

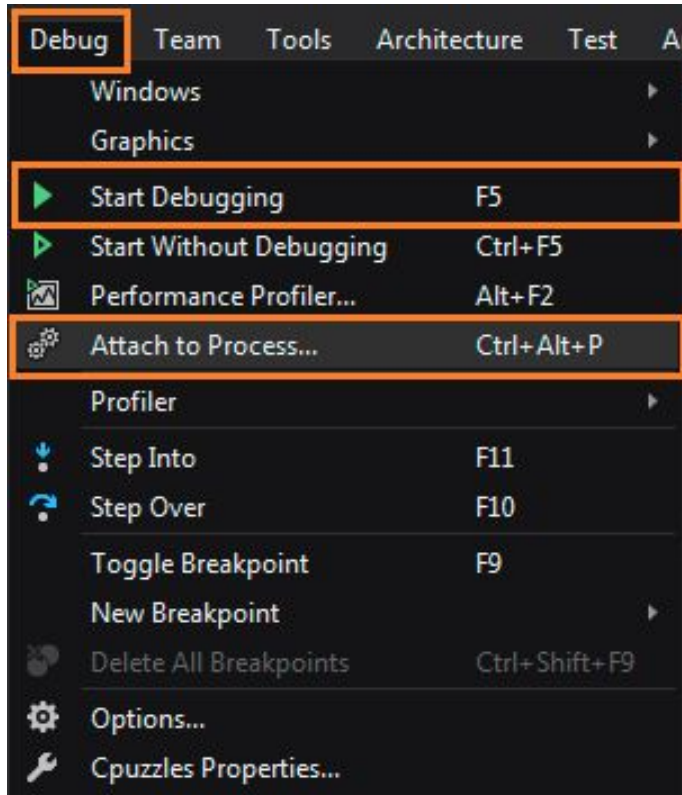
# Отладка в Visual Studio

# Отладка

Существуют две технологии отладки:

- Использование отладчиков — программ, которые включают в себя пользовательский интерфейс для пошагового выполнения программы: оператор за оператором, функция за функцией, с остановками на некоторых строках исходного кода или при достижении определённого условия.
- Вывод текущего состояния программы с помощью расположенных в критических точках программы операторов вывода — на экран или в файл (создание логов).

# Как начать



Нажать F5.  
Отладка начнется  
если стоят точки  
останова (**breakpoints**)

# Точки останова (Breakpoints)



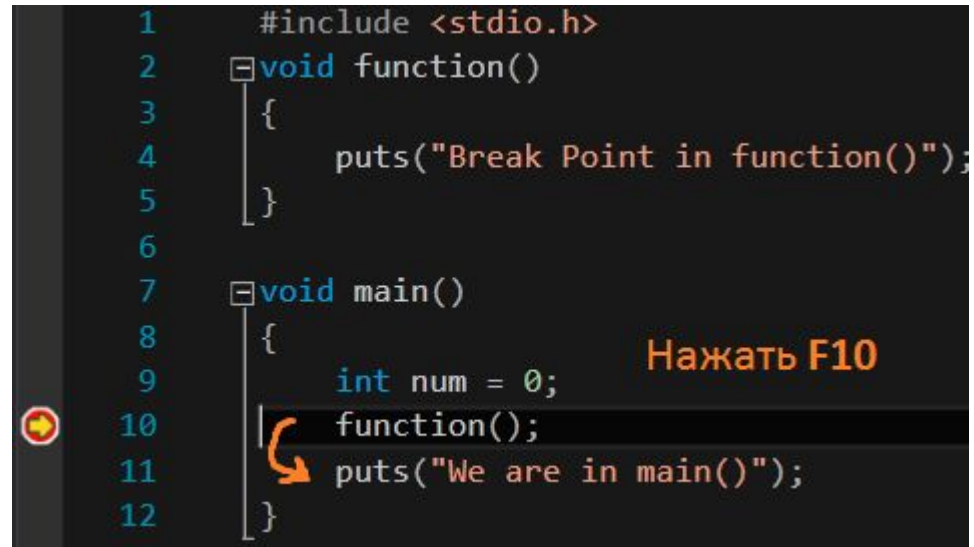
```
1  #include <stdio.h>
2
3  void main()
4  {
5      char *names[] = { "John", "Lisa", "Albert", "Lana"};
6
7      for(int i = 0; i < 4; i++)
8      puts(names[i]);
9  }
```

The image shows a code editor with a dark background. A green vertical bar on the left indicates line numbers 1 through 9. A red circle, representing a breakpoint, is located on the left margin next to line 8. An orange rectangular box highlights the entire line 8, which contains the code `puts(names[i]);`. The code is written in a light blue/cyan color.

