**Programming**

**Term Project Report**



|  |  |  |  |
| --- | --- | --- | --- |
| **제** | **출** | **일** | 2018년 12월 14일 |
| **학** |  | **부** | 컴퓨터공학부 |
| **학** |  | **번** | 20153734, 20173914 |
| **이** |  | **름** | 이정민, 김강산 |
| **분** |  | **반** | 프로그래밍 - 04분반 |
| **담 당 교 수** | | | 김무철 교수님 |



프 로 그 래 밍

**Index**

1. **Problems Description & Modeling**
2. Problem Description
3. Modeling
4. **Problem Solving**
5. Data Processing
6. Functions
7. **Implementation & Testing**
8. Debug with Use-case
9. **Evaluation**
10. Development issues
11. **Problems Description & Modeling**

**1)Problem Description**

1. Movie Lens Data를 통해 적절한 구조체를 만들어 데이터관리.
2. 콘솔을 통해 태그를 입력 받고, 해당 태그가 붙은 영화들을 출력하기.
3. 콘솔을 통해 영화이름을 입력 받고, 해당 영화에 붙은 태그들을 출력하기.
4. (Additional) 영화 이름의 일부만 입력하여, 상응하는 영화들의 전체 이름 출력
5. (Additional) 태그의 일부만 입력하여, 상응하는 태그의 전체 이름 출력
6. (Additional) 유저별로 남긴 태그들을 참조하여, 해당 유저가 본 영화들의 장르들을 카운트. 이후, 유저의 아이디를 입력 받아, 해당 유저가 많이 본 장르가 포함된 영화를 랜덤하게 찾아, 태그 및 기타 정보를 출력.

**2)Problem Modeling**

-Data Files

구분자를 “::”으로 하는 CSV파일

1. Movie.dat

Movie Index :: Movie Name :: Genre1 || Genre2 …

-Movie Index를 key값으로 하는 10681개의 레코드

-Movie Index를 기준으로 데이터가 정렬됨

1. Tags.dat

UserID :: Movie Index :: Tag :: TimeStamp

-Time Stamp를 key값으로 하는 95580개의 레코드

-User ID를 기준으로 데이터가 정렬됨

-Time Stamp를 제외한 모든 요소를 사용

사용할 구조체

Movie – Movie Index, 영화 명, 장르, 태그 포인터(Array) 관리

Tag – User ID, Movie Index, Tag내용 관리

User – User ID, 장르별 카운트 관리

1. **Problem Solving**

**1) Data Processing**

모든 데이터는 기본적으로, 파일을 읽으며, 매 레코드마다 상응하는 구조체를 생성했습니다. 또한, 이렇게 생성된 구조체들의 포인터를 저장하는 배열을 만들어 관리하였습니다. 결국 데이터는 movie 구조체의 포인터들을 관리하는 movie Block, Tag구조체의 포인터들을 관리하는 Tag Block, User 구조체의 포인터들을 관리하는 User Block으로 총 3종류의 포인터 배열을 사용하여 관리했습니다. 사용한 포인터 배열의 길이는 각각 사전에 파악한 레코드 개수를 고려하여 매크로에 각 Block에 대한 고정된 사이즈로 선언하였습니다. 이를 각각의 배열의 길이로 사용하였습니다.

데이터를 처리하는데 있어, 대부분의 요소는 파일 그대로 상응하게 할당할 수 있었지만, 구조체 간의 대응이 되는 요소들을 연결하기 위해선, 구조체 간의 관계와 데이터를 파악할 필요가 있었습니다.

먼저 각 구조체 생성을 위해 파일을 읽는 순서에 대한 부분을 말씀드리겠습니다.

Movie 구조체에서 tag포인터 배열을 제외한 모든 요소가 movie 파일을 통해 초기화 될 수 있습니다.

Tag 파일을 읽는 시점에는 Tag와 User의 구조체가 초기화 되어야 합니다. Tag의 경우, 모든 요소가 하나의 파일내 정보로 모든 요소가 초기화 될 수 있습니다. User구조체를 초기화 하기위해선, 장르별 카운팅이 필요했습니다. 하지만, Tag파일내에는 movie index정보 외엔 영화 구조체와 대응되는 값이 없었고, movie Index정보를 통해 해당 영화를 찾아내어, 장르를 파악해야 했습니다. 결국, movie의 장르정보를 파악하기 위해선, movie 구조체가 Tag 파일을 읽기전에 선행적으로 생성되어 이를 찾아내서 장르를 찾아야 했습니다.

