

Лекция 4.
Оператор switch.
Циклы while, do-while.
Отладка

Проверка на равенство вещ-х чисел

- Для вещественных чисел нельзя использовать проверку на равенство при помощи ==
- Это связано с ошибками округления – компьютер имеет ограниченную точность при работе с вещественными числами
- `Console.WriteLine(2.0 - 1.1 == 0.9);`
`// false`

Проверка на равенство вещ-х чисел

- Поэтому проверку на равенство нужно заменять на проверку, что число лежит в некотором небольшом диапазоне
- $a = b$
- $a - b = 0$
- Теперь позволяем разности отклоняться от 0 в диапазоне от $-\varepsilon$ до ε
- $-\varepsilon \leq a - b \leq \varepsilon$
- $|a - b| \leq \varepsilon$

Проверка на равенство вещ-х чисел

- $|a - b| \leq \varepsilon$

- В коде:

- `double epsilon = 1.0e-10;`

```
if (Math.Abs(a - b) <= epsilon)
{
    // a примерно равен b
}
```

- Команда `Math.Abs(x)` выдает модуль вещественного числа
- В качестве `epsilon` можно брать любое маленькое положительное число

Другие проверки

- А как проверить, что два вещественных числа не равны?
- $|a - b| > \varepsilon$
- Как проверить, что одно число больше другого?
- $a - b > \varepsilon$

Задача

- Прочитать два вещественных числа с консоли
- Проверить, что они равны с учетом погрешности
- Выдать соответствующее сообщение

Константы

Константа

- **Константа** – это переменная, значение которой нельзя изменить
- В C# используется ключевое слово `const`:
- `const int flatsOnFloorCount = 4;`
- Эта переменная обязана быть присвоена сразу же при объявлении
- Второй раз ничего присвоить переменной нельзя, будет ошибка компиляции

Константа

- Константы нужны, чтобы нельзя было случайно (или специально) переприсвоить переменную, значение которой не должно меняться
- Пример: число Пи – `Math.PI`
- В C# константы не являются настоящими переменными – компилятор при компиляции берет значение константы и подставляет во все места использования
- В итоговом коде переменной-константы не будет

Switch

Конструкция switch

- ```
switch (x)
{
 case 0:
 Console.WriteLine(0);
 break;
 case 1:
 Console.WriteLine(1);
 break;
 default:
 Console.WriteLine("иначе");
 break;
}
```
- ```
if (x == 0)
{
    Console.WriteLine(0);
}
else if (x == 1)
{
    Console.WriteLine(1);
}
else
{
    Console.WriteLine("иначе");
}
```

Case соответствует проверке в if
Default соответствует else

Конструкция switch

- ```
switch (x)
{
 case 0:
 Console.WriteLine(0);
 break;
 case 1:
 Console.WriteLine(1);
 break;
 default:
 Console.WriteLine("?");
 break;
}
```

Сначала вычисляется выражение в **switch()**, а затем результат последовательно сравнивается со значениями в **case**'ах

Если результат совпал, то выполняются команды, идущие после :

Каждую ветку **case** нужно завершать ключевым словом **break**. Оно означает, что выполнение блока **switch** закончится, и дальше будет исполняться код, идущий ниже **switch**'а

# Конструкция switch

- ```
switch (x)
{
    case 0:
        Console.WriteLine(0);
        break;
    case 1:
        Console.WriteLine(1);
        break;
    default:
        Console.WriteLine("?");
        break;
}
```

switch может содержать необязательную ветку **default**

Она означает **else**, и выполнится если результат выражения не совпал ни с одним значением

Если в **switch** есть ветка **default**, то она обязана быть последней

Конструкция switch

- В каждой ветке `case` и `default` может быть много команд
- В `case` можно указывать только литералы и константы
- В C# в `switch` можно использовать разные типы, но в основном используются целые числа, строки, символы и枚举 (пройдем на курсе ООП)
- Не стоит использовать в `switch` вещественные числа, т.к. проверка делается не через эпсилон

Конструкция switch

- Ветки в `case` могут ничего не содержать
- Это позволяет выполнить одни и те же команды для нескольких разных значений