- Точки останова используются чтобы показать , где отладчику необходимо остановиться.
- Точка ставится кликом на сайдбар слева от исходного кода, либо нажатием на **F9**.
- Точки останова обычно ставятся там, где есть сомнения в корректности кода.

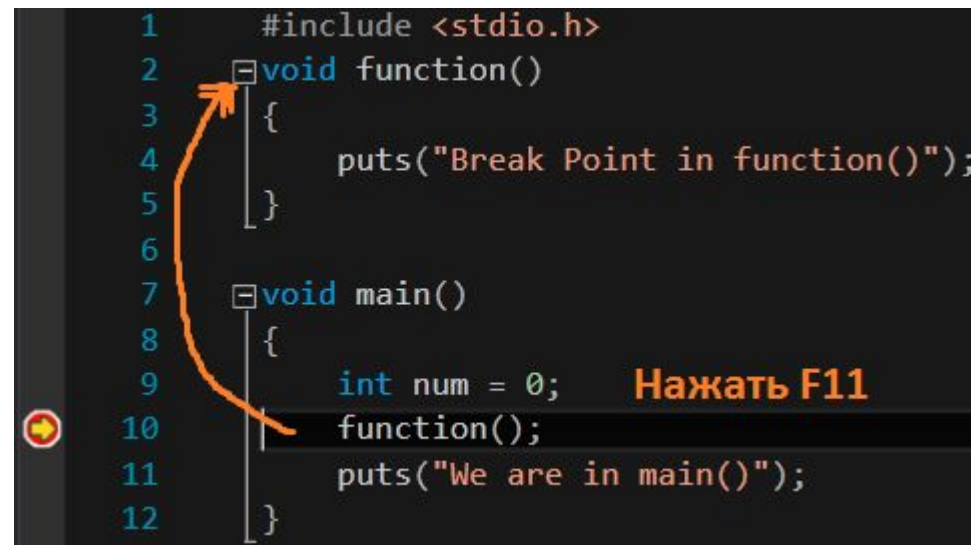
# Отладка с использованием точек останова (Debugging with Breakpoints)

- **Перешагнуть (Step Over) F10** – автоматически выполняет блок кода под курсором.
- **Зайти (Step Into) F11** – заходит в блок кода под курсором.
- **Выйти (Step Out) Shift + F11** - выходит из текущего блока.
- **Продолжить (Continue) F5** - переходит к следующей точки останова.



```
1  #include <stdio.h>
2  void function()
3  {
4      puts("Break Point in function()");
5  }
6
7  void main()
8  {
9      int num = 0;
10     function();
11     puts("We are in main()");
12 }
```

Нажать F10



```
1  #include <stdio.h>
2  void function()
3  {
4      puts("Break Point in function()");
5  }
6
7  void main()
8  {
9      int num = 0;
10     function();
11     puts("We are in main()");
12 }
```

