**ITU COMPUTER ENGINEERING DEPARTMENT**

**BLG 233E DATA STRUCTURES**

**HOMEWORK -2**

**Due Date : 23th of November, 2016.**

In this assignment, you are required to build up an extension of the linked list data structure, combining multi-lists, doubly linked lists and circular list structures. A diagram of the structure is given below. You are required to write this application **by yourself.**

**Structure:**

Head

nodes

**ABCD**

Slow

**CDAB**

Jazz

**BCDA**

Slow

**DABC**

Jazz

**ABCD**

Slow

**ABCD**

Slow

**BCDA**

Slow

**CDAB**

Jazz

**DABC**

Jazz

**BCDA**

Slow

**BCDA**

Slow

**CDAB**

Jazz

**ABCD**

Slow

**DABC**

Jazz

**#define**SNAME\_LENGTH **50**  
**#define**LNAME\_LENGTH **50**  
**#define**STYLE\_LENGTH **30**  
**#define**SINGER\_LENGTH **50**

**struct** song {

**char\*** name   = **new** **char[**SNAME\_LENGTH**]**;  
 **char\*** singer = **new** **char[**SINGER\_LENGTH**]**;  
 **char\*** style  = **new** **char[**STYLE\_LENGTH**]**;   
 **int** year;

};

**struct** song\_node {

**song\*** data;

**song\_node\*** next;  
 **song\_node\*** prev;

**playlist\*** parent;

};

**struct** playlist {

**int**songnumber;  
 **char\*** name = **new char[**LNAME\_LENGTH**]**;

**song\_node\*** head;

**playlist\*** next;  
 **playlist\*** prev;

};

//add file definitions and functions as needed.

**struct** list {

**int** playlist\_count;

**playlist\*** head;

};

Your assignment is to simulate a music player with a sequence of playlists. The user will be allowed to create new playlists, add new songs to any previously created playlists, play songs, and navigate (both forward and backward) through playlists**.**

By default, three playlists (chronological, sorted and random)will be **created at the beginning of the program**. The chronological playlist will be **sorted by year**, while the sorted playlist will be **sorted first by singer name, then by song name**. The random playlist is a shuffle playlist and should be **reshuffled at each request**. It will also be possible to **create other custom playlists** (e.g. the **slow** playlist in the example) on demand.

**Program Work Flow**

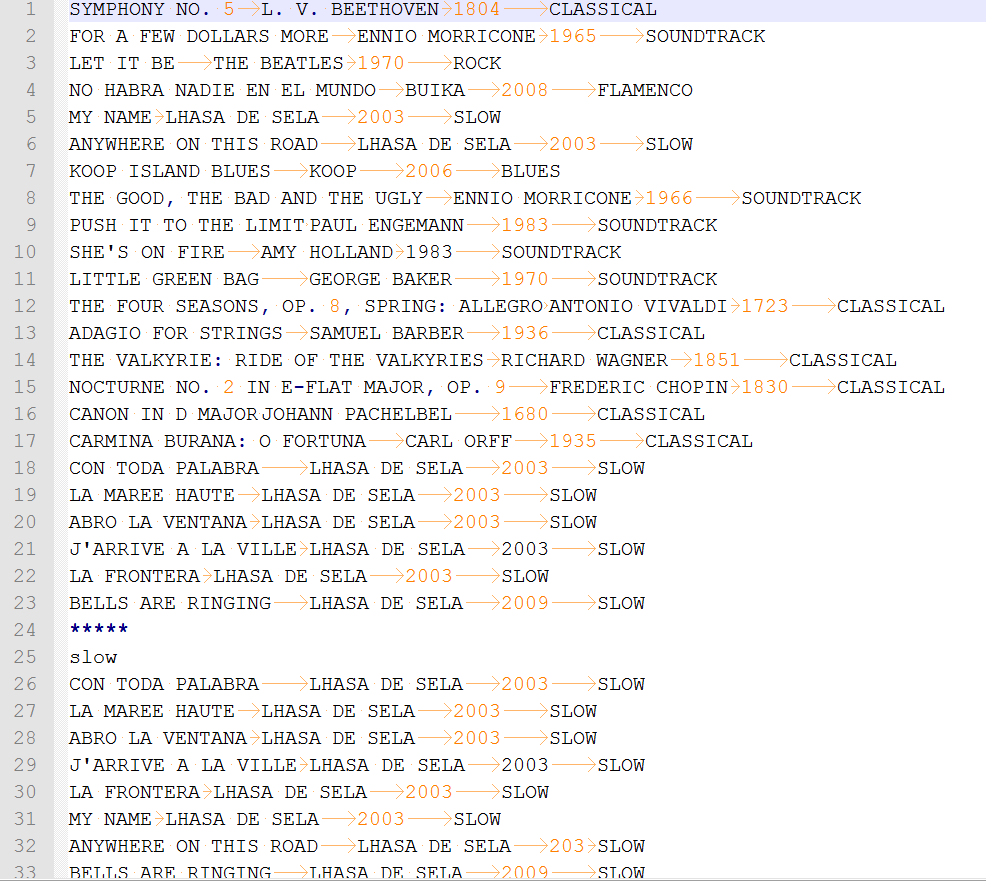
1. First, **read the records** from the file named “**songbook.txt**” provided with HW2.

