

IMPLEMENTATION OF MUSIC PLAYER

Name - Adya Singh

RA2211003010181

PROBLEM STATEMENT

In today's digital age, music is an integral part of people's lives. With the increasing popularity of digital audio formats and the ever-expanding music libraries available online, there is a growing need for efficient and feature-rich music player applications.

My project will focus on developing a music player application that offers a range of features and functionalities to enhance the music listening experience. The application will be capable of playing audio files, managing playlists, and providing a user-friendly interface for users to interact with.

DATA STRUCTURE - ARRAY

Using an array as a data structure to create a music player's playlist has both advantages and limitations.

Time Complexity - Basic operations like adding a song, deleting a song, or playing the next or previous song are easy to implement using arrays. These operations typically have time complexity of $O(1)$ for constant-time access.

Arrays can be used for creating a basic music player's playlist(with a small and fixed playlist size), they may not be the best choice for more feature-rich applications due to their fixed size and limitations in efficient insertions and deletions.

PROGRAM

main.c

```
1 #include <stdio.h>
2 #include <string.h>
3
4 #define MAX_SONGS 5
5
6 struct Song {
7     char title[50];
8     char artist[50];
9 };
10
11 void createPlaylist(struct Song playlist[], int *playlistSize) {
12     if (*playlistSize >= MAX_SONGS) {
13         printf("Playlist is full. Cannot add more songs.\n");
14         return;
15     }
16
17     struct Song newSong;
18
19     printf("Enter song title: ");
20     scanf(" %[\n]s", newSong.title);
21     printf("Enter artist: ");
22     scanf(" %[\n]s", newSong.artist);
23
24     playlist[(*playlistSize)++] = newSong;
25
26     printf("Song added to the playlist.\n");
27 }
28
29 void displayPlaylist(struct Song playlist[], int playlistSize) {
30     if (playlistSize == 0) {
31         printf("The playlist is empty.\n");
32     } else {
33         printf("Current Playlist:\n");
34         for (int i = 0; i < playlistSize; i++) {
35             printf("%d %s by %s\n", i + 1, playlist[i].title, playlist[i].artist);
```

main.c

```
35         printf("%d. %s by %s\n", i + 1, playlist[i].title, playlist[i].artist);
36     }
37 }
38 }
39
40 void playSong(struct Song playlist[], int playlistSize) {
41     if (playlistSize == 0) {
42         printf("The playlist is empty. Add songs before playing.\n");
43         return;
44     }
45
46     displayPlaylist(playlist, playlistSize);
47
48     int songIndex;
49     printf("Enter the song number to play: ");
50     scanf("%d", &songIndex);
51
52     if (songIndex >= 1 && songIndex <= playlistSize) {
53         printf("Now playing: %s by %s\n", playlist[songIndex - 1].title, playlist[songIndex - 1]
54             .artist);
55     } else {
56         printf("Invalid song number.\n");
57     }
58 }
59
60 void deleteSong(struct Song playlist[], int *playlistSize) {
61     if (*playlistSize == 0) {
62         printf("The playlist is empty. Nothing to delete.\n");
63         return;
64     }
65
66     displayPlaylist(playlist, *playlistSize);
67
68     int songIndex;
```

main.c

```
66
67 int songIndex;
68 printf("Enter the song number to delete: ");
69 scanf("%d", &songIndex);
70
71 if (songIndex >= 1 && songIndex <= *playlistSize) {
72     for (int i = songIndex - 1; i < *playlistSize - 1; i++) {
73         playlist[i] = playlist[i + 1];
74     }
75     (*playlistSize)--;
76     printf("Song deleted from the playlist.\n");
77 } else {
78     printf("Invalid song number.\n");
79 }
80 }
81 }
82
83 int main() {
84     struct Song playlist[MAX_SONGS];
85     int playlistSize = 0;
86     int choice;
87
88     while (1) {
89         printf("\nOptions:\n");
90         printf("1. Create a playlist\n");
91         printf("2. Display playlist\n");
92         printf("3. Play a song\n");
93         printf("4. Delete a song\n");
94         printf("5. Exit\n");
95         printf("Enter your choice: ");
96         scanf("%d", &choice);
97
98         switch (choice) {
99             case 1:
100                 createPlaylist(playlist, &playlistSize);
```

