CHAPTER – 1

INTRODUCTION

1.1 PROBLEM STATEMENT

Create a playlist that is a collection/list of songs. Each song is represented by Song name, Artist and Duration. Basic operations like inserting a song, deleting a song and displaying the playlist need to be included. While displaying the playlist the number of songs and total duration of the playlist should be shown. Along with basic operations, the following features need to be added:

1. Search for a song
2. Sort the playlist by Song name, Artist and Duration
3. Reverse the playlist
4. Shuffle the playlist in random order
5. Create a music library of different genres and the user should be able to select a particular song and add it to the playlist
6. Play option to go through the songs
7. While a song is playing the background color should keep changing
8. Save option to store the contents of the playlist in a separate file
9. Open option to retrieve a saved playlist

1.2 OBJECTIVES

Here is the list of actions that need to be accomplished -

1. Choose the appropriate data structure/s to implement the playlist and learn about the chosen data structure/s.
2. Design the overall algorithm/flowchart of the program.
3. Implement basic functionalities step by step: For insert and delete provide options for the user to choose a specific position to insert/delete the song. For display use a proper format for different headings and show the number of songs and total duration
4. Implement the extended functionalities mentioned in the problem statement.
5. Create a music library consisting of various music genres by creating different text files which the user can view and append songs to the playlist.
6. Implement a play option to go through the songs in the playlist.
7. Change the background color while a song is playing.
8. Save the contents of the playlist to a file and retrieve a saved playlist from a file.
9. Keep testing the program after every step.

1.3 METHODOLOGY

A menu-driven interface using switch case statements will be followed. Memory will be allocated at run time for the playlist. Based on the user's choice from the menu the respective function will be executed. For this project, a singly linked list has been used to implement the playlist wherein each node represents a song and its attributes. Arrays are used to store various names like song names and file names. The concept of threads has been used to change the background color simultaneously as a song is playing. File handling concepts were used to search for songs, read, and write to files.

1.4 EXPECTED OUTCOMES

1. The user gets to manage his/her songs in a proper playlist.
2. The playlist can be made offline and retrieved when the application is run again.
3. User saves time in searching for songs.
4. User gets to create special melodies by combining specific songs in the playlist.
5. Get an enhanced experience with a background colour change.

1.5 HARDWARE AND SOFTWARE REQUIREMENTS

Hardware requirements -
A computer
Dual-core 2Ghz processor recommended for optimal performance
At least 1GB of Storage
2GB RAM

Software requirements –
An IDE to develop C code (like Dev C++)
Preferably a Windows Operating system
Text files for Music library.

CHAPTER 2

DATA STRUCTURES

2.1 LINKED LIST

The items of a linked list are stored at non-contiguous memory addresses and are arranged in a linear data structure without a defined size.

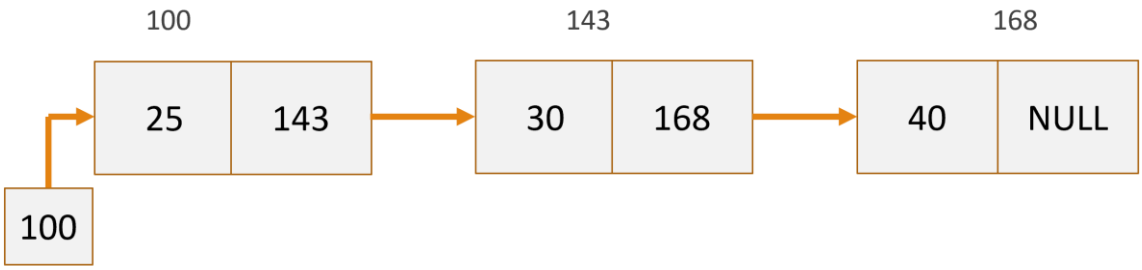

Fig. 2.1: Linked-list representation

The preceding is a logical illustration of a singly linked list. Each element is represented by a node, which has data and a reference to the node after it. The Head pointer is a reference to the first element. The final node's connection is NULL, indicating that the linked list has ended.

In this project, the entire playlist is represented by a singly linked list. Each node of the linked list has three data fields namely the Song name, Artist, duration and pointer to the next song.

Features of a Linked list:

- **Cost of Accessing an Element**: In a linked list, direct access is not feasible; instead, we must traverse the current reference to the requested element. The number of nodes affects how easily you can access the items.

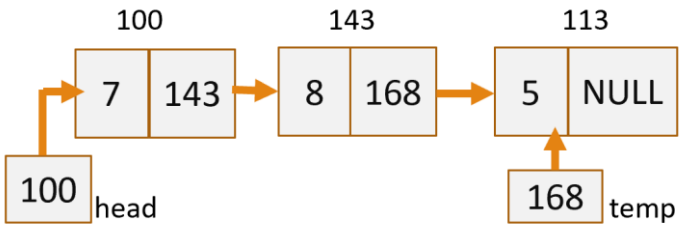

Fig. 2.2: Linked-list accessing

Here to access 5, we had to traverse temp all the way to 5 and then use the arrow operator (temp->data) to access 5.

Time complexity = O(n)

- **Memory requirement**: More in Linked list as extra memory is needed for pointer variable.
  E.g. In Linked list for 7 elements = 7*8 = 56 bytes
- **Memory Utilization**: More efficient in LL as memory is allocated dynamically.

- **Cost of insertion**:

At the beginning – Just update the links of head pointer and new node. Time – O(1)

At i^{th} position – Depending on which position we want to insert we traverse the pointer up to the required position. Time – O(n) for worst case

At end - Traverse the LL from beginning to the end to insert element. Only sequential access is possible in Linked list. Time – O(n)

- **Cost of deletion**: Same as cost of insertion
- **Easy to use**: Linked list is more complicated.
- **Searching**: Only linear search is possible

2.2 ARRAYS

A linear data structure called an array is a fixed-size data structure in which the elements of same data type are stored at adjacent memory locations. The address of the first element is signified by the array name.

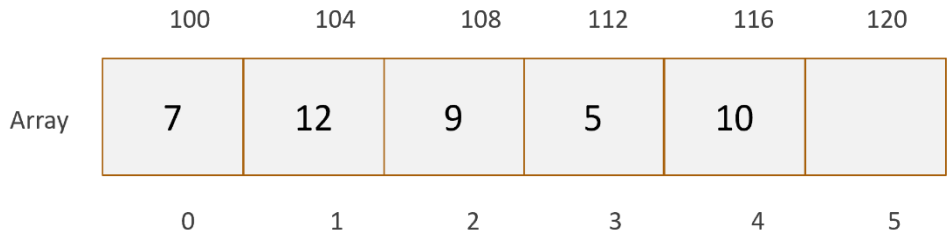

Fig. 2.3: Array representation

An array having a fixed size of 6 and a starting index of 0 is logically represented in the image above.

In this project, Arrays have been used to store Song names, artist names and file names.

Features of Arrays:

- **Cost of Accessing an element**: Since arrays allow for random access, elements can be accessed more quickly.

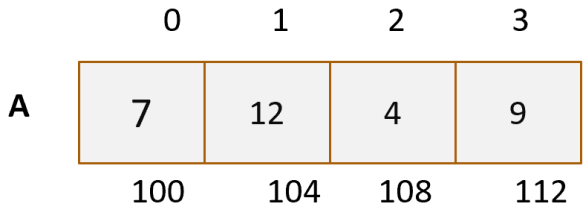

Fig 2.4 Array accessing

A[2] = base address + index value*size of data type
= 100 + 2*4=108

A[2] = 4

Time complexity = O(1)

- **Memory requirement**: Lesser in Array
  Eg: In Array for 7 elements = 28 bytes

- **Memory Utilization**: Inefficient in array as fixed size should be specified in advance.

- **Cost of insertion**:
  At the beginning - All the elements have to be shifted by one memory address to insert an element. Time complexity – O(n)
  At i^{th} position – Depending on which position we want to insert we shift the elements accordingly. Time complexity – O(n-p) ; p-position
  At end - We can directly insert at end. Time - O(1)

- **Cost of deletion**: Same as cost of insertion
- **Easy to use**: Array is easier to use.
- **Searching**: Easier in array. Linear and binary search is possible.

CHAPTER 3

DESIGN

3.1 DESIGN GOALS

Usability: The playlist should have an intuitive user interface and be simple to use.
Scalability: The playlist should be able to handle a large number of songs and make it simple to add and remove them.
Performance: The playlist should perform efficiently and quickly, with a fast search and retrieval time for songs.
Reliability: The playlist should offer a consistent user experience, be robust and stable, with few bugs and mistakes.

3.2 ALGORITHM/PSEUDO-CODE

Step-1: Start

Step-2: Display the menu of options to the user.

Step-3: Read the user's choice.

Step-4: Using a switch statement, execute the corresponding function for the selected option.

a. If the user selects "Create," call the create function.

4a.1 Allocate new memory block for a new node.
4a.2 Read the song name, artist name, and duration of the song from the user.
4a.3 Store the song name, artist name, and duration of the song in the new node.
4a.4 Set the link to next node of the new node to 0.
4a.5 If the linked list is empty (head is equal to 0), set the head and a temporary pointer to the new node.
4a.6 If the linked list is not empty, set the last node’s next pointer to the new node and update the temporary pointer to the new node.

b. If the user selects "Display," call the display function.