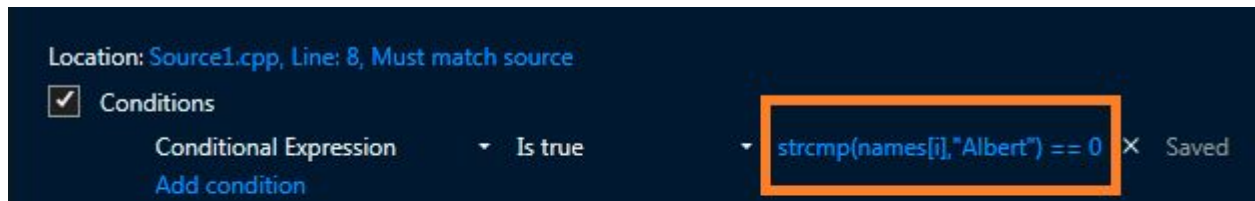
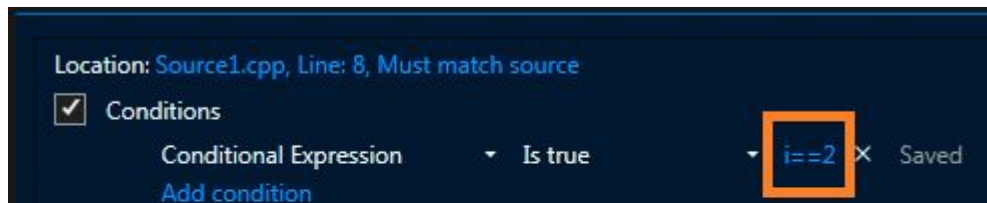
Нажать F11

# Условные остановки (Conditional Breakpoint)



- В циклах может обрабатываться большое количество данных.
- Условная остановка нужна чтобы остановить выполнение кода в нужном месте

# Условные остановки (Conditional Breakpoint)



```
{  
    char *names[] = { "John",  
    for(int i = 0; i < 4; i++)  
        puts(names[i]);  
}
```

```
{  
    char *names[] = { "John", "Lisa", "Albert",  
    for(int i = 0; i < 4; i++)  
        puts(names[i]);  
}
```

▶ names[i] 🔍 0x00f76b58 "Albert" ⇐

# Количество остановок (Breakpoint Hit Count)

The screenshot shows a debugger interface with a source code window and a breakpoint settings panel. The source code is a C++ program with a loop that prints names from an array. A breakpoint is set at line 8, which is inside the loop. The breakpoint settings panel shows the location as 'Source1.cpp, Line: 8, Must match source'. The 'Conditions' checkbox is checked, and the condition is set to 'Hit Count' followed by an equals sign and the number '3'. The current hit count is 3, and there are buttons for 'Reset' and 'Saved'. Below the breakpoint settings, there are two panels: 'Autos' and 'Breakpoints'. The 'Autos' panel shows variables 'i' (value 2), 'names' (array of strings), and 'names[i]' (current value 'Albert'). The 'Breakpoints' panel shows a list of breakpoints with columns for Name, Labels, Condition, and Hit Count. The breakpoint at 'Source1.cpp, line 8' has the label 'NAMES', condition '(no condition)', and hit count 'when hit count is equal to 3 (currently 3)'.

```
4 {  
5     char *names[] = { "John", "Lisa", "Albert", "Lana"};  
6  
7     for(int i = 0; i < 4; i++)  
8         puts(names[i]);
```

Breakpoint Settings

Location: Source1.cpp, Line: 8, Must match source

☒ Conditions

Hit Count = 3 (Current: 3 Reset) X Saved

Add condition

132 %

Autos

Name	Value	Type
i	2	int
names	0x002df8ec {0x00f76b30 "John", 0x00f76b50 "Lisa", 0x00f76b58 "Albert", 0x00f76b5f "Lana"}	char *[4]
names[i]	0x00f76b58 "Albert"	char *

Breakpoints

Name	Labels	Condition	Hit Count
Source1.cpp, line 8	NAMES	(no condition)	when hit count is equal to 3 (currently 3)

Отслеживание сколько остановок отладчик сделает на конкретной точке  
останова



# Подсказки (Data Tip)

```
#include <stdio.h>

void main()
{
    char *names[] = { "John", "Lisa", "Albert", "Lana"};

    for(int i = 0; i < 10; i++)
        puts(names[i]); ≤ 2ms elapsed
}
```

 i | 4

 (names)[0]	↗ 0x00366b30 "John"
 (names)[1]	↗ 0x00366b50 "Lisa"
 (names)[2]	↗ 0x00366b58 "Albert"
 (names)[3]	↗ 0x00366b60 "Lana"

 names[i] | 0xffffffff <Error reading characters of string.>

- Можно через подсказки менять значения

# Окно просмотра данных (Watch Windows)

The screenshot shows a debugger interface with a C program and its local variables. The program is as follows:

