§.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）

```
#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;
}
```

说明：程序正常运行输出。越界写操作成功写入，越界读操作正常读出，
未赋值的地址中为随机值。

在VS2022的x86/Debug模式下运行：
1、①②③全部注释，观察运行结果

```
addr:00A29B88
00A29B84:fffffffd
00A29B85:fffffffd
00A29B86:fffffffd
00A29B87:fffffffd
00A29B88:31
00A29B89:32
00A29B8A:33
00A29B8B:34
00A29B8C:35
00A29B8D:36
00A29B8E:37
00A29B8F:38
00A29B90:39
00A29B91:00
00A29B92:fffffffd
00A29B93:fffffffd
00A29B94:fffffffd
00A29B95:fffffffd
00A29B96:41
00A29B97:42

D:\大学\大一下\高级语言程
退出，代码为 0。
按任意键关闭此窗口. . .
```

§.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;
}
```

说明：程序正常运行输出。越界写操作成功写入，越界读操作正常读出，
未赋值的地址中为随机值。

在VS2022的x86/Debug模式下运行：
2、①放开，②③注释，观察运行结果

```
addr:00E99B88
00E99B84:fffffffd
00E99B85:fffffffd
00E99B86:fffffffd
00E99B87:fffffffd
00E99B88:31
00E99B89:32
00E99B8A:33
00E99B8B:34
00E99B8C:35
00E99B8D:36
00E99B8E:37
00E99B8F:38
00E99B90:39
00E99B91:00
00E99B92:61
00E99B93:fffffffd
00E99B94:fffffffd
00E99B95:fffffffd
00E99B96:41
00E99B97:42

D:\大学\大一下\高级语言程
退出，代码为 0。
按任意键关闭此窗口. . .
```

§.关于动态内存申请后越界访问的深度讨论

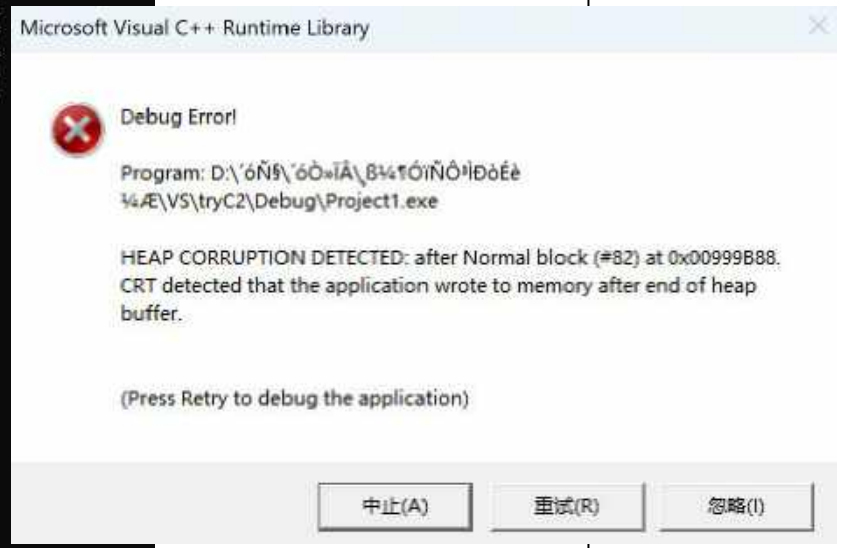
★ 如何判断动态申请越界（C方式，注意源程序后缀为.c）

```
#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;
}
```

说明：程序出现弹窗。

在VS2022的x86/Debug模式下运行：
3、①③放开，②注释，观察运行结果

```
addr:00999B88
00999B84:fffffffd
00999B85:fffffffd
00999B86:fffffffd
00999B87:fffffffd
00999B88:31
00999B89:32
00999B8A:33
00999B8B:34
00999B8C:35
00999B8D:36
00999B8E:37
00999B8F:38
00999B90:39
00999B91:00
00999B92:61
00999B93:fffffffd
00999B94:fffffffd
00999B95:fffffffd
00999B96:41
00999B97:42

D:\大学\大一下\高级语言程
出，代码为 -805306369。
按任意键关闭此窗口. . .
```



§.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;
}
```

