Вариант 8

Нулевая группа вопросов:

1) Для решения лабораторной работы по созданию потоков в ОС Windows с использованием WinAPI потребуется следующие функции и структуры:

CreateThread: Создание нового потока.
ExitThread: Завершение текущего потока.
WaitForSingleObject или WaitForMultipleObjects: Ожидание завершения потока(ов).
CloseHandle: Закрытие дескриптора потока.
HANDLE: Дескриптор потока для управления.

2) Процесс в ОС Windows - это выполняющаяся программа или приложение, которая имеет собственное адресное пространство памяти, собственный набор ресурсов и независимую область выполнения кода. Процессы обеспечивают изоляцию и многозадачность в операционной системе, позволяя нескольким приложениям работать параллельно без вмешательства друг в друга.

3) Критическая секция - это участок кода в программе, который должен быть выполнен только одним потоком одновременно. Она используется для синхронизации доступа нескольких потоков к общим ресурсам или данным, чтобы избежать конфликтов и обеспечить корректное выполнение программы. Критическая секция может быть защищена мьютексом или семафором.

4) Семафор - это средство синхронизации, которое позволяет управлять доступом нескольких потоков к общим ресурсам или критическим секциям. Семафор может использоваться для ограничения количества потоков, имеющих доступ к определенным ресурсам.

5) Сравнительный анализ стандарта C++98 с и без применения библиотеки Boost:

C++98 является старым стандартом C++, который не включает в себя многие современные функции и возможности, такие как многопоточность, сетевое программирование.

Boost - это сторонняя библиотека для C++, предоставляющая множество расширенных функций и инструментов, включая поддержку многопоточности, парсинг XML, работу с сетью.

Вторая группа вопросов:

1. Процедурная декомпозиция (или декомпозиция программы) - это методология разработки программного обеспечения, при которой программа разделяется на более мелкие подпрограммы или процедуры. Эти подпрограммы выполняют конкретные задачи и могут вызываться из других частей программы. Процедурная декомпозиция способствует улучшению структуры программы, повышению ее читаемости и обслуживаемости, а также уменьшению дублирования кода.

2. Динамический полиморфизм - это один из видов полиморфизма в объектно-ориентированном программировании. Он позволяет объектам разных классов реагировать на вызовы одних и тех же методов (или функций) специфичным для своего класса способом. В C++ и многих других языках программирования, динамический полиморфизм достигается с использованием виртуальных функций и механизма наследования. Это позволяет вызывать методы на основе типа объекта во время выполнения программы, а не на этапе компиляции.

3. Инкапсуляция - это один из основных принципов объектно-ориентированного программирования (ООП), который представляет собой механизм сокрытия внутренних деталей реализации класса и предоставления интерфейса для взаимодействия с этим классом. Инкапсуляция позволяет скрыть данные и методы, которые не должны быть доступными извне класса, и обеспечивает контролируемый доступ к ним через публичные

методы и свойства. Это упрощает управление и поддержку кода, а также обеспечивает безопасность и изоляцию данных и функциональности класса.

Третья группа вопросов:

1) Creation Design Pattern - Factory Method:

Описание через призму инкапсуляции:
Фабричный метод является порождающим паттерном проектирования, который обеспечивает инкапсуляцию процесса создания объектов. Он предоставляет интерфейс для создания объектов, но оставляет конкретную реализацию создания подклассам. Это означает, что клиентский код не знает о конкретных классах объектов, которые он создает, и следует интерфейсу фабричного метода.

Пример использования на практике:
Рассмотрим пример использования фабричного метода для создания различных типов транспортных средств (например, автомобилей). У нас есть абстрактный класс Transport и два конкретных подкласса: Car и Bicycle. Теперь мы создадим интерфейс TransportFactory, который будет определять фабричный метод createTransport(), а каждый подкласс этого интерфейса будет предоставлять свою реализацию метода createTransport(). Это обеспечит инкапсуляцию процесса создания транспортных средств.

2) Visitor Design Pattern:

Описание через призму инкапсуляции:
Паттерн "Посетитель" (Visitor) позволяет инкапсулировать операции над объектами вне их структуры. Этот паттерн представляет собой способ добавления новых операций без изменения классов объектов. Посетитель определяет новую операцию как визит к объектам различных классов, и каждый конкретный класс объекта должен предоставить метод, который позволит посетителю выполнить свою операцию.

Пример использования на практике:
Рассмотрим паттерн "Посетитель" на примере обработки структуры документа, состоящего из различных элементов, таких как параграфы, таблицы и изображения. Мы можем создать интерфейс "Посетитель" с методами для каждого типа элемента и конкретные реализации этого интерфейса для выполнения конкретных операций (например, подсчета слов в параграфе или определения размера изображения). Каждый элемент документа должен иметь метод accept(), который позволяет посетителю выполнить свою операцию. В этом примере, "Посетитель" инкапсулирует операции, выполняемые над элементами документа, и позволяет добавлять новые операции без изменения самих элементов документа.