HW6_202012468(pdf 파일로 설명) 202012468 김현서

주제 : 베스트 앨범

1. 문제에 대한 자신의 알고리즘 설명

**베스트앨범**

**문제 설명**

스트리밍 사이트에서 장르 별로 가장 많이 재생된 노래를 두 개씩 모아 베스트 앨범을 출시하려 합니다. 노래는 고유 번호로 구분하며, 노래를 수록하는 기준은 다음과 같습니다.

1. 속한 노래가 많이 재생된 장르를 먼저 수록합니다.
2. 장르 내에서 많이 재생된 노래를 먼저 수록합니다.
3. 장르 내에서 재생 횟수가 같은 노래 중에서는 고유 번호가 낮은 노래를 먼저 수록합니다.

노래의 장르를 나타내는 문자열 배열 genres와 노래별 재생 횟수를 나타내는 정수 배열 plays가 주어질 때, 베스트 앨범에 들어갈 노래의 고유 번호를 순서대로 return 하도록 solution 함수를 완성하세요.

**제한사항**

- genres[i]는 고유번호가 i인 노래의 장르입니다.
- plays[i]는 고유번호가 i인 노래가 재생된 횟수입니다.
- genres와 plays의 길이는 같으며, 이는 1 이상 10,000 이하입니다.
- 장르 종류는 100개 미만입니다.
- 장르에 속한 곡이 하나라면, 하나의 곡만 선택합니다.
- 모든 장르는 재생된 횟수가 다릅니다.

**입출력 예**

| genres | plays | return |
|---|---|---|
| ["classic", "pop", "classic", "classic", "pop"] | [500, 600, 150, 800, 2500] | [4, 1, 3, 0] |

**입출력 예 설명**

classic 장르는 1,450회 재생되었으며, classic 노래는 다음과 같습니다.

- 고유 번호 3: 800회 재생
- 고유 번호 0: 500회 재생
- 고유 번호 2: 150회 재생

pop 장르는 3,100회 재생되었으며, pop 노래는 다음과 같습니다.

- 고유 번호 4: 2,500회 재생
- 고유 번호 1: 600회 재생

따라서 pop 장르의 [4, 1]번 노래를 먼저, classic 장르의 [3, 0]번 노래를 그다음에 수록합니다.

※ 공지 - 2019년 2월 28일 테스트케이스가 추가되었습니다.

우선, 문제를 보자마자 구조체를 사용해 문제를 풀어야겠다는 생각이 들었습니다.

```c
struct MusicList {  // musiclist : 장르, play 횟수, index(고유번호)
    char genre[30];
    int play;
    int index;
};

struct Album {  // 베스트 앨범에 들어갈 곡 선정 전 앨범 리스트 : musiclist, 장르, 총 play 횟수, 수록곡 개수
    struct MusicList musiclist[100];
    char genre[30];
    int play_sum;
    int music_num;
};
```

그래서 이와 같은 구조체 2개를 만들었습니다.

Album 안에 들어갈 요소는 musiclist(음악 하나의 장르, play 횟수, index(고유번호)), 장르, 장르별 총 play 횟수, 장르별 수록곡 개수입니다.

```c
int main() {
    char genres[10001] = { 0, };    // 조건 : genres와 plays의 길이는 같으며 이는 1 이상 10000 이하입니다.
    char plays[10001] = { 0, }; // 조건 : genres와 plays의 길이는 같으며 이는 1 이상 10000 이하입니다.
    int num = 0, cnt = 0;

    printf("Genres : ");
    fflush(stdin);  // stdin 비워주기
    fgets(genres, 10001, stdin);    // genres에 입력
    genres[strlen(genres) - 1] = '₩0';  // ₩n 제거 위함

    printf("Plays : ");
    fflush(stdin);  // stdin 비워주기
    fgets(plays, 10001, stdin);
    plays[strlen(plays) - 1] = '₩0';    // ₩n 제거 위함

    printf("%s₩n", genres);
    printf("%s₩n", plays);

    printf("%d %d₩n", strlen(genres), strlen(plays));
    int GenLen = strlen(genres);
    int PlayLen = strlen(plays);
```

```
Microsoft Visual Studio 디버그 콘솔
Genres : ["dance","ballad","jazz","pop","ballad","kpop","hiphop","dance","pop","pop","kpop","ballad","classic","pop","ja
zz","pop","dance"]
Plays : [21,567,3,2,5,6345,345,2,134,10,273,20,374,2947,99,18,28374]
["dance","ballad","jazz","pop","ballad","kpop","hiphop","dance","pop","pop","kpop","ballad","classic","pop","jazz","pop"
,"dance"]
[21,567,3,2,5,6345,345,2,134,10,273,20,374,2947,99,18,28374]
129 60
dance
```

