Lab Task 7 [Music Playlist Manager]

Task:

You are hired as a software developer for a music streaming company developing a Music Playlist Manager.
Each playlist stores music tracks that can repeat automatically after the last song ends.
The system must support two types of playlists:
Circular Singly Linked List (CSLL) – Simple Playlist
This type is used for basic playlist looping where music only moves forward.
Each track contains:
- Track ID
- Track Name
- Artist Name
- Duration

TASKS:
1. Design and implement a Circular Singly Linked List to manage the playlist.
2. Provide the following operations:

Insertion
- Insert a new track at the beginning
- Insert a new track at the end
- Insert a new track at a specific position

Deletion
- Delete the first track
- Delete the last track
- Delete a track by Track ID

Display
- Display all songs by traversing the list from head until the first song appears again.

Circular Doubly Linked List (CDLL) – Premium Playlist
This advanced playlist allows:
Forward play
Backward play
Looping from both directions
Each track contains:
- Track ID
- Track Title
- Artist
- Album
- Duration

TASKS:
1. Implement a Circular Doubly Linked List for premium users.

Insertion Operations
- Insert track at the beginning
- Insert track at the end
- Insert track at any given position

Deletion Operations
- Delete first track
- Delete last track
- Delete a track by position

Display
- Display playlist in forward direction
- Display playlist in reverse direction

Additional Requirement
Implement track navigation:
- Play next song
- Play previous song

Code:

```cpp
#include <iostream>
#include <string>
using namespace std;

struct CSNode {
    int id;
    string name;
    string artist;
    float duration;
    CSNode* next;
};

CSNode* csHead = NULL;

CSNode* createCSNode(int id, string name, string artist, float duration) {
    CSNode* n = new CSNode();
    n->id = id;
    n->name = name;
    n->artist = artist;
    n->duration = duration;
    n->next = NULL;
    return n;
}

void csInsertBeginning(int id, string name, string artist, float duration) {
    CSNode* n = createCSNode(id, name, artist, duration);
    if (csHead == NULL) {
        csHead = n;
        n->next = n;
    } else {
        CSNode* temp = csHead;
        while (temp->next != csHead) temp = temp->next;
        n->next = csHead;
        temp->next = n;
        csHead = n;
    }
    cout << "Inserted" << endl;
}

void csInsertEnd(int id, string name, string artist, float duration) {
    CSNode* n = createCSNode(id, name, artist, duration);
    if (csHead == NULL) {
        csHead = n;
        n->next = n;
        cout << "Inserted" << endl;
        return;
    }
    CSNode* temp = csHead;
    while (temp->next != csHead) temp = temp->next;
    temp->next = n;
    n->next = csHead;
    cout << "Inserted" << endl;
}

void csInsertPosition(int pos, int id, string name, string artist, float duration) {
    if (pos < 0) {
        cout << "Invalid" << endl;
        return;
    }
    if (csHead == NULL) {
        if (pos == 0) csInsertBeginning(id, name, artist, duration);
        else cout << "Invalid" << endl;
        return;
    }
    if (pos == 0) {
        csInsertBeginning(id, name, artist, duration);
        return;
    }
    CSNode* temp = csHead;
    int index = 0;
    while (index < pos - 1 && temp->next != csHead) {
        temp = temp->next;
        index++;
    }
    if (index != pos - 1) {
        cout << "Invalid" << endl;
        return;
    }
    CSNode* n = createCSNode(id, name, artist, duration);
    n->next = temp->next;
    temp->next = n;
    cout << "Inserted" << endl;
}

void csDeleteBeginning() {
    if (csHead == NULL) {
        cout << "Empty" << endl;
        return;
    }
    if (csHead->next == csHead) {
        delete csHead;
        csHead = NULL;
        cout << "Deleted" << endl;
        return;
    }
    CSNode* last = csHead;
    while (last->next != csHead) last = last->next;
    CSNode* del = csHead;
    csHead = csHead->next;
    last->next = csHead;
    delete del;
    cout << "Deleted" << endl;
}

void csDeleteEnd() {
```

```
void csDeleteEnd() {
    if (csHead == NULL) {
        cout << "Empty" << endl;
        return;
    }
    if (csHead->next == csHead) {
        delete csHead;
        csHead = NULL;
        cout << "Deleted" << endl;
        return;
    }
    CSNode* temp = csHead;
    while (temp->next->next != csHead) temp = temp->next;
    CSNode* del = temp->next;
    temp->next = csHead;
    delete del;
    cout << "Deleted" << endl;
}

void csDeleteByID(int id) {
    if (csHead == NULL) {
        cout << "Empty" << endl;
        return;
    }
    if (csHead->id == id) {
        csDeleteBeginning();
        return;
    }
    CSNode* temp = csHead;
    while (temp->next != csHead && temp->next->id != id) temp = temp->next;
    if (temp->next == csHead) {
        cout << "Not Found" << endl;
        return;
    }
    CSNode* del = temp->next;
    temp->next = del->next;
    delete del;
    cout << "Deleted" << endl;
}

void csDisplay() {
    if (csHead == NULL) {
        cout << "Empty" << endl;
        return;
    }
    CSNode* temp = csHead;
    do {
        cout << temp->id << " " << temp->name << " " << temp->artist << " " << temp->duration << endl;
        temp = temp->next;
    } while (temp != csHead);
}

struct CDNode {
```

```
struct CDNode {
    int id;
    string title;
    string artist;
    string album;
    float duration;
    CDNode* next;
    CDNode* prev;
};

CDNode* cdHead = NULL;
CDNode* currentTrack = NULL;

CDNode* createCDNode(int id, string title, string artist, string album, float duration) {
    CDNode* n = new CDNode();
    n->id = id;
    n->title = title;
    n->artist = artist;
    n->album = album;
    n->duration = duration;
    n->next = n->prev = NULL;
    return n;
}

void cdInsertBeginning(int id, string title, string artist, string album, float duration) {
    CDNode* n = createCDNode(id, title, artist, album, duration);
    if (cdHead == NULL) {
        cdHead = n;
        n->next = n->prev = n;
        currentTrack = n;
    } else {
        CDNode* last = cdHead->prev;
        n->next = cdHead;
        n->prev = last;
        last->next = n;
        cdHead->prev = n;
        cdHead = n;
    }
    cout << "Inserted" << endl;
}

void cdInsertEnd(int id, string title, string artist, string album, float duration) {
    CDNode* n = createCDNode(id, title, artist, album, duration);
    if (cdHead == NULL) {
        cdHead = n;
        n->next = n->prev = n;
        currentTrack = n;
        cout << "Inserted" << endl;
        return;
    }
    CDNode* last = cdHead->prev;
    last->next = n;
    n->prev = last;
    n->next = cdHead;
    cdHead->prev = n;
    cout << "Inserted" << endl;
}

void cdInsertPosition(int pos, int id, string title, string artist, string album, float duration) {
```

```
void cdInsertPosition(int pos, int id, string title, string artist, string album, float duration) {
    if (pos < 0) {
        cout << "Invalid" << endl;
        return;
    }
    if (cdHead == NULL) {
        if (pos == 0) cdInsertBeginning(id, title, artist, album, duration);
        else cout << "Invalid" << endl;
        return;
    }
    if (pos == 0) {
        cdInsertBeginning(id, title, artist, album, duration);
        return;
    }
    CDNode* temp = cdHead;
    int index = 0;
    while (index < pos - 1 && temp->next != cdHead) {
        temp = temp->next;
        index++;
    }
    if (index != pos - 1) {
        cout << "Invalid" << endl;
        return;
    }
    CDNode* n = createCDNode(id, title, artist, album, duration);
    n->next = temp->next;
    n->prev = temp;
    temp->next->prev = n;
    temp->next = n;
    cout << "Inserted" << endl;
}

void cdDeleteBeginning() {
    if (cdHead == NULL) {
        cout << "Empty" << endl;
        return;
    }
    if (cdHead->next == cdHead) {
        delete cdHead;
        cdHead = NULL;
        currentTrack = NULL;
        cout << "Deleted" << endl;
        return;
    }
    CDNode* last = cdHead->prev;
    CDNode* del = cdHead;
    cdHead = cdHead->next;
    cdHead->prev = last;
    last->next = cdHead;
    if (currentTrack == del) currentTrack = cdHead;
    delete del;
    cout << "Deleted" << endl;
}

void cdDeleteEnd() {
    if (cdHead == NULL) {
        cout << "Empty" << endl;
        return;
    }
    if (cdHead->next == cdHead) {
        delete cdHead;
        cdHead = NULL;
        currentTrack = NULL;
        cout << "Deleted" << endl;
        return;
    }
    CDNode* last = cdHead->prev;
    CDNode* secondLast = last->prev;
    secondLast->next = cdHead;
    cdHead->prev = secondLast;
    if (currentTrack == last) currentTrack = cdHead;
    delete last;
    cout << "Deleted" << endl;
}

void cdDeletePosition(int pos) {
```

```
void cdDeletePosition(int pos) {
    if (cdHead == NULL) {
        cout << "Empty" << endl;
        return;
    }
    if (pos < 0) {
        cout << "Invalid" << endl;
        return;
    }
    if (pos == 0) {
        cdDeleteBeginning();
        return;
    }
    CDNode* temp = cdHead;
    int index = 0;
    while (index < pos && temp->next != cdHead) {
        temp = temp->next;
        index++;
    }
    if (index != pos) {
        cout << "Invalid" << endl;
        return;
    }
    if (temp == cdHead) {
        cout << "Invalid" << endl;
        return;
    }
    if (currentTrack == temp) currentTrack = temp->next;
    temp->prev->next = temp->next;
    temp->next->prev = temp->prev;
    delete temp;
    cout << "Deleted" << endl;
}

void cdDisplayForward() {
    if (cdHead == NULL) {
        cout << "Empty" << endl;
        return;
    }
    CDNode* temp = cdHead;
    do {
        cout << temp->id << " " << temp->title << " " << temp->artist << " " << temp->album << " " <<
temp->duration << endl;
        temp = temp->next;
    } while (temp != cdHead);
}

void cdDisplayBackward() {
    if (cdHead == NULL) {
        cout << "Empty" << endl;
        return;
    }
    CDNode* temp = cdHead->prev;
    do {
        cout << temp->id << " " << temp->title << " " << temp->artist << " " << temp->album << " " <<
temp->duration << endl;
        temp = temp->prev;
    } while (temp != cdHead->prev);
}

void playNext() {

}

void playPrevious() {
    if (currentTrack == NULL) {
        cout << "Empty" << endl;
        return;
    }
    currentTrack = currentTrack->prev;
    cout << "Now Playing: " << currentTrack->title << endl;
}

int main() {
```

