1. **в чем отличие динамического массива от статического массива в языке c++?**

Статический массив создается на стеке с фиксированным размером во время компиляции, размер должен быть известен заранее и его нельзя изменить. динамический массив выделяется в куче через new или malloc во время выполнения программы, можно задать размер на основе переменной и потом освободить память когда она не нужна. статический массив автоматически удаляется когда выходим из области видимости, динамический нужно чистить вручную через delete.

2. **что такое указатель и зачем он используется при работе с динамической памятью?**

Указатель это переменная которая хранит адрес другой переменной в памяти. при работе с динамической памятью он нужен потому что операторы new и malloc возвращают адрес выделенного блока памяти, а не сами данные. через указатель мы обращаемся к этой памяти, читаем и записываем данные, и потом используем тот же указатель чтобы освободить память через delete.

3. **почему важно корректно освобождать память после использования динамических массивов?**

Если не освобождать память получается утечка памяти. программа продолжает занимать оперативку даже когда данные уже не нужны, и эта память становится недоступна другим программам. при длительной работе или в цикле это может привести к тому что вся доступная память закончится и программа или система упадут. в современных ос память освобождается после завершения программы, но это не повод писать плохой код.

4. **в чем разница между последовательной и параллельной обработкой массива?**

Последовательная обработка это когда элементы массива обрабатываются один за другим в одном потоке выполнения. параллельная обработка разбивает массив на части и обрабатывает их одновременно в нескольких потоках на разных ядрах процессора. последовательная проще и предсказуемее, параллельная быстрее на больших данных но сложнее в реализации из-за необходимости синхронизации потоков.

5. **что делает директива #pragma omp parallel for?**

Эта директива говорит компилятору распараллелить следующий за ней цикл for между доступными потоками. openmp автоматически разбивает итерации цикла на части, создает нужное количество потоков, распределяет работу между ними и ждет

пока все потоки закончат перед продолжением программы. это самый простой способ распараллелить независимые вычисления в цикле.

6. **для чего используется механизм reduction в openmp?**

Reduction нужен когда в параллельном цикле надо накапливать результат в одной переменной, например считать сумму, находить минимум или максимум. без reduction получится гонка данных когда несколько потоков пытаются одновременно изменить одну переменную. reduction создает локальную копию переменной для каждого потока, потом автоматически комбинирует все копии в финальный результат используя указанную операцию.

7. **почему при параллельном вычислении суммы необходимо использовать reduction, а не обычную переменную?**

Если использовать обычную переменную то все потоки будут одновременно читать ее значение, прибавлять свою часть и записывать обратно. из-за того что эти операции не атомарны возникает race condition и результаты перезаписывают друг друга, итоговая сумма получается неправильной. reduction решает это создавая отдельную копию для каждого потока, каждый поток работает со своей копией без конфликтов, а потом openmp безопасно складывает все копии в финальный результат.

8. **какие факторы могут привести к тому что параллельная версия программы будет работать медленнее последовательной?**

Слишком маленький объем данных когда накладные расходы на создание потоков и их синхронизацию больше чем выигрыш от параллелизма. слишком много синхронизации или работы в критических секциях где потоки ждут друг друга. неравномерное распределение нагрузки когда одни потоки заканчивают быстро а другие долго работают. false sharing когда потоки работают с данными в одной линии кэша процессора и постоянно инвалидируют кэш друг другу. плохая локальность данных когда каждый поток обращается к разным участкам памяти и кэш не помогает.