

C/C++ Program Design

LAB 4

CONTENTS

- Learn to create and use pointers
- Learn to manage dynamic memory with **new** and **delete**

2 Knowledge Points

2.1 Pointers

2.2 Dynamic memory

2.1 Pointers

Pointer is a special type who holds the address of value.

```
#include <iostream>
using namespace std;

int main()
{
    int var1 = 3;
    float var2 = 24.8f;
    double var3 = 23.42;

    cout << &var1 << endl;
    cout << &var2 << endl;
    cout << &var3 << endl;

    return 0;
}
```

var1 is a variable, **&var1** gives its address.

Output:

```
0xffffcc1c
0xffffcc18
0xffffcc10
```

The 0x in the beginning represents the address in hexadecimal form.

```
#include <iostream>
using namespace std;
```

```
int main()
```

```
{
```

```
/* This pointer p can hold the address of an integer
 * variable, here p is a pointer and var is just a
 * simple integer variable.
 */
```

```
int *p, var;
```

Declare a pointer

```
/* This is how you assign the address the address of another
 * variable to the pointer.
 */
```

```
p = &var;
```

Assign the pointer

```
// This will print the address of variable var
cout << &var;
```

```
/* This will also print the address of variable var
 * because the pointer p holds the address of var
 */
```

```
cout << p;
```

Use the pointer to access the address

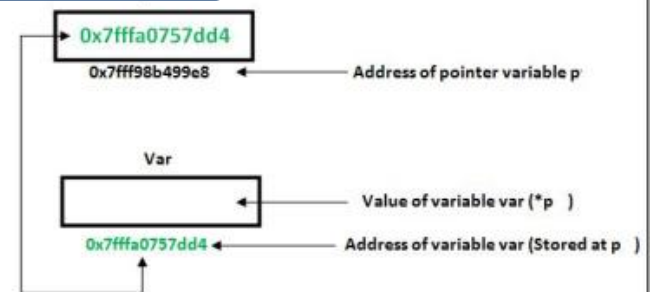
```
/* This will print the value of var. This is how we
 * access the value of variable through a pointer.
 */
```

```
cout << *p;
```

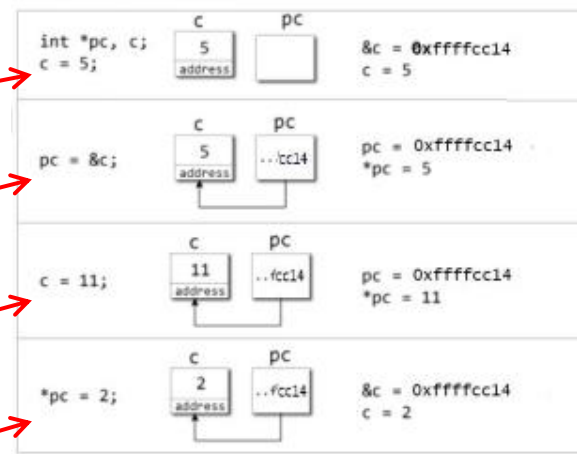
Use the ***pointer** to access the value

```
return 0;
```

```
}
```



```
pointer.cpp > ...
1  #include <iostream>
2  using namespace std;
3
4  int main()
5  {
6      int *pc, c;
7
8      c = 5;
9      cout << "Address of c (&c): " << &c << endl;
10     cout << "Value of c (c): " << c << endl << endl;
11
12     pc = &c;
13     cout << "Address that pointer pc holds (pc) " << pc << endl;
14     cout << "Content of the address pointer pc holds (*pc) " << *pc << endl << endl;
15
16     c = 11;
17     cout << "Address that pointer pc holds (pc) " << pc << endl;
18     cout << "Content of the address pointer pc holds (*pc) " << *pc << endl << endl;
19
20     *pc = 2;
21     cout << "Address of c (&c): " << &c << endl;
22     cout << "Value of c (c): " << c << endl << endl;
23
24     return 0;
25 }
```



```
Address of c (&c): 0xfffffcc14
Value of c (c): 5

Address that pointer pc holds (pc) 0xfffffcc14
Content of the address pointer pc holds (*pc) 5

Address that pointer pc holds (pc) 0xfffffcc14
Content of the address pointer pc holds (*pc) 11

Address of c (&c): 0xfffffcc14
Value of c (c): 2
```

Note:

1. **Reference operator(&)** gives the address of a variable.
2. **Dereference operator(*)** gets the value stored in the memory address.
3. The (*) sign used the declaration of pointer is not the dereference pointer. It is just a similar notation that creates a pointer.

Common mistakes when working with pointers:

```
int c, *pc;
```

```
pc = c;    //Wrong! pc is address whereas c is not an address.
```

```
*pc = &c; // Wrong! *pc is the value pointed by address whereas &c is an address.
```

```
pc = &c; // Correct! pc is an address and &c is also an address.
```

```
*pc = c; // Correct! *pc is the value pointed by address and c is also a value.
```

Various pointers :

```
double *pd;    // pointer to double
```

```
char **ppc;    // pointer to pointer to char
```

```
int *ap[15];    // array of 15 pointers to ints
```

```
int (*fp)(char *); // pointer to function taking a char* argument, returns an int
```

```
int *f(char *);    // function taking a char* argument, returns a pointer to int
```

Pointers to structure

```
pointer_structure.cpp > ...
1  #include <iostream>
2  using namespace std;
3
4  struct Distance
5  {
6      int feet;
7      float inch;
8  };
9
10 int main()
11 {
12     Distance d, *ptr;
13     ptr = &d;
14     cout << "Enter feet: ";
15     cin >> (*ptr).feet;
16     cout << "Enter inch: ";
17     cin >> ptr->inch;
18
19     cout << "Displaying information:" << endl;
20     cout << "Distance = " << (*ptr).feet << " feet " << ptr->inch << " inches." << endl;
21
22     return 0;
23 }
24
```

Creates a pointer **ptr** of type structure Distance.

ptr must point to the Distance variable.

These two ways can both access the members of structure, but **-> notation** is more common.

Sample output:

```
Enter feet: 4
Enter inch: 3.5
Displaying information:
Distance = 4 feet 3.5 inches.
```

Note: Since pointer **ptr** is pointed to variable **d** in this program, **(*ptr).inch**, **ptr->inch** and **d.inch** are exact the same.

Pointers and array

```
#include <iostream>
using namespace std;
```

```
int main()
```

```
{
```

```
    float arr[5];
    float* ptr;
```

```
    cout << "Displaying address using array: " << endl;
    for (int i = 0; i < 5; i++)
        cout << "&arr[" << i << "] = " << &arr[i] << endl;
```

```
    ptr = arr;
```

```
    cout << "\nDisplaying address using pointer:" << endl;
    for (int i = 0; i < 5; i++)
        cout << "ptr + " << i << " = " << ptr + i << endl;
```

```
    for (int i = 0; i < 5; i++)
        arr[i] = i * 2;
```

```
    cout << "\nDisplaying values of elements using pointer:" << endl;
    for (int i = 0; i < 5; i++)
        cout << "*(ptr + " << i << ") = " << *(ptr + i) << endl;
```

```
    return 0;
```

```
}
```

Access the address of each element by array.

ptr points to the array.

Access the address of each element by pointer.

Access the values of elements by pointer using * operator.

Displaying address using array:

```
&arr[0] = 006FFCD0
&arr[1] = 006FFCD4
&arr[2] = 006FFCD8
&arr[3] = 006FFCDC
&arr[4] = 006FFCE0
```

Displaying address using pointer:

```
ptr + 0 = 006FFCD0
ptr + 1 = 006FFCD4
ptr + 2 = 006FFCD8
ptr + 3 = 006FFCDC
ptr + 4 = 006FFCE0
```

Displaying values of elements using pointer:

```
*(ptr + 0) = 0
*(ptr + 1) = 2
*(ptr + 2) = 4
*(ptr + 3) = 6
*(ptr + 4) = 8
```

2.2 Dynamic Memory

2.2.1 C Dynamic Memory

These functions can be found in the **<stdlib.h>** header file.

Sr.No.	Function	Description
1	void *calloc(int num, int size);	This function allocates an array of num elements each of which size in bytes will be size .
2	void free(void *address);	This function releases a block of memory block specified by address.
3	void *malloc(int num);	This function allocates an array of num bytes and leave them uninitialized.
4	void *realloc(void *address, int newsize);	This function re-allocates memory extending it upto newsize .

When you are not in need of memory any more, you should release that memory by calling the function **free()**.

1. Allocating Memory Dynamically

When you declare an array, you must specify the number of the elements. Sometimes you don't know the amount of the elements, you can declare a pointer, and let it point to the memory which allocated dynamically.

```
C allocateMemory.c > ...
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4
5  int main()
6  {
7      char name[100];
8      char *description;
9
10     strcpy(name, "Zara Ali");
11
12     /* allocate memory dynamically */
13     description = (char *)malloc(200 * sizeof(char));
14     if(description == NULL)
15     {
16         fprintf(stderr, "Error- unable to allocate required memory.\n");
17     }
18     else
19     {
20         strcpy(description, "Zara Ali is a DPS student in class 10.");
21     }
22     printf("Name = %s\n", name);
23     printf("Description: %s\n", description);
24
25     free(description);
26
27     return 0;
28 }
```

Declare an array with 100 elements.

Declare a pointer.

Let the pointer point to the memory.

You can use **calloc(200, sizeof(char))** to replace malloc function.

Copy a string to the memory.

Release the memory.

Sample output:

```
Name = Zara Ali
Description: Zara Ali is a DPS student in class 10.
```

2. Resizing Memory

You can increase or decrease the size of an allocated memory block by calling the function `realloc()`.

```
C reallocateMemory.c > ...
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4
5  int main()
6  {
7      char name[100];
8      char *description;
9
10     strcpy(name, "Zara Ali");
11
12     /* allocate memory dynamically */
13     description = (char *)malloc(30 * sizeof(char));
14     if(description == NULL)
15         fprintf(stderr, "Error- unable to allocate required memory.\n");
16     else
17         strcpy(description, "Zara Ali is a DPS student.");
18
19     /* suppose you want to store a bigger description */
20     description = (char *)realloc(description, 100 * sizeof(char));
21     if(description == NULL)
22         fprintf(stderr, "Error- unable to allocate required memory.\n");
23     else
24         strcat(description, "She is in class 10.");
25
26     printf("Name = %s\n", name);
27     printf("Description: %s\n", description);
28
29     free(description);
30     return 0;
31 }
```

Resizing the memory.

Concatenate the string.

Release the memory.

Sample output:

```
Name = Zara Ali
Description: Zara Ali is a DPS student.She is in class 10.
```

2.2.2 C++ Dynamic Memory

1. **new** and **delete** Operators

new data-type;

Use **new** operator to allocate memory dynamically for any data-type.

data-type could be any built-in data type including an array or any user defined data types such as structure or class.

delete pointer variable;

Use **delete** operator to de-allocate memory that was previously allocated by new operator.

```
G+ newdouble.cpp > ...
1  #include <iostream>
2  using namespace std;
3
4  int main() {
5      double *pvalue = NULL;    //Pointer initial with null
6      pvalue = new double;      // Request memory for the variable
7
8      *pvalue = 1294948.98;      // Store value at all allocated address
9
10     cout << "Value of pvalue: " << *pvalue << endl;
11
12     delete pvalue;             // Free up the memory
13
14     return 0;
15 }
```

Sample out:

```
Value of pvalue: 1.29495e+06
```

2. Dynamic Memory Allocation for Arrays

```
char* pvalue = NULL;           // Pointer initialized with null  
pvalue = new char[20];         // Request memory for the variable
```

```
delete [] pvalue;              // Delete array pointed to by pvalue
```

```
double(*pvalue)[4] = NULL;     // Pointer initialized with null  
pvalue = new double [3][4];    // Allocate memory for a 3x4 array
```

```
delete [] pvalue;              // Delete array pointed to by pvalue
```

OR

```
int **p  
p = new int*[3];  
for(int i = 0; i < 3; i++)  
    p[i] = new int[4];
```

```
for(int i=0; i<3; i++)  
    delete [] p[i];  
delete [] p;
```

newarray.cpp > ...

```
1  #include <iostream>
2  using namespace std;
3
4  int main()
5  {
6      int * pArray = NULL ,*t ;
7      pArray = new int [10] ;
8      if ( pArray == NULL )
9      { cout << "allocation failure.\n" ;
10         exit(0) ;
11     }
12     for ( int i = 0 ; i < 10 ; i ++ )
13         pArray[i] = 100 + i ;
14
15     cout << "Displaying the Array Content" << endl;
16     for ( t = pArray ; t < pArray + 10 ; t ++ )
17         cout << *t << " " ;
18
19     delete [] pArray ;
20
21     return 0;
22 }
23
```

Allocate the memory to store 10 integers, and assign its address to the pointer **pArray**.

Assign 10 values to the memory by the pointer **pArray**.

If you access the value by * operator, be sure do not move the pointer which assign the address by new.

Release the memory.

Sample out:

```
Displaying the Array Content
100 101 102 103 104 105 106 107 108 109
```

```
#include <iostream>
using namespace std;
```

```
int main()
{
    int* pArray = NULL;
    pArray = new int[10];

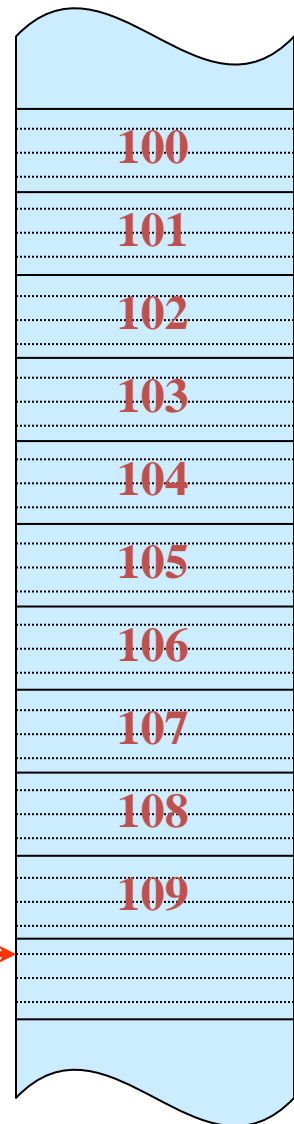
    if (pArray == NULL)
    {
        cout << "Allocation failure.\n";
        exit(0);
    }

    for (int i = 0; i < 10; i++)
        pArray[i] = 100 + i;

    cout << " Displaying the Array countents:" << endl;
    for (int i = 0; i < 10; i++, pArray++)
        cout << *pArray << " ";

    delete[] pArray;

    return 0;
}
```




After **for loop**, the pointer is now pointed to the memory out of the range you have requested.

The output in Visual studio 2019

```
C:\> D:\csourcecode\2021Spring\lab04\exampleCode\visualstudio\memoryleak\Debug\memoryleak.exe
```

Displaying the Array countents:
100 101 102 103 104 105 106 107 108 109

Microsoft Visual C++ Runtime Library (未响应)

 Debug Assertion Failed!

Program:
...ab04\exampleCode\visualstudio\memoryleak\Debug\memoryleak.exe
File: minkernel\crt\ucrt\src\appcrt\heap\debug_heap.cpp
Line: 904

Expression: _CrtIsValidHeapPointer(block)

For information on how your program can cause an assertion failure, see the Visual C++ documentation on asserts.