4b.1 Print the respective headings
4b.2 Calculate total number of songs and total duration of playlist.
4b.3 Print the song name, artist and duration
4b.4 While printing song’s duration, If the song's duration (in minutes) is less than 60, print duration in the format "MM:SS". If the song's duration is more than 60 minutes, calculate the hours and minutes by dividing the duration by 60 and print duration in the format "HH:MM:SS".

c. If the user selects "Insert," call the insert function.

4c.1 If List is empty, ask user to first create a song.
4c.2 If List is not empty, ask user whether he wants to insert song at beginning,

specific position or end.
4c.3 Allocate new memory block for new node.
4c.4 Read song name, artist and duration.
4c.5 Based on user's choice of position insert the song in the respective position.

d. If the user selects "Delete," call the del function.

4d.1 Similar to insert function, but here you have to free the node to be deleted based on user's choice.

e. If the user selects "Search," call the search function and display the result.

4e.1 Read the song name
4e.2 Traverse through the playlist and if match is found return position of song
4e.3 If match not found return -1.

f. If the user selects "Sort," call the sort function.

4f.1 Check if the linked list is empty, if yes, then display a message "First create a song".
4f.2 If the linked list is not empty, then display the sorting options: sort by song name, sort by artist name, and sort by duration.
4f.3 Read the user's choice for sorting.
4f.4 Based on whether the user chooses to sort by song name, artist or duration implement a bubble sort algorithm to sort the linked list by the song name, artist or duration.

g. If the user selects "Reverse," call the reverse function.

4g.1 Check if the head node is NULL. If it is, return a message that the list is empty.
4g.2 Initialize three pointers: prevnode to 0, currnode and nextnode to the head pointer of the list
4g.3 Enter a while loop, which continues until nextnode is not equal to 0 (end of the list has been reached).
4g.3.1 Within the loop:
4g3.2 Update nextnode pointer to the next node in the list.
4g3.3 Update currnode's next pointer to point to the previous node (prevnode).
4g3.4 Update prevnode to the current node (currnode).
4g3.5 Update currnode to the next node (nextnode).
4g.4 After the loop ends, update the head of the list to prevnode, which is the last node in the original list and is now the first node in the reversed list.

h. If the user selects "Play," call the play function.

4h.1 Read the user's choice of starting position for playing the songs.
4h.2 If the choice is to start from the beginning of the list, set a temporary pointer to the head of the list.

4h.3 If the choice is to start from a specific song, call the search function to get the position of the song.
4h.4 In both cases, create a new thread to run the colorchange function.
4h.5 Traverse the linked list and print the details of each song, until the end of the list is reached.
4h.6 In the colorchange function, change the terminal's background color repeatedly using the "system" function.

i. If the user selects "Shuffle," call the shuffle function.

4i.1 Check if the head node is NULL. If it is, return.
4i.2 The splithalf() function is called next, which takes the head node of the list as input and returns the node in the middle of the list
4i.2.1 The splithalf() function first calculates the number of nodes in the list, and then iterates through the list to find the node in the middle.
4i.2.3 The next pointer of the node before the middle node is set to NULL, effectively splitting the list into two halves.
4i.2.4 The middle node is then returned as the result.
4i.3 The makenew() function is called next, which takes two nodes as input and shuffles the elements of the linked list.
4i.3.1 The function uses two pointers: first and second, to traverse the two halves of the list.
4i.3.2 The elements are then rearranged in alternating order by updating the next pointers of the nodes.
4i.4 Finally, the head of the list is updated to the result of splithalf().

j. If the user selects "Display Music Library," call the displib function.

4j.1 The user is asked to choose a genre of music.
4j.2 The respective file containing the songs is opened and its contents are shown on the output
4j.3 The user is asked if he/she wants to append a song from the Music library to the playlist
4j.4 If yes, then the user is asked to input line number of song in the Music library
4j.5 The file pointer is set to 0
4j.6 Enter a while loop that continues reading the file line by line
4j.6.1 Check if the reading line number matches with chosen song line number
4j.6.2 If yes, call the string token function to tokenize the array containing the entire line with " " as the delimiter of the string token function
4j.6.3 After tokenizing the array, copy the attributes song name, artist and duration to 3 different character arrays.
4j.7 A new node is created and the respective attributes are appended to the new node of the playlist.
4j.8 The user is asked to choose where to insert the new song – beg, specific position or end and the song is appended accordingly to the playlist.

k. If the user selects "Save," call the save function.

4k.1 Get the name of the file from the user by printing the message "Enter file name (*.txt): " and reading the input using scanf.
4k.2 Open the file in write mode by calling fopen function with the filename and the "w" mode.
4k.3 If the file cannot be opened, print the message "Couldn't Open File" and exit the program.
4k.4 Write the header "SONG NAME", "ARTIST", and "DURATION" to the file by using the fprintf function.
4k.5 Traverse the linked list and write the details of each node to the file in the format "song name", "artist", and "duration". The duration is formatted as a string with minutes and seconds.
4k.6 If the data is written successfully to the file, print the message "Play List stored in the file successfully". Else, print "Error While Writing".
4k.7 Close the file using the fclose function.

l. If the user selects "Open," call the open function.

4l.1 The function open takes a file name as input and opens it in read mode.
4l.2 If the file is not found, it displays "Cannot open file".
4l.3 The file pointer is set to the exact position from which we want to read the file
4l.4 While The file is read line by line and stored in an array buff -
4l.4.1 The array which contains an individual line is tokenized to extract song name, artist and duration.
4l.4.2 The duration is further tokenized to separate the minutes and seconds. These values are then converted to integers using the atoi function and stored in variables min and sec.
4l.4.3 A new node is created and the song name, artist, minutes, and seconds are stored in it and inserted to the end of the playlist.
4l.5 The file is then closed.

m. If the user selects "Exit," exit the program.

Step-5: Repeat steps 2 to 4 until the user selects "Exit."

