§.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？

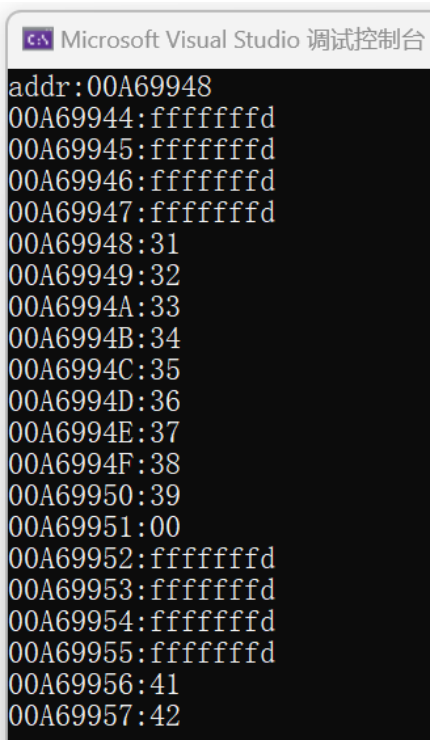
再观察下面四种环境下的运行结果：
VS2022 x86/Release
Dev 32bit-Debug
Dev 32bit-Release
Linux
每种讨论的结果可截图+文字说明，
如果几种环境的结果一致，用一个
环境的截图+文字说明即可(可加页)

§.关于动态内存申请后越界访问的深度讨论

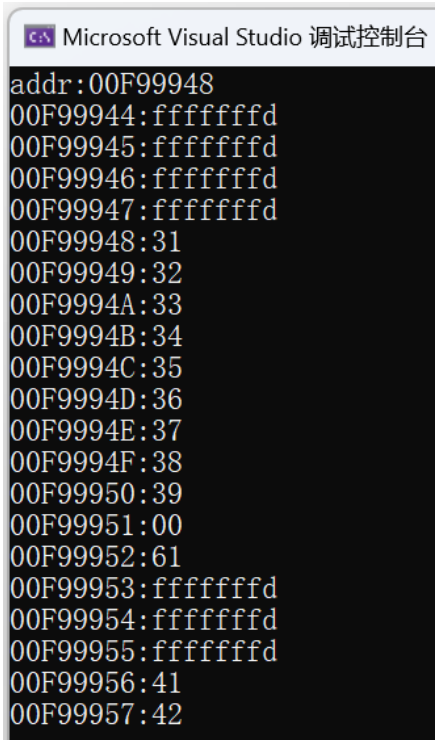
★ 如何判断动态申请越界（C方式，注意源程序后缀为.c）

VS2022 x86/Debug

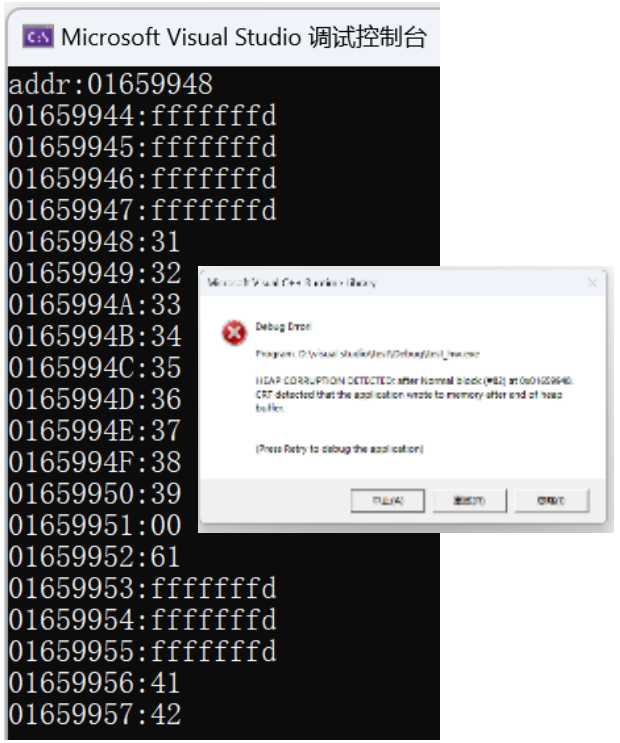
①②③全部注释



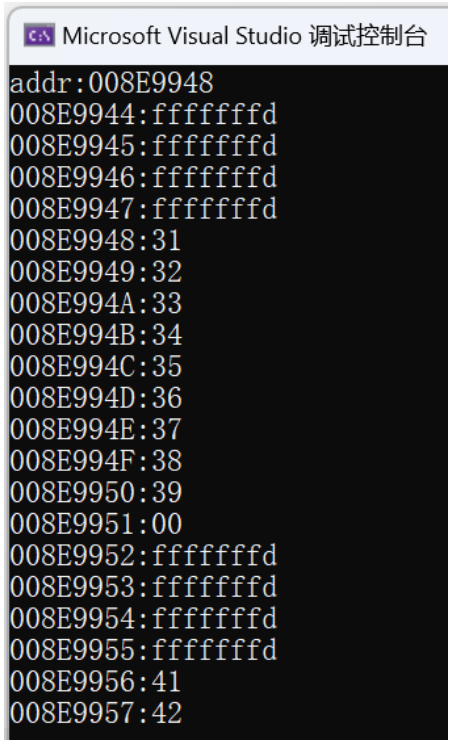
①放开，②③注释



①③放开，②注释



①②③全部放开



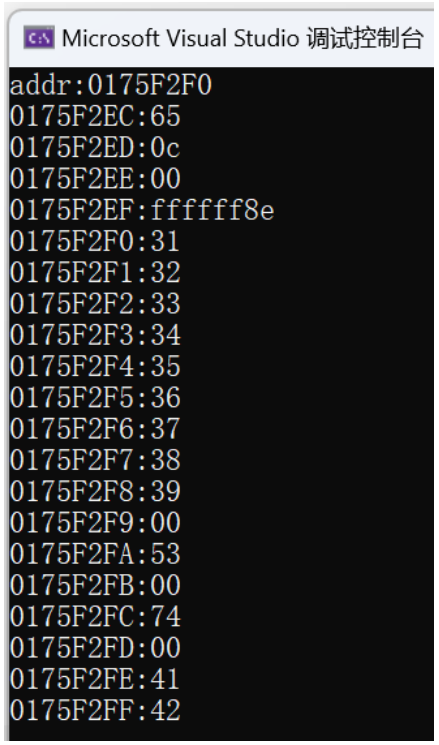
VS2022 Debug模式将用户申请的内存前、后四个字节内容置为fd。当用户释放动态申请的内存时，VS会检测这些位置的值是否仍均为fd，如果是，则认为用户没有越界；如果任何一处不是，则认为用户越界。所以修改已超过检测范围的p[14]、p[15]不会报错，但修改p[10]-p[13] 则会引起VS报错。如果用户不释放动态申请的内存或只读不写，无论怎么越界均不会报错。