(Press Retry to debug the application)

中止(A) 重试(R) 忽略(I)

memoryleak.cpp > ...

```
1  #include <iostream>
2  using namespace std;
3
4  int main()
5  {
6      int *pArray = NULL;
7      pArray = new int[10];
8
9      if(pArray == NULL)
10     {
11         cout << "Allocateion failure.\n";
12         exit(0);
13     }
14
15     for(int i = 0; i < 10; i++)
16         pArray[i] = 100 + i;
17
18     cout << " Displaying the Array countents:" << endl;
19     for(int i = 0; i < 10; i++, pArray++)
20         cout << *pArray << " ";
21
22     delete [] pArray;
23
24     return 0;
25
26 }
27
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

maydlee@LAPTOP-U1M00N2F:/mnt/d/csourcecode/2021Spring/Lab04/exampleCode\$ g++ memoryleak.cpp

maydlee@LAPTOP-U1M00N2F:/mnt/d/csourcecode\$./memoryleak

Displaying the Array countents:

Segmentation fault (core dumped)

maydlee@LAPTOP-U1M00N2F:/mnt/d/csourcecode\$

There is no output in VScode, just a message of "Segmentation fault (core dumped)"

```

#include <iostream>
using namespace std;

int main()
{
    int* pArray = NULL;
    pArray = new int[10];

    if (pArray == NULL)
    {
        cout << "Allocateion failure.\n";
        exit(0);
    }

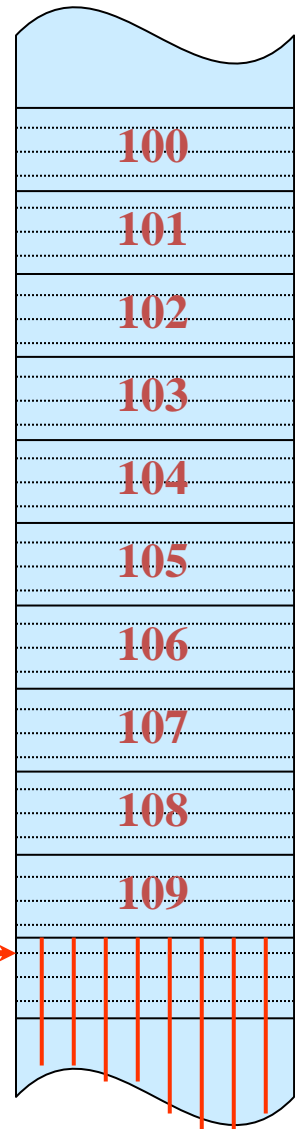
    for (int i = 0; i < 10; i++)
        pArray[i] = 100 + i;

    cout << " Displaying the Array countents:" << endl;
    for (int i = 0; i < 10; i++, pArray++)
        cout << *pArray << " ";

    delete[] pArray;

    return 0;
}

```



The memory you release will not what you requested.

```
#include <iostream>
using namespace std;
```

```
int main()
{
    int* pArray = NULL;
    pArray = new int[10];

    if (pArray == NULL)
    {
        cout << "Allocatemon failure.\n";
        exit(0);
    }

    for (int i = 0; i < 10; i++)
        pArray[i] = 100 + i;

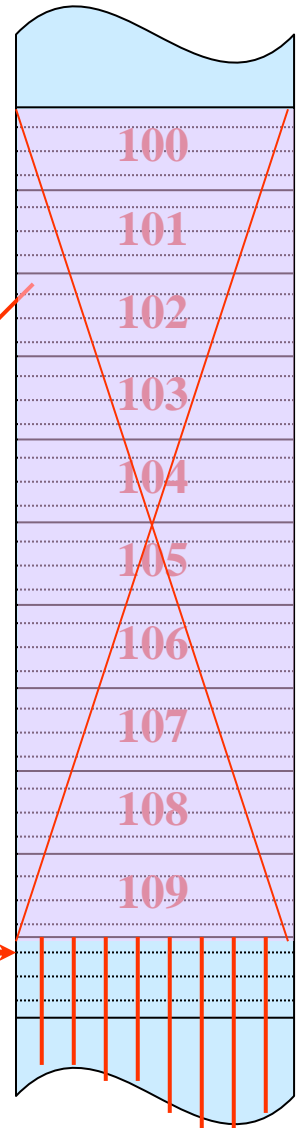
    cout << " Displaying the Array countents:" << endl;
    for (int i = 0; i < 10; i++, pArray++)
        cout << *pArray << " ";

    delete[] pArray;

    return 0;
}
```

memory leak
内存泄漏

pArray



Many times, you are not aware in advance how much memory you will need to store particular information in a defined variable, but the size of required memory can be determined at run time.

```
dynamic_array.cpp > main()
1  #include <iostream>
2  using namespace std;
3
4  int main()
5  {
6      int n;
7      cout << "How many classes did you take in last semester?";
8      cin >> n;
9
10     float *pScore = new float[n];
11     float *pt = pScore;
12
13     cout << "Input " << n << " scores:";
14     for(; pt < pScore + n; pt++)
15         cin >> *pt;
16
17     cout << "The scores are:\n";
18     pt = pt - n;
19     for(; pt < pScore + n; pt++)
20         cout << *pt << "\t";
21     cout << "\n";
22
23     delete []pScore;
24
25     return 0;
26
27 }
```

Sample out:

```
How many classes did you take in last semester?5
Input 5 scores:82 90 78.5 85.5 83
The scores are:
82      90      78.5      85.5      83
```

3. Dynamic Memory Allocation for Structures

```
newstructure.cpp > ...
1  #include <iostream>
2  struct inflatable // structure declaration
3  {
4      char name[20];
5      float volume;
6      double price;
7  };
8
9  int main()
10 {
11     using namespace std;
12     inflatable *ps = new inflatable; // allocate memory for structure
13
14     cout << "Enter name of inflatable item: ";
15     cin.get(ps->name, 20); // use -> to access the member
16     cout << "Enter volume of cubic feet: ";
17     cin >> (*ps).volume; // use (*) to access the member
18     cout << "Enter price: $";
19     cin >> ps->price;
20
21     cout << "Name: " << (*ps).name << endl;
22     cout << "Volume: " << ps->volume << "cubic feet\n";
23     cout << "Price: $" << ps->price << endl;
24
25     delete ps; // free memory used by structure
26
27     return 0;
28 }
```

Create an unnamed structure of the inflatable type and assign its address to **ps** pointer using **new** operator

Access the structure members using **->** or **(*)**.

Release the memory.

Sample output:

```
Enter name of inflatable item: Black Base
Enter volume of cubic feet: 35.4
Enter price: $91.25
Name: Black Base
Volume: 35.4cubic feet
Price: $91.25
```

Structured array

```
newstructurearray.cpp > ...
1  #include <iostream>
2  #include <new>
3  using namespace std;
4
5  struct Employee
6  {
7      string Name;
8      int Age;
9  };
10
11 int main()
12 {
13     Employee *DynArray;
14     DynArray = new (nothrow) Employee[3];
15
16     DynArray[0].Name = "Harvey ";
17     DynArray[0].Age = 33;
18     DynArray[1].Name = "Sally";
19     DynArray[1].Age = 26;
20     DynArray[2].Name = "Jeff";
21     DynArray[2].Age = 52;
22
23     cout << "Displaying the Array Contents" <<
24     for(int i = 0; i < 3; i++)
25         cout << "Name: " << DynArray[i].Name << "\tAge: " << DynArray[i].Age << endl;
26
27     delete [] DynArray;
28
29     return 0;
30 }
```

Create an unnamed structured array of the Employee type and assign its address to **DynArray** pointer using new operator

nothrow constant, this constant value is used as an argument for [operator new] and [operator new[]] to indicate that these functions shall not throw an exception on failure, but return a *null pointer* instead.

Release the memory.

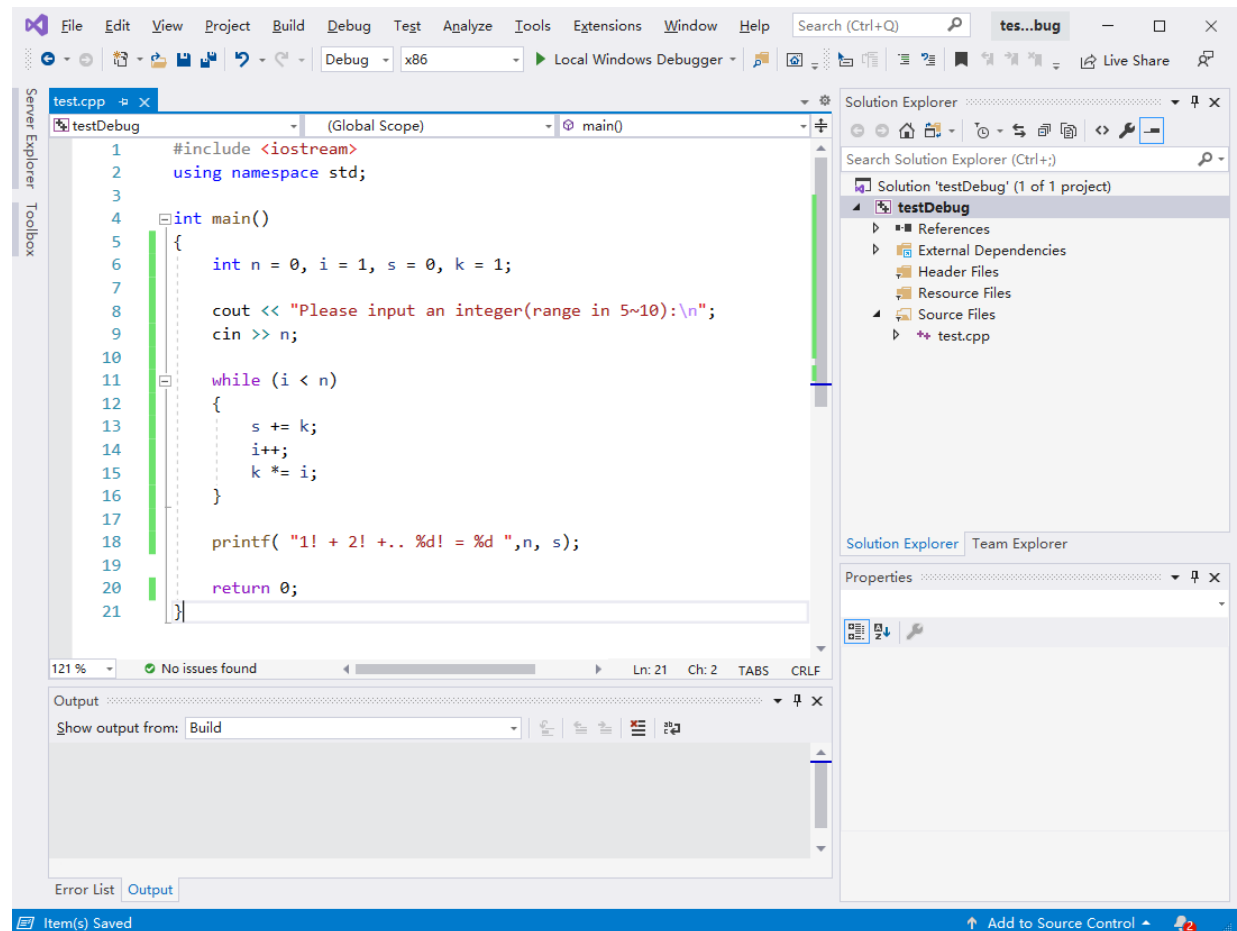
Sample output:

