CONTENTS

LIST OF FIGURES

1. INTRODUCTION

1.1 PROBLEM STATEMENT

- We all love listening to music. But having to search for the best music for your current mood can be annoying.
- It would be very convenient if our list of favorite songs was stored to different playlists.
- This project will help to manage songs and store them into different playlists.
- Moreover, we can retrieve a song according to its song id.
- Groovy – A Music Playlist Manager is a program based on the idea of storing list of songs along with its useful information as a playlist.

1.2 OBJECTIVE

- The objective of this project is to help the user (person in charge of the playlist) to maintain proper record of his/her favourite songs.
- This project helps them to create, add, delete and search for information of song like playlist name, song name, artist name, duration of the song, year of release and the genre of the song.

1.3 METHODOLOGY TO BE FOLLOWED

- Doubly Linked List helps us to store linear data in our case it is details, which is retrieved from user. The data is linked to each other. Ample number of data fields can be added in a DLL, which will provide brief information to users.
- Implementing various functions in a single project helps it to be more user friendly and also saves memory. A particular program using File Handling in storing data totally gets the job done which also helps for accessing it later.
- Structure, Array, Functions collective helps in implementing various modules of this program to obtain the objective.

1.4 EXPECTED OUTCOMES

This project will allow the user to select various option in the menu like:

1. Login with Username and Password

2.Create a playlist record (to create new items and it's information)

3. Add playlist record (to add new items)

4. Display the songs in the playlist (to display the added items)

5. Search for a song (to search an item using the item code)

6. Delete a song (to delete an item using the item code)

7. Length of the record (total no. of items added)

8. Exit the menu

1.5. HARDWARE AND SOFTWARE REQUIREMENTS

Hardware requirements

Processor : Any processor above 1.2GHz ; RAM : 8GB

Input devices : Standard keyboard and mouse

Output devices : High resolution monitor

Software requirements

Operating System : Windows 7/10 or Mac OS

Data base : Data Structures

Platform : C language

Compiler : Dev C++ or Visual Studio Code(Used for my mini project)

2. DATA STRUCTURES

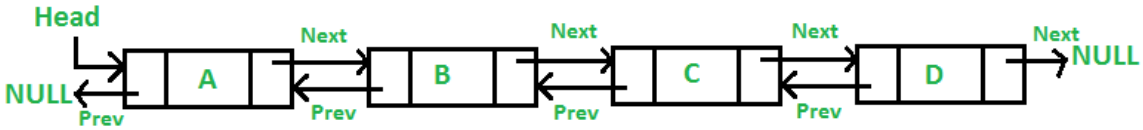
2.1 LINKED LIST:

A linked list is a linear data structure, in which the elements are not stored at contiguous memory locations. Elements are stored in random memory locations and are linked using pointers. Linked list consists of nodes where each node contains a data field and reference(link/pointer) to the next node in the list. Data field contains the value of the node and the reference contains the address of the next node.

2.1.1 DOUBLY LINKED LISTS:

Doubly Linked List is a type of Linked list in which navigation is possible in ways both, either forward and backward easily when as compared to Single Linked List.

Doubly linked list representation:



Doubly Linked List has links namely first and last.

Each of these links carry a data field(s) and two link fields called next and prev.

Each link in DLL is linked with its next link using its next link.

Each link is linked with its previous link using its previous link.

The last link has a link as null to mark the end of the list.

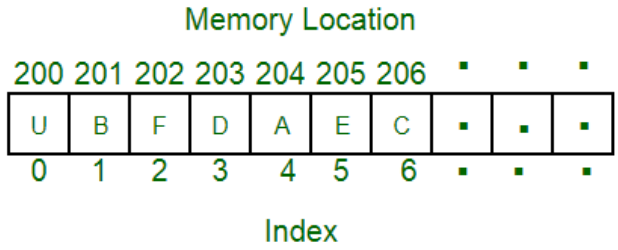
2.2. ARRAYS:

An array is used to store a collection of variables of the same type. A specific element in an array is accessed by an index.

The array elements are stored in contiguous memory locations. The first element is stored at the lowest address and the last element is stored at the highest address

An array is a finite collection of similar elements stored in adjacent memory locations. The number of elements to be used are mentioned as index usually ranging from o to N-1 where '0' is lower bound and N-1 is upper bound.

Arrays can be 1-D, 2-D or multi-dimensional. Syntax- Data-type array name[index]



2.3. POINTERS:

The pointer in C language is a variable which stores the address of another variable. This variable can be of type int, char, array, function, or any other pointer. Consider the following example to define a pointer which stores the address of an integer.

–**int** x = 5; , –**int** *p = &x;

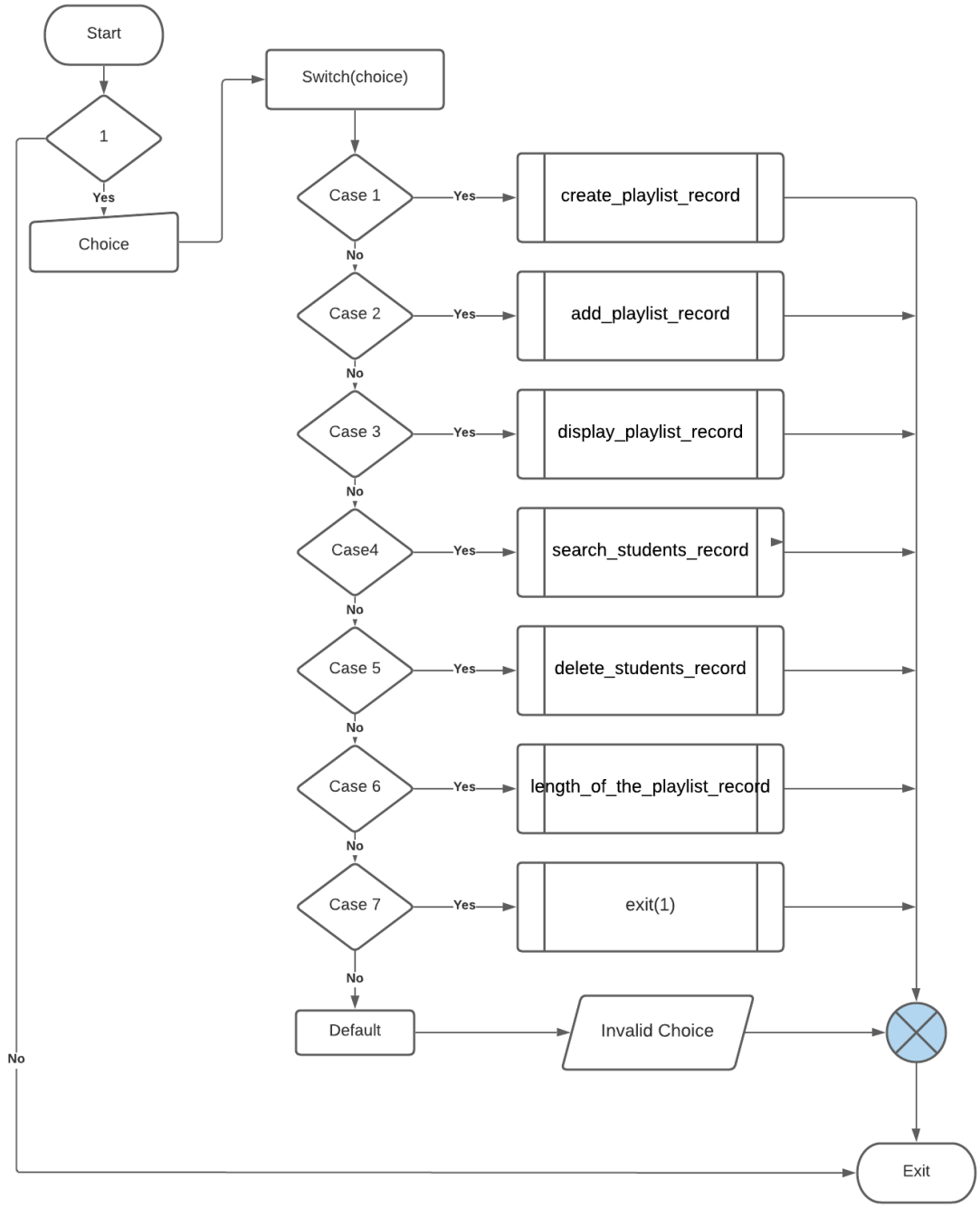
// Variable p of type pointer is pointing to the address of the variable n of type integer.

