DTS202TC Fundamentals of Parallel Computing

Tutorial 3: Introduction of Debugging

Xi'an Jiaotong-Liverpool University
西交利物浦大學

Debug: To look for and remove the faults in a computer program

Setting **Break point** to do the debug

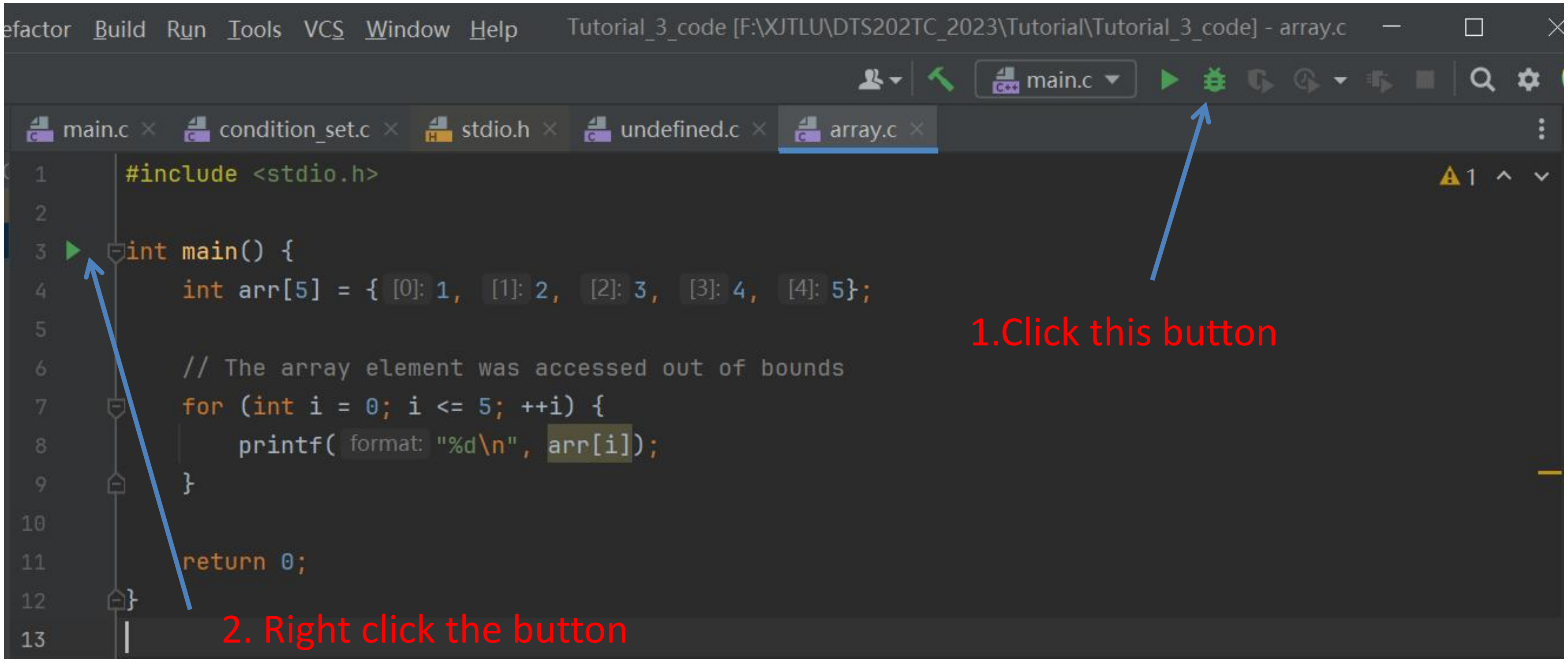
or add the flag **-g** when you use command line or write makefile file

For example: gcc -Wall **-g** -o the_name_of_executable_file the_name _of_.c_file

What is Break point:

Break Point can be understood as an obstacle.
People cannot walk when encountering an obstacle,
and the program suspends execution when it encounters a breakpoint.

