**Отладка проекта**

**Цель занятия**: закрепление практических навыки работы с системой Visual Studio 2019; научиться использовать инструментальные средства, помогающие провести отладку приложений.

**Оборудование, технические и программные средства:** персональный компьютер, среда программирования Visual Studio 2019.

**Продолжительность занятия:**2 часа.

**Задание:**

1. Создание программы.
2. Проведите отладку программы всеми доступными вам средствами среды разработки.

**Теоретические сведения:**

В C#, как и в других появившихся до .NET языках, главная методика по отладке состоит в добавлении точек останова и изучении того, что происходит в коде в конкретные моменты во время его выполнения.

Точки останова

***Точку останова (breakpoint)*** в Visual Studio можно помещать на любую строку кода, которая в действительности выполняется. Самый простой способ — щелчок на необходимой строке в окне редактора кода внутри затененной области вдоль левого края окна документа (или выделение нужной строки и нажатие клавиши <F9>). Это приводит к размещению в данной строке точки останова, которая вызывает прерывание процесса выполнения и передачу управления отладчику. Как и в предыдущих версиях Visual Studio, точка останова обозначается большим кружком слева от соответствующей строки в окне редактора кода. Кроме того, Visual Studio выделяет саму строку, отображая ее текст и фон разными цветами. Щелчок на кружке приводит к удалению точки останова.

Если останов на определенной строке каждый раз не подходит для решения имеющейся проблемы, можно создать так называемую условную точку останова. Для этого выберите в меню Debug (Отладка) пункт Windows --- Breakpoints (Окнo --- Точки останова). Откроется диалоговое окно, позволяющее указать желаемые детали для точки останова. В этом окне можно выполнять следующие действия:

* Указать, что выполнение должно прерываться лишь после прохождения точки останова определенное количество раз.
* Указать, что точка останова должна вступать в действие при каждом n-ном достижении строки, например, при каждом 20-м ее выполнении (это удобно при отладке больших циклов).
* Задать точки останова относительно переменных, а не команд. В таком случае наблюдение будет вестись за значением указанной переменной, и точки останова будут активизироваться при каждом изменении значения этой переменной. Однако этот вариант может сильно замедлить выполнение кода, поскольку на проверку, не изменилось ли значение отслеживаемой переменной после выполнения каждой очередной инструкции, будет тратиться дополнительное время процессора.

Слежения

После срабатывания точки останова обычно необходимо просмотреть значения переменных. Проще всего это сделать, наведя курсор мыши на имя интересующей переменной прямо в окне редактора кода. Это приводит к появлению небольшого всплывающего окошка, в котором показано значение данной переменной; это окошко можно развернуть, чтобы просмотреть дополнительные детали.

Для просмотра значений переменных можно также использовать *окно Autos (Автоматические)*. Окно Autos представляет собой окно с вкладками, которое появляется лишь тогда, когда программа выполняется в режиме отладки. Если вы его не видите, попробуйте выбрать в меню Debug (Отладка) пункт Windows --- Autos (Окна --- Автоматические).

В этом окне рядом с переменными, которые являются классами или структурами, отображается пиктограмма с изображением знака "плюс", щелчок на которой позволяет развернуть представление переменной и просмотреть значения ее полей.

Три предлагаемых в этом окне вкладки предназначены для наблюдения за переменными трех разных категорий:

* Вкладка *Autos (Автоматические)* позволяет просматривать значения нескольких последних переменных, к которым осуществлялся доступ в процессе выполнения программы.
* Вкладка *Locals (Локальные)* позволяет просматривать значения переменных, к которым получается доступ в методе, выполняемом в текущий момент
* Вкладка *Watch (Слежение)* позволяет просматривать значения любых интересующих переменных за счет явного указания их имен непосредственно в окне Watch.

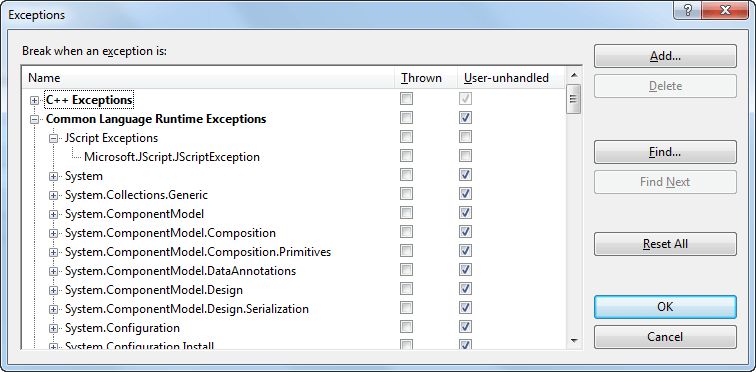
Исключения

Исключения являются замечательным средством для обеспечения надлежащей обработки ошибок в поставляемом приложении. В случае правильного применения они позволяют обрести уверенность в том, что приложению удастся справиться с трудностями, а перед пользователем никогда не появится диалоговое окно с техническим описанием неполадки. К сожалению, во время отладки исключения не столь замечательны. На то имеются *две причины*:

* Если исключение возникает во время отладки, часто не нужно, чтобы оно обрабатывалось автоматически, особенно если это подразумевает аккуратное завершение программы. Напротив, отладчик должен помочь выяснить, по какой причине возникло это исключение. Конечно же, трудность состоит в том, что в случае написания качественного надежного и отказоустойчивого кода, программа будет автоматически обрабатывать практически все, в том числе и неполадки, которые требуется обнаружить.
* Если возникло исключение, для которого не было предусмотрено обработчика, исполняющая среда .NET все равно будет пытаться его найти. К сожалению, к моменту, когда она обнаружит, что обработчик не существует, выполнение программы будет завершаться. Никакого стека вызовов, следовательно, не останется, и просматривать значения каких-либо переменных будет невозможно, потому что все они будут находиться за пределами области видимости.