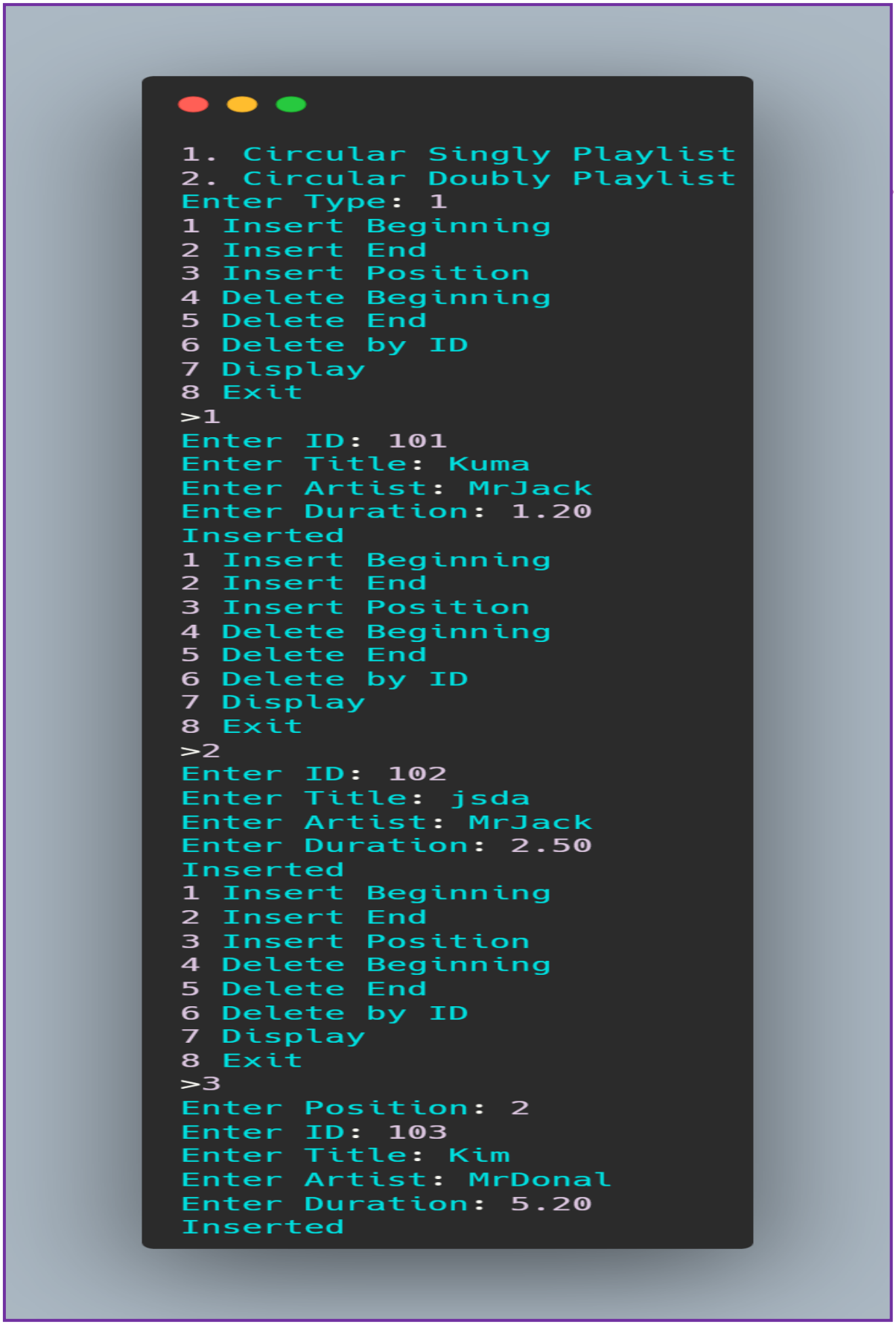
```cpp
int main() {
    int type;
    cout << "1. Circular Singly Playlist" << endl;
    cout << "2. Circular Doubly Playlist" << endl;
    cout << "Enter Type: ";
    cin >> type;

    int choice;

    if (type == 1) {
        do {
            cout << "1 Insert Beginning" << endl;
            cout << "2 Insert End" << endl;
            cout << "3 Insert Position" << endl;
            cout << "4 Delete Beginning" << endl;
            cout << "5 Delete End" << endl;
            cout << "6 Delete by ID" << endl;
            cout << "7 Display" << endl;
            cout << "8 Exit" << endl;
            cout<<">";
            cin >> choice;

            int id, pos;
            string name, artist;
            float duration;

            if (choice == 1) {
                cout << "Enter ID: ";
                cin >> id;
                cout << "Enter Title: ";
                cin >> name;
                cout << "Enter Artist: ";
                cin >> artist;
                cout << "Enter Duration: ";
                cin >> duration;
                csInsertBeginning(id, name, artist, duration);
            }
            else if (choice == 2) {
                cout << "Enter ID: ";
                cin >> id;
                cout << "Enter Title: ";
                cin >> name;
                cout << "Enter Artist: ";
                cin >> artist;
                cout << "Enter Duration: ";
                cin >> duration;
                csInsertEnd(id, name, artist, duration);
            }
            else if (choice == 3) {
                cout << "Enter Position: ";
                cin >> pos;
                cout << "Enter ID: ";
                cin >> id;
                cout << "Enter Title: ";
                cin >> name;
                cout << "Enter Artist: ";
                cin >> artist;
                cout << "Enter Duration: ";
                cin >> duration;
                csInsertPosition(pos, id, name, artist, duration);
            }
            else if (choice == 4) csDeleteBeginning();
            else if (choice == 5) csDeleteEnd();
            else if (choice == 6) {
                cout << "Enter Track ID to Delete: ";
                cin >> id;
                csDeleteByID(id);
            }
            else if (choice == 7) csDisplay();

        } while (choice != 8);
    }

    else if (type == 2) {
```

```
else if (type == 2) {
        do {
            cout << "1 Insert Beginning" << endl;
            cout << "2 Insert End" << endl;
            cout << "3 Insert Position" << endl;
            cout << "4 Delete Beginning" << endl;
            cout << "5 Delete End" << endl;
            cout << "6 Delete Position" << endl;
            cout << "7 Display Forward" << endl;
            cout << "8 Display Backward" << endl;
            cout << "9 Next Track" << endl;
            cout << "10 Previous Track" << endl;
            cout << "11 Exit" << endl;
            cout<<">";
            cin >> choice;

            int id, pos;
            string title, artist, album;
            float duration;

            if (choice == 1) {
                cout << "Enter ID: ";
                cin >> id;
                cout << "Enter Title: ";
                cin >> title;
                cout << "Enter Artist: ";
                cin >> artist;
                cout << "Enter Album: ";
                cin >> album;
                cout << "Enter Duration: ";
                cin >> duration;
                cdInsertBeginning(id, title, artist, album, duration);
            }
            else if (choice == 2) {
                cout << "Enter ID: ";
                cin >> id;
                cout << "Enter Title: ";
                cin >> title;
                cout << "Enter Artist: ";
                cin >> artist;
                cout << "Enter Album: ";
                cin >> album;
                cout << "Enter Duration: ";
                cin >> duration;
                cdInsertEnd(id, title, artist, album, duration);
            }
            else if (choice == 3) {
                cout << "Enter Position: ";
                cin >> pos;
                cout << "Enter ID: ";
                cin >> id;
                cout << "Enter Title: ";
                cin >> title;
                cout << "Enter Artist: ";
                cin >> artist;
                cout << "Enter Album: ";
                cin >> album;
                cout << "Enter Duration: ";
                cin >> duration;
                cdInsertPosition(pos, id, title, artist, album, duration);
            }
            else if (choice == 4) cdDeleteBeginning();
            else if (choice == 5) cdDeleteEnd();
            else if (choice == 6) {
                cout << "Enter Position: ";
                cin >> pos;
                cdDeletePosition(pos);
            }
            else if (choice == 7) cdDisplayForward();
            else if (choice == 8) cdDisplayBackward();
            else if (choice == 9) playNext();
            else if (choice == 10) playPrevious();

        } while (choice != 11);
    }

    return 0;
}
```

Table Of Output:

	Condition	Output
1	Insert (beginning / end / valid position)	`Inserted`
2	Insert position invalid (`pos < 0`, out of range)	`Invalid`
3	Delete beginning when list empty	`Empty`
4	Delete beginning (1 or more nodes)	`Deleted`
5	Delete end when list empty	`Empty`
6	Delete end (1 or more nodes)	`Deleted`
7	Delete by ID when list empty	`Empty`
8	Delete by ID when ID found	`Deleted`
9	Delete by ID when ID not found	`Not Found`
10	Display when list empty	`Empty`
11	Display when list not empty	Track details printed
	CS total output cases = 11	
	Condition	**Output**
12	Insert (beginning / end / valid position)	`Inserted`
13	Insert position invalid	`Invalid`
14	Delete beginning when list empty	`Empty`
15	Delete beginning (1 or more nodes)	`Deleted`
16	Delete end when list empty	`Empty`
17	Delete end (1 or more nodes)	`Deleted`
18	Delete position when list empty	`Empty`
19	Delete position invalid	`Invalid`
20	Delete position valid	`Deleted`
21	Display forward when empty	`Empty`
22	Display forward when not empty	Track details printed
23	Display backward when empty	`Empty`
24	Display backward when not empty	Track details printed
25	Play next when no track	`Empty`
26	Play previous when no track	`Empty`
27	Play next (track exists)	`Now Playing: <title>`
28	Play previous (track exists)	`Now Playing: <title>`
	CD total output cases = 17	
	Total 28 Possible Outputs	