The location of debug button

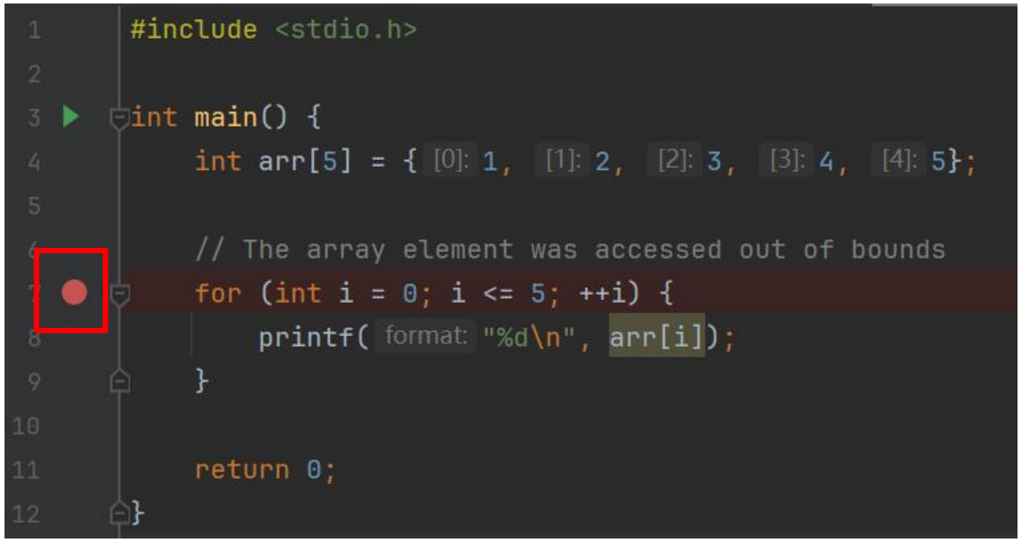

efactor Build Run Tools VCS Window Help
Tutorial_3_code [F:\XJTLU\DTS202TC_2023\Tutorial\Tutorial_3_code] - array.c
main.c
main.c
condition_set.c
stdio.h
undefined.c
array.c
1.Click this button
2. Right click the button
then choose Debug

```
#include <stdio.h>

int main() {
    int arr[5] = { [0]: 1, [1]: 2, [2]: 3, [3]: 4, [4]: 5};

    // The array element was accessed out of bounds
    for (int i = 0; i <= 5; ++i) {
        printf( format: "%d\n", arr[i]);
    }

    return 0;
}
```

Make sure that the whole file address is ENGLISH!

How to set a Break point

Left click the line you want to set the break point



```c
#include <stdio.h>

int main() {
    int arr[5] = { [0]: 1, [1]: 2, [2]: 3, [3]: 4, [4]: 5};

    // The array element was accessed out of bounds
    for (int i = 0; i <= 5; ++i) {
        printf( format: "%d\n", arr[i]);
    }

    return 0;
}
```

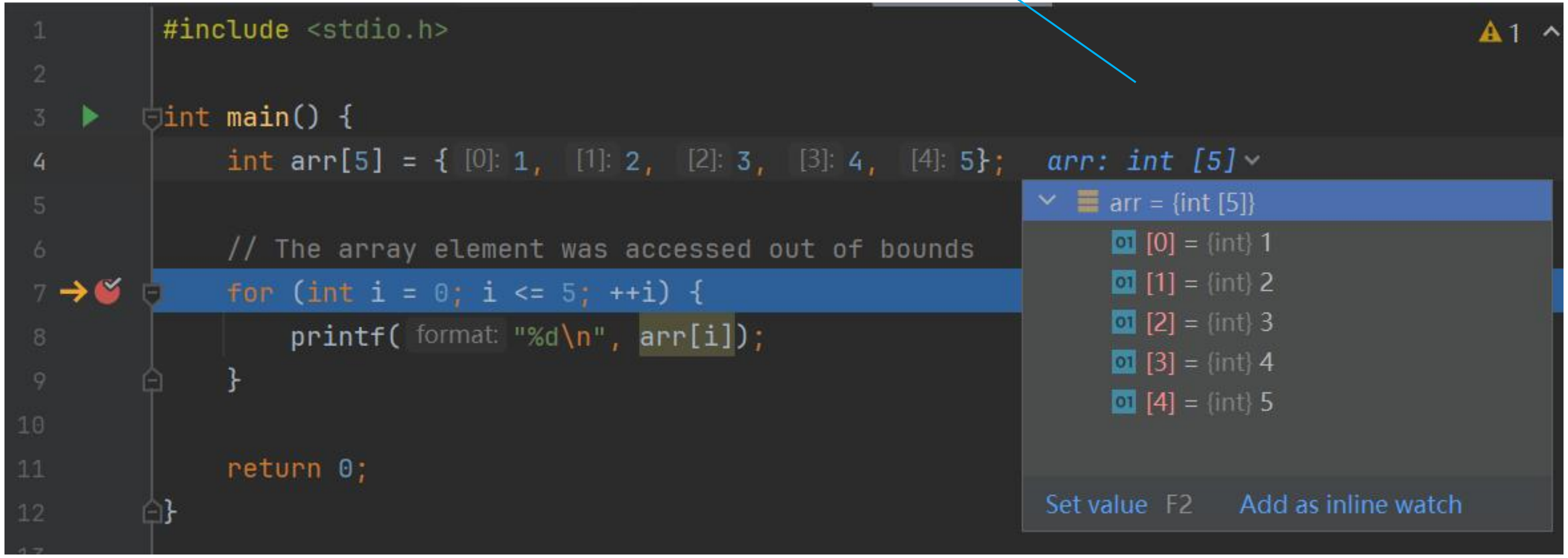
Break point will hang up the program and Put the program into a special state - the interrupt state