Конечно, можно устанавливать точки останова в блоках *catch*, но это часто особо не помогает, поскольку при достижении блока catch поток управления по определению покинет соответствующий блок *try*. Это означает, что переменные, значения которых, скорее всего, следовало изучить для выяснения того, что пошло не так, покинут область видимости. Не будет даже возможности просматривать трассировочные данные стека для выяснения, какой метод выполнялся во время срабатывания оператора *throw*, поскольку управление уже покинет этот метод. Разумеется, помещение точки останова в оператор throw позволит решить эту проблему, но надо учитывать, что при написании кода защищенным образом операторов throw будет в коде очень много. Как тогда угадать, какой из них срабатывает и приводит к генерации исключения?

На самом деле в Visual Studio предлагается очень действенное решение. Если заглянуть в **меню Debug (Отладка)**, то можно будет обнаружить там пункт *Exceptions (Исключения)*. В случае выбора этого пункта открывается диалоговое окно Exceptions (Исключения). Это окно позволяет указывать, что должно происходить при выдаче исключения. Здесь можно указать, что выполнение должно продолжаться или же останавливаться с переходом в режим отладки, в случае чего произойдет останов, а отладчик окажется прямо на самом операторе throw:



Visual Studio известно обо всех классах исключений, которые доступны в базовых классах .NET, и о многих таких исключениях, которых могут выдаваться за пределами среды .NET. Распознавать автоматически специальные классы исключений, создаваемые разработчиками, Visual Studio не умеет, но позволяет вручную добавлять такие классы исключений в список и, следовательно, указывать, какие из таких исключений должны приводить к немедленному прекращению выполнения приложения. Для этого необходимо щелкнуть на кнопке *Add (Добавить)*, которая активизируется при выборе самого верхнего узла в дереве, и ввести имя специального класса исключения.

Дополнительные команды отладки исходного кода

Компиляция практически всего коммерческого программного обеспечения на стадии отладки и на стадии подготовки окончательной версии продукта должна проводиться немного по-разному. Среда Visual Studio способна понимать это, поскольку сохраняет информацию обо всех параметрах, которые ей надлежит передавать компилятору. Для поддержки разных вариантов компоновки проекта Visual Studio потребуется сохранять подобную информацию в более чем одном экземпляре. Разные экземпляры такой информации называются конфигурациями. При создании проекта Visual Studio автоматически предлагает на выбор две таких конфигурации, которые называются, соответственно, **Debug (Отладка)** и **Release (Выпуск)**:

* *Конфигурация Debug* обычно указывает, что никакие операции по оптимизации выполняться не должны, в исполняемом коде должна присутствовать дополнительная отладочная информация, а компилятор должен предполагать, что в коде определен препроцессорный символ отладки Debug, если только он не был явно отменен с помощью директивы #undefined.
* *Конфигурация Release* указывает, что компилятор должен проводить в отношении компилируемого кода оптимизацию, в исполняемом коде не должно присутствовать никакой дополнительной информации, а компилятор не должен предполагать наличие препроцессорного символа Debug.

Можно также определять собственные конфигурации. Это необходимо, например, для компоновки приложения с несколькими отличающимися версиями. Раньше из-за проблем, связанных с поддержкой кодировки Unicode в Windows NT, но не в Windows 95, для многих проектов на С++ было принято создавать две конфигурации — одну для Unicode, а вторую для *MBCS* (multibyte character set — набор многобайтных символов).

**Выполнение работы:**

1. Разработайте проект (Для наглядности лучше разработать программу, решающую какую-либо сложную функцию, либо с использованием итераций (цикла)).

2. перейдите в режим отладки. При остановке выполнения программы добавьте в окно просмотра наименования нескольких интересующих Вас переменных.

3. Вызовите окно просмотра. Далее выполняйте программу по шагам. В окне просмотра можно наблюдать интересующие Вас переменные.

4. Установите в программе несколько контрольных точек и запустите программу на выполнение.

5. Просмотрите значения интересующих Вас переменных с помощью окна просмотра, так же как это было предложено в предыдущем пункте.

6. При остановке выполнения программы откройте окно и просмотрите значения интересующих Вас переменных. Измените значение какой-нибудь переменной, записав новое в окно.

7. Выполните такое же задания, используя точки останова (Breakpoint).

8. Продолжите отладку разрабатываемой программы, используя навыки работы с отладчиком.

**Требования к отчету:** Текст должен быть написан шрифтом Times New Roman, 12. Интервал между строками и абзацами – 1,5. Отступ слева 1,5. Ориентация текста – по ширине страницы. Скриншоты необходимо подписать. Название практической работы, цель работы, ход работы, вывод, ответы на контрольные вопросы, должны быть выделены жирным шрифтом, так же как в методичке.

**Контрольные вопросы:**

1. Что такое отладка?

2. Какие инструменты отладки вам известны?

3. Методы отладки.