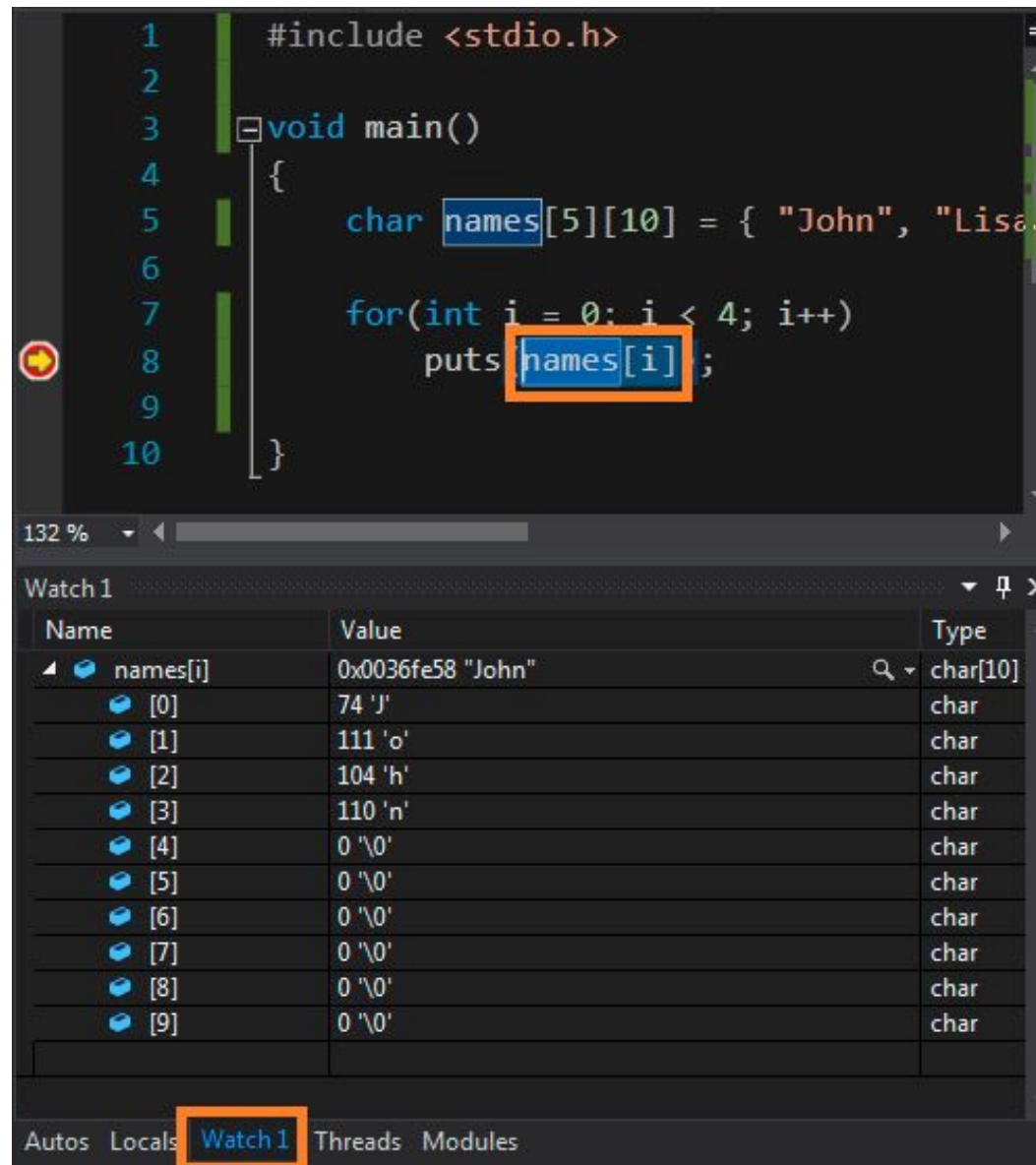
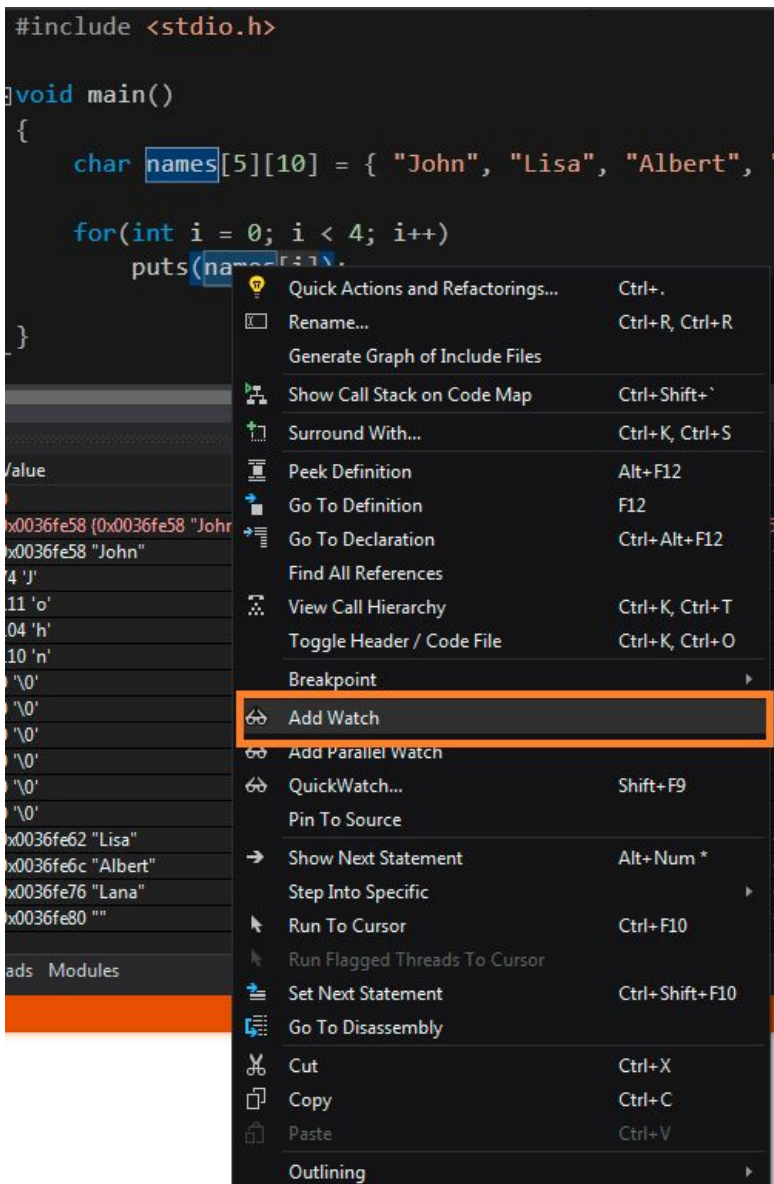
```
1  #include <stdio.h>
2
3  void main()
4  {
5      char names[5][10] = { "John", "Lisa", "Albert", "Lana"};
6
7      for(int i = 0; i < 4; i++)
8          puts(names[i]);
9  }
```

