VIT®
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

School of Computer Science and Engineering

Winter Semester 2023-24

Continuous Assessment Test – II

SLOT:C1

Programme Name & Branch:B.Tech(Computer Science and Engineering)

Course Name & Code: Structured and Object-Oriented Programming (BCSE102L)

Class Number (s): Common to All C1 Slots

Faculty Name (s): Dr. Senthilkumar K

Exam Duration: 90 Min. **Maximum Marks: 50**

General instruction(s):

Since CAT -2 will be conducted in open book mode, Students can carry handwritten notes and/or text books

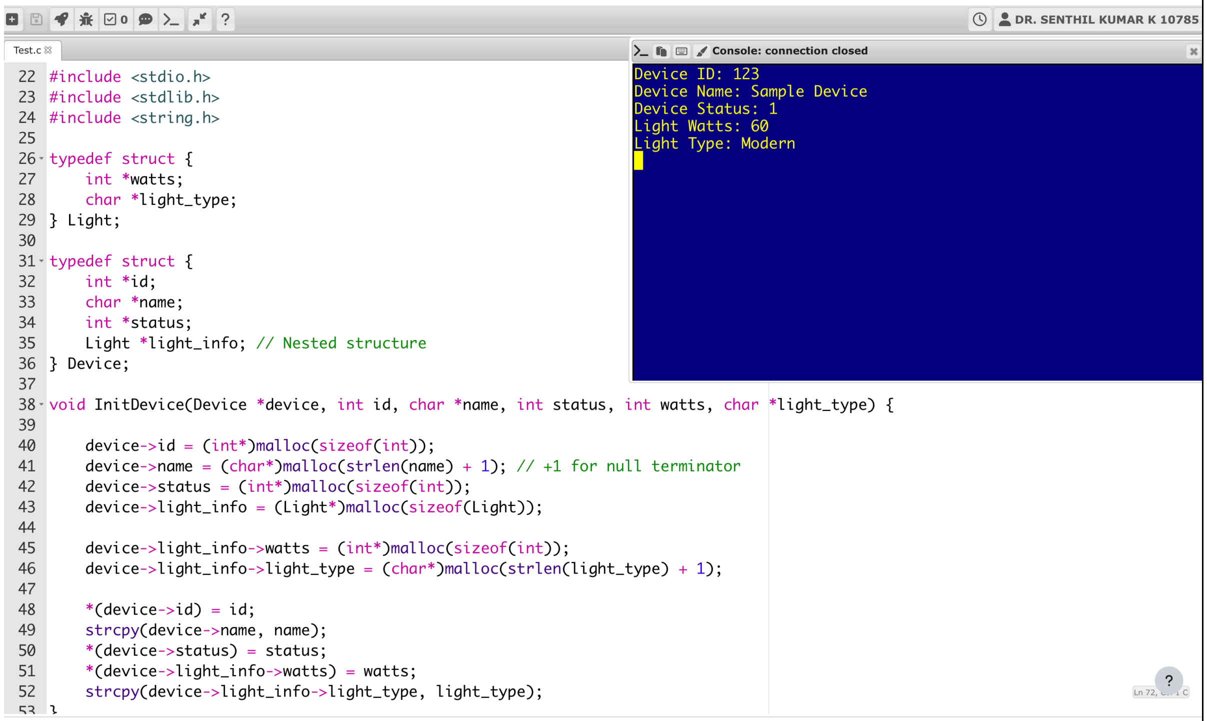
Answer ALL the Questions(5 * 10 = 50 marks)

Carefully read the following scenario, which will be helpful for ALL the questions that follow:		
In Bombay, **Jass**, a skilled software engineer, was approached by TechGenius, a local startup, to contribute to their smart home automation project, "HomeGenius." Recognising the potential, **Jass** suggested an object-oriented approach, proposing a "Device" class to serve as the foundation for all devices, with specialised subclasses for specific functionalities. **Jass** then suggested deriving specialised device classes from the base "Device" class, like such as lights, thermostats, and security cameras. Each device would have common attributes like an ID, name, and status. **Jass** emphasised the importance of encapsulation, suggesting that private member variables should store device-specific details. **Sakthi**, TechGenius's CEO, embraced **Jass's** ideas, acknowledging the benefits of encapsulation and inheritance. They saw how this approach would facilitate scalability and adaptability, allowing for seamless integration of new devices in the future. Together, they aimed to create a system that could efficiently meet evolving technological and customer demands.		
Q. No.	**Question**	**Max Marks**
1.	Write a C program that defines a structure named "Device" with pointer attributes including an ID (integer), name (string), and status (boolean). Additionally, define a nested structure called "Light" within "Device," which features pointer attributes such as Watts (integer) and Light_Type (Modern/Classic). Implement a main() function to test the structure variables, utilising dynamic memory concepts (malloc()/calloc() function) for the pointer attributes in both structures (Device and Light). Furthermore, ensure the main() function reads values for these attributes and prints the results accordingly	10 **(CO2, BL3)**