When Insert (beginning / end / valid position

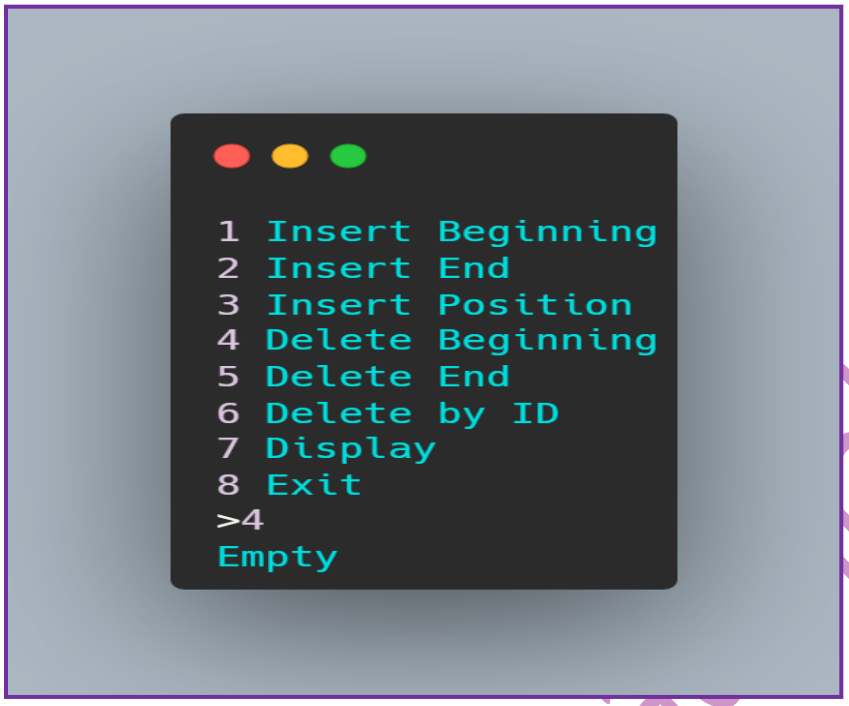


When

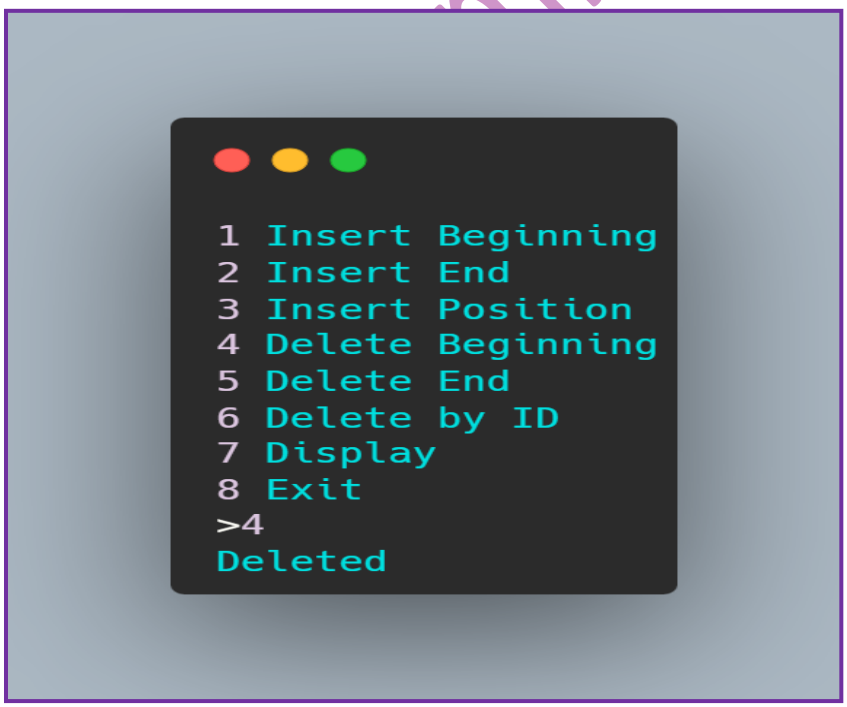
Insert position invalid (`pos < 0`, out of range)

```
1 Insert Beginning
2 Insert End
3 Insert Position
4 Delete Beginning
5 Delete End
6 Delete by ID
7 Display
8 Exit
>3
Enter Position: -1
Enter ID: 106
Enter Title: jackal
Enter Artist: Mj
Enter Duration: 5.20
Invalid
1 Insert Beginning
2 Insert End
3 Insert Position
4 Delete Beginning
5 Delete End
6 Delete by ID
7 Display
8 Exit
>3
Enter Position: 8
Enter ID: 108
Enter Title: harc
Enter Artist: Kimj
Enter Duration: 5.41
Invalid
```

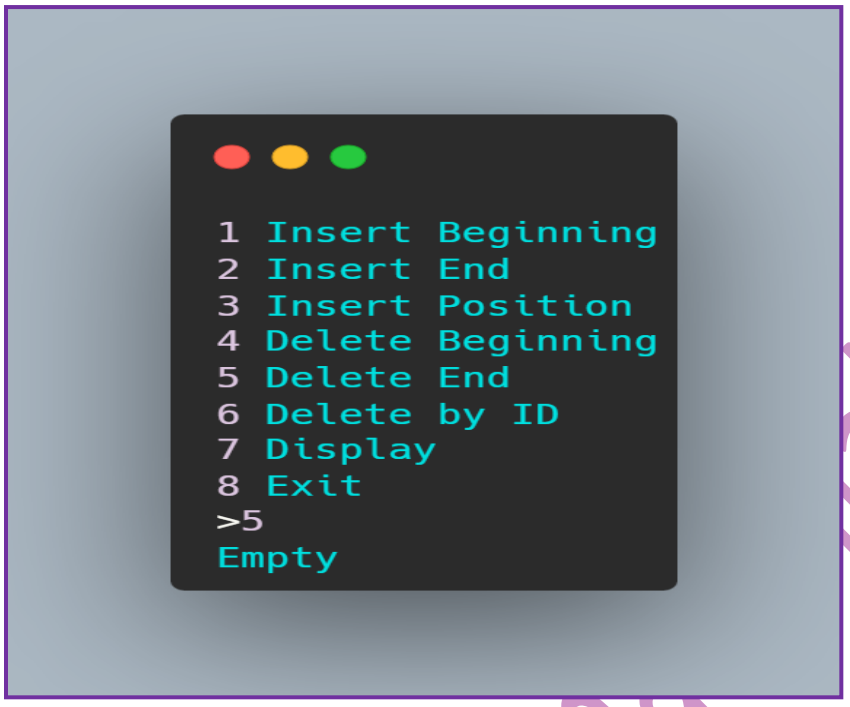
Delete beginning When list empty



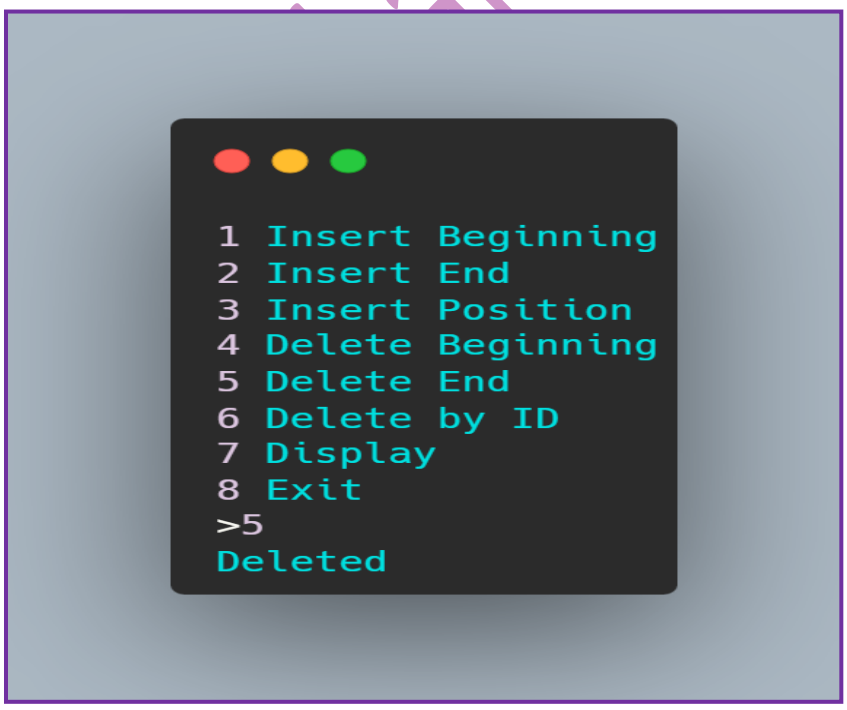
When Delete beginning (1 or more nodes)



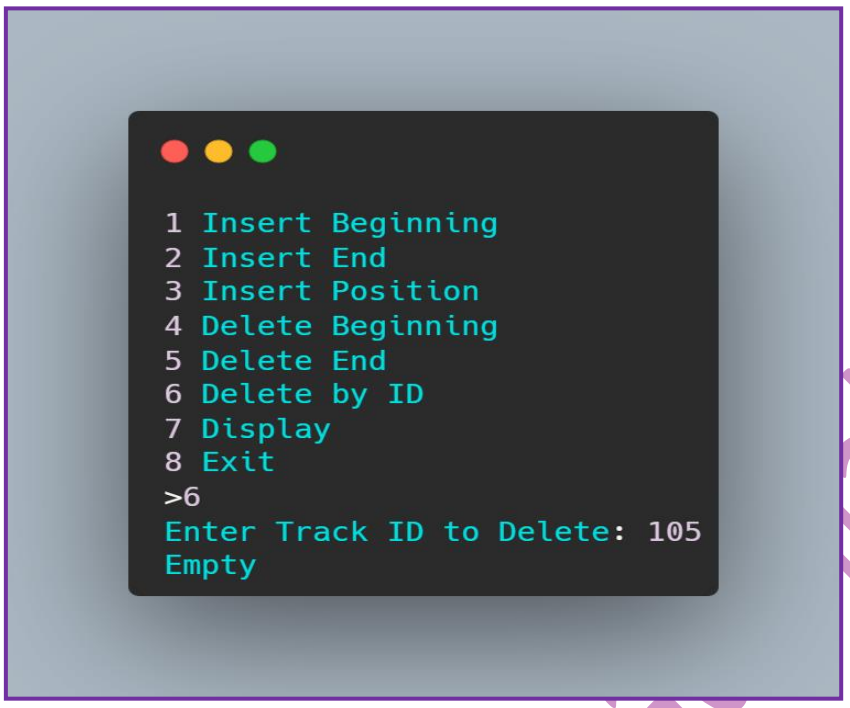
Delete end **When** list empty



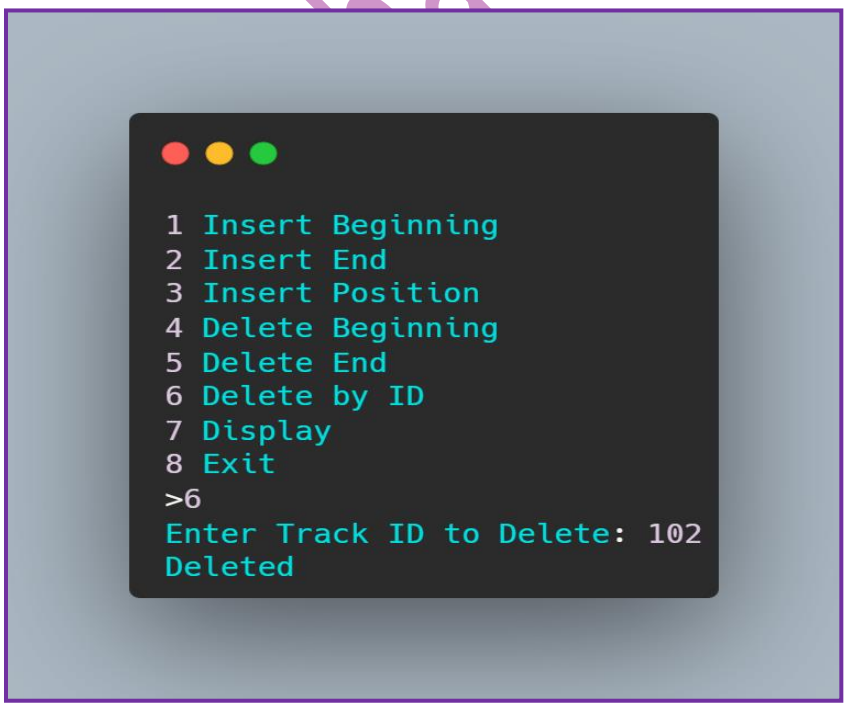
When Delete end (1 or more nodes)