또한, Movie구조체내 Tag 포인터의 경우 특정 영화 구조체를 조회할 때, 태그정보를 보여주기 위해 관리하였습니다. tag포인터를 movie 구조체에 할당하기 위해선, 선행적으로 생성된 movie 구조체의 포인터 배열에 매 레코드별로 생성되는 태그의 포인터를 할당해주는 방식으로 수행했습니다. 또한, 고정된 배열의 크기를 이용하다 보니, 구조체내에서 할당된 마지막 인덱스를 별도의 변수로 지정하여, 관리했습니다.

태그데이터를 읽을 때 사전에 생성된movie 구조체의 포인터를 초기화 하기 위해서 저희는 이진탐색을 활용했습니다. 레코드별 movie index를 통해 movie 구조체에 접근해야 하는데, 이때, index로 정렬되어 있던 데이터 특성을 활용해서, movie index를 통해 접근하는 복잡도를 현저하게 낮추었습니다. 즉, movie index를 얻은 이후에 movie구조체에 접근할 때는 log\_2(n+1)의 복잡도로 접근하여 초기화 하였습니다.

위와 같은 구조로 정리하면, 영화이름을 통해 태그를 검색할 때, 순차탐색을 통해 movie block 길이만큼의 복잡도로 상응하는 태그 포인터가 포함된 movie 구조체에 접근할 수 있었고, Tag검색의 경우, 순차탐색을 이용해 tag block 길이만큼의 복잡도로 movie index를 알아낼 수 있었습니다. Tag 검색에서 Movie index를 파악한 후에 movie 이름을 출력하기 위해서는 초기화 과정 중에 index를 통해 movie에 접근한 방식과 동일하게 이진탐색을 이용하여 접근했습니다.

다음은 추가기능에 필요한 User와 장르에 대한 내용을 말씀드리겠습니다.

우선 movie 데이터의 장르를 파악해본 결과, 다음과 같은 총 19종의 장르로 구성되어 있었습니다.

Adventure

Comedy

Action

Animation

Drama

Crime

Sci - Fi

Children

Documentary

Mystery

Thriller

Horror

Fantasy

Western

Film - Noir

Romance

War

Musical

IMAX

이에 따라서, User 구조체 내에 장르를 카운트할 때, 배열의 길이가 19인 정수형 배열을 이용하여 관리([0]-Adventure, [1]-Comedy, … , [18] IMAX) 하였습니다.

구체적으로 User에서 데이터를 저장하는 방식은 다음과 같습니다.

Tag 파일에서 매 레코드마다, User ID를 확인하고, Tag 데이터 특성상 User ID로 정렬된 점을 고려하여, User ID가 바뀌는 레코드마다 User 구조체를 생성하게 하였고, 레코드의 movie index를 통해 접근한 movie 구조체의 장르를 토큰화 하여, 상응하는 배열 인덱스의 카운트 수를 늘렸습니다.

**2) functions**

기본 기능

영화를 검색하여 그 영화에 달린 태그들을 보여주는 기능과 태그를 검색하여 그 태그가 달린 영화들을 보여주는 기능이 있다. 그리고 추가 기능에는 단어 일부분만 검색해도 영화의 전체 이름 혹은 태그의 전체 내용을 보여주는 기능과 User ID를 입력하면 그 User가 가장 많이 본 장르를 알아내서 그 장르의 영화들 중 랜덤으로 한 영화를 추천해주는 기능이 있다.

먼저, 영화에 달린 태그를 보여주는 기능은 사용자에게 입력 받은 영화를 buffer에 저장하여 strcmp로 영화 구조체 안에 있는 영화 이름이랑 비교하여 같으면 그 영화 구조체 안에 있는 영화 ID랑 태그 구조체 안에 있는 영화 ID랑 비교하여 같으면 그 태그 구조체 안에 있는 태그 내용을 보여준다.

다음으로, 태그가 달린 영화를 보여주는 기능은 사용자에게 입력 받은 태그를 buffer에 저장한 후 strcmp로 태그 구조체 안에 있는 태그 내용이랑 비교하여 같으면 그 태그 구조체 안에 있는 영화 ID값을 얻어낸다.

