



Techniki śledzenia aplikacji

Adrian Kunikowski
Zaawansowane Techniki Programowania
Wydział Matematyki i Informatyki Uniwersytetu Łódzkiego

Czym jest śledzenie aplikacji?

Śledzenie aplikacji to monitorowanie jej wykonywania w trakcie jej działania. Możemy wyróżnić dwie główne metody śledzenia aplikacji:

- Śledzenie kodu
- Debugowanie

Śledzenie kodu

Śledzenie kodu to nic innego, jak sprawdzanie jego przebiegu. Możemy to robić metodą prostą – zwyczajnie powoli analizując przebieg określonych danych, rozplanowując ścieżkę do przodu.

```
private static void Func1(int num)
{
    Console.WriteLine("Wszedłem w func1!");
    while (num > 2)
    {
        num = num / 2;
    }
    Console.WriteLine("Wyszedłem z func1!");
}
```

Możemy również zostawiać sobie „informacje” które fragmenty kodu się wywołały – jest wiele podejść do tego, w celach testowych z grubsza możemy wykorzystywać każde z nich – nie możemy tylko zapomnieć, by potem takie „wskaźniki” usunąć z kodu gdy wejdziemy w fazę produkcyjną.

Adrian Kunikowski

Zaawansowane Techniki Programowania

Wydział Matematyki i Informatyki Uniwersytetu Łódzkiego

Dlaczego to jest ważne?

Jesteśmy ludźmi, co sprawia, że od czasu do czasu popełniamy błędy. Śledzenie kodu ułatwia nam redukcję błędów, ale nie zmienia faktu, że potencjalnie będziemy je dalej popełniać. Śledzenie kodu pozwoli nam namierzyć gdzie taki błąd wystąpił, co przy gigantycznych projektach jest kluczowe.

```
Wszedłem w func1!  
Wyszedłem z func1!  
  
Wszedłem w func2!  
Wyszedłem z func2!  
  
Wszedłem w func3!
```

Debugowanie – co to jest?

Debugowanie jak sama nazwa wskazuje, służy pozbywaniu się błędów z programu. Na poniższym przykładzie nasz program zamarł w metodzie func3. W takim razie wiemy, gdzie powinniśmy zajrzeć.

```
Wszedłem w func1!  
Wyszedłem z func1!  
  
Wszedłem w func2!  
Wyszedłem z func2!  
  
Wszedłem w func3!
```