```
Displaying the Array Contents
Name: Harvey      Age: 33
Name: Sally       Age: 26
Name: Jeff        Age: 52
```

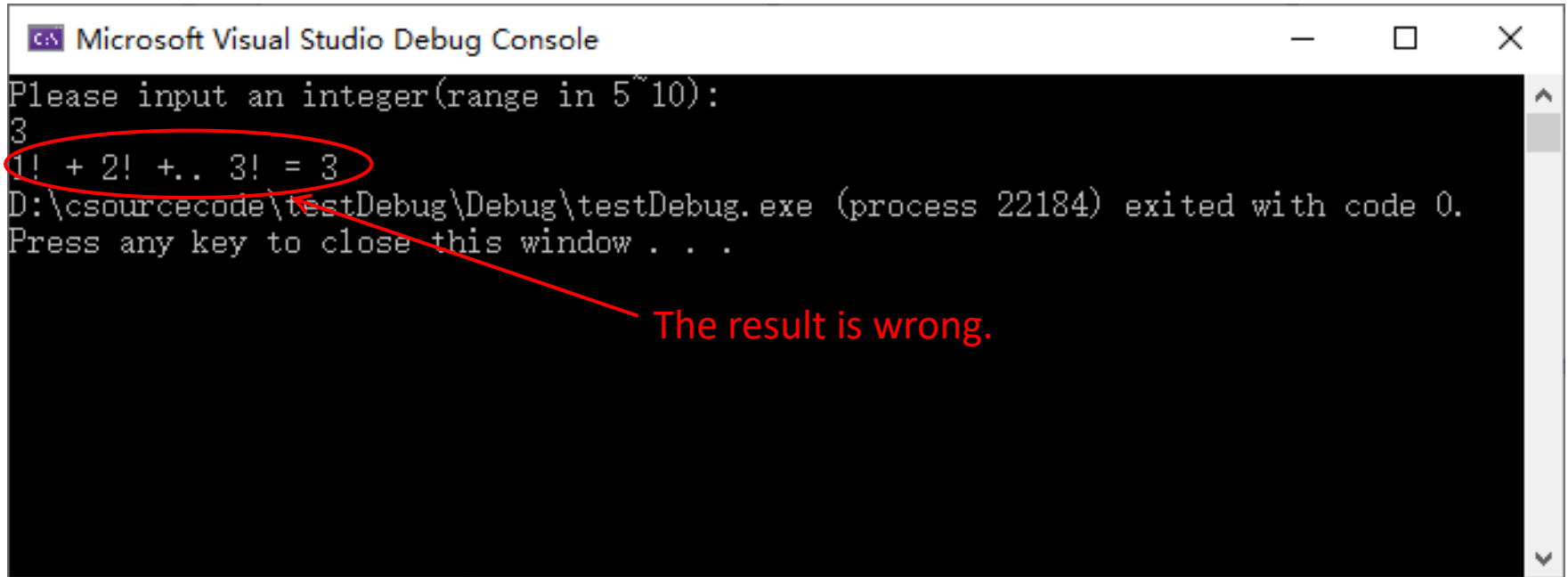
2.3 Debug C++ program

The act of debugging helps you to track “What went wrong logically?” in the program code.

1. Type a C++ program in VS 2019



2. Run the program in VS 2019 and check whether the result is right.



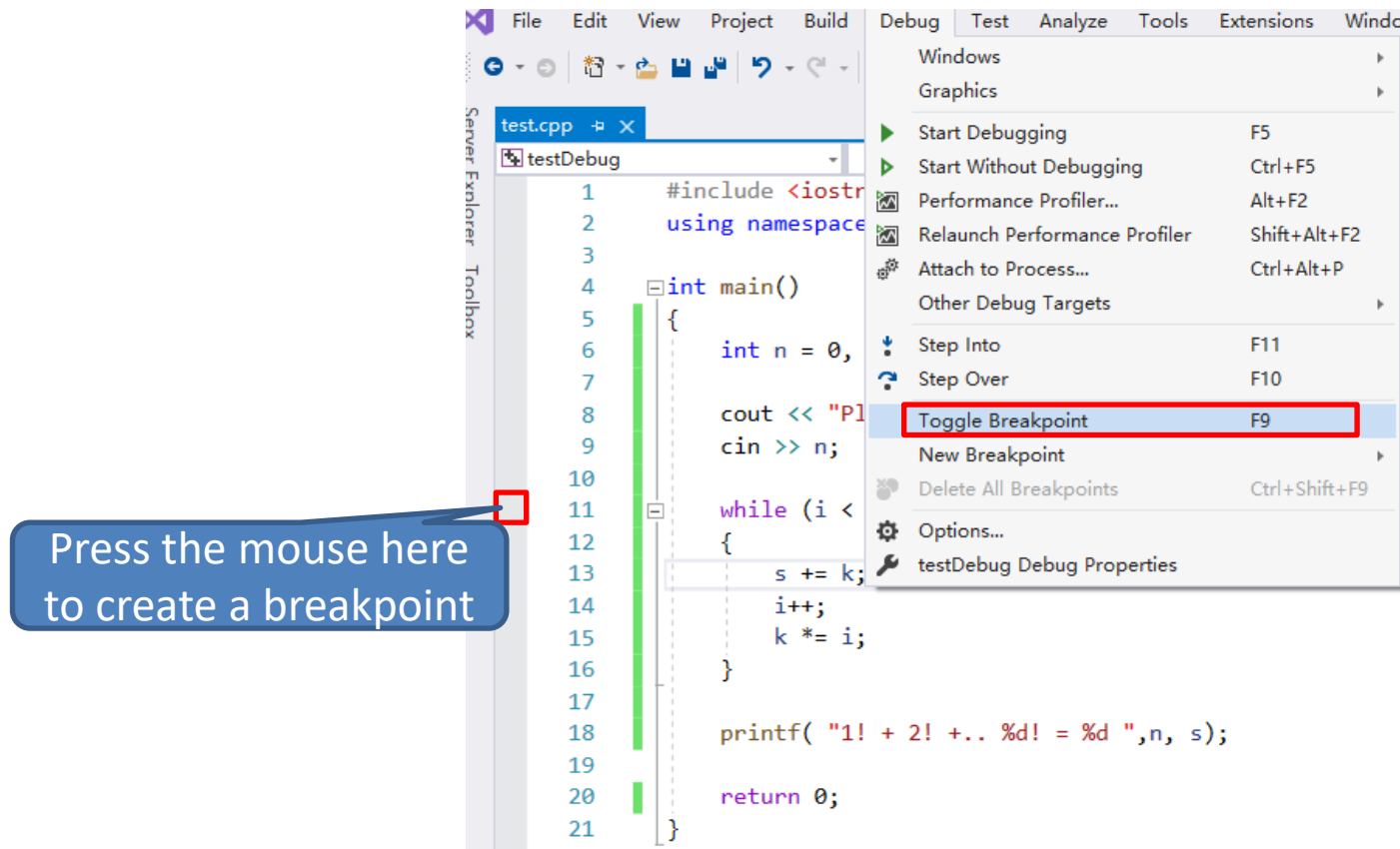
```
Microsoft Visual Studio Debug Console
Please input an integer(range in 5~10):
3
1! + 2! +.. 3! = 3
D:\csourcecode\testDebug\Debug\testDebug.exe (process 22184) exited with code 0.
Press any key to close this window . . .
```

The result is wrong.

You may start your debugging to find the logic errors.

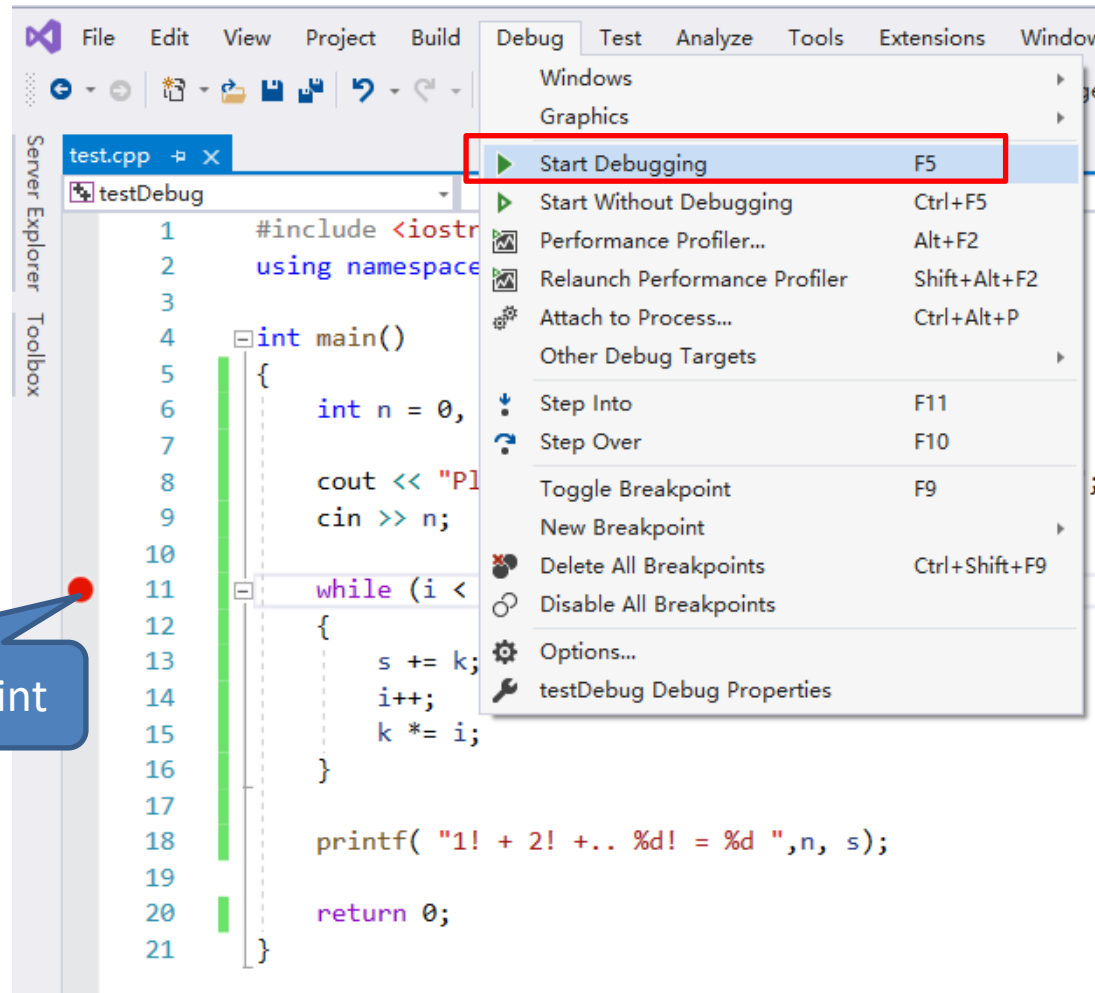
3. Add a breakpoint

- (1) Place the cursor at start of code where breakpoint needs to be placed
- (2) Go to Debug-> Toggle Breakpoint(shortcut:F9)
or press left button of mouse at the left grey edge



A red point will appear at the location, this red point is the breakpoint.

4. Run your program in debugging way
Go to Debug-> Start Debugging (shortcut:F5)



After you input an integer, the running stops at the breakpoint.

The screenshot shows the Visual Studio IDE with a C++ project named 'testDebug'. The code in 'test.cpp' is as follows:

```
1  #include <iostream>
2  using namespace std;
3
4  int main()
5  {
6      int n = 0, i = 1, s = 0, k = 1;
7
8      cout << "Please input an integer(range in 5~10):\n";
9      cin >> n;
10
11     while (i < n)
12     {
13         // ...
14     }
15
16     printf( "1! + 2! +.. %d! = %d ",n, s);
17
18     return 0;
19 }
```

A breakpoint is set at line 11, which is highlighted with a red box. A blue callout box points to the breakpoint with the text "The running stops here".

The right sidebar shows the 'Diagnostics Tools' window with the following data:

- Diagnostics session: 3 seconds (3.731 s sele...)
- Events: 10s
- Process Memory (KB): 915
- CPU (% of all processors): 100

The bottom left shows the 'Autos' window with the following data:

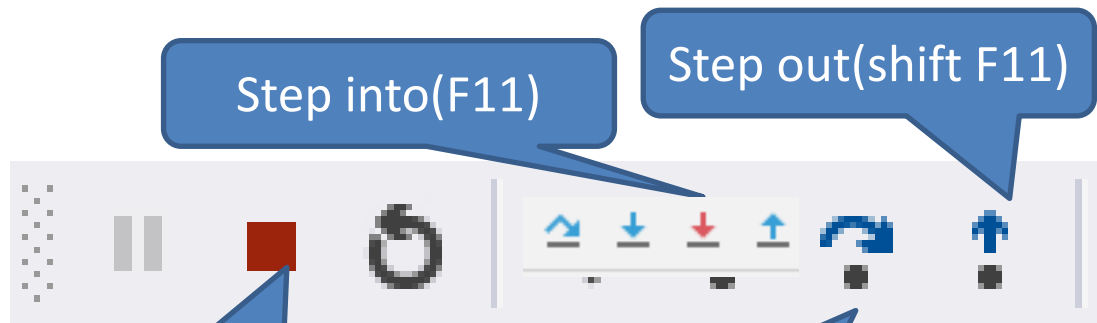
Name	Value	Type
i	1	int
n	3	int

The bottom right shows the 'Call Stack' window with the following data:

Name	Lang
testDebug.exe!main() Line 11	C++
[External Code]	
kernel32.dll!...	Un...

The status bar at the bottom shows 'Ready' and 'Add to Source Control'.

- **Step into(F11):** run step by step and trace into the function
- **Step over(F10):** run step by step and not trace into the function
- **Step out(shift F11):** jump out of the function



Stop debugging(shift F5)

5. Press “step over” button or F10, to run the program

Step over(F10)

The yellow arrow represents the line that will be run next

Watch the value of auto variables

test.cpp

```
1 #include <iostream>
2 using namespace std;
3
4 int main()
5 {
6     int n = 0, i = 1, s = 0, k = 1;
7
8     cout << "Please input an integer(range in 5~10):\n";
9     cin >> n;
10
11     while (i < n)
12     {
13         s += k;
14         i++;
15         k *= i;
16     }
17 }
```

Process: [25228] testDebug.exe Thread: [12432] Main Thread

Events

Process Memory (KB)

CPU (% of all processors)

Summary Events Memory Usage CPU Usage

Events

Memory Usage

CPU Usage

Autos

Name	Value	Type
i	1	int
k	1	int
n	3	int
s	0	int

Call Stack

testDebug.exe!main() Line 13

[External Code]

kernel32.dll![]

tes...bug

File Edit View Project Build Debug Test Analyze Tools Extensions Window Help Search (Ctrl+Q)

Debug x86 Continue

Process: [25228] testDebug.exe Lifecycle Events Thread: [12432] Main Thread

test.cpp

testDebug (Global Scope) main()

```
1 #include <iostream>
2 using namespace std;
3
4 int main()
5 {
6     int n = 0, i = 1, s = 0, k = 1;
7
8     cout << "Please input an integer(range in 5~10):\n";
9     cin >> n;
10
11     while (i < n)
12     {
13         s += k;
14         i++;
15         k *= i;
16     }
17
18     printf( "1! + 2! +.. %d! = %d ",n, s);
19
20     return 0;
21 }
```

121 % No issues found Ln: 13 Ch: 1 TABS CRLF

Diagnostic Tools

Diagnostics session: 3 seconds (3.731 s sele... 3.73s

Events

Process Memory (KB) 915 915

CPU (% of all processors) 100 100

Summary Events Memory Usage CPU Usage

Events

Show Events (2 of 2)

Memory Usage

Take Snapshot

Enable heap profiling (affects performanc

CPU Usage

Locals

Search (Ctrl+E) Search Depth: 3

Name	Value	Type
i	1	int
k	1	int
n	3	int
s	0	int

Autos Locals Watch 1

Call Stack

Name	Lang
testDebug.exe!main() Line 13	C++
[External Code]	
kernel32.dll![]	Un...

on Setti... Command Win... Immediate Wi... Output

Ready Add to Source Control

Watch the value of auto variables

tes...bug

File Edit View Project Build Debug Test Analyze Tools Extensions Window Help Search (Ctrl+Q)

Process: [25228] testDebug.exe Lifecycle Events Thread: [12432] Main Thread

test.cpp

testDebug (Global Scope) main()