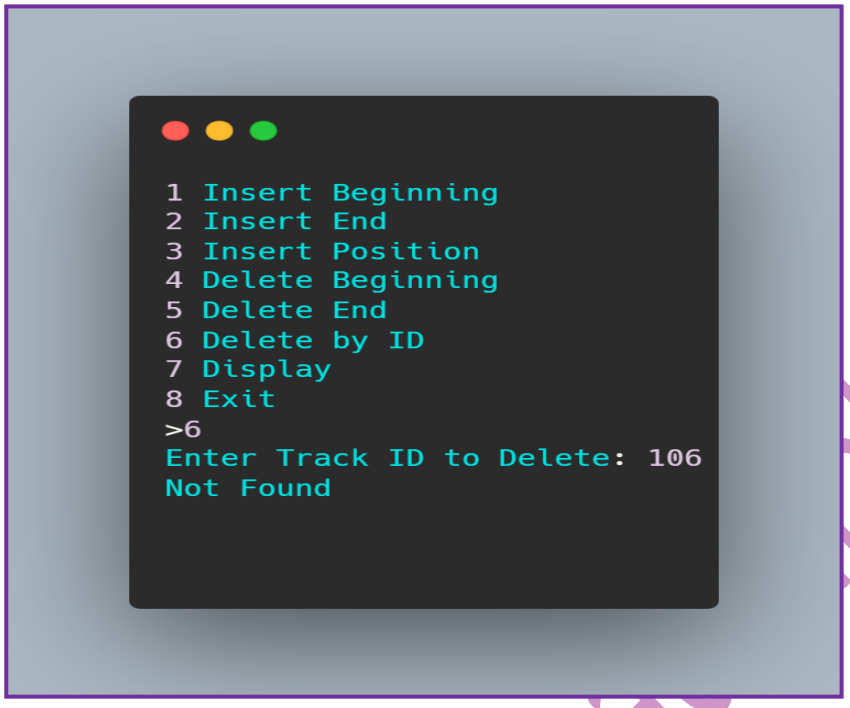
Delete by ID When list empty



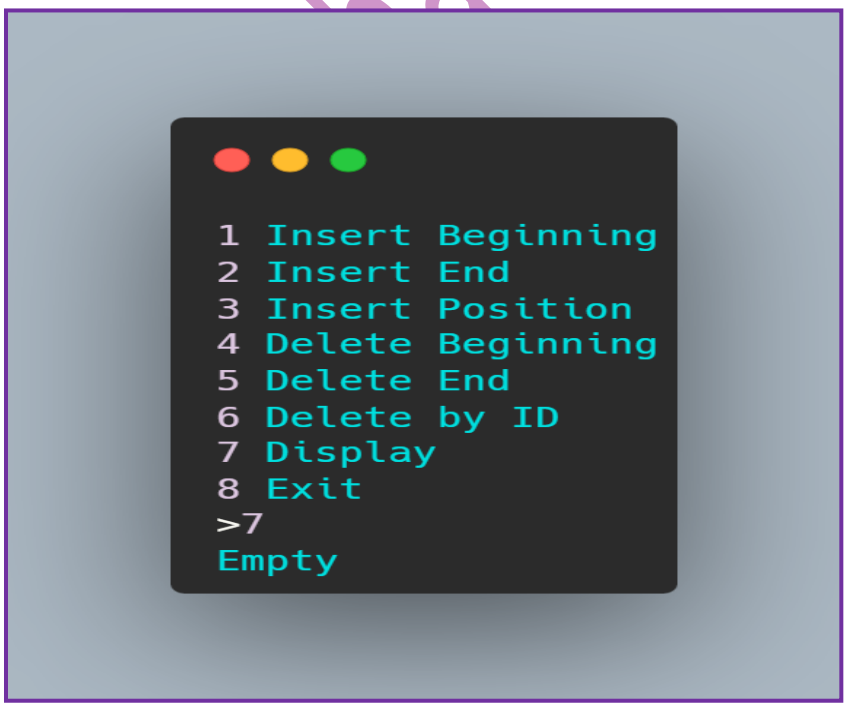
Delete by ID When ID found



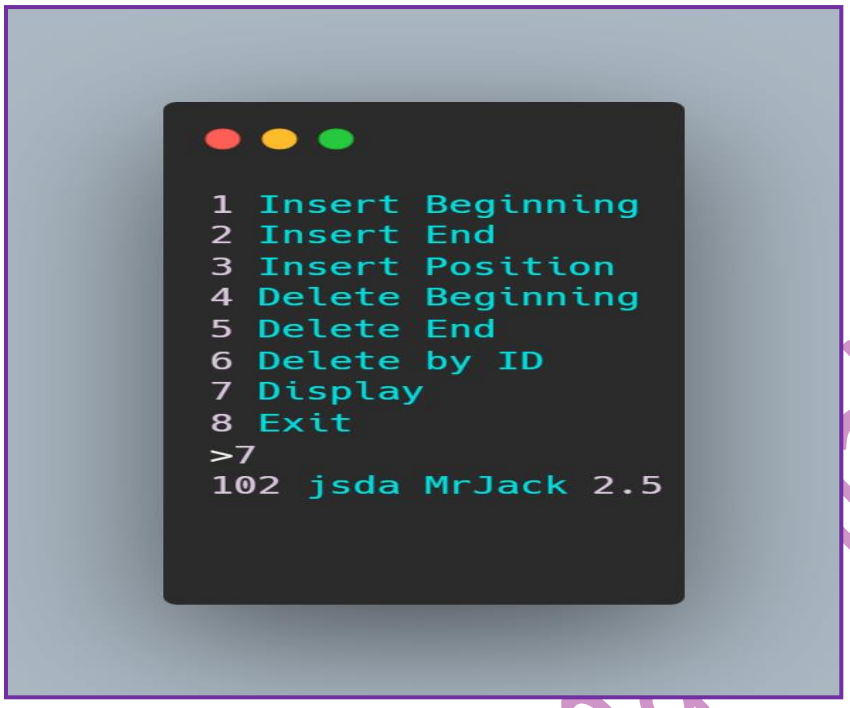
Delete by ID When ID not found



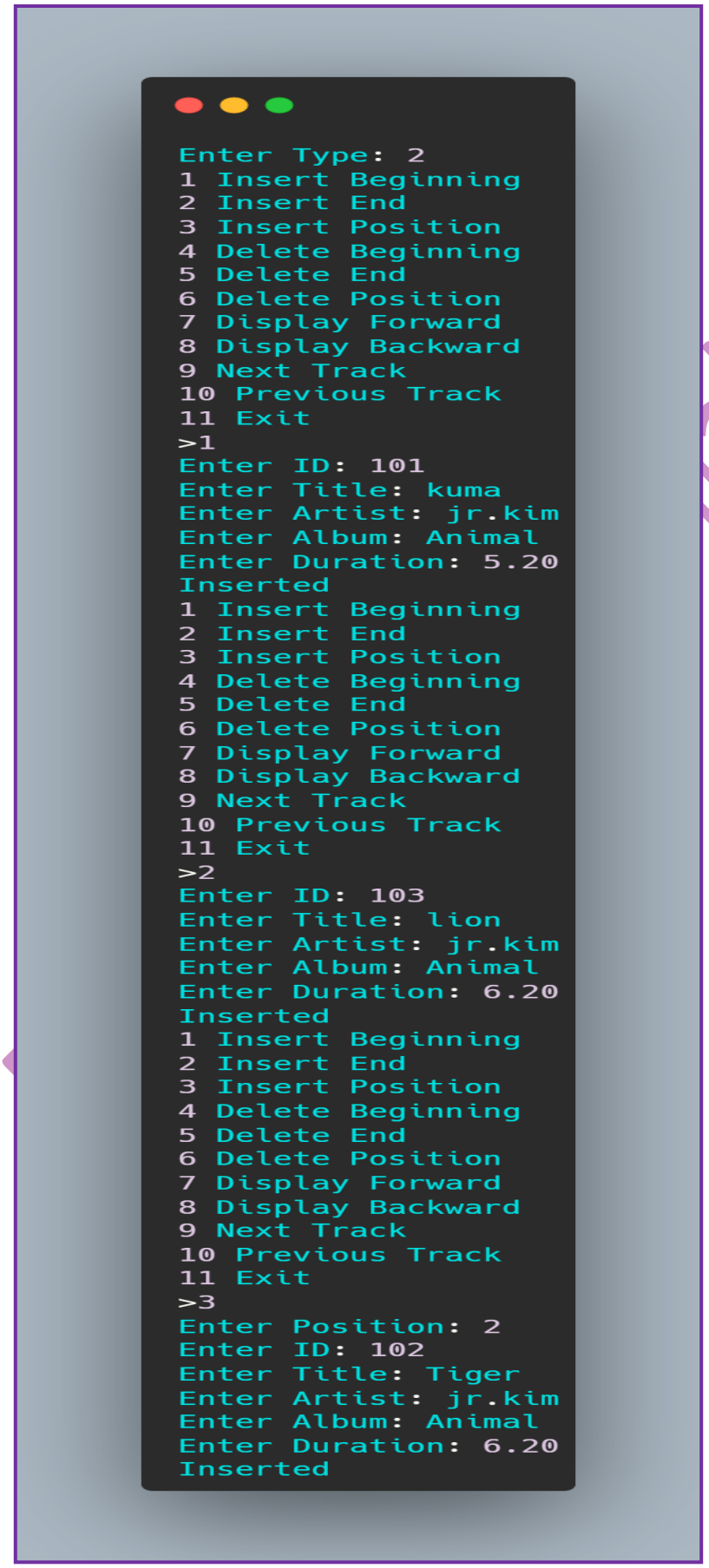
Display When list empty



Display **When** list not empty

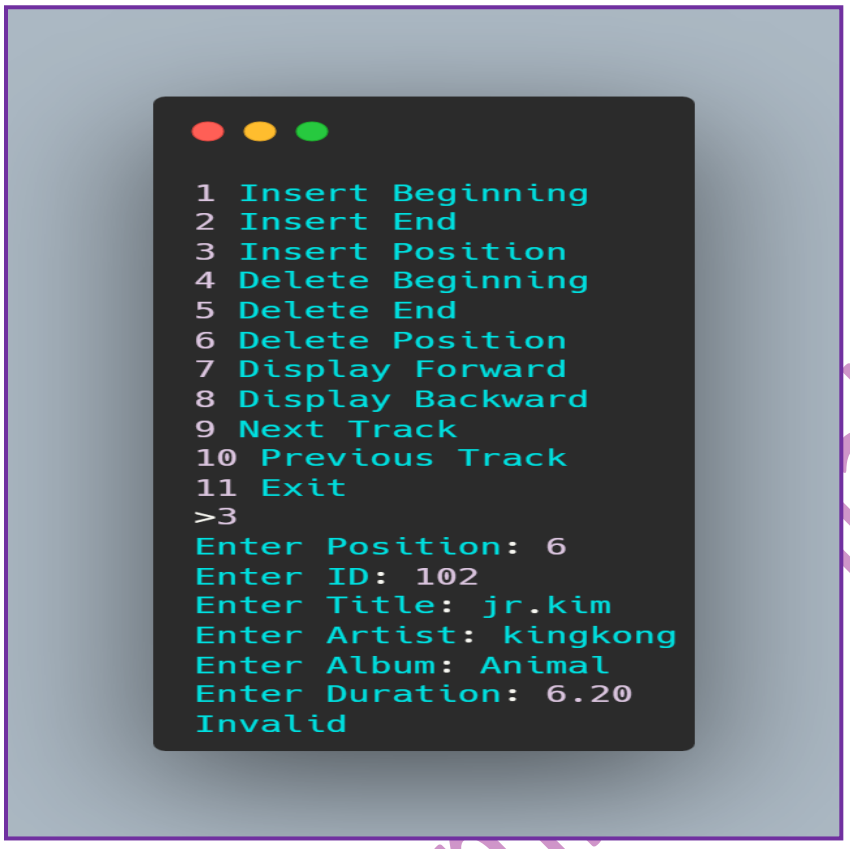


When Insert (beginning / end / valid position)

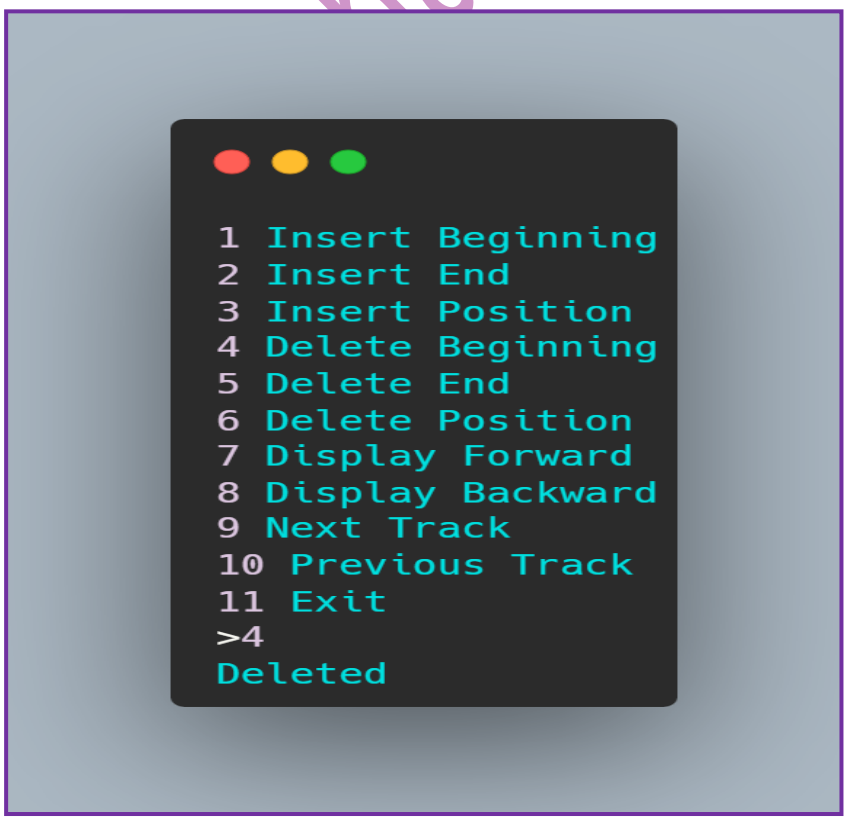


```
Enter Type: 2
1 Insert Beginning
2 Insert End
3 Insert Position
4 Delete Beginning
5 Delete End
6 Delete Position
7 Display Forward
8 Display Backward
9 Next Track
10 Previous Track
11 Exit
>1
Enter ID: 101
Enter Title: kuma
Enter Artist: jr.kim
Enter Album: Animal
Enter Duration: 5.20
Inserted
1 Insert Beginning
2 Insert End
3 Insert Position
4 Delete Beginning
5 Delete End
6 Delete Position
7 Display Forward
8 Display Backward
9 Next Track
10 Previous Track
11 Exit
>2
Enter ID: 103
Enter Title: lion
Enter Artist: jr.kim
Enter Album: Animal
Enter Duration: 6.20
Inserted
1 Insert Beginning
2 Insert End
3 Insert Position
4 Delete Beginning
5 Delete End
6 Delete Position
7 Display Forward
8 Display Backward
9 Next Track
10 Previous Track
11 Exit
>3
Enter Position: 2
Enter ID: 102
Enter Title: Tiger
Enter Artist: jr.kim
Enter Album: Animal
Enter Duration: 6.20
Inserted
```

