```cpp
#include "TO_DO_LIST.h"
#include <iostream>
using namespace std;
// Constructor
sortedlinkedlist::sortedlinkedlist()
{
    listdata = nullptr;
}
// Insert method
void sortedlinkedlist::insert(const string& task) {
    node* temp = listdata;
    node* t = nullptr; // Initialize t to nullptr
    while (temp != nullptr)
    {
        if (task < temp->info)
            break;
        t = temp;
        temp = temp->next;
    }
    node* location = new node;
    location->info = task;
    location->isDone = false; // Default to not done
    if (temp == listdata)
    {
        location->next = listdata;
        listdata = location;
    }
    else
    {
        t->next = location;
        location->next = temp;
    }
}
// Search method
bool sortedlinkedlist::search(const string& task) const {
    node* temp = listdata;
    while (temp != nullptr)
    {
        if (task == temp->info)
            return true;
        temp = temp->next;
    }
    return false;
}
// Delete method
void sortedlinkedlist::del(const string& task) {
    node* temp = listdata;
    node* t = nullptr;
    while (temp != nullptr)
    {
        if (task == temp->info)
            break;
        t = temp;
```

```
        temp = temp->next;
    }
    if (temp == listdata)
    {
        listdata = listdata->next;
        delete temp;
    }
    else
    {
        if (temp != nullptr)
        {
            t->next = temp->next;
            delete temp;
        }
    }
}
// Make empty method
void sortedlinkedlist::makeempty() {
    node* temp;
    while (listdata != nullptr)
    {
        temp = listdata;
        listdata = listdata->next;
        delete temp;
    }
}
// Get length method
int sortedlinkedlist::getlength() const {
    node* temp = listdata;
    int count = 0;
    while (temp != nullptr)
    {
        count++;
        temp = temp->next;
    }
    return count;
}
// Display method
void sortedlinkedlist::display() const {
    if (listdata == nullptr)
    {
        cout << " list is empty" << endl;
    }
    node* temp = listdata;
    while (temp != nullptr)
    {
        cout << temp->info << " [" << (temp->isDone ? "done" : "not done")
            << "]" << endl;
        temp = temp->next;
    }
}
// Is full method
bool sortedlinkedlist::isfull() const {
```

```
    node* temp;
    try
    {
        temp = new node;
        delete temp;
        return false;
    }
    catch (bad_alloc exception)
    {
        return true;
    }
}
// Add task method
void sortedlinkedlist::addTask(const string& task) {
    insert(task);
}
// Remove task method
void sortedlinkedlist::removeTask(const string& task) {
    del(task);
}
// Display tasks method
void sortedlinkedlist::displayTasks() const {
    display();
}
// Check if task is done method
bool sortedlinkedlist::isTaskDone(const string& task) const {

    node* temp = listdata;
    while (temp != nullptr)
    {
        if (task == temp->info)
            return temp->isDone;
        temp = temp->next;
    }
    return false;
}
// Mark task as done method
void sortedlinkedlist::markTaskDone(const string& task) {

    node* temp = listdata;
    while (temp != nullptr)
    {
        if (task == temp->info)
        {
            temp->isDone = true;
            return;
        }
        temp = temp->next;
    }
    cout << "Task not found: " << task << endl;
}
void sortedlinkedlist::counttasksdone() {
    if (listdata == nullptr)
```

```
    {
        cout << " list is empty" << endl;
        return;
    }
    else
    {
        node* temp = listdata;
        int count = 0;
        while (temp != nullptr)
        {
            count++;
            temp = temp->next;

        }
        cout << " count of tasks = " << count << endl;
    }

}
// Display uncompleted tasks method
void sortedlinkedlist::displayUncompletedTasks() const {
    node* temp = listdata;
    while (temp != nullptr) {
        if (!temp->isDone) {
            cout << temp->info << " [not done]" << endl;
        }
        temp = temp->next;
    }
}
```