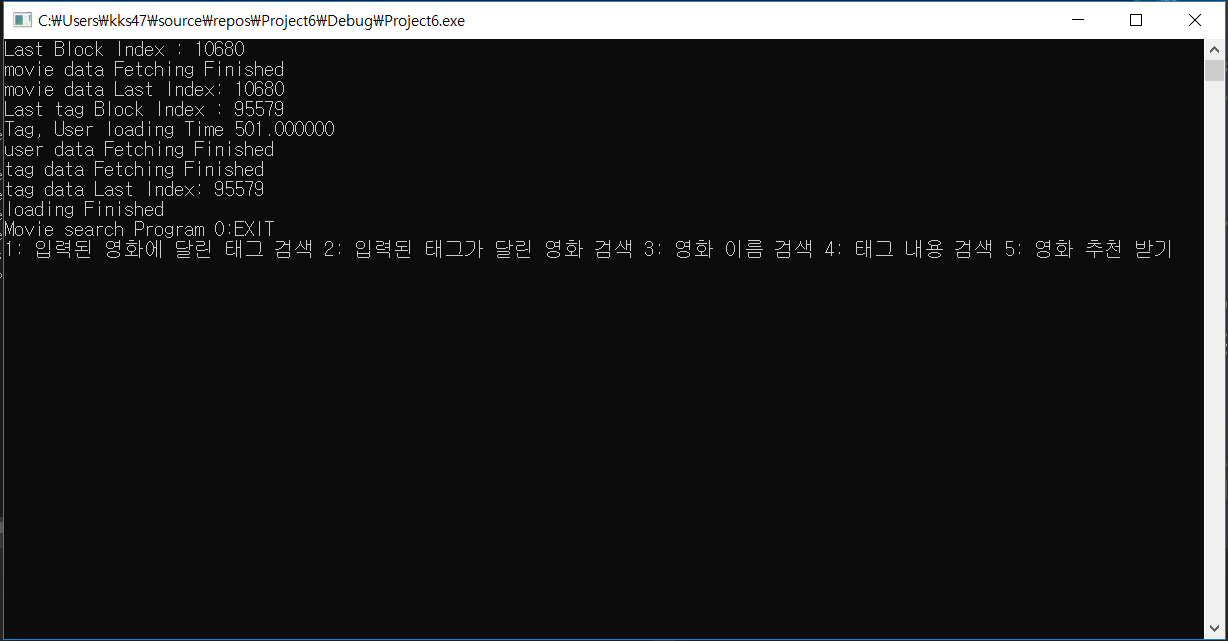
해당 ID값을 토대로, movie index를 통해 이진 탐색하는 함수를 활용하여 영화 구조체를 얻고, 구조체 안에 있는 영화 이름을 보여준다.

추가 기능 중 첫 번째로 단어 일부분만 검색해도 영화의 전체 이름을 보여주는 기능은 사용자에게 단어의 일부분을 입력 받아 그 단어를 buffer에 저장하고 strlen으로 buffer의 길이를 구한 후 그 길이를 n이라는 변수에 저장하여 strncmp로 영화 구조체 안에 있는 영화 이름과 buffer를 n만큼만 비교하여 같으면 그 영화 이름을 보여준다.

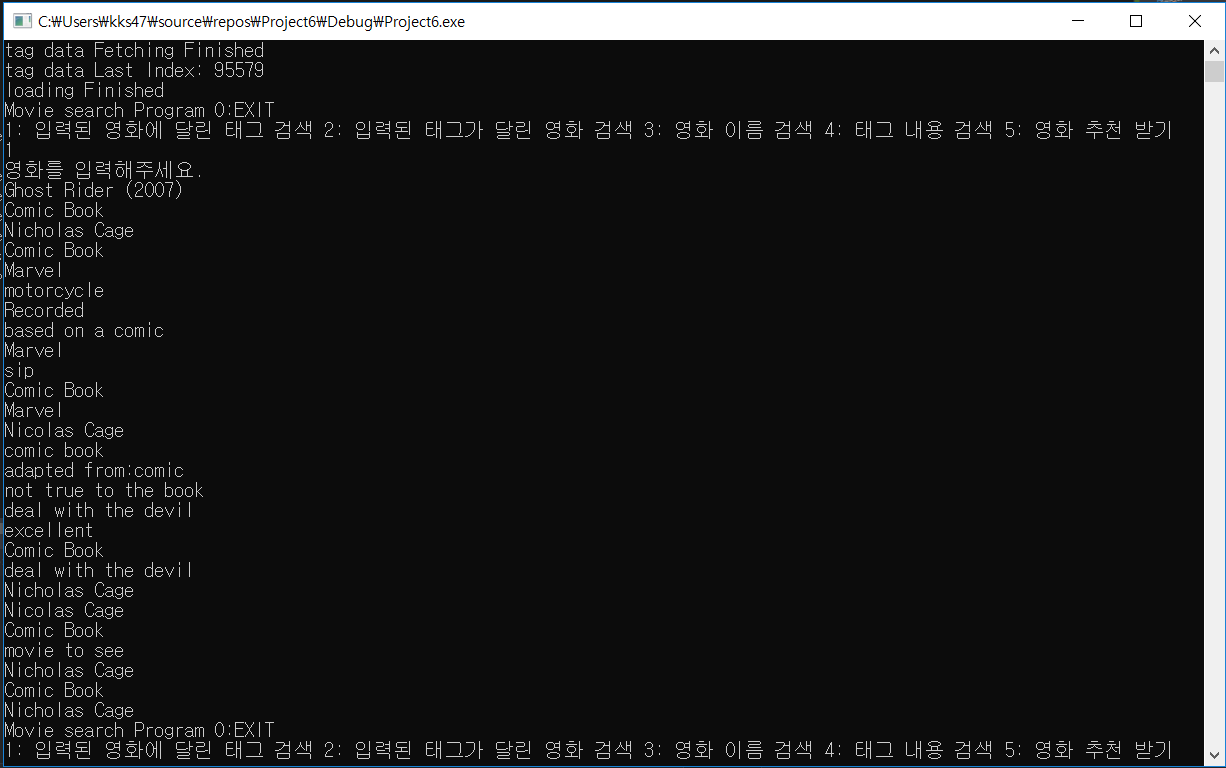
두 번째로 단어 일부분만 검색해도 태그의 전체 내용을 보여주는 기능은 사용자에게 단어의 일부분을 입력 받아 그 단어를 buffer에 저장하고 strlen으로 buffer의 길이를 구한 후 그 길이를 n이라는 변수에 저장하여 strncmp로 태그 구조체 안에 있는 태그 내용과 buffer를 n만큼만 비교하여 같으면 그 태그 내용을 보여준다.