- `switch (number)`

```
{
```

```
    case 0:
```

```
    case 1:
```

```
        Console.WriteLine("Это маленькие числа");
```

```
        break;
```

```
    case 2:
```

```
    case 3:
```

```
        Console.WriteLine("А это большие");
```

```
        break;
```

```
}
```

Этот код выполнится если number равен 0 или 1

Этот код выполнится если number равен 2 или 3

Задача

- Прочитать с консоли строку с названием команды
- Если ввели слово `print`, то прочитать с консоли еще одну строку, и напечатать ее
- Если ввели слово `sum`, то прочитать с консоли два вещественных числа, и вывести их сумму
- Если ввели что-то другое, то напечатать, что это неизвестная команда
- Использовать `switch`

Задача на дом «Switch»

- Прочитать с консоли три числа – два операнда и код команды
- Код команды должен быть от 1 до 4
- Если он равен 1, то выполнить сложение первых двух чисел. Если 2, то вычитание, если 3, то умножение, если 4, то деление.
- Если ввели число не от 1 до 4, то вывести, что неизвестная операция
- Использовать `switch`

Краткие операторы. Инкремент и декремент

Операции с присваиванием

- Можно писать

$x += 1;$ вместо $x = x + 1;$

$x -= 4;$ вместо $x = x - 4;$

- Аналогично существуют $*=$, $/=$, $\%=$

Инкремент и декремент

- **Инкремент** – увеличение значения переменной на единицу
- **Декремент** – уменьшение значения переменной на единицу

- Эквивалентны:

$x = x + 1;$

$x += 1;$

$++x;$

$x++;$

$x = x - 1;$

$x -= 1;$

$--x;$

$x--;$

Постфиксный и префиксный варианты

- Вариант $x++$; и $x--$; называется **постфиксным**
- Вариант $++x$; и $--x$; называется **префиксным**
- Разница между ними:
 - Префиксный вариант выполняет инкремент/декремент и выдает новое значение
 - Постфиксный вариант запоминает значение до инкремента/декремента, затем выполняет инкремент/декремент, а затем выдает запомненное старое значение

Постфиксный и префиксный варианты

```
int x = 4;
```

```
Console.WriteLine(++x);
```

```
// выведет 5, новое значение
```

```
Console.WriteLine(x++);
```

```
// 5, т.к. постфиксный оператор выдает старое  
// значение
```

```
Console.WriteLine(x);
```

```
// выведет 6
```

Вопрос

- Что выведет следующий код?
- `int x = 30;`
`int y = 5;`
`Console.WriteLine(x++ + y--);`
`Console.WriteLine(++x - ++y);`
`Console.WriteLine(x);`
`Console.WriteLine(y);`

Печать чисел от 1 до 100

- `Console.WriteLine(1);`
`Console.WriteLine(2);`
...
`Console.WriteLine(99);`
`Console.WriteLine(100);`
- Как напечатать числа от 1 до n, если n мы читаем с консоли?

Циклы

Циклы

- **Цикл** — это конструкция языка, которая позволяет выполнять один и тот же блок кода много раз, пока выполняется некоторое условие
- В C# существует 4 вида циклов:
 - `while`
 - `do-while`
 - `for`
 - `foreach`

Цикл while

- **while** (логическое выражение)
инструкция
- Как работает:
 - Шаг 1. Вычисляется значение логического выражения (**условие цикла**)
 - Шаг 2. Если оно ложно, то цикл завершается
 - Шаг 3. Если оно истинно, то выполняется **тело цикла (инструкция)**. Затем переход на шаг 1

Сумма чисел от 0 до 9

- Часто цикл используют, чтобы пройти по диапазону чисел

- `int i = 0;` `// счетчик цикла`
`int sum = 0;`

```
while (i <= 9)
{
    sum += i;
    ++i;
}
```

```
Console.WriteLine("Сумма = " + sum);
```

Названия счетчиков

- Для названий переменных-счетчиков цикла часто используют короткие имена, состоящие из одной буквы
- Общепринято называть переменную-счетчик буквой *i*
- Если имя *i* уже занято, то использовать *j, k, m, n* и так далее
- Букву *l* пропускают, т.к. она похожа на 1

