Meesho Screening Test

Four Pillars of Object-Oriented Programming (OOP): In the context of programming, the "Four Pillars of Object-Oriented Programming" refer to the core principles of OOP. These principles are:

- **Encapsulation:** The bundling of data (attributes) and the methods (functions) that operate on the data into a single unit, known as a class. It restricts access to some of an object's components and prevents the accidental modification of data.
- **Inheritance:** A mechanism that allows a new class (subclass/derived class) to inherit properties and behaviors from an existing class (superclass/base class). This promotes code reuse and establishes a relationship between classes.
- **Abstraction:** The process of simplifying complex systems by modeling classes based on the essential features and ignoring irrelevant details. Abstraction helps in managing the complexity of software systems.
    - **Polymorphism:** Refers to the ability of a single function or method to operate on different types of data or, in the case of OOP, the ability of different classes to be treated as instances of the same class through a common interface. This promotes flexibility and adaptability in code.

Stored Procedure

A stored procedure is a precompiled collection of one or more SQL statements or procedural logic that is stored in the database and can be executed by a database management system. It allows developers to group SQL statements into a single unit, providing a modular and reusable way to interact with the database.

Modularization and Code Reusability:
Improved Performance:
Security:
Transaction Management:
Parameterized Queries:
Encapsulation of Business Logic:
Reduced Network Traffic:
Ease of Maintenance:

Overall, stored procedures provide a structured and efficient way to manage database interactions, enhance security, and promote code organization and reuse in a database-driven application

Dynamic Linking

Dynamic linking is a mechanism employed by operating systems to link executable code with shared libraries or dynamic-link libraries (DLLs) during the runtime of a program. Unlike static linking, where the linking is done before the program is run, dynamic linking happens at runtime. Here are the key concepts associated with dynamic linking in operating systems:

Shared Libraries:
Dynamic Linking Process:
Advantages:

- Memory Efficiency:
- Flexibility and Updates:
- smaller executable files.
- memory efficient.
- limits application executable file size
- dynamic loading
  Reduced Disk Space:

How you can achieve data hiding in C++

Data hiding in C++ is achieved through the use of access specifiers and encapsulation. Encapsulation is one of the fundamental principles of object-oriented programming that allows bundling data and methods that operate on the data into a single unit, known as a class. Access specifiers control the visibility of class members (data and functions) from outside the class. The primary access specifiers in C++ are public, private, and protected.

Here's how you achieve data hiding in C++:

1. *Private Access Specifier:*

   - Declare the data members as private within the class. This restricts direct access to these members from outside the class.

   cpp
   class MyClass {
   private:
   int privateData;

   public:
   void setData(int value) {

privateData = value;
}

```
int getData() {
    return privateData;
}
```

};

2. *Public Member Functions:*

    - Provide public member functions (getters and setters) to manipulate the private data. These functions act as interfaces for accessing and modifying the data.

cpp
int main() {
MyClass obj;
obj.setData(42);
std::cout << "Data: " << obj.getData() << std::endl;
return 0;
}

3. *Encapsulation:*
    - Encapsulation involves bundling the data (private members) and the functions (public members) that operate on the data into a single unit (class).
    - Users of the class interact with it through its public interface, unaware of the internal implementation details.

By employing private access specifiers and encapsulation, you ensure that the internal details of the class are hidden from external code. This promotes information hiding, reduces dependencies, and allows for better maintenance and modification of the class without affecting other parts of the program.

Describe the role of cache memory in improving operating system performance.

Cache memory plays a crucial role in enhancing operating system (OS) performance by providing a faster access layer between the CPU and the main memory (RAM). Here's how it contributes:

1. *Faster Access to Frequently Used Data:*

- Cache memory stores frequently accessed data and instructions.
- The CPU can quickly retrieve information from the cache rather than fetching it from the slower main memory, reducing latency.

2. *Reducing Memory Access Time:*
   - Accessing data from cache is faster than accessing it from RAM.
   - By keeping frequently used data in cache, the OS minimizes the time the CPU spends waiting for data to be fetched from the slower RAM.
3. *Enhancing CPU Utilization:*
   - With faster access times, the CPU spends less time idling, waiting for data to be retrieved from the main memory.
   - This improves overall CPU utilization, making the system more efficient.
4. *Minimizing Memory Bandwidth Usage:*
   - Cache helps reduce the demand on the system's memory bandwidth.
   - Frequently used data is readily available in the cache, reducing the need to access the main memory and saving bandwidth for other operations.
5. *Improving Overall System Responsiveness:*
   - Faster access to data and instructions results in quicker execution of programs and tasks.
   - This leads to an overall improvement in system responsiveness and user experience.
6. *Mitigating the Impact of Memory Latency:*
   - Cache helps mitigate the impact of memory latency, which is the time it takes to retrieve data from RAM.
   - By providing a faster intermediate storage, cache minimizes the performance bottleneck caused by slower main memory.

