Q1: Dynamic Array of Objects (20-25 minutes)

Problem Statement:

In many real-world applications, we often need to manage multiple records dynamically. Instead of using a fixed-size array, a **dynamic array of objects** allows flexibility in handling data.

Task:

1. Create a class **`Student`** with the following **private** data members:
   - `string name`
   - `int rollNumber`
   - `float CGPA`
2. Implement the following **member functions**:
   - A **parameterized constructor** to initialize the student's data.
   - A **display function** to show student details.
3. In `main()`, **dynamically allocate an array of 3-4 `Student` objects** using pointers.
4. Ask the user to input details for each student and store them in the dynamic array.
5. Print the details of all students using the display function.
6. **Free the allocated memory** to avoid memory leaks.

Expected Output Example:

```
yaml
CopyEdit
Enter details for student 1:
Name: Alice
Roll Number: 101
CGPA: 3.75

Enter details for student 2:
Name: Bob
Roll Number: 102
CGPA: 3.90

Student Records:
Name: Alice | Roll No: 101 | CGPA: 3.75
Name: Bob | Roll No: 102 | CGPA: 3.90
```

Q2: Deep Copy & Dynamic Memory (20-25 minutes)

Problem Statement:

When working with dynamically allocated memory in a class, the default copy constructor performs a **shallow copy**, which can lead to memory-related issues. To avoid this, we implement a **deep copy constructor**.

Task:

1. Create a class `Book` with the following **private** data members:
   - `string title` (stores the book title)
   - `char* content` (dynamically allocated character array for book content)
2. Implement the following functions:
   - **Parameterized constructor** to allocate memory and initialize data.
   - **Deep copy constructor** to properly copy dynamically allocated memory.
   - **Destructor** to free allocated memory.
   - **Display function** to show book details.
3. In `main()`, create an object of `Book`, then use the **copy constructor** to initialize another object.
4. Modify one book's content and ensure that the other object remains unaffected.

Expected Output Example:

```
yaml
CopyEdit
Original Book Title: C++ Programming
Original Book Content: Object-Oriented Concepts

Copying book using deep copy...

Modified Original Book Content: Advanced OOP
Copied Book Content Remains: Object-Oriented Concepts
```

Q3: Using `this` Pointer & Constant Functions (20 minutes)

Problem Statement:

The **`this` pointer** is used inside a class to refer to the current object. Constant functions ensure that a method does not modify any class member.

Task:

1. Create a class `BankAccount` with the following **private** data members:
   - `string accountHolder`
   - `double balance`
2. Implement the following functions:
   - A **parameterized constructor** that initializes the attributes using the **`this` pointer**.
   - A **constant getter function** `getBalance()` to return the balance.
   - A **deposit function** that adds an amount to the balance.
   - A **withdraw function** that subtracts an amount (if sufficient balance).
3. Demonstrate the use of the **`this` pointer** in the constructor and member functions.
4. In `main()`, create an object, deposit money, and display the updated balance.

Expected Output Example:

```
nginx
CopyEdit
Account Holder: John Doe
Initial Balance: $1000

Depositing $500...
New Balance: $1500

Withdrawing $200...
Remaining Balance: $1300
```

Q4: Static Data Members & Static Functions (20-25 minutes)

Problem Statement:

Sometimes, we need a variable that is **shared among all objects of a class** rather than being unique to each object. This is achieved using **static data members**. **Static functions** can access only static data members.

Task:

1. Create a class **`Employee`** with the following **private** data members:
   - `string name` (stores employee name)
   - `int ID` (stores employee ID)
   - A **static data member** `employeeCount` (keeps track of the total number of employees).
2. Implement the following functions:
   - A **constructor** that initializes name and ID and increments `employeeCount`.
   - A **static function** `getEmployeeCount()` that returns the number of employees created.
   - A **display function** to show employee details.
3. In `main()`, create multiple `Employee` objects and observe how the `employeeCount` variable updates.

Expected Output Example:

```
yaml
CopyEdit
Employee 1: Alice (ID: 101)
Employee 2: Bob (ID: 102)

Total Employees: 2

Adding one more employee...
Employee 3: Charlie (ID: 103)

Total Employees: 3
```

Submission Guidelines:

1. Write **well-structured** and **properly commented** code.
2. Ensure correct **memory allocation & deallocation** where applicable.
3. Test all functionalities and check expected outputs.
4. Submit your solutions before the deadline.