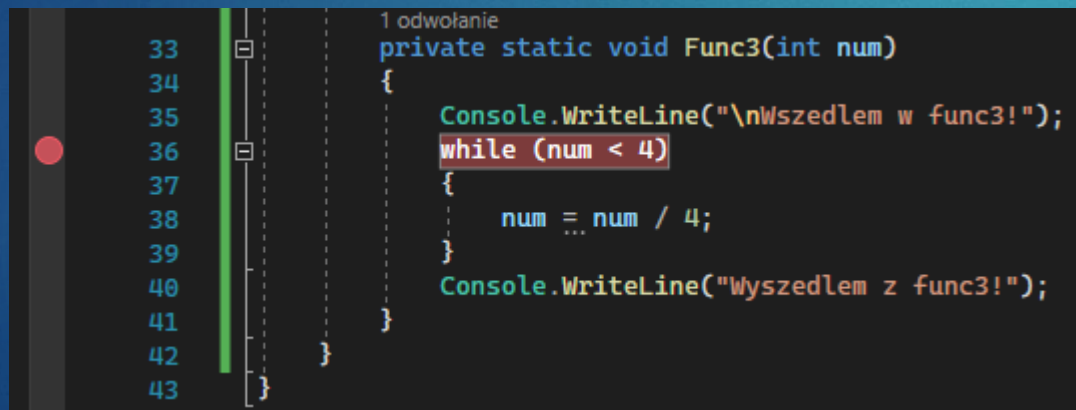
Adrian Kunikowski

Zaawansowane Techniki Programowania

Wydział Matematyki i Informatyki Uniwersytetu Łódzkiego

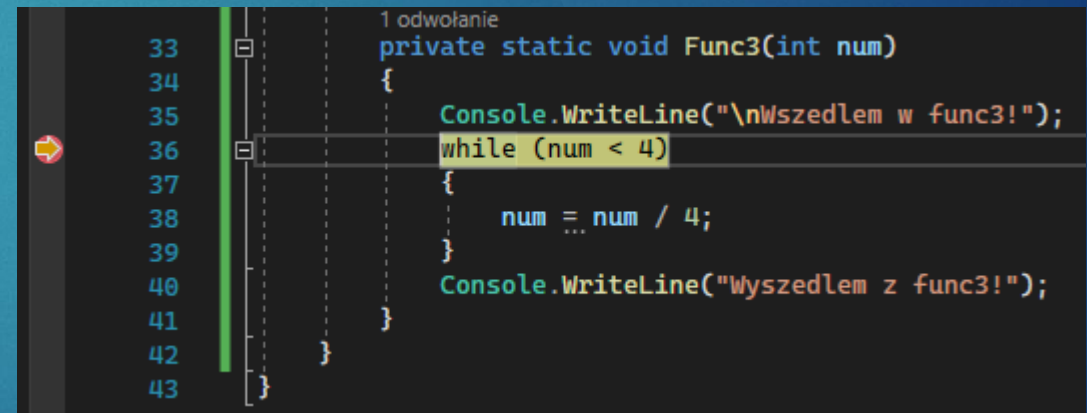
Breakpointy

Breakpointy pozwalają nam zrobić punkt „STOP” w naszym kodzie. W momencie wykonywania kodu zatrzymamy się na wybranym momencie. W wypadku zatrzymywania się podczas instrukcji pętli – zatrzymamy się każdorazowo.



The screenshot shows a code editor with a dark background. On the left, a vertical line of line numbers from 33 to 43 is visible. A red dot, representing a breakpoint, is placed on line 36. To the right of the line numbers, the code for a function `Func3` is displayed. The code includes a `while` loop on line 36. A vertical dashed line indicates the current execution point, which is aligned with the breakpoint on line 36.

```
1 odwołanie
private static void Func3(int num)
{
    Console.WriteLine("\nWszedłem w func3!");
    while (num < 4)
    {
        num = num / 4;
    }
    Console.WriteLine("Wyszedłem z func3!");
}
```



The screenshot shows the same code editor as the previous one, but with a yellow highlight on the `while (num < 4)` condition on line 36. A yellow arrow points to this condition, indicating that the breakpoint is set to trigger every time the loop condition is evaluated.

```
1 odwołanie
private static void Func3(int num)
{
    Console.WriteLine("\nWszedłem w func3!");
    while (num < 4)
    {
        num = num / 4;
    }
    Console.WriteLine("Wyszedłem z func3!");
}
```

Breakpointy – c.d

Takie zatrzymanie pozwala nam zobaczyć stan programu w momencie, w którym się zatrzymaliśmy, zmieniać je w biegu itd.

Jesteśmy w stanie zobaczyć wartości wszystkich zmiennych

Jak i również je modyfikować

```
private static void Func3(int num)
{
    Console.WriteLine("\nWszedłem w func3!");
    while (num < 4)
    {
        num = num / 4;
    }
    Console.WriteLine("Wszedłem z func3!");
}
```

1 odwołanie

```
private static void Func3(int num)
{
    Console.WriteLine("\nWszedłem w func3!");
    while (num < 4)
    {
        num = num / 4;
    }
    Console.WriteLine("Wszedłem z func3!");
}
```