In this state, the program **will not** be stopped and the element will not be deleted like variable.

During the debugging process, multiple breakpoints can be set, and the program will pause every time a breakpoint is encountered during execution

Things you need to focus while Debugging

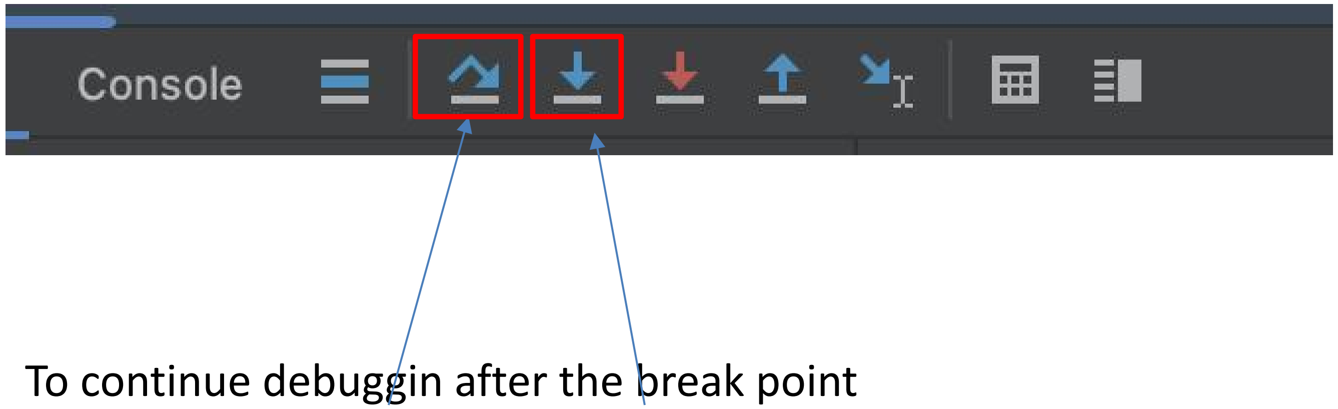
During the debugging, the most important thing to do is to check the state of the variable
In clion, During debugging, the state of the variable will be shown on the right of the variable



```c
#include <stdio.h>

int main() {
    int arr[5] = { [0]: 1, [1]: 2, [2]: 3, [3]: 4, [4]: 5};   arr: int [5]

    // The array element was accessed out of bounds
    for (int i = 0; i <= 5; ++i) {
        printf( format: "%d\n", arr[i]);
    }

    return 0;
}
```

```
arr = {int [5]}
    [0] = {int} 1
    [1] = {int} 2
    [2] = {int} 3
    [3] = {int} 4
    [4] = {int} 5
```