§.关于动态内存申请后越界访问的深度讨论

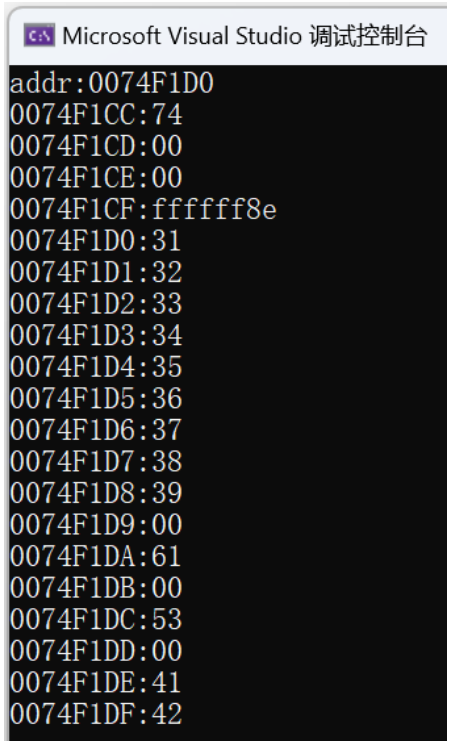
★ 如何判断动态申请越界（C方式，注意源程序后缀为.c）

VS2022 x86/Release

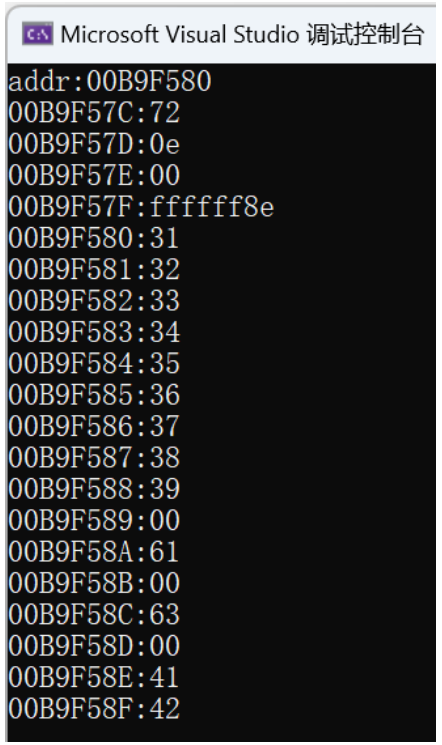
①②③全部注释



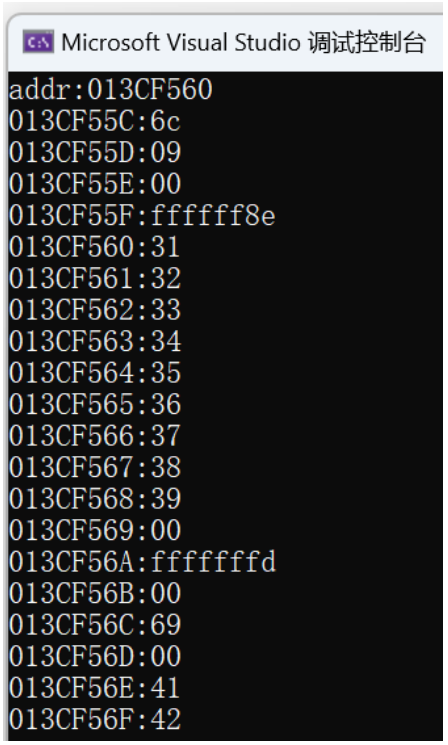
①放开，②③注释



①③放开，②注释



①②③全部放开



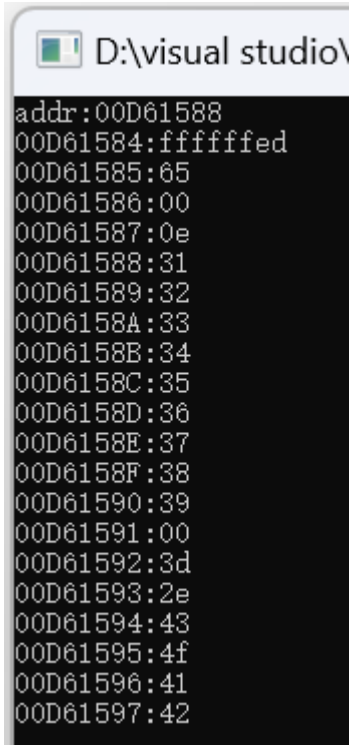
VS2022 x86/Release模式对于越界的读写不敏感，无论是否释放动态申请的空间，对内存区域的越界读写都可以实现且不会报错。