≤ 1 ms upłynęło

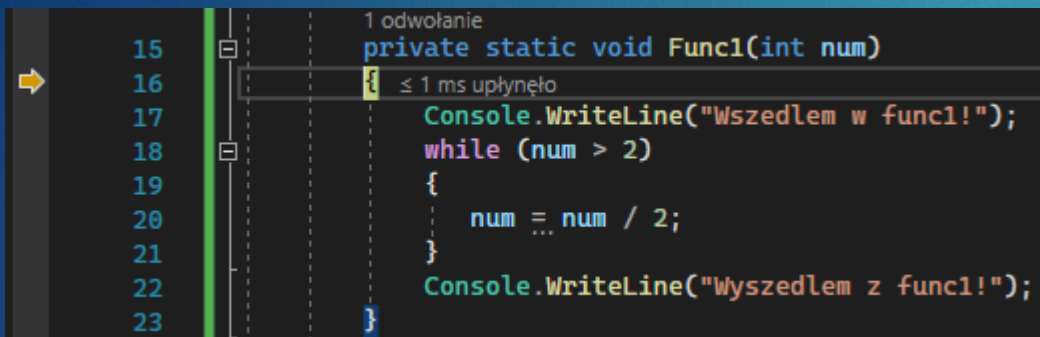
```
private static void Func3(int num)
{
    Console.WriteLine("\nWszedłem w func3!");
    while (num < 4)
    {
        num = num / 4;
    }
    Console.WriteLine("Wszedłem z func3!");
}
```

1 odwołanie

Step Into oraz Step Over

Wyróżniamy jeszcze dwie funkcje związane z breakpointami:

Step Into

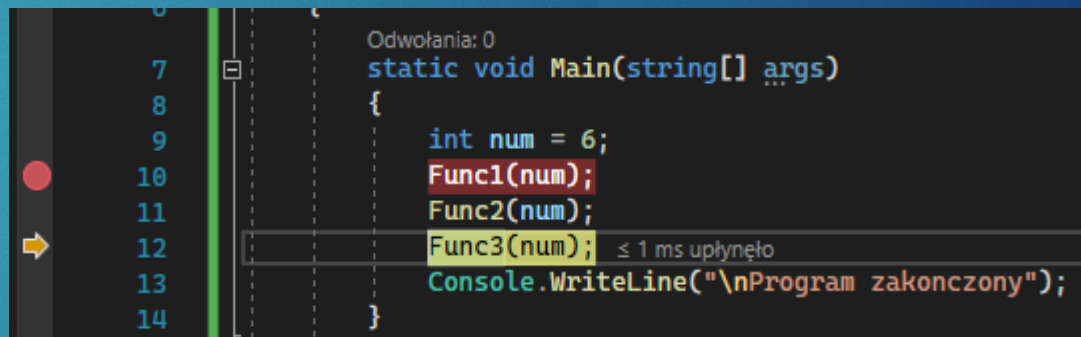


The screenshot shows a code editor with a C# method `Func1`. A yellow arrow on the left margin points to line 16, indicating the Step Into action. A tooltip on the right shows the call stack: `1 odwołanie` (1 call) at line 16, with a duration of `≤ 1 ms upłynęło` (≤ 1 ms has passed). The code is as follows:

```
15 private static void Func1(int num)
16 {
17     Console.WriteLine("Wszedłem w func1!");
18     while (num > 2)
19     {
20         num = num / 2;
21     }
22     Console.WriteLine("Wyszedłem z func1!");
23 }
```

(Pozwala wejść do metody)

Step Over



The screenshot shows a code editor with a C# `Main` method. A yellow arrow on the left margin points to line 12, indicating the Step Over action. A tooltip on the right shows the call stack: `Odwołania: 0` (Calls: 0) at line 12, with a duration of `≤ 1 ms upłynęło` (≤ 1 ms has passed). The code is as follows:

```
7 static void Main(string[] args)
8 {
9     int num = 6;
10    Func1(num);
11    Func2(num);
12    Func3(num);
13    Console.WriteLine("\nProgram zakończony");
14 }
```

(Pozwala „przeskoczyć” do kolejnej linijki kodu)

Adrian Kunikowski

Zaawansowane Techniki Programowania

Wydział Matematyki i Informatyki Uniwersytetu Łódzkiego

Warte wspomnienia

Wśród funkcji debugowania warta wspomnienia jest również możliwość „wykonania kodu” do momentu zaznaczonego przez nas, bez konieczności wstawiania Breakpointów.

```
Odwołania: 0
static void Main(string[] args)
{
    int num = 6;
    Func1(num);
    Func2(num);
    Func3(num);
    Console.WriteLine("\nProgram zakończony");
}
```



Dziękuję za uwagę

Adrian Kunikowski
Zaawansowane Techniki Programowania
Wydział Matematyki i Informatyki Uniwersytetu Łódzkiego