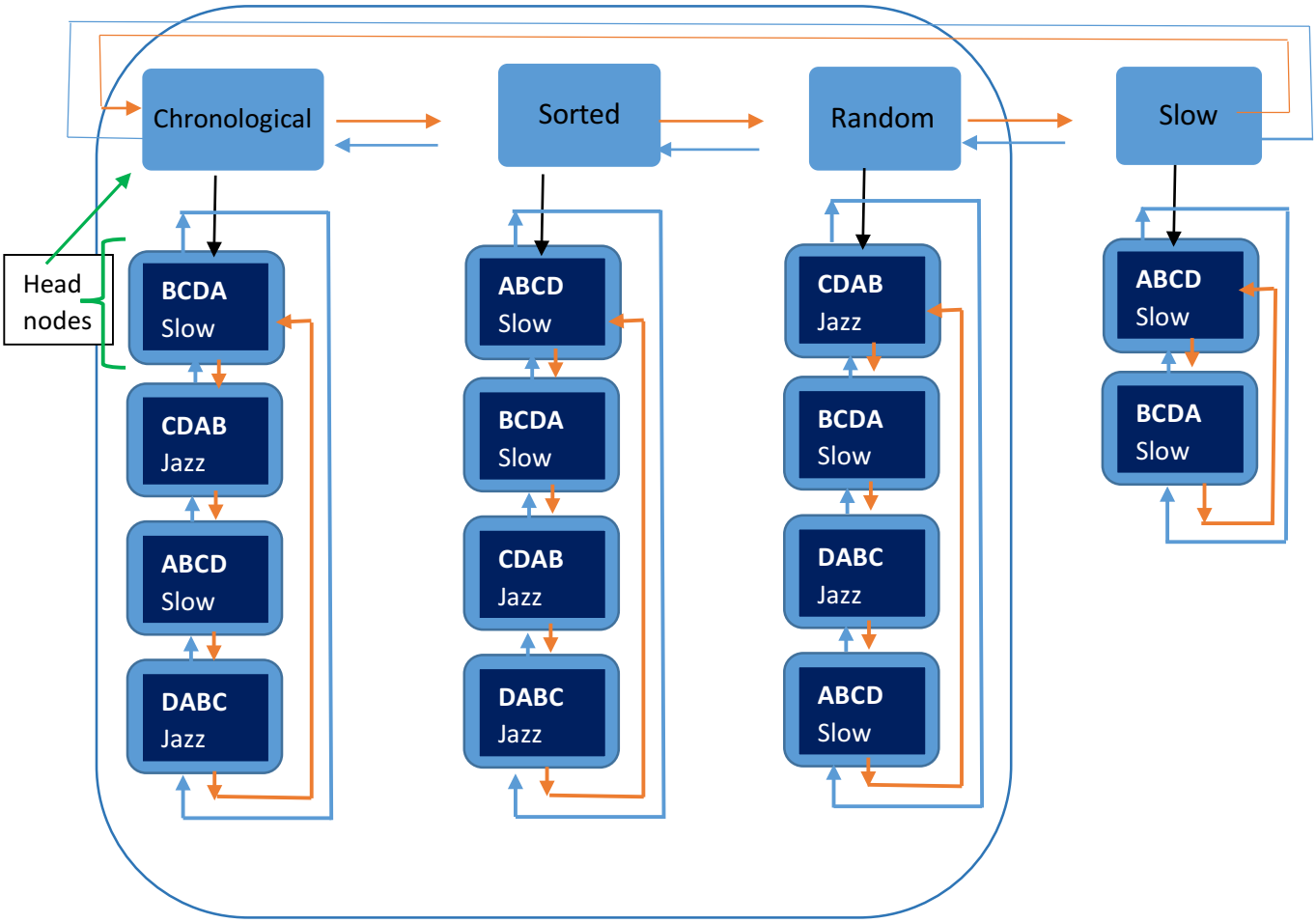
ITU COMPUTER ENGINEERING DEPARTMENT

BLG 233E DATA STRUCTURES

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:




```
#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 :

