```c
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <string.h>

#include "assignment2.h"

// typedef struct {
//  char* title;
//  char* artist;
//  int year;
// } song;
int add_song(const char* file_name, song s) {
 FILE *file_s;
 file_s = fopen(file_name, "a+");

 song *existed_song = find_song(file_name, s.title);
 if (existed_song)
 {
  return 0;
 }
 int title_l = strlen(s.title);
 int artist_l = strlen(s.artist);

 fprintf(file_s, "%d %d %d %s %s\n", title_l, artist_l, s.year, s.title, s.artist);
 fclose(file_s);
 return 1;
}

song* find_song(const char* file_name, const char* title) {
 FILE *file_song;

 file_song = fopen(file_name, "r");
```

```
if (!file_song)
{
 return NULL;
}

char* song_title;
char* song_artist;
int song_year;
int title_len;
int artist_len;
bool check = false;
song* check_s = NULL;

while (!check)
{
 if(fscanf(file_song, "%d %d %d", &title_len, &artist_len, &song_year) == EOF)
 {
  check = true;
 }
 else
 {
  song_title = (char)malloc((title_len+1)sizeof(char));
  song_artist = (char)malloc((artist_len+1)sizeof(char));

  fgetc(file_song); //this is for skipping the space
  for (int i =0; i<title_len; i++)
  {
   song_title[i] = (char)fgetc(file_song);
  }
  song_title[title_len] = '\0';
  fgetc(file_song); //and then skipping the space again
  for (int j = 0; j<artist_len; j++)
  {
```

```
    song_artist[j] = (char)fgetc(file_song);
   }
   song_artist[artist_len] = '\0';
   fgetc(file_song); //this is for skipping \n

   if (strcmp(song_title, title) == 0)
   {
    check_s = (song*)malloc(sizeof(song)); //make the check_s and then store the information, and return at the last.
    check_s -> title = song_title;
    check_s -> artist = song_artist;
    check_s -> year = song_year;
    check = true;
   }
   else
   {
    free(song_title);  // since I used malloc above, I need to free the data.
    free(song_artist);
   }
  }
 }
 fclose(file_song);
 return check_s;
}


// using iteration
// unsigned long fib3(unsigned int n) {
//  // implement me
//  long *fib = (long*)malloc((n+1)*sizeof(long));
//  if(fib==NULL)
//  {
//   return -1;
//  }
//  fib[0] = 0;
```

```
//  fib[1] = 1;
//  fib[2] = 2;
//  for(int i = 3; i < n; i++)
//  {
//   fib[i] = fib[i-1] + fib[i-2];
//  }
//  return fib[n];
// }

// using recursion
unsigned long fib3(unsigned int n) {
 if(n==0)
 {
  return 0;
 }
 if(n==1)
 {
  return 1;
 }
 if(n==2)
 {
  return 2;
 }
 return fib3(n-1) + fib3(n-2) + fib3(n-3);
}

int linear_search_rec_first(int* ar, int length, int number) {
 int index;
 if(ar[0] == number)
 {
  return 0;
 }
 if(length == 1)
 {
```

```
 if(ar[0] == number)
 {
  return 0;
 }
 else
 {
  return -1;
 }
}
else
{
 index = linear_search_rec_first(ar+1, length-1, number);
 if(index != -1)
 {
  return index + 1;
 }
 else
 {
  return -1;
 }
}
}

int linear_search_rec_last(int* ar, int length, int number) {
 while(length>0)
 {
 if(ar[length] == number)
 {
  return length;
 }
 else
 {
  return linear_search_rec_last(ar, length-1, number);
 }
```

```
 }
 return -1;
}



int count_tokens(const char* str, char delim) {
 int i = 1;
 int count;

 if(str[0] == delim)
 {
  count = 0;
 }
 else
 {
  count = 1;
 }

 while(1)
 {
  if(str[i] == '\0')
  {
   break;
  }
  else if(str[i-1] == delim && str[i] != delim)
  {
   count++;
  }
  i++;
 }
 return count;
}
```

```
char* get_tokens(const char str, char delim) {
 int count = count_tokens(str, delim);
 if (count==0)
  return NULL;

 char* ret = (char**) malloc(sizeof(char)*count);
 if (ret==NULL)
 {
  return NULL;
 }

 // pointer to the beginning and end of a token
 const char* token_start = str;
 const char* token_end;
 // length of a token (equal to token_end-token_start)
 int len_str_i;

 // create the i'th token
 for (int i=0; i<count; i++) {

  // find the beginning of the first token
  while (*token_start==delim)
   token_start++;

  // find end of token
  token_end = token_start+1;
  while (*token_end != 0 && *token_end!=delim)
   token_end++;

  len_str_i = token_end-token_start;
  ret[i] = (char) malloc((len_str_i+1)sizeof(char));
  strncpy(ret[i], token_start, len_str_i);

  token_start = token_end+1;
```

```
    }


    return ret;
}
```