Introduction to the interface



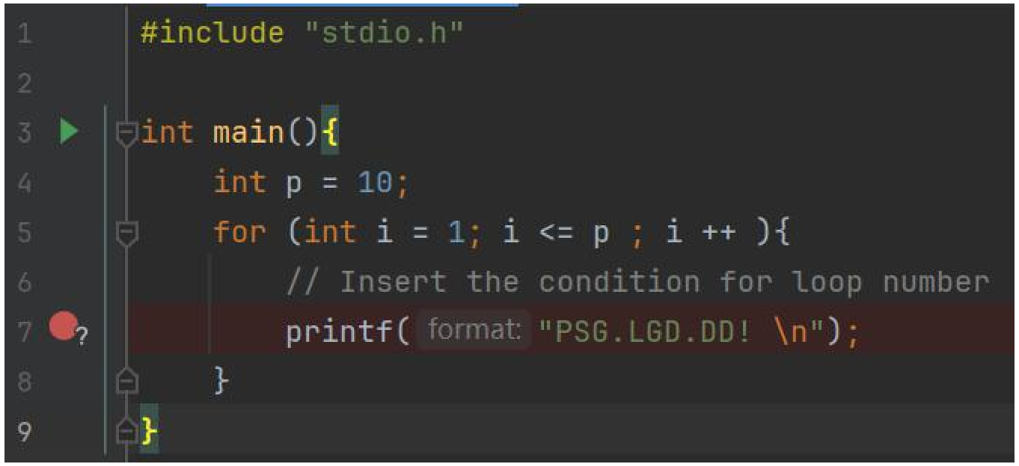
To continue debuggin after the break point
You can click **step over** or **step into**

Here some difference, step over means if you use some package from other .c it will not go into the package but step into means it will go to the package and debug the package

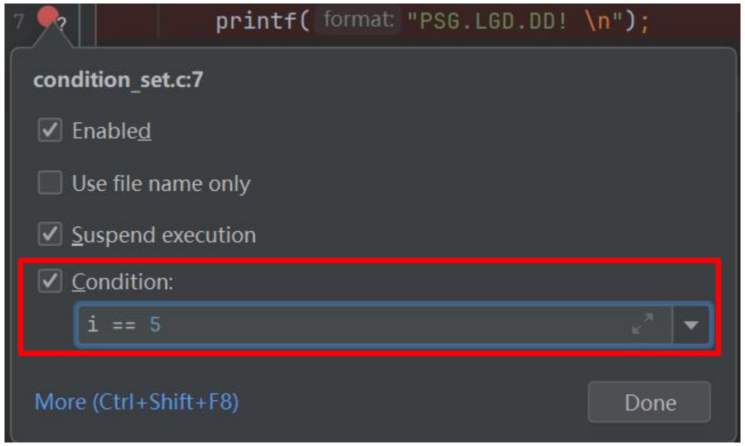
Process demonstration

To set break point condition

If you have loop and you want to make the break point work on several loops you can set the break point condition



```
#include "stdio.h"

int main(){
    int p = 10;
    for (int i = 1; i <= p ; i ++ ){
        // Insert the condition for loop number
        printf( format: "PSG.LGD.DD! \n");
    }
}
```



```
        printf( format: "PSG.LGD.DD! \n");
```

condition_set.c:7

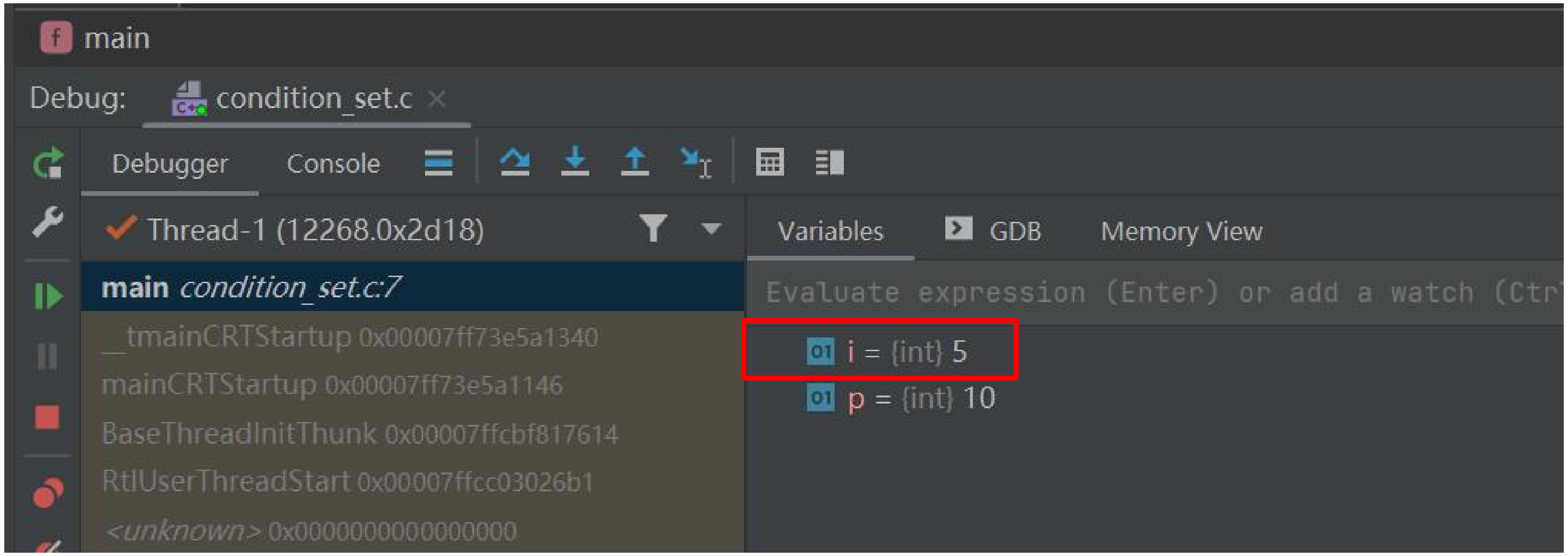
- [✓] Enabled
- [ ] Use file name only
- [✓] Suspend execution
- [✓] Condition:

```
i == 5
```