Термины циклов

- **while** (логическое выражение)
инструкция
- Логическое выражение называется **условием цикла** (или **условием продолжения цикла**)
- Инструкция называется **телом цикла** – код, который выполняется внутри цикла
- Одно выполнение тела цикла называется **итерацией**

Задача

- Найти сумму чисел от 0 до 9
- Переделать программу, чтобы найти сумму от 3 до 21 включительно
- Переделать программу, чтобы найти сумму только четных чисел от 3 до 21 включительно
- Дополнительно найти количество четных чисел от 3 до 21 включительно

Цикл while

- Тело цикла может не выполниться **ни разу**, если условие сразу было ложным
- Если условие всегда истинно, то цикл выполняется бесконечно. Это называется **зацикливанием** и обычно является ошибкой
- В примере зацикливание может произойти если забыть сделать `++i`;

Задача на дом «Среднее арифметическое»

- Написать программу, вычисляющую среднее арифметическое чисел из некоторого диапазона чисел (например, от 3 до 17)
- Концы диапазона задать переменными, начальное число должно быть > 1 , чтобы было посложнее
- **Среднее арифметическое чисел** – нужно сумму всех чисел поделить на количество этих чисел
- В этом же классе - найти среднее арифметическое только четных чисел из этого диапазона чисел

Задача на курс «Числа Фибоначчи»

- Написать программу, которая принимает с консоли целое число n и возвращает число Фибоначчи с номером n .
- Числа Фибоначчи задаются следующим образом:
- $f_0 = 0, f_1 = 1, f_n = f_{n-1} + f_{n-2}$

Цикл do-while

- do
инструкция
while (логическое выражение);
- Как всегда, инструкция – 1 команда или блок кода в фигурных скобках
- Как работает:
 - Шаг 1. Выполняется тело цикла
 - Шаг 2. Проверятся условие. Если истинно, то возвращаемся на шаг 1. Если ложно, то конец цикла

Цикл do-while: сумма чисел от 0 до 100

- `int i = 0;`
`int sum = 0;`

`do`

`{`

`sum += i;`

`++i;`

`}`

`while (i <= 100);`

`Console.WriteLine("Сумма = " + sum);`

Цикл do-while

- Отличие от `while`: тело цикла `do-while` всегда выполняется хотя бы 1 раз, т.к. первая проверка условия происходит после 1 итерации цикла

Задача на дом «Do-while»

- Сделать задачу про среднее арифметическое с циклом `do-while`
- Эту задачу нужно присылать в одну цепочку писем с версией с `while`

Задача на дом «Сумма ряда»

- Написать программу, которая находит результат такого выражения:
 - $1 - 4 + 9 - 16 + 25 - 36 \dots$
- Количество чисел в этом выражении должно быть параметром программы

Задача на дом «10 чисел в строке»

- Распечатать числа от 1 до 100 при помощи цикла `while`, но выводить по 10 чисел в строке, дальше делать перевод строки
- 1 2 3 4 5 6 7 8 9 10
11 12 13 14 15 16 17 18 19 20
...

• Сложная версия задачи:
 - Выводить числа ровно, чтобы они были друг под другом. Использовать возможности `Write/WriteLine`
 - Возможность задать начальное и конечное число, и по сколько чисел в строке выводить

Задача на дом «Цифры числа»

- Прочитать с консоли целое число
- Найдите сумму его цифр
- Найдите сумму только тех цифр числа, которые являются нечетными числами
- Найдите максимальную цифру числа

