Problem statement:

Write a program that allows users to create a music playlist consisting of n songs and then display the list of the songs. The program should prompt users to enter the name of each song and add it to the playlist in a sequential order. The playlist should be designed to repeat once it reaches the end of the list.

Discussion on the program:

The program is to create a music playlist where the user can input song names, store them in a list, and then display the playlist. The playlist should loop when it reaches the end. We will collect the songs using user input, store them in a list, and simulate the looping by using indexing. The program will display the list and allow the user to play songs repeatedly in a cycle.

To optimize functionality, we will use a linked list to store the songs instead of an array. where songs are stored as nodes, and the last song links back to the first, ensuring continuous repetition. Each song is stored in a linked list node with a pointer to the next node. Once all songs are added, the last node points back to the first, making traversal continuous. The program should then display the playlist by traversing from the first song until it completes a full cycle. This approach allows seamless looping of the playlist, mimicking real-world music player functionality. A linked list provides several advantages for this type of program, as highlighted in the table below:

Feature	Linked List	Array
Size	Dynamic size (can grow/shrink as needed)	Fixed size (must define capacity in advance)
Memory Allocation	Efficient, memory is allocated as needed	Can waste memory if size is too large or too small
Insertion/Deletion	Efficient (no need to shift elements)	Inefficient (shifting elements is required)
Accessing Elements	Sequential access (must traverse list)	Direct access by index (fast)

Circular Playlist Implementation	Easy to implement (just link last node to the first)	Difficult to implement (requires manual index tracking)
Flexibility	Highly flexible, grows and shrinks with input	Less flexible, requires resizing for more elements

Due to these benefits, the linked list is a more suitable choice for implementing this music playlist program.

Explanation of the code:

This function, insert_songs_in_playlist, adds a new song to the end of a playlist represented as a linked list. If the playlist is empty, the new song becomes the first song. If not, it finds the last song and appends the new song after it. It works by creating a new node for the song and linking it to the existing list.

```
void insert_songs_in_playlist(struct Node **head, char *song){
  struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
  strcpy(newNode->song, song);
  newNode->next = NULL;

  if (*head == NULL){
    *head = newNode;
  }
  else{
    struct Node* temp = *head;
    while (temp->next != NULL){
      temp = temp->next;
    }
    temp->next = newNode;
  }
}
```

This function, print_the_playlist prints all the songs in the playlist, numbered from 1. It goes through each song in the linked list and displays the song's number and name until it reaches the end of the list.

```
void print_the_playlist(struct Node *head){
  printf("\nYour playlist:\n");
  int index = 1;
  while (head != NULL){
    printf("%d. %s\n", index++, head->song);
    head = head->next;
  }
}
```

This function, play_the_playlist plays all the songs in the playlist. After playing the entire playlist, it asks the user if they want to repeat it. If the user enters 1, the playlist starts over; if they enter 0, the function exits and displays a message. The playlist continues to loop until the user chooses to stop.

```
void play_the_playlist(struct Node *head){
  int choice;
  do{
    struct Node* temp = head;
    while (temp != NULL){
      printf("Now playing: %s\n", temp->song);
      temp = temp-> next;
    }
    printf("\nDo you want to repeat the playlist? (1/0): ");
    scanf(" %d", &choice);
  }while (choice == 1);
  printf("\nExit the playlist.\n");
}
```

This part of the program prompts the user to input the number of songs, then allows them to enter each song's name and add the songs to the playlist. It checks for invalid input and stops the program if an error is found.

```
int main(){
   struct Node *playlist = NULL;
   int n;

   printf("Enter the total number of songs: ");
   scanf("%d", &n);

   if (n <= 0){
      printf("ERROR!!! Enter again\n");
      return 1;
   }
   char song[MaxSongs];
   for (int i = 0; i < n; i++){
      printf("Enter the name of song %d: ", i + 1);
      scanf(" %[^\n]", song);
     insert_songs_in_playlist(&playlist, song);
   }
```

