1. В каких ситуациях в современном C++ рекомендуется использовать структуры?

Структуры полезны, когда нам надо объединить несколько переменных под одним именем. Их удобно использовать, чтобы сгруппировать данные. По умолчанию все члены структуры публичные, поэтому если какие-то данные нужно скрыть, следует использовать классы.

2. Какие аспекты следует учитывать при проектировании и разработке классов?

При проектировании классов нужно соответствовать принципу инкапсуляции - когда данные или то, как устроены и работают методы и классы, помещают в виртуальную капсулу, чтобы их нельзя было повредить извне. Для пользователя должен быть разработан интерфейс для работы с классом, чтобы пользователь не мог изменять какие-либо данные, которые изменять нельзя, предупреждает неправильное использование данных и методов. Каждый класс должен отвечать только за одну функциональность, и она должна быть полностью инкапсулирована в класс.
При написании класса также нужно уделить внимание конструктору, чтобы, к примеру, при инициализации объекта без указания значений полей они не заполнялись мусорными значениями.

3. Почему практически никогда не следует явным образом вызывать деструктор?

В C++ десктруктор вызывается автоматически, когда мы выходим из зоны видимости объекта. Вызов деструктора явным образом приведет к неопределенному поведению, так как он снова будет вызван после }, при выходе из блока, в котором был объект.
Если объект создан динамически при помощи *new*, вызывать для него деструктор тоже не стоит. При использовании *delete* он вызовется автоматически.
Даже если вы хотите, чтобы объект уничтожился перед окончанием блока, деструктор вызывать ни в коем случае нельзя. Лучше просто свернуть весь код, который должен выполняться перед уничтожением объекта, в отдельный блок. Тогда при окончании этого блока объект будет разрушен.

4. В каких ситуациях можно использовать статические данные и функции классов?

Для статических элементов выделение памяти происходит только один раз и существуют эти элементы до завершения программы. Хранятся они не в heap и не на stack, а в специальных сегментах памяти, которые называются .data и .bss (зависит от того инициализированы статические данные или нет).
Статические данные класса относятся ко всем объектам класса. Независимо от количества объектов данного класса, существует только одна копия статического элемента. Статические данные класса можно использовать, когда есть какие-то данные, которые должны

быть общими для всех элементов класса. Например, если нам нужно хранить количество объектов класса, счетчик, который должен быть общим для всех объектов класса или какие-либо общие характеристики объектов.
Статическую функция-член может использоваться без создания объекта класса. Они имеют область видимости класса, в котором они находятся. Их можно использовать, если нужно выполнить операцию, которая не зависит от состояния объекта и не требует доступа к его данным. Также статические функции-члены используются, когда требуется общая функциональность, которая может быть использована всеми экземплярами класса. Например, функция для проверки валидности данных или генерации уникального идентификатора.

5. Почему использование друзей стандартными способами ухудшает инкапсуляцию?

Когда мы делаем класс B другом класса A, все приватные данные и методы класса A становятся доступны классу B. Но часто нам нужен доступ не ко всем приватным данным, а лишь к некоторым. Таким образом, используя дружбу, мы нарушаем инкапсуляцию класса A. В таких случаях лучше использовать идиому Attorney-Client или паттерн Passkey.
Тем не менее, если функция друга используется правильно, она лишь улучшит инкапсуляцию. Есть хорошее эмпирическое правило - "no long distance friends". Это значит, что дружбу стоит использовать только для внутренних и вложенных классов.