3. DESIGN

3.1. ALGORITHM:

1) Declaration of a struct for storing the data of playlist in the file.
2) Declaration and defining a function for logging with user name and password
3) Declaration and defining a function for creating song playlist. Store the record in a linked list and that of linked list in a file.
4) Declaration and defining of a function for adding a playlist's record. Store the record in a linked list and that of linked list in a file.
5) Declaration and defining of a function for displaying the song details through DLL.
6) Declaration and defining of a function for searching a song.
7) Declaration and defining of a function for deleting a song.
8) In the main function use the switch case to select the options(create, add, display, search , delete, total, exit) .

3.2. FLOWCHART:



4. IMPLEMENTATION

4.1 MODULE 1 : HEADER FILES AND STRUCTURE:

```
/**
                Playlist Record
          --------------------------------
**/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

struct node
{
   int id;
   char playlist[50], year[20];
   char genre[7];
   char duration[100], song[50], artist[50];

   struct node *next;
   struct node *previous;

} *start = NULL, *end = NULL;
FILE *file;
```

A Structure has been created which is used to implement doubly linked list in the program. It will be used to store the details of the playlist. These details include song id, playlist name, song name, artist name, song duration, year of release, genre of song.

4.2 MODULE 2: LOGIN:

```
void login()
{
   char uname[10], c = ' ';
   char pword[10], code[10];
   char user[10] = "user";
   char pass[10] = "pass";

   printf("  \xdb\xdb\xdb\xdb\xdb\xdb\xdb\xdb\xdb\xdb\xdb\xdb\xdb\xdb\xdb\xdb\xdb\xdb\xdb\xdb
              \xdb\xdb\xdb\xdb\xdb\xdb\xdb\xdb\xdb\xdb\xdb\xdb       LOGIN       \xdb\xdb\xdb
              \xdb\xdb\xdb\xdb\xdb\xdb\xdb\xdb\xdb\xdb\xdb\xdb\xdb\xdb\xdb\xdb\xdb\xdb\xdb\xdb\xdb\
              xdb\xdb\xdb\xdb\xdb\xdb\xdb\xdb\xdb\xdb\xdb\xdb  ");
   printf(" \n\n\t\t                USERNAME:-");
   scanf("%s", uname);
   printf(" \n\t\t                PASSWORD:-");
   scanf("%s", pword);

   if (strcmp(uname, "admin") == 0 && strcmp(pword, "groovy") == 0)
   {
      printf("  \n\n\n       WELCOME TO GROOVY - A MUSIC PLAYLIST MANAGER !!!! LOGIN IS SUCCESSFUL");
      //getch();
   }
   else
   {
      printf("\n       SORRY !!!!  LOGIN IS UNSUCESSFUL");
      exit(0);
      //getch();
   }
}
```

This module is used for logging in with username and password. The given username is “admin” and password is “groovy”. If login is successful, the menu will be displayed. If it is unsuccessful, an error message will be displayed.

4.3 MODULE 3: CREATING PLAYLIST RECORD:

```
void create_playlist_record()
{
   struct node *new_node, *current;
   int i, number_of_song;

   printf("\n\n\n\n\n\n\t\t\tEnter Number of song's for playlist: ");
   scanf("%d", &number_of_song);

   for (i = 1; i <= number_of_song; i++)
   {
      new_node = (struct node *)malloc(sizeof(struct node));

      if (new_node == NULL)
      {
         printf("\nMemory Does Not Created.\n");
         exit(0);
      }
      else
      {
         file = fopen("Playlist Record.txt", "a+");
         if (file == NULL)
         {
            printf("File does not create.\n");
         }
         else
         {
            printf("\n\n\t\t\tPlaylist Details\n");
            fprintf(file, "\n\n\t\t\tPlaylist Details\n");
            printf("\t\t_____\n");
            fprintf(file, "\t\t_____\n");

            printf("\n\n\tPlaylist Name: ");
            fflush(stdin);
            gets(new_node->playlist);
            fprintf(file, "\n\tPlaylist Name: %s", new_node->playlist);

            printf("\n\tSong ID: ");
            scanf("%d", &new_node->id);
            fprintf(file, "\n\tSong ID: %d", new_node->id);

            printf("\n\tSong Name: ");
            fflush(stdin);
            gets(new_node->song);
            fprintf(file, "\n\tSong Name: %s", new_node->song);
```

```
            printf("\n\tArtist: ");
            fflush(stdin);
            gets(new_node->artist);
            fprintf(file, "\n\tArtist: %s", new_node->artist);

            printf("\n\tYear of release: ");
            fflush(stdin);
            gets(new_node->year);
            fprintf(file, "\n\tYear of release: %s", new_node->year);

            printf("\n\tSong Duration: ");
            fflush(stdin);
            gets(new_node->duration);
            fprintf(file, "\n\tSong Duration: %s", new_node->duration);

            printf("\n\tGenre: ");
            fflush(stdin);
            gets(new_node->genre);
            fprintf(file, "\n\tGenre: %s", new_node->genre);

            fclose(file);
            fopen("Playlist Record.txt", "a+");
        }

        new_node->next = NULL;
        new_node->previous = NULL;

        if (start == NULL && end == NULL)
        {
            start = new_node;
            end = new_node;
            current = new_node;
        }
        else
        {
            current->next = new_node;
            new_node->previous = current;
            current = new_node;
            end = new_node;
        }
    }
  }
}
```

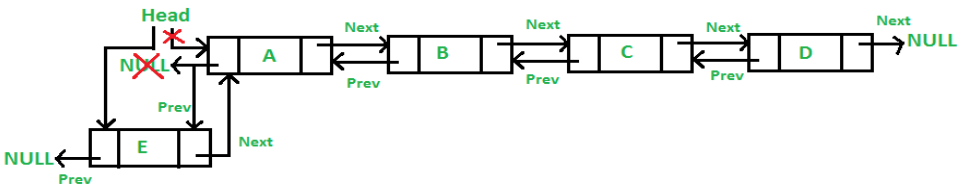
This module is used to create a new playlist record. Memory allocation is done with malloc( ) to the new node to be created. Using doubly linked list the playlist record is stored. File Handling helps to retrieve information from user and is printed in the file using “fprintf( )” .