```
1 #include <iostream>
2 using namespace std;
3
4 int main()
5 {
6     int n = 0, i = 1, s = 0, k = 1;
7
8     cout << "Please input an integer(range in 5~10):\n";
9     cin >> n;
10
11     while (i < n)
12     {
13         s += k; ≤ 1ms elapsed
14         i++;
15         k *= i;
16     }
17
18     printf( "1! + 2! +.. %d! = %d ",n, s);
19
20     return 0;
21 }
```

121 % No issues found Ln: 13 Ch: 1 TABS CRLF

Diagnostic Tools

Diagnostics session: 3 seconds (3.731 s sele... 3.73s

Events

Process Memory (KB) 915 915

CPU (% of all processors) 100 100

Summary Events Memory Usage CPU Usage

Events

Show Events (2 of 2)

Memory Usage

Take Snapshot

Enable heap profiling (affects performanc

CPU Usage

Watch 1

Search (Ctrl+E) Search Depth: 3

Name	Value	Type
Add item to watch		

Call Stack

Name	Lang
testDebug.exe!main() Line 13	C++
[External Code]	
kernel32.dll![]	Un...

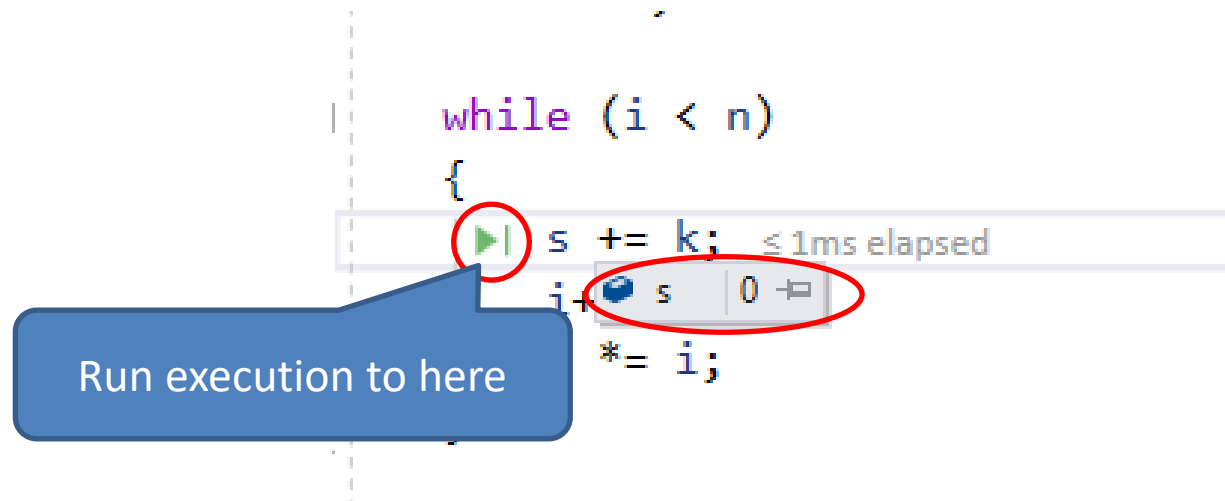
Call Stack Breakpoints Exception Setti... Command Win... Immediate Wi... Output

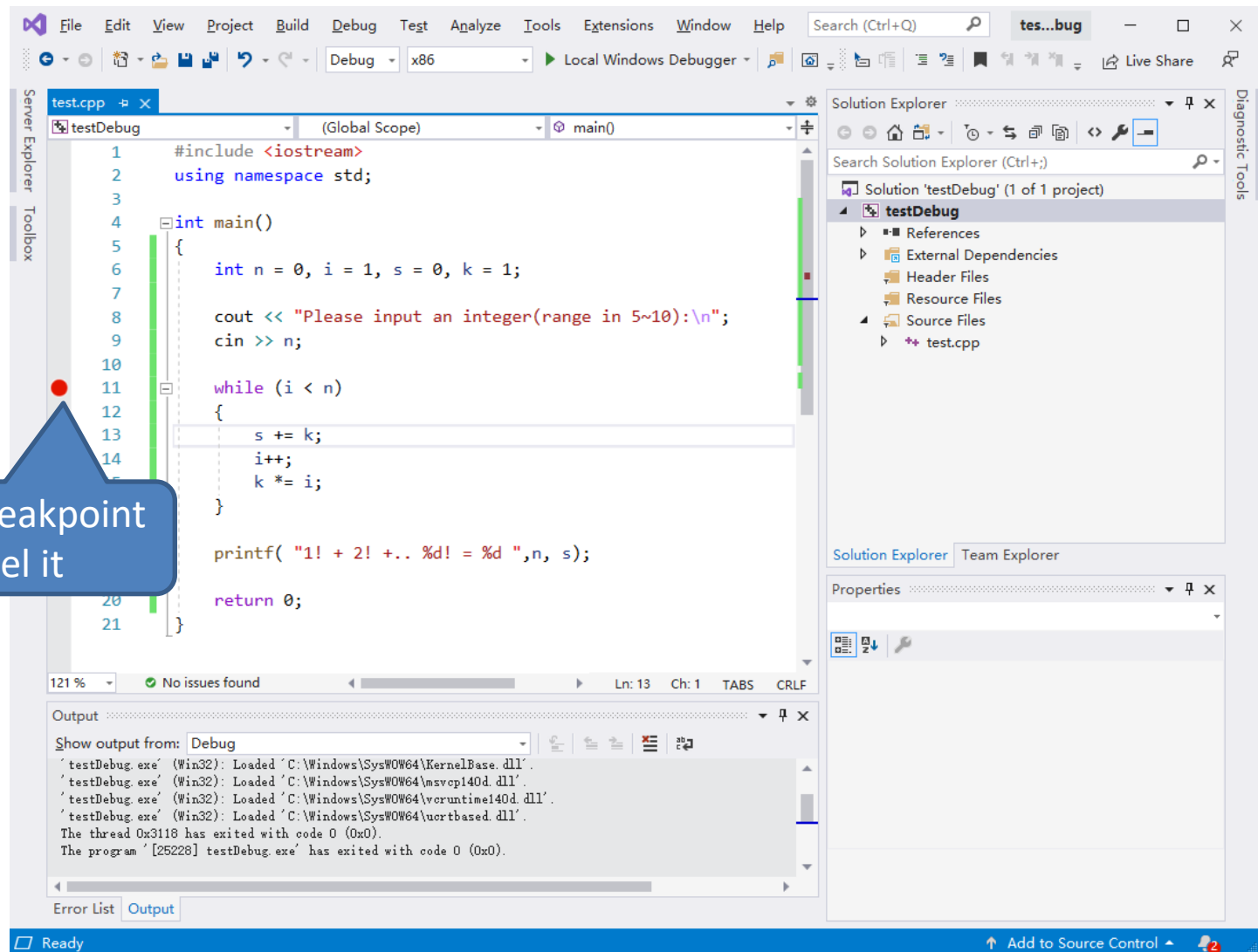

Ready Add to Source Control

You can input variable name which you want to watch in "Watch" window

When the cursor stays at a certain line, the value of the variable at this line is shown.

Press the green button , the program will run at the current line that the green indicates.



You can find the logic errors by watching the value of variables and checking the flow of the program. At last, press “stop debugging 

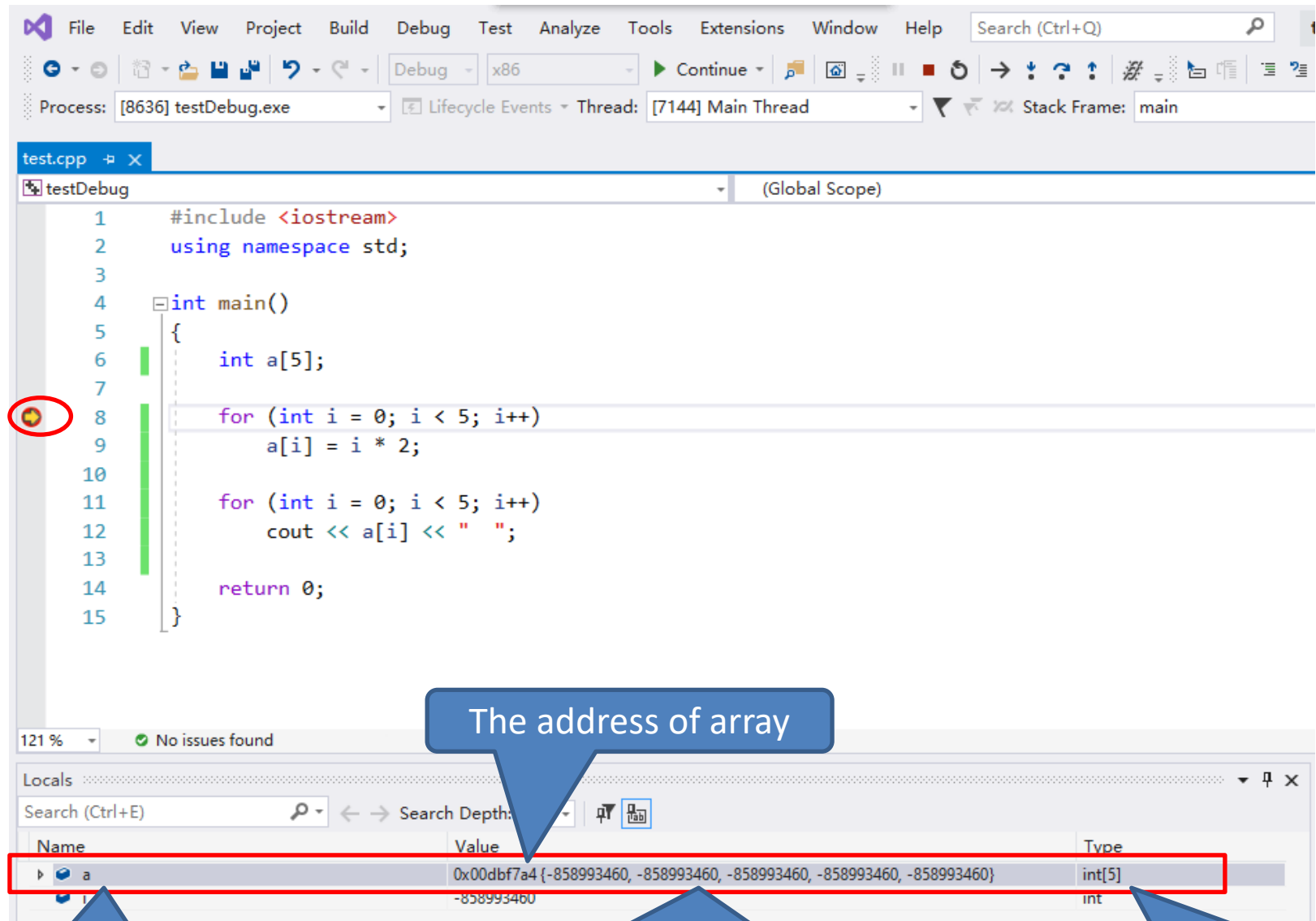
The screenshot shows the Visual Studio IDE with a C++ project named 'testDebug'. The code in 'test.cpp' is as follows:

```
1  #include <iostream>
2  using namespace std;
3
4  int main()
5  {
6      int n = 0, i = 1, s = 0, k = 1;
7
8      cout << "Please input an integer(range in 5~10):\n";
9      cin >> n;
10
11     while (i < n)
12     {
13         s += k;
14         i++;
15         k *= i;
16     }
17
18     printf( "1! + 2! +.. %d! = %d ", n, s);
19
20     return 0;
21 }
```

A red dot breakpoint is set on line 11. A blue callout bubble with the text "Press the breakpoint to cancel it" points to this breakpoint. The Solution Explorer on the right shows the project structure, and the Output window at the bottom shows the program's execution logs.

How to check the value of an array?

Set a breakpoint and Start Debugging (shortcut:F5)



Visual Studio interface showing a C++ program being debugged. The code defines an array `a` of size 5 and fills it with values `0, 2, 4, 6, 8` using a `for` loop. The debugger is paused at line 11, and the `Locals` window shows the state of the variables.

```
1  #include <iostream>
2  using namespace std;
3
4  int main()
5  {
6      int a[5];
7
8      for (int i = 0; i < 5; i++)
9          a[i] = i * 2;
10
11     for (int i = 0; i < 5; i++) ≤ 1ms elapsed
12         cout << a[i] << " ";
13
14     return 0;
15 }
```

The `Locals` window displays the following variables:

Name	Value	Type
a	0x00dbf7a4 {0, 2, 4, 6, 8}	int[5]
i	-858993460	int

After loop, the values of elements in array are correct

How to check the value of a structure?

Set a breakpoint and Start Debugging (shortcut:F5)

The screenshot shows the Visual Studio IDE with a C++ file named `test.cpp` open. The code defines a `Student` struct with members `id`, `name`, `age`, and `score`. The `main` function creates an array `s` of two `Student` objects and prints their details. A breakpoint is set at line 12, which is circled in red. The `testDebug` window is active, showing the program has stopped at the breakpoint. The `Locals` window is open, displaying the array `s` and its elements. The `Call Stack` window shows the current frame is `testDebug.exe!main() Line 12`.

```
1  #include <iostream>
2  using namespace std;
3
4  struct Student {
5      int id;
6      string name;
7      int age;
8      double score[3];
9  };
10
11 int main()
12 {
13     Student s[2] = { {102, "Tom", 18, 78.5, 89, 92.5},
14                     {104, "Peter", 19, 67, 75, 80.5} };
15
16     for (int i = 0; i < 2; i++)
17         cout << s[i].id << " " << s[i].name << " " << s[i].age << s[i].score[0] << " " << s[i].score[1] << " " << s[i].score[2] << endl;
18
19     return 0;
20 }
```

Name	Value	Type
s	0x004ff728 ({id=-858993460 name=<Error reading characters of string.> age=-8589...})	Student[2]
[0]	{id=-858993460 name=<Error reading characters of string.> age=-858993460 ...}	Student
[1]	{id=-858993460 name=<Error reading characters of string.> age=-858993460 ...}	Student

Call Stack:

- testDebug.exe!main() Line 12
- [External Code]
- kernel32.dll![] (Frames below may be incorrect and/or missing, no

The elements of structure array

The member values of each structure element

Visual Studio IDE showing a C++ program being debugged. The code defines a `Student` struct and a `main` function that initializes an array `s` of `Student` objects.

```
1  #include <iostream>
2  using namespace std;
3
4  struct Student {
5      int id;
6      string name;
7      int age;
8      double score[3];
9  };
10 int main()
11 {
12     Student s[2] = { {102, "Tom", 18, 78.5, 89, 92.5},
13                     {104, "Peter", 19, 67, 75, 80.5} };
14
15     for (int i = 0; i < 2; i++)
16         cout << s[i].id << " " << s[i].name << " " << s[i].age << s[i].score[0] << " " << s[i].score[1] << " " << s[i].score[2] << endl;
17
18     return 0;
19 }
```

The **Locals** window shows the state of the `s` array after initialization:

Name	Value	Type
<code>s</code>	<code>0x004ff728 {{id=102 name="Tom" age=18 ...}, {id=104 name="Peter" age=19 ...}}</code>	<code>Student[2]</code>
<code>s[0]</code>	<code>{id=102 name="Tom" age=18 ...}</code>	<code>Student</code>
<code>s[1]</code>	<code>{id=104 name="Peter" age=19 ...}</code>	<code>Student</code>

The values in the `Value` column are highlighted in red, indicating they are the current values.

Call Stack:

- `testDebug.exe!main() Line 12`
- `[External Code]`
- `kernel32.dll![[Frames below may be incorrect and/or missing, no s`

After initialization, the values of member are set down.
Red means the current values.

How to check the value of a pointer ?

Set a breakpoint and Start Debugging (shortcut:F5)

The screenshot shows the Visual Studio IDE with a C++ project named 'testDebug'. The main window displays the source code of 'testPointer.cpp'. A breakpoint is set at line 12, which is highlighted with a red circle. The code in the main function is as follows:

```
4 int main()
5 {
6     int* pc, c;
7
8     c = 5;
9     cout << "Address of c (&c): " << &c << endl;
10    cout << "Value of c (c): " << c << endl << endl;
11
12    pc = &c;
13    cout << "Address that pointer pc holds (pc) " << pc << endl;
14    cout << "Content of the address pointer pc holds (*pc) " << *pc << endl << endl;
15
16    c = 11;
17    cout << "Address that pointer pc holds (pc) " << pc << endl;
18    cout << "Content of the address pointer pc holds (*pc) " << *pc << endl << endl;
19
20    *pc = 2;
21    cout << "Address of c (&c): " << &c << endl;
22    cout << "Value of c (c): " << c << endl << endl;
23
24    return 0;
25 }
```

The 'Diagnostics Tools' pane on the right shows the 'Events' tab with a 'Show Events (1 of 1)' button. The 'Process Memory' and 'CPU (% of all processors)' sections show graphs. The 'Summary' tab is selected, showing 'Events', 'Memory Usage', and 'CPU Usage'.

The 'Locals' pane at the bottom left shows the current state of local variables:

Name	Value	Type
c	5	int
pc	0x00000000 (???)	int *

The 'Call Stack' pane on the right shows the current call stack with the entry 'testPointer.exe!main() Line 12'.

Annotations in the image provide context for the 'Locals' window:

- A blue callout points to the variable 'pc' in the 'Locals' window, stating: "pointer's name".
- A blue callout points to the value '0x00000000 (???)' for 'pc', stating: "The value of pointer is an address which is now uncertain, because this statement is not executed."
- A blue callout points to the type 'int *' for 'pc', stating: "the type of pointer".

The status bar at the bottom indicates 'Ready' and 'Add to Source Control'.

Press “step over(F10)” to run the program step by step

The screenshot shows the Visual Studio Code interface with a C++ program being debugged. The code is in `test.cpp`, and the program is running at line 13. The 'Locals' window shows the variable `pc` with a value of `0x0093f850`, which is an address. A call stack shows the current function is `main()`. A blue callout box explains that the value of the pointer is an address, while the value in the curly brace is the value of a variable which the pointer points to.

```
4 int main()
5 {
6     int* pc, c;
7
8     c = 5;
9     cout << "Address of c (&c): " << &c << endl;
10    cout << "Value of c (c): " << c << endl << endl;
11
12    pc = &c;
13    cout << "Address that pointer pc holds (pc) " << pc << endl;
14    cout << "Content of the address pointer pc holds (*pc) " << *pc << endl << endl;
15
16    c = 11;
17    cout << "Address that pointer pc holds (pc) " << pc << endl;
18    cout << "Content of the address pointer pc holds (*pc) " << *pc << endl << endl;
19
20    *pc = 2;
21    cout << "Address of c (&c): " << &c << endl;
22    cout << "Value of c (c): " << c << endl << endl;
23
24    return 0;
25 }
```

Locals

Name	Value	Type
c	5	int
pc	0x0093f850 {5}	int*

Call Stack

Name	Lang
testPointer.exe!main() Line 13	C++
[External Code]	
kernel32.dll![]	Un...

The value of the pointer is an address, while the value in the curly brace is the value of a variable which the pointer points to.

Visual Studio Code interface showing a C++ program being debugged. The main window displays the source code of `testPointer.cpp`, which includes a `main()` function. The code assigns a pointer `pc` to the address of `c`, prints its address and content, then modifies `c` to 11 and prints the address and content of `pc` again. The `Locals` window shows the current state of variables: `c` is 11 and `pc` is 0x0093f850 {11}. The `Call Stack` window shows the current frame is `testPointer.exe!main() Line 17`.

The `testPointer.cpp` source code:

```
4 int main()
5 {
6     int* pc, c;
7
8     c = 5;
9     cout << "Address of c (&c): " << &c << endl;
10    cout << "Value of c (c): " << c << endl << endl;
11
12    pc = &c;
13    cout << "Address that pointer pc holds (pc) " << pc << endl;
14    cout << "Content of the address pointer pc holds (*pc) " << *pc << endl << endl;
15
16    c = 11;
17    cout << "Address that pointer pc holds (pc) " << pc << endl;
18    cout << "Content of the address pointer pc holds (*pc) " << *pc << endl << endl;
19
20    *pc = 2;
21    cout << "Address of c (&c): " << &c << endl;
22    cout << "Value of c (c): " << c << endl << endl;
23
24    return 0;
25 }
```

The `Locals` window shows:

Name	Value	Type
c	11	int
pc	0x0093f850 {11}	int*

The `Call Stack` window shows:

Name	Lang
testPointer.exe!main() Line 17	C++
[External Code]	
kernel32.dll![]	Un...

A blue callout box at the bottom states: "Modifying the value of the variable, the `*pc` is changed synchronously."

testDebug

Process: [21692] testPointer.exe Lifecycle Events Thread: [21696] Main Thread Stack Frame: main

Pointer.cpp testArray.cpp test.cpp

(Global Scope) main()

```
4 int main()
5 {
6     int* pc, c;
7
8     c = 5;
9     cout << "Address of c (&c): " << &c << endl;
10    cout << "Value of c (c): " << c << endl << endl;
11
12    pc = &c;
13    cout << "Address that pointer pc holds (pc) " << pc << endl;
14    cout << "Content of the address pointer pc holds (*pc) " << *pc << endl << endl;
15
16    c = 11;
17    cout << "Address that pointer pc holds (pc) " << pc << endl;
18    cout << "Content of the address pointer pc holds (*pc) " << *pc << endl << endl;
19
20    *pc = 2;
21    cout << "Address of c (&c): " << &c << endl;
22    cout << "Value of c (c): " << c << endl << endl;
23
24    return 0;
25
26 }
```

121 % No issues found Ln: 21 Ch: 1 TABS CRLF

Locals

Search (Ctrl+E) Search Depth: 3

Name	Value	Type
c	2	int
pc	0x0093f850 {2}	int*

Call Stack

Name	Lang
testPointer.exe!main() Line 21	C++
[External Code]	
kernel32.dll![] (Frames below may be incorrect and/or missing, no symbols loaded for kernel32.dll)	Un...

Diagnostic Tools

Diagnostics session: 0 seconds (559 ms sele...)

558.2ms 558

Events

Process Memory (KB)

845 845

CPU (% of all processors)

100 100

Summary Events Memory Usage CPU Usage

Events

Show Events (8 of 8)

Memory Usage

Take Snapshot

Enable heap profiling (affects performanc

CPU Usage

Record CPU Profile

Ready

Add to Source Control

You can modify the value of the variable by pointer **pc**.

How to check the value of a pointer who points to an array?

Set a breakpoint and Start Debugging (shortcut:F5)

The screenshot shows the Visual Studio Code interface with a C++ project named "testDebug". The main window displays the source code of "PointToArray.cpp". A breakpoint is set at line 9, which is circled in red. The code defines an array "arr" of 5 floats and a pointer "ptr". The program prints the address of "arr" and then attempts to print the address using "ptr".