main.c

```
87
88 while (1) {
89     printf("\nOptions:\n");
90     printf("1. Create a playlist\n");
91     printf("2. Display playlist\n");
92     printf("3. Play a song\n");
93     printf("4. Delete a song\n");
94     printf("5. Exit\n");
95     printf("Enter your choice: ");
96     scanf("%d", &choice);
97
98     switch (choice) {
99         case 1:
100             createPlaylist(playlist, &playlistSize);
101             break;
102         case 2:
103             displayPlaylist(playlist, playlistSize);
104             break;
105         case 3:
106             playSong(playlist, playlistSize);
107             break;
108         case 4:
109             deleteSong(playlist, &playlistSize);
110             break;
111         case 5:
112             printf("Goodbye!\n");
113             return 0;
114         default:
115             printf("Invalid choice. Please select a valid option.\n");
116     }
117 }
118
119 return 0;
120 }
121
```

OUTPUT

```
Output Clear
/tmp/0R1c9KTkDz.o
Options:
1. Create a playlist
2. Display playlist
3. Play a song
4. Delete a song
5. Exit
Enter your choice: 1
Enter song title: Ceilings
Enter artist: Lizzy
Song added to the playlist.

Options:
1. Create a playlist
2. Display playlist
3. Play a song
4. Delete a song
5. Exit
Enter your choice: 1
Enter song title: August
Enter artist: Taylor Swift
Song added to the playlist.

Options:
1. Create a playlist
2. Display playlist
3. Play a song
4. Delete a song
5. Exit
Enter your choice: 2
Current Playlist:
1. Ceilings by Lizzy
2. August by Taylor Swift
```

```
Output Clear
Options:
1. Create a playlist
2. Display playlist
3. Play a song
4. Delete a song
5. Exit
Enter your choice: 3
Current Playlist:
1. Ceilings by Lizzy
2. August by Taylor Swift
Enter the song number to play: 2
Now playing: August by Taylor Swift

Options:
1. Create a playlist
2. Display playlist
3. Play a song
4. Delete a song
5. Exit
Enter your choice: 4
Current Playlist:
1. Ceilings by Lizzy
2. August by Taylor Swift
Enter the song number to delete: 2
Song deleted from the playlist.

Options:
1. Create a playlist
2. Display playlist
3. Play a song
4. Delete a song
5. Exit
Enter your choice: 5
Goodbye!
```

PROS AND CONS OF USING ARRAY

PROS	CONS
1. Simplicity: They provide a linear and contiguous block of memory to store a fixed number of elements. This can make implementation relatively simple and efficient.	1. Fixed Size: If the user wants to add more songs than the array's capacity, you'll need to handle dynamic resizing, which can be complex.
2. Constant-Time Access: Accessing songs in an array is very fast and efficient. You can directly access a song by its index.	2. Inefficient Insertions and Deletions: It requires shifting elements, resulting in time complexity of $O(n)$, where n is the number of elements.
3. Predictable Memory Usage: You know in advance how much memory the playlist will consume.	3. No Built-In Sorting: Arrays do not offer built-in sorting capabilities.
4. Straightforward Playlist Operations: operations typically have time complexity of $O(1)$ for constant-time access.	4. Wasted Memory: If the array size is chosen too large, it may lead to wasted memory if the playlist isn't filled to capacity.

DATA STRUCTURE - LINKED LIST

Using a linked list as a data structure to create a music player's playlist has several advantages and can be well-suited for managing dynamic playlists.

Time Complexity - $O(n)$

Explanation: To display the entire playlist or play songs sequentially, you need to traverse the entire linked list.

Using a linked list to create a music player's playlist provides flexibility and efficiency, particularly in scenarios where the playlist size changes frequently, and when you need features like dynamic sorting and reordering.