마지막으로, User의 genre 선호도를 기준으로 추천해주는 기능은 사용자에게 user ID를 입력을 받고, 해당 ID를 user 구조체 배열에서 이진탐색으로, user 구조체를 찾아낸다. 찾아낸 user 구조체에서, genre count 배열에서 각각의 요소 중 가장 큰 값을 기준으로, 장르를 파악하고, 파악된 장르로 상응하는 문자열로 다시 변환해주고 이를 movie block에서 랜덤하게 배열에 접근하여 해당 장르가 포함된 영화를 찾아낸다. 찾아낸 영화 구조체를 기반으로, showAllMovieComponent(movie \* movie)라는 함수를 통해 해당 구조체 내부에 정보들을 출력하게 해준다.

**3. Implementation & Testing**

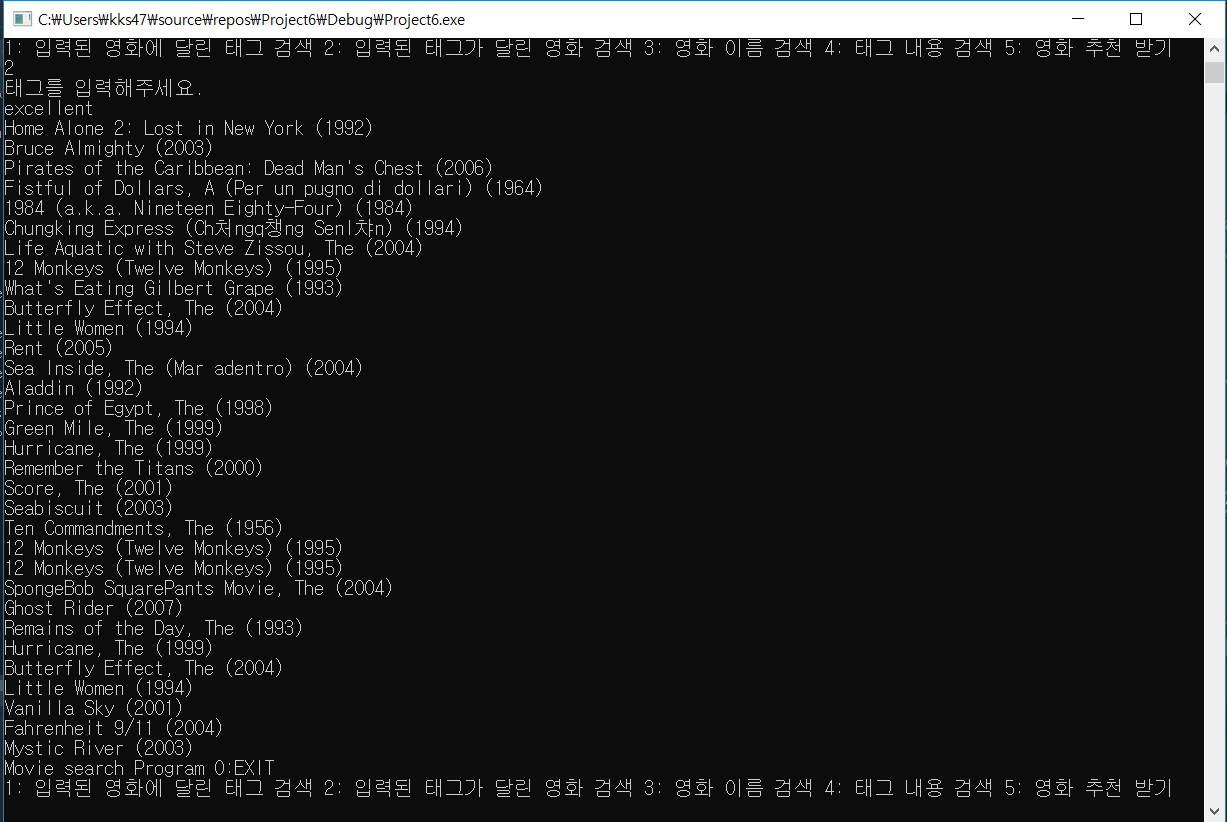
1 – 첫 실행화면

첫 실행하면 영화 개수와 태그 개수를 알려주고 실행할 기능을 선택할 수 있는 UI가 나온다.

2 – 입력된 영화에 달린 태그 검색

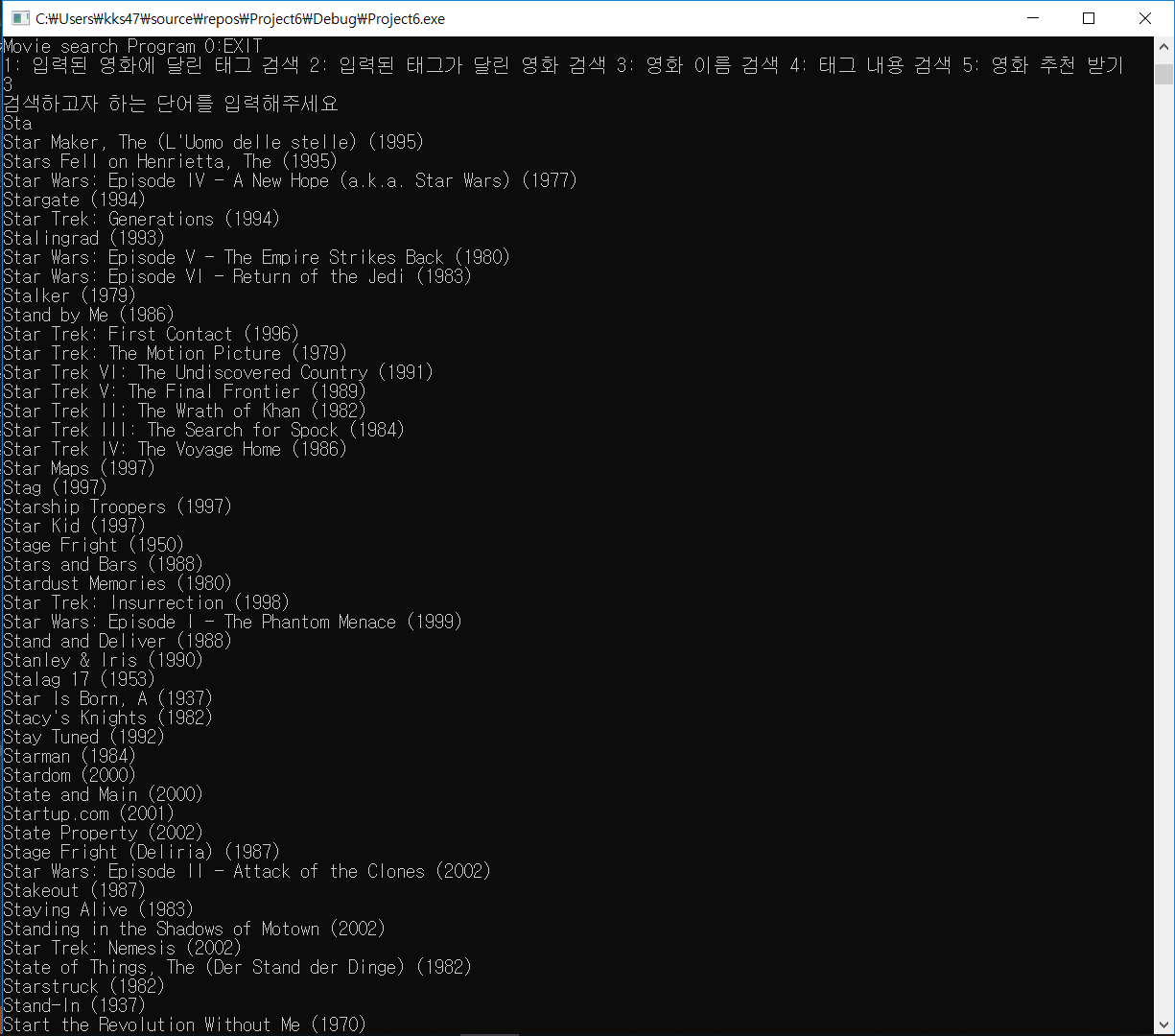
1번을 입력하면 영화를 입력해달라는 문구가 나오고 검색하고자 하는 영화를 입력하면 그 영화에 달린 태그들을 보여준다.