说明：程序正常运行输出。越界写操作只有p[10]未被写入，越界读操作正常读出，未赋值的地址中为随机值。

在VS2022的x86/Debug模式下运行：

4、①②③全部放开，观察运行结果

```
addr:00CC9B88
00CC9B84:fffffffd
00CC9B85:fffffffd
00CC9B86:fffffffd
00CC9B87:fffffffd
00CC9B88:31
00CC9B89:32
00CC9B8A:33
00CC9B8B:34
00CC9B8C:35
00CC9B8D:36
00CC9B8E:37
00CC9B8F:38
00CC9B90:39
00CC9B91:00
00CC9B92:fffffffd
00CC9B93:fffffffd
00CC9B94:fffffffd
00CC9B95:fffffffd
00CC9B96:41
00CC9B97:42

D:\大学\大一下\高级语言程
退出，代码为 0。
```

§.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？

答：有时对动态内存申请的空间进行越界写入，VS的Debug模式并不会立即给出错误提示。这是因为动态内存越界并不总是会触发即时的崩溃或错误。当越界写入并且最后进行动态内存释放时，会出现弹窗警告越界写入。

如何判断动态申请内存访问越界：

1.内存填充：在动态分配和释放内存时，Debug 模式往往会用特殊字节（如0xCC或0xFD）填充未使用或释放的内存区域，以检测对未初始化或释放后内存的访问。

2.动态内存管理器如malloc和free会维护分配的内存块信息，在释放内存时会检查是否存在非法的访问行为。

§.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
Dev 32bit-Debug
Dev 32bit-Release
Linux下运行：

表现相同，此处仅贴了在VS2022 x86/Release环境下的表现

1、①②③全部注释，观察运行结果

```
addr:0157F630
0157F62C:20
0157F62D:08
0157F62E:00
0157F62F:ffffff8e
0157F630:31
0157F631:32
0157F632:33
0157F633:34
0157F634:35
0157F635:36
0157F636:37
0157F637:38
0157F638:39
0157F639:00
0157F63A:2e
0157F63B:00
0157F63C:34
0157F63D:00
0157F63E:41
0157F63F:42

D:\大学\大一下\高
已退出，代码为 0。
```

2、①放开，②③注释，观察运行结果

```
addr:0134E860
0134E85C:44
0134E85D:0a
0134E85E:00
0134E85F:ffffff8e
0134E860:31
0134E861:32
0134E862:33
0134E863:34
0134E864:35
0134E865:36
0134E866:37
0134E867:38
0134E868:39
0134E869:00
0134E86A:61
0134E86B:00
0134E86C:65
0134E86D:00
0134E86E:41
0134E86F:42

D:\大学\大一下\高
已退出，代码为 0。
```

§.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
Dev 32bit-Debug
Dev 32bit-Release
Linux下运行：

3、①③放开，②注释，观察运行结果

```
addr:0131E7A0
0131E79C:69
0131E79D:0c
0131E79E:00
0131E79F:ffffff8e
0131E7A0:31
0131E7A1:32
0131E7A2:33
0131E7A3:34
0131E7A4:35
0131E7A5:36
0131E7A6:37
0131E7A7:38
0131E7A8:39
0131E7A9:00
0131E7AA:61
0131E7AB:00
0131E7AC:5c
0131E7AD:00
0131E7AE:41
0131E7AF:42

D:\大学\大一下\高级
已退出，代码为 0。
```

4、①②③全部放开，观察运行结果

```
addr:00B5F618
00B5F614:6e
00B5F615:06
00B5F616:00
00B5F617:ffffff8e
00B5F618:31
00B5F619:32
00B5F61A:33
00B5F61B:34
00B5F61C:35
00B5F61D:36
00B5F61E:37
00B5F61F:38
00B5F620:39
00B5F621:00
00B5F622:fffffffd
00B5F623:00
00B5F624:74
00B5F625:00
00B5F626:41
00B5F627:42

D:\大学\大一下\高
退出，代码为 0。
```

§.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）

```
#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

```
addr:008A2850
008A284C:fffffffd
008A284D:fffffffd
008A284E:fffffffd
008A284F:fffffffd
008A2850:31
008A2851:32
008A2852:33
008A2853:34
008A2854:35
008A2855:36
008A2856:37
008A2857:38
008A2858:39
008A2859:0
008A285A:fffffffd
008A285B:fffffffd
008A285C:fffffffd
008A285D:fffffffd
008A285E:41
008A285F:42

D:\大学\大一下\高级
出，代码为 0。
```

§.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：

2、①放开，②③注释，观察运行结果