4.4 MODULE 4: (I)ADDING PLAYLIST RECORD AT BEGINNING:

```
void add_playlist_record_at_first()
{
   struct node *new_node, *current;
   new_node = (struct node *)malloc(sizeof(struct node));

   if (new_node == NULL)
   {
      printf("\nMemory Is Not Created.\n");
      exit(0);
   }
   else
   {
      file = fopen("Playlist Record.txt", "a+");
      if (file == NULL)
      {
         printf("File is not created.\n");
      }
      else
      {
         printf("\n\n\t\t\tPlaylist Details\n");
         fprintf(file, "\n\n\t\t\tPlaylist Details\n");
         printf("\t\t_____\n");
         fprintf(file, "\t\t_____\n");

         printf("\n\n\tEnter Playlist Name: ");
         fflush(stdin);
         gets(new_node->playlist);
         fprintf(file, "\n\tPlaylist Name: %s", new_node->playlist);

         printf("\n\tSong ID: ");
         scanf("%d", &new_node->id);
         fprintf(file, "\n\tSong ID: %d", new_node->id);

         printf("\n\tSong Name: ");
         fflush(stdin);
         gets(new_node->song);
         fprintf(file, "\n\tSong Name: %s", new_node->song);

         printf("\n\tArtist: ");
         fflush(stdin);
         gets(new_node->artist);
         fprintf(file, "\n\tArtist: %s", new_node->artist);
```

```
        printf("\n\tYear of release: ");
        fflush(stdin);
        gets(new_node->year);
        fprintf(file, "\n\tYear of release: %s", new_node->year);

        printf("\n\tSong Duration: ");
        fflush(stdin);
        gets(new_node->duration);
        fprintf(file, "\n\tSong Duration: %s", new_node->duration);

        printf("\n\tGenre: ");
        fflush(stdin);
        gets(new_node->genre);
        fprintf(file, "\n\tGenre: %s", new_node->genre);

        fclose(file);
        fopen("Playlist Record.txt", "a+");
    }
  }

  new_node->next = NULL;
  new_node->previous = NULL;

  current = start;
  new_node->next = current;
  current->previous = new_node;
  start = new_node;
}
```

This module is used to add a new playlist record at the beginning/above the existing record. It was very similar to creating a new record. The details are retrieved from the user and stored in file using file handling. The new record is inserted before the first record as pictorially represented below.

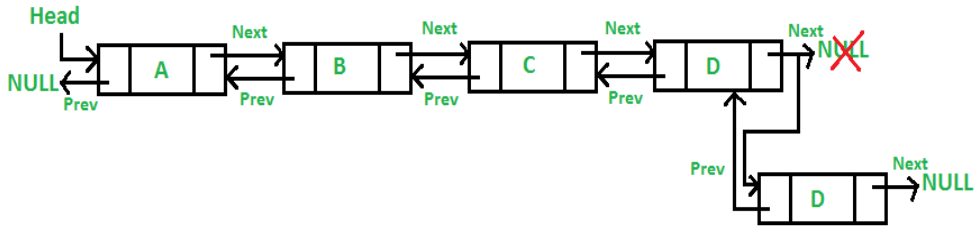


4.4 MODULE 4: (II)ADDING PLAYLIST RECORD AT END:

```
void add_playlist_record_at_last()
{
   struct node *new_node, *current;
   new_node = (struct node *)malloc(sizeof(struct node));

   if (new_node == NULL)
   {
      printf("\nMemory Is Not Created.\n");
      exit(0);
   }
   else
   {
      file = fopen("Playlist Record.txt", "a+");
      if (file == NULL)
      {
         printf("File is not created.\n");
      }
      else
      {
         printf("\n\n\t\t\tPlaylist Details\n");
         fprintf(file, "\n\n\t\t\tPlaylist Details\n");
         printf("\t\t_____\n");
         fprintf(file, "\t\t_____\n");

         printf("\n\n\tEnter Playlist Name: ");
         fflush(stdin);
         gets(new_node->playlist);
         fprintf(file, "\n\tPlaylist Name: %s", new_node->playlist);

         printf("\n\tSong ID: ");
         scanf("%d", &new_node->id);
         fprintf(file, "\n\tSong ID: %d", new_node->id);

         printf("\n\tSong Name: ");
         fflush(stdin);
         gets(new_node->song);
         fprintf(file, "\n\tSong Name: %s", new_node->song);

         printf("\n\tArtist: ");
         fflush(stdin);
         gets(new_node->artist);
         fprintf(file, "\n\tArtist: %s", new_node->artist);
```

```
            printf("\n\tArtist: ");
            fflush(stdin);
            gets(new_node->artist);
            fprintf(file, "\n\tArtist: %s", new_node->artist);

            printf("\n\tYear of release: ");
            fflush(stdin);
            gets(new_node->year);
            fprintf(file, "\n\tYear of release: %s", new_node->year);

            printf("\n\tSong Duration: ");
            fflush(stdin);
            gets(new_node->duration);
            fprintf(file, "\n\tSong Duration: %s", new_node->duration);

            printf("\n\tGenre: ");
            fflush(stdin);
            gets(new_node->genre);
            fprintf(file, "\n\tGenre: %s", new_node->genre);
            fclose(file);
            fopen("Playlist Record.txt", "a+");
        }
    }

    new_node->next = NULL;
    new_node->previous = NULL;

    current = end;
    current->next = new_node;
    new_node->previous = current;
    end = new_node;
}
```

This module is used to add new record at the end/after the existing record. The new record is inserted after the last record as pictorially represented below. The details are retrieved from the user and stored in file using file handling.



4.5 MODULE 5: (I)DISPLAYING THE STORED RECORD FROM FORWARD:

```
void display_playlist_record_from_forward()
{
   struct node *current;

   current = start;

   if (current == NULL)
   {
      printf("\n\n\n\n\n\n\n\t\t\tThere Are No Record In The List.\n");
   }
   else
   {
      while (current != NULL)
      {
         printf("\n\t\t\tPlaylist Details\n");
         printf("\t\t_____");

         printf("\n\tPlaylist Name: %s", current->playlist);

         printf("\n\tSong ID: %d", current->id);

         printf("\n\tSong Name: %s", current->song);

         printf("\n\tArtist: %s", current->artist);

         printf("\n\tYear of release: %s", current->year);

         printf("\n\tSong Duration: %s", current->duration);

         printf("\n\tGenre: %s", current->genre);

         current = current->next;
         printf("\n");
      }
   }
}
```

This module is used for displaying the playlist records in forward direction. It traverses from starting to the end of stored list(until the next pointer points to NULL) and prints all the details of a song using a while loop.

