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


class Songs {
    private:
        string songs[20];
        int SN;
    public:
        Songs() {
            string initialSongs[10] = {"Bohemian Rhapsody", "Hotel California", "Stairway to Heaven", "Imagine", "Smells Like
            for (int i = 0; i < 10; i++) {
                songs[i] = initialSongs[i];
            }
            SN = 10;
        }
        void addSong(string s) {
            if (SN < 20) {
                songs[SN] = s;
                SN++;
            } else {
                cout << "Song is full. Can't add a new song." << endl;
            }
        }
        void displaySongs() {
```

```
    for (int i = 0; i < SN; i++) {
        cout << songs[i] << endl;
    }
}
void sortSongs() {
    for (int i = 1; i < SN; i++) {
        string key = songs[i];
        int j = i - 1;
        while (j >= 0 && songs[j] > key) {
            songs[j + 1] = songs[j];
            j = j - 1;
        }
        songs[j + 1] = key;
    }
}
void inverseSort() {
    for (int i = 1; i < SN; i++) {
        string key = songs[i];
        int j = i - 1;
        while (j >= 0 && songs[j] < key) {
            songs[j + 1] = songs[j];
            j = j - 1;
        }
        songs[j + 1] = key;
    }
}
```

```
};


class Node {
    public:
        string value;
        Node* next;
        Node* pre;
        Node(string s1) {
            value = s1;
            next = NULL;
            pre = NULL;
        }
};


class SearchHistory {
    private:
        Node* top;
    public:
        SearchHistory() {
            top = NULL;
        }
        bool isEmpty() {
            return top == NULL;
        }
```

```
bool isEmpty() {
    return top == NULL;
}
void addHistory(string s) {
    Node* newNode = new Node(s);
    if (isEmpty()) {
        top = newNode;
        return;
    }
    newNode->pre = top;
    top->next = newNode;
    top = newNode;
}
void deleteHistory() {
    if (isEmpty()) {
        cout << "History is already empty." << endl;
        return;
    }
    Node* temp = top;
    top = top->pre;
    delete temp;
}
void showHistory() {
    if (isEmpty()) {
        cout << "History is Empty." << endl;
        return;
```

```
void deleteHistory() {
    if (isEmpty()) {
        cout << "History is already empty." << endl;
        return;
    }
    Node* temp = top;
    top = top->pre;
    delete temp;
}
void showHistory() {
    if (isEmpty()) {
        cout << "History is Empty." << endl;
        return;
    }
    Node* temp = top;
    cout << "Search history.\n" << endl;
    while (temp != NULL) {
        cout << temp->value << endl;
        temp = temp->pre;
    }
}
void searchBrowser() {
    string Q1 = "what is dsa?";
    string Q2 = "what is stack?";
    string Q3 = "what is queue?";
    string userQ;
    cout << "Enter search query: ";
    getline(cin, userQ);
    string history = userQ;
    addHistory(history);
    if (userQ == Q1) {
        cout << "Data Structures and Algorithms; techniques for organizing and processing data." << endl;
    } else if (userQ == Q2) {
        cout << "LIFO data structure, last element added is first removed." << endl;
    } else if (userQ == Q3) {
        cout << "FIFO data structure, first element added is first removed." << endl;
```

```
public:
    string task;
    TaskNode* next;
    TaskNode(string t) {
        task = t;
        next = NULL;
    }
};

class TaskScheduler {
public:
    TaskNode* rear;
    TaskNode* front;
    TaskScheduler() {
        rear = front = NULL;
    }
    bool isempty() {
        return front == NULL;
    }
    void Enqueue(string task) {
        TaskNode* newnode = new TaskNode(task);
        if (isempty()) {
            front = rear = newnode;
        } else {
            rear->next = newnode;
            rear = newnode;
        }
        cout << "Task is Added: " << task << endl;
    }
    void dequeue() {
        if (isempty()) {
            cout << "There is no task." << endl;
            return;
        }
        TaskNode* temp = front;
        string task = front->task;
        front = front->next;
```

```
struct treeNode {
    int id;
    string name;
    treeNode* left;
    treeNode* right;


    treeNode(int id, string name) {
        this->id = id;
        this->name = name;
        left = right = nullptr;
    }
};

class EmployeeBST {
public:
    treeNode* root;

