Repeat coursework: build a playlist

SOFT 7019 C Programming
due: 13th August 2021

Introduction
This assignment requires you to develop a program where a user can create a playlist of individual songs. You will implement a text based menu, this will allow users to edit the playlist. To gain full marks in this assignment you will need to implement the playlist using a linked list; an array-based implementation means that marks gained will scaled downwards by 30% so the overall grade will be out of a maximum of 70%.

Requirements
Each *Song* in the playlist should have the following attributes:

- the name of the song
- the name of the artist
- the duration of the song (in seconds)

The *Playlist* will contain *Songs* in an order imposed by the user. This can be implemented using a singly linked list for full marks or an array for a lower grade. Each position in the playlist should have the following attributes:

- the name of the playlist
- a pointer to a song
- a pointer to the *next* position in the playlist (if using a linked list implementation)

The text based interface allows the user to interact with the playlist. The following options should make up your text based interface:

1. Create a new *Playlist* - the user should have the option to give the playlist a name
2. Add a *Song* to the end of the *Playlist* - the user should be prompted to provide the details of the new song
3. Play the *Playlist* - print out each song in the playlist in order

4. Search for a *Song* based on the title or the artist's name - prompt the user to input a string, if this string matches, or is a sub-string of, either the title or artist of a song then return that song (these criteria could return multiple songs)
5. Delete a *Song* from the *Playlist* - the user should be prompted to select the the index of the song to be removed
6. Swap the position of two *Songs* - the user should be prompted to select the indexes of two songs which will swap positions in the list
7. Play a random *Song* from the list - use random function in C to randomly print out one of the songs
8. Shuffle the order of the *Playlist* - reorder the songs of the playlist with some degree of randomness and print out the new order (see note below on shuffling)
9. Calculate the duration of the *Playlist* - print the total cumulative time of all songs in the playlist
10. Save the complete *Playlist* to a file
11. Load a complete *Playlist* from a file

Rough grading criteria
General implementation (10/100)

- Playlist implementation
  - linked list (graded out of 100%)
  - array (grade scaled by 70%)
- Struct to hold each song 5
- Interactive text based menu 5

Basic interface options (45/100)

- Create a new playlist 5
- Add a song to the end of the playlist 5
- Delete a song from the playlist 10
- Swap the position of two songs 10
- Play the playlist 5

- Play a random song from the playlist 10

Advanced interface options (45/100)

- Save the complete playlist to a file 5
- Load the complete playlist from a file 5
- Search for a song 10
  - return all exact matches for title or artist 5/10
  - return all exact matches and any sub-string matches for title or artist 10/10
- Shuffle the order of the playlist 20
  - shuffle in place 20/20
  - shuffle with a copy 10/20
- Calculate the duration of the playlist 5

Random numbers in C
Use the random function from stdlib.h:

```
int random(void);
```

This function returns an integer between 0 and a large constant, RAND_MAX. If you need to generate a bounded random number between 0 and N you can use the modulo (%) operator as follows:

```
int rand_n;
rand_n = random() % N;
```

The modulo operator (A%B) divides A by B and returns the remainder, which will always be strictly smaller than B.

Seeding the random number generator
Random numbers are generated using a Pseudo Random Number Generator (PRNG). The PRNG uses an algorithm to generate a long list of seemingly random numbers, based on an initial value known as the seed. In C the seed of the PRNG is set with the srandom function:

```
void srandom(unsigned int seed);
```

For example,

```
srandom(1);
```

The PRNG seed should be set before generating numbers, preferably at the start of the code. It is important to note that if a program that relies on random numbers uses the same seed, its behaviour will be the same because the PRNG will generate the same list of pseudo random numbers. This can be useful for validation purposes.

Shuffling a list

Many algorithms of various complexity have been developed for shuffling lists. They can range from a very simple switching between two elements in the list, to using cryptographic algorithms. The following is an example of an algorithm that shuffles all elements in a list:

```
Input = L (the list); Output = S (the shuffled list)
N = length(L)
while N > 1
    R = random number between 0 and N
    append L[R] to S
    remove L[R] from L
    N = length(L)
append L[0] to S
```

At each step:

- an element is taken from the list at random, by generating a random index
- the element is appended to the destination list
- the element is then removed from the original list.

This algorithm requires you to create space in memory for a completely new list; elements from the original list are copied into the new list in a random order. This means that the algorithm requires double the space of the original list, this could be a problem if our list was very long or we were on a device with low memory resources. Other list shuffling algorithms shuffle the elements in-place, meaning that the elements in the original list are shuffled without the need for any copies. In this assignment shuffling the elements in-place will gain you more marks, but you must ensure that the algorithm generates a randomly ordered list each time.

Honor code

- it is ok to exchange ideas with colleagues
- it is **not** ok to share a solution or parts of a solution with colleagues
- it is **not** ok to copy solutions from other sources (e.g. Internet)
- we use automated plagiarism detection software to find indications of plagiarism

- plagiarism consequences range from zero marks on sections or the entire assignment to higher level investigation
- when several parties share a solution (or parts), all will be penalised, regardless of who was the source and who copied.

Late policy
This assignment is due the 13th of August 2021, the usual late penalties will apply if coursework is submitted after that date. If it is less than 7 days late a 10% deduction will be applied to marks gained, if it is less than 14 days late a 20% deduction will be applied, after that the submission window will close and you will receive no marks.