**1. Основные понятия**

**1. Что такое отладчик и для чего он используется?**

Отладчик — это инструмент для пошагового выполнения программы, анализа её состояния и поиска ошибок.  
**Основные функции:**

* Пошаговое выполнение кода
* Просмотр значений переменных
* Остановка в заданных точках (breakpoints)
* Анализ вызовов функций

**2. В чем разница между отладчиком и дизассемблером?**

| **Отладчик** | **Дизассемблер** |
| --- | --- |
| Работает с исполняемым кодом в реальном времени | Преобразует бинарный код в ассемблерный/псевдокод |
| Позволяет изменять состояние программы | Только статический анализ (без запуска) |
| Используется для поиска логических ошибок | Применяется для реверс-инжиниринга |

**3. Какие типы ошибок можно найти с помощью отладчика?**

* Логические ошибки (неверные условия, бесконечные циклы)
* Ошибки памяти (выход за границы массива, утечки)
* Исключения (деление на ноль, NullReference)
* Неправильные значения переменных

**4. Что такое точка останова (breakpoint)?**

Точка останова — это метка в коде, при достижении которой программа приостанавливает выполнение.  
**Виды breakpoints:**

* Обычные (остановка на строке)
* Условные (срабатывают при выполнении условия)
* Логические (активируются при определённом состоянии программы)

**2. Работа с отладчиком**

**5. Как запустить программу в режиме отладки в Visual Studio?**

1. Откройте проект.
2. Нажмите **F5** (или **Debug → Start Debugging**).
3. Для пошагового выполнения используйте **F10** (Step Over) или **F11** (Step Into).

**6. Основные команды отладчика**

| **Команда** | **Горячая клавиша** | **Описание** |
| --- | --- | --- |
| **Continue** | F5 | Продолжить выполнение до следующего breakpoint |
| **Step Over** | F10 | Выполнить текущую строку без входа в функции |
| **Step Into** | F11 | Войти в вызываемую функцию |
| **Step Out** | Shift+F11 | Выйти из текущей функции |

**7. Как просмотреть значения переменных во время отладки?**

* **Locals** — автоматически отображает локальные переменные.
* **Watch** — позволяет добавить переменные для отслеживания.
* **Hover** — наведите курсор на переменную в коде.

**8. Что такое Watch-окно? Как его использовать?**

**Watch** — панель для отслеживания значений переменных и выражений.  
**Как добавить переменную:**

1. В меню отладчика выберите **Debug → Windows → Watch**.
2. Введите имя переменной или выражение (например, array[0]).

**9. Как отладить программу, если она падает без сообщения об ошибке?**

1. Включите **все исключения** (Debug → Windows → Exception Settings).
2. Используйте **логирование** (вывод в консоль или файл).
3. Анализируйте **дамп памяти** (если программа крашится).

**3. Дизассемблирование и анализ кода**

**10. Что такое дизассемблер?**

Дизассемблер преобразует машинный код в ассемблерный или псевдокод (например, Ghidra декомпилирует в C-подобный код).

**11. Популярные дизассемблеры**

* **Ghidra** (бесплатный, от NSA)
* **IDA Pro** (продвинутый, платный)
* **radare2** (консольный, для Linux)

**12. Как найти функцию main в дизассемблированной программе?**

* В **C/C++** ищите main или \_start.
* В **Ghidra** анализируйте граф вызовов.
* В **IDA Pro** используйте поиск по имени (main).

**13. Как найти проверку пароля в дизассемблированной программе?**

1. Ищите строки типа "Enter password".
2. Анализируйте вызовы функций сравнения (strcmp, memcmp).
3. Проверьте условные переходы (jz, jnz) после сравнения.

**14. Как определить, какие библиотеки использует программа?**

* **Windows:** Process Explorer, Dependency Walker.
* **Linux:** ldd, objdump -p.

**4. Практическое применение**

**15. Как найти ошибку "деление на ноль"?**

1. Установите breakpoint перед делением.
2. Проверьте значение делителя (F10 + **Watch**).
3. Если делитель 0, добавьте проверку (if (divisor != 0)).

**5. Обратная разработка (Reverse Engineering)**

**16. Как восстановить алгоритм без исходного кода?**

1. Дизассемблируйте программу (Ghidra/IDA).
2. Анализируйте ключевые функции (ввод, обработка, вывод).
3. Восстанавливайте логику по вызовам API и условиям.

**17. Как найти ключевые функции?**

* Ищите вызовы важных API (ReadFile, CreateProcess).
* Анализируйте строки (пароли, URL, ключи).
* Смотрите на граф вызовов в IDA/Ghidra.

**18. Как определить, использует ли программа шифрование?**

* Ищите функции типа AES\_encrypt, RSA\_generate\_key.
* Анализируйте вызовы криптобиблиотек (OpenSSL, CryptoAPI).

**19. Как анализировать вредоносное ПО?**

1. Запустите в песочнице (Cuckoo Sandbox).
2. Дизассемблируйте (Ghidra/IDA).
3. Ищите:
   * Сетевые подключения (socket, connect).
   * Инъекции кода (VirtualAlloc, CreateRemoteThread).

**6. Дополнительные вопросы**

**20. Разница между статическим и динамическим анализом?**

| **Статический анализ** | **Динамический анализ** |
| --- | --- |
| Без запуска программы | С запуском программы |

* Примеры: дизассемблирование, декомпиляция | Примеры: отладка, трассировка |

**21. Этические аспекты работы с дизассемблерами и отладчиками**

* **Можно:** Анализ собственного ПО, исследование безопасности (с разрешения).
* **Нельзя:** Взлом чужих программ, обход лицензий.
* **Важно:** Соблюдать законы (DMCA, GDPR) и лицензии.

**Итог:**

* Отладчик — для поиска ошибок в своём коде.
* Дизассемблер — для реверс-инжиниринга.
* Этичность важна: используйте инструменты только в законных целях.