```
1 #include <iostream>
2 using namespace std;
3
4 int main()
5 {
6     float arr[5];
7     float* ptr;
8
9     cout << "Displaying address using array: " << endl;
10    for (int i = 0; i < 5; i++)
11        cout << "&arr[" << i << "] = " << &arr[i] << endl;
12
13    ptr = arr;
14    cout << "\nDisplaying address using pointer:" << endl;
15    for (int i = 0; i < 5; i++)
16        cout << "ptr + " << i << " = " << ptr + i << endl;
17
18    for (int i = 0; i < 5; i++)
19        arr[i] = i * 2;
20
21    cout << "\nDisplaying values of elements using pointer:" << endl;
22    for (int i = 0; i < 5; i++)
```

The "Locals" window shows the state of the program at the breakpoint:

Name	Value	Type
arr	0x005af888 (-107374176, -107374176, -107374176, -107...	float[5]
[0]	-107374176.	float
[1]	-107374176.	float
[2]	-107374176.	float
[3]	-107374176.	float
[4]	-107374176.	float
ptr	0x00000000 (???)	float*

Callouts explain the state of the variables:

- The address of arr is certain**: Points to the "arr" variable in the "Locals" window.
- The value of ptr is not certain because it doesn't point to any variable now.**: Points to the "ptr" variable in the "Locals" window.

The "Call Stack" window shows the current function call: "PointToArray.exe!main() Line 9".

Press "step over(F10)" to run the program step by step

The screenshot shows the Visual Studio Code interface with a C++ program being debugged. The top toolbar has the 'Step Over' button (F10) highlighted with a red box. The code in the editor is as follows:

```
1 #include <iostream>
2 using namespace std;
3
4 int main()
5 {
6     float arr[5];
7     float* ptr;
8
9     cout << "Displaying address using array: " << endl;
10    for (int i = 0; i < 5; i++)
11        cout << "&arr[" << i << "] = " << &arr[i] << endl;
12
13    ptr = arr;
14    cout << "\nDisplaying address using pointer:" << endl;
15    for (int i = 0; i < 5; i++)
16        cout << "ptr + " << i << " = " << ptr + i << endl;
17
18    for (int i = 0; i < 5; i++)
19        arr[i] = i * 2;
20
21    cout << "\nDisplaying values of elements using pointer:" << endl;
22    for (int i = 0; i < 5; i++)
```

The 'Locals' window at the bottom left shows the following data:

Name	Value	Type
arr	0x005af888 {-107374176, -107374176, -107374176, -107374176, -107374176}	float[5]
[0]	-107374176	float
[1]	-107374176	float
[2]	-107374176	float
[3]	-107374176	float
[4]	-107374176	float
ptr	0x005af888 {-107374176,}	float *

The 'Diagnostics Tools' window on the right shows the following performance metrics:

- Diagnostics session: 0 seconds (91 ms selection)
- Events: 86ms, 88ms
- Process Memory (KB): 937
- CPU (% of all processors): 100

The 'Call Stack' window at the bottom right shows the following call stack:

- PointToArray.exe!main() Line 14
- [External Code]
- kernel32.dll![] Frames below may be incorrect and/or missing, no symbols loaded for kernel32.dll

The pointer points to the array, the values(address) of array and pointer are the same. Because the values of elements are not certain, the value in the curly brace are not certain either.

Visual Studio IDE showing a C++ program named `PointToArray.exe` running in Debug mode. The program is currently paused at line 23, which is highlighted in yellow. The code is as follows:

```
5 {  
6     float arr[5];  
7     float* ptr;  
8  
9     cout << "Displaying address using array: " << endl;  
10    for (int i = 0; i < 5; i++)  
11        cout << "&arr[" << i << "] = " << &arr[i] << endl;  
12  
13    ptr = arr;  
14    cout << "\nDisplaying address using pointer:" << endl;  
15    for (int i = 0; i < 5; i++)  
16        cout << "ptr + " << i << " = " << ptr + i << endl;  
17  
18    for (int i = 0; i < 5; i++)  
19        arr[i] = i * 2;  
20  
21    cout << "\nDisplaying values of elements using pointer:" << endl;  
22    for (int i = 0; i < 5; i++)  
23        cout << "*(ptr + " << i << ") = " << *(ptr + i) << endl; ≤ 1ms elapsed  
24  
25    return 0;  
26 }
```

The `for` loop on line 18 and the expression `*(ptr + i)` on line 23 are highlighted with red boxes.

The **Locals** window shows the state of the program:

Name	Value	Type
arr	0x005af888 {0.000000000, 2.000000000, 4.000000000, 6.000000000, 8.000000000}	float[5]
arr[0]	0.000000000	float
arr[1]	2.000000000	float
arr[2]	4.000000000	float
arr[3]	6.000000000	float
arr[4]	8.000000000	float
i	0	int
ptr	0x005af888 {0.000000000}	float *

The **Call Stack** window shows the current call frame:

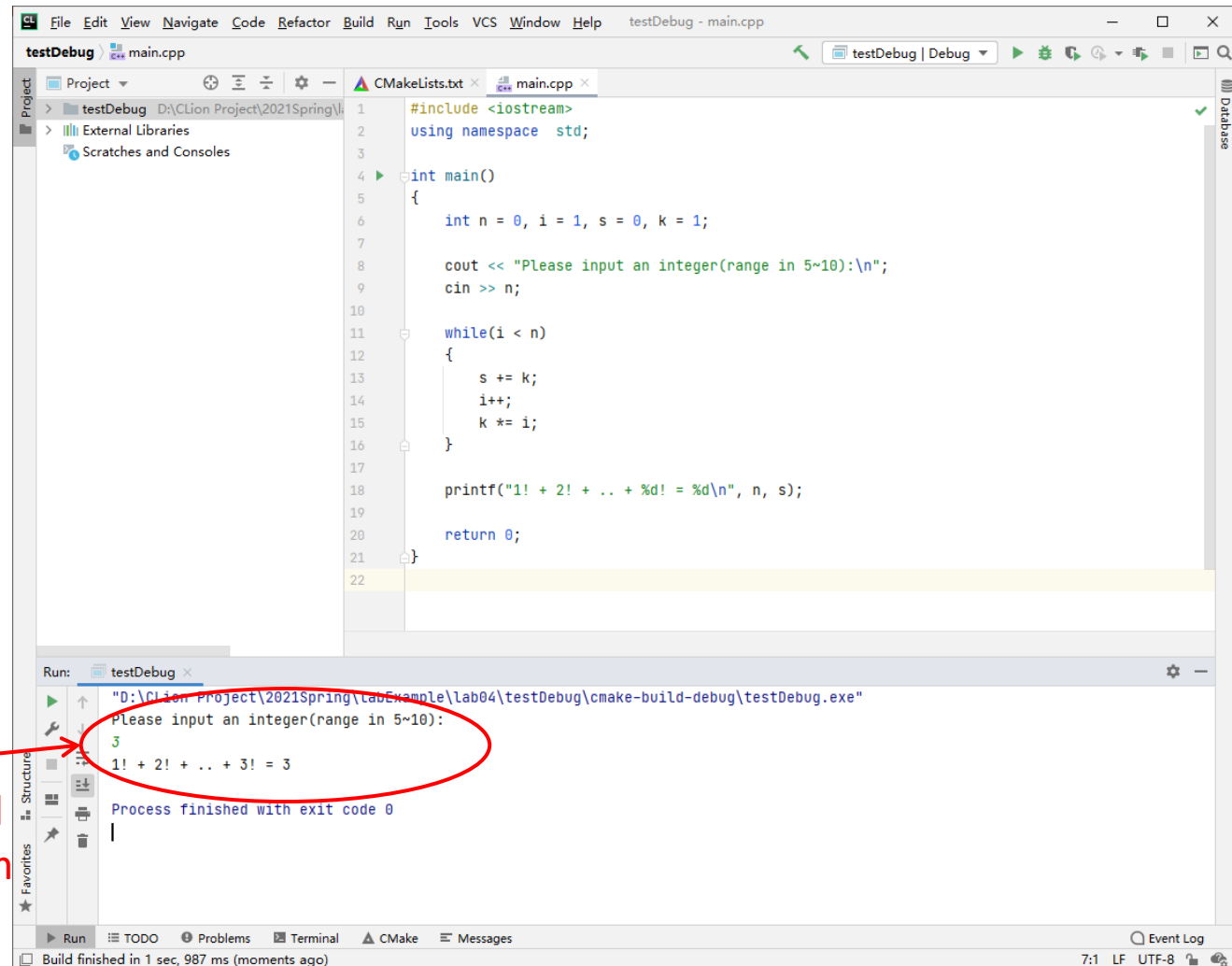
Name	Lang
PointToArray.exe!main() Line 23	C++
[External Code]	
kernel32.dll![]	Un...

The **Diagnostics Tools** window shows the session status: 0 seconds (106 ms selected). The **Events** window shows a timeline of events. The **Process Memory (KB)** window shows a memory usage graph. The **CPU (% of all processors)** window shows a CPU usage graph.

The values in the curly brace are certain after assigning values to the array. You can access the elements of array by the pointer.

Debug C++ program in CLion

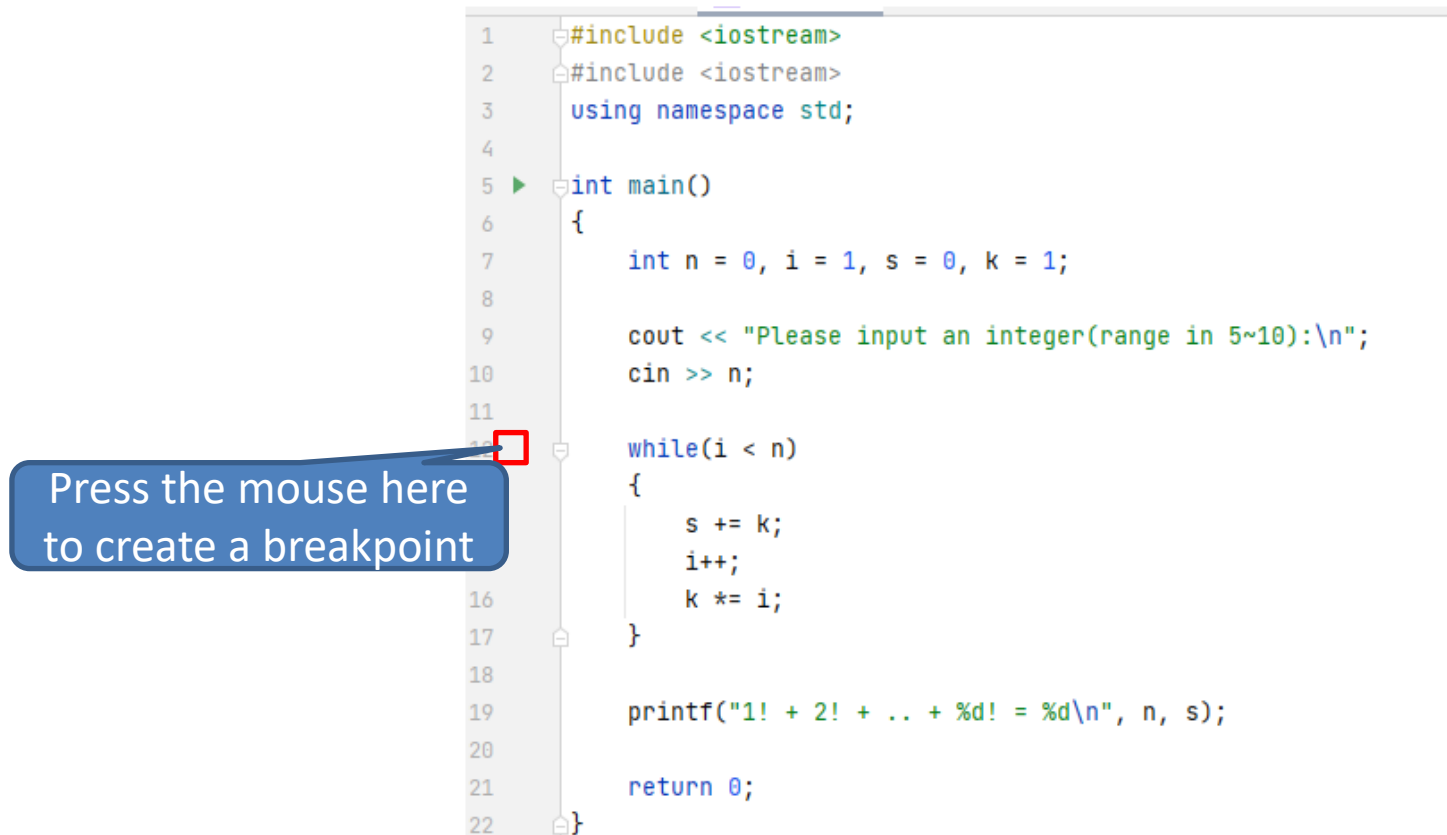
Type a C++ program in CLion and run



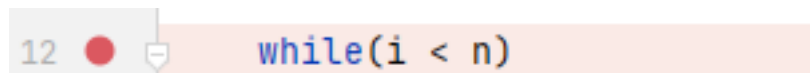
The result is wrong:
You are recommended
to debug your program
for finding the logical
errors.

Step 1: Add a breakpoint

Move the mouse to the line where you want to set breakpoint, press left button of mouse at the left grey edge

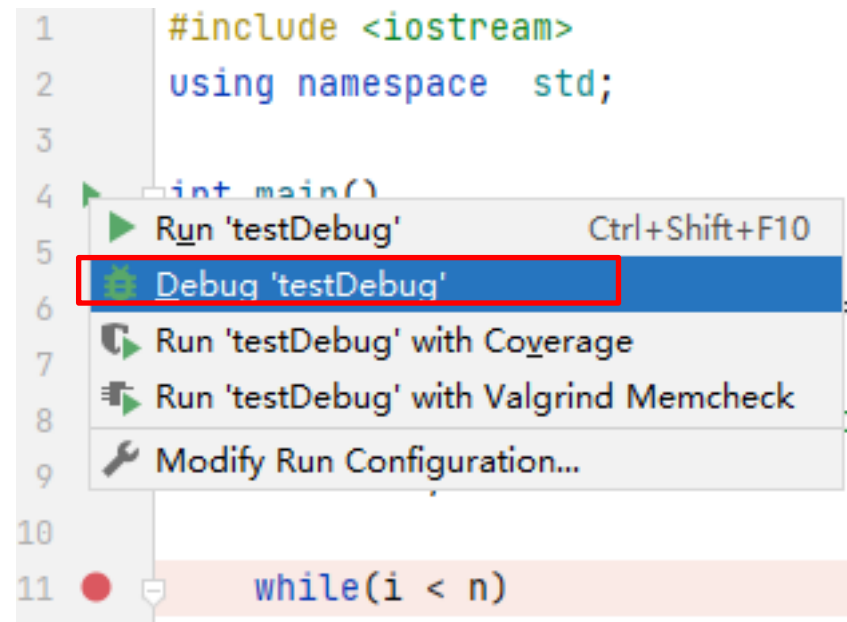
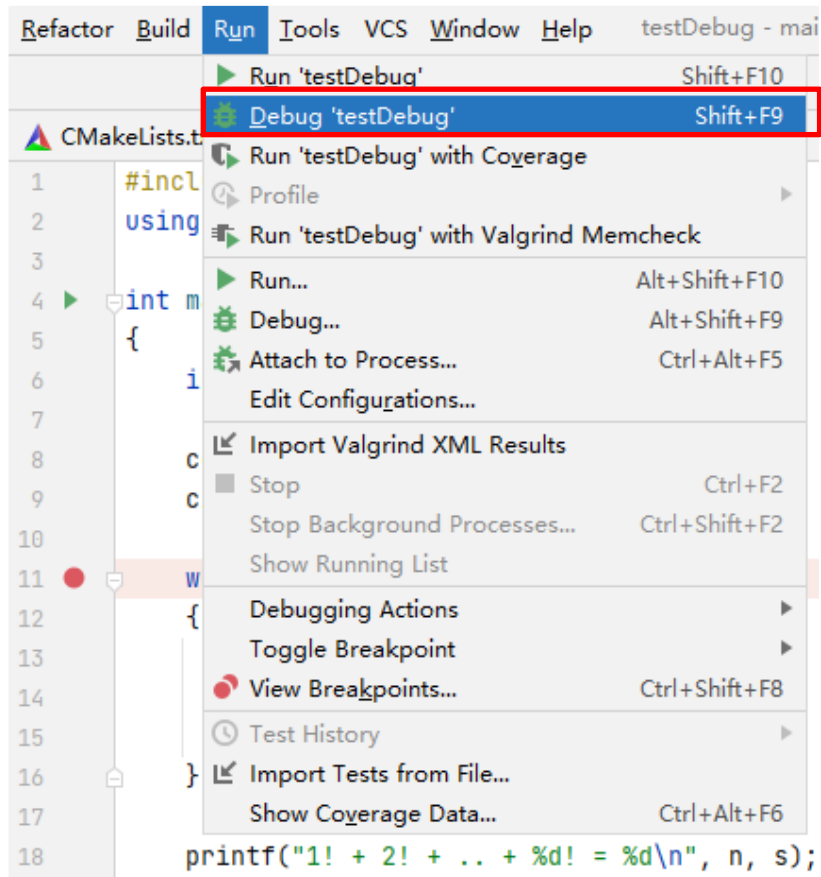


A red point will appear at the edge, this red point is the breakpoint.

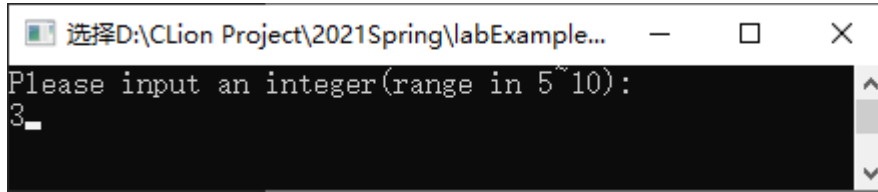


Step 2: Run your program in debugging way

Go to Run-> Debug... or press the green triangle button and choose “Debug “ or press the Debug button 

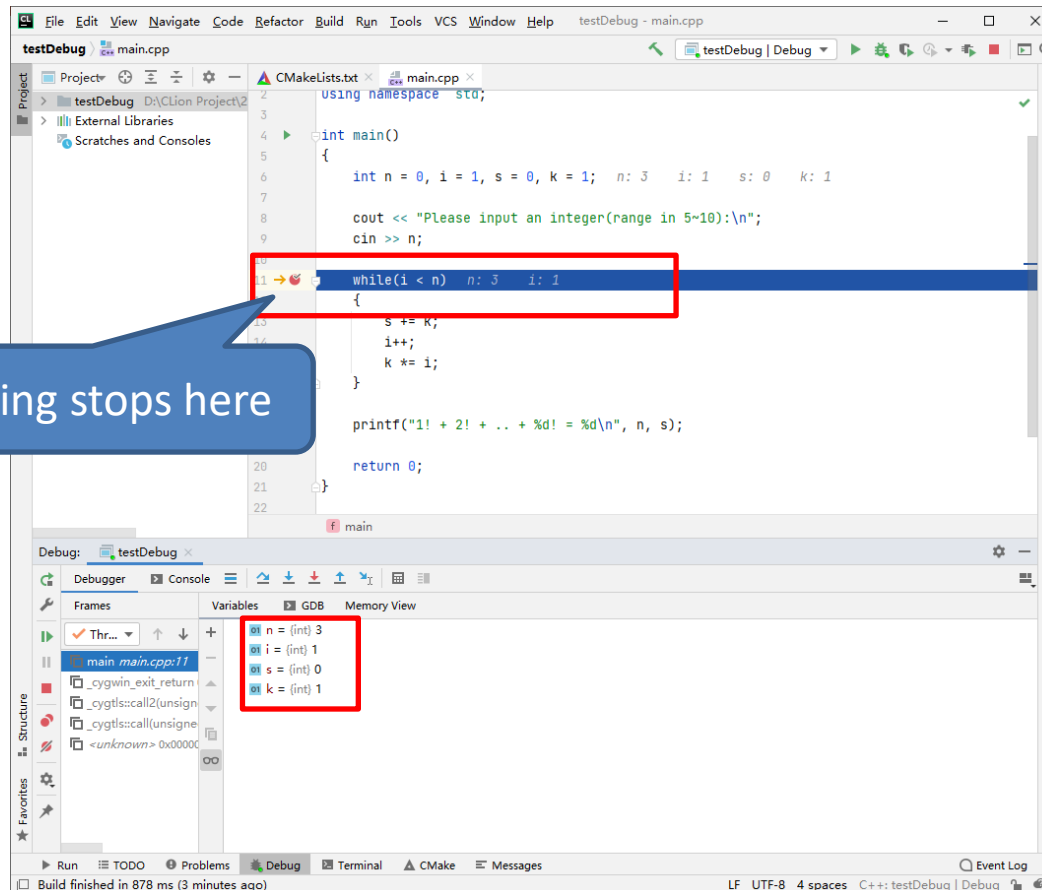


Input an integer and press Enter key

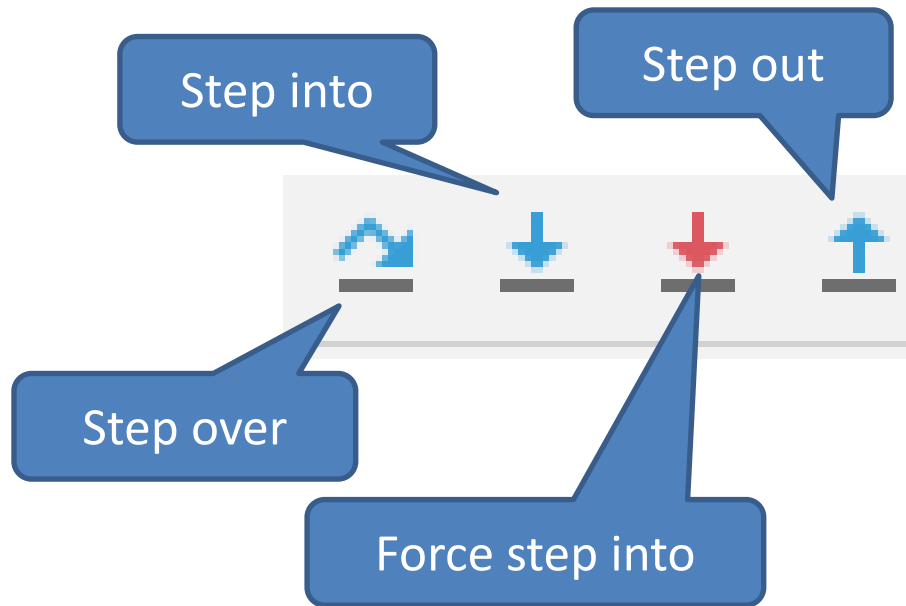