There are several records in **songbook.txt**. Each row of the file corresponds to a different record. The parameters of each record are separated by a single tab character (‘\t’).

<song\_name> **→** <singer\_name> **→** <year> **→** <style>

A screenshot of the file is given below. **Your program will be tested with this file.**

**Songbook.txt :**

****

1. **Store the data you read** in your program, and **initialize your linked list structure** with the three default playlists (chronological, sorted and random).
2. **Dynamically allocate memory** for your variables. Their sizes should be adapted to the contained data.

**Implementation**

**Implement the following methods** with appropriate arguments and return types for your structure:

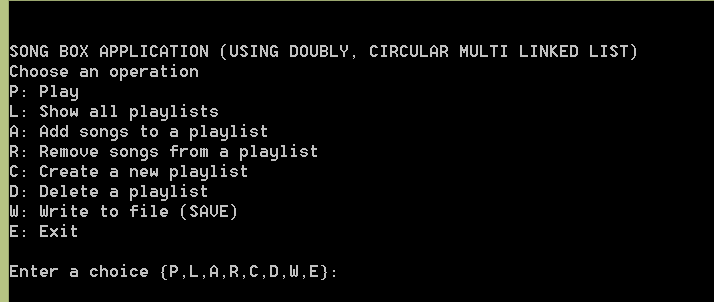
* 1. **createList()**: Creates a new playlist, prompting the user with the following options: 1) songs of **a specific style**, 2) songs of **a specific singer**, 3) **a combination of existing playlists**, or 4) **a combination of existing songs** (from the sorted playlist). Concatenates all songs of the selected lists when combining existing playlists.
  2. **addSong()**: Adds a new song (from the sorted playlist) to the specified user‑generated playlist. A user‑generated playlist can include duplicate songs.
  3. **play()**: Prompts the user with the following options: 1) **playing a playlist** starting from the first song, 2) **playing a playlist starting from a specific song**, or 3) **playing a single song**.
     + **Playing means printing** the name, singer, year and the style of the song.
     + For the first option, **prompt for the name of the playlist**. Pressing the keys **N** (NEXT), **P** (PREVIOUS) and **E** (EXIT) (use getchar()) respectively causes the player to move on to the next song, go back to the previous song, or exit to main menu. Remember that all playlists must be circular, so the first and last songs should be connected.
     + For the second option, **list all playlists** and let the user choose one, **then list the songs in that playlist** and let the user choose again.
     + For the third option, **list the songs in the sorted playlist** and let the user choose one.
  4. **removeSong()**: **Lists all playlists** and lets the user choose one, **then lists the songs in that playlist** and lets the user choose the song to be removed from that playlist.
  5. **deleteList()**: **Lists all playlists** and let the user choose the oneto be deleted.
  6. **printList(): Lists all playlists**, then print the first one. Pressing the keys **N** (NEXT), **P** (PREVIOUS) , **S** (SELECT) and **E** (EXIT) (use getchar()) respectively causes the player to move on to the next playlist, go back to the previous playlist, select the current playlist, or exit to main menu. After selecting a playlist, prompt the user for the following options: **D** (delete the playlist), **A** (add song), **R** (remove song) and **P** (play the songs in the playlist).
  7. **writeToFile():** Writes the chronological playlist first, and then all the user‑generated lists, to “**songbook.txt**”. Separates playlists by a line with 5 asterisks (“\*\*\*\*\*”) as given in the file screenshot. User‑generated playlists should be written exactly after the chronological playlist.
  8. **exit():** Saves the current playlists and terminates the program.

