



## §. 关于动态内存申请后越界访问的深度讨论

★ 如何判断动态申请越界 (C方式, 注意源程序后缀为.c)

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
```

```
int main()
{
```

```
    char *p;
    p = (char *)malloc(10 * sizeof(char));
    if (p == NULL)
        return -1;
```

```
    strcpy(p, "123456789");
```

```
① p[10] = 'a';    //此句越界
   p[14] = 'A';    //此句越界
   p[15] = 'B';    //此句越界
```

```
② p[10] = '\xfd'; //此句越界
   printf("addr:%p\n", p);
```

```
   for (int i = -4; i < 16; i++) //注意, 只有0-9是合理范围, 其余都是越界读
       printf("%p:%02x\n", (p+i), p[i]);
```

```
③ free(p);
```

```
   return 0;
```

```
}
```

在VS2022的x86/Debug模式下运行:

- 1、①②③全部注释, 观察运行结果
- 2、①放开, ②③注释, 观察运行结果
- 3、①③放开, ②注释, 观察运行结果
- 4、①②③全部放开, 观察运行结果

结论: VS的Debug模式是如何判断  
动态申请内存访问越界的?

再观察下面四种环境下的运行结果:

VS2022 x86/Release

Dev 32bit-Debug

Dev 32bit-Release

Linux

每种讨论的结果可截图+文字说明,  
如果几种环境的结果一致, 用一个  
环境的截图+文字说明即可(可加页)



# §. 关于动态内存申请后越界访问的深度讨论

★ 如何判断动态申请越界（C方式，**注意源程序后缀为.c**）

VS2022 x86/Debug

①②③全部注释

```
Microsoft Visual Studio 调试控制台
addr:00A69948
00A69944: ffffffff
00A69945: ffffffff
00A69946: ffffffff
00A69947: ffffffff
00A69948: 31
00A69949: 32
00A6994A: 33
00A6994B: 34
00A6994C: 35
00A6994D: 36
00A6994E: 37
00A6994F: 38
00A69950: 39
00A69951: 00
00A69952: ffffffff
00A69953: ffffffff
00A69954: ffffffff
00A69955: ffffffff
00A69956: 41
00A69957: 42
```

①放开，②③注释

```
Microsoft Visual Studio 调试控制台
addr:00F99948
00F99944: ffffffff
00F99945: ffffffff
00F99946: ffffffff
00F99947: ffffffff
00F99948: 31
00F99949: 32
00F9994A: 33
00F9994B: 34
00F9994C: 35
00F9994D: 36
00F9994E: 37
00F9994F: 38
00F99950: 39
00F99951: 00
00F99952: 61
00F99953: ffffffff
00F99954: ffffffff
00F99955: ffffffff
00F99956: 41
00F99957: 42
```

①③放开，②注释

```
Microsoft Visual Studio 调试控制台
addr:01659948
01659944: ffffffff
01659945: ffffffff
01659946: ffffffff
01659947: ffffffff
01659948: 31
01659949: 32
0165994A: 33
0165994B: 34
0165994C: 35
0165994D: 36
0165994E: 37
0165994F: 38
01659950: 39
01659951: 00
01659952: 61
01659953: ffffffff
01659954: ffffffff
01659955: ffffffff
01659956: 41
01659957: 42
```

Microsoft Visual Studio 调试控制台

Debug Error!

Program: D:\Visual Studio\bin\Debug\bin\New.exe

HEAP CORRUPTION DETECTED: after normal block (4B2) at 00A69946.  
CRT detected that the application wrote to memory after end of heap block.

(Press Retry to debug the application)

[OK] [Debug] [Cancel]

①②③全部放开

```
Microsoft Visual Studio 调试控制台
addr:008E9948
008E9944: ffffffff
008E9945: ffffffff
008E9946: ffffffff
008E9947: ffffffff
008E9948: 31
008E9949: 32
008E994A: 33
008E994B: 34
008E994C: 35
008E994D: 36
008E994E: 37
008E994F: 38
008E9950: 39
008E9951: 00
008E9952: ffffffff
008E9953: ffffffff
008E9954: ffffffff
008E9955: ffffffff
008E9956: 41
008E9957: 42
```

VS2022 Debug模式将用户申请的内存前、后四个字节内容置为fd。当用户释放动态申请的内存时，VS会检测这些位置的值是否仍均为fd，如果是，则认为用户没有越界；如果任何一处不是，则认为用户越界。所以修改已超过检测范围的p[14]、p[15]不会报错，但修改p[10]-p[13]则会引起VS报错。如果用户不释放动态申请的内存或只读不写，无论怎么越界均不会报错。



# §. 关于动态内存申请后越界访问的深度讨论

★ 如何判断动态申请越界（C方式，**注意源程序后缀为.c**）

VS2022 x86/Release

①②③全部注释

```
Microsoft Visual Studio 调试控制台
addr:0175F2F0
0175F2EC:65
0175F2ED:0c
0175F2EE:00
0175F2EF:ffffff8e
0175F2F0:31
0175F2F1:32
0175F2F2:33
0175F2F3:34
0175F2F4:35
0175F2F5:36
0175F2F6:37
0175F2F7:38
0175F2F8:39
0175F2F9:00
0175F2FA:53
0175F2FB:00
0175F2FC:74
0175F2FD:00
0175F2FE:41
0175F2FF:42
```

①放开，②③注释

```
Microsoft Visual Studio 调试控制台
addr:0074F1D0
0074F1CC:74
0074F1CD:00
0074F1CE:00
0074F1CF:ffffff8e
0074F1D0:31
0074F1D1:32
0074F1D2:33
0074F1D3:34
0074F1D4:35
0074F1D5:36
0074F1D6:37
0074F1D7:38
0074F1D8:39
0074F1D9:00
0074F1DA:61
0074F1DB:00
0074F1DC:53
0074F1DD:00
0074F1DE:41
0074F1DF:42
```

①③放开，②注释

```
Microsoft Visual Studio 调试控制台
addr:00B9F580
00B9F57C:72
00B9F57D:0e
00B9F57E:00
00B9F57F:ffffff8e
00B9F580:31
00B9F581:32
00B9F582:33
00B9F583:34
00B9F584:35
00B9F585:36
00B9F586:37
00B9F587:38
00B9F588:39
00B9F589:00
00B9F58A:61
00B9F58B:00
00B9F58C:63
00B9F58D:00
00B9F58E:41
00B9F58F:42
```

①②③全部放开

```
Microsoft Visual Studio 调试控制台
addr:013CF560
013CF55C:6c
013CF55D:09
013CF55E:00
013CF55F:ffffff8e
013CF560:31
013CF561:32
013CF562:33
013CF563:34
013CF564:35
013CF565:36
013CF566:37
013CF567:38
013CF568:39
013CF569:00
013CF56A:fffffffd
013CF56B:00
013CF56C:69
013CF56D:00
013CF56E:41
013CF56F:42
```

VS2022 x86/Release模式对于越界的读写不敏感，无论是否释放动态申请的空间，对内存区域的越界读写都可以实现且不会报错。



## §. 关于动态内存申请后越界访问的深度讨论

★ 如何判断动态申请越界（C方式，**注意源程序后缀为.c**）

Dev 32bit-Release | Dev 32bit-Debug

①②③全部注释

```
D:\visual studio\  
addr:00D61588  
00D61584:ffffffed  
00D61585:65  
00D61586:00  
00D61587:0e  
00D61588:31  
00D61589:32  
00D6158A:33  
00D6158B:34  
00D6158C:35  
00D6158D:36  
00D6158E:37  
00D6158F:38  
00D61590:39  
00D61591:00  
00D61592:3d  
00D61593:2e  
00D61594:43  
00D61595:4f  
00D61596:41  
00D61597:42
```

①放开，②③注释

```
D:\visual studio\  
addr:00C71588  
00C71584:ffffff9b  
00C71585:ffffffdd  
00C71586:00  
00C71587:0e  
00C71588:31  
00C71589:32  
00C7158A:33  
00C7158B:34  
00C7158C:35  
00C7158D:36  
00C7158E:37  
00C7158F:38  
00C71590:39  
00C71591:00  
00C71592:61  
00C71593:2e  
00C71594:43  
00C71595:4f  
00C71596:41  
00C71597:42
```

①③放开，②注释

```
D:\visual studio\  
addr:00C91588  
00C91584:2a  
00C91585:ffffffae  
00C91586:00  
00C91587:0e  
00C91588:31  
00C91589:32  
00C9158A:33  
00C9158B:34  
00C9158C:35  
00C9158D:36  
00C9158E:37  
00C9158F:38  
00C91590:39  
00C91591:00  
00C91592:61  
00C91593:2e  
00C91594:43  
00C91595:4f  
00C91596:41  
00C91597:42
```

①②③全部放开

```
D:\visual studio\  
addr:00DE1588  
00DE1584:ffffff82  
00DE1585:5b  
00DE1586:00  
00DE1587:0e  
00DE1588:31  
00DE1589:32  
00DE158A:33  
00DE158B:34  
00DE158C:35  
00DE158D:36  
00DE158E:37  
00DE158F:38  
00DE1590:39  
00DE1591:00  
00DE1592:fffffffd  
00DE1593:2e  
00DE1594:43  
00DE1595:4f  
00DE1596:41  
00DE1597:42
```

Dev 32bit-Release/Debug模式得到的运行结果一致。

两种环境动态申请空间将申请空间前两个字节分别置为00,0e，其余位置无固定初始化值。对于越界的读写不敏感，无论是否释放动态申请的空间，对内存区域的越界读写都可以实现且不会报错。





# §. 关于动态内存申请后越界访问的深度讨论

★ 如何判断动态申请越界（C方式，**注意源程序后缀为.c**）

## Linux

①②③全部注释