**3. Implementation & Testing**

3 – 입력된 태그가 달린 영화 검색

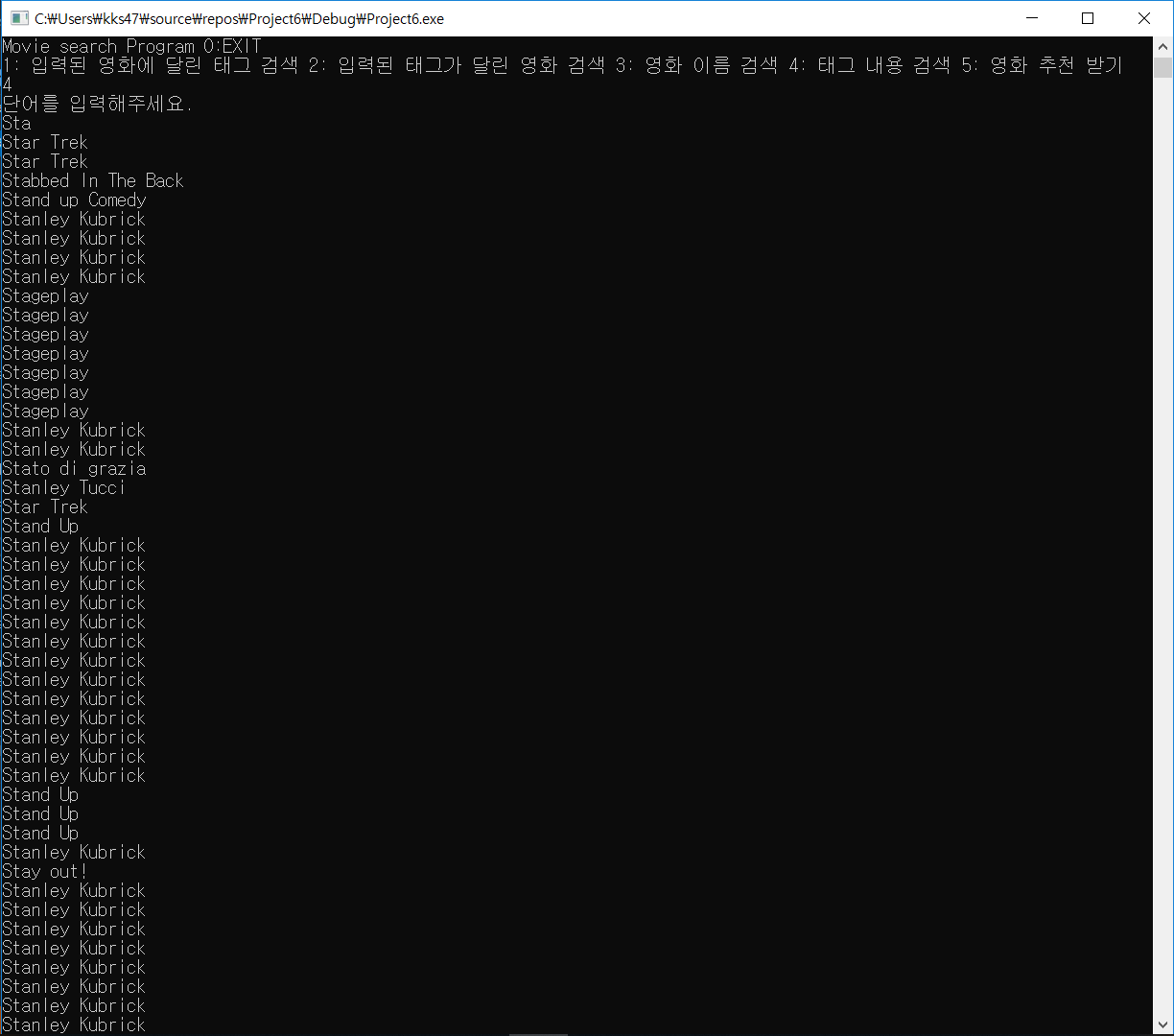
2번을 입력하면 태그를 입력해달라는 문구가 나오고 검색하고자 하는 태그를 입력하면 그 태그가 달린 영화들을 보여준다.