§.关于动态内存申请后越界访问的深度讨论

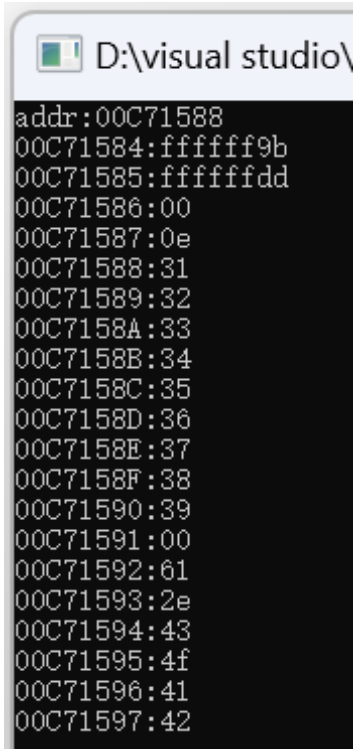
★ 如何判断动态申请越界（C方式，注意源程序后缀为.c）

Dev 32bit-Release | Dev 32bit-Debug

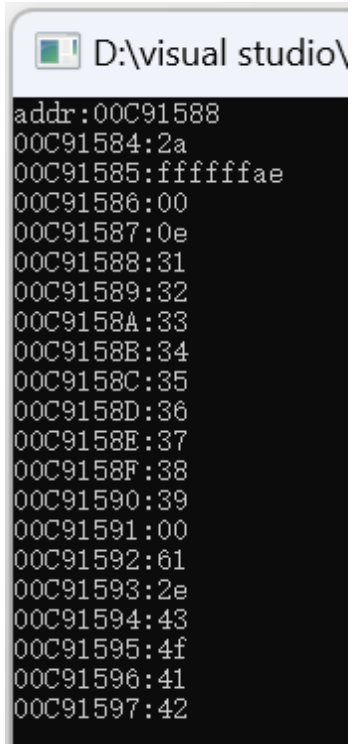
①②③全部注释



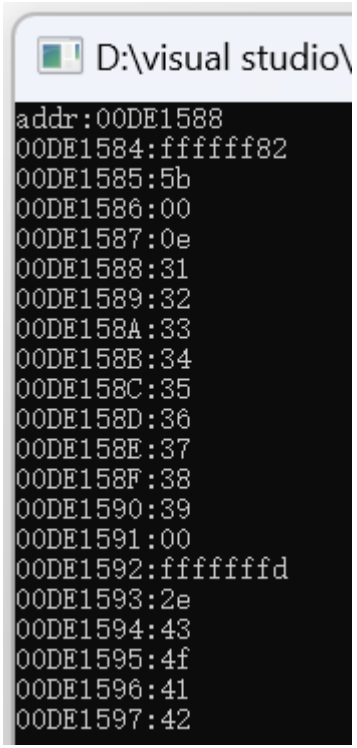
①放开，②③注释



①③放开，②注释



①②③全部放开



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:fffffffd
009D1C1D:fffffffd
009D1C1E:fffffffd
009D1C1F:fffffffd
009D1C20:31
009D1C21:32
009D1C22:33
009D1C23:34
009D1C24:35
009D1C25:36
009D1C26:37
009D1C27:38
009D1C28:39
009D1C29:0
009D1C2A:fffffffd
009D1C2B:fffffffd
009D1C2C:fffffffd
009D1C2D:fffffffd
009D1C2E:41
009D1C2F:42
```

①放开，②③注释

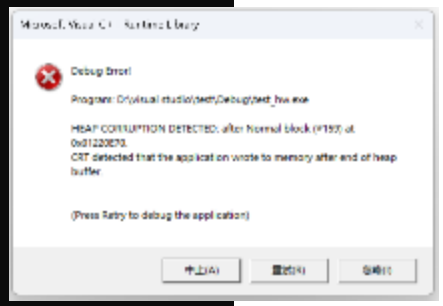
Microsoft Visual Studio 调试控制台

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

①③放开，②注释

Microsoft Visual Studio 调试控制台

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



①②③全部放开

Microsoft Visual Studio 调试控制台