In summary, cache memory acts as a high-speed buffer between the CPU and main memory, optimizing data retrieval and improving the overall performance of the operating system by reducing latency, enhancing CPU utilization, and ensuring faster access to frequently used data.

How memory management is different in C and C++?

Memory management in C++ and C shares some similarities due to C++ being an extension of C, but there are key differences, primarily related to features introduced in C++ for better memory handling.

1. Dynamic Memory Allocation:

- In both C and C++, you can use malloc, calloc, and free for dynamic memory allocation and deallocation.
- C++ introduces the new and delete operators, providing a more convenient and type-safe way to allocate and deallocate memory. Additionally, C++ introduces new[] and delete[] for array allocation and deallocation.

2. Constructors and Destructors:

- C++ introduces constructors and destructors as part of classes. Constructors are called when an object is created, and destructors are called when an object goes out of scope or is explicitly deleted.
- This allows for more controlled and automated memory management in C++ compared to C.

3. Operator Overloading:

- C++ supports operator overloading, enabling customization of memory management through overloaded new and delete operators.
- This feature allows developers to implement custom memory allocation strategies for specific classes.

4. Smart Pointers:

- C++ introduces smart pointers (e.g., std::shared_ptr, std::unique_ptr), which are objects that manage the memory they point to. They automatically release memory when it is no longer needed.
- Smart pointers help prevent memory leaks and make memory management more robust compared to raw pointers in C.

5. Memory Safety:

- C++ provides features like RAII (Resource Acquisition Is Initialization) through constructors and destructors, improving memory safety compared to C.
- C++ also introduces features like references, which can enhance safety by avoiding some common pointer-related errors.

6. Memory Leaks:

- Both C and C++ can suffer from memory leaks if not managed properly. However, with features like smart pointers, C++ provides better tools for managing resources and reducing the likelihood of memory leaks.

In summary, while C++ retains many memory management features from C, it introduces additional features like constructors, destructors, operator overloading, and smart pointers to provide a more convenient, type-safe, and robust approach to memory management. These features aim to enhance code readability, maintainability, and safety in comparison to C.

ACID is an acronym that stands for Atomicity, Consistency, Isolation, and Durability. These properties are fundamental principles in database management systems (DBMS) to ensure the reliability and integrity of transactions. Here's a brief explanation of each of the ACID properties:

1. **Atomicity:**
    - **Definition:** Atomicity ensures that a transaction is treated as a single, indivisible unit of work. Either all the changes made by the transaction are committed to the database, or none of them are.
    - **Example:** If a transaction involves multiple SQL statements, and any one of them fails, the entire transaction is rolled back to its original state, and no changes are applied to the database.
2. **Consistency:**
    - **Definition:** Consistency ensures that a transaction brings the database from one consistent state to another. It enforces integrity constraints, preserving the overall correctness and validity of the data.
    - **Example:** If a transaction violates any integrity constraint (such as a primary key or foreign key constraint), it is rolled back, and the database remains in a consistent state.
3. **Isolation:**
    - **Definition:** Isolation ensures that the execution of transactions is independent of each other. Even if multiple transactions are executed concurrently, the result should be the same as if they were executed in some sequential order.
    - **Example:** Transactions are isolated from each other to prevent interference. Changes made by one transaction are not visible to other transactions until the first transaction is committed.
4. **Durability:**
    - **Definition:** Durability guarantees that once a transaction is committed, its changes are permanent and survive any subsequent failures (such as a power outage or system crash). The changes are stored in non-volatile storage (e.g., disk) and can be recovered in case of a system failure.
    - **Example:** Once a user receives a confirmation that their transaction is committed, they can be assured that the changes made by the

transaction will persist, even in the event of a system failure.

In summary, ACID properties provide a framework for designing and managing transactions in a way that ensures the reliability, consistency, and durability of a database. These properties are crucial in scenarios where data integrity is of utmost importance, such as in financial systems, e-commerce platforms, and other applications where accurate and reliable data handling is essential.