3.3 FLOWCHART

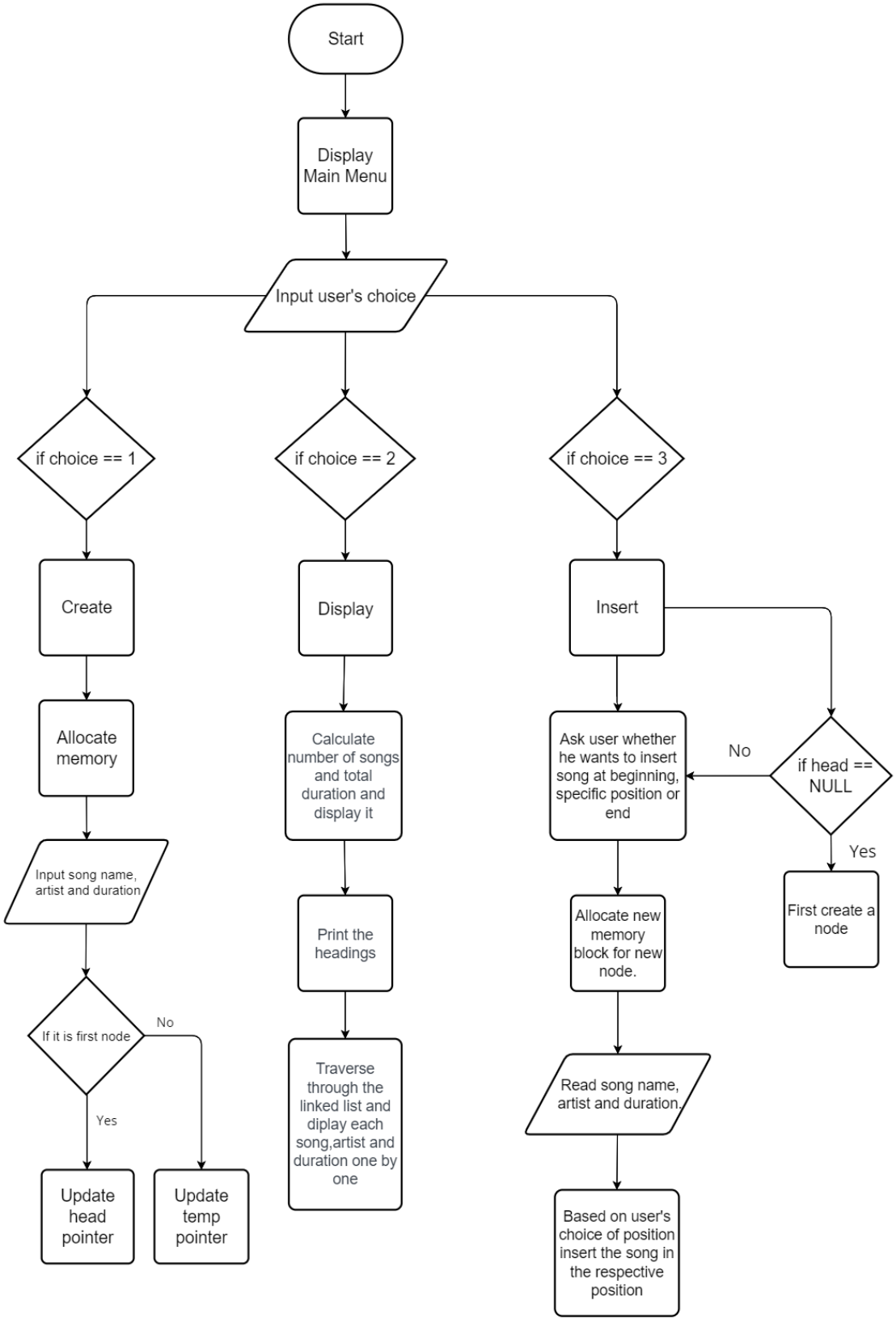

Fig 3.1 – Flowchart 1

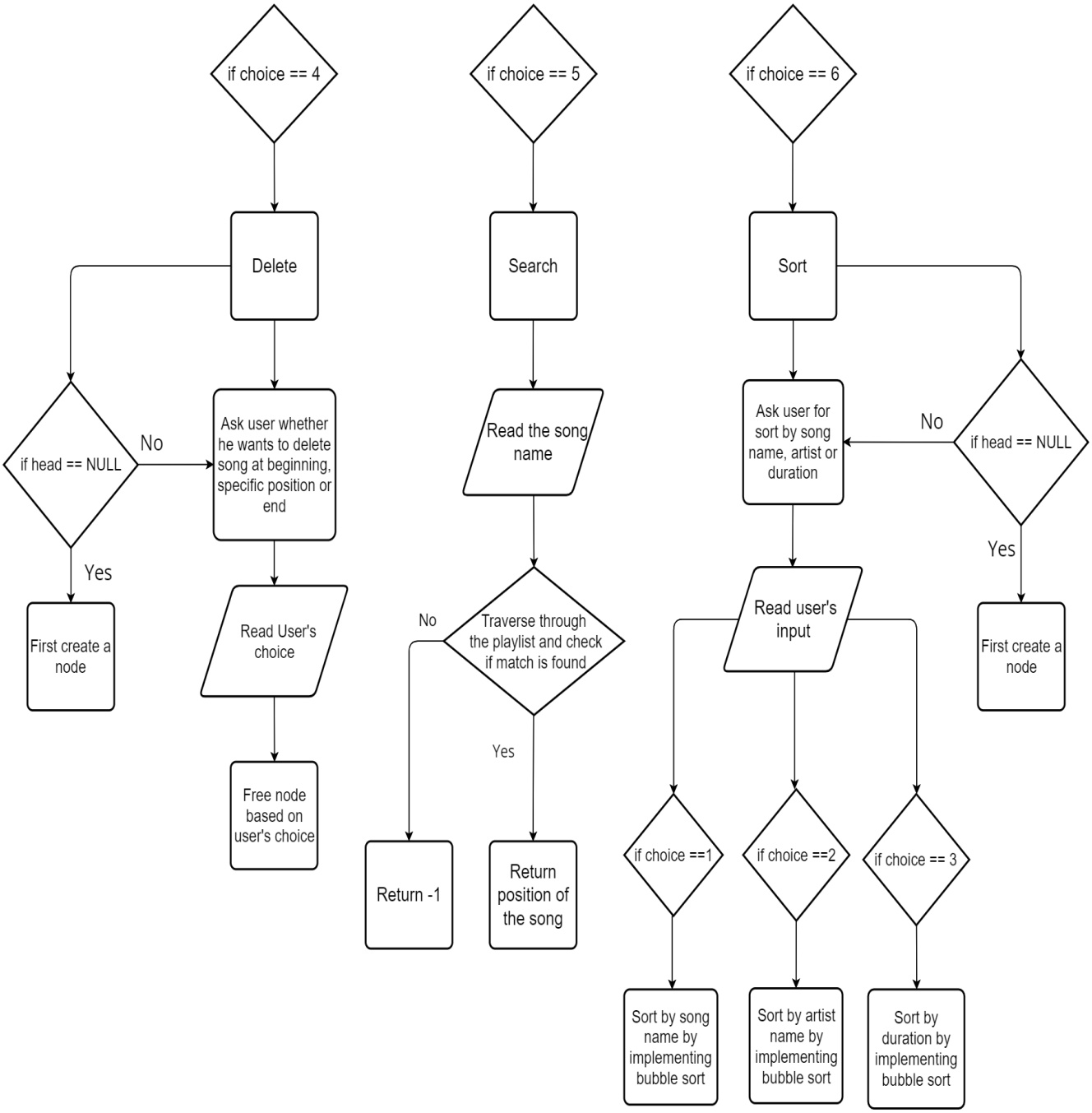

Fig 3.2 – Flowchart 2

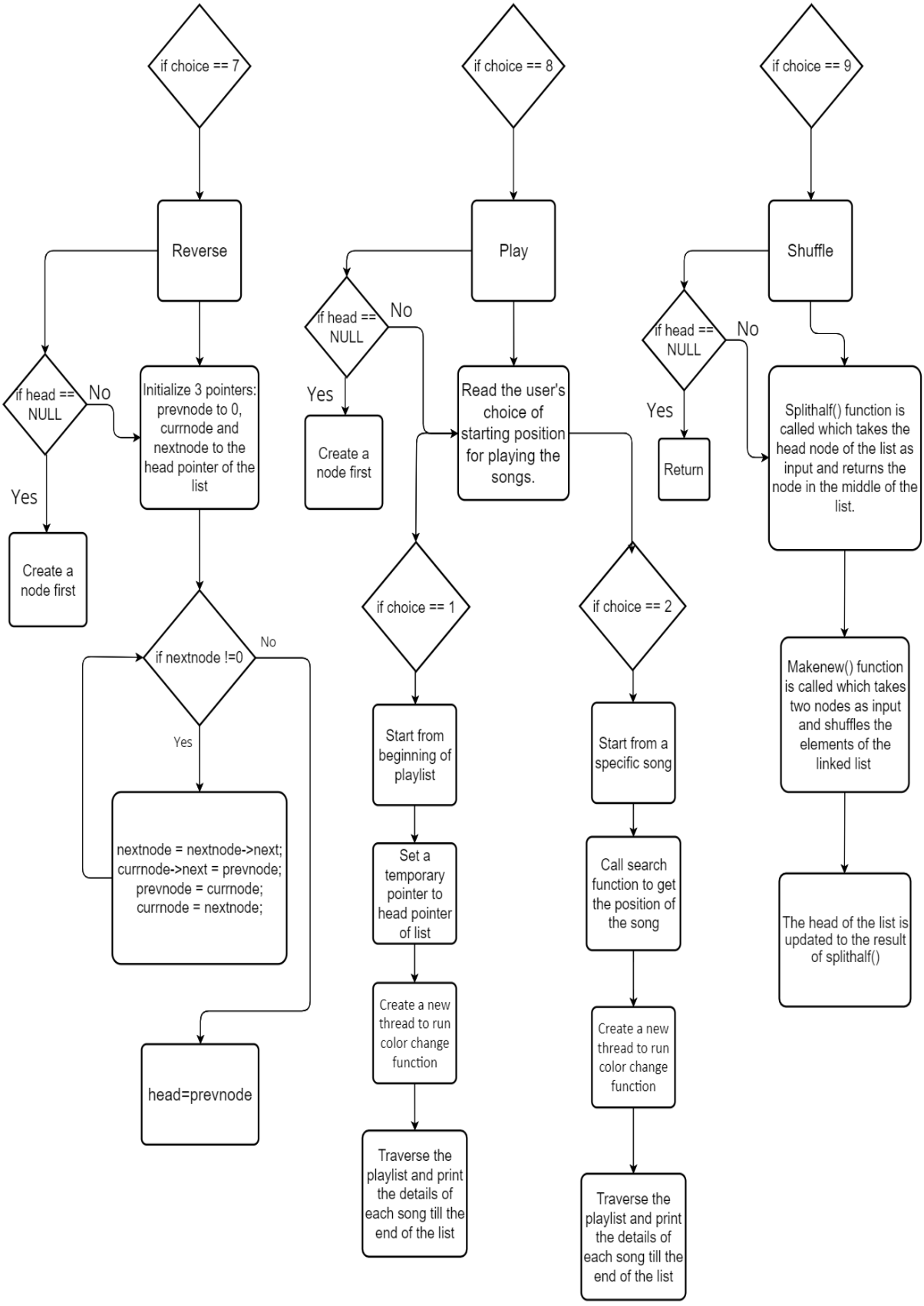

Fig 3.3 – Flowchart 3

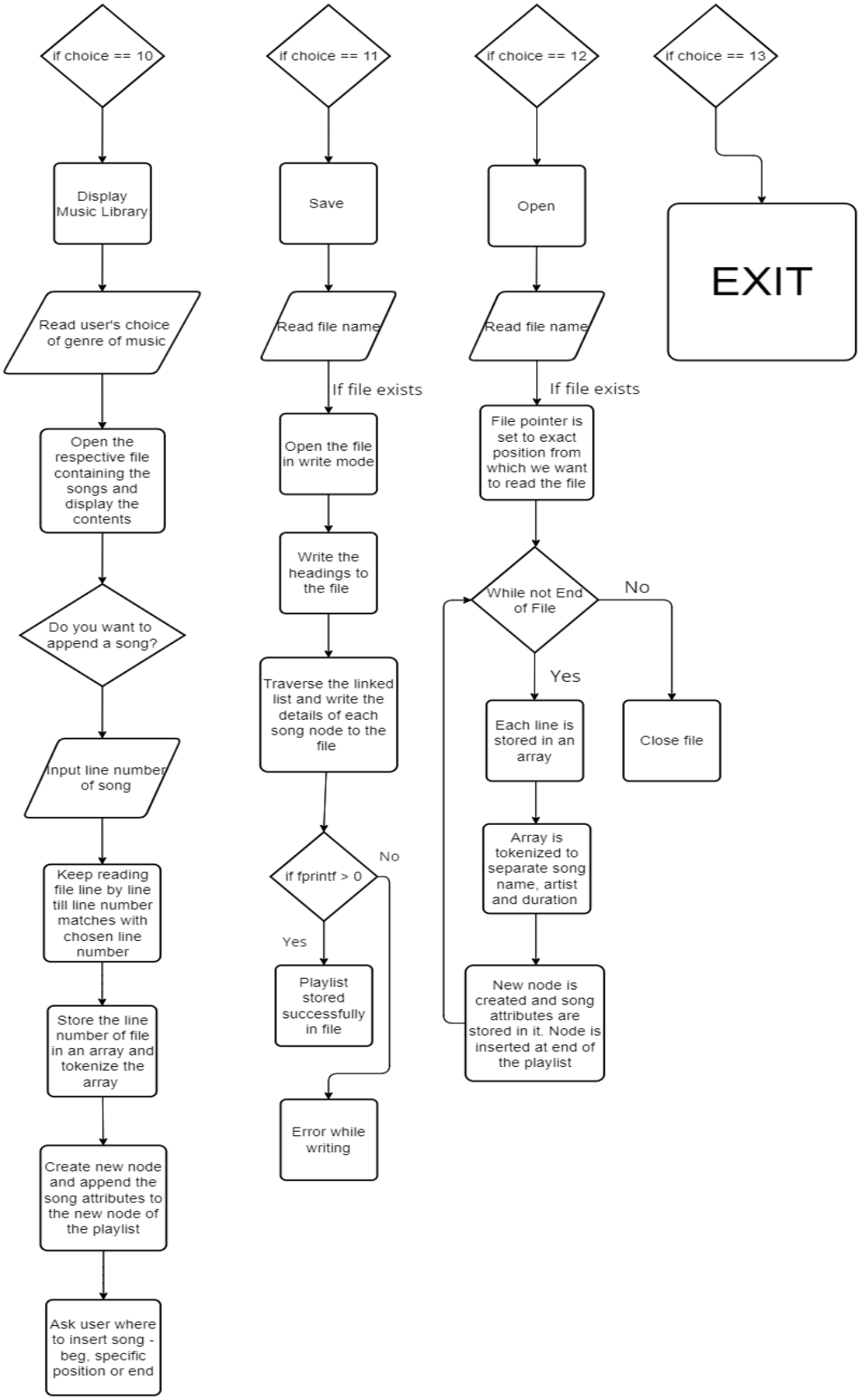

Fig 3.4 – Flowchart 4

CHAPTER 4

IMPLEMENTATION

4.1 MODULE 1 - FUNCTIONALITY

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4  #include <math.h>
5  #include <pthread.h>
6
7  struct node {
8      char songname[50];
9      int minutes;
10     int seconds;
11     char artist[50];
12     struct node* next;
13 };
14
15 typedef struct node node;
16 node* head = 0, * temp = 0, * newnode = 0,*prevnode=0,*nextnode=0,*p=0,*q=0,*r,*s,*tmp,*currnode;
```

Fig 4.1 – Code 1

This module enlists the header files needed for the project and the structure of each song node of the linked list. String.h is used to perform string functions namely strcpy, strcat, strcmp and strtok. Math.h is used for floor and ceil functions while displaying duration if the songs. Pthread.h is used to create a thread to run background color change function parallelly while play function is running. The struct node has the attributes of each song and the link to the next song. Various node pointers are used.

4.2 MODULE 2 - FUNCTIONALITY

```
18 void create() {
19     struct node * newnode = (node*)malloc(sizeof(node));
20     printf("\nEnter song name: ");
21     //gets(song);
22     scanf("%s", &newnode->songname);
23     printf("\nEnter artist name: ");
24     scanf("%s",&newnode->artist);
25     do{
26     printf("\nEnter the duration of the song (MM:SS) : ");
27     scanf("%d:%d",&newnode->minutes,&newnode->seconds);
28     }while(newnode->seconds<0 || newnode->seconds>60);
29     newnode->next = 0;
30     if (head == 0) {
31         head = newnode;
32         temp = newnode;
33     }
34     else {
35         temp->next = newnode;
36         temp = newnode;
37     }
38 }
```

Fig 4.2 – Code 2

```c
void insert() {
   int choice, i = 1, pos;
   if(head==NULL){
      printf("\nFirst create a song ");
   }else{
   printf("\n1.Insert at beg\n2.Insert at end\n3.Insert at a particular position");
   scanf("%d", &choice);
   newnode = (node*)malloc(sizeof(node));
   printf("\nEnter songname\n");
   scanf("%s", newnode->songname);
   printf("\nEnter artist name: ");
   scanf("%s",&newnode->artist);
   do{
   printf("\nEnter the duration of the song (MM:SS): ");
   scanf("%d:%d",&newnode->minutes,&newnode->seconds);
   }while(newnode->seconds<0 || newnode->seconds>60);
   switch (choice) {
   case 1:
      newnode->next = head;
      head = newnode;
      break;
   case 2:
      newnode->next = 0;
      temp = head;
      while (temp->next != 0) {
         temp = temp->next;
      }
      temp->next = newnode;
      break;
   case 3:
      printf("\nEnter position at which element is to be inserted:\n");
      scanf("%d", &pos);
      temp = head;
      while (i < pos - 1) {
         temp = temp->next;
         i++;
      }
      newnode->next = temp->next;
      temp->next = newnode;
      break;
   default:
      printf("\nEnter Valid option\n");
      break;
   }
   }
}
```

Fig 4.3 – Code 3

```c
void del() {
   int pos=1,i=1,choice;
   if(head==NULL){
      printf("\nFirst create a song ");
   }else{
   printf("\n1.Delete from beg\n2.Delete from end\n3.Delete from specific position\n");
   scanf("%d",&choice);
   switch (choice) {
   case 1:
      if (head != 0) {
         temp = head;
         head = head->next;
         free(temp);