```
addr:00D91B40
00D91B3C:fffffffd
00D91B3D:fffffffd
00D91B3E:fffffffd
00D91B3F:fffffffd
00D91B40:31
00D91B41:32
00D91B42:33
00D91B43:34
00D91B44:35
00D91B45:36
00D91B46:37
00D91B47:38
00D91B48:39
00D91B49:0
00D91B4A:fffffffd
00D91B4B:fffffffd
00D91B4C:fffffffd
00D91B4D:fffffffd
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
```

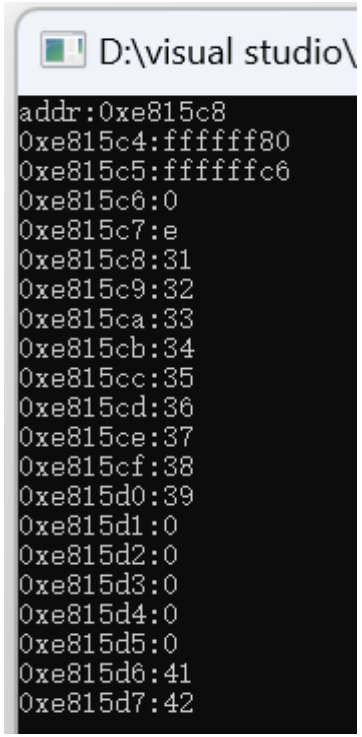
VS2022 x86/Release模式对于越界的读写不敏感，动态申请空间的前、后邻近位置均无固定初始化值，无论是否释放动态申请的空间，对内存区域的越界读写都可以实现且不会报错。

§.关于动态内存申请后越界访问的深度讨论

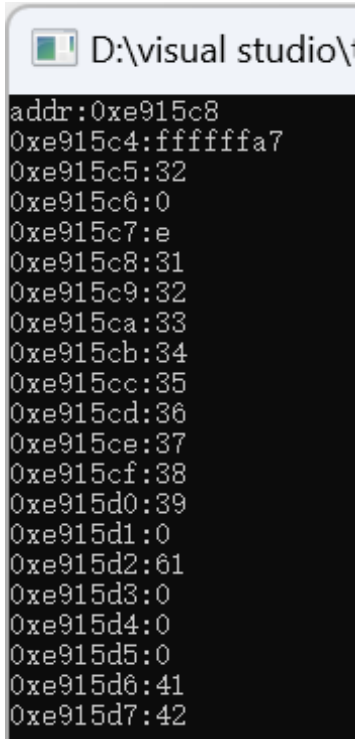
★ 如何判断动态申请越界（C++方式，注意源程序后缀为.cpp）

Dev 32bit-Release | Dev 32bit-Debug

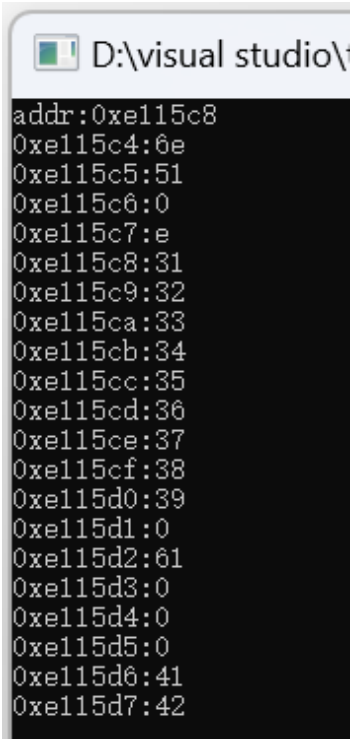
①②③全部注释



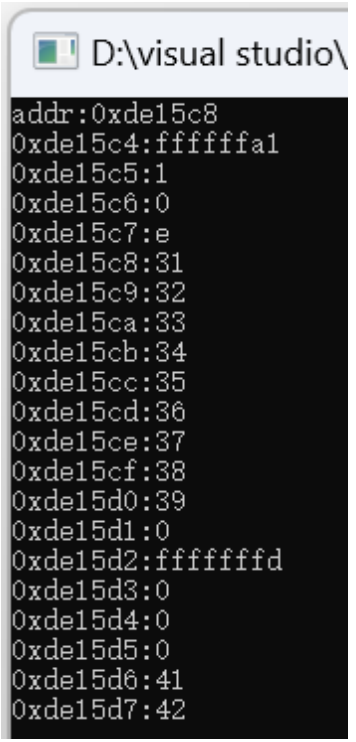
①放开，②③注释



①③放开，②注释



①②③全部放开



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

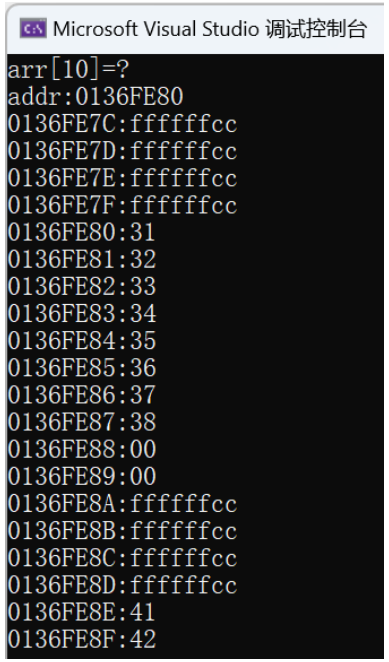
4、每种讨论的结果可截图+文字说明，如果几种环境的结果一致，用一个环境的截图+文字说明即可(可加页)

§.关于动态内存申请后越界访问的深度讨论

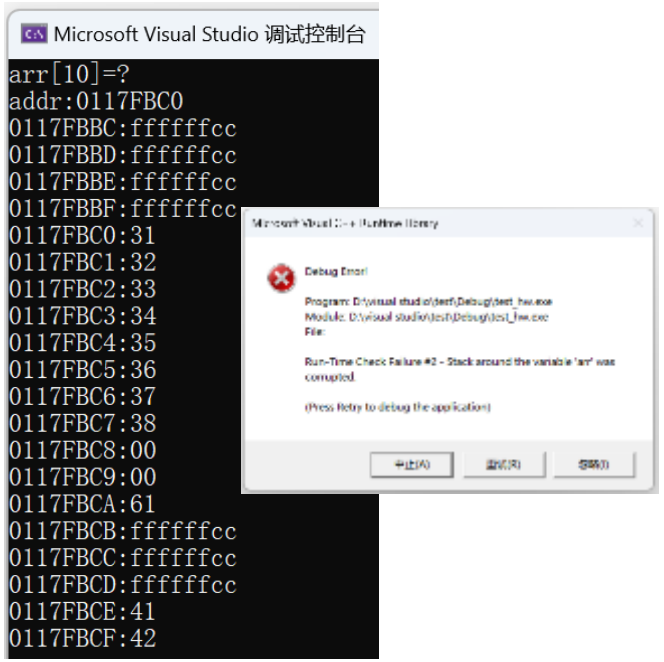
★ 如何判断普通数组的越界访问（C方式，注意源程序后缀为.c）　　char数组

VS2022 x86/Debug

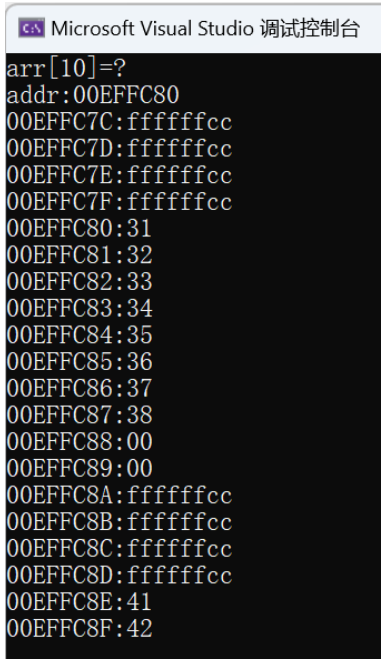
①②全部注释



①放开，②注释



①②全部放开



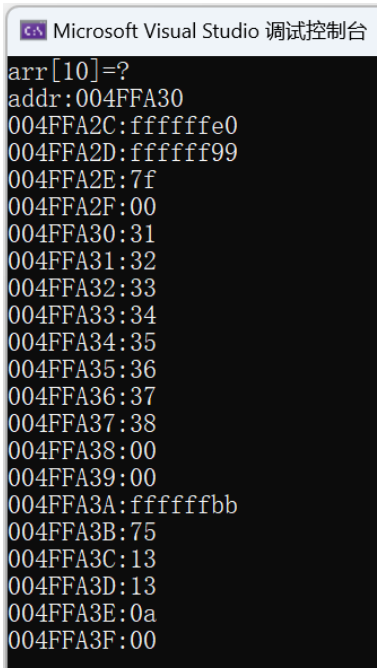
VS2022 Debug模式char数组将用户申请的内存前、后四个字节内容置为cc。当用户释放动态申请的内存时，VS会检测这些位置的值是否仍均为cc，如果是，则认为用户没有越界；如果任何一处不是，则认为用户越界。所以修改已超过检测范围的arr[14]、arr[15]不会报错，但修改arr[10]-arr[13] 则会引起VS报错。如果用户不释放动态申请的内存或只读不写，无论怎么越界均不会报错。

§.关于动态内存申请后越界访问的深度讨论

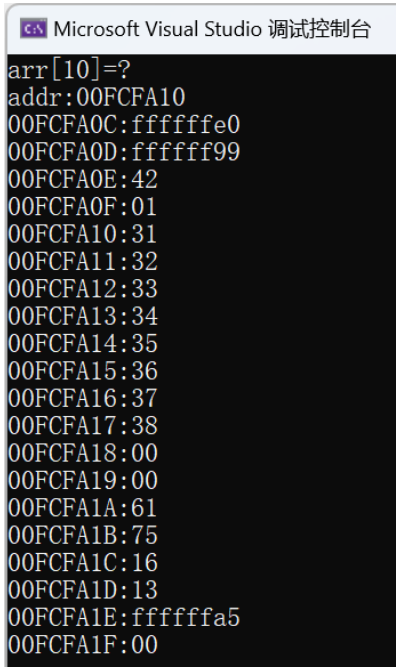
★ 如何判断普通数组的越界访问（C方式，注意源程序后缀为.c）　　char数组

VS2022 x86/Release

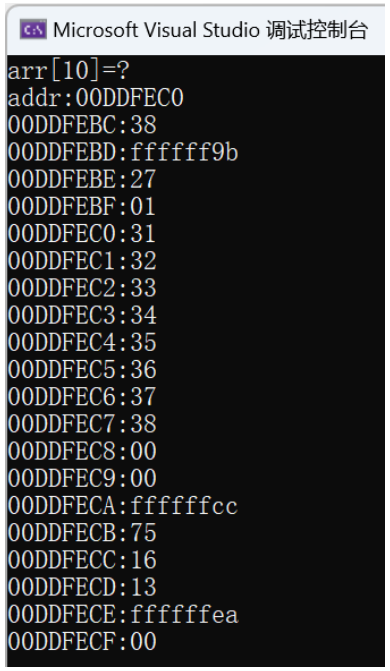
①②全部注释



①放开，②注释



①②全部放开



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

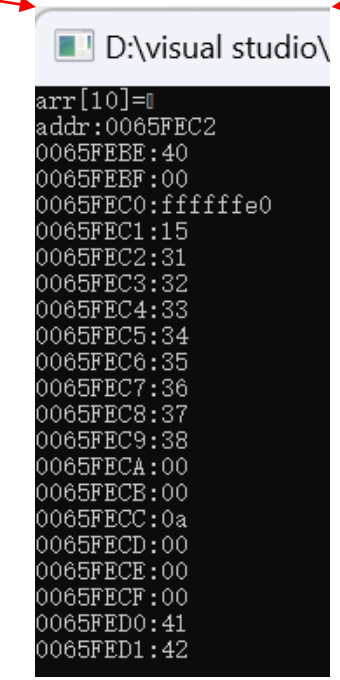
★ 如何判断普通数组的越界访问（C方式，注意源程序后缀为.c）　　char数组

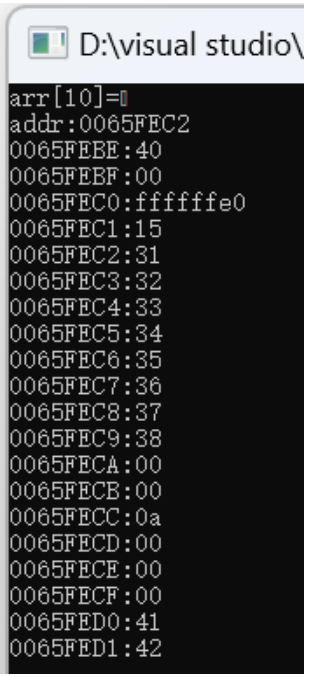
Dev 32bit-Release | Dev 32bit-Debug

①②全部注释

①放开，②注释

①②全部放开





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

