1. Key Differences Between Programming in C vs. C++

Aspect	C	C++	Explanation & Example
Input/Output	printf / scanf	cout / cin	C uses printf and scanf for I/O with format specifiers, while C++ uses cout and cin with the << and >> operators, making syntax more readable. Example in C: printf("Enter a number: "); scanf("%d", &num); Example in C++: cout << "Enter a number: "; cin >> num;
Programming Paradigm	Procedural	Object-Oriented	C is primarily procedural, where functions are the core building blocks. C++ is object-oriented, enabling the creation of classes and objects, which helps model complex systems.
Memory Management	malloc / free	new / delete	C uses malloc and free for dynamic memory allocation, while C++ provides new and delete, allowing constructors/destructors to handle initialization/cleanup.
Error Handling	Return Codes	Exceptions	In C, errors are managed through return codes, while C++ has structured exception handling (try-catch blocks).
Namespace Management	No namespaces	namespace keyword	C++ uses namespaces to avoid name conflicts, enabling code modularity. C lacks this feature, which can lead to symbol conflicts.

2. Defining a Class

```
class Person {
  private:
    string name;
    int age;

  public:
    // Constructors
    Person(); // Default constructor
    Person(string n, int a); // Parameterized constructor
    Person(const Person &other); // Copy constructor

    // Destructor
    ~Person();

    // Getters and Setters
    string getName() const;
    void setName(string n);
    int getAge() const;
    void setAge(int a);
    // Additional Methods
    void displayInfo() const;
};
```

Explanation of Each Component:

- Data Members and Methods: Data members like name and age store information about the object, while methods perform operations on this data.
- Public vs. Private Access Modifiers: private members are only accessible within the class, while public members are accessible from outside.
- Setters & Getters: Setters modify private data members, and getters retrieve their values.
- Constructors and Destructors:
- Default Constructor: Initializes data members to default values.
- Parameterized Constructor: Initializes data members to specified values.
- Copy Constructor: Initializes an object as a copy of another object.
- Destructor: Cleans up resources when the object is destroyed.
- Additional Methods: displayInfo() provides functionality to output data members' values.

3. Example Linked List Class Implementation

```
class Node {
  public:
    int data;
    Node* next;

    Node(int value) : data(value), next(nullptr) {}
};

class LinkedList {
  private:
    Node* head;

  public:
    // Constructors
    LinkedList() : head(nullptr) {}
    LinkedList(const LinkedList &list); // Copy constructor
```

```
        // Destructor
        ~LinkedList();

        // Methods
        void insert(int value);
        void display() const;

        // Additional Method
        bool search(int value) const;
};

// Method implementations
void LinkedList::insert(int value) {
    Node* newNode = new Node(value);
    if (!head) {
        head = newNode;
    } else {
        Node* temp = head;
        while (temp->next) temp = temp->next;
        temp->next = newNode;
    }
}

void LinkedList::display() const {
    Node* temp = head;
    while (temp) {
        cout << temp->data << " -> ";
        temp = temp->next;
    }
    cout << "NULL" << endl;
}

bool LinkedList::search(int value) const {
    Node* temp = head;
    while (temp) {
        if (temp->data == value) return true;
        temp = temp->next;
    }
    return false;
```

```
}
```

- Data Members: Node* head points to the first node in the list.
- Constructors: Initializes the list, optionally with a copy constructor.
- Destructor: Deletes nodes to prevent memory leaks.
- Methods:
- insert: Adds a new node to the end of the list.
- display: Outputs the entire list.
- search: Checks if a value exists in the list.

4. Declaring Objects and Using Them

After defining classes, you can declare objects and use them to perform meaningful tasks.

Example Usage of LinkedList

```
int main() {
  LinkedList list;
  list.insert(10);
  list.insert(20);
  list.insert(30);
  list.display(); // Output: 10 -> 20 -> 30 -> NULL

  if (list.search(20)) {
    cout << "20 is in the list." << endl;
  } else {
    cout << "20 is not in the list." << endl;
  }

  return 0;
}
```