PROGRAM

main.c

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4
5 struct Song {
6     char title[50];
7     char artist[50];
8     struct Song *next;
9 };
10
11 struct Song *head = NULL;
12
13 void createPlaylist() {
14     struct Song *newSong = (struct Song *)malloc(sizeof(struct Song));
15     if (newSong == NULL) {
16         printf("Memory allocation failed. Playlist is full.\n");
17         return;
18     }
19
20     printf("Enter song title: ");
21     scanf("%[^\n]s", newSong->title);
22     printf("Enter artist: ");
23     scanf("%[^\n]s", newSong->artist);
24     newSong->next = NULL;
25
26     if (head == NULL) {
27         head = newSong;
28     } else {
29         struct Song *current = head;
30         while (current->next != NULL) {
31             current = current->next;
32         }
33         current->next = newSong;
34     }
35 }
```

main.c

```
34 }
35
36 printf("Song added to the playlist.\n");
37 }
38
39 void displayPlaylist() {
40     if (head == NULL) {
41         printf("The playlist is empty.\n");
42         return;
43     }
44
45     struct Song *current = head;
46     int index = 1;
47
48     printf("Playlist:\n");
49     while (current != NULL) {
50         printf("%d. %s by %s\n", index, current->title, current->artist);
51         current = current->next;
52         index++;
53     }
54 }
55
56 void playSong() {
57     if (head == NULL) {
58         printf("The playlist is empty. Add songs before playing.\n");
59         return;
60     }
61
62     displayPlaylist();
63
64     int songIndex;
65     printf("Enter the song number to play: ");
66     scanf("%d", &songIndex);
67 }
```

```

main.c
67
68 struct Song *current = head;
69 int currentIndex = 1;
70
71 while (current != NULL) {
72     if (currentIndex == songIndex) {
73         printf("Now playing: %s by %s\n", current->title, current->artist);
74         return;
75     }
76     current = current->next;
77     currentIndex++;
78 }
79
80 printf("Invalid song number.\n");
81 }
82
83 void deleteSong() {
84     if (head == NULL) {
85         printf("The playlist is empty. Nothing to delete.\n");
86         return;
87     }
88
89     displayPlaylist();
90
91     int songIndex;
92     printf("Enter the song number to delete: ");
93     scanf("%d", &songIndex);
94
95     if (songIndex < 1) {
96         printf("Invalid song number.\n");
97         return;
98     }
99
100     if (songIndex == 1) {
101         struct Song *temp = head;

```

```

main.c
101     struct Song *temp = head;
102     head = head->next;
103     free(temp);
104     printf("Song deleted from the playlist.\n");
105     return;
106 }
107
108 struct Song *current = head;
109 int currentIndex = 1;
110
111 while (current != NULL) {
112     if (currentIndex == songIndex - 1) {
113         if (current->next != NULL) {
114             struct Song *temp = current->next;
115             current->next = temp->next;
116             free(temp);
117             printf("Song deleted from the playlist.\n");
118             return;
119         }
120     }
121
122     current = current->next;
123     currentIndex++;
124 }
125
126 printf("Invalid song number.\n");
127 }
128
129 void insertSong() {
130     struct Song *newSong = (struct Song *)malloc(sizeof(struct Song));
131     if (newSong == NULL) {
132         printf("Memory allocation failed. Playlist is full.\n");
133         return;
134     }
135

```

```

main.c
134 }
135
136 printf("Enter song title: ");
137 scanf("%[^\n]s", newSong->title);
138 printf("Enter artist: ");
139 scanf("%[^\n]s", newSong->artist);
140 newSong->next = head;
141 head = newSong;
142 printf("Song inserted into the playlist.\n");
143 }
144
145 void freePlaylist() {
146     struct Song *current = head;
147     while (current != NULL) {
148         struct Song *temp = current;
149         current = current->next;
150         free(temp);
151     }
152 }
153
154 int main() {
155     int choice;
156
157     while (1) {
158         printf("\nOptions:\n");
159         printf("1. Create a playlist\n");
160         printf("2. Display playlist\n");
161         printf("3. Play a song\n");
162         printf("4. Delete a song\n");
163         printf("5. Insert a song\n");
164         printf("6. Exit\n");
165         printf("Enter your choice: ");
166         scanf("%d", &choice);
167

```

```

main.c
161         printf("3. Play a song\n");
162         printf("4. Delete a song\n");
163         printf("5. Insert a song\n");
164         printf("6. Exit\n");
165         printf("Enter your choice: ");
166         scanf("%d", &choice);
167
168         switch (choice) {
169             case 1:
170                 createPlaylist();
171                 break;
172             case 2:
173                 displayPlaylist();
174                 break;
175             case 3:
176                 playSong();
177                 break;
178             case 4:
179                 deleteSong();
180                 break;
181             case 5:
182                 insertSong();
183                 break;
184             case 6:
185                 freePlaylist();
186                 printf("Goodbye!\n");
187                 return 0;
188             default:
189                 printf("Invalid choice. Please select a valid option.\n");
190         }
191     }
192
193     return 0;
194 }
195

```

OUTPUT

```
Output Clear
Options:
1. Create a playlist
2. Display playlist
3. Play a song
4. Delete a song
5. Insert a song
6. Exit
Enter your choice: 1
Enter song title: Treat You Better
Enter artist: Shawn Mendes
Song added to the playlist.