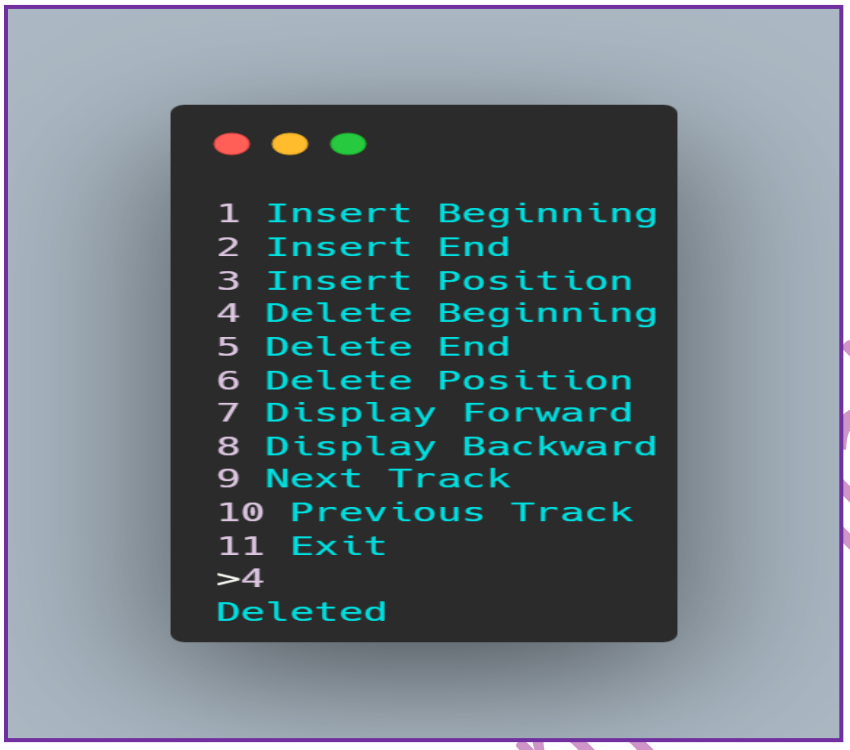
When Insert position invalid



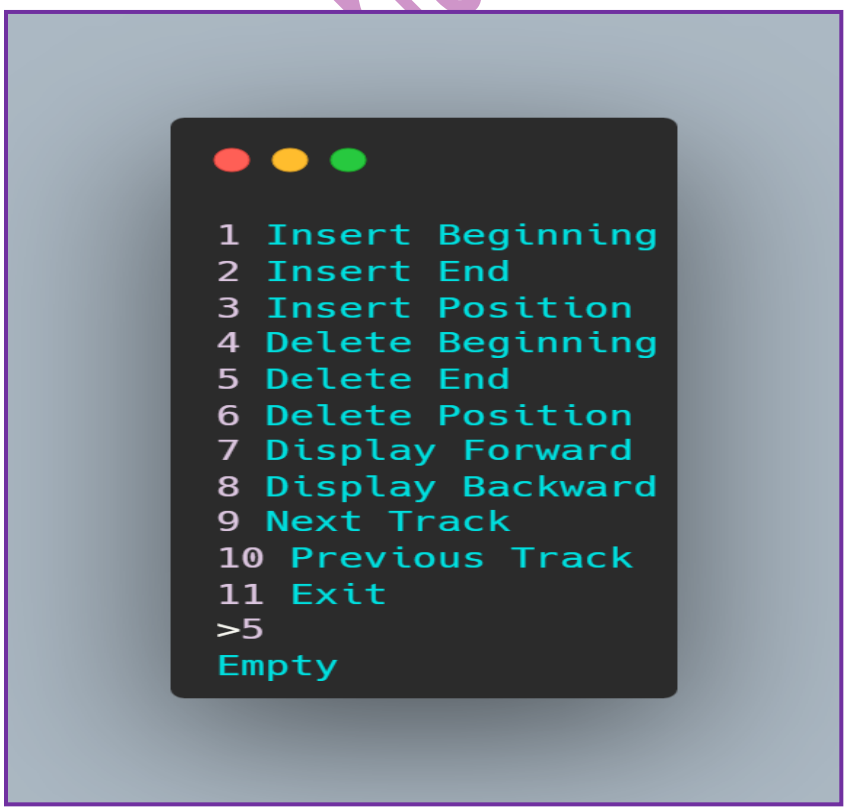
Delete beginning When list empty



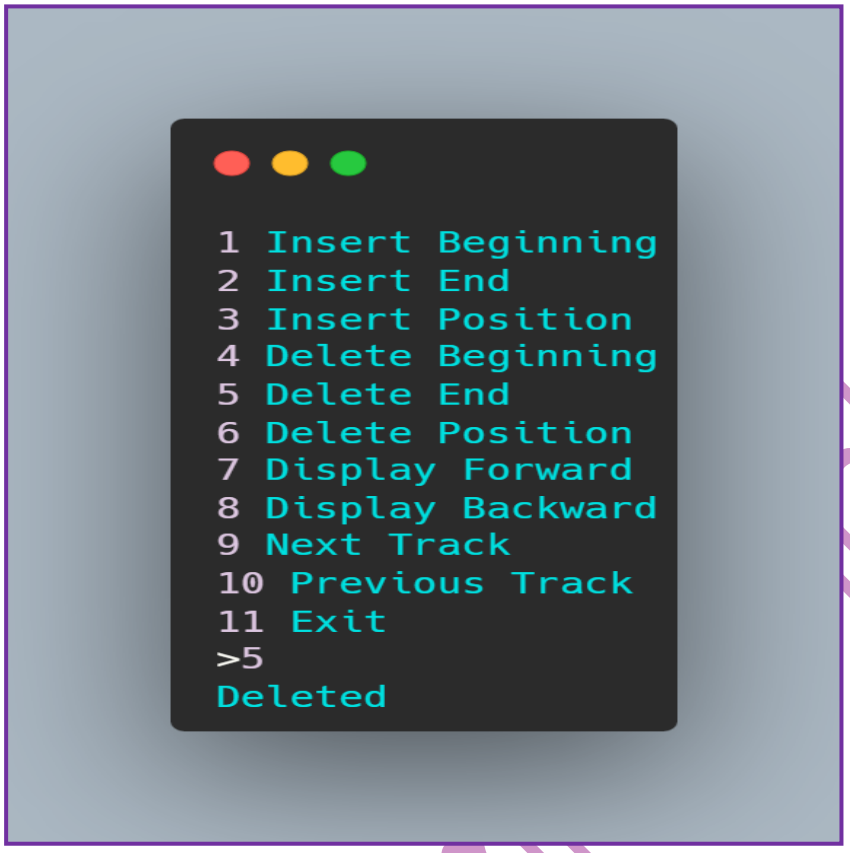
When Delete beginning (1 or more nodes)



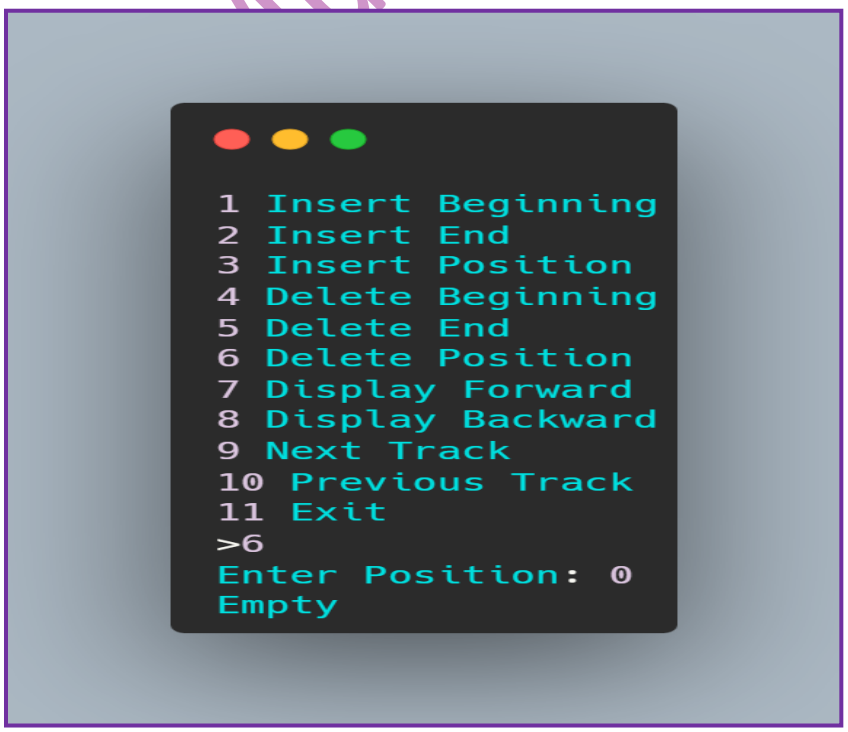
Delete end When list empty



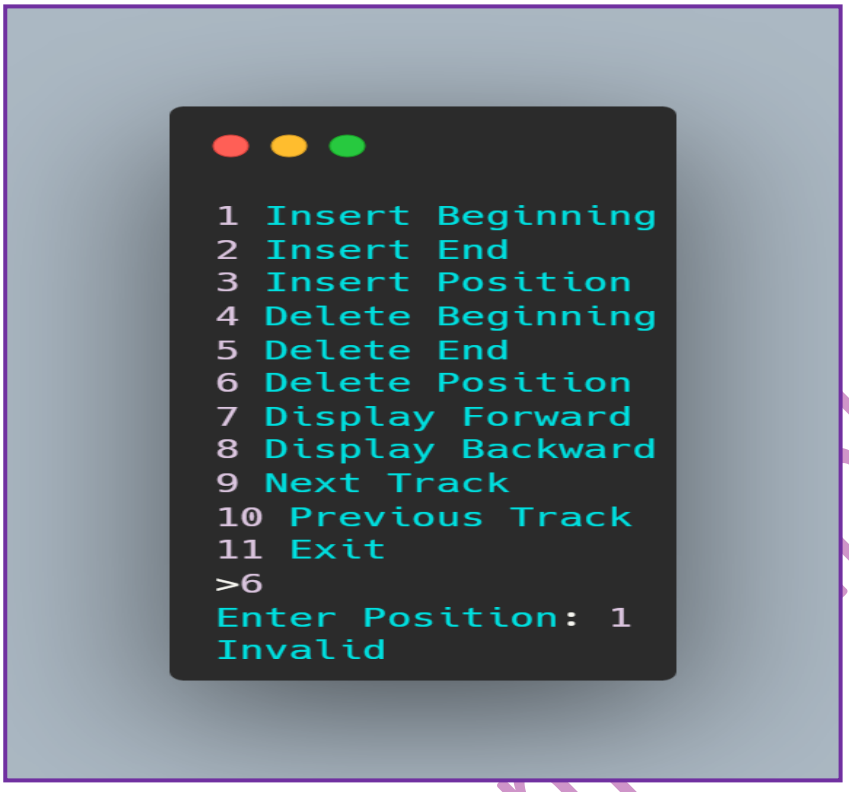
When Delete end (1 or more nodes)