```
addr:0x265502a0
0x2655029c:00
0x2655029d:00
0x2655029e:00
0x2655029f:00
0x265502a0:31
0x265502a1:32
0x265502a2:33
0x265502a3:34
0x265502a4:35
0x265502a5:36
0x265502a6:37
0x265502a7:38
0x265502a8:39
0x265502a9:00
0x265502aa:00
0x265502ab:00
0x265502ac:00
0x265502ad:00
0x265502ae:41
0x265502af:42
```

①放开，②③注释

```
addr:0x2cb502a0
0x2cb5029c:00
0x2cb5029d:00
0x2cb5029e:00
0x2cb5029f:00
0x2cb502a0:31
0x2cb502a1:32
0x2cb502a2:33
0x2cb502a3:34
0x2cb502a4:35
0x2cb502a5:36
0x2cb502a6:37
0x2cb502a7:38
0x2cb502a8:39
0x2cb502a9:00
0x2cb502aa:61
0x2cb502ab:00
0x2cb502ac:00
0x2cb502ad:00
0x2cb502ae:41
0x2cb502af:42
```

①③放开，②注释

```
addr:0x215d02a0
0x215d029c:00
0x215d029d:00
0x215d029e:00
0x215d029f:00
0x215d02a0:31
0x215d02a1:32
0x215d02a2:33
0x215d02a3:34
0x215d02a4:35
0x215d02a5:36
0x215d02a6:37
0x215d02a7:38
0x215d02a8:39
0x215d02a9:00
0x215d02aa:61
0x215d02ab:00
0x215d02ac:00
0x215d02ad:00
0x215d02ae:41
0x215d02af:42
```

①②③全部放开

```
addr:0x19e802a0
0x19e8029c:00
0x19e8029d:00
0x19e8029e:00
0x19e8029f:00
0x19e802a0:31
0x19e802a1:32
0x19e802a2:33
0x19e802a3:34
0x19e802a4:35
0x19e802a5:36
0x19e802a6:37
0x19e802a7:38
0x19e802a8:39
0x19e802a9:00
0x19e802aa:fd
0x19e802ab:00
0x19e802ac:00
0x19e802ad:00
0x19e802ae:41
0x19e802af:42
```

Linux系统所有空间的初始值均为0，对于越界的读写不敏感，无论是否释放动态申请的空间，对内存区域的越界读写都可以实现且不会报错。



## §. 关于动态内存申请后越界访问的深度讨论

★ 如何判断动态申请越界 (C++方式, 注意源程序后缀为.cpp)

```
#define _CRT_SECURE_NO_WARNINGS
#include <iostream>
#include <cstring>
using namespace std;
```

```
int main()
{
```

```
    char *p;
    p = new(nothrow) char[10];
    if (p == NULL)
        return -1;
    strcpy(p, "123456789");
```

```
① p[10] = 'a';    //此句越界
   p[14] = 'A';    //此句越界
   p[15] = 'B';    //此句越界
```

```
② p[10] = '\xfd'; //此句越界
```

```
    cout << "addr:" << hex << (void *) (p) << endl;
    for (int i = -4; i < 16; i++) //注意, 只有0-9是合理范围, 其余都是越界读
        cout << hex << (void *) (p + i) << ":" << int(p[i]) << endl;
```

```
③ delete[] p;
```

```
    return 0;
```

```
}
```

在VS2022的x86/Debug模式下运行:

- 1、①②③全部注释, 观察运行结果
- 2、①放开, ②③注释, 观察运行结果
- 3、①③放开, ②注释, 观察运行结果
- 4、①②③全部放开, 观察运行结果

结论: VS的Debug模式是如何判断  
动态申请内存访问越界的?

再观察下面四种环境下的运行结果:

VS2022 x86/Release

Dev 32bit-Debug

Dev 32bit-Release

Linux

每种讨论的结果可截图+文字说明,  
如果几种环境的结果一致, 用一个  
环境的截图+文字说明即可(可加页)



# §. 关于动态内存申请后越界访问的深度讨论

★ 如何判断动态申请越界（C++方式，**注意源程序后缀为.cpp**）

VS2022 x86/Debug

①②③全部注释

```
Microsoft Visual Studio 调试控制台
addr:009D1C20
009D1C1C:ffffffffd
009D1C1D:ffffffffd
009D1C1E:ffffffffd
009D1C1F:ffffffffd
009D1C20:31
009D1C21:32
009D1C22:33
009D1C23:34
009D1C24:35
009D1C25:36
009D1C26:37
009D1C27:38
009D1C28:39
009D1C29:0
009D1C2A:ffffffffd
009D1C2B:ffffffffd
009D1C2C:ffffffffd
009D1C2D:ffffffffd
009D1C2E:41
009D1C2F:42
```

①放开，②③注释

```
Microsoft Visual Studio 调试控制台
addr:010B16E0
010B16DC:ffffffffd
010B16DD:ffffffffd
010B16DE:ffffffffd
010B16DF:ffffffffd
010B16E0:31
010B16E1:32
010B16E2:33
010B16E3:34
010B16E4:35
010B16E5:36
010B16E6:37
010B16E7:38
010B16E8:39
010B16E9:0
010B16EA:61
010B16EB:ffffffffd
010B16EC:ffffffffd
010B16ED:ffffffffd
010B16EE:41
010B16EF:42
```

①③放开，②注释

```
Microsoft Visual Studio 调试控制台
addr:01220E70
01220E6C:ffffffffd
01220E6D:ffffffffd
01220E6E:ffffffffd
01220E6F:ffffffffd
01220E70:31
01220E71:32
01220E72:33
01220E73:34
01220E74:35
01220E75:36
01220E76:37
01220E77:38
01220E78:39
01220E79:0
01220E7A:61
01220E7B:ffffffffd
01220E7C:ffffffffd
01220E7D:ffffffffd
01220E7E:41
01220E7F:42
```

①②③全部放开

```
Microsoft Visual Studio 调试控制台
addr:00D91B40
00D91B3C:ffffffffd
00D91B3D:ffffffffd
00D91B3E:ffffffffd
00D91B3F:ffffffffd
00D91B40:31
00D91B41:32
00D91B42:33
00D91B43:34
00D91B44:35
00D91B45:36
00D91B46:37
00D91B47:38
00D91B48:39
00D91B49:0
00D91B4A:ffffffffd
00D91B4B:ffffffffd
00D91B4C:ffffffffd
00D91B4D:ffffffffd
00D91B4E:41
00D91B4F:42
```

VS2022 Debug模式将用户申请的内存前、后四个字节内容置为fd。当用户释放动态申请的内存时，VS会检测这些位置的值是否仍均为fd，如果是，则认为用户没有越界；如果任何一处不是，则认为用户越界。所以修改已超过检测范围的p[14]、p[15]不会报错，但修改p[10]-p[13]则会引起VS报错。如果用户不释放动态申请的内存或只读不写，无论怎么越界均不会报错。



# §. 关于动态内存申请后越界访问的深度讨论

★ 如何判断动态申请越界（C++方式，**注意源程序后缀为.cpp**）

VS2022 x86/Release

①②③全部注释

```
Microsoft Visual Studio 调试控制台
addr:0124EA70
0124EA6C:76
0124EA6D:10
0124EA6E:0
0124EA6F:ffffff8e
0124EA70:31
0124EA71:32
0124EA72:33
0124EA73:34
0124EA74:35
0124EA75:36
0124EA76:37
0124EA77:38
0124EA78:39
0124EA79:0
0124EA7A:69
0124EA7B:0
0124EA7C:6e
0124EA7D:0
0124EA7E:41
0124EA7F:42
```

①放开，②③注释

```
Microsoft Visual Studio 调试控制台
addr:0146E9E0
0146E9DC:54
0146E9DD:a
0146E9DE:0
0146E9DF:ffffff8e
0146E9E0:31
0146E9E1:32
0146E9E2:33
0146E9E3:34
0146E9E4:35
0146E9E5:36
0146E9E6:37
0146E9E7:38
0146E9E8:39
0146E9E9:0
0146E9EA:61
0146E9EB:0
0146E9EC:65
0146E9ED:0
0146E9EE:41
0146E9EF:42
```

①③放开，②注释

```
Microsoft Visual Studio 调试控制台
addr:00A2E8F0
00A2E8EC:54
00A2E8ED:0
00A2E8EE:0
00A2E8EF:ffffff8e
00A2E8F0:31
00A2E8F1:32
00A2E8F2:33
00A2E8F3:34
00A2E8F4:35
00A2E8F5:36
00A2E8F6:37
00A2E8F7:38
00A2E8F8:39
00A2E8F9:0
00A2E8FA:61
00A2E8FB:0
00A2E8FC:4c
00A2E8FD:0
00A2E8FE:41
00A2E8FF:42
```

①②③全部放开