Options:
1. Create a playlist
2. Display playlist
3. Play a song
4. Delete a song
5. Insert a song
6. Exit
Enter your choice: 1
Enter song title: Entertainer
Enter artist: Zayn
Song added to the playlist.

Options:
1. Create a playlist
2. Display playlist
3. Play a song
4. Delete a song
5. Insert a song
6. Exit
Enter your choice: 2
Playlist:
1. Treat You Better by Shawn Mendes
```

```
Output Clear
4. Delete a song
5. Insert a song
6. Exit
Enter your choice: 2
Playlist:
1. Treat You Better by Shawn Mendes
2. Entertainer by Zayn

Options:
1. Create a playlist
2. Display playlist
3. Play a song
4. Delete a song
5. Insert a song
6. Exit
Enter your choice: 5
Enter song title: August
Enter artist: Taylor Swift
Song inserted into the playlist.

Options:
1. Create a playlist
2. Display playlist
3. Play a song
4. Delete a song
5. Insert a song
6. Exit
Enter your choice: 4
Playlist:
1. August by Taylor Swift
2. Treat You Better by Shawn Mendes
3. Entertainer by Zayn
Enter the song number to delete: 2
Song deleted from the playlist.
```

PROS AND CONS OF USING LINKED LIST

PROS	CONS
1. Dynamic Size: Linked lists are dynamic data structures, meaning that they can grow or shrink as needed.	1. Inefficient Random Access: To access the n th song, you need to traverse the list from the beginning to reach the desired position.
2. Efficient Insertions and Deletions: Inserting a new song or deleting an existing one can often be done in constant time ($O(1)$) if you have a reference to the node.	2. Memory Overhead: Each node in a linked list requires extra memory to store references (pointers) to the next and, in the case of doubly linked lists, the previous node.
3. Memory Efficiency: Linked lists allocate memory only as needed, which can save memory compared to fixed-size arrays.	3. Inefficient Searching: Searching for a song in a linked list, requires traversing the list sequentially to find the desired song.
4. Cyclic Playlists: Linked lists can be used to create cyclic playlists, where the last song is followed by the first song, creating a continuous loop.	4. Time Complexity - $O(n)$ To display the entire playlist or play songs sequentially, you need to traverse the entire linked list.

COMPARISON

LINKED LIST	ARRAY
1. Dynamic Size: Linked lists can grow or shrink dynamically, accommodating changes in the playlist size without fixed constraints.	1. Fixed Size: Arrays have a fixed size, which may limit the number of songs in the playlist and require dynamic resizing to accommodate changes.
2. Efficient Insertions and Deletions: Inserting and deleting songs in a linked list is efficient, especially when using a doubly linked list.	2. Inefficient Insertions and Deletions: Inserting or deleting songs in the middle is inefficient ($O(n)$).
3. Memory Efficiency: Linked lists allocate memory only as needed, potentially saving memory when the playlist is not full.	3. Wasted Memory: If the array size is chosen too large, it may lead to wasted memory if the playlist isn't filled to capacity.
4. Versatile Sorting and Reordering: Easily reorder songs within the playlist.	4. Limited Sorting and Reordering: Arrays offer limited support for complex sorting and reordering.

CONCLUSION

Using a linked list as the underlying data structure for creating a music player's playlist can be a superior choice for several compelling reasons. A linked list offers dynamic playlist management capabilities, efficient insertions and deletions, versatile sorting and reordering options, and memory efficiency that align with the ever-changing and dynamic nature of music playlists.

Using a linked list to create a music player's playlist provides flexibility and efficiency, particularly in scenarios where the playlist size changes frequently, and when you need features like dynamic sorting and reordering.