      }
      else {
         printf("\nList is empty");
      }
      break;
   case 2:
      temp = head;
      if(head!=0){
      while (temp->next != 0) {
         prevnode = temp;
         temp = temp->next;
      }
      if (head->next == 0) {
         head = 0;
      }
      else {
         prevnode->next = 0;
      }
      free(temp);
      }
      break;
   case 3:
      temp = head;
      printf("\nEnter the position:\n");
      scanf("%d",&pos);
      while (i < pos - 1) {
         temp = temp->next;
         i++;
      }
      nextnode = temp->next;
      temp->next = nextnode->next;
      free(nextnode);
      break;
   default:
      printf("\nInvalid choice\n");
   }
}
}
```

Fig 4.4 – Code 4

```c
void getlength(){
    int count=0;
    temp=head;
    if(head==NULL){
        printf("\nPlaylist is empty");
    }else{
    while(temp!=0){
        count++;
        temp=temp->next;
    }
    if(count==1){
    printf("\n1 song");
    }
    else{
    printf("\n%-1d songs",count);
    }
}
}
void getduration(){
    float totaltime, hours, totalseconds;
    int m=0,s=0,totalmin,sec, totalsec, totalhours,min,minute;
    int totaldurationh=0, totaldurationm=0,totaldurations=0;
    temp=head;

    if(head==NULL){
      printf("%60s\n","0 min 0sec\n");
    }else{
    while(temp!=0){
        m+=temp->minutes;
        s+=temp->seconds;
        temp=temp->next;
    }

    sec = (m*60) + s;
    totaltime =(float)sec/60;
    totalmin = (int)totaltime;

     if(totalmin<60){
     totalseconds = (totaltime-totalmin)*60;
     if((totalseconds-(int)totalseconds) < 0.5)
     totalsec = floor(totalseconds);
     else
     totalsec = ceil(totalseconds);
     printf("%60d min %d sec\n",totalmin, totalsec);
     }
     else
     {
        hours = (float)totalmin/60;
        totalhours = (int)hours;
        min = (hours-totalhours)*60;
        if((min - (int)min)<0.5)
        minute = floor(min);
        else
        minute = ceil(min);
        printf("%60d hr %d min\n",totalhours, minute);
    }
}
}
```

Fig 4.5 – Code 5

```c
void display() {
    int i;
    float h,m;
    int hours,min;
    for(i=0;i<78;i++)
    printf("%s","=");
    printf("\n\nYOUR PLAYLIST\n");
    getlength();
    getduration();
    for(i=0;i<78;i++)
    printf("%s","=");
    printf("\n%-30s %-30s %13s\n","SONG NAME","ARTIST","DURATION");
    for(i=0;i<78;i++)
    printf("%s","-");
    printf("\n");
    temp=head;
    while (temp != 0) {
        if(temp->minutes<60){
        printf("%-30s %-30s %4s %2c %02d:%02d", temp->songname,
        temp->artist, "|",' ',temp->minutes,temp->seconds);
        }else{
        if(temp->minutes>=60){
        h=(float)(temp->minutes)/60;
        hours=(int)h;
        m=(h-hours)*60;
        }

        if((m-(int)m)<0.5)
        min=floor(m);
        else
        min=ceil(m);

        printf("%-30s %-30s %4s %02d:%02d:%02d", temp->songname,
        temp->artist, "|",hours, min ,temp->seconds);
        }
        temp = temp->next;
        printf("\n");
    }
}
```

Fig 4.6 – Code 6

Module – 2 enlists the primitive functions of the playlist such as Create, Insert, Delete and Display. It is necessary that a node should be first created before using insert or delete functions. Update the links in these functions based on choice of position – beg, specific position or end. In display function, additional getlength() and getduration() functions are used to get the total number of songs and total duration of the playlist.

4.3 MODULE 3 - FUNCTIONALITY