4.5 MODULE 5: (II)DISPLAYING THE STORED RECORD FROM BACKWARD:

```
void display_playlist_record_from_backword()
{
   struct node *current;

   current = end;

   if (current == NULL)
   {
      printf("\n\n\n\n\n\n\n\t\t\tThere Are No Record In The List.\n");
   }
   else
   {
      while (current != NULL)
      {
         printf("\n\t\t\tPlaylist Details\n");
         printf("\t\t_____");

         printf("\n\tPlaylist Name: %s", current->playlist);

         printf("\n\tSong ID: %d", current->id);

         printf("\n\tSong Name: %s", current->song);

         printf("\n\tArtist: %s", current->artist);

         printf("\n\tYear of release: %s", current->year);

         printf("\n\tSong Duration: %s", current->duration);

         printf("\n\tGenre: %s", current->genre);

         current = current->previous;
         printf("\n");
      }
   }
}
```

This module is used for displaying the playlist records in backward direction. It traverses from the end to the beginning of stored list(until the previous pointer points to Head) and prints all the details of a song using a while loop.

4.6 MODULE 6: POSITIONING OF THE NODE:

```
int pos(song_id)
{
   int position = 0;
   struct node *current;

   current = start;

   while (current != NULL)
   {
      position++;
      if (song_id == current->id)
      {
         return position;
      }
      current = current->next;
   }
   return -1;
}
```

This position function is used for positioning a particular node. Using while loop, the linked list is traversed until the song id is equal to the current node . When it is equal, the position of the node id retrieved.

4.7 MODULE 7: SEARCHING A PARTICULAR SONG:

```
int search_playlist_record(song_id)
{
   int position = 0;
   struct node *current;

   current = start;

   while (current != NULL)
   {
      position++;
      if (song_id == current->id)
      {
         printf("\n\t\t\tPlaylist's Details\n");
         printf("\t\t_____");

         printf("\n\tPlaylist Name: %s", current->playlist);

         printf("\n\tSong ID: %d", current->id);

         printf("\n\tSong Name: %s", current->song);

         printf("\n\tArtist: %s", current->artist);

         printf("\n\tYear of release: %s", current->year);

         printf("\n\tSong Duration: %s", current->duration);

         printf("\n\tGenre: %s", current->genre);

         printf("\n\n");

         return position;
      }
      current = current->next;
   }
   return -1;
}
```

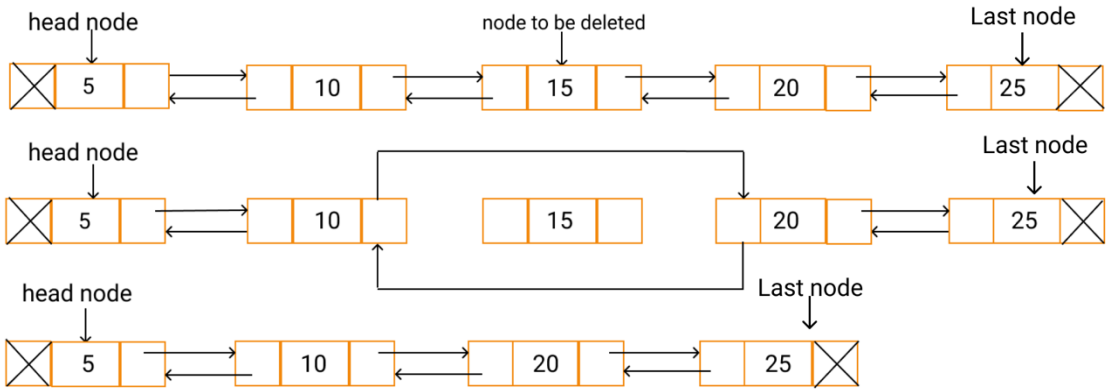
This module is used for searching for a particular song. A pointer current = start along with a while loop is used to traverse through the linked list until current is not equal to NULL. If the current code is equal to the song id then the details of that song will be printed/displayed.

4.8 MODULE 8: DELETING A SONG FROM RECORD:

```
void delete_playlist_record()
{
   struct node *current, *temp1, *temp2;
   int i, delet_id, position;

   printf("\t\t\tEnter ID for delete: ");
   scanf("%d", &delet_id);
   position = pos(delet_id);

   current = start;
   for (i = 1; i <= (position - 1); i++)
   {
      current = current->next;
   }
   if (current == start && current->previous == NULL)
   {
      current = current->next;
      start = current;
      current->previous = NULL;
      printf("\nFirst ID Delete Successfully.\n");
   }
   else if (current->next == NULL && current == end)
   {
      current = current->previous;
      end = current;
      current->next = NULL;
      printf("\nLast ID Delete Successfully.\n");
   }
   else
   {
      temp2 = current->next;
      temp1 = current->previous;
      temp1->next = temp2;
      temp2->previous = temp1;
      printf("\nDelete Successfully.\n");
   }
}
```

This module is used for deleting a record of a song (node) from the linked list. Initially we take input of the position of the song to be deleted. Then ,we traverse using a temporary node called 'current ' until position is reached ,we assign 2 temporary nodes called temp1 and temp2. temp2 is pointing to current's next pointer ,whereas temp1is pointing to current's previous pointer, then we assign the next of temp2 to the current and previous of current to temp1. Whereas ,if the current is start ,then we assign the current as next ,and current as start . Later the current's previous to "NULL" so as to delete the first node.

Meanwhile ,the current's next is "NULL", then assign current to its previous and assign the current node's next to NULL to delete the last node. A pictorial example is given below:



4.9 MOUDLE 9: LENGTH OF THE PLAYLIST:

```
void length_of_the_playlist_record()
{
   struct node *current;
   int count = 0;

   current = start;

   while (current != NULL)
   {
      count++;
      current = current->next;
   }

   printf("\n\n\n\n\n\n\n\t\t\tThere is/are %d song(s) in the playlist.", count);
   printf("\n                    Thereforth the length is %d.\n", count);
}
```

The above function is used to display the total number of songs in the DLL. Integer count =0, pointer current is made equal to ptointer start. While loop is used to traverse through the linked list until the current pointer is not equal to NULL. Then the count is incremented to calculate the total and then the number of songs is listed.