```
Microsoft Visual Studio 调试控制台
addr:0122E9E0
0122E9DC:54
0122E9DD:9
0122E9DE:0
0122E9DF:ffffff8e
0122E9E0:31
0122E9E1:32
0122E9E2:33
0122E9E3:34
0122E9E4:35
0122E9E5:36
0122E9E6:37
0122E9E7:38
0122E9E8:39
0122E9E9:0
0122E9EA:fffffffd
0122E9EB:0
0122E9EC:65
0122E9ED:0
0122E9EE:41
0122E9EF:42
```

VS2022 x86/Release模式对于越界的读写不敏感，动态申请空间的前、后邻近位置均无固定初始化值，无论是否释放动态申请的空间，对内存区域的越界读写都可以实现且不会报错。





## §. 关于动态内存申请后越界访问的深度讨论

★ 如何判断动态申请越界（C++方式，注意源程序后缀为.cpp）

Dev 32bit-Release | Dev 32bit-Debug

①②③全部注释

```
D:\visual studio\  
addr:0xe815c8  
0xe815c4:ffffff80  
0xe815c5:ffffffc6  
0xe815c6:0  
0xe815c7:e  
0xe815c8:31  
0xe815c9:32  
0xe815ca:33  
0xe815cb:34  
0xe815cc:35  
0xe815cd:36  
0xe815ce:37  
0xe815cf:38  
0xe815d0:39  
0xe815d1:0  
0xe815d2:0  
0xe815d3:0  
0xe815d4:0  
0xe815d5:0  
0xe815d6:41  
0xe815d7:42
```

①放开，②③注释

```
D:\visual studio\  
addr:0xe915c8  
0xe915c4:ffffffa7  
0xe915c5:32  
0xe915c6:0  
0xe915c7:e  
0xe915c8:31  
0xe915c9:32  
0xe915ca:33  
0xe915cb:34  
0xe915cc:35  
0xe915cd:36  
0xe915ce:37  
0xe915cf:38  
0xe915d0:39  
0xe915d1:0  
0xe915d2:61  
0xe915d3:0  
0xe915d4:0  
0xe915d5:0  
0xe915d6:41  
0xe915d7:42
```

①③放开，②注释

```
D:\visual studio\  
addr:0xe115c8  
0xe115c4:6e  
0xe115c5:51  
0xe115c6:0  
0xe115c7:e  
0xe115c8:31  
0xe115c9:32  
0xe115ca:33  
0xe115cb:34  
0xe115cc:35  
0xe115cd:36  
0xe115ce:37  
0xe115cf:38  
0xe115d0:39  
0xe115d1:0  
0xe115d2:61  
0xe115d3:0  
0xe115d4:0  
0xe115d5:0  
0xe115d6:41  
0xe115d7:42
```

①②③全部放开

```
D:\visual studio\  
addr:0xde15c8  
0xde15c4:ffffffa1  
0xde15c5:1  
0xde15c6:0  
0xde15c7:e  
0xde15c8:31  
0xde15c9:32  
0xde15ca:33  
0xde15cb:34  
0xde15cc:35  
0xde15cd:36  
0xde15ce:37  
0xde15cf:38  
0xde15d0:39  
0xde15d1:0  
0xde15d2:fffffffd  
0xde15d3:0  
0xde15d4:0  
0xde15d5:0  
0xde15d6:41  
0xde15d7:42
```

Dev 32bit-Release/Debug模式得到的运行结果一致。

两种环境动态申请空间将申请空间前两个字节分别置为00,0e，后四个未赋值字节置为0，其余位置无固定初始化值。对于越界的读写不敏感，无论是否释放动态申请的空间，对内存区域的越界读写都可以实现且不会报错。



## §. 关于动态内存申请后越界访问的深度讨论

★ 如何判断动态申请越界（C++方式，**注意源程序后缀为.cpp**）

### Linux

①②③全部注释

```
addr:0x1db81eb0
0x1db81eac:0
0x1db81ead:0
0x1db81eae:0
0x1db81eaf:0
0x1db81eb0:31
0x1db81eb1:32
0x1db81eb2:33
0x1db81eb3:34
0x1db81eb4:35
0x1db81eb5:36
0x1db81eb6:37
0x1db81eb7:38
0x1db81eb8:39
0x1db81eb9:0
0x1db81eba:0
0x1db81ebb:0
0x1db81ebc:0
0x1db81ebd:0
0x1db81ebe:41
0x1db81ebf:42
```

①放开，②③注释

```
addr:0x22bd1eb0
0x22bd1eac:0
0x22bd1ead:0
0x22bd1eae:0
0x22bd1eaf:0
0x22bd1eb0:31
0x22bd1eb1:32
0x22bd1eb2:33
0x22bd1eb3:34
0x22bd1eb4:35
0x22bd1eb5:36
0x22bd1eb6:37
0x22bd1eb7:38
0x22bd1eb8:39
0x22bd1eb9:0
0x22bd1eba:61
0x22bd1ebb:0
0x22bd1ebc:0
0x22bd1ebd:0
0x22bd1ebe:41
0x22bd1ebf:42
```

①③放开，②注释

```
addr:0x160d1eb0
0x160d1eac:0
0x160d1ead:0
0x160d1eae:0
0x160d1eaf:0
0x160d1eb0:31
0x160d1eb1:32
0x160d1eb2:33
0x160d1eb3:34
0x160d1eb4:35
0x160d1eb5:36
0x160d1eb6:37
0x160d1eb7:38
0x160d1eb8:39
0x160d1eb9:0
0x160d1eba:61
0x160d1ebb:0
0x160d1ebc:0
0x160d1ebd:0
0x160d1ebe:41
0x160d1ebf:42
```

①②③全部放开

```
addr:0x12eb1eb0
0x12eb1eac:0
0x12eb1ead:0
0x12eb1eae:0
0x12eb1eaf:0
0x12eb1eb0:31
0x12eb1eb1:32
0x12eb1eb2:33
0x12eb1eb3:34
0x12eb1eb4:35
0x12eb1eb5:36
0x12eb1eb6:37
0x12eb1eb7:38
0x12eb1eb8:39
0x12eb1eb9:0
0x12eb1eba:fd
0x12eb1ebb:0
0x12eb1ebc:0
0x12eb1ebd:0
0x12eb1ebe:41
0x12eb1ebf:42
```

Linux系统所有空间的初始值均为0，对于越界的读写不敏感，无论是否释放动态申请的空间，对内存区域的越界读写都可以实现且不会报错。



# §. 关于动态内存申请后越界访问的深度讨论

★ 如何判断普通数组的越界访问 (C方式, 注意源程序后缀为.c) char数组

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main()
{
    char arr[10] = "12345678";
    printf("arr[10]=%c\n", arr[10]); //此句越界读
    ① arr[10] = 'a';                //此句越界
    arr[14] = 'A';                  //此句越界
    arr[15] = 'B';
    ② arr[10] = '\xfd';
    printf("addr:%p\n", arr);
    for (int i = -4; i < 16; i++)
        printf("%p:%02x\n", (arr + i), arr[i]);
    return 0;
}
```

在理解P.1/P.2的情况下, 自行构造相似的程序, 来观察数组越界后的内存表现, 并验证与动态申请是否相似

- 1、①②全部注释, 观察运行结果
- 2、①放开, ②注释, 观察运行结果
- 3、①②全部放开, 观察运行结果

要求:

- 1、数组用 char a[10]; 形式
- 2、数组用 int a[10]; 形式
- 3、测试程序在下面五种环境下运行
  - VS2022 x86/Debug
  - VS2022 x86/Release
  - Dev 32bit-Debug
  - Dev 32bit-Release
  - Linux
- 4、每种讨论的结果可截图+文字说明, 如果几种环境的结果一致, 用一个环境的截图+文字说明即可(可加页)



# §. 关于动态内存申请后越界访问的深度讨论

★ 如何判断普通数组的越界访问（C方式，**注意源程序后缀为.c**） **char数组**

VS2022 x86/Debug

①②全部注释

```
Microsoft Visual Studio 调试控制台
arr[10]=?
addr:0136FE80
0136FE7C:ffffffcc
0136FE7D:ffffffcc
0136FE7E:ffffffcc
0136FE7F:ffffffcc
0136FE80:31
0136FE81:32
0136FE82:33
0136FE83:34
0136FE84:35
0136FE85:36
0136FE86:37
0136FE87:38
0136FE88:00
0136FE89:00
0136FE8A:ffffffcc
0136FE8B:ffffffcc
0136FE8C:ffffffcc
0136FE8D:ffffffcc
0136FE8E:41
0136FE8F:42
```

①放开，②注释

```
Microsoft Visual Studio 调试控制台
arr[10]=?
addr:0117FBC0
0117FBBC:ffffffcc
0117FBBD:ffffffcc
0117FBBE:ffffffcc
0117FBBF:ffffffcc
0117FBC0:31
0117FBC1:32
0117FBC2:33
0117FBC3:34
0117FBC4:35
0117FBC5:36
0117FBC6:37
0117FBC7:38
0117FBC8:00
0117FBC9:00
0117FBCA:61
0117FBCB:ffffffcc
0117FBCC:ffffffcc
0117FBCE:41
0117FBCF:42
```

Microsoft Visual Studio 调试控制台

Debug Error!

Program: D:\visual studio\test\Debug\test\_hu.exe  
Module: D:\visual studio\test\Debug\test\_hu.exe  
File:

Run-Time Check Failure #2 - Stack around the variable 'arr' was corrupted.