```
addr:00F918D8
00F918D4:fffffffd
00F918D5:fffffffd
00F918D6:fffffffd
00F918D7:fffffffd
00F918D8:31
00F918D9:32
00F918DA:33
00F918DB:34
00F918DC:35
00F918DD:36
00F918DE:37
00F918DF:38
00F918E0:39
00F918E1:0
00F918E2:61
00F918E3:fffffffd
00F918E4:fffffffd
00F918E5:fffffffd
00F918E6:41
00F918E7:42

D:\大学\大一下\高级
```

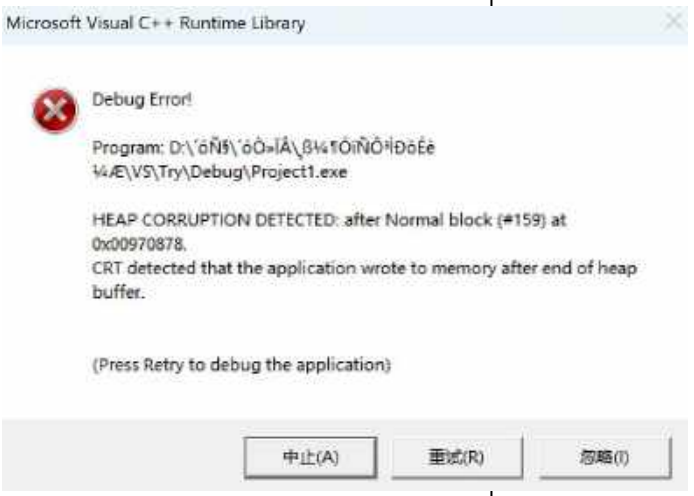
§.关于动态内存申请后越界访问的深度讨论

★ 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：
3、①③放开，②注释，观察运行结果

```
addr:00970878
00970874:fffffffd
00970875:fffffffd
00970876:fffffffd
00970877:fffffffd
00970878:31
00970879:32
0097087A:33
0097087B:34
0097087C:35
0097087D:36
0097087E:37
0097087F:38
00970880:39
00970881:0
00970882:61
00970883:fffffffd
00970884:fffffffd
00970885:fffffffd
00970886:41
00970887:42

D:\大学\大一下\高级语言
出，代码为 -805306369。
```



§.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：
4、①②③全部放开，观察运行结果

```
addr:00BD0AA8
00BD0AA4:fffffffd
00BD0AA5:fffffffd
00BD0AA6:fffffffd
00BD0AA7:fffffffd
00BD0AA8:31
00BD0AA9:32
00BD0AAA:33
00BD0AAB:34
00BD0AAC:35
00BD0AAD:36
00BD0AAE:37
00BD0AAF:38
00BD0AB0:39
00BD0AB1:0
00BD0AB2:fffffffd
00BD0AB3:fffffffd
00BD0AB4:fffffffd
00BD0AB5:fffffffd
00BD0AB6:41
00BD0AB7:42

D:\大学\大一下\高级
出，代码为 0。
```

§.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
Dev 32bit-Debug
Dev 32bit-Release
Linux下运行：

表现相同，此处仅贴了在VS2022 x86/Release环境下的表现

1、①②③全部注释，观察运行结果

```
addr:00CCD1D8
00CCD1D4:20
00CCD1D5:24
00CCD1D6:0
00CCD1D7:ffffff8e
00CCD1D8:31
00CCD1D9:32
00CCD1DA:33
00CCD1DB:34
00CCD1DC:35
00CCD1DD:36
00CCD1DE:37
00CCD1DF:38
00CCD1E0:39
00CCD1E1:0
00CCD1E2:0
00CCD1E3:0
00CCD1E4:10
00CCD1E5:ffffffc5
00CCD1E6:41
00CCD1E7:42

D:\大学\大一下\高级
退出，代码为 0。
```

2、①放开，②③注释，观察运行结果

```
addr:0125E0E8
0125E0E4:74
0125E0E5:7
0125E0E6:0
0125E0E7:ffffff8e
0125E0E8:31
0125E0E9:32
0125E0EA:33
0125E0EB:34
0125E0EC:35
0125E0ED:36
0125E0EE:37
0125E0EF:38
0125E0F0:39
0125E0F1:0
0125E0F2:61
0125E0F3:0
0125E0F4:54
0125E0F5:0
0125E0F6:41
0125E0F7:42