**3. Implementation & Testing**

4 – 영화 이름 검색

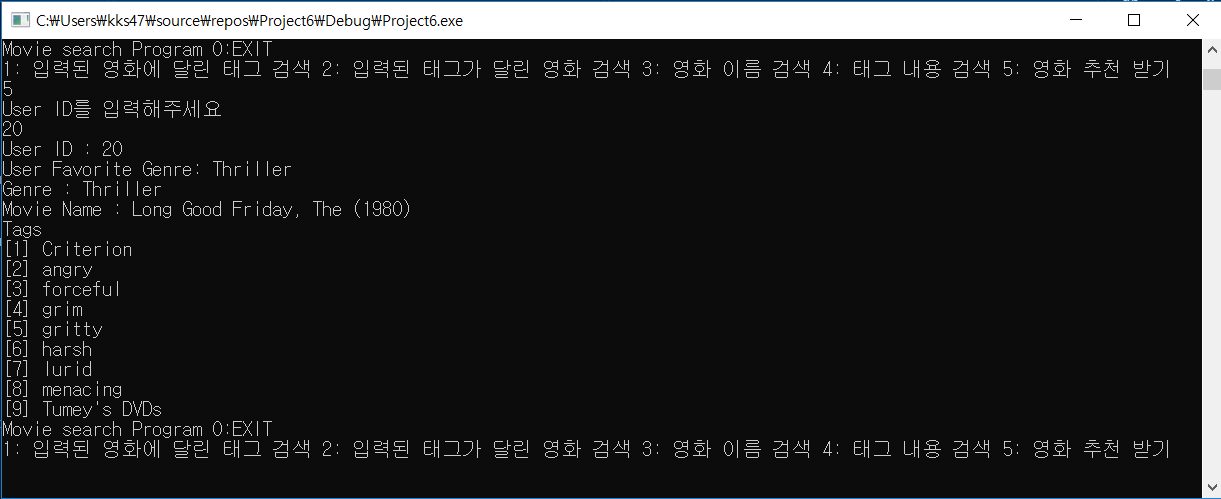
3번을 입력하면 단어를 입력해달라는 문구가 나오고 검색하고자 하는 단어를 입력하면 그 단어로 시작하는 영화들을 완전한 이름으로 보여준다.

**3. Implementation & Testing**

5 – 태그 내용 검색

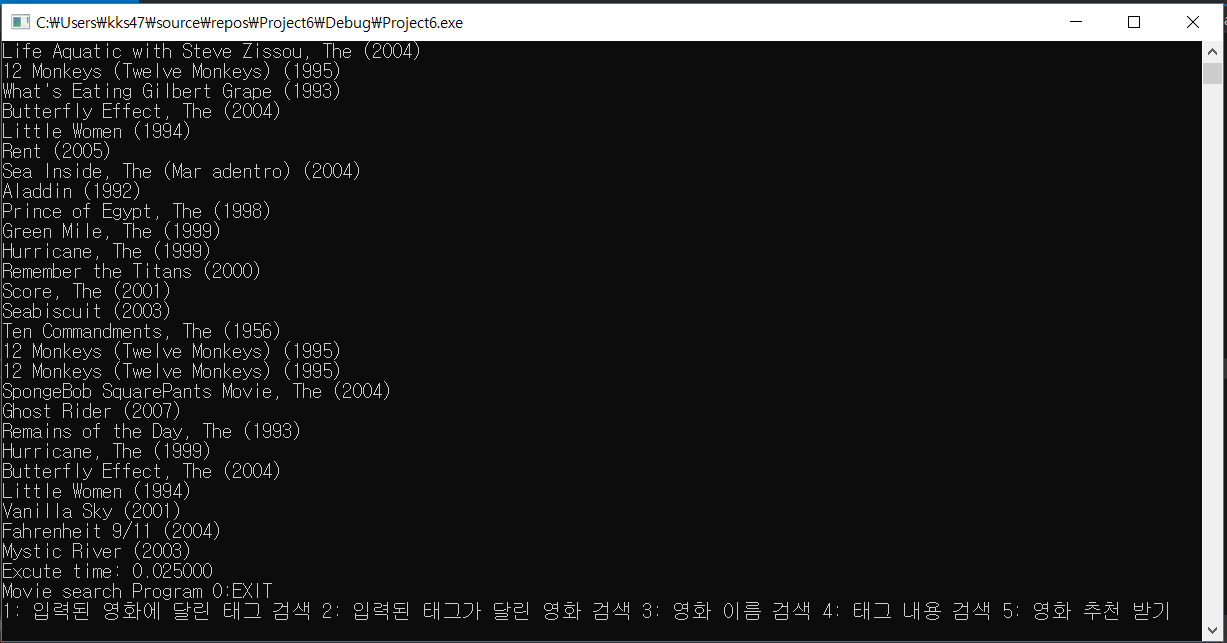
4번을 입력하면 단어를 입력해달라는 문구가 나오고 검색하고자 하는 단어를 입력하면 그 단어로 시작하는 태그들을 보여준다.