```
238  int search()
239  {
240      char song[50];
241      int pos=1;
242      printf("Enter song name\n");
243      scanf("%s", song);
244      temp = head;
245      while (temp != NULL)
246      {
247          if (strcmp(temp->songname, song) != 0) {
248              temp = temp->next;
249              pos++;
250          }
251          else
252          {
253              return pos;
254          }
255      }
256      return -1;
257  }
```

Fig 4.7 – Code 7

```
259  void sort(){
260      int choice;
261      if(head==NULL){
262          printf("\nFirst create a song ");
263      }else{
264      printf("\n1. Sort by Song name ");
265      printf("\n2. Sort by Artist name");
266      printf("\n3. Sort by Duration");
267      choice =scanf("%d",&choice);
268      if(choice == 1){
269      for(r=p=head; p->next!=NULL; r=p,p=p->next){
270          for(s=q=p->next; q!=NULL; s=q,q=q->next){
271              if(strcmp(p->songname,q->songname)>0){
272                  tmp=p->next;
273                  p->next=q->next;
274                  q->next=tmp;
275                  if(p!=head)
276                  r->next=q;
277                  s->next=p;
278                  if(p==head)
279                  head=q;
280                  tmp=p;
281                  p=q;
282                  q=tmp;
283              }
284          }
285      }
286      }else if(choice == 2){
287      for(r=p=head; p->next!=NULL; r=p,p=p->next){
288          for(s=q=p->next; q!=NULL; s=q,q=q->next){
289              if(strcmp(p->artist,q->artist)>0){
290                  tmp=p->next;
291                  p->next=q->next;
292                  q->next=tmp;
293                  if(p!=head)
294                  r->next=q;
295                  s->next=p;
296                  if(p==head)
297                  head=q;
298                  tmp=p;
299                  p=q;
300                  q=tmp;
301              }
302          }
303      }
```

Fig 4.8 – Code 8

```
304     }else if(choice == 3){
305      for(r=p=head; p->next!=NULL; r=p,p=p->next){
306         for(s=q=p->next; q!=NULL; s=q,q=q->next){
307            if(p->minutes > q->minutes){
308               tmp=p->next;
309               p->next=q->next;
310               q->next=tmp;
311               if(p!=head)
312               r->next=q;
313               s->next=p;
314               if(p==head)
315               head=q;
316               tmp=p;
317               p=q;
318               q=tmp;
319            }
320         }
321      }
322   }
323   }
324   }
326   void reverse(){
327      if(head==NULL){
328         printf("\nList is empty\n");
329      }
330      prevnode=0;
331      currnode=nextnode=head;
332      while(nextnode!=0){
333         nextnode = nextnode->next;
334         currnode->next = prevnode;
335         prevnode = currnode;
336         currnode = nextnode;
337      }
338      head=prevnode;
339   }
340
```

Fig 4.9 – Code 9

Module 3 – enlists the search, sort and reverse functions. Search function returns the position of the song if found. Sort function uses bubble sort algorithm by swapping values bases ASCII values of the characters, and arranges in ascending order by songname, artist or duration based on use's choice. Reverse function reverses the order of the playlist.

4.4 MODULE 4 - FUNCTIONALITY

```
341 void displib(){
342    int choice,ch,i=0;
343    FILE *fp;
344    char buffer[400];
345    printf("\nChoose Genre: ");
346    printf("\n1. Classical ");
347    printf("\n2. Rock");
348    printf("\n3. Country");
349    printf("\n4. Jazz");
350    printf("\n5. Devotional\n");
351    scanf("%d",&choice);
352    if(fp==NULL){
353        printf("Error\n");
354     }
355    printf("\n");
356    if(choice == 1){
357    fp = fopen("LibClassical.txt","r");
358    while(fgets(buffer,400,fp)){
359    printf("%s",buffer);
360    }
361    }else if(choice == 2){
362    fp = fopen("LibRock.txt","r");
363    while(fgets(buffer,400,fp)){
364    printf("%s",buffer);
365    }
366    }
367    else if(choice == 3){
368     fp = fopen("LibCountry.txt","r");
369    while(fgets(buffer,400,fp)){
370    printf("%s",buffer);
371    }}
372    else if(choice == 4){
373     fp = fopen("LibJazz.txt","r");
374    while(fgets(buffer,400,fp)){
375    printf("%s",buffer);
376    }}else{
377     fp = fopen("LibDevo.txt","r");
378    while(fgets(buffer,400,fp)){
379    printf("%s",buffer);
380    }}
381
382    printf("\nDo you want to append a song? (1 for yes, 0 for no): ");
383    scanf("%d",&ch);
384
```

Fig 4.10 – Code 10

```
386    if(ch==1) {
387
388    char song[50]={};
389    char artist[50]={};
390    char duration[10]={};
391    int min,sec;
392    char buff[60];
393    int readline=0;
394
395    printf("Select line number of song: ");
396    scanf("%d",&readline);
397    fseek(fp,0,SEEK_SET);
398
399    while(fgets(buff,sizeof(buff),fp)){
400          i++;
401          if(i==(readline+1)){
402             const char s[2] = " ";
403             char *token;
404             char num[2];
405
406             token = strtok(buff, s);
407
408             if(token!=NULL)
409               strcat(num,token);
410               token=strtok(NULL," ");
411               strcat(song,token);
412               token=strtok(NULL," ");
413               strcat(artist,token);
414               token=strtok(NULL," ");
415               strcat(duration,token);
416          }
417    }
```

Fig 4.11 – Code 11

```
419   newnode = (node*)malloc(sizeof(node));
420   strcpy(newnode->songname, song);
421   strcpy(newnode->artist, artist);
422
423   char m[3]={};
424   char s[3]={};
425
426   char *token2;
427
428   token2 = strtok(duration,":");
429   if(token2!=NULL)
430   strcat(m,token2);
431   token2 = strtok(NULL,"\0");
432   strcat(s,token2);
433
434   min = atoi(m);
435   sec = atoi(s);
436
437   newnode->minutes = min;
438   newnode->seconds = sec;
439
440   newnode->next = 0;
441
442    if (head == 0) {
443       head = newnode;
444       temp = newnode;
445    }
446    else {
447     int choi,pos;
448     printf("\n1.Insert at beg\n2.Insert at end\n3.Insert at a particular position");
449     scanf("%d", &choi);
450     switch (choi) {
451       case 1:
452          newnode->next = head;
453          head = newnode;
454          break;
455       case 2:
456          newnode->next = 0;
457          temp = head;
458          while (temp->next != 0) {
459            temp = temp->next;
460          }
461          temp->next = newnode;
463        case 3:
464          printf("\nEnter position at which element is to be inserted:\n");
465          scanf("%d", &pos);
466          temp = head;
467          while (i < pos - 1) {
468              temp = temp->next;
469              i++;
470          }
471          newnode->next = temp->next;
472          temp->next = newnode;
473          break;
474       default:
475          printf("\nEnter Valid option\n");
476          break;
477    }
478  }
479  }
480   fclose(fp);
481  }
482
```

Fig 4.12 – Code 12

```
484 void *colorchange(){
485     int i;
486     for(i=0;i<2;i++){
487     system("COLOR 90");
488     sleep(1);
489     system("COLOR A0");
490     sleep(1);
491     system("COLOR B0");
492     sleep(1);
493     system("COLOR C0");
494     sleep(1);
495     system("COLOR D0");
496     sleep(1);
497     system("COLOR E0");
498     sleep(1);
499     system("COLOR F0");
500     sleep(1);
501     }
502  system("COLOR 07");
503 }
505 void play(){
506     int choice,pos,i=1;
507     char arr[50];
508     pthread_t thread1;
509     if(head==NULL){
510         printf("\nList is empty\n");
511     }
512     printf("\n1. From start ");
513     printf("\n2. Choose from which song ");
514     scanf("%d",&choice);
515     if(choice == 1){
516     temp = head;
517     pthread_create(&thread1,NULL,colorchange,NULL);
518     while(temp!=NULL){
519         printf("\nNow Playing: %-25s %-20s %4s %2c %02d:%02d", temp->songname,
520          temp->artist, "|", ' ',temp->minutes,temp->seconds);
521         getch();
522         temp=temp->next;
523 
524     }
525     }
526     else{
527     pos = search();
528     while(i<pos-1){
529         temp=temp->next;
530         i++;
531     }
532     pthread_create(&thread1,NULL,colorchange,NULL);
533     while(temp!=NULL){
534         printf("\nNow Playing: %-25s %-20s %4s %2c %02d:%02d", temp->songname,
535          temp->artist, "|", ' ',temp->minutes,temp->seconds);
536         getch();
537         temp=temp->next;
538     }
539     }
540 }
```

Fig 4.13 – Code 13

Module 4 consists of the Music Library function and Play function. The Music library opens the file based on the user's choice of genre. Fseek is used to set the file pointer to the start

of the file. Fgets is used to read the file line by line. Strtok is used to tokenize the array containing the line of the file. Atoi function is used to convert a string of characters to an integer. For color change system function is used. Pthread_t thread 1 is a pointer to the thread id of the thread that is created. Pthread_create is a function used to create a new thread. The 1st arg of this function is the id of the thread. The 2nd arg is taken as NULL. The 3rd arg is the function name and the final arg is the value we want to pass to the new thread.

4.5 MODULE 5 - FUNCTIONALITY