Right click the break point we set and choose the "condition", then type the condition "i == 5"

Process demonstration

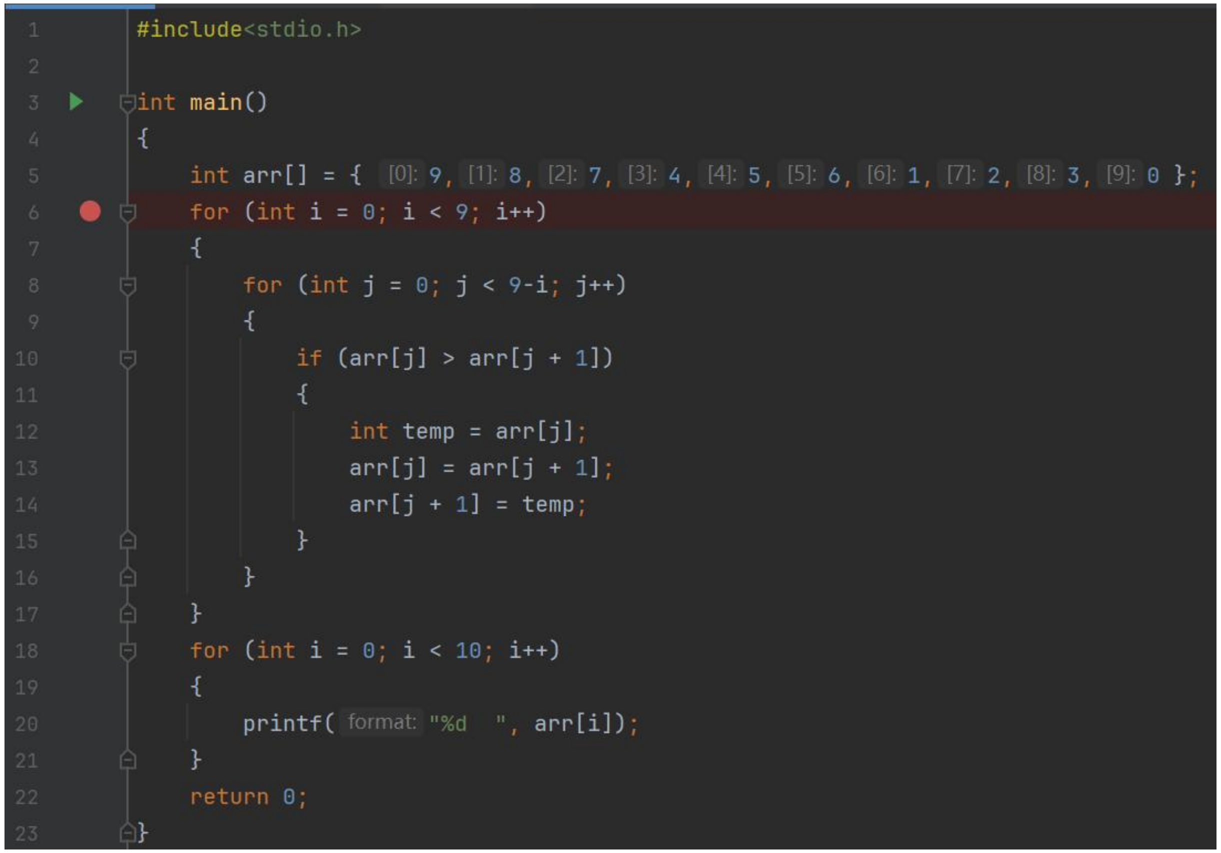
You can see the program pause at condition i = 5



```
Thread-1 (12268.0x2d18)
main condition_set.c:7
__tmainCRTStartup 0x00007ff73e5a1340
mainCRTStartup 0x00007ff73e5a1146
BaseThreadInitThunk 0x00007ffcbf817614
RtlUserThreadStart 0x00007ffcc03026b1
<unknown> 0x0000000000000000
```

```
Evaluate expression (Enter) or add a watch (Ctr
i = {int} 5
p = {int} 10
```