(Press Retry to debug the application)

中止(A) 重试(R) 忽略(I)

①②全部放开

```
Microsoft Visual Studio 调试控制台
arr[10]=?
addr:00EFFC80
00EFFC7C:ffffffcc
00EFFC7D:ffffffcc
00EFFC7E:ffffffcc
00EFFC7F:ffffffcc
00EFFC80:31
00EFFC81:32
00EFFC82:33
00EFFC83:34
00EFFC84:35
00EFFC85:36
00EFFC86:37
00EFFC87:38
00EFFC88:00
00EFFC89:00
00EFFC8A:ffffffcc
00EFFC8B:ffffffcc
00EFFC8C:ffffffcc
00EFFC8D:ffffffcc
00EFFC8E:41
00EFFC8F:42
```

VS2022 Debug模式char数组将用户申请的内存前、后四个字节内容置为cc。当用户释放动态申请的内存时，VS会检测这些位置的值是否仍均为cc，如果是，则认为用户没有越界；如果任何一处不是，则认为用户越界。所以修改已超过检测范围的arr[14]、arr[15]不会报错，但修改arr[10]-arr[13]则会引起VS报错。如果用户不释放动态申请的内存或只读不写，无论怎么越界均不会报错。





# §. 关于动态内存申请后越界访问的深度讨论

★ 如何判断普通数组的越界访问（C方式，**注意源程序后缀为.c**） **char数组**

**VS2022 x86/Release**

①②全部注释

```
Microsoft Visual Studio 调试控制台
arr[10]=?
addr:004FFA30
004FFA2C:ffffffe0
004FFA2D:ffffff99
004FFA2E:7f
004FFA2F:00
004FFA30:31
004FFA31:32
004FFA32:33
004FFA33:34
004FFA34:35
004FFA35:36
004FFA36:37
004FFA37:38
004FFA38:00
004FFA39:00
004FFA3A:ffffffbb
004FFA3B:75
004FFA3C:13
004FFA3D:13
004FFA3E:0a
004FFA3F:00
```

①放开，②注释

```
Microsoft Visual Studio 调试控制台
arr[10]=?
addr:00FCFA10
00FCFA0C:ffffffe0
00FCFA0D:ffffff99
00FCFA0E:42
00FCFA0F:01
00FCFA10:31
00FCFA11:32
00FCFA12:33
00FCFA13:34
00FCFA14:35
00FCFA15:36
00FCFA16:37
00FCFA17:38
00FCFA18:00
00FCFA19:00
00FCFA1A:61
00FCFA1B:75
00FCFA1C:16
00FCFA1D:13
00FCFA1E:ffffffa5
00FCFA1F:00
```

①②全部放开

```
Microsoft Visual Studio 调试控制台
arr[10]=?
addr:00DDFEC0
00DDFECB:38
00DDFEBD:ffffff9b
00DDFEBE:27
00DDFEBF:01
00DDFEC0:31
00DDFEC1:32
00DDFEC2:33
00DDFEC3:34
00DDFEC4:35
00DDFEC5:36
00DDFEC6:37
00DDFEC7:38
00DDFEC8:00
00DDFEC9:00
00DDFECA:ffffffcc
00DDFECB:75
00DDFECC:16
00DDFECD:13
00DDFECE:ffffffea
00DDFECE:00
```

**VS2022 x86/Release模式char数组其前、后邻近位置无固定初始化值，对于越界的读写不敏感，对内存区域的越界读写都不会报错。但越界的写操作无效，无法修改越界空间的值。**



# §. 关于动态内存申请后越界访问的深度讨论

★ 如何判断普通数组的越界访问（C方式，注意源程序后缀为.c） char数组

Dev 32bit-Release | Dev 32bit-Debug

①②全部注释

①放开，②注释

①②全部放开

```
D:\visual studio\  
arr[10]=  
addr:0065FEC2  
0065FEBE:40  
0065FEBF:00  
0065FEC0:ffffffe0  
0065FEC1:15  
0065FEC2:31  
0065FEC3:32  
0065FEC4:33  
0065FEC5:34  
0065FEC6:35  
0065FEC7:36  
0065FEC8:37  
0065FEC9:38  
0065FECA:00  
0065FECB:00  
0065FECC:0a  
0065FECD:00  
0065FECE:00  
0065FECF:00  
0065FED0:41  
0065FED1:42
```

```
D:\visual studio\  
arr[10]=  
addr:0065FEC2  
0065FEBE:40  
0065FEBF:00  
0065FEC0:ffffffe0  
0065FEC1:15  
0065FEC2:31  
0065FEC3:32  
0065FEC4:33  
0065FEC5:34  
0065FEC6:35  
0065FEC7:36  
0065FEC8:37  
0065FEC9:38  
0065FECA:00  
0065FECB:00  
0065FECC:0a  
0065FECD:00  
0065FECE:00  
0065FECF:00  
0065FED0:41  
0065FED1:42
```

Dev 32bit-Release/Debug模式得到的运行结果一致。

两种环境char数组前后紧邻位置无固定初始化值。对于越界的读写不敏感，对内存区域的越界读写不会报错。对于char数组区域前的内存空间可以成功地进行读写，但对于其后的紧邻四字节空间的写操作是无效的，对于四字节后的空间又可以成功地进行读写。



# §. 关于动态内存申请后越界访问的深度讨论

★ 如何判断普通数组的越界访问（C方式，**注意源程序后缀为.c**） **char**数组

**Linux**

①②全部注释

```
arr[10]=  
addr:0xfffff001c250  
0xfffff001c24c:ff  
0xfffff001c24d:ff  
0xfffff001c24e:00  
0xfffff001c24f:00  
0xfffff001c250:31  
0xfffff001c251:32  
0xfffff001c252:33  
0xfffff001c253:34  
0xfffff001c254:35  
0xfffff001c255:36  
0xfffff001c256:37  
0xfffff001c257:38  
0xfffff001c258:00  
0xfffff001c259:00  
0xfffff001c25a:00  
0xfffff001c25b:00  
0xfffff001c25c:0c  
0xfffff001c25d:00  
0xfffff001c25e:00  
0xfffff001c25f:00
```

①放开，②注释

```
arr[10]=  
addr:0xffffd8d14a80  
0xffffd8d14a7c:ff  
0xffffd8d14a7d:ff  
0xffffd8d14a7e:00  
0xffffd8d14a7f:00  
0xffffd8d14a80:31  
0xffffd8d14a81:32  
0xffffd8d14a82:33  
0xffffd8d14a83:34  
0xffffd8d14a84:35  
0xffffd8d14a85:36  
0xffffd8d14a86:37  
0xffffd8d14a87:38  
0xffffd8d14a88:00  
0xffffd8d14a89:00  
0xffffd8d14a8a:61  
0xffffd8d14a8b:00  
0xffffd8d14a8c:0c  
0xffffd8d14a8d:00  
0xffffd8d14a8e:00  
0xffffd8d14a8f:00
```

①②全部放开

```
arr[10]=  
addr:0xffffd4ed4830  
0xffffd4ed482c:ff  
0xffffd4ed482d:ff  
0xffffd4ed482e:00  
0xffffd4ed482f:00  
0xffffd4ed4830:31  
0xffffd4ed4831:32  
0xffffd4ed4832:33  
0xffffd4ed4833:34  
0xffffd4ed4834:35  
0xffffd4ed4835:36  
0xffffd4ed4836:37  
0xffffd4ed4837:38  
0xffffd4ed4838:00  
0xffffd4ed4839:00  
0xffffd4ed483a:cc  
0xffffd4ed483b:00  
0xffffd4ed483c:0c  
0xffffd4ed483d:00  
0xffffd4ed483e:00  
0xffffd4ed483f:00
```

Linux系统char数组对于越界的读写不敏感，但对于其后的紧邻四字节空间的写操作是无效的，对于四字节后的空间又可以进行读写。



## §. 关于动态内存申请后越界访问的深度讨论

★ 如何判断普通数组的越界访问（C方式，**注意源程序后缀为.c**）

int数组

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main()
{
    int arr[10];
    for (int i = 0; i < 10; ++i)
        arr[i] = i;
    printf("arr[10]=%c\n", arr[10]); //此句越界读
    ① arr[10] = 'a';                //此句越界
    arr[14] = 'A';                  //此句越界
    arr[15] = 'B';
    ② arr[10] = 0xcccccccc;
    printf("addr:%p\n", arr);
    for (int i = -4; i < 16; i++)
        printf("%p:%02x\n", (arr + i), arr[i]);
    return 0;
}
```

在理解P.1/P.2的情况下，自行构造相似的程序，来观察数组越界后的内存表现，并验证与动态申请是否相似

- 1、①②全部注释，观察运行结果
- 2、①放开，②注释，观察运行结果
- 3、①②全部放开，观察运行结果

要求：

- 1、数组用 char a[10]; 形式
- 2、数组用 int a[10]; 形式
- 3、测试程序在下面五种环境下运行  
VS2022 x86/Debug  
VS2022 x86/Release  
Dev 32bit-Debug  
Dev 32bit-Release  
Linux
- 4、每种讨论的结果可截图+文字说明，如果几种环境的结果一致，用一个环境的截图+文字说明即可(可加页)





# §. 关于动态内存申请后越界访问的深度讨论

★ 如何判断普通数组的越界访问（C方式，**注意源程序后缀为.c**） **int**数组

VS2022 x86/Debug

①②全部注释

```
Microsoft Visual Studio 调试控制台
arr[10]=?
addr:0093FA60
0093FA50:cccccccc
0093FA54:0a
0093FA58:cccccccc
0093FA5C:cccccccc
0093FA60:00
0093FA64:01
0093FA68:02
0093FA6C:03
0093FA70:04
0093FA74:05
0093FA78:06
0093FA7C:07
0093FA80:08
0093FA84:09
0093FA88:cccccccc
0093FA8C:198f3184
0093FA90:93fab0
0093FA94:1121b3
0093FA98:41
0093FA9C:42
```

①放开，②注释