Possible Solution:

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

typedef struct {
    int *watts;
    char *light_type;
} Light;

typedef struct {
    int *id;
    char *name;
    int *status;
    Light *light_info; // Nested structure
} Device;

void InitDevice(Device *device, int id, char *name, int status, int watts, char *light_type) {

    device->id = (int*)malloc(sizeof(int));
    device->name = (char*)malloc(strlen(name) + 1); // +1 for null terminator
    device->status = (int*)malloc(sizeof(int));
    device->light_info = (Light*)malloc(sizeof(Light));

    device->light_info->watts = (int*)malloc(sizeof(int));
    device->light_info->light_type = (char*)malloc(strlen(light_type) + 1);

    *(device->id) = id;
    strcpy(device->name, name);
    *(device->status) = status;
    *(device->light_info->watts) = watts;
    strcpy(device->light_info->light_type, light_type);
}

void FreeDevice(Device *device) {
    free(device->id);
    free(device->name);
    free(device->status);
    free(device->light_info->watts);
    free(device->light_info->light_type);
    free(device->light_info);
}



int main() {



    Device device;

    InitDevice(&device, 123, "Sample Device", 1, 60, "Modern");



    printf("Device ID: %d\n", *(device.id));
    printf("Device Name: %s\n", device.name);
    printf("Device Status: %d\n", *(device.status));
    printf("Light Watts: %d\n", *(device.light_info->watts));
    printf("Light Type: %s\n", device.light_info->light_type);

    FreeDevice(&device);

    return 0;
}
```

Sample Output:



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

typedef struct {
    int *watts;
    char *light_type;
} Light;

typedef struct {
    int *id;
    char *name;
    int *status;
    Light *light_info; // Nested structure
} Device;

void InitDevice(Device *device, int id, char *name, int status, int watts, char *light_type) {

    device->id = (int*)malloc(sizeof(int));
    device->name = (char*)malloc(strlen(name) + 1); // +1 for null terminator
    device->status = (int*)malloc(sizeof(int));
    device->light_info = (Light*)malloc(sizeof(Light));

    device->light_info->watts = (int*)malloc(sizeof(int));
    device->light_info->light_type = (char*)malloc(strlen(light_type) + 1);

    *(device->id) = id;
    strcpy(device->name, name);
    *(device->status) = status;
    *(device->light_info->watts) = watts;
    strcpy(device->light_info->light_type, light_type);
}
```

```
Device ID: 123
Device Name: Sample Device
Device Status: 1
Light Watts: 60
Light Type: Modern
```

2. How does encapsulation, achieved through securely bundling device-specific details within the "Device" class and utilising friend functions, enhance data security and abstraction in "HomeGenius"? How does this cohesive approach align with Abstract Data Type (ADT) principles, ensuring scalability and modularity? Discuss with examples.

10 **(CO3, BL2)**

Possible Solution:

Encapsulation and Friend Functions:

In the context of the "HomeGenius" project, encapsulation ensures that the internal details of the "Device" class are hidden from external entities, enhancing data security. Friend functions, when appropriately utilized, can access private members of the "Device" class, providing controlled access for designated entities.

Alignment with ADT Principles:

Encapsulation, along with friend functions, adheres to ADT principles by abstracting the implementation details of devices. This abstraction promotes modularity, allowing developers to interact with devices through well-defined interfaces without concerning themselves with internal complexities.

Scalability and Modularity:

Encapsulation and ADT principles facilitate scalability by allowing new device types to be seamlessly integrated into the system. Developers can extend the functionality of the

"Device" class by deriving specialized subclasses, ensuring that the system can adapt to evolving technological requirements.