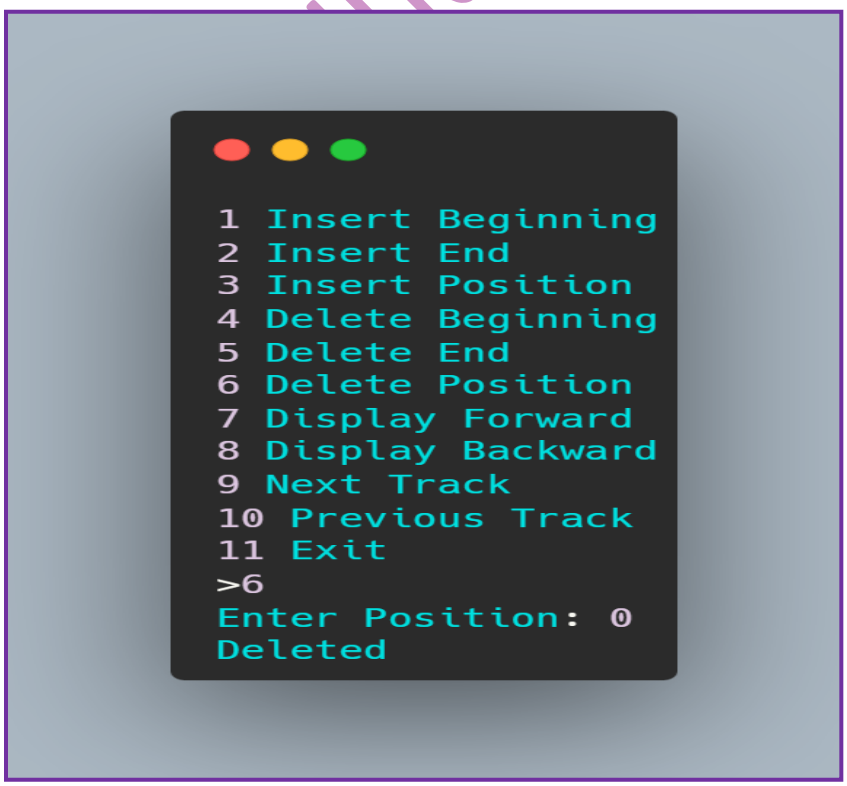
Delete position When list empty



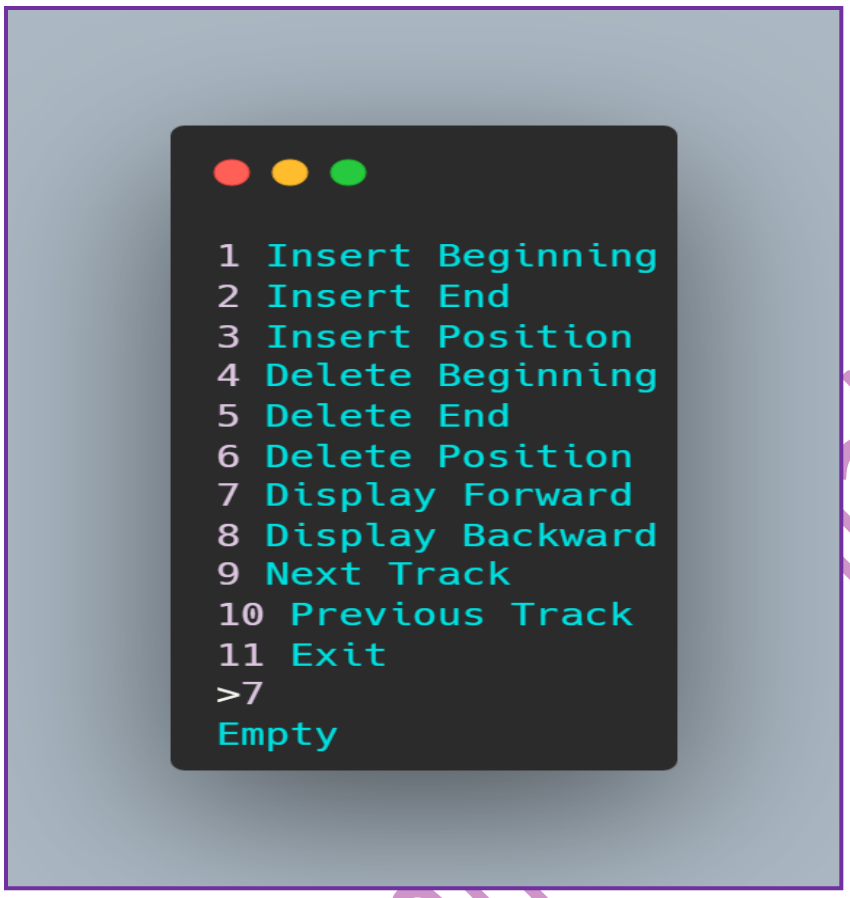
When Delete position invalid



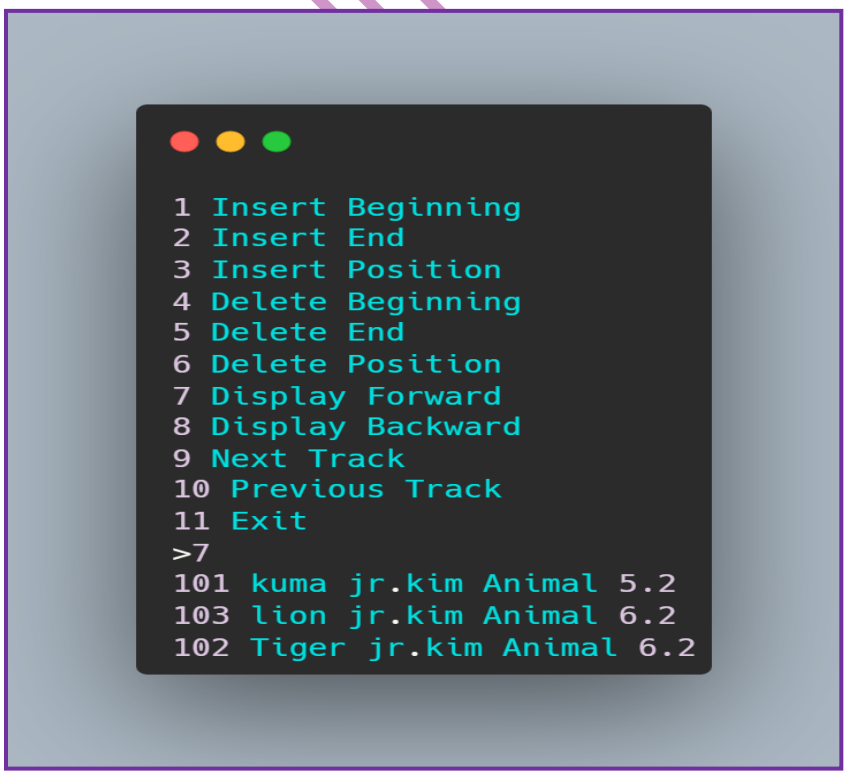
When Delete position valid



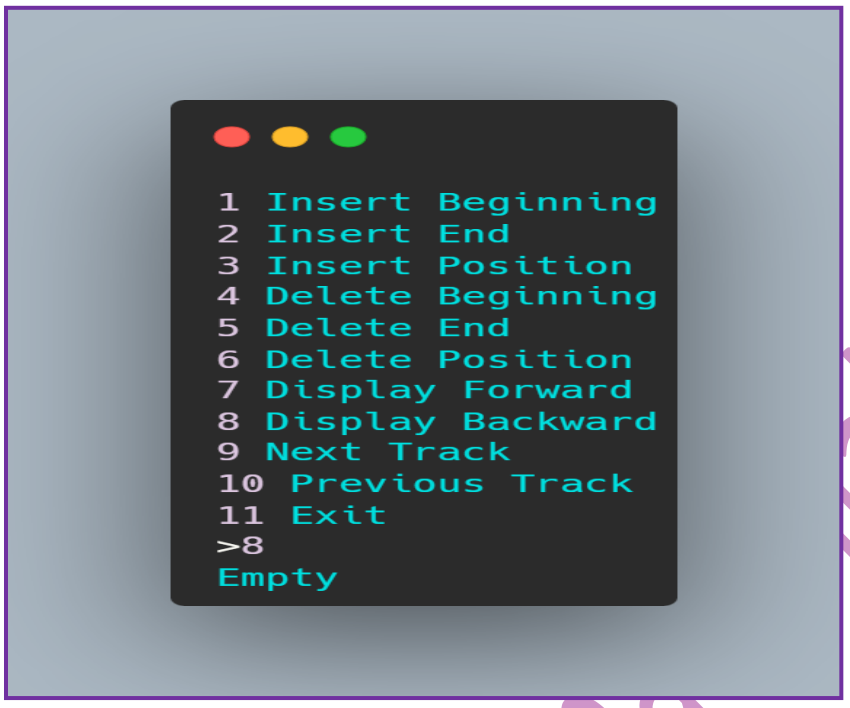
Display forward When empty



Display forward When not empty

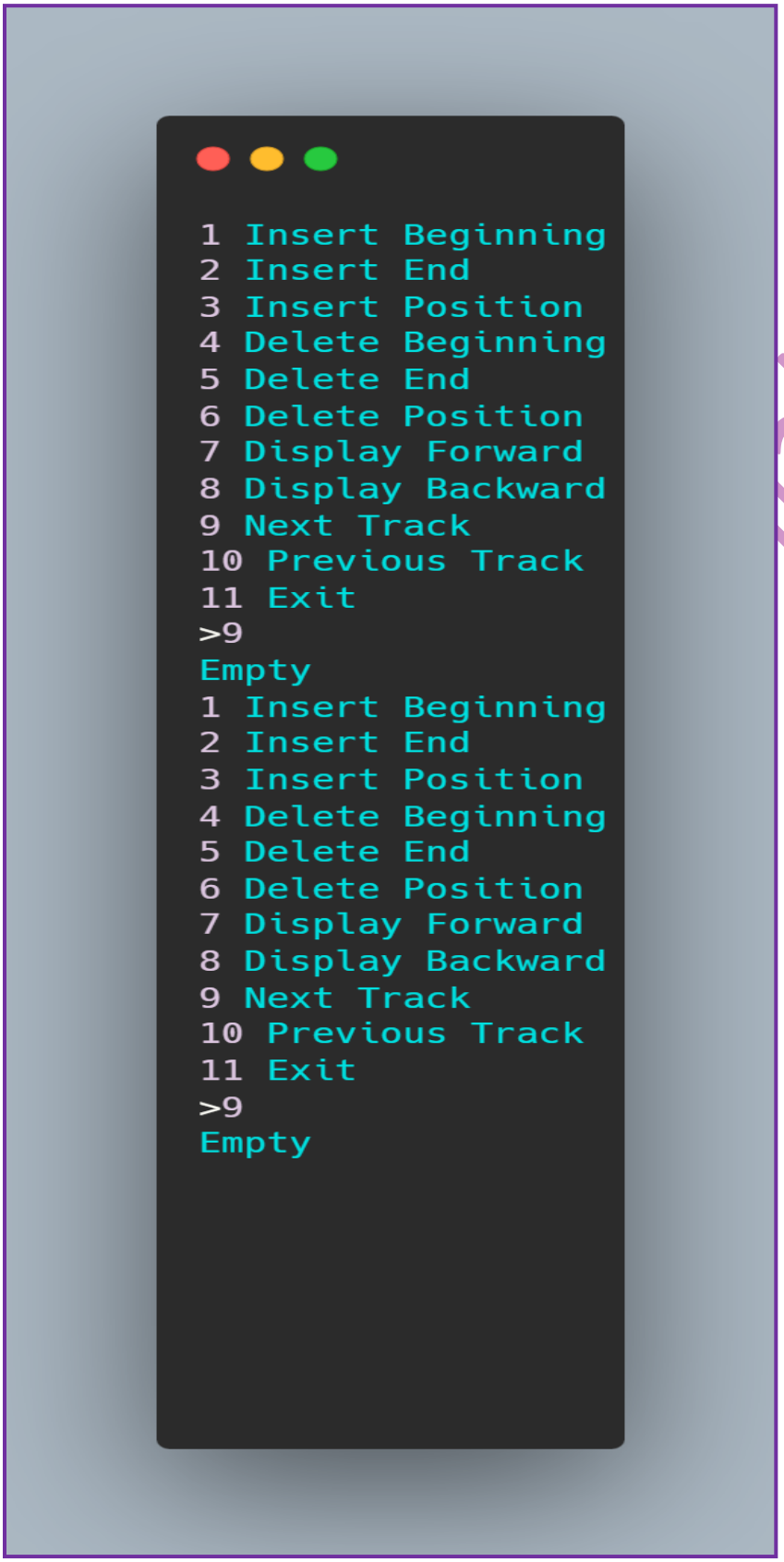


Display backward When empty