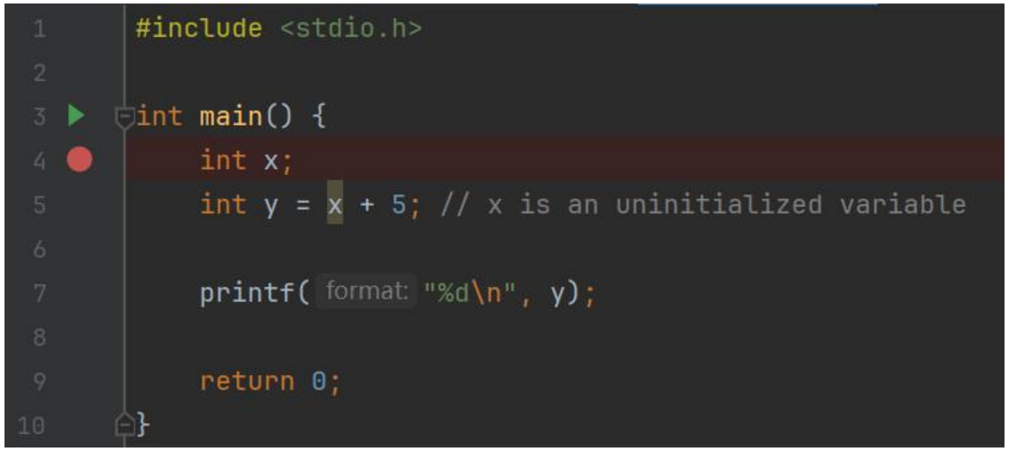
Demo 1: Sorting an array



```c
#include<stdio.h>

int main()
{
    int arr[] = { [0]: 9, [1]: 8, [2]: 7, [3]: 4, [4]: 5, [5]: 6, [6]: 1, [7]: 2, [8]: 3, [9]: 0 };
    for (int i = 0; i < 9; i++)
    {
        for (int j = 0; j < 9-i; j++)
        {
            if (arr[j] > arr[j + 1])
            {
                int temp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = temp;
            }
        }
    }
    for (int i = 0; i < 10; i++)
    {
        printf( format: "%d  ", arr[i]);
    }
    return 0;
}
```

In this code we implement an array from low to high, and try to use the debug function to see the change of different variables in each loop

Demo 2: Uninitialized variable



```
#include <stdio.h>

int main() {
    int x;
    int y = x + 5; // x is an uninitialized variable

    printf( format: "%d\n", y);

    return 0;
}
```

In this code, we do not assign a value to the integer x, and use x to add the value to the integer y, we will use the debug function to explore its output and why