4.10 MODULE 10: MENU FUNTION:

```
void menu()
{

   int choice, position, song_id, insert_choice, display_choice, delete_choice, search_choice;

   printf("\n\t\t\t\t     Playlist's Record\n");
   printf("\t\t\t_____________________________________________________");
   printf("\n\n");

   printf("-------------------------------------------------------------------------------------------");
   printf("\n\n");
   printf("\t\t\t\t1. Create Playlist Record.\n");
   printf("\t\t\t\t2. Add Playlist Record.\n");
   printf("\t\t\t\t3. Display Playlist Record.\n");
   printf("\t\t\t\t4. Search Playlist Record.\n");
   printf("\t\t\t\t5. Delete Playlist Record.\n");
   printf("\t\t\t\t6. Length of the Playlist.\n");
   printf("\t\t\t\t7. Exit.\n\n");
   printf("-------------------------------------------------------------------------------------------");

   printf("\n\t\t\tPlease Enter your choice: ");
   scanf("%d", &choice);
   printf("\n");

   switch (choice)
   {
   case 1:
      create_playlist_record();
      printf("\n\n\n\n\n\n\n\t\t\tRecord Created Successfully.");
      break;

   case 2:
      while (1)
      {
         printf("\n\n\n\n\n");
         printf("-------------------------------------------------------------------------------------------");
         printf("\n\n");
         printf("\t\t\t\t1. Add Record At First.\n");
         printf("\t\t\t\t2. Add Record At Last.\n");
         printf("\t\t\t\t3. Back To Main Program.\n\n");
         printf("-------------------------------------------------------------------------------------------");

         printf("\n\n\n\t\t\tPlease Enter your choice: ");
         scanf("%d", &insert_choice);
         printf("\n\n");
```

```
        switch (insert_choice)
        {
        case 1:
            add_playlist_record_at_first();

            printf("\n\n\n\n\n\n\n\t\t\tRecord Added successfully.");

            break;

        case 2:
            add_playlist_record_at_last();

            printf("\n\n\n\n\n\n\n\t\t\tRecord Added successfully.");

            break;

        case 3:
            menu();

            break;

        default:
            printf("\n\n\n\n\n\n\n\t\t\tInvalid input!!!");
            printf("\n\t\t\tPlease Enter Correct Key to Access.");
        }
    }

case 3:
    while (1)
    {
        printf("\n\n\n\n\n");
        printf("-------------------------------------------------------------------------------------------");
        printf("\n\n");
        printf("\t\t\t\t1. Display From Forward.\n");
        printf("\t\t\t\t2. Display From Backward.\n");
        printf("\t\t\t\t3. Back To Main Program.\n\n");
        printf("-------------------------------------------------------------------------------------------");

        printf("\n\n\n\t\t\tPlease Enter your choice: ");
        scanf("%d", &display_choice);
        printf("\n\n");
```

```
        switch (display_choice)
        {
        case 1:
            display_playlist_record_from_forward();

            break;

        case 2:
            display_playlist_record_from_backword();

            break;

        case 3:
            menu();

            break;

        default:
            printf("\n\n\n\n\n\n\n\t\t\t\tInvalid input!!!");
            printf("\n\t\t\tPlease Enter Correct Key to Access.");
        }
    }

case 4:
    while (1)
    {
        printf("\n\n\n\n\n");
        printf("-------------------------------------------------------------------------------------------");
        printf("\n\n");
        printf("\t\t\t\t1. Search using Song ID.\n");
        printf("\t\t\t\t2. Back To Main Program.\n\n");
        printf("-------------------------------------------------------------------------------------------");
        printf("\n\n\n\t\t\tPlease Enter your choice: ");
        scanf("%d", &search_choice);
        printf("\n\n");

        switch (search_choice)
        {
        case 1:
            printf("\n\n\n\n\n\n\t\t\tEnter Song ID: ");
            scanf("%d", &song_id);

            position = search_playlist_record(song_id);
            if (position == -1)
            {
                printf("\n\n\n\n\n\n\t\t\tThis Song ID is not in the Record.\n");
```

```
            }
            else
            {
               printf("\n\n\n\n\n\n\t\t\tThe Position of this Record is at Number %d.\n", position);
            }
            break;

         case 2:
            menu();
            break;

         default:
            printf("\n\n\n\n\n\n\n\t\t\t\tInvalid input!!!");
            printf("\n\t\t\tPlease Enter Correct Key to Access.");
            printf("\n\t\t\t\tOr Enter 2 to Main menu.\n");
         }
      }

   case 5:
      while (1)
      {
         printf("\n\n\n\n\n");
         printf("-------------------------------------------------------------------------------------------");
         printf("\n\n");
         printf("\t\t\t\t1. Delete.\n");
         printf("\t\t\t\t2. Back To Main Program.\n\n");
         printf("-------------------------------------------------------------------------------------------");
         printf("\n\n\n\t\t\tPlease Enter your choice: ");
         scanf("%d", &delete_choice);
         printf("\n\n");

         switch (delete_choice)
         {
         case 1:
            delete_playlist_record();
            break;

         case 2:
            menu();
            break;

         default:
            printf("\n\n\n\n\n\n\n\t\t\t\tInvalid input!!!");
            printf("\n\t\t\tPlease Enter Correct Key to Access.");
            printf("\n\t\t\t\tOr Enter 2 to Main menu.\n");
         }
```

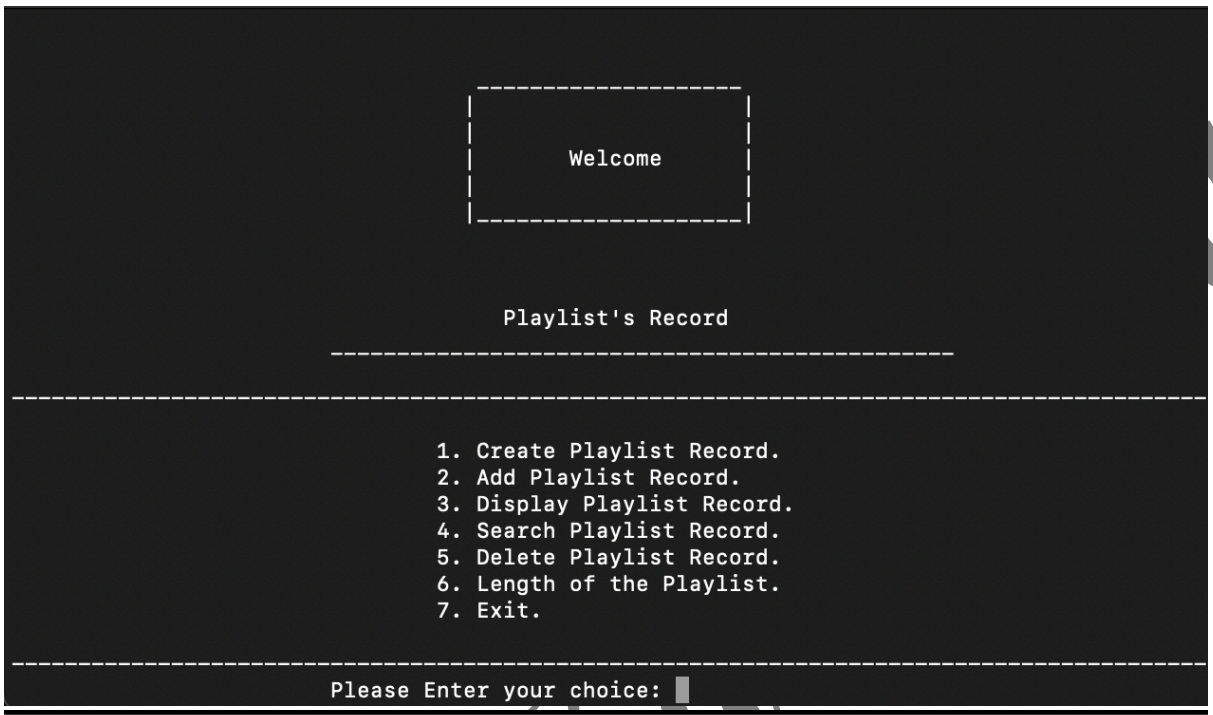
```
        }

    case 6:
        length_of_the_playlist_record();
        menu();
        break;

    case 7:
        exit(1);
        break;

    default:
        printf("\n\n\n\n\n\n\n\n\t\t\t\tInvalid input!!!");
        printf("\n\t\t\tPlease Enter Correct Key to Access.");
        printf("\n\t\t\t\tOr Enter 8 to Exit.\n");
    }
}
```