```
Microsoft Visual Studio 调试控制台
arr[10]=?
addr:001DF770
001DF760:cccccccc
001DF764:0a
001DF768:cccccccc
001DF76C:cccccccc
001DF770:00
001DF774:01
001DF778:02
001DF77C:03
001DF780:04
001DF784:05
001DF788:06
001DF78C:07
001DF790:08
001DF794:09
001DF798:61
001DF79C:c18dcada
001DF7A0:1df7c0
001DF7A4:f421b3
001DF7A8:41
001DF7AC:42
```

Debug Error!

Program: D:\visual studio\test\Debug\test.exe  
Module: D:\visual studio\test\Debug\test.exe  
File:

Run-time Check Failure #2 - Stack around the variable 'test' was corrupted.

(Press Retry to debug the application)

重试(R) 继续(C) 忽略(I)

①②全部放开

```
Microsoft Visual Studio 调试控制台
arr[10]=?
addr:001EF8A0
001EF890:cccccccc
001EF894:0a
001EF898:cccccccc
001EF89C:cccccccc
001EF8A0:00
001EF8A4:01
001EF8A8:02
001EF8AC:03
001EF8B0:04
001EF8B4:05
001EF8B8:06
001EF8BC:07
001EF8C0:08
001EF8C4:09
001EF8C8:cccccccc
001EF8CC:8de6f370
001EF8D0:1ef8f0
001EF8D4:dd21d3
001EF8D8:41
001EF8DC:42
```

VS2022 Debug模式int数组情况下，空间后紧邻元素arr[10]的值是cccccccc，申请空间前四个元素值也是固定的。当用户释放动态申请的内存时，VS会检测这些位置的值是否仍均为cc，如果是，则认为用户没有越界；如果任何一处不是，则认为用户越界。所以修改已超过检测范围的arr[14]、arr[15]不会报错，但修改arr[10]-arr[13]则会引起VS报错。如果用户不释放动态申请的内存或只读不写，无论怎么越界均不会报错。



# §. 关于动态内存申请后越界访问的深度讨论

★ 如何判断普通数组的越界访问（C方式，**注意源程序后缀为.c**） **int数组**

**VS2022 x86/Release**

①②全部注释

```
Microsoft Visual Studio 调试控制台
arr[10]=
addr:00B1FC80
00B1FC70:a12108
00B1FC74:a12108
00B1FC78:1152710
00B1FC7C:11599e0
00B1FC80:00
00B1FC84:01
00B1FC88:02
00B1FC8C:03
00B1FC90:04
00B1FC94:05
00B1FC98:06
00B1FC9C:07
00B1FCA0:08
00B1FCA4:09
00B1FCA8:a1131f
00B1FCAC:ac6ef449
00B1FCB0:b1fcf8
00B1FCB4:a11297
00B1FCB8:01
00B1FCBC:11599e0
```

①放开，②注释

```
Microsoft Visual Studio 调试控制台
arr[10]=%
addr:00A5FAB0
00A5FAA0:b52108
00A5FAA4:b52108
00A5FAA8:10d2710
00A5FAAC:10d9c20
00A5FAB0:00
00A5FAB4:01
00A5FAB8:02
00A5FABC:03
00A5FAC0:04
00A5FAC4:05
00A5FAC8:06
00A5FACC:07
00A5FAD0:08
00A5FAD4:09
00A5FAD8:61
00A5FADC:8d22cc98
00A5FAE0:a5fb28
00A5FAE4:b5129d
00A5FAE8:01
00A5FAEC:10d9c20
```

①②全部放开

```
Microsoft Visual Studio 调试控制台
arr[10]=%
addr:007CF7A4
007CF794:3d2108
007CF798:3d2108
007CF79C:bc2710
007CF7A0:bc0d18
007CF7A4:00
007CF7A8:01
007CF7AC:02
007CF7B0:03
007CF7B4:04
007CF7B8:05
007CF7BC:06
007CF7C0:07
007CF7C4:08
007CF7C8:09
007CF7CC:cccccccc
007CF7D0:55c88276
007CF7D4:7cf81c
007CF7D8:3d129d
007CF7DC:01
007CF7E0:bc0d18
```

VS2022 x86/Release模式int数组其前、后邻近位置无固定初始化值，对于越界的读写不敏感，对内存区域的越界读写都不会报错。但越界的写操作无效，无法修改越界空间的值。



# §. 关于动态内存申请后越界访问的深度讨论

★ 如何判断普通数组的越界访问（C方式，注意源程序后缀为.c） int数组

Dev 32bit-Release | Dev 32bit-Debug

①②全部注释

①放开，②注释

①②全部放开

注释arr[15]的赋值

```
D:\visual studio\  
arr[10]=*  
addr:0065FEA0  
0065FE90:40400c  
0065FE94:65fe90  
0065FE98:65fe90  
0065FE9C:4014ef  
0065FEA0:00  
0065FEA4:01  
0065FEA8:02  
0065FEAC:03  
0065FEB0:04  
0065FEB4:05  
0065FEB8:06  
0065FEBc:07  
0065FEC0:08  
0065FEC4:09  
0065FEC8:0a  
0065FECC:0a  
0065FED0:2a  
0065FED4:d01504  
0065FED8:41  
0065FEDC:42  
return value 3221225477
```

```
D:\visual studio\  
arr[10]=*  
addr:0065FEA0  
0065FE90:40400c  
0065FE94:65fe90  
0065FE98:65fe90  
0065FE9C:4014ef  
0065FEA0:00  
0065FEA4:01  
0065FEA8:02  
0065FEAC:03  
0065FEB0:04  
0065FEB4:05  
0065FEB8:06  
0065FEBc:07  
0065FEC0:08  
0065FEC4:09  
0065FEC8:0a  
0065FECC:0a  
0065FED0:2a  
0065FED4:d01504  
0065FED8:41  
0065FEDC:42  
return value 3221225477
```

```
D:\visual studio\  
arr[10]=*  
addr:0065FEA0  
0065FE90:40400c  
0065FE94:65fe90  
0065FE98:65fe90  
0065FE9C:4014ef  
0065FEA0:00  
0065FEA4:01  
0065FEA8:02  
0065FEAC:03  
0065FEB0:04  
0065FEB4:05  
0065FEB8:06  
0065FEBc:07  
0065FEC0:08  
0065FEC4:09  
0065FEC8:0a  
0065FECC:0a  
0065FED0:2a  
0065FED4:9d1504  
0065FED8:41  
0065FEDC:401386  
return value 0
```

arr[10] = 'a';  
arr[14] = 'A';  
//arr[15] = 'B';  
arr[10] = 0xffffffff;

Dev 32bit-Release/Debug模式得到的运行结果一致。

两种环境int数组前后紧邻位置无固定初始化值。对于越界的写不会弹窗报错，且除了arr[10]位置固定为0a，其余位置均可以被成功修改。但对于arr[15]被修改的情况会出现返回代码错误的情况，如果不修改arr[15]的值，则返回代码正常。



# §. 关于动态内存申请后越界访问的深度讨论

★ 如何判断普通数组的越界访问（C方式，**注意源程序后缀为.c**） **int数组**

**Linux**

①②全部注释

```
arr[10]=  
addr:0xffffd8a71e30  
0xffffd8a71e20:d8a71e60  
0xffffd8a71e24:ffff  
0xffffd8a71e28:9fdc4384  
0xffffd8a71e2c:ffff  
0xffffd8a71e30:00  
0xffffd8a71e34:01  
0xffffd8a71e38:02  
0xffffd8a71e3c:03  
0xffffd8a71e40:04  
0xffffd8a71e44:05  
0xffffd8a71e48:06  
0xffffd8a71e4c:07  
0xffffd8a71e50:08  
0xffffd8a71e54:09  
0xffffd8a71e58:0a  
0xffffd8a71e5c:0a  
0xffffd8a71e60:00  
0xffffd8a71e64:00  
0xffffd8a71e68:41  
0xffffd8a71e6c:42
```

①放开，②注释

```
arr[10]=  
addr:0xfffff719d520  
0xfffff719d510:f719d550  
0xfffff719d514:ffff  
0xfffff719d518:a5834384  
0xfffff719d51c:ffff  
0xfffff719d520:00  
0xfffff719d524:01  
0xfffff719d528:02  
0xfffff719d52c:03  
0xfffff719d530:04  
0xfffff719d534:05  
0xfffff719d538:06  
0xfffff719d53c:07  
0xfffff719d540:08  
0xfffff719d544:09  
0xfffff719d548:0a  
0xfffff719d54c:0a  
0xfffff719d550:00  
0xfffff719d554:00  
0xfffff719d558:41  
0xfffff719d55c:42
```

①②全部放开

```
arr[10]=  
addr:0xffffd6eea930  
0xffffd6eea920:d6eea960  
0xffffd6eea924:ffff  
0xffffd6eea928:b75f4384  
0xffffd6eea92c:ffff  
0xffffd6eea930:00  
0xffffd6eea934:01  
0xffffd6eea938:02  
0xffffd6eea93c:03  
0xffffd6eea940:04  
0xffffd6eea944:05  
0xffffd6eea948:06  
0xffffd6eea94c:07  
0xffffd6eea950:08  
0xffffd6eea954:09  
0xffffd6eea958:0a  
0xffffd6eea95c:0a  
0xffffd6eea960:00  
0xffffd6eea964:00  
0xffffd6eea968:41  
0xffffd6eea96c:42
```

其他测试用例

