Вопросы :

1. **Отладка – процесс запуска и выполнения программы в режиме отладки. Назначение отладчика – устранение ошибок в коде программы**.
2. **Запуск отладчика, прекращение отладки, установка точки останова и запуск отладчика, пошаговая отладка, проход по коду в отладчике с помощью пошаговых команд, быстрый перезапуск приложения, проверка переменных и изменение их значений с помощью подсказок по данным**.
3. **Точка останова (breakpoint) – это точка, в которой процесс выполнения программы приостанавливается и отладчик получает управление.**
4. **Установить точки останова можно следующими способами:**

**- Поместить курсор редактирования на выбранную строку программы и нажать клавишу команды Toggle Breakpoint (по умолчанию это клавиша <F5>) для установки или удаления точки останова в этой строке. То же самое можно выполнить и с помощью контекстного меню.**

**-F9**

**-Ctrl+B**

1. **Способы запуска отладчика в Visual Studio для C++:**

**- пункт главного меню Отладка >> Начать отладку(или Шаг с Обходом, или Шаг с Заходом);**

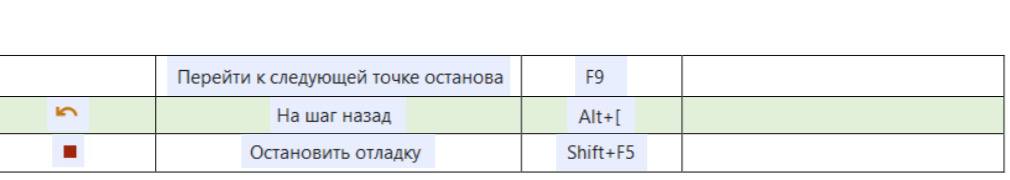
**- горячая клавиша F5;**

**- горячая клавиша F10 (запуск в пошаговом режиме);**

**- иконка на панели инструментов;**

**­­- через обозреватель решений.**

1. **Некоторые возможности управления режимом отладки**:



1. **Способы для отображения значений переменных:**

1**. Отладочная панель**: можно отображать значения переменных во время выполнения программы, устанавливая точки останова и просматривая значения переменных в отладочной панели.

2. **Окно «Локальные»**: позволяет просматривать значения локальных переменных в текущем контексте выполнения.

3. **Окно «Видимые»**: позволяет отслеживать значения определенных переменных во время выполнения программы.

4. **Окно "Immediate"**: позволяет выполнять код на лету и просматривать значения переменных в текущем контексте выполнения.

5. Использование **функции "Console.WriteLine**" для вывода значений переменных в консольное окно.

1. **Отладочные окна в среде разработки или отладчике (Debugger) предоставляют разработчикам информацию о состоянии программы во время её выполнения, что помогает в процессе отладки и исправления ошибок. Эти окна предназначены для следующих целей**:

**Просмотр значений перемен**ных: Отладочные окна позволяют отслеживать значения переменных в определенные моменты времени в процессе выполнения программы. Разработчик может посмотреть текущие значения переменных, что полезно для выявления ошибок или неправильного поведения программы.

**Стек вызовов (Call Stack**): Это окно отображает текущее состояние стека вызовов функций в программе. Оно показывает последовательность функций, вызванных до текущего момента, что помогает понять, какие функции вызываются перед возникновением ошибки.

**Процесс выполнения (Execution Process**): Отладочные окна могут отображать последовательность операций и команд, выполняемых программой. Это позволяет разработчику следить за шагами, которые программа выполняет в определенный момент времени.

**Управление точками останова (Breakpoints**): Отладочные окна позволяют устанавливать точки останова в коде. Когда выполнение программы достигает точки останова, выполнение приостанавливается, что позволяет разработчику изучить состояние программы на этом этапе.

**Анализ памяти (Memory Inspection**): Некоторые отладчики предоставляют возможность просмотра содержимого памяти в определенные моменты времени, что полезно для отслеживания изменений данных или поиска проблем с выделением памяти.