```
选择D:\CLion Project\2021Spring\labExample...  
Please input an integer(range in 5~10):  
3_
```

The running stops at the breakpoint, some values of variables are shown at the debugger window at the bottom of the window.



- **Step Into:** run step by step and trace into the function
- **Step Over:** run step by step and not trace into the function
- **Force Step Into:** trace into the function which defines by system
- **Step Out:** jump out of the function



Step 3: Press “step over” button to run the program

The yellow arrow represents the line that will be run next

```
using namespace std;

int main()
{
    int n = 0, i = 1, s = 0, k = 1;  n: 3   i: 1   s: 0   k: 1

    cout << "Please input an integer(range in 5~10):\n";
    cin >> n;

    while(i < n)  n: 3   i: 1
    {
        s += k;  s: 0   k: 1
        i++;
        k *= i;
    }

    printf("1! + 2! + .. + %d! = %d\n", n, s);

    return 0;
}
```

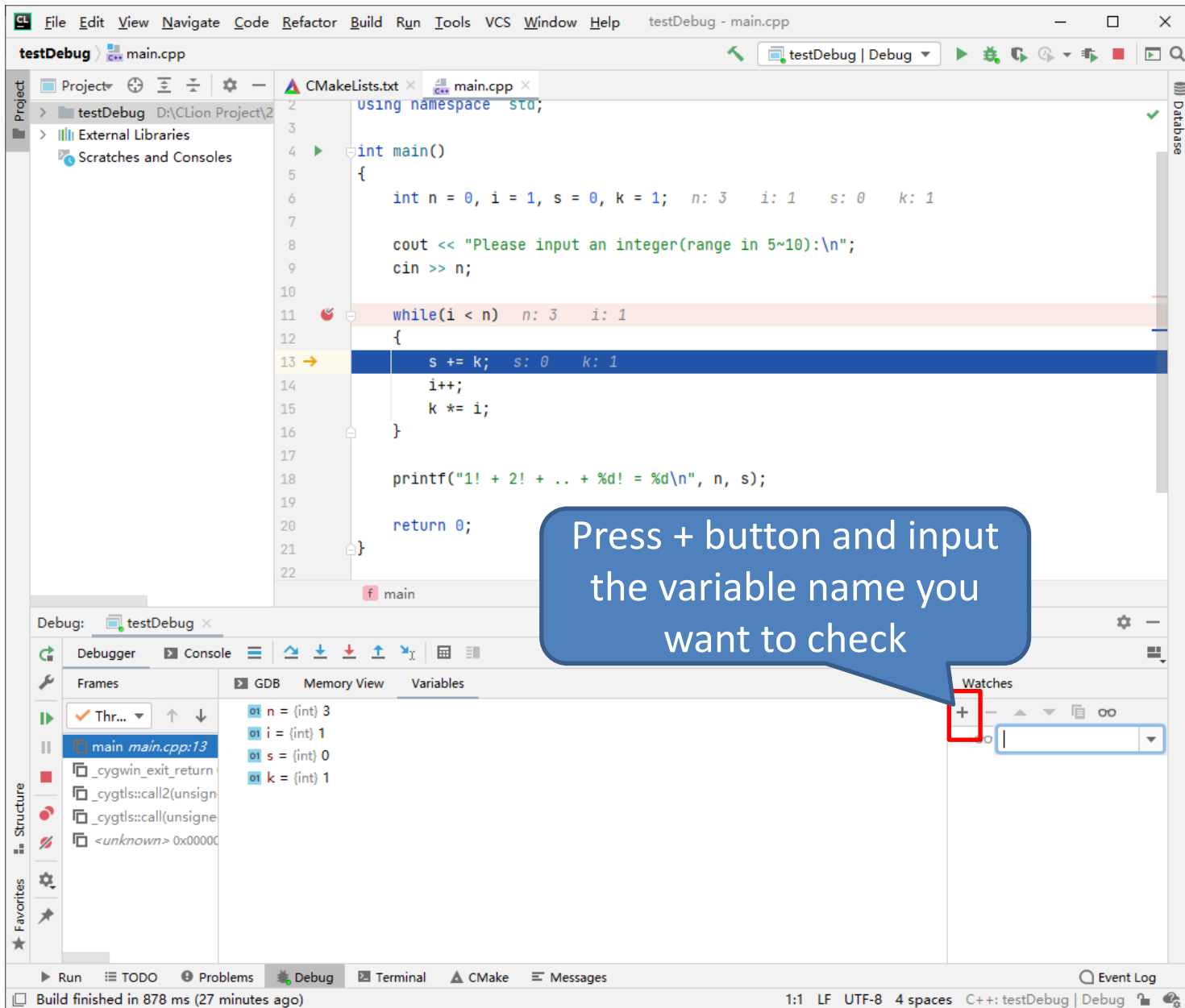
The values of variables


Variable	Value
n	(int) 3
i	(int) 1
s	(int) 0
k	(int) 1

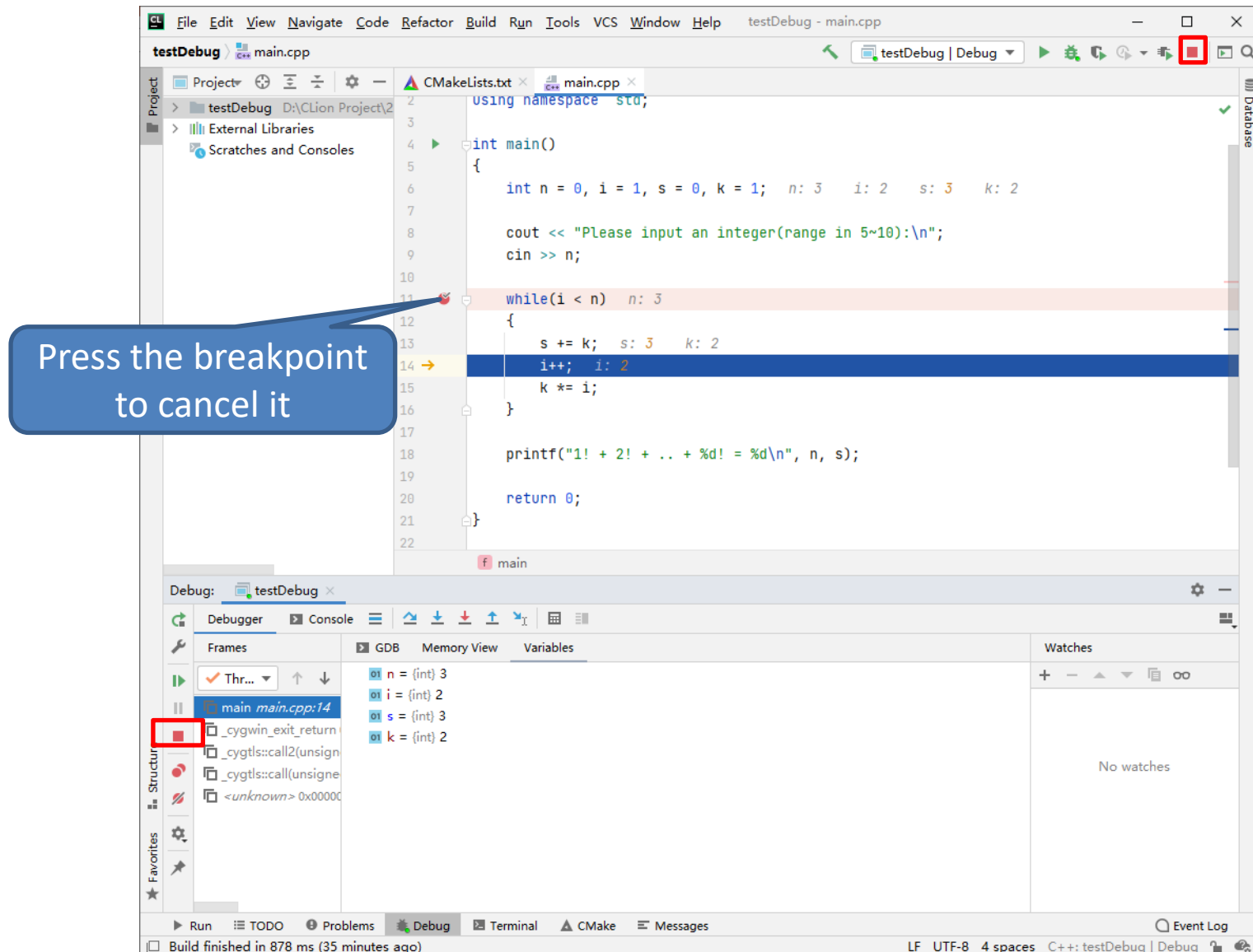
Press Watch button to show Watch window

Build finished in 878 ms (14 minutes ago)

LF UTF-8 4 spaces C++: testDebug | Debug



You can find the logic errors by watching the value of variables and checking the flow of the program. At last, press “stop”  to return back to the edit status.



How to check the value of an array?

Set a breakpoint and Debug

The screenshot shows an IDE with a C++ project named 'testDebug'. The main.cpp file is open, showing the following code:

```
21 int a[5]; a: int [5]
22
23 for(int i = 0; i < 5; i++)
24     a[i] = i * 2;
25
26 for(int e:a)
27     cout << e << " ";
28     cout << "\n";
29
30 return 0;
31 }
```

A breakpoint is set at line 23, indicated by a red circle around the line number. The debugger window is open, showing the 'Variables' tab. The variable 'a' is listed as 'a = (int [5])'. A blue callout box points to the variable 'a' in the debugger window with the text: 'Press > to watch the values of elements in array'.

The array 'a' is expanded, showing the following values:

Index	Value
01 [0]	{int} -2144427776
01 [1]	{int} 1
01 [2]	{int} -2145163392
01 [3]	{int} 1
01 [4]	{int} 0

testDebug - main.cpp

testDebug | main.cpp

```
21 int a[5]; a: int [5]
22
23 for(int i = 0; i < 5; i++)
24     a[i] = i * 2;
25
26 → for(int e:a) a: int [5]
27     cout << e << " ";
28     cout << "\n";
29
30 return 0;
31 }
32
```

main

Debug: testDebug

Debugger Console Frames Variables GDB Memory View

Thr... main main.cpp:26

a = {int [5]}

- [0] = (int) 0
- [1] = (int) 2
- [2] = (int) 4
- [3] = (int) 6
- [4] = (int) 8

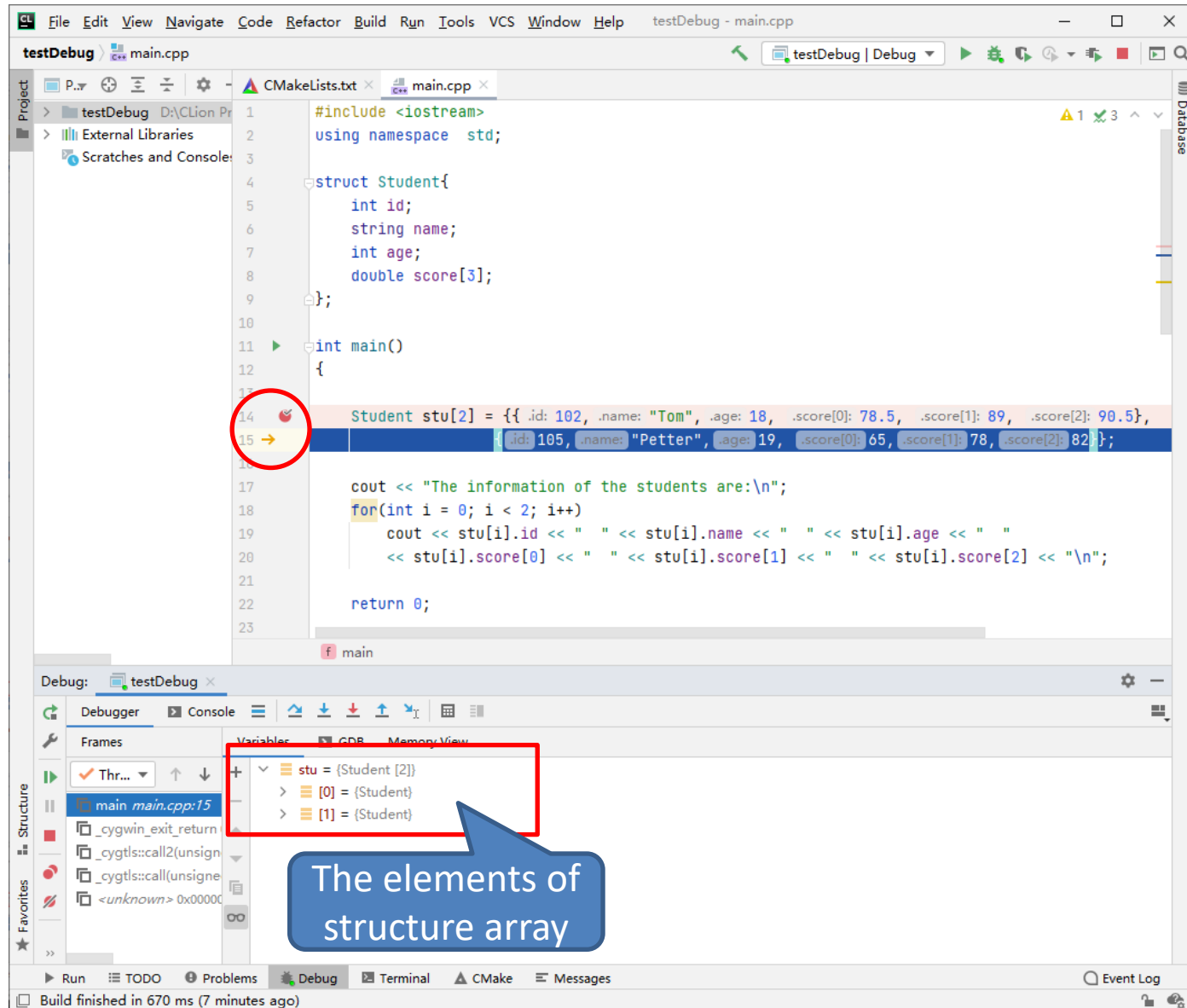
After loop, the values of elements in array are correct

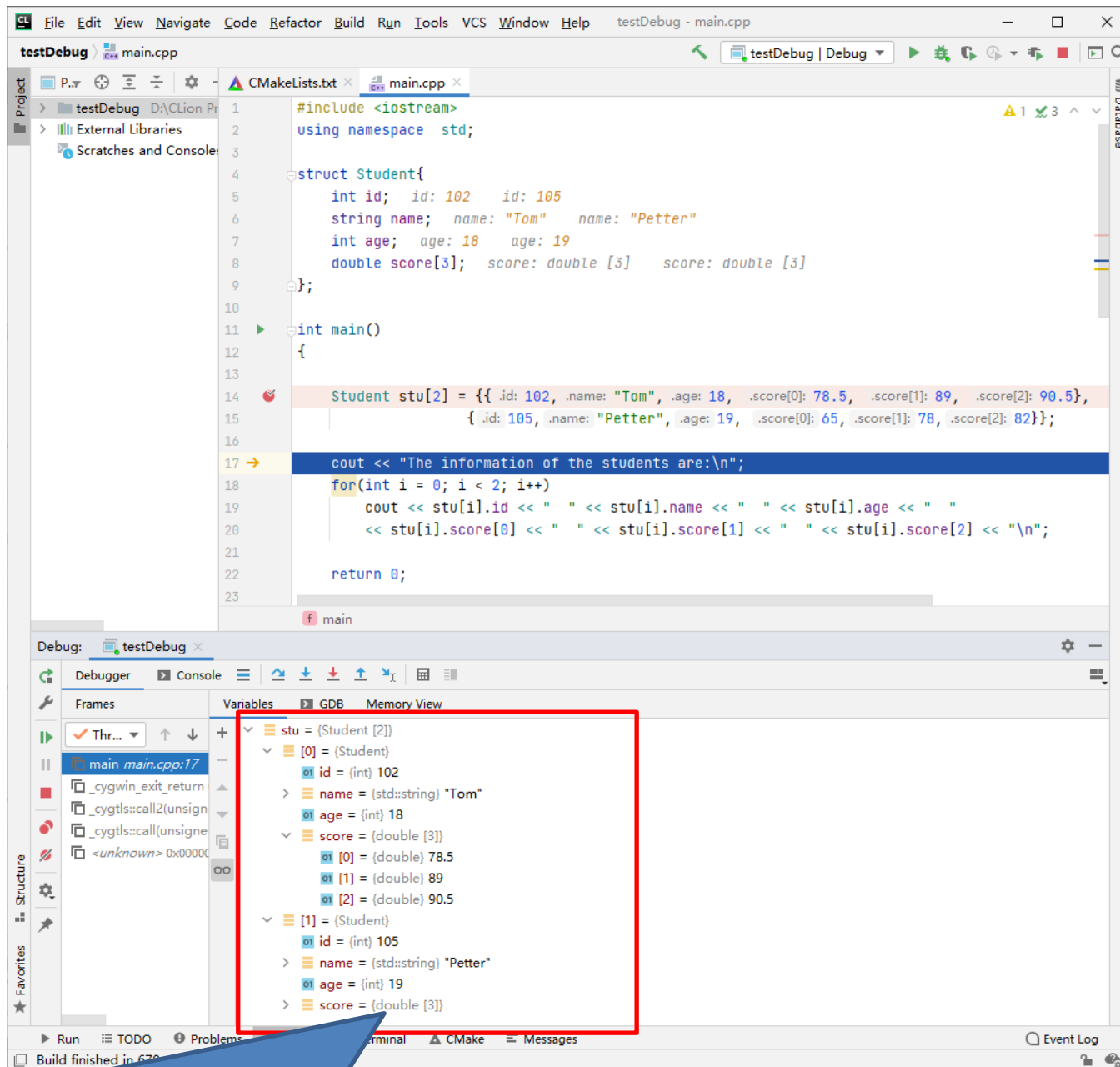
Run TODO Problems Debug Terminal CMake Messages Event Log

Build finished in 2 sec, 282 ms (6 minutes ago) 23:1 LF UTF-8 4 spaces C++: testDebug | Debug

How to check the value of a structure?

Set a breakpoint and Debug





After initialization, the values of member are set down.

How to check the value of a pointer ?

Set a breakpoint and Debug

The screenshot shows a C++ IDE with a project named 'testDebug'. The main.cpp file contains the following code:

```
1 #include <iostream>
2 using namespace std;
3
4 int main()
5 {
6     int *pc; pc: 0xffffcc40
7
8
9     int c = 5;
10    cout << "Address of c (&c): " << &c << endl;
11    cout << "Value of c (c): " << c << endl << endl;
12
13    pc = &c;
14    cout << "Address that pointer pc holds (pc) " << pc << endl;
15    cout << "Content of the address pointer pc holds (*pc) " << *pc << endl << endl;
16
17    c = 11;
18    cout << "Address that pointer pc holds (pc) " << pc << endl;
19    cout << "Content of the address pointer pc holds (*pc) " << *pc << endl << endl;
20
21    *pc = 2;
22    cout << "Address of c (&c): " << &c << endl;
23    cout << "Value of c (c): " << c << endl << endl;
24
25    return 0;
}
```

A breakpoint is set at line 6, indicated by a red apple icon. The IDE is in a debug state, and the 'Variables' panel shows the following information:

Variable	Type	Value
pc	(int *)	0xffffcc40

Annotations on the image:

- A blue callout bubble points to the variable declaration `int *pc;` and the debug output, stating: "The value of pointer is an address. The compiler gives an address to the pointer."
- A blue callout bubble points to the variable name `pc` in the 'Variables' panel, stating: "pointer's name".
- A blue callout bubble points to the variable type `(int *)` in the 'Variables' panel, stating: "the type of pointer".

The status bar at the bottom indicates: "Build finished in 773 ms (moments ago)".