```
arr[10]=  
addr:0xffffc4a932e0  
0xffffc4a932d0:c4a93310  
0xffffc4a932d4:ffff  
0xffffc4a932d8:9dd14384  
0xffffc4a932dc:01  
0xffffc4a932e0:00  
0xffffc4a932e4:01  
0xffffc4a932e8:01  
0xffffc4a932ec:03  
0xffffc4a932f0:04  
0xffffc4a932f4:05  
0xffffc4a932f8:06  
0xffffc4a932fc:07  
0xffffc4a93300:08  
0xffffc4a93304:09  
0xffffc4a93308:0a  
0xffffc4a9330c:0a  
0xffffc4a93310:01  
0xffffc4a93314:00  
0xffffc4a93318:41  
0xffffc4a9331c:42  
段错误（核心已转储）
```

```
arr[10] = 'a';  
arr[12] = 1;  
arr[-1] = 1;  
arr[2] = 1;  
arr[14] = 'A';  
arr[15] = 'B';  
arr[16] = 1;  
arr[10] = 0xcccccccc;
```

```
arr[10]=  
addr:0xffffe515d020  
0xffffe515d010:e515d050  
0xffffe515d014:ffff  
0xffffe515d018:89044384  
0xffffe515d01c:ffff  
0xffffe515d020:00  
0xffffe515d024:01  
0xffffe515d028:01  
0xffffe515d02c:03  
0xffffe515d030:04  
0xffffe515d034:05  
0xffffe515d038:06  
0xffffe515d03c:07  
0xffffe515d040:08  
0xffffe515d044:09  
0xffffe515d048:0a  
0xffffe515d04c:0a  
0xffffe515d050:01  
0xffffe515d054:00  
0xffffe515d058:41  
0xffffe515d05c:42
```

```
arr[10] = 'a';  
arr[12] = 1;  
//arr[-1] = 1;  
arr[2] = 1;  
arr[14] = 'A';  
arr[15] = 'B';  
arr[16] = 1;  
arr[10] = 0xcccccccc;
```

Linux系统int数组对于越界的读写不敏感，但对于其后的紧邻四字节空间的写操作是无效的，对于四字节后的空间又可以进行读写。但arr[10]位置固定为0a，arr[-4]到arr[-1]之间的数据无法被修改，写操作会报错。





## §. 关于动态内存申请后越界访问的深度讨论

★ 如何判断普通数组的越界访问（C++方式，注意源程序后缀为.cpp） char数组

```
#define _CRT_SECURE_NO_WARNINGS
#include <iostream>
#include <cstring>
using namespace std;
int main()
{
    char arr[10] = "12345678";
    ① arr[10] = 'a';    //此句越界
    arr[14] = 'A';    //此句越界
    arr[15] = 'B';    //此句越界
    ② arr[10] = '\xcc';
    cout << "addr:" << hex << (void*)(arr) << endl;
    for (int i = -4; i < 16; i++)
        cout << hex << (void*)(arr + i) << ":" <<
int(arr[i]) << endl;
    return 0;
}
```

在理解P.1/P.2的情况下，自行构造相似的程序，来观察数组越界后的内存表现，并验证与动态申请是否相似

- 1、①②全部注释，观察运行结果
- 2、①放开，②注释，观察运行结果
- 3、①②全部放开，观察运行结果

要求：

- 1、数组用 char a[10]; 形式
- 2、数组用 int a[10]; 形式
- 3、测试程序在下面五种环境下运行  
VS2022 x86/Debug  
VS2022 x86/Release  
Dev 32bit-Debug  
Dev 32bit-Release  
Linux
- 4、每种讨论的结果可截图+文字说明，如果几种环境的结果一致，用一个环境的截图+文字说明即可(可加页)



# §. 关于动态内存申请后越界访问的深度讨论

★ 如何判断普通数组的越界访问（C++方式，**注意源程序后缀为.cpp**） char数组

VS2022 x86/Debug

①②全部注释

```
Microsoft Visual Studio 调试控制台
addr: 00CFF7BC
00CFF7B8: ffffffffcc
00CFF7B9: ffffffffcc
00CFF7BA: ffffffffcc
00CFF7BB: ffffffffcc
00CFF7BC: 30
00CFF7BD: 31
00CFF7BE: 32
00CFF7BF: 33
00CFF7C0: 34
00CFF7C1: 35
00CFF7C2: 36
00CFF7C3: 37
00CFF7C4: 38
00CFF7C5: 0
00CFF7C6: ffffffffcc
00CFF7C7: ffffffffcc
00CFF7C8: ffffffffcc
00CFF7C9: ffffffffcc
00CFF7CA: 41
00CFF7CB: 42
```

①放开，②注释

```
Microsoft Visual Studio 调试控制台
addr: 0059FD4C
0059FD48: ffffffffcc
0059FD49: ffffffffcc
0059FD4A: ffffffffcc
0059FD4B: ffffffffcc
0059FD4C: 30
0059FD4D: 31
0059FD4E: 32
0059FD4F: 33
0059FD50: 34
0059FD51: 35
0059FD52: 36
0059FD53: 37
0059FD54: 38
0059FD55: 0
0059FD56: 61
0059FD57: ffffffffcc
0059FD58: ffffffffcc
0059FD59: ffffffffcc
0059FD5A: 41
0059FD5B: 42
```

Microsoft Visual C++ Runtime Library

Debug Error!

Program: D:\visual studio\test\Debug\test.exe  
Module: D:\visual studio\test\Debug\test.exe  
File:  
Run-Time Check #2 - Stack around the variable 'var' was corrupted.  
(Press Retry to debug the application)

中止(A) 继续(C) 忽略(I)

①②全部放开

```
Microsoft Visual Studio 调试控制台
addr: 0053F8BC
0053F8B8: ffffffffcc
0053F8B9: ffffffffcc
0053F8BA: ffffffffcc
0053F8BB: ffffffffcc
0053F8BC: 31
0053F8BD: 32
0053F8BE: 33
0053F8BF: 34
0053F8C0: 35
0053F8C1: 36
0053F8C2: 37
0053F8C3: 38
0053F8C4: 0
0053F8C5: 0
0053F8C6: ffffffffcc
0053F8C7: ffffffffcc
0053F8C8: ffffffffcc
0053F8C9: ffffffffcc
0053F8CA: 41
0053F8CB: 42
```

VS2022 Debug模式int数组情况下，空间后紧邻元素arr[10]的值是cccccccc，申请空间前四个元素值也是固定的。当用户释放动态申请的内存时，VS会检测这些位置的值是否仍均为cc，如果是，则认为用户没有越界；如果任何一处不是，则认为用户越界。所以修改已超过检测范围的arr[14]、arr[15]不会报错，但修改arr[10]-arr[13]则会引起VS报错。如果用户不释放动态申请的内存或只读不写，无论怎么越界均不会报错。



# §. 关于动态内存申请后越界访问的深度讨论

★ 如何判断普通数组的越界访问（C++方式，**注意源程序后缀为.cpp**）**char**数组

VS2022 x86/Release

①②全部注释

```
Microsoft Visual Studio 调试控制台
addr:010FFEAC
010FFEAC:ffffffe8
010FFEAD:ffffff9c
010FFEEA:2a
010FFEEB:1
010FFEEC:31
010FFEEF:32
010FFEEA:33
010FFEEF:34
010FFEB0:35
010FFEB1:36
010FFEB2:37
010FFEB3:38
010FFEB4:0
010FFEB5:0
010FFEB6:fffffffb
010FFEB7:75
010FFEB8:54
010FFEB9:16
010FFEBA:ffffffcb
010FFEBB:0
```

①放开，②注释

```
Microsoft Visual Studio 调试控制台
addr:0043FA4C
0043FA48:48
0043FA49:10
0043FA4A:ffffffb2
0043FA4B:0
0043FA4C:31
0043FA4D:32
0043FA4E:33
0043FA4F:34
0043FA50:35
0043FA51:36
0043FA52:37
0043FA53:38
0043FA54:0
0043FA55:0
0043FA56:61
0043FA57:0
0043FA58:52
0043FA59:ffffff8b
0043FA5A:fffffffb
0043FA5B:75
```

①②全部放开

```
Microsoft Visual Studio 调试控制台
addr:009AFE8C
009AFE88:ffffff80
009AFE89:ffffff9d
009AFE8A:ffffffdc
009AFE8B:0
009AFE8C:31
009AFE8D:32
009AFE8E:33
009AFE8F:34
009AFE90:35
009AFE91:36
009AFE92:37
009AFE93:38
009AFE94:0
009AFE95:0
009AFE96:ffffffcc
009AFE97:0
009AFE98:52
009AFE99:ffffff8b
009AFE9A:fffffffb
009AFE9B:75
```

VS2022 x86/Release模式char数组其前、后邻近位置无固定初始化值。对于越界的读写不敏感，对内存区域的越界读写都不会报错。但越界的写操作无效，无法修改越界空间的值。arr[10]位置可以进行越界写。



# §. 关于动态内存申请后越界访问的深度讨论

★ 如何判断普通数组的越界访问（C++方式，**注意源程序后缀为.cpp**） **char**数组

Dev 32bit-Release | Dev 32bit-Debug

①②全部注释

①放开，②注释

①②全部放开