Examples:
For instance, encapsulation ensures that sensitive device data, such as security camera footage, remains inaccessible to unauthorized entities. Friend functions, such as an administrator's access to device diagnostics, exemplify controlled access to encapsulated data. This approach aligns with ADT principles by abstracting device functionality, enabling developers to focus on high-level interactions while ensuring scalability and modularity.

3. Develop a C++ program featuring a class "Device" with private attributes including ID (integer), name (string), and status (boolean), alongside a static attribute tracking the total number of devices created. Implement dynamic memory allocation within the constructor and include a copy constructor to facilitate deep copying of objects. Employ a destructor for memory deallocation and integrate a static member function to retrieve the count of devices. Showcase the program's functionality by dynamically creating two device objects and displaying their details. Lastly, ensure proper memory management by deallocating memory after usage.

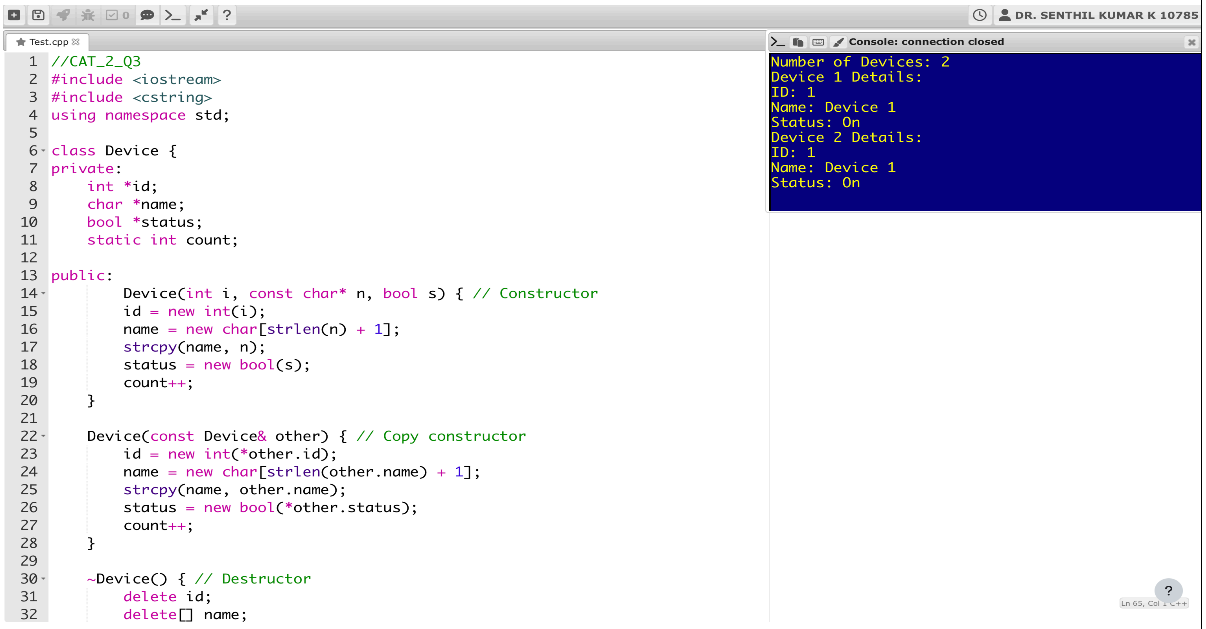
10 **(CO3, BL3)**

Possible Solution:

```
#include <iostream>
#include <cstring>
using namespace std;

class Device {
private:
   int *id;
   char *name;
   bool *status;
   static int count;

public:
      Device(int i, const char* n, bool s) { // Constructor
      id = new int(i);
      name = new char[strlen(n) + 1];
      strcpy(name, n);
      status = new bool(s);
      count++;
   }

   Device(const Device& other) { // Copy constructor
      id = new int(*other.id);
      name = new char[strlen(other.name) + 1];
      strcpy(name, other.name);
      status = new bool(*other.status);
      count++; }
   ~Device() { // Destructor
      delete id;
      delete[] name;
      delete status;
      count--;}
   static int getCount() { // Static member function to get count
      return count;}

  void display() {     // Display device details
```

```
        cout << "ID: " << *id << endl;
        cout << "Name: " << name << endl;
        cout << "Status: " << (*status ? "On" : "Off") << endl;}
};

int Device::count = 0; // Initializing static member variable

int main() {
    // Dynamically creating device objects
    Device* d1 = new Device(1, "Device 1", true);
    // Using copy constructor to create a new device object
    Device* d2 = new Device(*d1);
    // Displaying device details
    cout << "Number of Devices: " << Device::getCount() << endl;
    cout << "Device 1 Details:" << endl;
    d1->display();
    cout << "Device 2 Details:" << endl;
    d2->display();

    // Deallocating memory
    delete d1;
    delete d2;
    return 0;}
```