**Вывод сообщений и журнал событий (Output and Event Log)**: Окна вывода сообщений позволяют отображать текстовые сообщения, выдаваемые программой или отладчиком, а также записи о событиях, произошедших во время отладки.

**В целом, отладочные окна предоставляют разработчикам ценные инструменты для анализа и отслеживания состояния программы во время выполнения, что помогает обнаруживать и устранять ошибки**.

1. **Окно «Памяти» позволяет просматривать содержимое ячеек памяти. Содержимое памяти может отображаться в различных форматах, которые выбираются из контекстного меню**
2. **Окно регистров в отладчике предоставляет информацию о текущем состоянии регистров процессора во время выполнения программы. Оно используется для нескольких целей:**

**Отслеживание значений регистров**: Позволяет разработчику просматривать текущие значения регистров процессора в определенный момент времени выполнения программы. Это особенно важно при отладке, поскольку помогает понять, какие данные хранятся в регистрах в определенные моменты выполнения кода.

**Отладка и исправление ошибок**: При возникновении ошибок в программе разработчики могут использовать окно регистров для выявления некорректных значений, которые могут приводить к неправильному выполнению программы.

**Мониторинг состояния программы**: Просмотр регистров может помочь в отслеживании состояния программы на низком уровне. Например, значения регистров могут помочь определить, какие данные используются в текущий момент времени или как происходит передача параметров между функциями.

**Оптимизация кода**: Анализ значений регистров может помочь в оптимизации кода, так как позволяет понять, какие данные чаще всего используются и какие регистры часто заняты при выполнении определенных участков кода.

**Анализ производительности и оптимизации**: Регистры играют важную роль в производительности программы. Мониторинг регистров может помочь разработчикам оптимизировать код, используя регистры эффективнее для улучшения производительности.

**Реверс-инжиниринг и анализ программ**: В области реверс-инжиниринга анализ значений регистров может помочь при изучении работы программы, понимании алгоритмов и внутреннего устройства кода.

**Таким образом, окно регистров в отладчике предоставляет важную информацию о состоянии процессора и используется для множества целей, связанных с отладкой, анализом производительности и пониманием работы программы на более низком уровне.**

1. **Окно дизассемблированного кода в отладчике используется для просмотра программного кода в виде, более понятном для человека, называемом дизассемблированным кодом. Этот код представляет собой низкоуровневую версию машинного кода, переведенную в более читаемый и понятный вид, состоящий из инструкций ассемблера**.

**Цель использования окна дизассемблированного кода включает в себя следующие аспекты**:

**Анализ кода**: Разработчики могут использовать дизассемблированный код для более подробного анализа того, что происходит в программе на уровне ассемблера. Это может быть полезно при отладке и оптимизации кода, особенно когда требуется более глубокое понимание работы программы.

**Изучение процессорных инструкций**: Дизассемблированный код позволяет разработчикам увидеть, какие конкретные инструкции выполняет процессор во время выполнения программы. Это полезно для понимания производительности кода и его взаимодействия с аппаратным обеспечением.

**Отладка низкоуровневых проблем**: Иногда проблемы возникают на низком уровне, например, связанные с некорректным использованием регистров процессора или неправильными инструкциями. Дизассемблированный код помогает выявлять и исправлять такие проблемы.

**Понимание оптимизации и ассемблерных возможностей**: Разработчики могут изучать дизассемблированный код для понимания того, как компилятор преобразует высокоуровневый код в машинный код и какие ассемблерные инструкции используются для выполнения определенных операций.

**Изучение защиты программы**: В области кибербезопасности и реверс-инжиниринга анализ дизассемблированного кода помогает изучать программы с целью выявления уязвимостей или понимания работы защиты от несанкционированного доступа.

Таким образом, окно дизассемблированного кода предоставляет разработчикам возможность получить более глубокое понимание того, как именно выполняется и работает их программный код на уровне ассемблера, что может быть ценно при отладке, оптимизации и изучении работы программы.

1. **В окне «Дизассемблированный код» отображается код сборки, соответствующий инструкциям, созданным компилятором**.