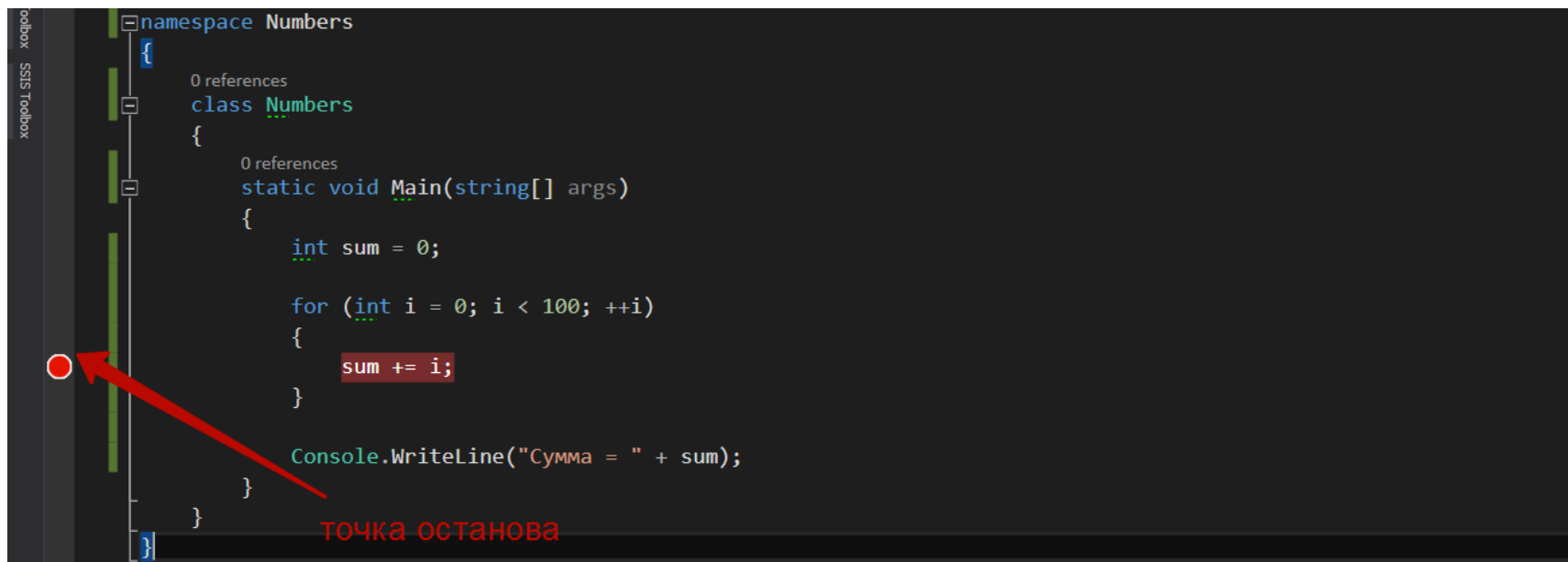
Отладка программ

Отладка программ

- **Отладка программ** – процесс поиска ошибок
- По-английски – **debug**
- Среда разработки, в том числе Visual Studio предоставляют удобные средства отладки

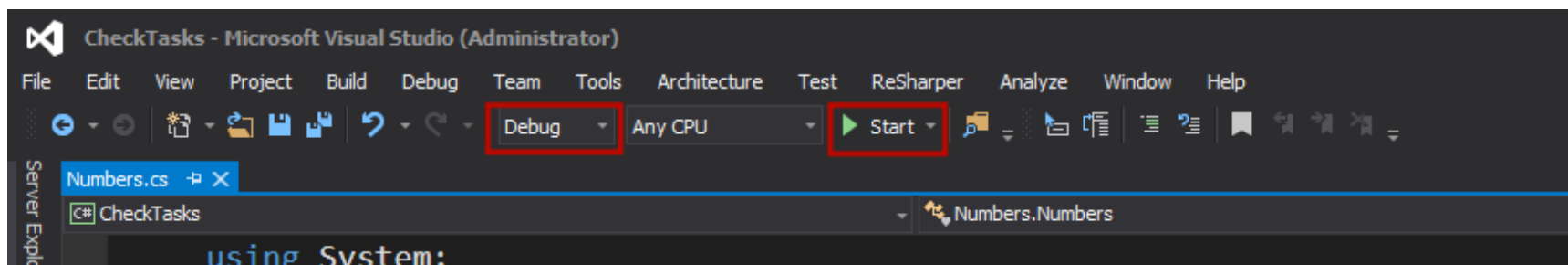
Точки останова

- Точки останова (**breakpoints**)
- Позволяют остановить исполнение программы в указанном месте, когда поток исполнения достигнет его
- Добавляются/убираются кликом по столбцу слева