Sample Output:



```
//CAT_2_Q3
#include <iostream>
#include <cstring>
using namespace std;

class Device {
private:
    int *id;
    char *name;
    bool *status;
    static int count;

public:
        Device(int i, const char* n, bool s) { // Constructor
        id = new int(i);
        name = new char[strlen(n) + 1];
        strcpy(name, n);
        status = new bool(s);
        count++;
    }

    Device(const Device& other) { // Copy constructor
        id = new int(*other.id);
        name = new char[strlen(other.name) + 1];
        strcpy(name, other.name);
        status = new bool(*other.status);
        count++;
    }

    ~Device() { // Destructor
        delete id;
        delete[] name;
```

```
Number of Devices: 2
Device 1 Details:
ID: 1
Name: Device 1
Status: On
Device 2 Details:
ID: 1
Name: Device 1
Status: On
```

4. Explain the pivotal role of inheritance in "HomeGenius" smart home automation project. How does inheritance foster code extensibility and maintainability? Elaborate on how specialised device classes derived from the base "Device" class contribute to the project's scalability and adaptability, ensuring seamless integration of new devices to meet evolving technological and customer demands.

10 **(CO3 , BL2)**

Possible Solution:

Code Reusability:

By defining a base class "Device" with common attributes and behaviors shared among different types of devices (such as lights, thermostats, and security cameras), inheritance allows for the reuse of code across multiple device classes. For example, common functionalities like device initialization, status monitoring, and control mechanisms can be implemented in the base class. Subclasses can then inherit these functionalities, reducing the need for redundant code and promoting code reusability.

Scalability:
Inheritance supports scalability by providing a flexible framework for extending the functionality of the "HomeGenius" system. As new types of devices are introduced or existing ones are updated, new subclasses can be created to represent these devices. The hierarchical structure of classes allows for easy integration of new device types without the need for extensive modifications to the existing codebase. This enables the "HomeGenius" system to scale up to accommodate a growing range of devices and functionalities.

Adaptability:
Inheritance facilitates adaptability by allowing the "HomeGenius" system to adapt to changing requirements and technological advancements. The modular nature of inheritance enables developers to easily modify or extend the behavior of existing device classes to meet new requirements. For example, if new features or functionalities need to be added to specific device types, developers can simply create new subclasses or override existing methods in the subclass. This flexibility ensures that the "HomeGenius" system can evolve over time to address evolving customer needs and technological advancements while maintaining its overall architecture.