```c
#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

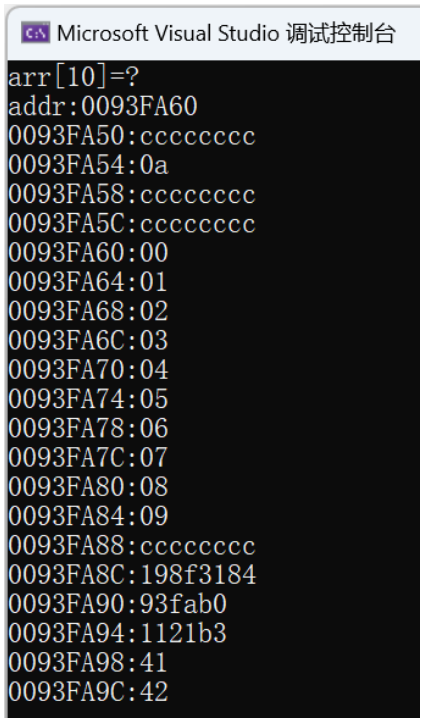
4、每种讨论的结果可截图+文字说明，如果几种环境的结果一致，用一个环境的截图+文字说明即可(可加页)

§.关于动态内存申请后越界访问的深度讨论

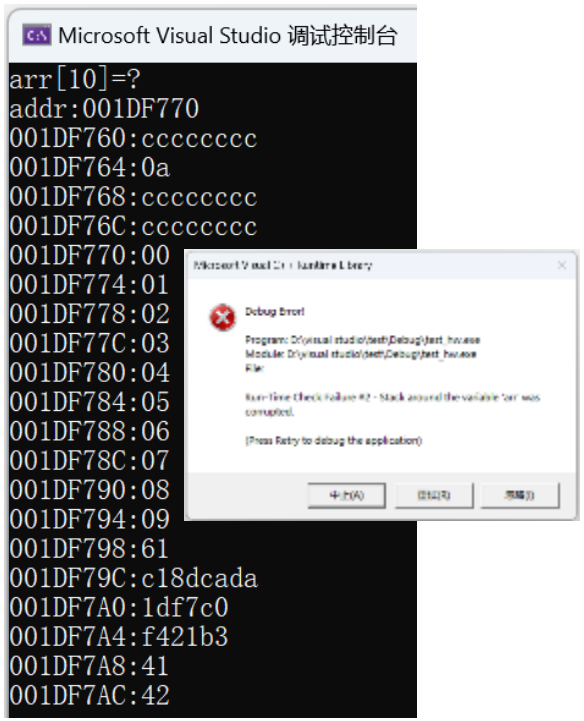
★ 如何判断普通数组的越界访问（C方式，注意源程序后缀为.c）　int数组

VS2022 x86/Debug

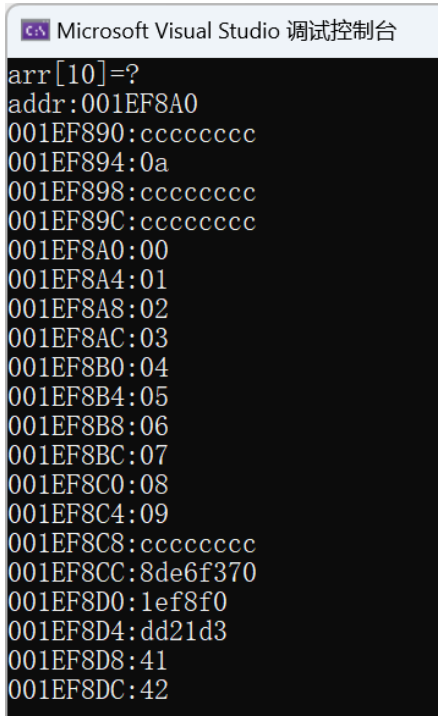
①②全部注释



①放开，②注释



①②全部放开



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
```

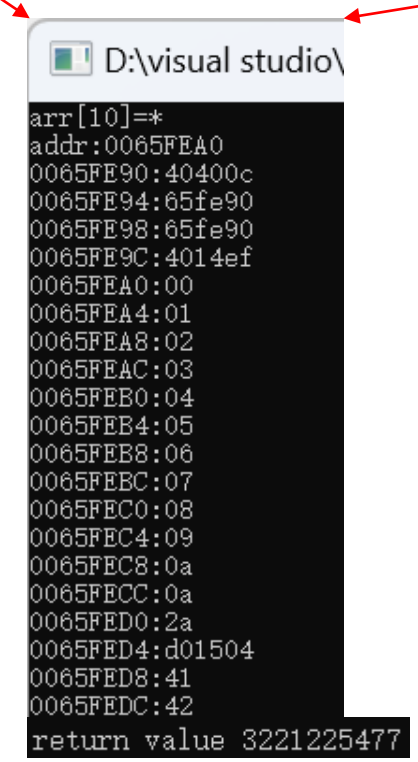
VS2022 x86/Release模式int数组其前、后邻近位置无固定初始化值，对于越界的读写不敏感，对内存区域的越界读写都不会报错。但越界的写操作无效，无法修改越界空间的值。

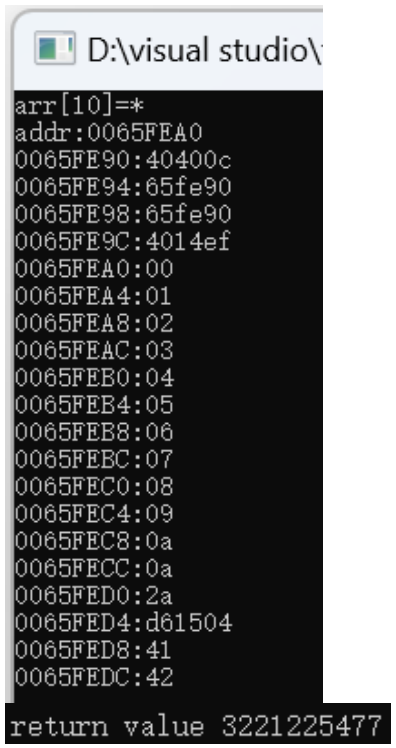
§.关于动态内存申请后越界访问的深度讨论

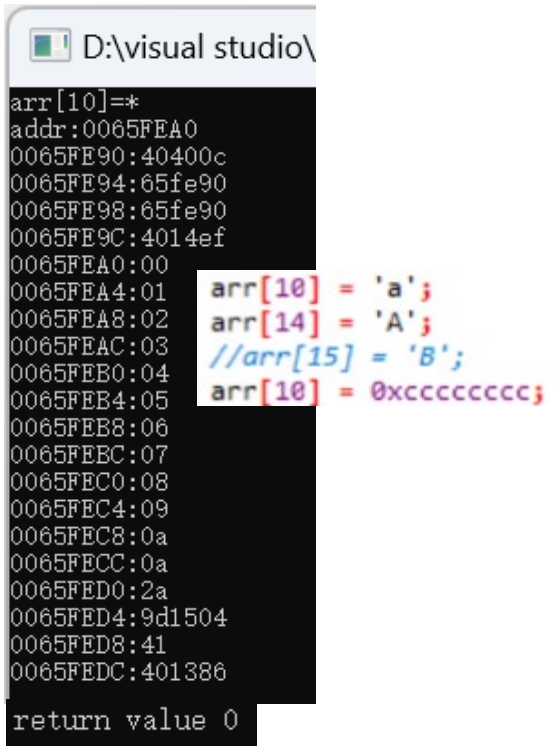
★ 如何判断普通数组的越界访问（C方式，注意源程序后缀为.c）　　int数组

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







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：

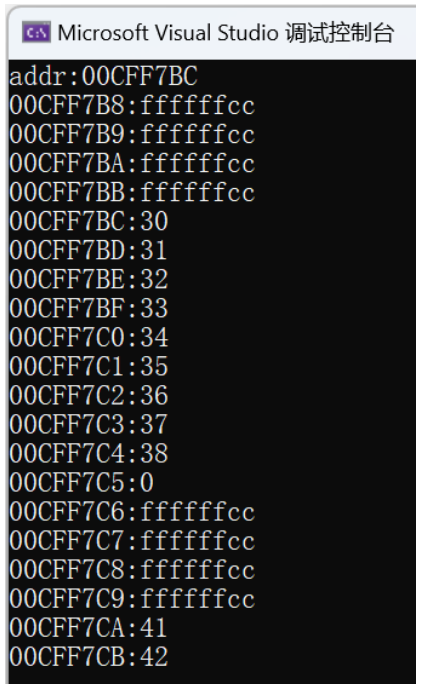
1、数组用 char a[10]; 形式
2、数组用 int a[10]; 形式
3、测试程序在下面五种环境下运行
   - VS2022 x86/Debug
   - VS2022 x86/Release
   - Dev 32bit-Debug
   - Dev 32bit-Release
   - Linux