우선 Genres와 Plays를 각각 배열로 입력 받고, 제대로 입력이 되었는지 확인해보는 과정을 겪었습니다.

```
struct MusicList list[10001] = { {{0},0,0} };   // 초기화

for (int i = 0; i < GenLen; i++) {
    char temp_genres[30] = { 0 };
    int j = 0;

    if (genres[i] == 34) {  // "(큰따옴표) 아스키 코드 = 34
        i++;
        while (genres[i] != 34) {   // 닫는 큰 따옴표가 나오기 전까지
            list[cnt].genre[j] = genres[i]; // list의 genre에 genres 저장
            i++;
            j++;
        }
        cnt++;
    }
}
int COUNT = cnt;

for (int i = 0; i < PlayLen; i++) {
    if (plays[i] >= 48 && plays[i] <= 57) { // ASCII code 48 = 10진수 '0' ~ ASCII code 57 = 10진수 '9'
        while (plays[i] >= 48 && plays[i] <= 57) {  // ASCII code 48 = 10진수 '0' ~ ASCII code 57 = 10진수 '9'
            list[num].play = list[num].play * 10 + plays[i] - 48;
            // 한 자리수씩 올라가기 때문에 (이전 수 *10), 48을 빼주는 이유는 아스키 코드에서 정수로 바꾸기 위함
            i++;
        }
        list[num].index = num;
        num++;
    }
}
int NUMBER = num;

if (COUNT != NUMBER) {
    printf("genres와 plays의 개수가 다릅니다.");
    exit(-1);
}

for (int k = 0; k < COUNT; k++)
    printf("%s₩n", list[k].genre);

for (int k = 0; k < COUNT; k++)
    printf("%d₩n", list[k].play);

for (int k = 0; k < COUNT; k++)
    printf("%d₩n", list[k].index);

printf("-------₩n");
for (int k = 0; k < COUNT; k++)
    printf("%s, %d, %d₩n", list[k].genre, list[k].play, list[k].index);
printf("-------₩n");
```

다음으로는 MusicList 구조체 list를 이차원배열로 불러오고, 입력받았던 genres를 musiclist 안에 각각 하나씩 넣고, plays에 입력 받았던 것도 musiclist 안에 각각 하나씩 넣으면서, index로 고유 번호까지 표시해줬습니다.

만약 Genres와 Plays의 개수가 다르다면 잘못 입력했음을 알려주는 식도 넣었습니다.

그리고, MusicList 구조체 안에 제대로 들어갔는지 확인하기 위해서 print문들도 넣어줬습니다.

: 결과

```
struct Album album[100] = { {{0},0,0}, {0}, 0, 0 };   // genre 종류 100개 이하
int count = 0;  // album에 들어있는 장르의 개수 0으로 초기화

for (int k = 0; k < COUNT; k++) {
    int result = 1, n = 0, GenreLen = strlen(list[k].genre), N = 0;

    for (n = 0; n < count + 1; n++) {
        for (int i = 0; i < GenreLen; i++) {
            if (album[n].genre[i] != list[k].genre[i]) {
                break;  // 한 자라도 어긋나면 바로
            }

            else if (album[n].genre[i] == list[k].genre[i]) {
                result = 0;
            }
            N = n;  // 모두 일치하는 경우 N = n
        }
    }

    if (result == 1) {  // 같은 genre가 없는 경우
        for (int i = 0; i < GenreLen; i++) {
            album[count].genre[i] = list[k].genre[i];   // album genre로 넣기
            album[count].musiclist[0].genre[i] = list[k].genre[i];  // music list 안의 genre로 넣기
        }
        album[count].play_sum = list[k].play;   // 같은 장르의 총 play 횟수
        album[count].musiclist[0].play = list[k].play;  // music list 안의 play 횟수로 넣기
        album[count].musiclist[0].index = list[k].index;    // music list 안의 index 번호
        album[count].music_num = 1;     // 같은 장르의 총 music 개수
        count++;    // album에 들어있는 장르의 개수
    }

    else if (result == 0) { // 같은 genre가 있는 경우
        int temp_num = album[N].music_num;  // 같은 장르의 노래 개수
        for (int i = 0; i < GenreLen; i++)
            album[N].musiclist[temp_num].genre[i] = list[k].genre[i];   // 같은 장르의 music list에 k번째 곡 장르명 추가하기

        album[N].play_sum += list[k].play;  // 같은 장르의 총 play 횟수
        album[N].musiclist[temp_num].play = list[k].play;   // 같은 장르의 music list에 k번째 곡 play 횟수 추가하기
        album[N].musiclist[temp_num].index = list[k].index; // 같은 장르의 music list에 k번째 곡 index 번호 추가하기
        album[N].music_num += 1;    // 같은 장르의 곡 수 1개 증가
    }
}
```

다음은 album 안에 장르별 곡들을 넣는 과정입니다. Album 구조체 album을 선언하고 같은 장르별로 총 play횟수와 장르별 곡 개수, 곡별 장르이름, play횟수, index번호를 입력했습니다.

```
for (int j = 0; j < count; j++) {
    printf("%s₩n", album[j].genre);
    printf("%d₩n", album[j].play_sum);
    for (int p = 0; p < album[j].music_num; p++) {
        printf("%s, %d, %d₩n", album[j].musiclist[p].genre, album[j].musiclist[p].play, album[j].musiclist[p].index);
    }
    printf("-₩n");
}
```