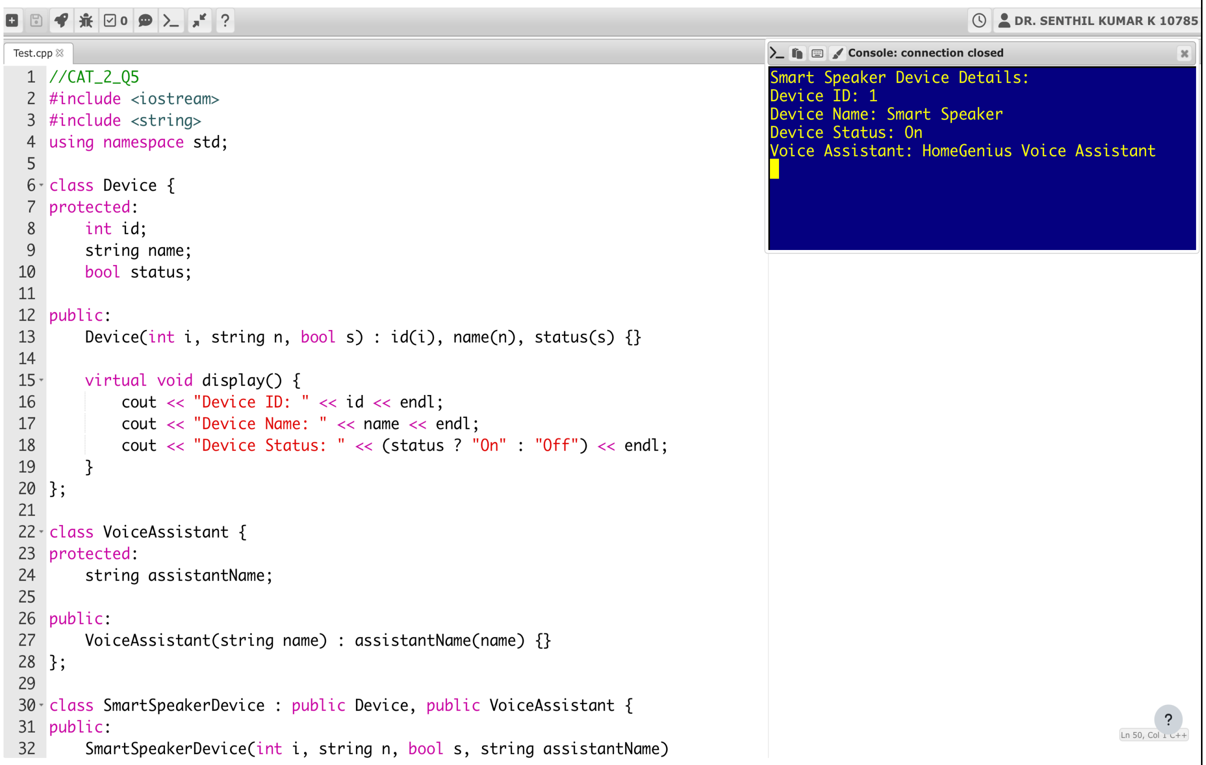
5. In the HomeGenius smart home system, there is a need to represent a special type of device called "SmartSpeakerDevice" that combines the functionalities of both a regular device and a voice-controlled assistant. Implement a class called "SmartSpeakerDevice" that inherits from both the "Device" class and the "VoiceAssistant" class (having a single string attribute named “voice-controlled assistant”). Write a C++ program that demonstrates the concept of multiple inheritance and test the "SmartSpeakerDevice" class by creating an object in the main() function. 10 **(CO3, BL3)**

Possible Solution:

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

class Device {
protected:
    int id;
    string name;
    bool status;

public:
    Device(int i, string n, bool s) : id(i), name(n), status(s) {}
    virtual void display() {
        cout << "Device ID: " << id << endl;
        cout << "Device Name: " << name << endl;
        cout << "Device Status: " << (status ? "On" : "Off") << endl;
    }
};
```

```
class VoiceAssistant {
protected:
    string assistantName;
public:
    VoiceAssistant(string name) : assistantName(name) {}
};

class SmartSpeakerDevice : public Device, public VoiceAssistant {
public:
    SmartSpeakerDevice(int i, string n, bool s, string assistantName)
        : Device(i, n, s), VoiceAssistant(assistantName) {}
    void display(){
        Device::display();
        cout << "Voice Assistant: " << assistantName << endl;
    }
};
int main() {

    SmartSpeakerDevice smartSpeaker(1, "Smart Speaker", true,   "HomeGenius Voice Assistant");

    cout << "Smart Speaker Device Details:" << endl;

    smartSpeaker.display();

    return 0;
}
```

Sample Output:



```
//CAT_2_Q5
#include <iostream>
#include <string>
using namespace std;

class Device {
protected:
    int id;
    string name;
    bool status;

public:
    Device(int i, string n, bool s) : id(i), name(n), status(s) {}

    virtual void display() {
        cout << "Device ID: " << id << endl;
        cout << "Device Name: " << name << endl;
        cout << "Device Status: " << (status ? "On" : "Off") << endl;
    }
};

class VoiceAssistant {
protected:
    string assistantName;

public:
    VoiceAssistant(string name) : assistantName(name) {}
};

class SmartSpeakerDevice : public Device, public VoiceAssistant {
public:
    SmartSpeakerDevice(int i, string n, bool s, string assistantName)
```

```
Smart Speaker Device Details:
Device ID: 1
Device Name: Smart Speaker
Device Status: On
Voice Assistant: HomeGenius Voice Assistant
```