4、每种讨论的结果可截图+文字说明，如果几种环境的结果一致，用一个环境的截图+文字说明即可(可加页)

§.关于动态内存申请后越界访问的深度讨论

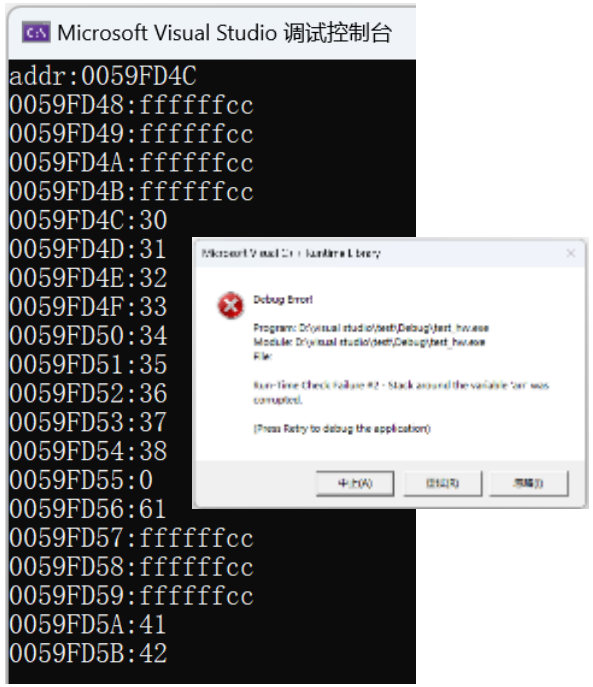
★ 如何判断普通数组的越界访问（C++方式，注意源程序后缀为.cpp） char数组

VS2022 x86/Debug

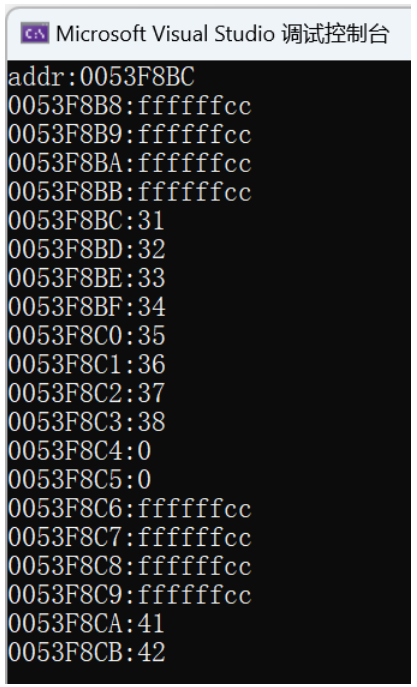
①②全部注释



①放开，②注释



①②全部放开



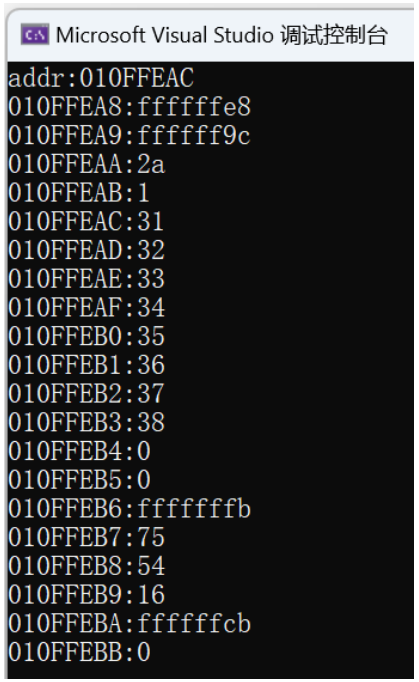
VS2022 Debug模式int数组情况下，空间后紧邻元素arr[10]的值是cccccccc，申请空间前四个元素值也是固定的。当用户释放动态申请的内存时，VS会检测这些位置的值是否仍均为cc，如果是，则认为用户没有越界；如果任何一处不是，则认为用户越界。所以修改已超过检测范围的arr[14]、arr[15]不会报错，但修改arr[10]-arr[13] 则会引起VS报错。如果用户不释放动态申请的内存或只读不写，无论怎么越界均不会报错。

§.关于动态内存申请后越界访问的深度讨论

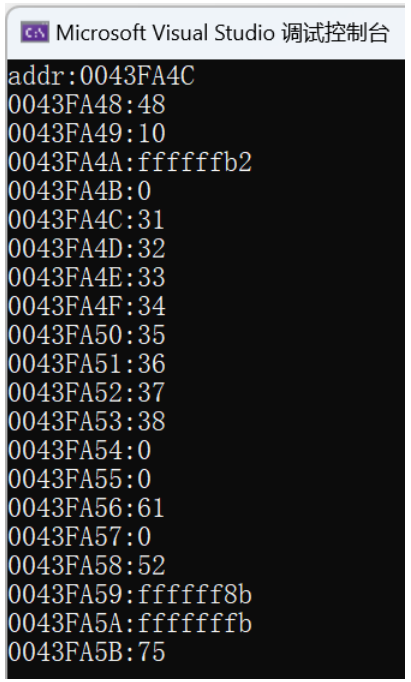
★ 如何判断普通数组的越界访问（C++方式，注意源程序后缀为.cpp） char数组

VS2022 x86/Release

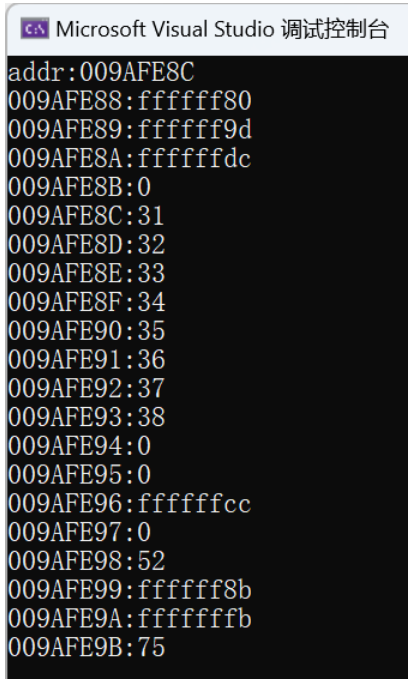
①②全部注释



①放开，②注释



①②全部放开



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

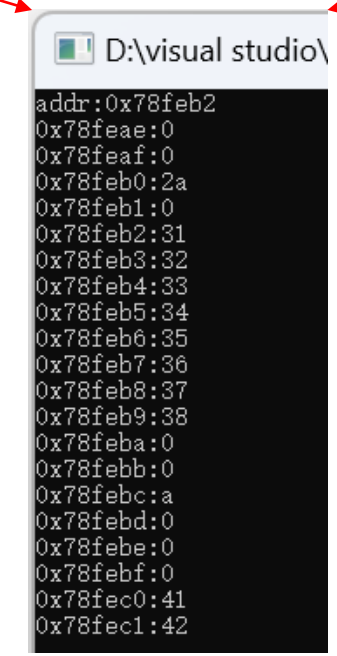
★ 如何判断普通数组的越界访问（C++方式，注意源程序后缀为.cpp） char数组

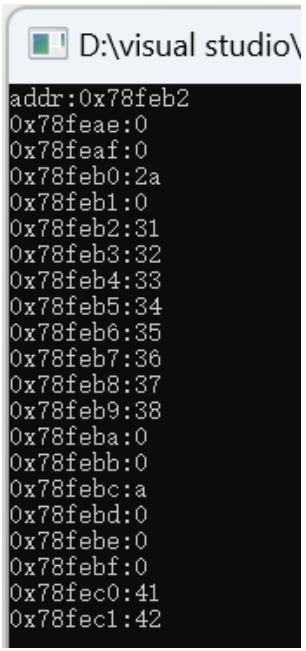
Dev 32bit-Release | Dev 32bit-Debug

①②全部注释

①放开，②注释

①②全部放开





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] = 0xcccccccc;