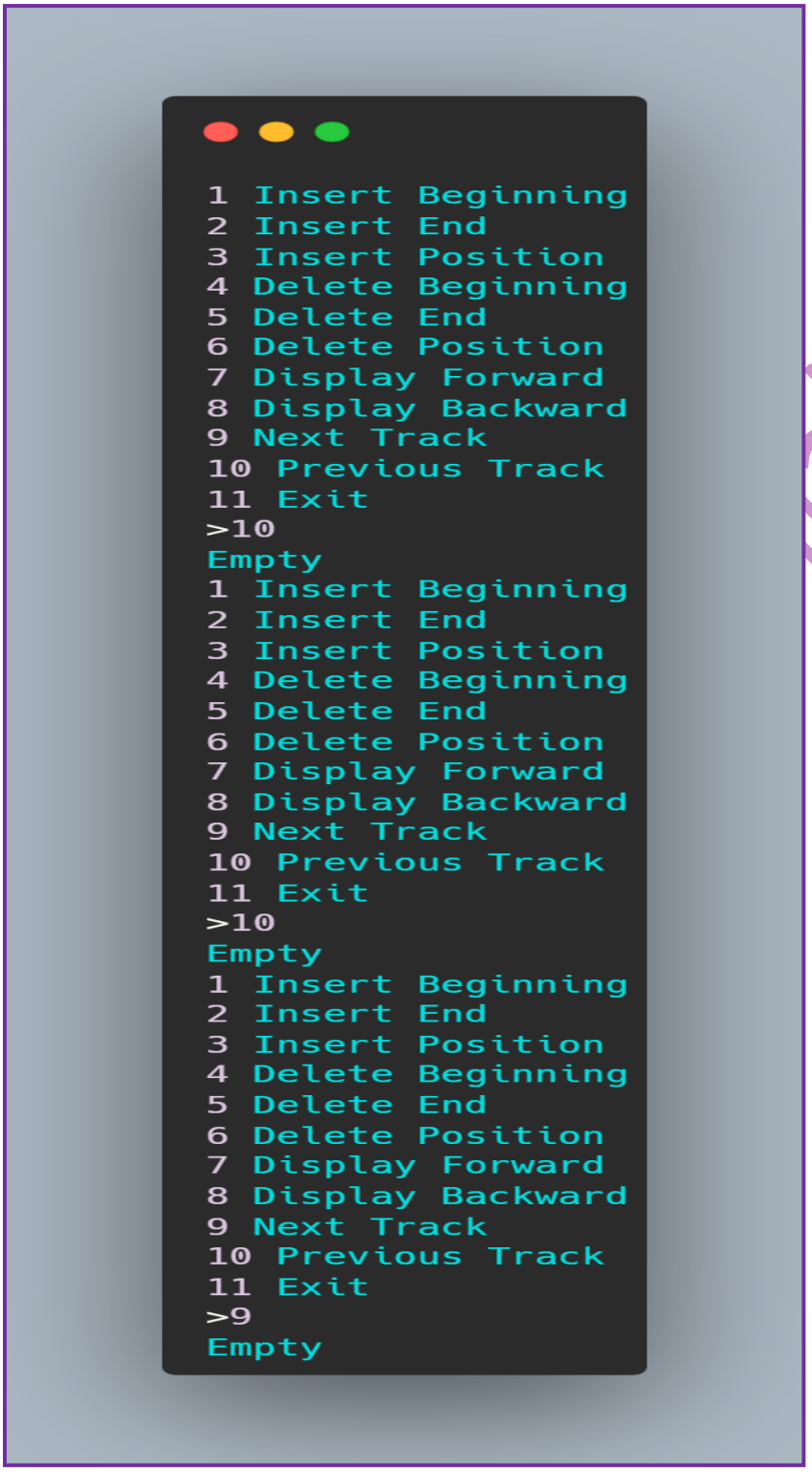


Display backward When not empty

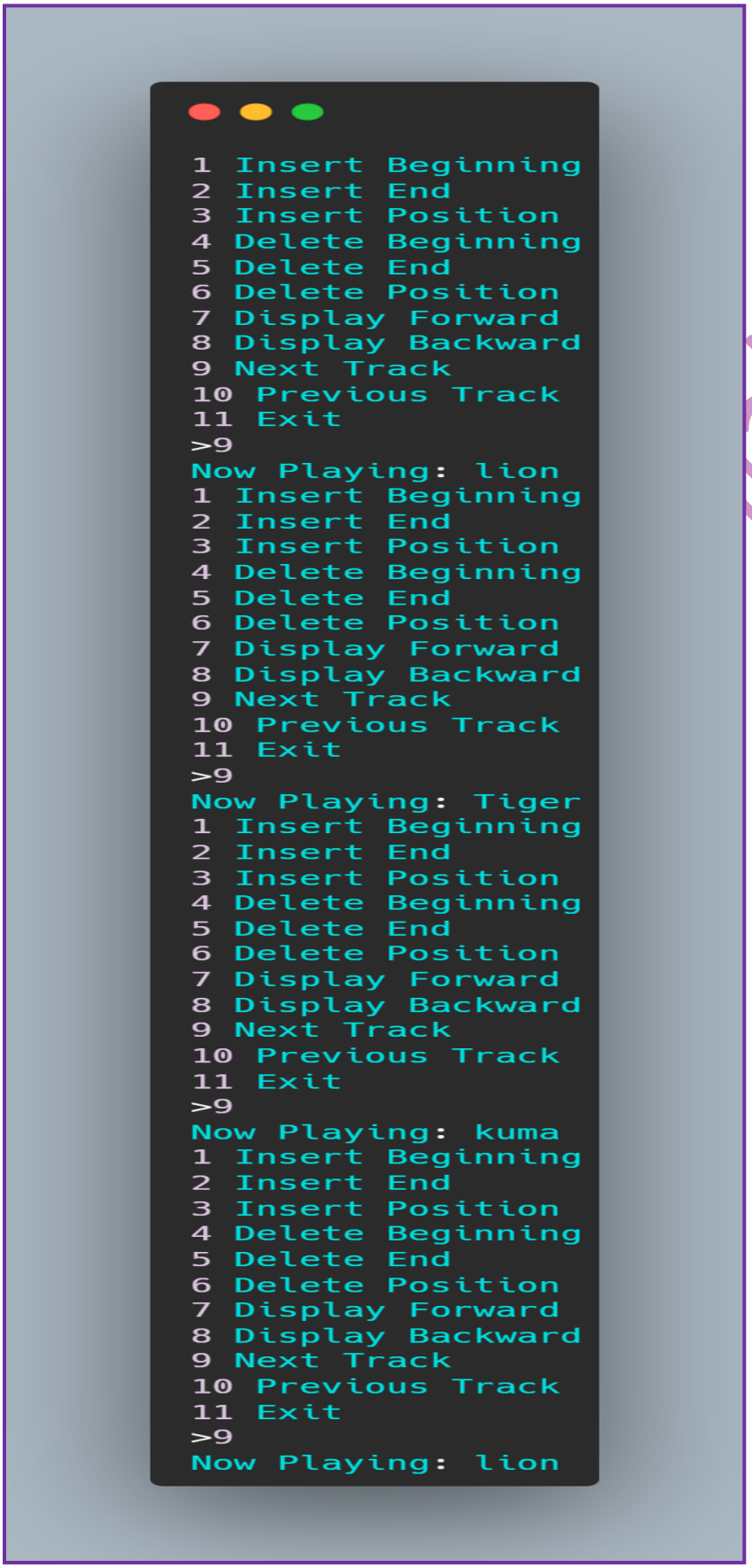
Play next **When** no track



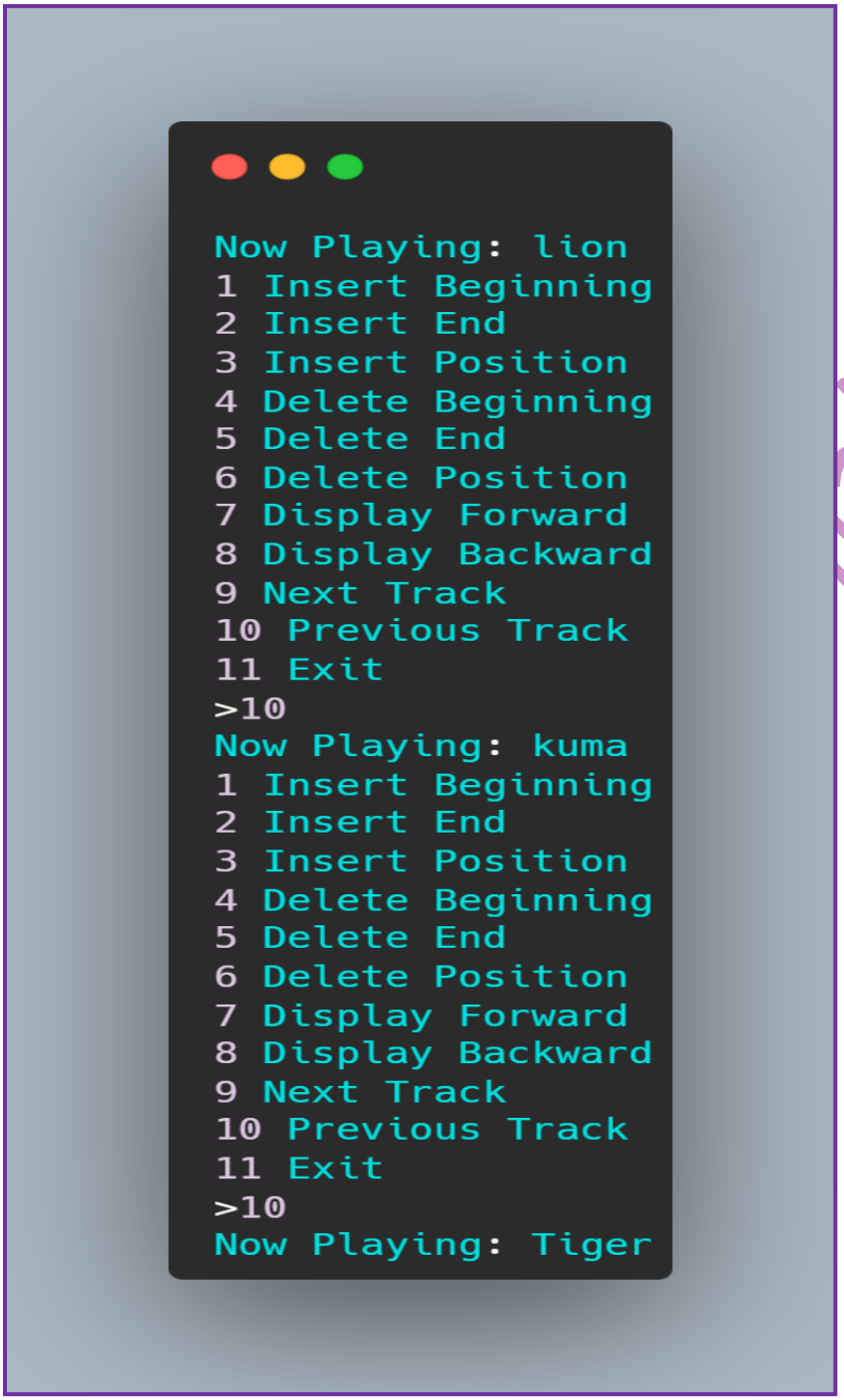
Play previous **When** no track



When Play next (track exists)



When Play previous (track exists)



Click here to Get this code on GitHub

Click here to Test this Code by Yourself.