The `names` array is highlighted with an orange box. The `Locals` window below shows the current state of the program's local variables:

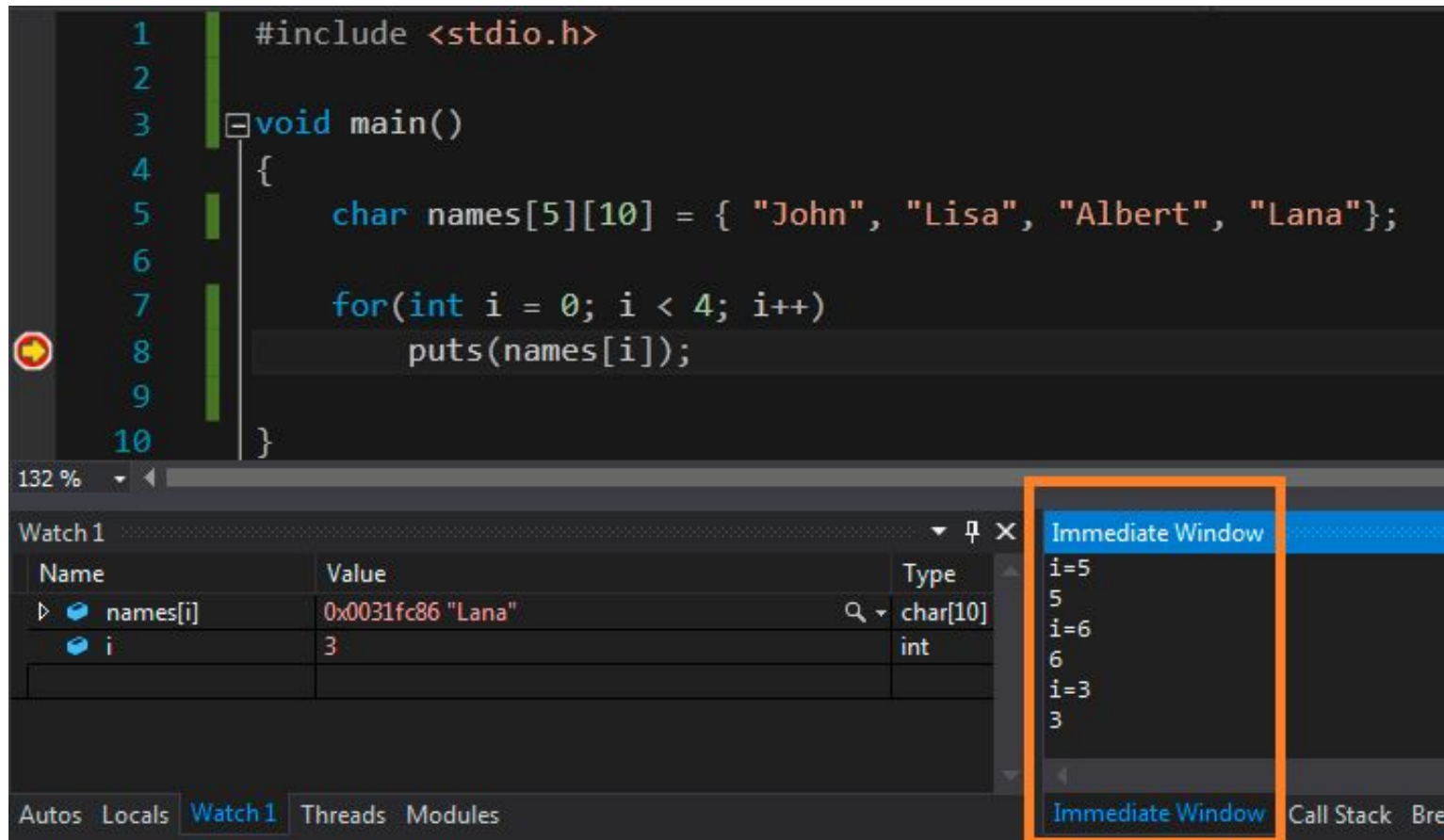
Name	Value	Type
<code>i</code>	0	int
<code>names</code>	0x0036fe58 {0x0036fe58 "John", 0x0036fe62 "Lisa", 0x0036fe6c "Albert", 0x0036fe76 "Lana", 0x0036fe80 ""}	char[5][10]
<code>[0]</code>	0x0036fe58 "John"	char[10]
<code>[0]</code>	74 'J'	char
<code>[1]</code>	111 'o'	char
<code>[2]</code>	104 'h'	char
<code>[3]</code>	110 'n'	char
<code>[4]</code>	0 '\0'	char
<code>[5]</code>	0 '\0'	char
<code>[6]</code>	0 '\0'	char
<code>[7]</code>	0 '\0'	char
<code>[8]</code>	0 '\0'	char
<code>[9]</code>	0 '\0'	char
<code>[1]</code>	0x0036fe62 "Lisa"	char[10]
<code>[2]</code>	0x0036fe6c "Albert"	char[10]
<code>[3]</code>	0x0036fe76 "Lana"	char[10]
<code>[4]</code>	0x0036fe80 ""	char[10]

The `Locals` window is highlighted with an orange box. The `Autos` window is also visible at the bottom left.

# Окно просмотра данных (Watch Windows)



# Оперативные изменения (Immediate Window)



- **Debug > Window > Immediate Window**
- Позволяет задавать значения выражений/переменных во время отладки