```
testDebug \ main.cpp
CMakeLists.txt x main.cpp x
> testDebug D:\CLion Pr
> External Libraries
Scratches and Console
1 #include <iostream>
2 using namespace std;
3
4 int main()
5 {
6
7 int *pc; pc: 0xffffcbd4
8
9 int c = 5; c: 5
10 cout << "Address of c (&c): " << &c << endl;
11 cout << "Value of c (c): " << c << endl << endl;
12
13 pc = &c; c: 5
14 cout << "Address that pointer pc holds (pc) " << pc << endl; pc: 0xffffcbd4
15 cout << "Content of the address pointer pc holds (*pc) " << *pc << endl << endl;
16
17 c = 11;
18 cout << "Address that pointer pc holds (pc) " << pc << endl;
19 cout << "Content of the address pointer pc holds (*pc) " << *pc << endl << endl;
20
21 *pc = 2;
```

Press "step over" to run the program step by step

Debug: testDebug x

Debugger Console Frames Variables GDB Memory View

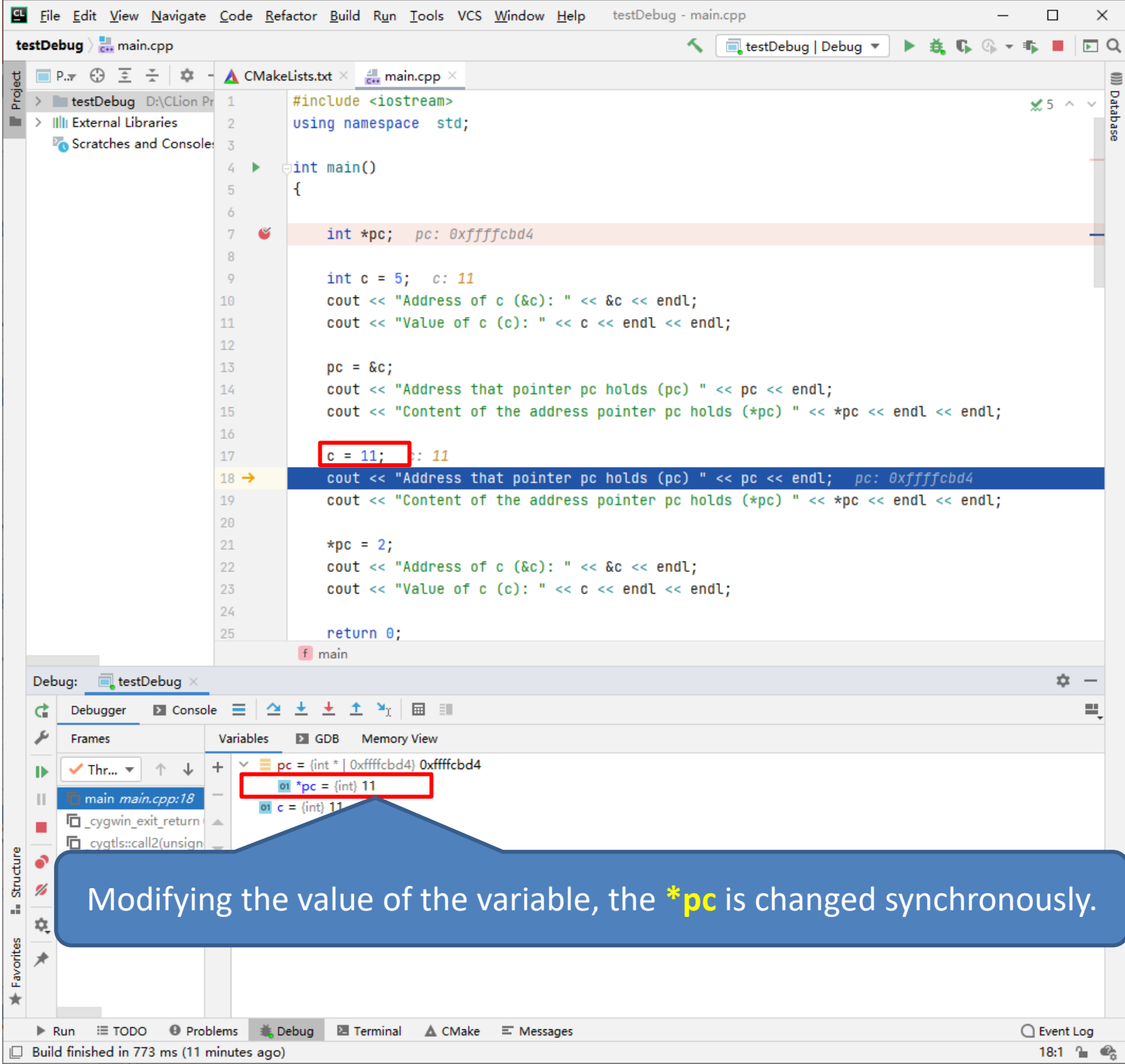
pc = (int * | 0xffffcbd4) 0xffffcbd4
*pc = (int) 5
c = (int) 5

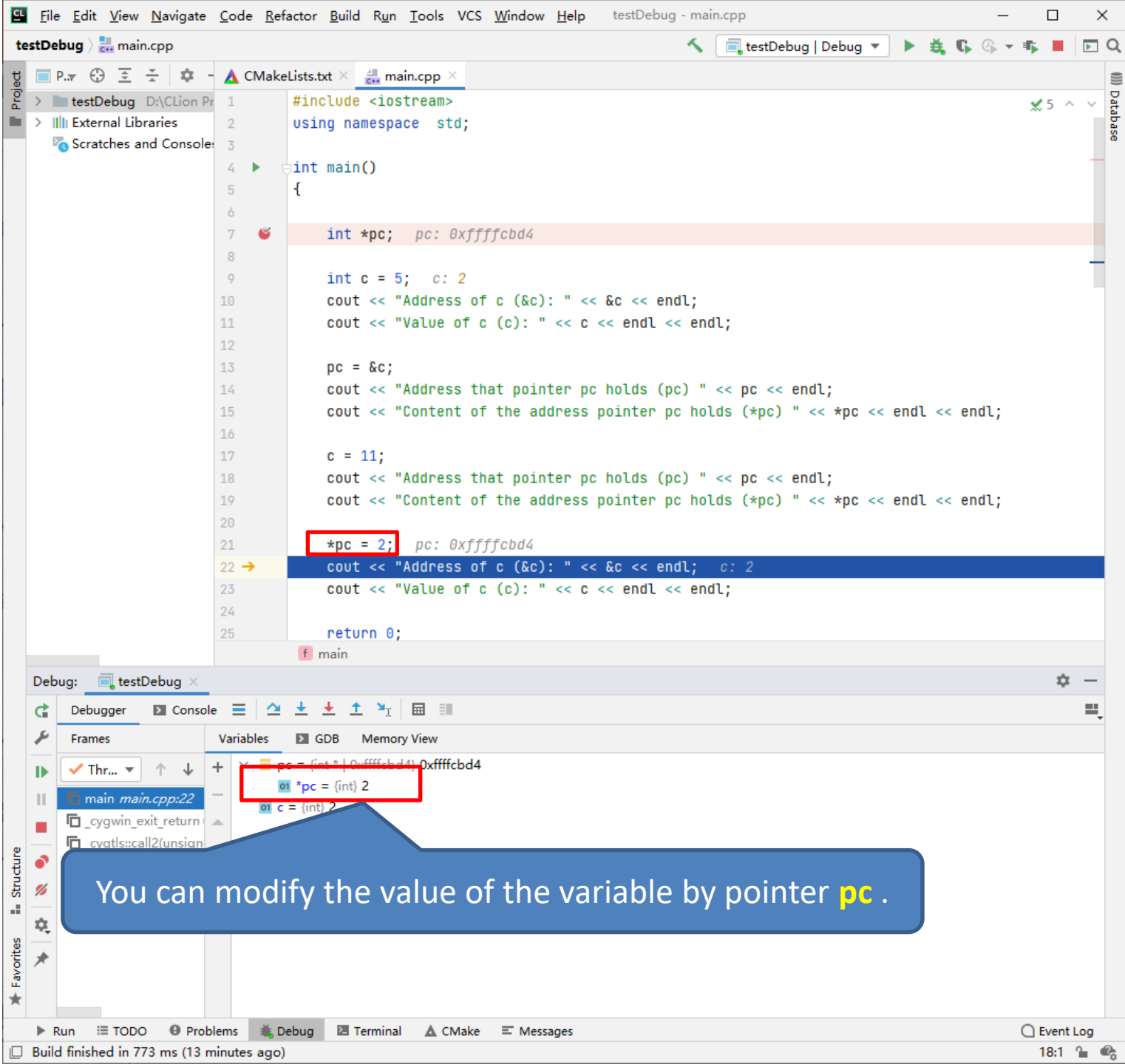
main main.cpp:14
_cygwin_exit_return
_cygtls::call2(unsig
_cygtls::call(unsig
<unknown> 0x00000000

Run TODO Problems Debug Terminal CMake Messages Event Log

Build finished in 773 ms (7 minutes ago) 14:1

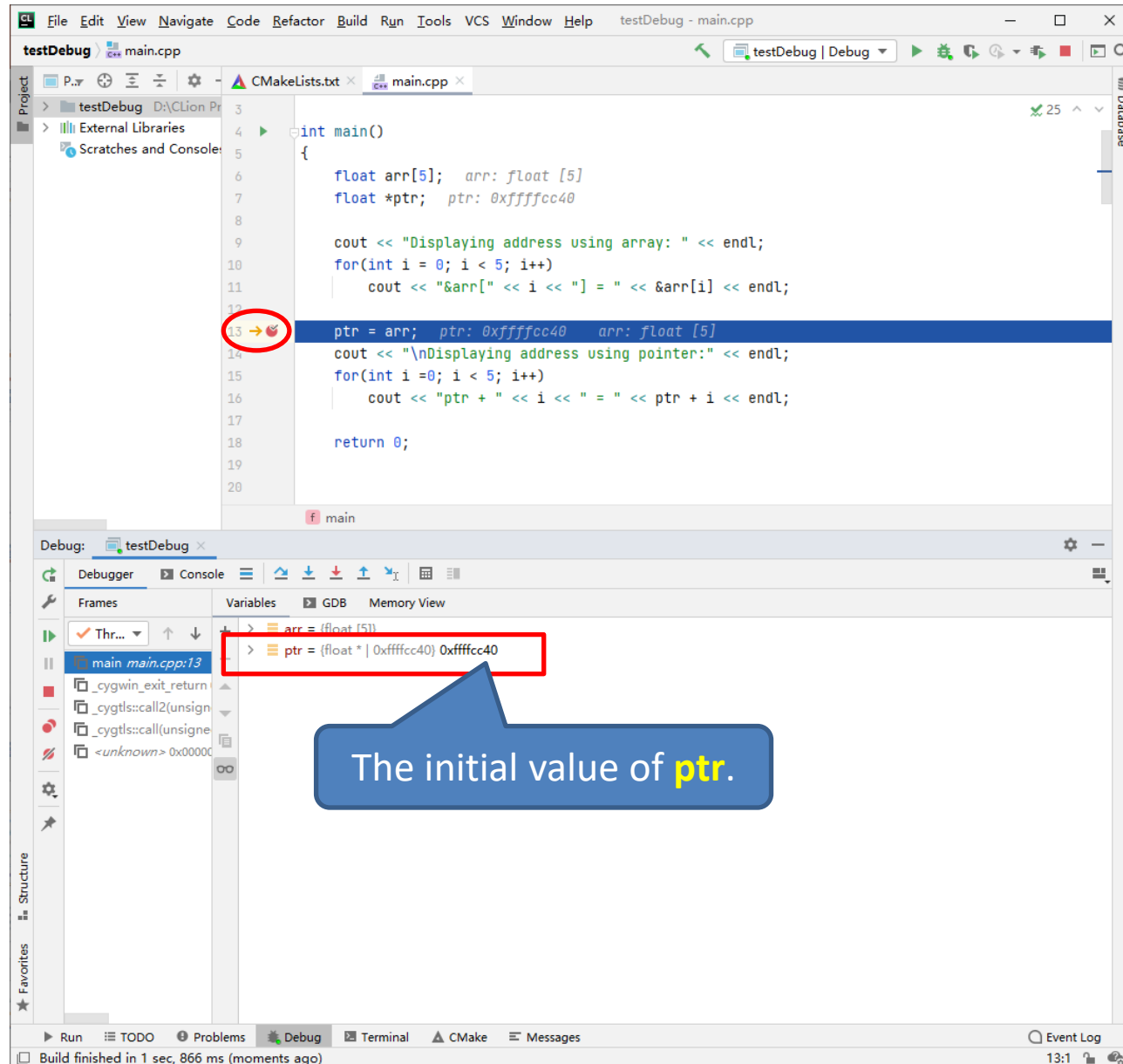
The address is changed to which the pointer points.
***pc** is the value of the variable to which the pointer points.

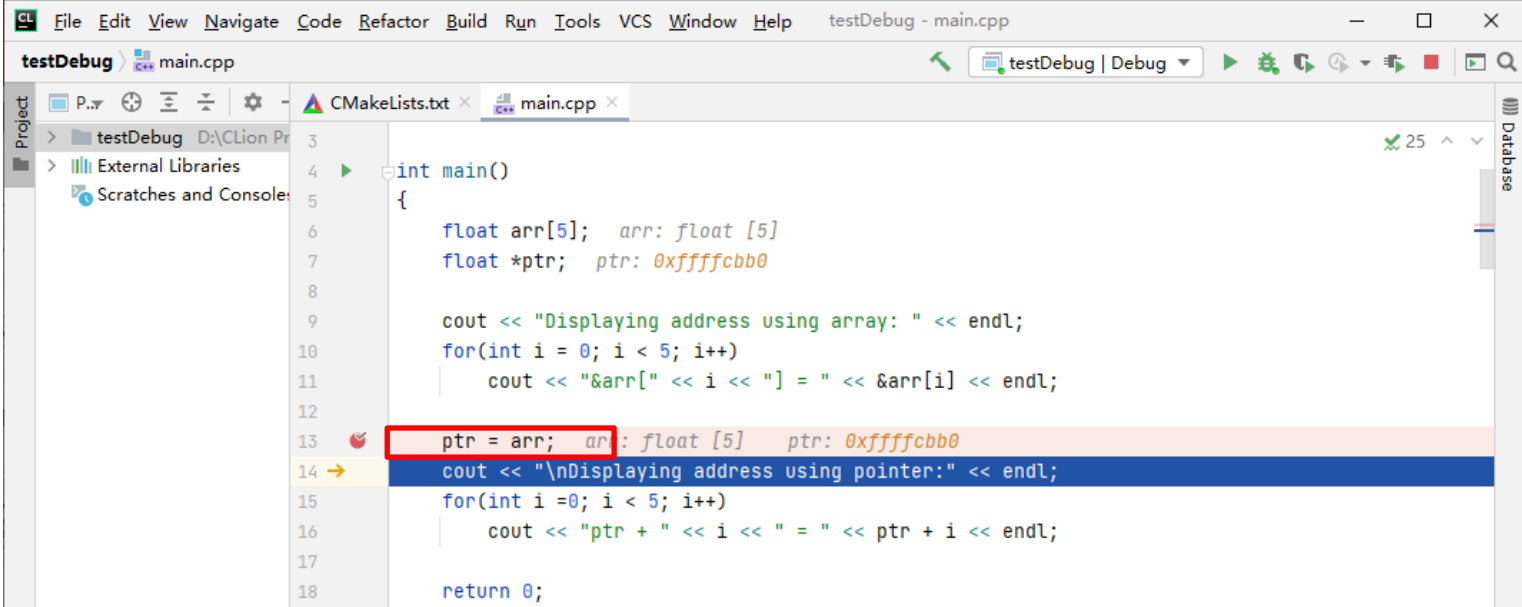




How to check the value of a pointer who points to an array?

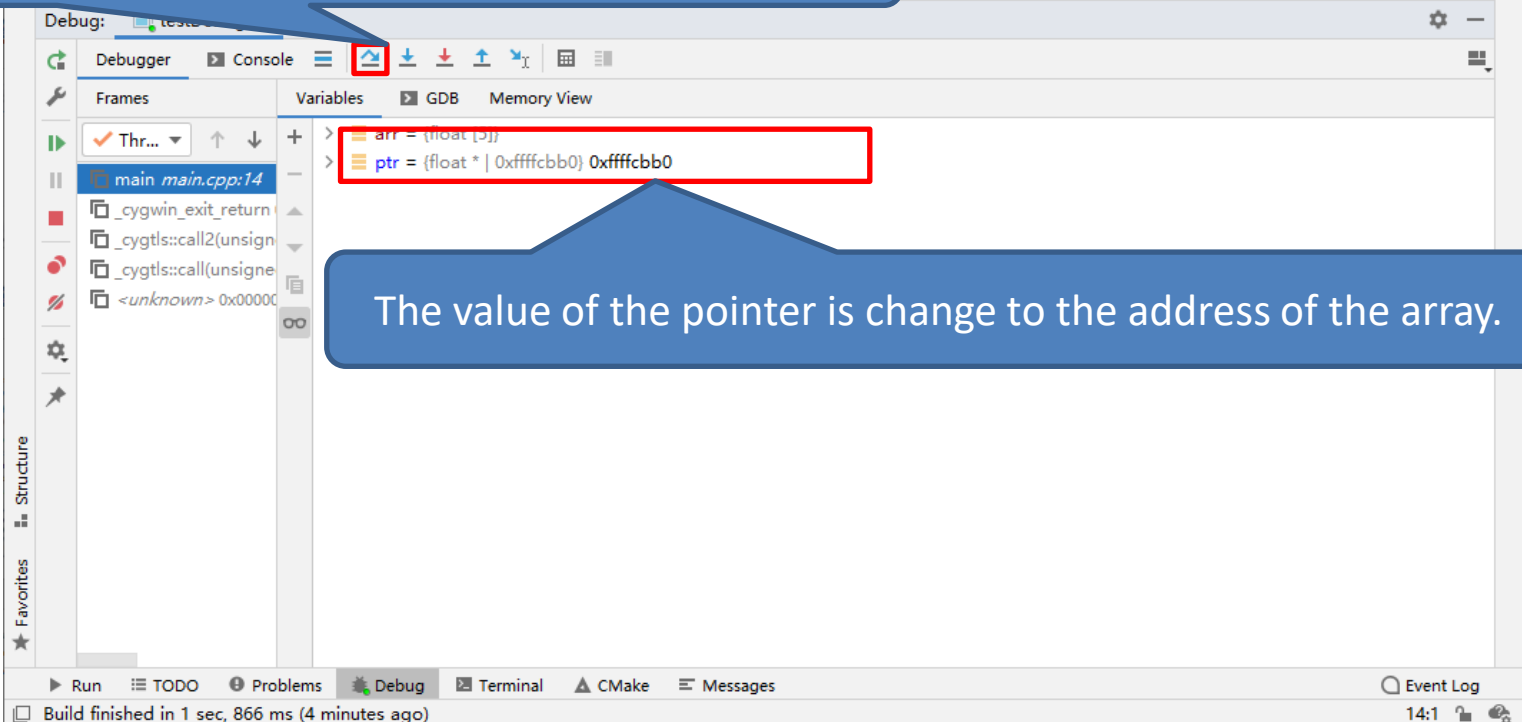
Set a breakpoint and Debug





```
1 // testDebug - main.cpp
2
3
4 int main()
5 {
6     float arr[5]; arr: float [5]
7     float *ptr; ptr: 0xffffcbb0
8
9     cout << "Displaying address using array: " << endl;
10    for(int i = 0; i < 5; i++)
11        cout << "&arr[" << i << "] = " << &arr[i] << endl;
12
13    ptr = arr; arr: float [5] ptr: 0xffffcbb0
14    cout << "\nDisplaying address using pointer:" << endl;
15    for(int i = 0; i < 5; i++)
16        cout << "ptr + " << i << " = " << ptr + i << endl;
17
18    return 0;
19 }
```

Press “step over” to run the program step by step



Debug: testDebug

Debugger Console

Frames

- main main.cpp:14
- _cygwin_exit_return
- _cygtls::call2(unsigned)
- _cygtls::call(unsigned)
- <unknown> 0x00000000

Variables

- arr = {float [5]}
- ptr = {float * | 0xffffcbb0} 0xffffcbb0

The value of the pointer is change to the address of the array.

Run TODO Problems Debug Terminal CMake Messages Event Log

Build finished in 1 sec, 866 ms (4 minutes ago) 14:1