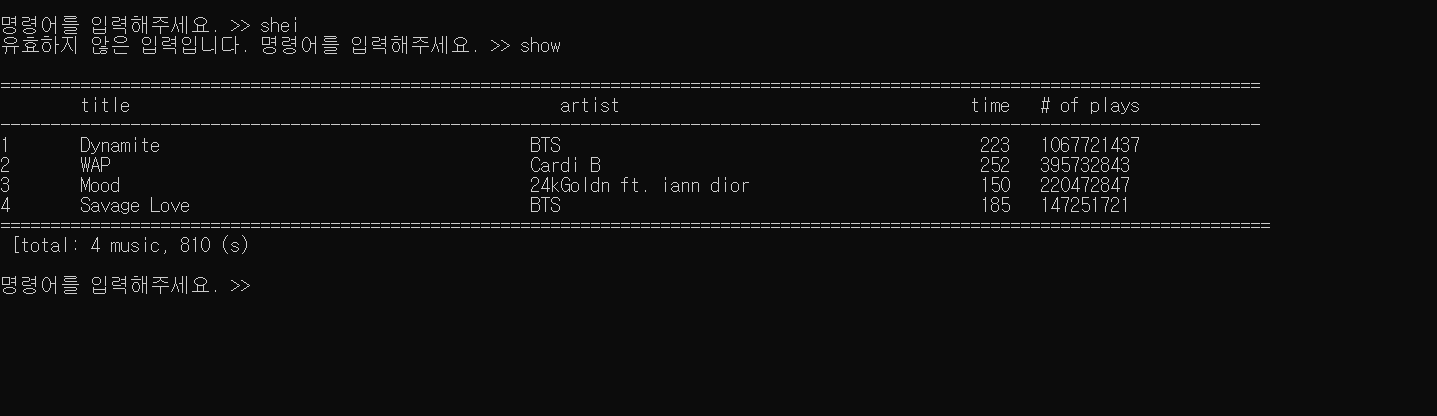
<처음 실행 화면>



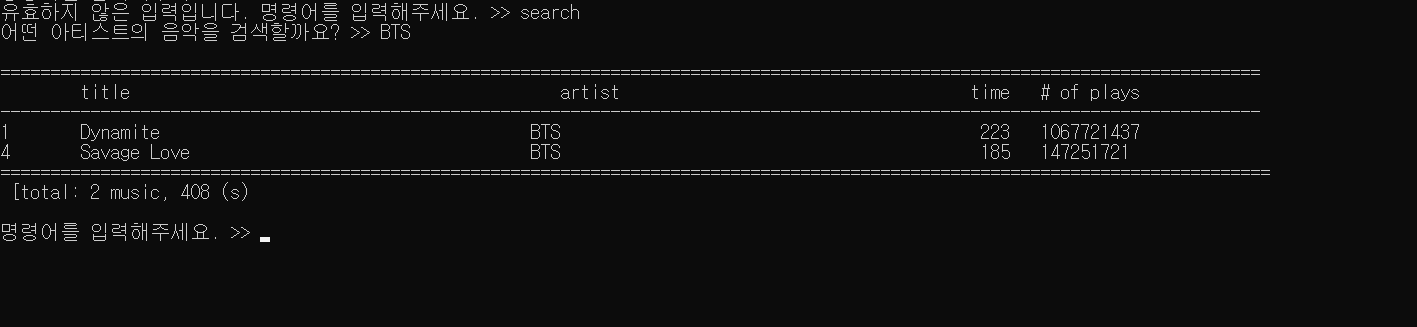
<상위 3개 음악 출력 및 명령어 입력>



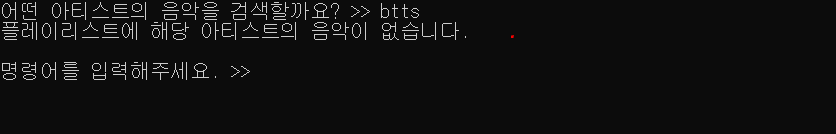
<잘못된 명령어 예외처리>



<”show” 명령어에 따른 출력>



<”search” 명령어에 따른 출력>



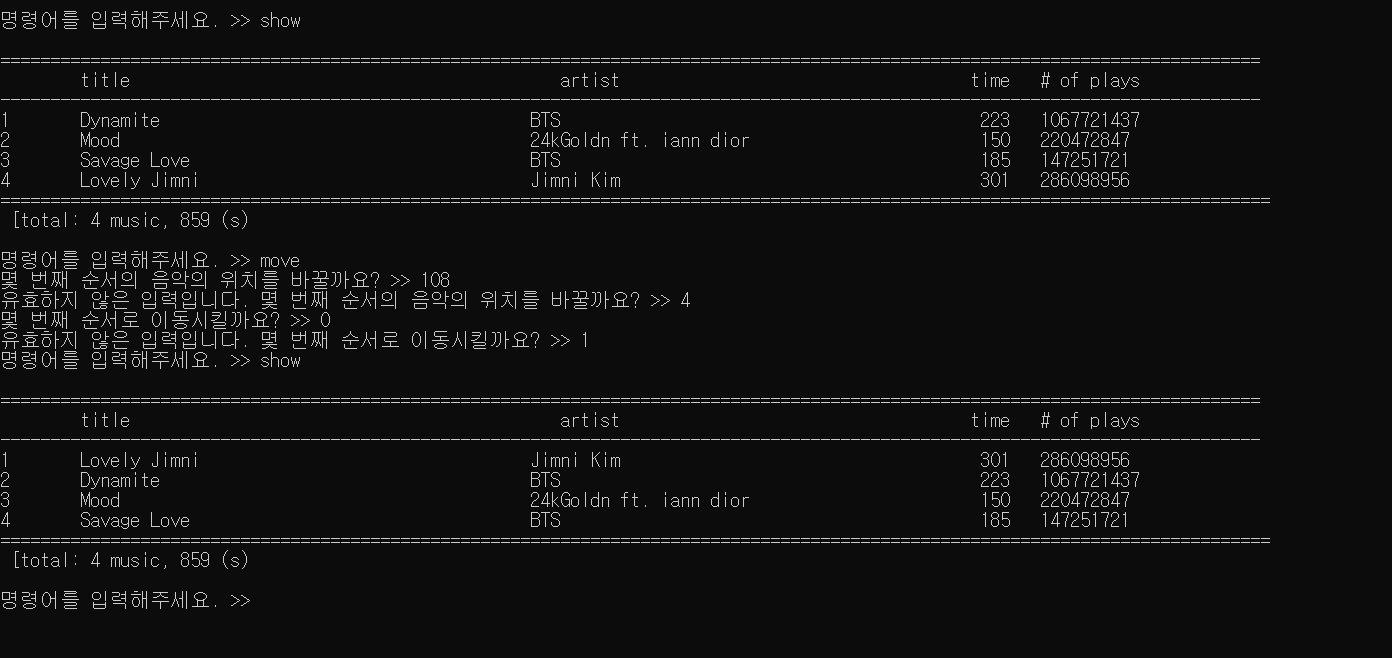
<잘못된 아티스트 입력 예외처리>



<”add”명령어에 따른 음악 추가 / 범위 밖 순서 입력에 따른 예외처리>



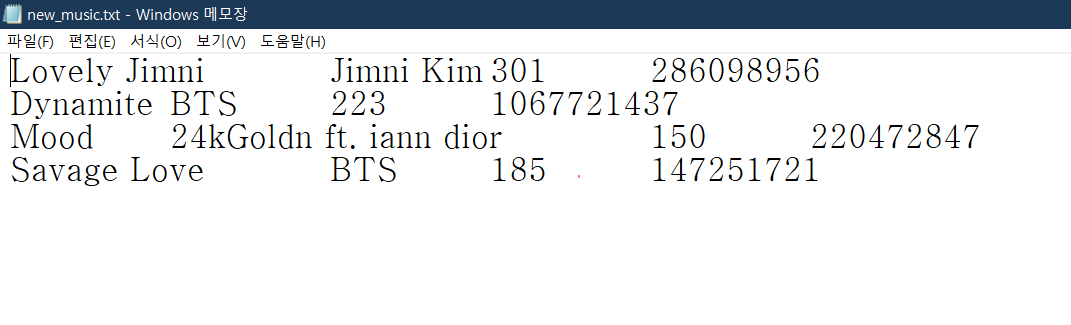
<”delete” 명령어에 따른 음악 삭제 / 범위 밖 순서 입력에 따른 예외처리>



<”move” 명령어에 따른 순서 이동 및 범위 밖 순서 입력에 따른 예외처리>



<”quit” 명령어에 따른 파일 출력 및 프로그램 종료>



<파일 출력된 모습>