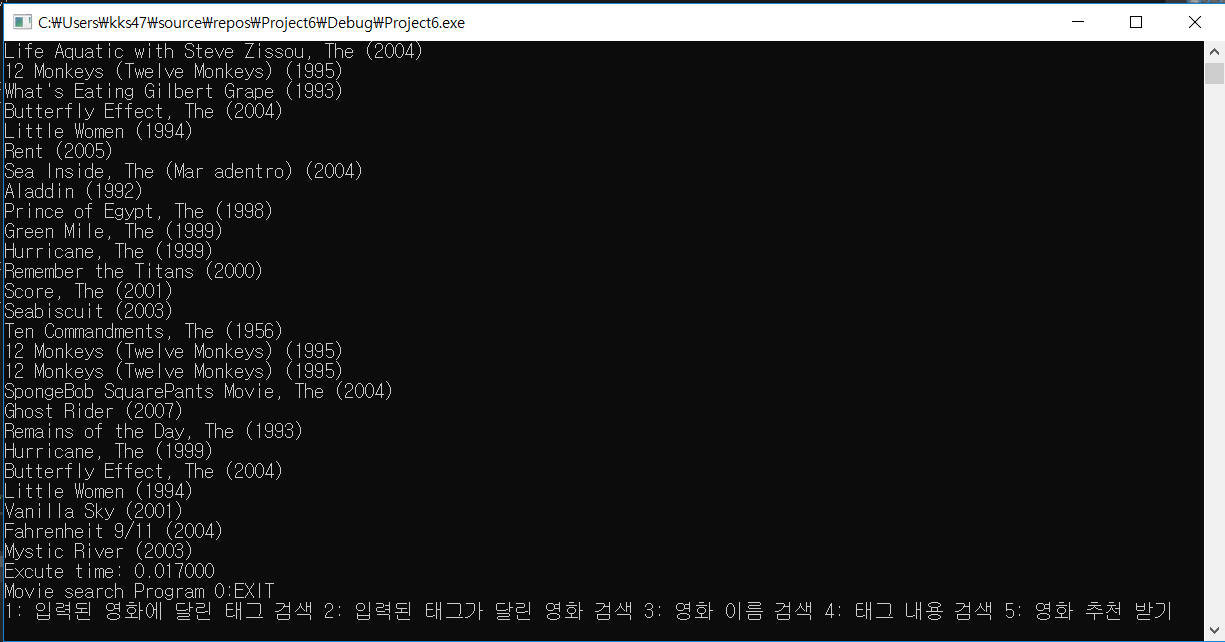
**3. Implementation & Testing**

6 – 영화 추천 받기

5번을 입력하면 User ID를 입력해달라는 문구가 나오고 검색하고자 하는 User ID를 입력하면 그 유저가 가장 많이 본 장르를 보여주고 그 장르의 영화들 중 랜덤으로 한 영화를 추천하여 그 영화의 장르 제목 달린 태그들을 보여준다.

**3. Implementation & Testing**

Testing

단순검색

이진탐색

단순 검색했을 때보다 이진탐색을 했을 때 더 빠른 것을 볼 수 있다.

**4. Evaluation**

**-세부적인 구현관련-**

초기에는 장르가 총 19개로 한정되어 있어 Enum객체를 통해 매크로를 활용하려고 했으나, 장르에서, 특수문자 ‘–‘가 들어있어서 처리가 되질 않아 임의의 정수 값을 지정해 사용하게 되어 코드 가독성이 떨어지게 되어 다소 아쉬웠습니다.

또한, 가변 길이 배열이 지원되지 않는 c언어 특성상, 데이터를 어느정도 파악하고 배열을 고정된 길이로 선언할 수밖에 없어서 이부분이 다소 아쉬웠습니다. 만약 이 부분이 해소가 된다면 더욱 Robust한 프로그램이 될 수 있을 것 같습니다.

데이터의 특성을 파악하고 접근하여 별도의 정렬이나 파일 조작 없이 초기화와 런타임 효율성을 최대화하기 위해 노력했고 만족스러운 결과를 얻은 것 같습니다.

**-느낀 점-**

초기에 프로그램 구상에 거의 대부분의 시간을 소요하였습니다. 대략 전체 시간의 70프로 정도를 프로그램 구상과 의견 개진에 사용하였고, 이로 인해 구현에는 아주 적은 소요시간이 걸리게 되어 매우 만족스러웠습니다. 또한 조원 모두가 csv데이터를 제대로 다뤄 본적이 없어서 단순하게 진행하면 될 줄 알았지만, 데이터를 파악하기 위해 고민이 많이 필요함을 느꼈습니다.