    EmployeeBST() {
        root = nullptr;
    }


    void addEmployee(int id, string name) {
        root = insert(root, id, name);
    }


    void removeEmployee(int id) {
        root = deleteNode(root, id);
    }


    treeNode* findEmployee(int id) {
        return search(root, id);
    }
```

```
treeNode* deleteNode(treeNode* node, int id) {
    if (node == nullptr) return node;
    if (id < node->id) {
        node->left = deleteNode(node->left, id);
    } else if (id > node->id) {
        node->right = deleteNode(node->right, id);
    } else {
        if (node->left == nullptr) {
            treeNode* temp = node->right;
            delete node;
            return temp;
        } else if (node->right == nullptr) {
            treeNode* temp = node->left;
            delete node;
            return temp;
        }
        treeNode* temp = minValueNode(node->right);
        node->id = temp->id;
        node->name = temp->name;
        node->right = deleteNode(node->right, temp->id);
    }
    return node;
}


treeNode* minValueNode(treeNode* node) {
    treeNode* current = node;
    while (current && current->left != nullptr) {
        current = current->left;
    }
    return current;
}
```

```
    void inorder(treeNode* node) {
        if (node != nullptr) {
            inorder(node->left);
            cout << "ID: " << node->id << ", Name: " << node->name << endl;
            inorder(node->right);
        }
    }
};



int main() {
    Songs songs;
    SearchHistory history;
    TaskScheduler tasks;
    EmployeeBST tree;

    int choice;
    do {
        cout << "\n--- Home Screen ---\n";
        cout << "1. Manage Songs\n";
        cout << "2. Manage Search History\n";
        cout << "3. Manage Tasks\n";
        cout << "4. Manage Binary Search Tree\n";
        cout << "5. Exit\n";
        cout << "Enter your choice: ";
        cin >> choice;
        cin.ignore();

        switch (choice) {
            case 1: {
                int songChoice;
                do {
                    cout << "\n--- Manage Songs ---\n";
```

```
case 1: {
    int songChoice;
    do {
        cout << "\n--- Manage Songs ---\n";
        cout << "1. Display Songs\n";
        cout << "2. Add a Song\n";
        cout << "3. Sort Songs\n";
        cout << "4. Inverse Sort Songs\n";
        cout << "5. Back to Home Screen\n";
        cout << "Enter your choice: ";
        cin >> songChoice;
        cin.ignore();
        switch (songChoice) {
            case 1: songs.displaySongs(); break;
            case 2: {
                string song;
                cout << "Enter song name: ";
                getline(cin, song);
                songs.addSong(song);
                break;
            }
            case 3: songs.sortSongs(); break;
            case 4: songs.inverseSort(); break;
            case 5: break;
            default: cout << "Invalid choice. Try again.\n";
        }
    } while (songChoice != 5);
    break;
}
case 2: {
    int historyChoice;
    do {
        cout << "\n--- Manage Search History ---\n";
        cout << "1. Search Browser\n";
        cout << "2. Show History\n";
        cout << "3. Delete Last History\n";
        cout << "4. Back to Home Screen\n";
```

```
                    break;
                }
                case 2: {
                    int id;
                    cout << "Enter employee ID to delete: ";
                    cin >> id;
                    tree.removeEmployee(id);
                    break;
                }
                case 3: {
                    int id;
                    cout << "Enter employee ID to search: ";
                    cin >> id;
                    treeNode* result = tree.findEmployee(id);
                    if (result != nullptr) {
                        cout << "Employee found: ID = " << result->id << ", Name = " << result->n
                    } else {
                        cout << "Employee not found.\n";
                    }
                    break;
                }
                case 4: tree.displayEmployees(); break;
                case 5: break;
                default: cout << "Invalid choice. Try again.\n";
            }
        } while (treeChoice != 5);
        break;
    }
    case 5: cout << "Exiting program.\n"; break;
    default: cout << "Invalid choice. Try again.\n";
  }
} while (choice != 5);

return 0;
```

```
--- Home Screen ---
1. Manage Songs
2. Manage Search History
3. Manage Tasks
4. Manage Binary Search Tree
5. Exit
Enter your choice: 1

--- Manage Songs ---
1. Display Songs
2. Add a Song
3. Sort Songs
4. Inverse Sort Songs
5. Back to Home Screen
Enter your choice: 1
Bohemian Rhapsody
Hotel California
Stairway to Heaven
Imagine
Smells Like Teen Spirit
Hey Jude
Sweet Child O' Mine
Wonderwall
Shape of You
Blinding Lights

--- Manage Songs ---
1. Display Songs
2. Add a Song
3. Sort Songs
4. Inverse Sort Songs
5. Back to Home Screen
Enter your choice: 2
Enter song name: tu hi tu ha

--- Manage Songs ---
1. Display Songs
2. Add a Song
3. Sort Songs
4. Inverse Sort Songs
5. Back to Home Screen
Enter your choice: 5
```