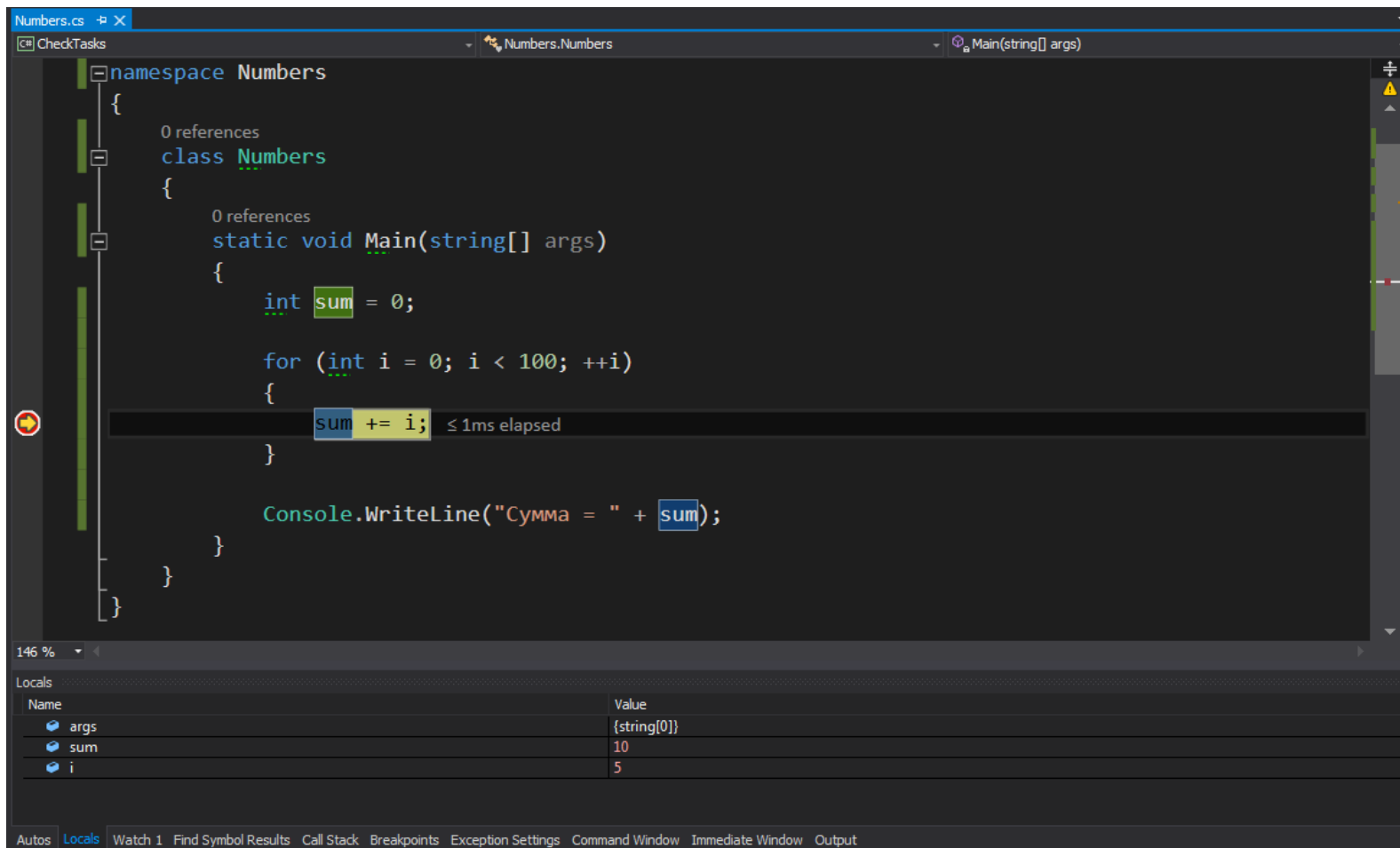
Запуск отладки

- Программа останавливается на точках останова только в режиме Debug
- Для отладки нужно запускать программу через Debug



Просмотр значений переменных

- Когда программа остановлена, во вкладке Locals можно смотреть текущие значения переменных



The screenshot shows the Visual Studio IDE with a C# project named 'Numbers'. The code is as follows:

```
namespace Numbers
{
    0 references
    class Numbers
    {
        0 references
        static void Main(string[] args)
        {
            int sum = 0;

            for (int i = 0; i < 100; ++i)
            {
                sum += i; ≤ 1ms elapsed
            }

            Console.WriteLine("Сумма = " + sum);
        }
    }
}
```