    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

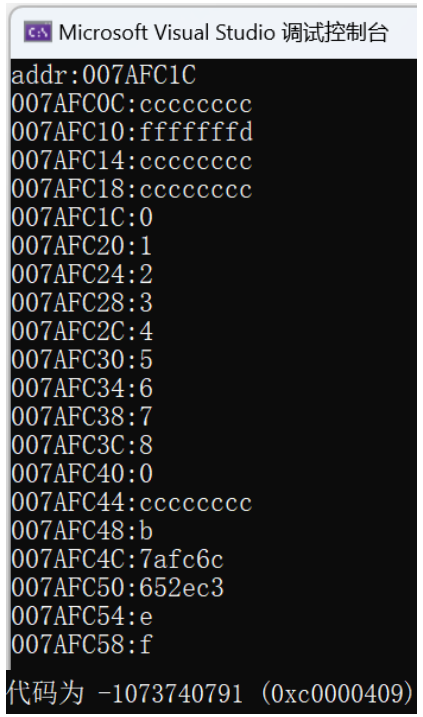
4、每种讨论的结果可截图+文字说明，如果几种环境的结果一致，用一个环境的截图+文字说明即可(可加页)

§.关于动态内存申请后越界访问的深度讨论

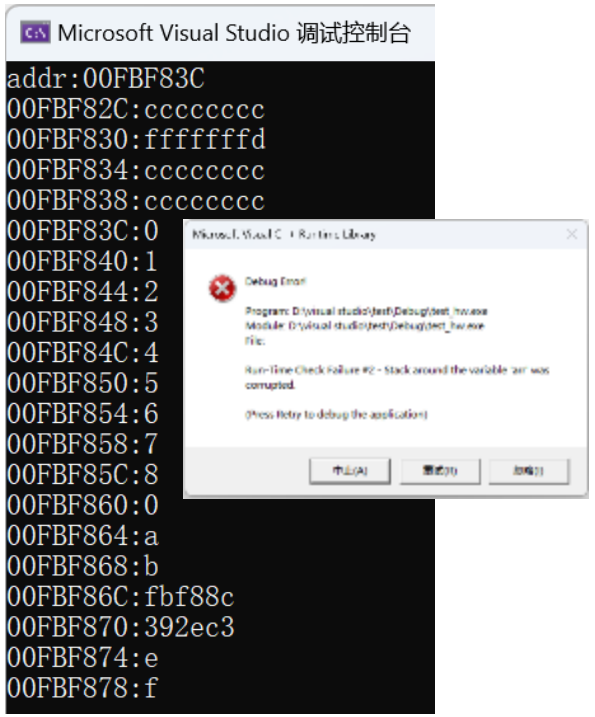
★ 如何判断普通数组的越界访问（C++方式，注意源程序后缀为.cpp） int数组

VS2022 x86/Debug

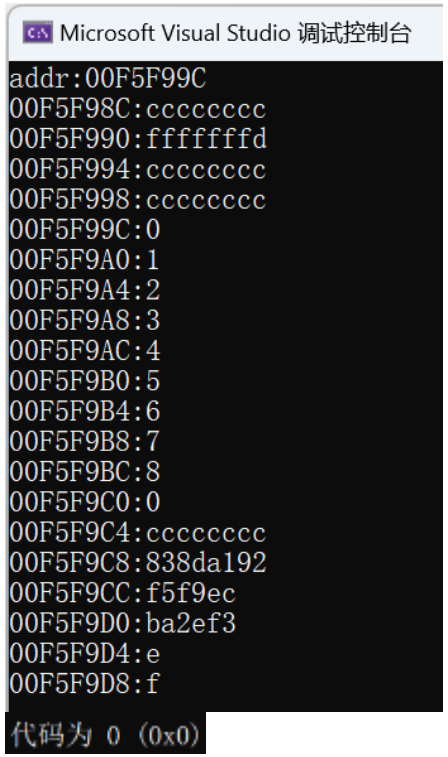
①②全部注释



①放开，②注释



①②全部放开



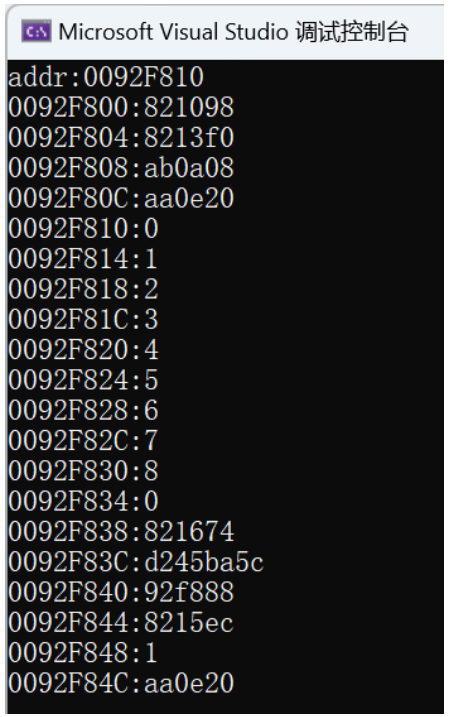
VS2022 Debug模式int数组情况下，将申请空间前、后的邻近四字节初始化为cc。当用户释放动态申请的内存时，VS会检测这些位置的值是否仍均为cc，如果是，则认为用户没有越界；如果任何一处不是，则认为用户越界。所以修改已超过检测范围的arr[14]、arr[15]不会报错，但修改arr[10]-arr[13] 则会引起VS代码错误。如果用户不释放动态申请的内存或只读不写，无论怎么越界均不会报错。

§.关于动态内存申请后越界访问的深度讨论

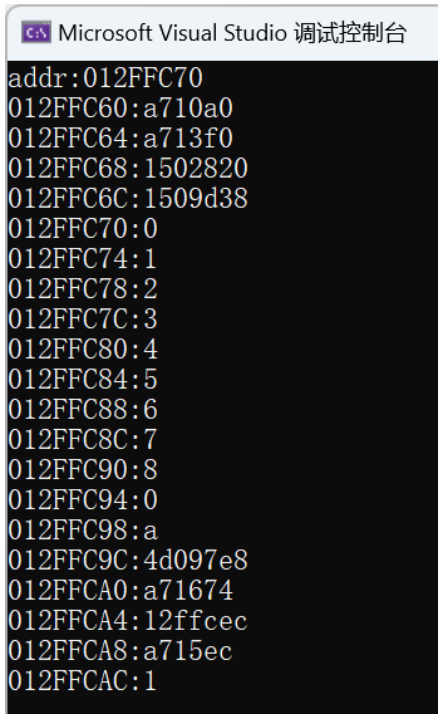
★ 如何判断普通数组的越界访问（C++方式，注意源程序后缀为.cpp） int数组

VS2022 x86/Release

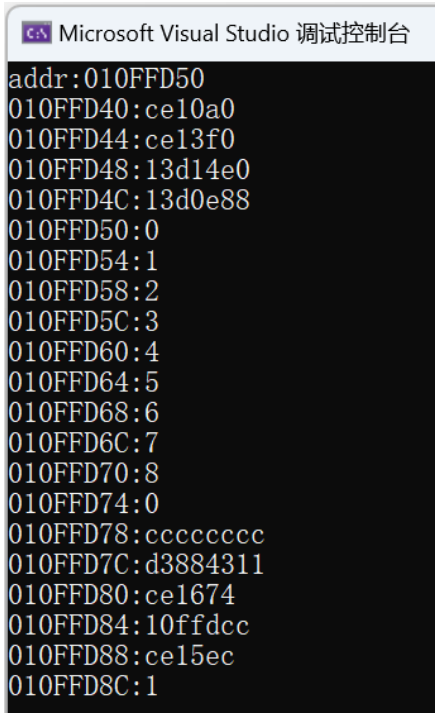
①②全部注释



①放开，②注释



①②全部放开



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

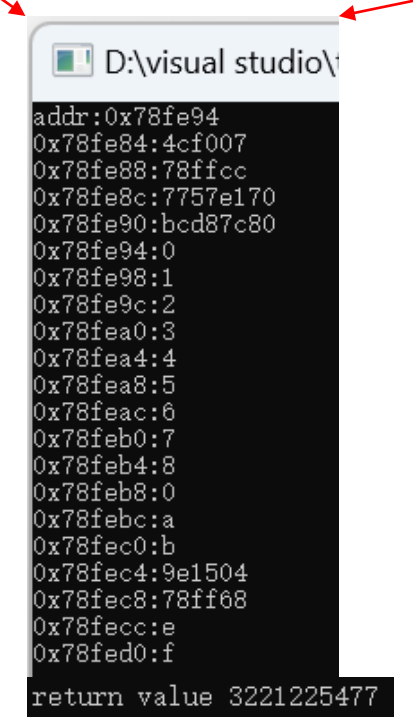
Dev 32bit-Release | Dev 32bit-Debug

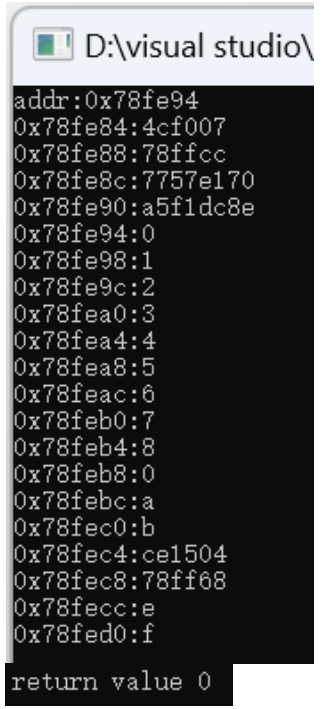
①②全部注释

①放开，②注释

①②全部放开

注释arr[11]的赋值