This menu function provides the user with all the available operation in the program like, creating a playlist, adding songs to the record from forward as well as from backward, displaying songs to the record from forward as well as from backward, searching a particular song using the song id, deleting a song in the record from forward as well as from backward, finding the length of playlist and exiting from the program.

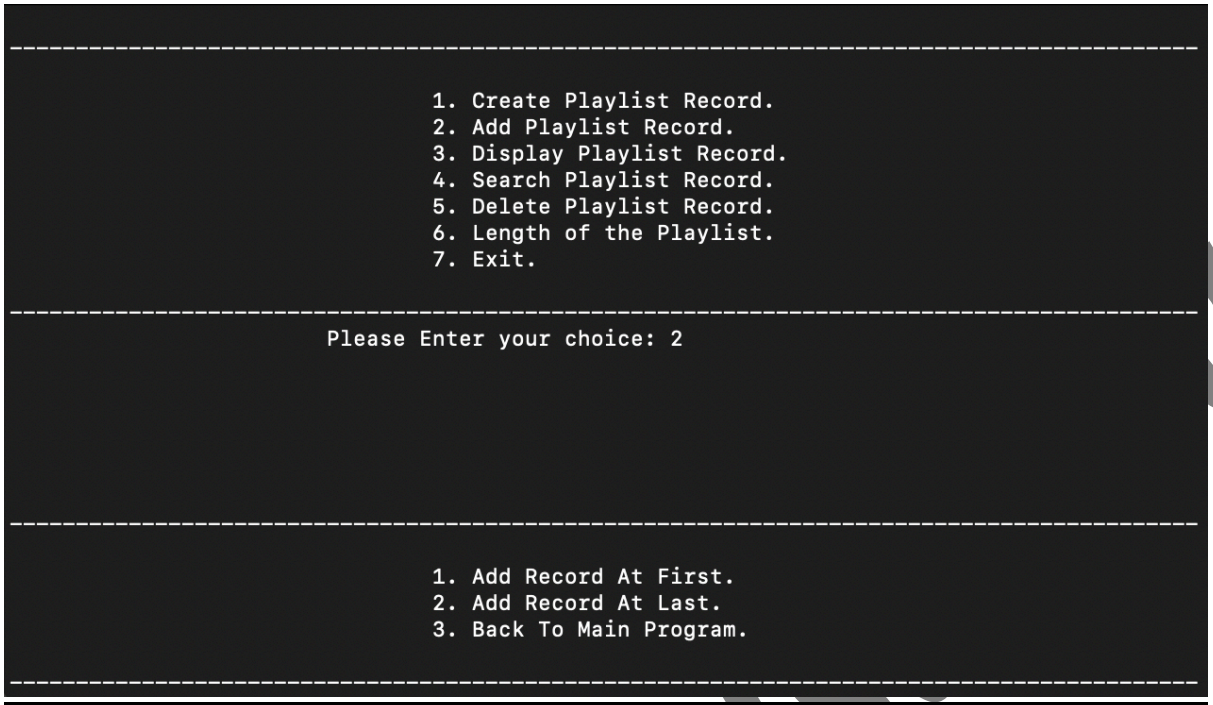
4.11 MODULE 11: MAIN FUNTION:

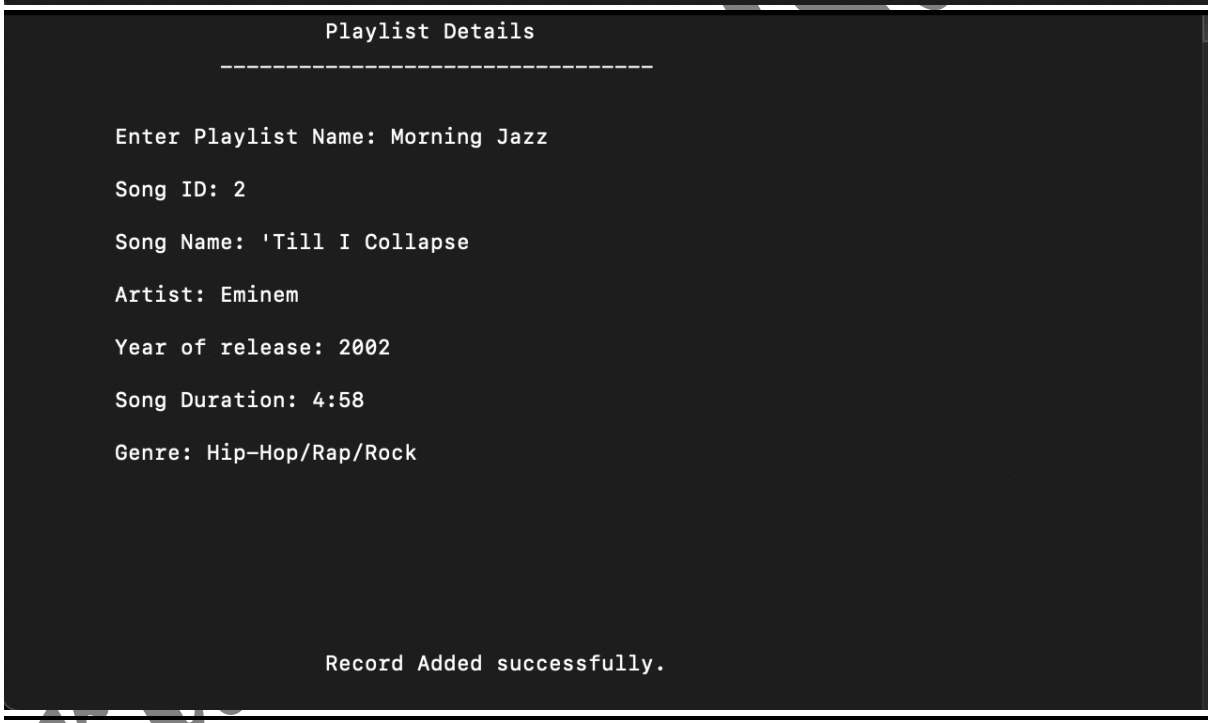
```
int main()
{

    login();

    printf("\n\n\n\n\n\n\n\n\t\t\t\t _____\n\t\t\t\t |          |\n\t\t\t\t |          |
            \n\t\t\t\t |\t Welcome    |\n\t\t\t\t |          |\n\t\t\t\t |_____|\n\n\n");

    while (1)
    {
        menu();
    }
}
```

5. OUTPUT



Creating a playlist called “Morning Jazz”. Retrieving basic data about the song from user and storing it in a file called “Paylist Record.txt”.