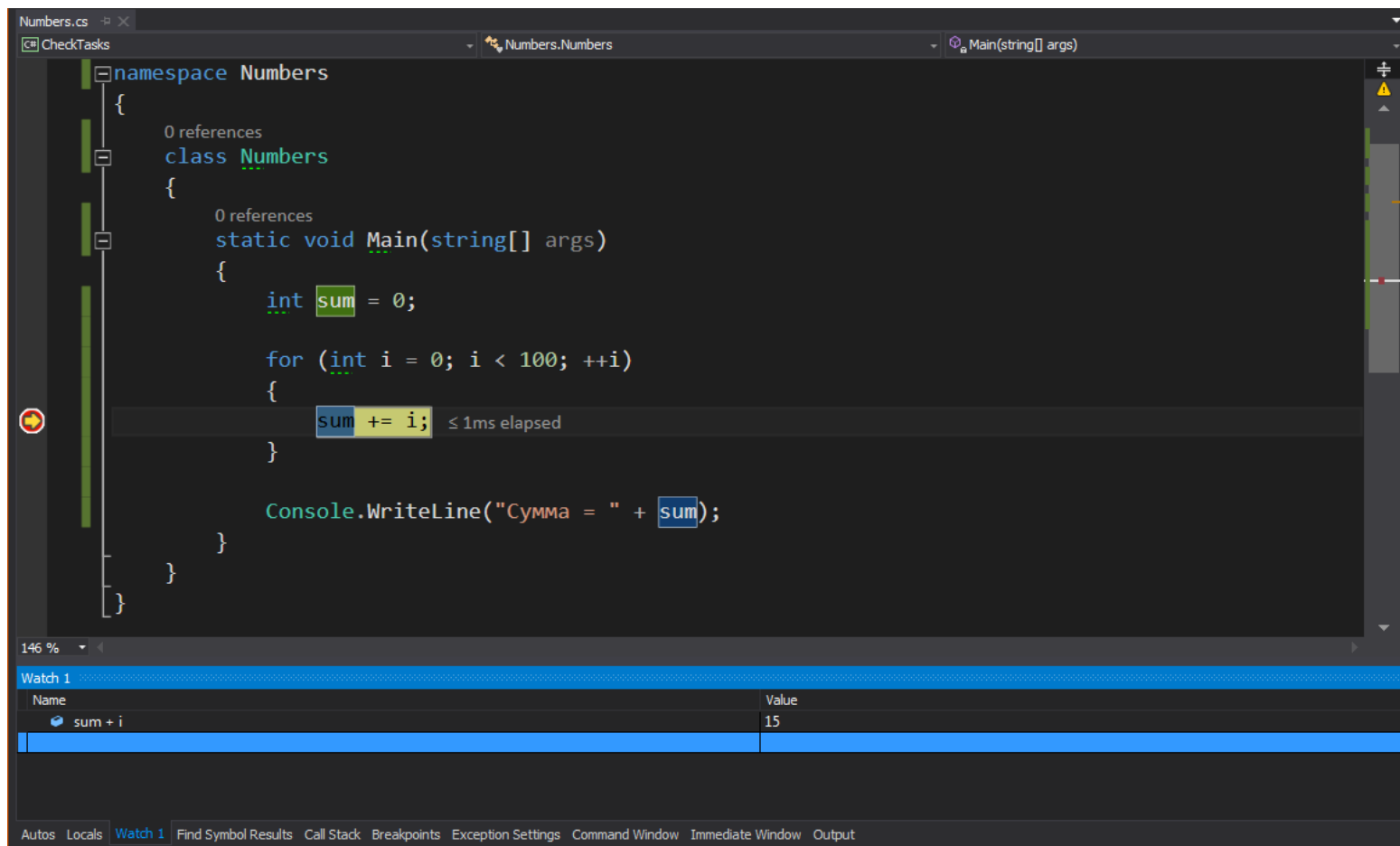
The program is paused at the line `sum += i;`. The 'Locals' window at the bottom displays the following variables and their values:

Name	Value
args	{string[0]}
sum	10
i	5

The status bar at the bottom indicates the 'Locals' tab is active, along with other debugging tools like 'Watch 1', 'Find Symbol Results', 'Call Stack', 'Breakpoints', 'Exception Settings', 'Command Window', 'Immediate Window', and 'Output'.