**Commands:**

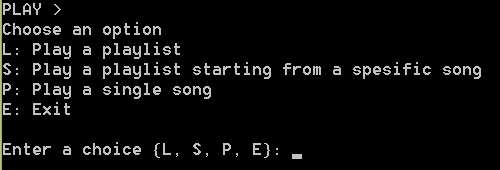
* **P play()**
* **L printList()**
* **C createList()**
* **A** **addSong()**
* **R removeSong()**
* **D** **deleteList()**
* **W** **writeToFile()**
* **E exit()**

**Screenshots:**

Main menu:



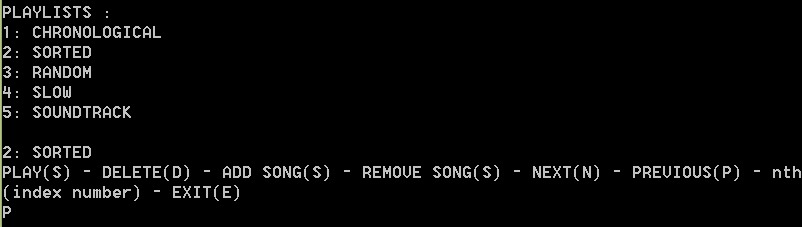
Playing options:

****

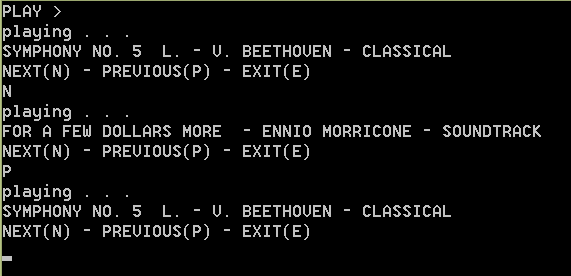
Listing all playlists:

****

Next/Previous:

****

Playing a playlist:

****

**Generating random numbers:**

Randomness is simulated by a computer using a pseudo-random number generator. The C library contains a few built-in options for generating random numbers.

**Usage:**

**#include** <time.h>

…..

**int** randomnumber;

srand( time( NULL ) ); // initializes the pseudo-random number generator

randomnumber = 5 + rand() % 20; // generating random numbers between 5 and 25

When creating the randomly ordered playlist each element in the list should be equally likely to be chosen as the new head for the random ordering playlist. And each remaining element is equally likely to be chosen as the “next” node after the head, etc. Finally, the last node in random order points back to the head node (circular).

**Submission**

1. Make sure you write your name and number in all of the files of your project, in the following format:

/\* @Author

\* Student Name: <student\_name>

\* Student ID : <student\_id>

\* Date: <date> \*/

1. Use comments wherever necessary in your code to explain what you did.
2. **Compile the code in the Secure Shell Client (SSH) before you send your homework.**
3. After you make sure that everything is compiled smoothly, archive all files into a zip file. Submit this file through www.ninova.itu.edu.tr. Ninova enables you to change your submission before the submission deadline.

**Do not** miss submission deadline. **Do not** leave your submission until the last minute. The submission system tends to become less responsive due to high network traffic.

**HOMEWORKS SENT VIA E-MAIL WILL NOT BE GRADED**.

**Academic dishonesty including but not limited to cheating, plagiarism and collaboration is unacceptable and subject to disciplinary actions. Your homeworks will be checked with a plagiarism checker system, any student found guilty will receive 0 as his/her grade for the homework and subject to disciplinary actions.**

If you have any question about the homework, contact the teaching assistant **Cumali TÜRKMENOĞLU** via e-mail (**turkmenogluc@itu.edu.tr**) or in **Research Lab 1**.