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:fcae1900
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:0xffffc37baa90
0xffffc37baa80:c37baac0
0xffffc37baa84:ffff
0xffffc37baa88:83634384
0xffffc37baa8c:ffff
0xffffc37baa90:0
0xffffc37baa94:1
0xffffc37baa98:2
0xffffc37baa9c:3
0xffffc37baaa0:4
0xffffc37baaa4:5
0xffffc37baaa8:6
0xffffc37baaac:7
0xffffc37baab0:8
0xffffc37baab4:0
0xffffc37baab8:a
0xffffc37baabc:b
0xffffc37baac0:0
0xffffc37baac4:0
0xffffc37baac8:e
0xffffc37baacc:f
```

①②全部放开

```
addr:0xffffd2985d80
0xffffd2985d70:d2985db0
0xffffd2985d74:ffff
0xffffd2985d78:90924384
0xffffd2985d7c:ffff
0xffffd2985d80:0
0xffffd2985d84:1
0xffffd2985d88:2
0xffffd2985d8c:3
0xffffd2985d90:4
0xffffd2985d94:5
0xffffd2985d98:6
0xffffd2985d9c:7
0xffffd2985da0:8
0xffffd2985da4:0
0xffffd2985da8:cccccccc
0xffffd2985dac:b
0xffffd2985db0:0
0xffffd2985db4:0
0xffffd2985db8:e
0xffffd2985dbc: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] = 0xcccccccc;
```

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。