```
542 node* splithalf(node* start) {
543  int num = 0,i;
544  node* cur = start;
545 for (cur = start; cur != NULL; cur = cur->next) {
546  num++;
547  }
548
549 for (i = 0; i < num / 2 - 1; i++) {
550  start = start->next;
551  }
552
553  node* result = start->next;
554  start->next = NULL;
555  return result;
556  }
557
558 void makenew(node* first, node* second) {
559  node* last = NULL;
560 while (second != NULL) {
561
562 if (last == NULL) {
563  last = second;
564  } else {
565  last->next = second;
566  }
567
568  node* next = second->next;
569  second->next = first;
570  second = next;
571  last = first;
572
573  next = first->next;
574  first->next = NULL;
575  first = next;
576  }
577  }
578
579 void shuffle(node** head){
580     if (* head == NULL)
581     return;
582     node* half = splithalf(*head);
583     makenew(*head, half);
584     *head = half;
585  }
```

Fig 4.14 – Code 14

```
587 void save(){
588     char filename[30]={};
589     temp = head;
590     printf("Enter file name (*.txt): ");
591     scanf("%s",&filename);
592     FILE* file;
593     file = fopen (filename, "w");
594     if (file == NULL)
595     {
596         printf("\nCouldn't Open File\n");
597         exit (1);
598     }
599
600     fprintf(file, "%-30s %-30s %13s\n","SONG NAME","ARTIST","DURATION\n");
601     while(temp!=NULL)
602     {
603         fprintf(file, "%-30s %-30s %4c %02d:%02d\n", temp->songname,
604         temp->artist,' ',temp->minutes,temp->seconds);
605         temp = temp->next;
606     }
607
608     if(fprintf > 0)
609         printf("Play List stored in the file successfully\n");
610     else
611         printf("Error While Writing\n");
612
613     fclose(file);
614 }
```

Fig 4.15 – Code 15

```
616 void open() {
617
618     char file[20]={};
619     char buff[200]={};
620     FILE *fp;
621
622     printf("\nEnter filename: ");
623     scanf("%s", &file);
624     fp = fopen(file,"r");
625     if(fp == NULL){
626         printf("\nCannot open file\n");
627     }
628
629     fseek(fp,78,SEEK_CUR);
630
631     while(fgets(buff,sizeof(buff),fp)){
632     char song[50]={};
633     char artist[50]={};
634     char duration[10]={};
635     int min,sec,i=1;
636     char m[3]={};
637     char s[3]={};
638     char *token;
639
640     token = strtok(buff, " ");
641
642         if(token!=NULL)
643             strcat(song,token);
644             token=strtok(NULL," ");
645             strcat(artist,token);
646             token=strtok(NULL," ");
647             strcat(duration,token);
648
649
650     char *token2;
651
652     token2 = strtok(duration,":");
653     if(token2!=NULL)
654     strcat(m,token2);
655     token2 = strtok(NULL,"\0");
656     strcat(s,token2);
657
658     min = atoi(m);
659     sec = atoi(s);
660
661     newnode = (node*)malloc(sizeof(node));
662     strcpy(newnode->songname, song);
663     strcpy(newnode->artist, artist);
664
665     newnode->minutes = min;
666     newnode->seconds = sec;
```

Fig 4.16 – Code 16

```
668     if (head == 0) {
669         head = newnode;
670         temp = newnode;
671         newnode->next = 0;
672     }
673     else {
674         newnode->next = 0;
675         temp = head;
676         while (temp->next != 0) {
677             temp = temp->next;
678         }
679         temp->next = newnode;
680     }
681 }
682 fclose(fp);
683 }
```

Fig 4.17 – Code 17

Module 5 consists of the shuffle function, save and open function. The shuffle function calls splithalf to get the middle node address and makenew function to arrange the linked list in alternative order and thereby shuffling the playlist. Save is used to make an offline playlist. In open function, fp is set to offset 78 using the SEEK_CUR constant. The contents of the file are then converted into a linked list and hence the playlist is restored.

4.6 MODULE 6 - FUNCTIONALITY

```
685 int main() {
686
687     int choice,pos;
688
689     while (1) {
690         printf("\n***************MAIN MENU***************\n");
691         printf("\n1. Create");
692         printf("\n2. Display");
693         printf("\n3. Insert");
694         printf("\n4. Delete");
695         printf("\n5. Search");
696         printf("\n6. Sort");
697         printf("\n7. Reverse");
698         printf("\n8. Play");
699         printf("\n9. Shuffle");
700         printf("\n10. Display Music Library");
701         printf("\n11. Save");
702         printf("\n12. Open");
703         printf("\n13. Exit\n");
704         scanf("\n%d", &choice);
705         switch (choice) {
706         case 1:
707             create();
708             break;
709         case 2:
710             display();
711             break;
712         case 3:
713             insert();
714             break;
715         case 4:
716             del();
717             break;
718         case 5:
719             pos = search();
720             if(pos!=-1){
721             printf("\nSong found at position %d",pos);
722             }
723             else{
724                 printf("\nSong not found\n");
725             }
726             break;
727         case 6:
728             sort();
729             break;
730         case 7:
731             reverse();
732             break;
733         case 8:
734             play();
735             break;
```

Fig 4.18 – Code 18

```
736         case 9:
737             shuffle(&head);
738             break;
739         case 10:
740             displib();
741             break;
742         case 11:
743             save();
744             break;
745         case 12:
746             open();
747             break;
748         case 13:
749             exit(0);
750         default:
751             printf("\nEnter valid option: ");
752             break;
753         }
754     }
755     return 0;
756 }
757
```

Fig 4.19 – Code 19

Module 6 entails the main driver function of the Song Playlist. It is a menu driven program which displays the Main Menu and based on User's choice of options the respective functions are executed. Option 13 exits the application with exit status 0.

CHAPTER 5

OUTPUT

```
***************MAIN MENU***************

1.  Create
2.  Display
3.  Insert
4.  Delete
5.  Search
6.  Sort
7.  Reverse
8.  Play
9.  Shuffle
10. Display Music Library
11. Save
12. Open
13. Exit
1

Enter song name: Our_Father

Enter artist name: Don_Moen

Enter the duration of the song (MM:SS) : 5:51
```

Fig 5.1 – Output 1

```
***************MAIN MENU***************

1.  Create
2.  Display
3.  Insert
4.  Delete
5.  Search
6.  Sort
7.  Reverse
8.  Play
9.  Shuffle
10. Display Music Library
11. Save
12. Open
13. Exit
3

1.Insert at beg
2.Insert at end
3.Insert at a particular position2

Enter songname
Joy_to_the_world

Enter artist name: Pentatonix

Enter the duration of the song (MM:SS): 3:59
```

Fig 5.2 – Output 2

```
***************MAIN MENU***************

1.  Create
2.  Display
3.  Insert
4.  Delete
5.  Search
6.  Sort
7.  Reverse
8.  Play
9.  Shuffle
10. Display Music Library
11. Save
12. Open
13. Exit
3

1.Insert at beg
2.Insert at end
3.Insert at a particular position3

Enter songname
Clap_your_hands

Enter artist name: Moe_Bandy

Enter the duration of the song (MM:SS): 1:58

Enter position at which element is to be inserted:
2
```

Fig 5.3 – Output 3

```
***************MAIN MENU***************

1.  Create
2.  Display
3.  Insert
4.  Delete
5.  Search
6.  Sort
7.  Reverse
8.  Play
9.  Shuffle
10. Display Music Library
11. Save
12. Open
13. Exit
2
================================================================================

YOUR PLAYLIST

3 songs                                                      11 min 48 sec
================================================================================
SONG NAME                 ARTIST                                  DURATION
--------------------------------------------------------------------------------
Our_Father                Don_Moen                          |     05:51
Clap_your_hands           Moe_Bandy                         |     01:58
Joy_to_the_world          Pentatonix                        |     03:59
```

Fig 5.4 – Output 4

```
1.  Create
2.  Display
3.  Insert
4.  Delete
5.  Search
6.  Sort
7.  Reverse
8.  Play
9.  Shuffle
10. Display Music Library
11. Save
12. Open
13. Exit
6

1. Sort by Song name
2. Sort by Artist name
3. Sort by Duration1

**************MAIN MENU**************

1.  Create
2.  Display
3.  Insert
4.  Delete
5.  Search
6.  Sort
7.  Reverse
8.  Play
9.  Shuffle
10. Display Music Library
11. Save
12. Open
13. Exit
2
================================================================================

YOUR PLAYLIST

3 songs                                                    11 min 48 sec
================================================================================
SONG NAME                    ARTIST                          DURATION
--------------------------------------------------------------------------------
Clap_your_hands              Moe_Bandy                     |    01:58
Joy_to_the_world             Pentatonix                    |    03:59
Our_Father                   Don_Moen                      |    05:51
```

Fig 5.5 – Output 5

```
**************MAIN MENU**************

1.  Create
2.  Display
3.  Insert
4.  Delete
5.  Search
6.  Sort
7.  Reverse
8.  Play
9.  Shuffle
10. Display Music Library
11. Save
12. Open
13. Exit
5
Enter song name
Our_Father