This part of the program first prints the playlist by calling the print_the_playlist function. It then plays the playlist using the play_the_playlist function, allowing the user to repeat it if desired. After the playlist finishes playing, the program ends with a return 0; statement, indicating successful completion.

```
   print_the_playlist(playlist);

   printf("\nPlaying the playlist:\n");
   play_the_playlist(playlist);

   return 0;
}
```

Key Points:

- Linked List: Used to store songs dynamically, allowing flexible playlist size.
- Input Playlist Length: The user is asked for the number of songs.
- Storing Songs: Each song is stored in a dynamically allocated node in a linked list.
- Memory Allocation: Memory for each song is dynamically allocated with malloc().
- String Handling: scanf(" %[^\n]", song) is used to allow spaces in song names.
- Display Playlist: The playlist is displayed with song names and their indices.
- Looping Playback: The playlist is played repeatedly, and the user can choose whether to continue.

Conclusion:

This program efficiently manages a looping music playlist using a circular linked list. Unlike fixed-size arrays, linked lists offer greater flexibility by allowing easy addition and removal of songs. Each song is stored in a node, with the last node linking back to the first, ensuring seamless playback.

With dynamic resizing and continuous looping, the program provides a user-friendly way to add, display, and play songs effortlessly. Overall, it offers a simple yet effective solution for managing and enjoying music.

Screenshots of the program:

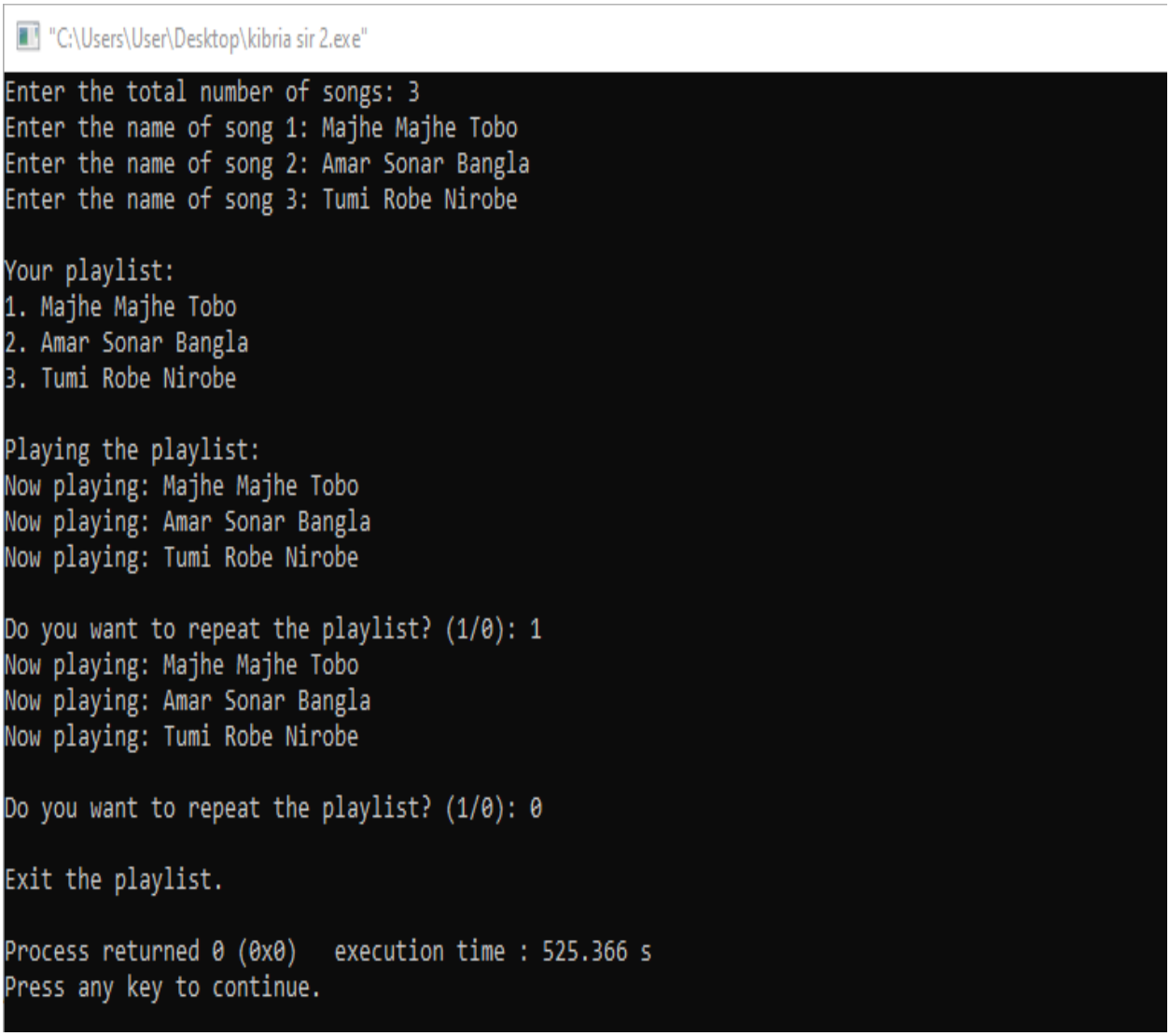
```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define MaxSongs 100

struct Node{
    char song[MaxSongs];
    struct Node* next;
};
```

```c
void insert_songs_in_playlist(struct Node **head, char *song){
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    strcpy(newNode->song, song);
    newNode->next = NULL;

    if (*head == NULL){
        *head = newNode;
    }
    else{
        struct Node* temp = *head;
        while (temp->next != NULL){
            temp = temp->next;
        }
        temp->next = newNode;
    }
}
```

```c
void print_the_playlist(struct Node *head){
    printf("\nYour playlist:\n");
    int index = 1;
    while (head != NULL){
        printf("%d. %s\n", index++, head->song);
        head = head->next;
    }
}
```

```c
void play_the_playlist(struct Node *head){
    int choice;
    do{
        struct Node* temp = head;
        while (temp != NULL){
            printf("Now playing: %s\n", temp->song);
            temp = temp-> next;
        }
        printf("\nDo you want to repeat the playlist? (1/0): ");
        scanf(" %d", &choice);
    }while (choice == 1);

    printf("\nExit the playlist.\n");
}

int main(){
    struct Node *playlist = NULL;
    int n;

    printf("Enter the total number of songs: ");
    scanf("%d", &n);

    if (n <= 0){
        printf("ERROR!!! Enter again\n");
        return 1;
    }

    char song[MaxSongs];
    for (int i = 0; i < n; i++){
        printf("Enter the name of song %d: ", i + 1);
        scanf(" %[^\n]", song);
       insert_songs_in_playlist(&playlist, song);
    }

    print_the_playlist(playlist);

    printf("\nPlaying the playlist:\n");
    play_the_playlist(playlist);

    return 0;
}
```

Screenshots of output:



Explanation of the output:

Input the total number and names of the songs, display the playlist, and highlight which songs are currently playing

```
"C:\Users\User\Desktop\kibria sir 2.exe"
Enter the total number of songs: 3
Enter the name of song 1: Majhe Majhe Tobo
Enter the name of song 2: Amar Sonar Bangla
Enter the name of song 3: Tumi Robe Nirobe

Your playlist:
1. Majhe Majhe Tobo
2. Amar Sonar Bangla
3. Tumi Robe Nirobe

Playing the playlist:
Now playing: Majhe Majhe Tobo
Now playing: Amar Sonar Bangla
Now playing: Tumi Robe Nirobe
```

Press 1 to repeat the playlist.

```
Do you want to repeat the playlist? (1/0): 1
Now playing: Majhe Majhe Tobo
Now playing: Amar Sonar Bangla
Now playing: Tumi Robe Nirobe
```

Press 0 to exit the playlist.

```
Do you want to repeat the playlist? (1/0): 0

Exit the playlist.

Process returned 0 (0x0)   execution time : 525.366 s
Press any key to continue.
```