```
D:\visual studio\  
addr:0x78feb2  
0x78feae:0  
0x78feaf:0  
0x78feb0:2a  
0x78feb1:0  
0x78feb2:31  
0x78feb3:32  
0x78feb4:33  
0x78feb5:34  
0x78feb6:35  
0x78feb7:36  
0x78feb8:37  
0x78feb9:38  
0x78feba:0  
0x78febb:0  
0x78febc:a  
0x78febd:0  
0x78febe:0  
0x78febf:0  
0x78fec0:41  
0x78fec1:42
```

```
D:\visual studio\  
addr:0x78feb2  
0x78feae:0  
0x78feaf:0  
0x78feb0:2a  
0x78feb1:0  
0x78feb2:31  
0x78feb3:32  
0x78feb4:33  
0x78feb5:34  
0x78feb6:35  
0x78feb7:36  
0x78feb8:37  
0x78feb9:38  
0x78feba:0  
0x78febb:0  
0x78febc:a  
0x78febd:0  
0x78febe:0  
0x78febf:0  
0x78fec0:41  
0x78fec1:42
```

Dev 32bit-Release/Debug模式得到的运行结果一致。

两种环境char数组前后紧邻位置无固定初始化值。对于越界的读写不敏感，对内存区域的越界读写不会报错。对于char数组区域前的内存空间可以成功地进行读写，但对于其后的紧邻四字节空间的写操作是无效的，对于四字节后的空间又可以成功地进行读写。





# §. 关于动态内存申请后越界访问的深度讨论

★ 如何判断普通数组的越界访问（C++方式，**注意源程序后缀为.cpp**） char数组

Linux

①②全部注释

```
addr:0xfffff2d9d170
0xfffff2d9d16c:ff
0xfffff2d9d16d:ff
0xfffff2d9d16e:0
0xfffff2d9d16f:0
0xfffff2d9d170:31
0xfffff2d9d171:32
0xfffff2d9d172:33
0xfffff2d9d173:34
0xfffff2d9d174:35
0xfffff2d9d175:36
0xfffff2d9d176:37
0xfffff2d9d177:38
0xfffff2d9d178:0
0xfffff2d9d179:0
0xfffff2d9d17a:0
0xfffff2d9d17b:0
0xfffff2d9d17c:c
0xfffff2d9d17d:0
0xfffff2d9d17e:0
0xfffff2d9d17f:0
```

①放开，②注释

```
addr:0xfffff8a84930
0xfffff8a8492c:ff
0xfffff8a8492d:ff
0xfffff8a8492e:0
0xfffff8a8492f:0
0xfffff8a84930:31
0xfffff8a84931:32
0xfffff8a84932:33
0xfffff8a84933:34
0xfffff8a84934:35
0xfffff8a84935:36
0xfffff8a84936:37
0xfffff8a84937:38
0xfffff8a84938:0
0xfffff8a84939:0
0xfffff8a8493a:61
0xfffff8a8493b:0
0xfffff8a8493c:c
0xfffff8a8493d:0
0xfffff8a8493e:0
0xfffff8a8493f:0
```

①②全部放开

```
addr:0xffffd5bc9c40
0xffffd5bc9c3c:ff
0xffffd5bc9c3d:ff
0xffffd5bc9c3e:0
0xffffd5bc9c3f:0
0xffffd5bc9c40:31
0xffffd5bc9c41:32
0xffffd5bc9c42:33
0xffffd5bc9c43:34
0xffffd5bc9c44:35
0xffffd5bc9c45:36
0xffffd5bc9c46:37
0xffffd5bc9c47:38
0xffffd5bc9c48:0
0xffffd5bc9c49:0
0xffffd5bc9c4a:cc
0xffffd5bc9c4b:0
0xffffd5bc9c4c:c
0xffffd5bc9c4d:0
0xffffd5bc9c4e:0
0xffffd5bc9c4f:0
```

Linux系统char数组对于越界的读写不敏感，但对于其后的紧邻四字节空间的写操作是无效的，对于四字节后的空间又可以进行读写。



## §. 关于动态内存申请后越界访问的深度讨论

★ 如何判断普通数组的越界访问（C++方式，注意源程序后缀为.cpp） int数组

```
#define _CRT_SECURE_NO_WARNINGS
#include <iostream>
#include <cstring>
using namespace std;
int main()
{
    int arr[10] = { 0, 1, 2, 3, 4, 5, 6, 7, 8 };
    ① arr[10] = 10;    //此句越界
    arr[11] = 11;    //此句越界
    arr[14] = 14;    //此句越界
    arr[15] = 15;    //此句越界
    ② arr[10] = 0xc0000000;
    cout << "addr:" << hex << (void*)(arr) << endl;
    for (int i = -4; i < 16; i++)
        cout << hex << (void*)(arr + i) << ":" <<
int(arr[i]) << endl;
    return 0;
}
```

在理解P. 1/P. 2的情况下，自行构造相似的程序，来观察数组越界后的内存表现，并验证与动态申请是否相似

- 1、①②全部注释，观察运行结果
- 2、①放开，②注释，观察运行结果
- 3、①②全部放开，观察运行结果

要求：

- 1、数组用 char a[10]; 形式
- 2、数组用 int a[10]; 形式
- 3、测试程序在下面五种环境下运行  
VS2022 x86/Debug  
VS2022 x86/Release  
Dev 32bit-Debug  
Dev 32bit-Release  
Linux
- 4、每种讨论的结果可截图+文字说明，如果几种环境的结果一致，用一个环境的截图+文字说明即可(可加页)



# §. 关于动态内存申请后越界访问的深度讨论

★ 如何判断普通数组的越界访问（C++方式，**注意源程序后缀为.cpp**） **int**数组

VS2022 x86/Debug

①②全部注释

```
Microsoft Visual Studio 调试控制台
addr:007AFC1C
007AFC0C:cccccccc
007AFC10:fffffffffd
007AFC14:cccccccc
007AFC18:cccccccc
007AFC1C:0
007AFC20:1
007AFC24:2
007AFC28:3
007AFC2C:4
007AFC30:5
007AFC34:6
007AFC38:7
007AFC3C:8
007AFC40:0
007AFC44:cccccccc
007AFC48:b
007AFC4C:7afc6c
007AFC50:652ec3
007AFC54:e
007AFC58:f
代码为 -1073740791 (0xc0000409)
```

①放开，②注释

```
Microsoft Visual Studio 调试控制台
addr:00FBF83C
00FBF82C:cccccccc
00FBF830:fffffffffd
00FBF834:cccccccc
00FBF838:cccccccc
00FBF83C:0
00FBF840:1
00FBF844:2
00FBF848:3
00FBF84C:4
00FBF850:5
00FBF854:6
00FBF858:7
00FBF85C:8
00FBF860:0
00FBF864:a
00FBF868:b
00FBF86C:fbf88c
00FBF870:392ec3
00FBF874:e
00FBF878:f
```

Microsoft Visual Studio - Runtime Library

Debug Error!

Program: D:\visual studio\test\Debug\test.exe  
Module: D:\visual studio\test\Debug\test.exe  
File:

Run-Time Check Failure #2 - Stack around the variable 'arr' was corrupted.

(Press F5 to debug the application)

中止(A) 重试(R) 忽略(I)

①②全部放开

```
Microsoft Visual Studio 调试控制台
addr:00F5F99C
00F5F98C:cccccccc
00F5F990:fffffffffd
00F5F994:cccccccc
00F5F998:cccccccc
00F5F99C:0
00F5F9A0:1
00F5F9A4:2
00F5F9A8:3
00F5F9AC:4
00F5F9B0:5
00F5F9B4:6
00F5F9B8:7
00F5F9BC:8
00F5F9C0:0
00F5F9C4:cccccccc
00F5F9C8:838da192
00F5F9CC:f5f9ec
00F5F9D0:ba2ef3
00F5F9D4:e
00F5F9D8:f
代码为 0 (0x0)
```

VS2022 Debug模式int数组情况下，将申请空间前、后的邻近四字节初始化为cc。当用户释放动态申请的内存时，VS会检测这些位置的值是否仍均为cc，如果是，则认为用户没有越界；如果任何一处不是，则认为用户越界。所以修改已超过检测范围的arr[14]、arr[15]不会报错，但修改arr[10]-arr[13]则会引起VS代码错误。如果用户不释放动态申请的内存或只读不写，无论怎么越界均不会报错。



# §. 关于动态内存申请后越界访问的深度讨论

★ 如何判断普通数组的越界访问（C++方式，**注意源程序后缀为.cpp**） **int**数组

**VS2022 x86/Release**

①②全部注释

```
Microsoft Visual Studio 调试控制台
addr:0092F810
0092F800:821098
0092F804:8213f0
0092F808:ab0a08
0092F80C:aa0e20
0092F810:0
0092F814:1
0092F818:2
0092F81C:3
0092F820:4
0092F824:5
0092F828:6
0092F82C:7
0092F830:8
0092F834:0
0092F838:821674
0092F83C:d245ba5c
0092F840:92f888
0092F844:8215ec
0092F848:1
0092F84C:aa0e20
```

①放开，②注释

```
Microsoft Visual Studio 调试控制台
addr:012FFC70
012FFC60:a710a0
012FFC64:a713f0
012FFC68:1502820
012FFC6C:1509d38
012FFC70:0
012FFC74:1
012FFC78:2
012FFC7C:3
012FFC80:4
012FFC84:5
012FFC88:6
012FFC8C:7
012FFC90:8
012FFC94:0
012FFC98:a
012FFC9C:4d097e8
012FFCA0:a71674
012FFCA4:12ffcec
012FFCA8:a715ec
012FFCAC:1
```

①②全部放开