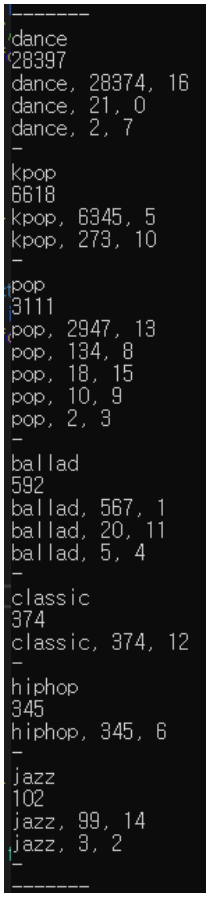
```
dance
28397
dance, 21, 0
dance, 2, 7
dance, 28374, 16
-
ballad
592
ballad, 567, 1
ballad, 5, 4
ballad, 20, 11
-
jazz
102
jazz, 3, 2
jazz, 99, 14
-
pop
3111
pop, 2, 3
pop, 134, 8
pop, 10, 9
pop, 2947, 13
pop, 18, 15
-
kpop
6618
kpop, 6345, 5
kpop, 273, 10
-
hiphop
345
hiphop, 345, 6
-
classic
374
classic, 374, 12
-
```

다음은 album이 장르별로 제대로 나눠졌는지 확인하기 위한 printf 과정입니다.

```
struct Album temp = { {{0},0,0} , { 0 }, 0, 0 };
for (int j = 0; j < count + 1; j++) {
    // 버블정렬로 plays_sum 기준으로 Album 장르별 내림차순 정렬
    for (int p = 0; p < count - j; p++) {
        if (album[p].play_sum < album[p + 1].play_sum) {
            // p번째 장르의 총 play 횟수가 p+1번째 총 play 횟수보다 적다면 순서 바꾸기
            temp = album[p];
            album[p] = album[p + 1];
            album[p + 1] = temp;
        }
    }
}

struct MusicList temp_musiclist = { {0},0,0 };
for (int j = 0; j < count + 1; j++) {
    for (int i = 0; i < album[j].music_num + 1; i++) {
        // 버블정렬로 plays 기준으로 Album 내의 각 곡 인덱스별 내림차순 정렬
        for (int p = 0; p < album[j].music_num - i; p++) {
            if (album[j].musiclist[p].play < album[j].musiclist[p + 1].play) {
                // j장르 내의 p번째 곡의 play 횟수가 p+1번째 곡의 play 횟수보다 적다면 두 곡 순서 바꾸기
                temp_musiclist = album[j].musiclist[p];
                album[j].musiclist[p] = album[j].musiclist[p + 1];
                album[j].musiclist[p + 1] = temp_musiclist;
            }

            if (album[j].musiclist[p].play == album[j].musiclist[p + 1].play) {
                // j 장르 내의 두 곡 순서가 같다면 index가 작은 순으로 정렬하기
                if (album[j].musiclist[p].index > album[j].musiclist[p + 1].index) {
                    temp_musiclist = album[j].musiclist[p];
                    album[j].musiclist[p] = album[j].musiclist[p + 1];
                    album[j].musiclist[p + 1] = temp_musiclist;
                }
            }
        }
    }
}
```

다음은 album을 장르별로 play_sum(총 재생 횟수)의 내림차순으로 정렬한 후, 장르별로 곡의 plays를 내림차순으로 정렬한 것입니다. 만약 곡별로 plays(재생횟수)가 같다면 index가 작은 순으로 정렬합니다.

```
printf("-------₩n");

for (int j = 0; j < count; j++) {
    printf("%s₩n", album[j].genre);
    printf("%d₩n", album[j].play_sum);
    for (int p = 0; p < album[j].music_num; p++) {
        printf("%s, %d, %d₩n", album[j].musiclist[p].genre, album[j].musiclist[p].play, album[j].musiclist[p].index);
    }
    printf("-₩n");
}

printf("-------₩n");
```



장르별로 제대로 정렬이 되었는지 확인하는 과정입니다.

```
printf("[");
for (int j = 0; j < count; j++) {
    if (album[j].music_num == 1)    // 한 장르 내의 곡이 1개일 때
        printf("%d,", album[j].musiclist[0].index);
    if (album[j].music_num >= 2) {  // 한 장르 내의 곡이 2개 이상일 때
        printf("%d,", album[j].musiclist[0].index);
        printf("%d,", album[j].musiclist[1].index);
    }
}
printf("₩b]₩n");    // ₩b는 마지막 반점 하나 지워주기 위함
```

마지막으로, 정렬해 놓은 앨범의 순서에 따라, 총 가장 많이 재생된 장르의 곡 2개, 다음으로 많이 재생된 장르의 곡 2개 … 순서로 출력합니다. 단, 만약 장르 내에 곡이 하나밖에 없는 경우는 하나만 출력합니다.