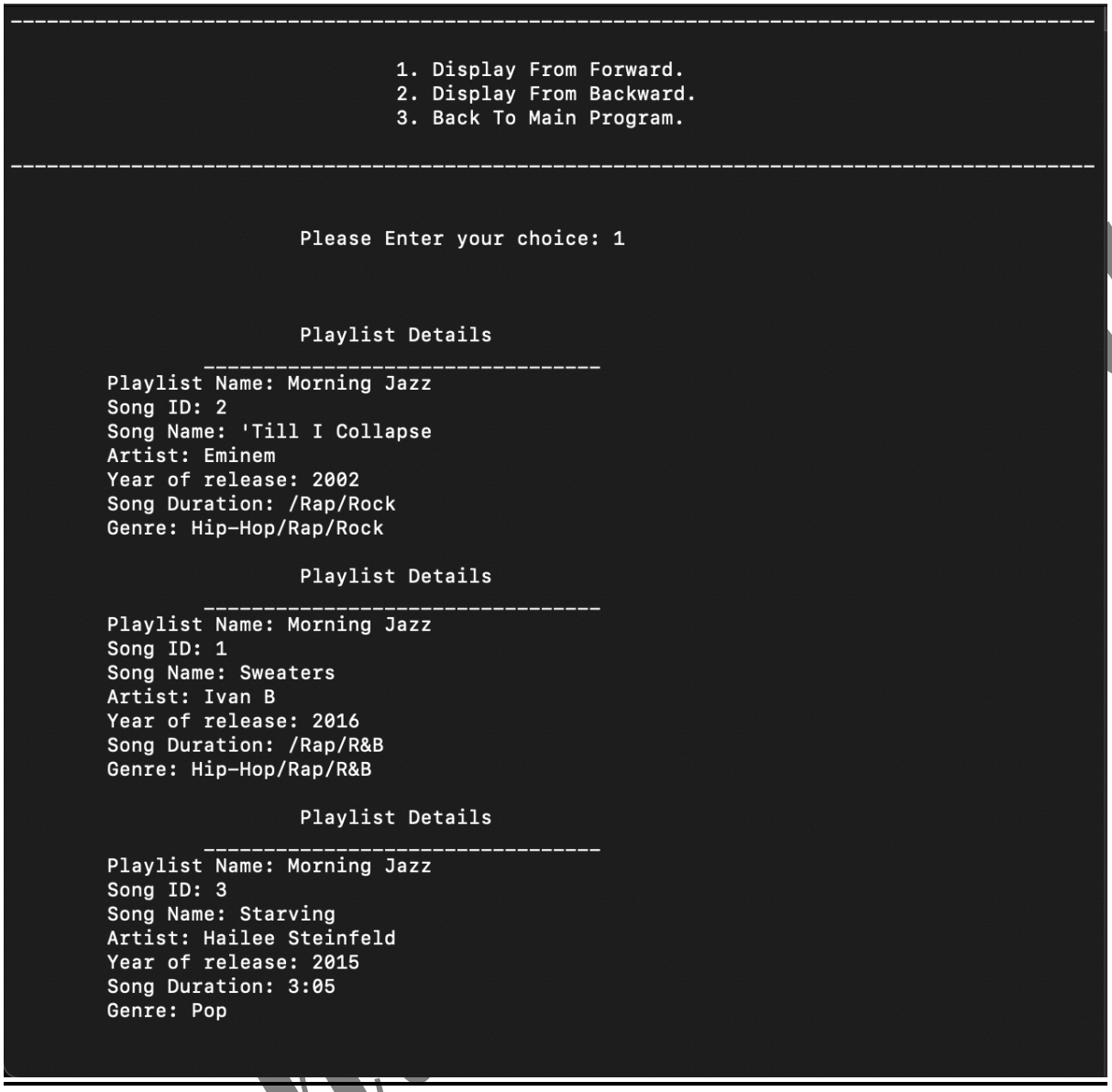

1. Create Playlist Record.
2. Add Playlist Record.
3. Display Playlist Record.
4. Search Playlist Record.
5. Delete Playlist Record.
6. Length of the Playlist.
7. Exit.
Please Enter your choice: 2
1. Add Record At First.
2. Add Record At Last.
3. Back To Main Program.


Playlist Details
Enter Playlist Name: Morning Jazz
Song ID: 2
Song Name: 'Till I Collapse
Artist: Eminem
Year of release: 2002
Song Duration: 4:58
Genre: Hip-Hop/Rap/Rock
Record Added successfully.

```
                    Playlist Details
            --------------------------------

       Enter Playlist Name: Morning Jazz

       Song ID: 3

       Song Name: Starving

       Artist: Hailee Steinfeld

       Year of release: 2015

       Song Duration: 3:05

       Genre: Pop




                    Record Added successfully.
```

Adding two more songs, one from forward and another from backward.

```
--------------------------------------------------------------------------------

                            1. Create Playlist Record.
                            2. Add Playlist Record.
                            3. Display Playlist Record.
                            4. Search Playlist Record.
                            5. Delete Playlist Record.
                            6. Length of the Playlist.
                            7. Exit.

--------------------------------------------------------------------------------
                    Please Enter your choice: 3




--------------------------------------------------------------------------------

                            1. Display From Forward.
                            2. Display From Backward.
                            3. Back To Main Program.

--------------------------------------------------------------------------------
```