1. 토론

* 파일에서 플레이리스트를 입력 받음과 동시에 내림차순으로 정렬하려 했으나 메인 함수의 코드가 너무 복잡해진다는 생각이 들었다.
* 우선 음악 정보를 입력 받는 대로 연결한 다음 내림차순으로 정렬하는 descending이라는 함수를 따로 선언하여 코드를 단순화하였다.

1. 결론

* 본 과제는 리스트와 구조체를 정확히 이해하고 있는지를 묻고 있다. 플레이리스트의 음악 정보를 내림차순으로 정렬하는 과정에서 각 포인터들이 어떻게 연결되어있는지 생각하는 능력을 키울 수 있었다. 또한 각 구조체에 접근하는 방법을 익힐 수 있었다.

1. 개선방향

* 본 과제에서 요구하는 사항들은 리스트와 구조체에 대한 이해가 충분히 되었다면 큰 어려움 없이 진행할 수 있던 것들이었다. 그러나 본인은 특히 리스트에 관한 이해가 부족하고 응용도가 떨어졌기 때문에 코딩 초기 방향을 잡는 것에 어려움을 겪었다. 이번 과제를 통해 리스트에 대한 응용을 연습해볼 수 있었으며 비슷한 문제가 나오면 큰 어려움 없이 진행할 수 있을 것이라 기대한다.