```
SYMPHONY NO. 5→L. V. BEETHOVEN→1804→CLASSICAL
FOR A FEW DOLLARS MORE→ENNIO MORRICONE→1965→SOUNDTRACK
LET IT BE→THE BEATLES→1970→ROCK
NO HABRA NADIE EN EL MUNDO→BUIKA→2008→FLAMENCO
MY NAME→LHASA DE SELA→2003→SLOW
ANYWHERE ON THIS ROAD→LHASA DE SELA→2003→SLOW
KOOP ISLAND BLUES→KOOP→2006→BLUES
THE GOOD, THE BAD AND THE UGLY→ENNIO MORRICONE→1966→SOUNDTRACK
PUSH IT TO THE LIMIT PAUL ENGEMANN→1983→SOUNDTRACK
SHE'S ON FIRE→AMY HOLLAND→1983→SOUNDTRACK
LITTLE GREEN BAG→GEORGE BAKER→1970→SOUNDTRACK
THE FOUR SEASONS, OP. 8, SPRING: ALLEGRO ANTONIO VIVALDI→1723→CLASSICAL
ADAGIO FOR STRINGS→SAMUEL BARBER→1936→CLASSICAL
THE VALKYRIE: RIDE OF THE VALKYRIES→RICHARD WAGNER→1851→CLASSICAL
NOCTURNE NO. 2 IN E-FLAT MAJOR, OP. 9→FREDERIC CHOPIN→1830→CLASSICAL
CANON IN D MAJOR JOHANN PACHELBEL→1680→CLASSICAL
CARMINA BURANA: O FORTUNA→CARL ORFF→1935→CLASSICAL
CON TODA PALABRA→LHASA DE SELA→2003→SLOW
LA MAREE HAUTE→LHASA DE SELA→2003→SLOW
ABRO LA VENTANA→LHASA DE SELA→2003→SLOW
J'ARRIVE A LA VILLE→LHASA DE SELA→2003→SLOW
LA FRONTERA→LHASA DE SELA→2003→SLOW
BELLS ARE RINGING→LHASA DE SELA→2009→SLOW
*****
slow
CON TODA PALABRA→LHASA DE SELA→2003→SLOW
LA MAREE HAUTE→LHASA DE SELA→2003→SLOW
ABRO LA VENTANA→LHASA DE SELA→2003→SLOW
J'ARRIVE A LA VILLE→LHASA DE SELA→2003→SLOW
LA FRONTERA→LHASA DE SELA→2003→SLOW
MY NAME→LHASA DE SELA→2003→SLOW
ANYWHERE ON THIS ROAD→LHASA DE SELA→203→SLOW
BELLS ARE RINGING→LHASA DE SELA→2009→SLOW
```

2. **Store the data you read** in your program, and **initialize your linked list structure** with the three default playlists (`chronological`, `sorted` and `random`).

3. **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:

a. **`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.

b. **`addSong():`** Adds a new song (from the `sorted` playlist) to the specified user-generated playlist. A user-generated playlist can include duplicate songs.

c. **`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.

d. **`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.

e. **`deleteList():`** **Lists all playlists** and let the user choose the one to be deleted.

f. **`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).

g. **`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.

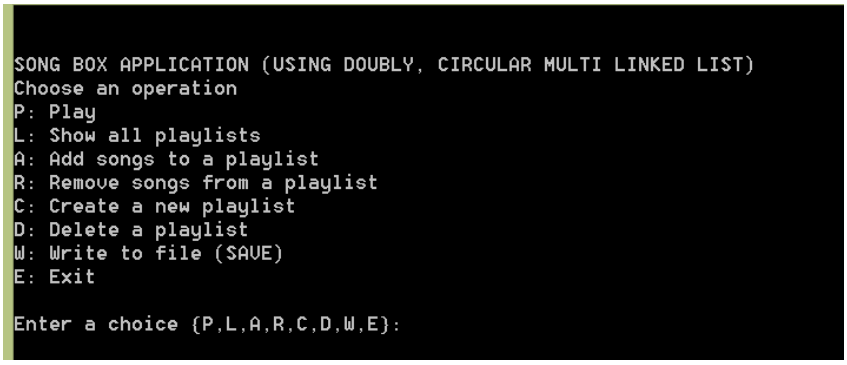
h. **`exit():`** Saves the current playlists and terminates the program.

Commands:

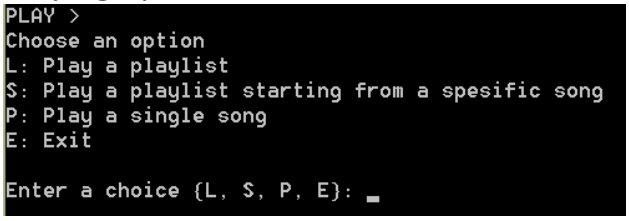
- **P play()**
- **L printList()**
- **C createList()**
- **A addSong()**
- **R removeSong()**
- **D deleteList()**
- **W writeToFile()**
- **E exit()**

Screenshots:

Main menu:



Playing options:



Listing all playlists:

Next/Previous:

```
PLAYLISTS :
1: CHRONOLOGICAL
2: SORTED
3: RANDOM
4: SLOW
5: SOUNDTRACK

2: SORTED
PLAY(S) - DELETE(D) - ADD SONG(S) - REMOVE SONG(S) - NEXT(N) - PREVIOUS(P) - nth
(index number) - EXIT(E)
P
```

Playing a playlist:

```
PLAY >
playing . . .
SYMPHONY NO. 5  L. - V. BEETHOVEN - CLASSICAL
NEXT(N) - PREVIOUS(P) - EXIT(E)
N
playing . . .
FOR A FEW DOLLARS MORE  - ENNIO MORRICONE - SOUNDTRACK
NEXT(N) - PREVIOUS(P) - EXIT(E)
P
playing . . .
SYMPHONY NO. 5  L. - V. BEETHOVEN - CLASSICAL
NEXT(N) - PREVIOUS(P) - EXIT(E)
```

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).