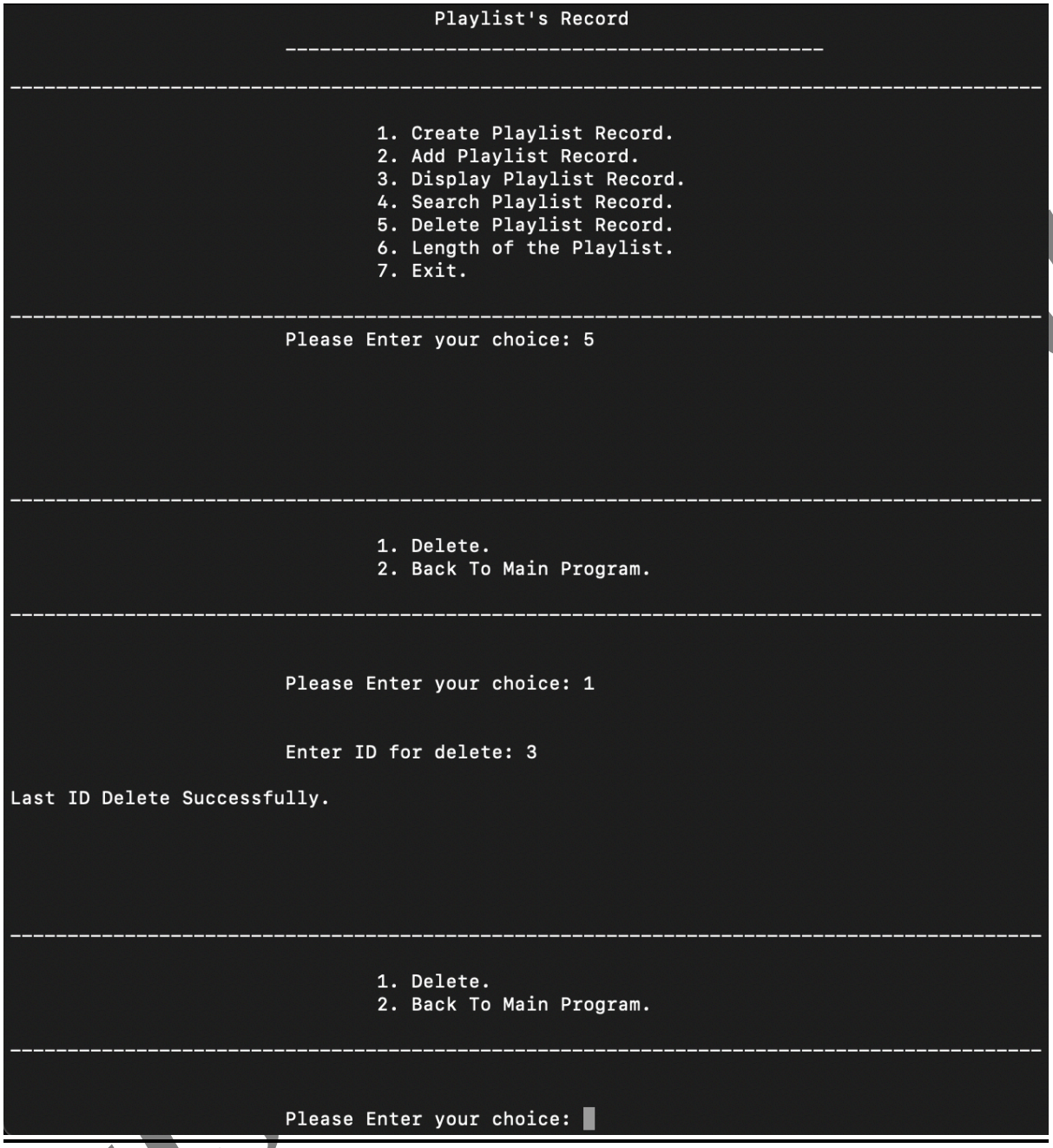
Displaying record from forward.

```
--------------------------------------------------------------------------------

                            1. Display From Forward.
                            2. Display From Backward.
                            3. Back To Main Program.

--------------------------------------------------------------------------------


                        Please Enter your choice: 2



                        Playlist Details
                    ---------------------------------
        Playlist Name: Morning Jazz
        Song ID: 3
        Song Name: Starving
        Artist: Hailee Steinfeld
        Year of release: 2015
        Song Duration: 3:05
        Genre: Pop

                        Playlist Details
                    ---------------------------------
        Playlist Name: Morning Jazz
        Song ID: 1
        Song Name: Sweaters
        Artist: Ivan B
        Year of release: 2016
        Song Duration: /Rap/R&B
        Genre: Hip-Hop/Rap/R&B

                        Playlist Details
                    ---------------------------------
        Playlist Name: Morning Jazz
        Song ID: 2
        Song Name: 'Till I Collapse
        Artist: Eminem
        Year of release: 2002
        Song Duration: /Rap/Rock
        Genre: Hip-Hop/Rap/Rock




--------------------------------------------------------------------------------
```

Displaying record from backward.

```
--------------------------------------------------------------------------------

                              1. Create Playlist Record.
                              2. Add Playlist Record.
                              3. Display Playlist Record.
                              4. Search Playlist Record.
                              5. Delete Playlist Record.
                              6. Length of the Playlist.
                              7. Exit.

--------------------------------------------------------------------------------
                         Please Enter your choice: 4




--------------------------------------------------------------------------------

                              1. Search using Song ID.
                              2. Back To Main Program.

--------------------------------------------------------------------------------


                         Please Enter your choice: 1




                         Enter Song ID: 2

                         Playlist's Details
                 ---------------------------------
       Playlist Name: Morning Jazz
       Song ID: 2
       Song Name: 'Till I Collapse
       Artist: Eminem
       Year of release: 2002
       Song Duration: /Rap/Rock
       Genre: Hip-Hop/Rap/Rock
```

Searching for song using its song id. In this case searching for a song containing song id 2.



Deleting a song using its song id. In this case, song containing song id 3 has been deleted.

```
                                Playlist's Record
                        ------------------------------------------

--------------------------------------------------------------------------------------------

                                1. Create Playlist Record.
                                2. Add Playlist Record.
                                3. Display Playlist Record.
                                4. Search Playlist Record.
                                5. Delete Playlist Record.
                                6. Length of the Playlist.
                                7. Exit.

--------------------------------------------------------------------------------------------
                        Please Enter your choice: 6









                        There is/are 2 song(s) in the playlist.
                                    Thereforth the length is 2.
```

Checking the length of existing record in the program. In this case only 2 songs are in the record since we created playlist with 1 song, then added 2 songs and at last deleted one song. That leaves us with 2 songs remaining in the record.

```
                        ------------------------------------------

--------------------------------------------------------------------------------------------

                                1. Create Playlist Record.
                                2. Add Playlist Record.
                                3. Display Playlist Record.
                                4. Search Playlist Record.
                                5. Delete Playlist Record.
                                6. Length of the Playlist.
                                7. Exit.

--------------------------------------------------------------------------------------------
                        Please Enter your choice: 7

vrushil@VRUSHILs-MacBook-Air Groovy %
```

Exiting the program.

6. CONCLUSION

In conclusion, I would like to point that remembering and storing information about all songs can be a very hectic, irritating and sometimes a very time consuming task. We might need to modify our playlist by adding more songs, deleting the one we are bored listening to or just searching for the right one. There this mini project of playlist manager comes in handy to access all the info you need from the records.

7. REFERENCE

https://www.geeksforgeeks.org/menu-driven-program-for-all-operations-on-doubly-linked-list-in-c/

https://www.geeksforgeeks.org/pointers-in-c-and-c-set-1-introduction-arithmetic-and-array/?ref=gcse

https://www.javatpoint.com/doubly-linked-list

College lecture notes

https://www.javatpoint.com/file-handling-in-c