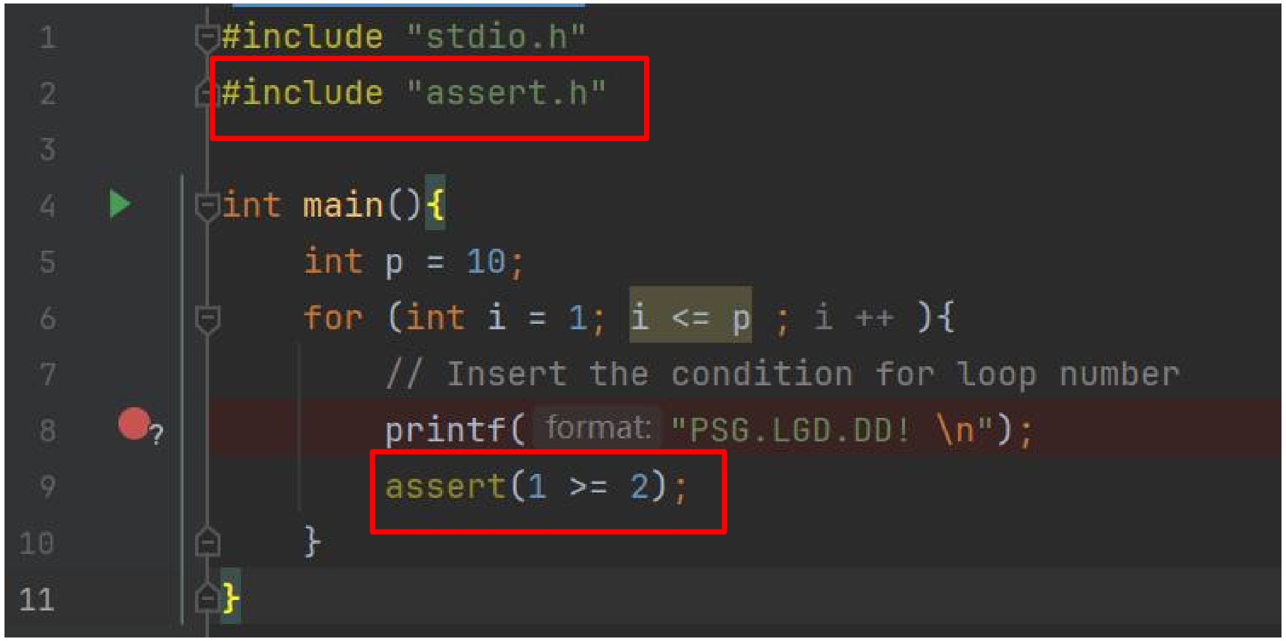
Other approaches

We also use Try except or ASSERT funtion to do the debug in the program in Python

In C you can also use assert function.

First you should use #include <assert.h>

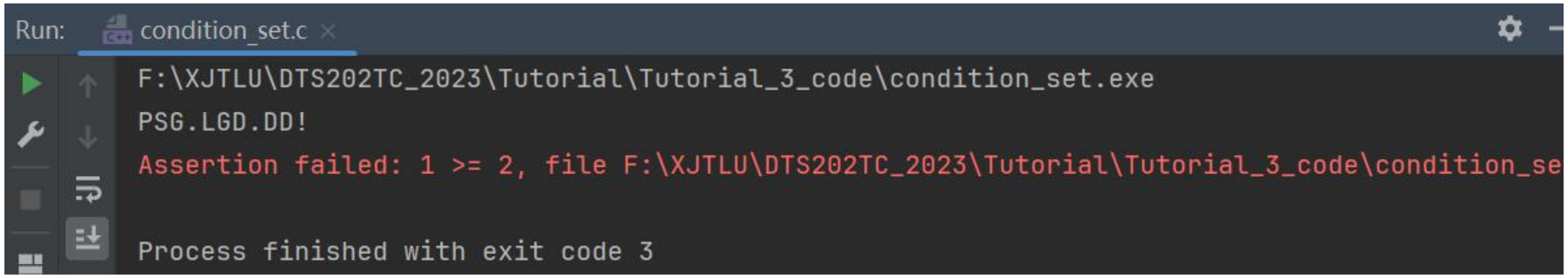
Then write assert:



```
#include "stdio.h"
#include "assert.h"

int main(){
    int p = 10;
    for (int i = 1; i <= p ; i ++ ){
        // Insert the condition for loop number
        printf( format: "PSG.LGD.DD! \n");
        assert(1 >= 2);
    }
}
```

Other approaches

Run the program if trigger assert
It will show like this


Run: condition_set.c

```
F:\XJTLU\DTS202TC_2023\Tutorial\Tutorial_3_code\condition_set.exe
PSG.LGD.DD!
Assertion failed: 1 >= 2, file F:\XJTLU\DTS202TC_2023\Tutorial\Tutorial_3_code\condition_se

Process finished with exit code 3
```

However I usually write this in case of read file fault or some unexcept problem
Usually I recommand use break point to do the debugging