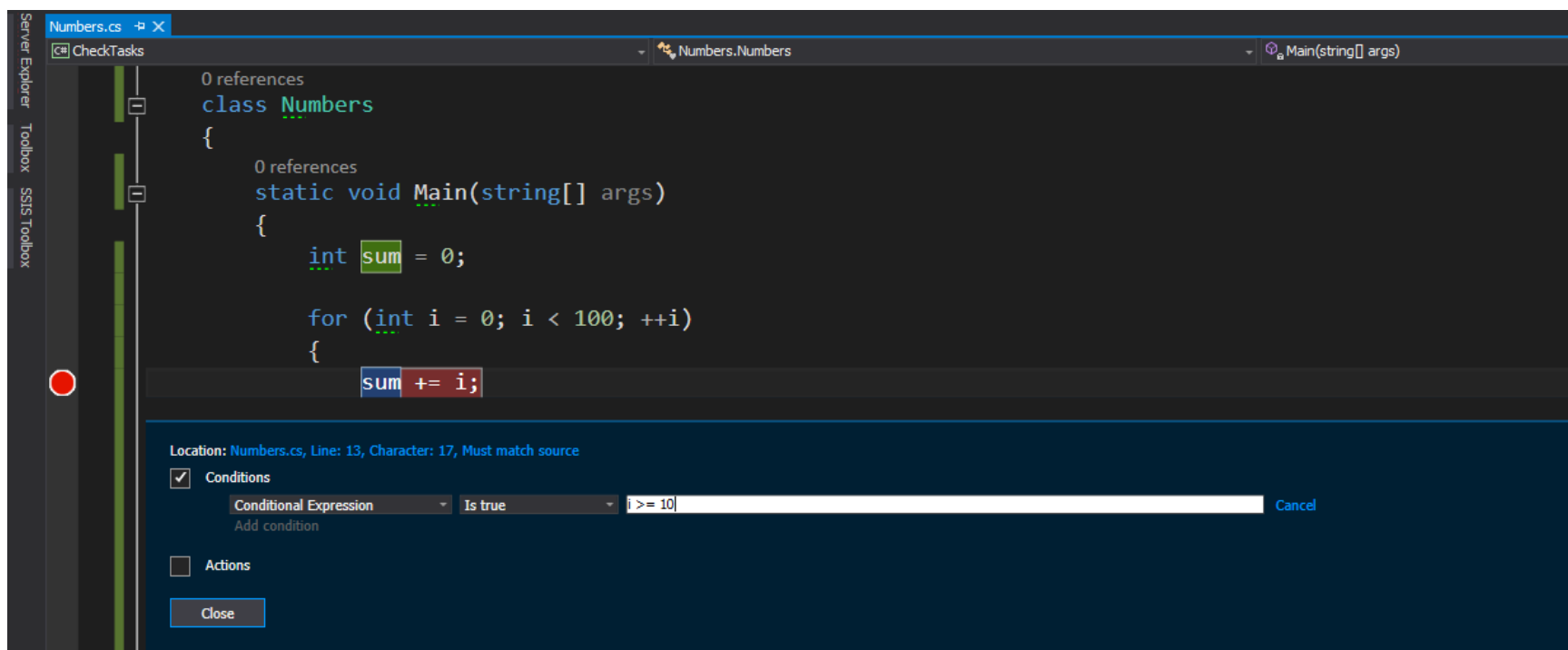
Просмотр результатов выражений

- Когда программа остановлена, во вкладке Watch можно смотреть значение любого выражения, которое хочется проверить



Точки останова с условием

- Для точки останова можно задать условие, когда она будет срабатывать



Пошаговая отладка

- Часто бывает полезна **пошаговая отладка** – по нажатию кнопки будет выполняться по одной команде
- Есть два вида пошаговой отладки:
 - с заходом в функцию (**step into**) – F11
 - без захода в функцию (**step over**) – F10

Пошаговая отладка

- С заходом в функцию (**Step Into**) – F11
- Без захода в функцию (**Step Over**) – F10