Song found at position 3
```

Fig 5.6 – Output 6

```
Enter songname
Salve_Regina

Enter artist name: Vox_Silentii

Enter the duration of the song (MM:SS): 2:4

**************MAIN MENU**************

1.  Create
2.  Display
3.  Insert
4.  Delete
5.  Search
6.  Sort
7.  Reverse
8.  Play
9.  Shuffle
10. Display Music Library
11. Save
12. Open
13. Exit
2
================================================================================

YOUR PLAYLIST

4 songs                                                    13 min 52 sec
================================================================================
SONG NAME                    ARTIST                          DURATION
--------------------------------------------------------------------------------
Clap_your_hands              Moe_Bandy                     |    01:58
Joy_to_the_world             Pentatonix                    |    03:59
Our_Father                   Don_Moen                      |    05:51
Salve_Regina                 Vox_Silentii                  |    02:04
```

Fig 5.7 – Output 7

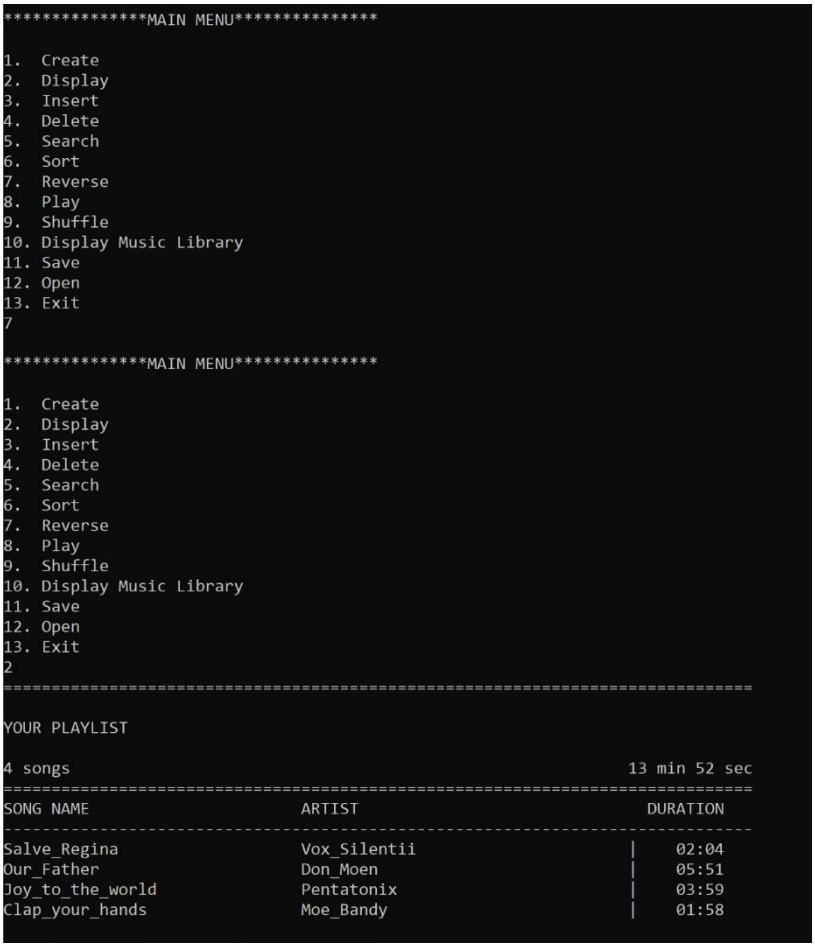

Fig 5.8– Output 8

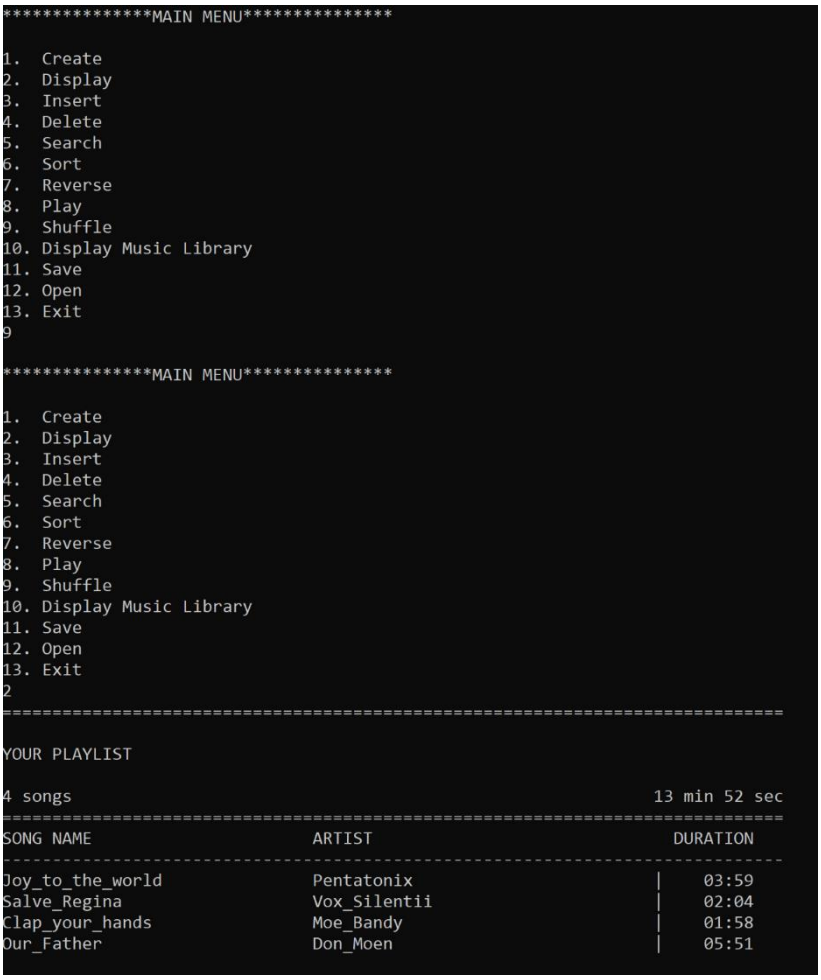

Fig 5.9 – Output 9

```
**************MAIN MENU**************

1.  Create
2.  Display
3.  Insert
4.  Delete
5.  Search
6.  Sort
7.  Reverse
8.  Play
9.  Shuffle
10. Display Music Library
11. Save
12. Open
13. Exit
10

Choose Genre:
1. Classical
2. Rock
3. Country
4. Jazz
5. Devotional
3

    SONGNAME                ARTIST              DURATION
1.  Mountain_dew            Stanley_bros          2:45
2.  Where_were_you          Alan_Jackson          3:40
3.  Country_roads           John_Denver           3:09
4.  Ring_of_fire            Johnny_Cash           2:38
5.  Take_my_hand            Jim_Reeves            2:44
6.  Sixteen_Tons            Ernie_Ford            4:30
7.  Crazy_Arms              Ray_Price             2:10
8.  Waterloo                Stonewall_Jack        3:29
9.  Elvira                  Oak_ridge_bros        4:30
10. Forever_and_ever        Randy_Travis          3:34



Do you want to append a song? (1 for yes, 0 for no): 1
Select line number of song: 3

1.Insert at beg
2.Insert at end
3.Insert at a particular position2
```

Fig 5.10 – Output 10

```
**************MAIN MENU**************

1.  Create
2.  Display
3.  Insert
4.  Delete
5.  Search
6.  Sort
7.  Reverse
8.  Play
9.  Shuffle
10. Display Music Library
11. Save
12. Open
13. Exit
2
================================================================================