```
Microsoft Visual Studio 调试控制台
addr:010FFD50
010FFD40:ce10a0
010FFD44:ce13f0
010FFD48:13d14e0
010FFD4C:13d0e88
010FFD50:0
010FFD54:1
010FFD58:2
010FFD5C:3
010FFD60:4
010FFD64:5
010FFD68:6
010FFD6C:7
010FFD70:8
010FFD74:0
010FFD78:cccccccc
010FFD7C:d3884311
010FFD80:ce1674
010FFD84:10ffdcc
010FFD88:ce15ec
010FFD8C:1
```

VS2022 x86/Release模式int数组其前、后邻近位置无固定初始化值，对于越界的读写不敏感，对内存区域的越界读写都不会报错。但越界的写操作无效，无法修改越界空间的值。



# §. 关于动态内存申请后越界访问的深度讨论

★ 如何判断普通数组的越界访问（C++方式，注意源程序后缀为.cpp） int数组

Dev 32bit-Release | Dev 32bit-Debug

①②全部注释

①放开，②注释

①②全部放开

注释arr[11]的赋值

```
D:\visual studio\
addr:0x78fe94
0x78fe84:4cf007
0x78fe88:78ffcc
0x78fe8c:7757e170
0x78fe90:bcd87c80
0x78fe94:0
0x78fe98:1
0x78fe9c:2
0x78fea0:3
0x78fea4:4
0x78fea8:5
0x78feac:6
0x78feb0:7
0x78feb4:8
0x78feb8:0
0x78febc:a
0x78fec0:b
0x78fec4:9e1504
0x78fec8:78ff68
0x78fecc:e
0x78fed0:f
return value 3221225477
```

```
D:\visual studio\
addr:0x78fe94
0x78fe84:4cf007
0x78fe88:78ffcc
0x78fe8c:7757e170
0x78fe90:a5f1dc8e
0x78fe94:0
0x78fe98:1
0x78fe9c:2
0x78fea0:3
0x78fea4:4
0x78fea8:5
0x78feac:6
0x78feb0:7
0x78feb4:8
0x78feb8:0
0x78febc:a
0x78fec0:b
0x78fec4:ce1504
0x78fec8:78ff68
0x78fecc:e
0x78fed0:f
return value 0
```

```
D:\visual studio\
addr:0x78fe94
0x78fe84:4cf007
0x78fe88:78ffcc
0x78fe8c:7757e170
0x78fe90:6c6d70a
0x78fe94:0
0x78fe98:1
0x78fe9c:2
0x78fea0:3
0x78fea4:4
0x78fea8:5
0x78feac:6
0x78feb0:7
0x78feb4:8
0x78feb8:0
0x78febc:a
0x78fec0:78fee0
0x78fec4:d21504
0x78fec8:78ff68
0x78fecc:e
0x78fed0:f
return value 0
```

```
arr[10] = 10;
//arr[11] = 11;
arr[14] = 14;
arr[15] = 15;
//arr[10] = 0xffffffff;
```

Dev 32bit-Release/Debug模式得到的运行结果一致。

两种环境int数组前后紧邻位置无固定初始化值。对于越界的写不会弹窗报错，且除了arr[10]位置固定为0a，其余位置均可以被成功修改。但对于arr[11]被修改的情况会出现返回代码错误的情况，如果不修改arr[15]的值，则返回代码正常。





# §. 关于动态内存申请后越界访问的深度讨论

★ 如何判断普通数组的越界访问（C++方式，**注意源程序后缀为.cpp**） **int**数组

## Linux

### ①②全部注释

```
addr:0xfffffcae18d0
0xfffffcae18c0:fcac1900
0xfffffcae18c4:ffff
0xfffffcae18c8:8df44384
0xfffffcae18cc:ffff
0xfffffcae18d0:0
0xfffffcae18d4:1
0xfffffcae18d8:2
0xfffffcae18dc:3
0xfffffcae18e0:4
0xfffffcae18e4:5
0xfffffcae18e8:6
0xfffffcae18ec:7
0xfffffcae18f0:8
0xfffffcae18f4:0
0xfffffcae18f8:0
0xfffffcae18fc:b
0xfffffcae1900:0
0xfffffcae1904:0
0xfffffcae1908:e
0xfffffcae190c:f
```

### ①放开，②注释

```
addr:0xfffffc37baa90
0xfffffc37baa80:c37baac0
0xfffffc37baa84:ffff
0xfffffc37baa88:83634384
0xfffffc37baa8c:ffff
0xfffffc37baa90:0
0xfffffc37baa94:1
0xfffffc37baa98:2
0xfffffc37baa9c:3
0xfffffc37baaa0:4
0xfffffc37baaa4:5
0xfffffc37baaa8:6
0xfffffc37baaac:7
0xfffffc37baab0:8
0xfffffc37baab4:0
0xfffffc37baab8:a
0xfffffc37baabc:b
0xfffffc37baac0:0
0xfffffc37baac4:0
0xfffffc37baac8:e
0xfffffc37baacc:f
```

### ①②全部放开

```
addr:0xfffffd2985d80
0xfffffd2985d70:d2985db0
0xfffffd2985d74:ffff
0xfffffd2985d78:90924384
0xfffffd2985d7c:ffff
0xfffffd2985d80:0
0xfffffd2985d84:1
0xfffffd2985d88:2
0xfffffd2985d8c:3
0xfffffd2985d90:4
0xfffffd2985d94:5
0xfffffd2985d98:6
0xfffffd2985d9c:7
0xfffffd2985da0:8
0xfffffd2985da4:0
0xfffffd2985da8:cccccccc
0xfffffd2985dac:b
0xfffffd2985db0:0
0xfffffd2985db4:0
0xfffffd2985db8:e
0xfffffd2985dbc:f
```

### 注释arr[11]的赋值

```
addr:0xffffff55c090
0xffffff55c080:ff55c0c0
0xffffff55c084:ffff
0xffffff55c088:905e4384
0xffffff55c08c:ffff
0xffffff55c090:0
0xffffff55c094:1
0xffffff55c098:2
0xffffff55c09c:3
0xffffff55c0a0:4
0xffffff55c0a4:5
0xffffff55c0a8:6
0xffffff55c0ac:7
0xffffff55c0b0:8
0xffffff55c0b4:0
0xffffff55c0b8:a
0xffffff55c0bc:b
0xffffff55c0c0:0
0xffffff55c0c4:0
0xffffff55c0c8:e
0xffffff55c0cc:f
```

```
arr[10] = 10;
//arr[11] = 11;
arr[14] = 14;
arr[15] = 15;
//arr[10] = 0xc0000000;
```

Linux系统int数组对于越界的读写敏感，对于arr[10]-arr[15]范围内的数据，如果被修改则会报错。对于arr[16]及以后的数据被修改是成功的，且不会报错。但arr[-4]到arr[-1]之间的数据无法被修改，是随机值。



# §. 关于动态内存申请后越界访问的深度讨论

★ 最后一页：仔细总结本作业（多种形式的测试程序/多个编译器环境/不同结论），谈谈你对内存越界访问的整体理解  
包括但不限于操作系统/编译器如何防范越界、你应该养成怎样的使用习惯来尽量防范越界

不同环境下数组越界表现的详细描述：

## 1. Visual Studio 2022

### •Debug模式：

- **越界检测**：在内存分配时，编译器会在数组前后插入一些保护字节（通常是特定值，如0xDEADBEEF）。当程序尝试访问这些保护区域时，调试器会检测到修改，并提示越界错误。
- **报错情况**：越界写入会导致错误报告，程序可能会中止；而越界读取则通常不会被捕捉到。

### •Release模式：

- **宽松性**：没有越界检查，可能会导致未定义行为。开发者需要确保手动管理数组的边界，错误可能不会立即显现，但在某些情况下可能导致崩溃或数据损坏。

## 2. Dev环境

### •动态内存申请：

- **读写操作**：对动态分配的内存进行越界访问通常不会产生错误，可能会访问到其他合法内存区域，导致不可预测的结果。

### •数组状态：

- **越界行为**：对于数组的越界写入，通常没有保护机制。可能会影响到堆栈或其他变量的值，但不会立即导致错误。某些特定的越界操作可能会引发运行时错误。

## 3. Linux环境

### •动态内存申请：

- **正常运行**：动态内存越界访问通常不会引发错误，可能读取到随机数据或写入其他合法内存，存在不稳定性。

### •数组状态：

- **字符数组**：越界访问不报错，依然可以进行读写操作。
- **整型数组**：如果越界访问在检测范围内可能会触发错误。但超出检测范围的访问通常不会报错。

4. 操作系统保护机制（内存保护）：现代操作系统支持内存保护机制，如分页和虚拟内存。这些机制可以防止程序直接访问不属于它的内存页。当程序尝试访问未分配或受保护的内存时，操作系统会引发异常，导致程序崩溃。这种机制确保了进程之间的内存隔离。

5. 编译器的静态检查（静态分析）：编译器在编译时通过代码分析，检测数组越界的潜在风险，可能发出警告或错误提示，提醒开发者注意。

6. 防范越界的方法：使用安全的标准库函数（如std::vector替代数组）。在Debug模式下开发和测试，以便及时捕获问题。尽可能申请足够大的连续内存空间。始终检查边界条件，尤其是在循环和迭代中。