YOUR PLAYLIST

5 songs                                                        17 min 1 sec
================================================================================
SONG NAME                   ARTIST                            DURATION
--------------------------------------------------------------------------------
Joy_to_the_world            Pentatonix                    |     03:59
Salve_Regina                Vox_Silentii                  |     02:04
Clap_your_hands             Moe_Bandy                     |     01:58
Our_Father                  Don_Moen                      |     05:51
Country_roads               John_Denver                   |     03:09
```

Fig 5.11 – Output 11

Fig 5.12 – Output 12

Fig 5.13 – Output 13

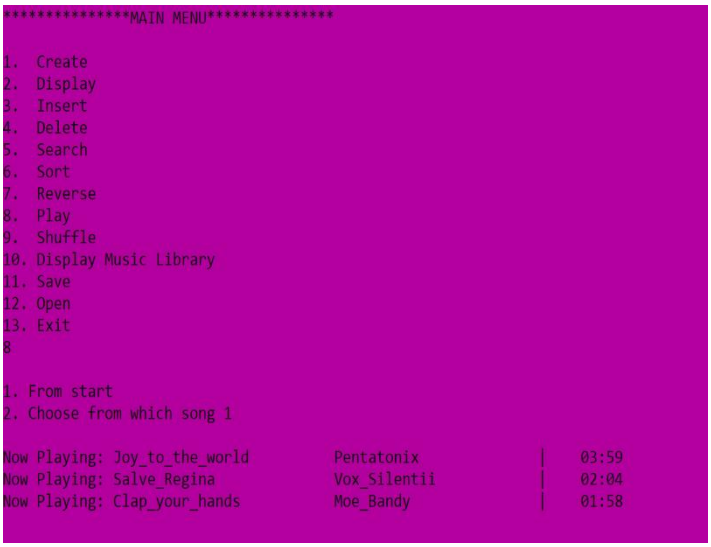

Fig 5.14 – Output 14

Fig 5.15 – Output 15

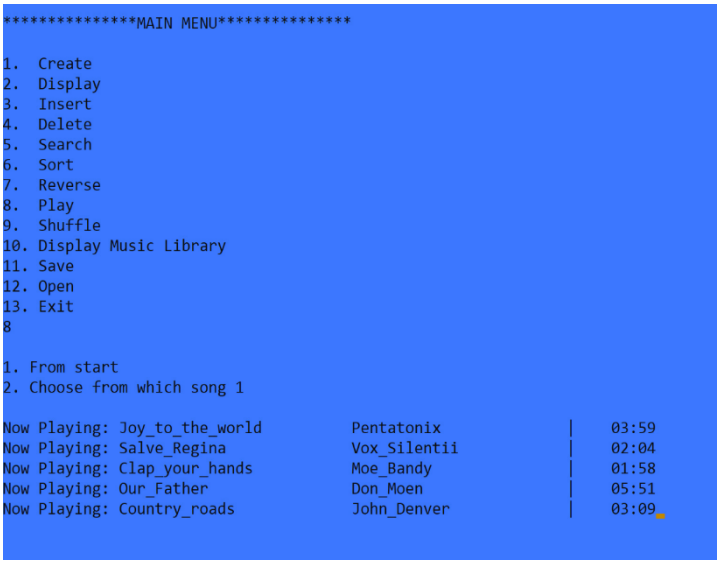

Fig 5.16 – Output 16

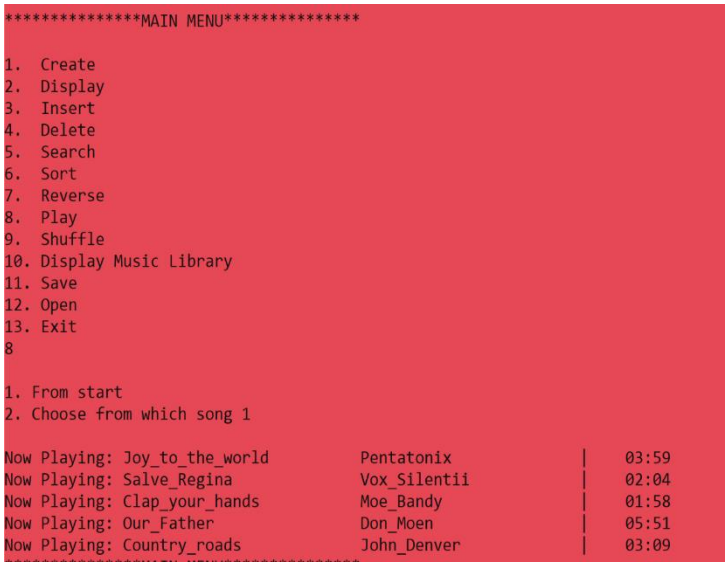

Fig 5.17 – Output 17

```
**************MAIN MENU**************

1.  Create
2.  Display
3.  Insert
4.  Delete
5.  Search
6.  Sort
7.  Reverse
8.  Play
9.  Shuffle
10. Display Music Library
11. Save
12. Open
13. Exit
4

1.Delete from beg
2.Delete from end
3.Delete from specific position
2
```

Fig 5.18 – Output 18

```
**************MAIN MENU**************

1.  Create
2.  Display
3.  Insert
4.  Delete
5.  Search
6.  Sort
7.  Reverse
8.  Play
9.  Shuffle
10. Display Music Library
11. Save
12. Open
13. Exit
2
================================================================================

YOUR PLAYLIST

4 songs                                                    13 min 52 sec
================================================================================
SONG NAME                 ARTIST                          DURATION
--------------------------------------------------------------------------------
Joy_to_the_world          Pentatonix                    |   03:59
Salve_Regina              Vox_Silentii                  |   02:04
Clap_your_hands           Moe_Bandy                     |   01:58
Our_Father                Don_Moen                      |   05:51
```

Fig 5.19 – Output 19

```
**************MAIN MENU**************

1.  Create
2.  Display
3.  Insert
4.  Delete
5.  Search
6.  Sort
7.  Reverse
8.  Play
9.  Shuffle
10. Display Music Library
11. Save
12. Open
13. Exit
11
Enter file name (*.txt): mysongs.txt
Play List stored in the file successfully

**************MAIN MENU**************

1.  Create
2.  Display
3.  Insert
4.  Delete
5.  Search
6.  Sort
7.  Reverse
8.  Play
9.  Shuffle
10. Display Music Library
11. Save
12. Open
13. Exit
13

--------------------------------
Process exited after 1226 seconds with return value 0
Press any key to continue . . .
```

Fig 5.20 – Output 20

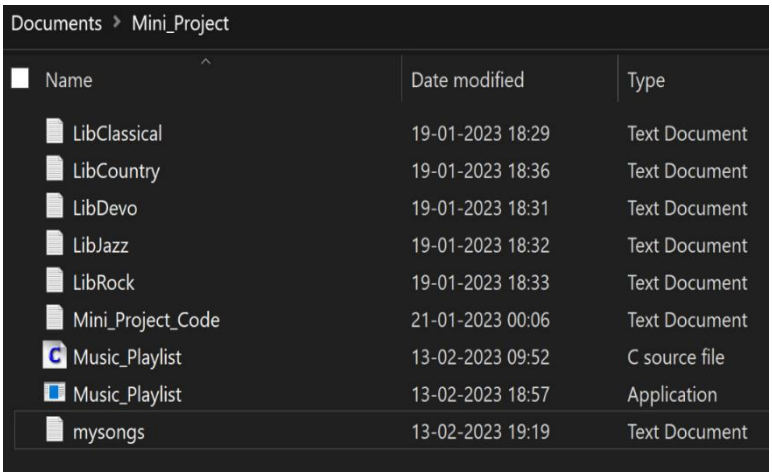

Fig 5.21 – Output 21

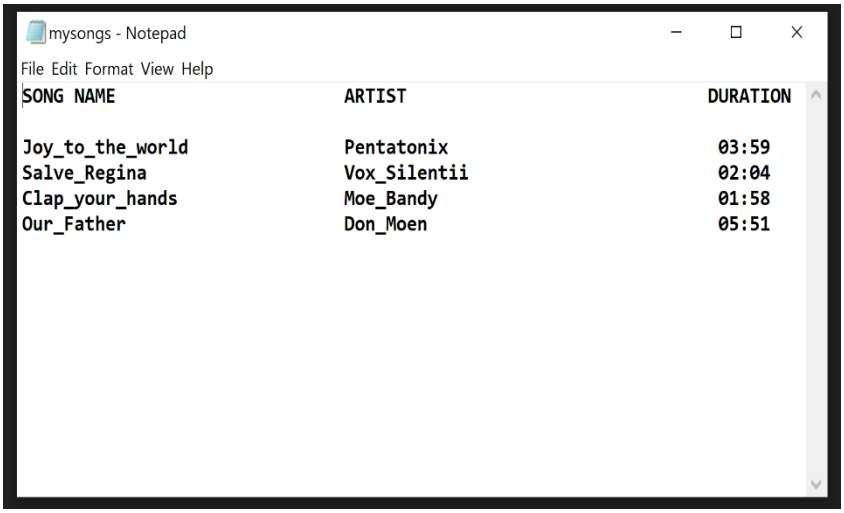

Fig 5.22 – Output 22

```
**************MAIN MENU**************

1.  Create
2.  Display
3.  Insert
4.  Delete
5.  Search
6.  Sort
7.  Reverse
8.  Play
9.  Shuffle
10. Display Music Library
11. Save
12. Open
13. Exit
12

Enter filename: mysongs.txt

**************MAIN MENU**************

1.  Create
2.  Display
3.  Insert
4.  Delete
5.  Search
6.  Sort
7.  Reverse
8.  Play
9.  Shuffle
10. Display Music Library
11. Save
12. Open
13. Exit
2
================================================================================

YOUR PLAYLIST

4 songs                                                      13 min 52 sec
================================================================================
SONG NAME                    ARTIST                            DURATION
--------------------------------------------------------------------------------
Joy_to_the_world             Pentatonix                    |     03:59
Salve_Regina                 Vox_Silentii                  |     02:04
Clap_your_hands              Moe_Bandy                     |     01:58
Our_Father                   Don_Moen                      |     05:51
```

Fig 5.23 – Output 23

CHAPTER 6

CONCLUSION

Playlists offer a convenient way to organize and listen to music, allowing users to have all their favourite songs in one place, ready to play at any time. The music playlist has successfully been implemented using the C programming language. Various algorithms, functions and data structures were used to store, manage and play songs in a user-friendly manner. They can also arrange the order of the songs to match their mood or preferences. Playlists can act as a musical diary, helping to recall memories and emotions associated with specific songs. The Music library serves as a source to refer to for new song names. The playlist can be saved offline and revived when the application is run again. Some of the future enhancements that can be integrated are – when the project is extended to play real MP3 songs, a remix feature can be added to make a new version of a song by combining different songs taking their timestamps and making new melodies. Currently, only one playlist can be created, it can be extended to manage multiple playlists for different categories.

REFERENCES

1. https://www.geeksforgeeks.org/
2. https://www.programiz.com/dsa/linked-list
3. https://www.youtube.com/@nesoacademy
4. https://www.programiz.com/c-programming/c-file-input-output
5. https://stackoverflow.com/
6. https://byjus.com/gate/file-handling-in-c/
7. https://www.youtube.com/@JacobSorber/videos
8. https://www.ibm.com/docs/en/zos/2.4.0?topic=functions-pthread-create-create-thread
9. Data Structures through C in depth by authors S.K. Srivastava and Deepali Srivastava