D:\大学\大一下\高
退出，代码为 0。
```

§.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
Dev 32bit-Debug
Dev 32bit-Release
Linux下运行：

3、①③放开，②注释，观察运行结果

```
addr:015827D0
015827CC:74
015827CD:e
015827CE:0
015827CF:ffffff8e
015827D0:31
015827D1:32
015827D2:33
015827D3:34
015827D4:35
015827D5:36
015827D6:37
015827D7:38
015827D8:39
015827D9:0
015827DA:61
015827DB:0
015827DC:73
015827DD:0
015827DE:41
015827DF:42

D:\大学\大一下\高
退出，代码为 0。
```

4、①②③全部放开，观察运行结果

```
addr:0118E190
0118E18C:43
0118E18D:e
0118E18E:0
0118E18F:ffffff8e
0118E190:31
0118E191:32
0118E192:33
0118E193:34
0118E194:35
0118E195:36
0118E196:37
0118E197:38
0118E198:39
0118E199:0
0118E19A:fffffffd
0118E19B:0
0118E19C:69
0118E19D:0
0118E19E:41
0118E19F:42

D:\大学\大一下\高
退出，代码为 0。
```

§.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）

1、数组用 char a[10]; 形式

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

int main()
{
    char a[10];
    char* p = a;
    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]);

    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

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）

1、数组用 char a[10]; 形式

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

int main()
{
    char a[10];
    char* p = a;
    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]);

    return 0;
}
```

p[14] = 'A';p[15] = 'B'; 越界写入，程序正常运行，
与动态申请的情况1（全部注释）相似

VS2022 x86/Debug环境下运行

1、①②全部注释，观察运行结果

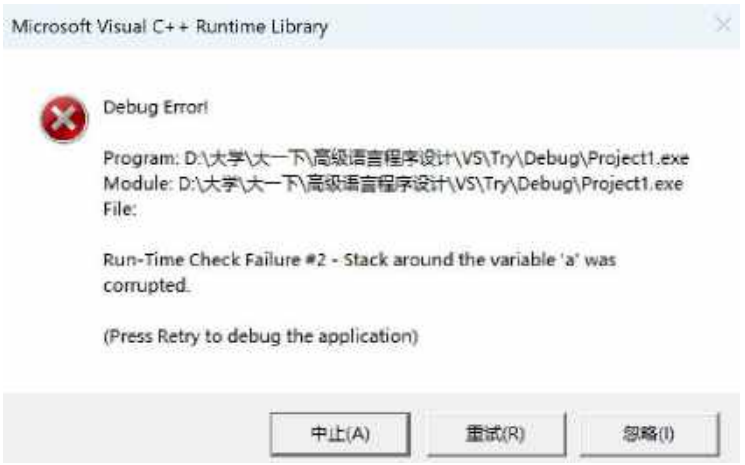
```
addr:00EFFE24
00EFFE20:ffffffcc
00EFFE21:ffffffcc
00EFFE22:ffffffcc
00EFFE23:ffffffcc
00EFFE24:31
00EFFE25:32
00EFFE26:33
00EFFE27:34
00EFFE28:35
00EFFE29:36
00EFFE2A:37
00EFFE2B:38
00EFFE2C:39
00EFFE2D:00
00EFFE2E:ffffffcc
00EFFE2F:ffffffcc
00EFFE30:ffffffcc
00EFFE31:ffffffcc
00EFFE32:41
00EFFE33:42

D:\大学\大一下\高级
出，代码为 0。
```

§.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）

1、数组用 char a[10]; 形式

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

int main()
{
    char a[10];
    char* p = a;
    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]);

    return 0;
}
```



VS2022 x86/Debug环境下运行

2.①放开，②注释，观察运行结果

```
addr:004FF8F0
004FF8EC:ffffffcc
004FF8ED:ffffffcc
004FF8EE:ffffffcc
004FF8EF:ffffffcc
004FF8F0:31
004FF8F1:32
004FF8F2:33
004FF8F3:34
004FF8F4:35
004FF8F5:36
004FF8F6:37
004FF8F7:38
004FF8F8:39
004FF8F9:00
004FF8FA:61
004FF8FB:ffffffcc
004FF8FC:ffffffcc
004FF8FD:ffffffcc
004FF8FE:41
004FF8FF:42

D:\大学\大一下\高级
出，代码为 3。
```

p[10] = 'a'; 越界写入，程序出现弹窗，退出代码为3

与动态申请的情况3（只有//p[10] = '\xfd';注释）相似，

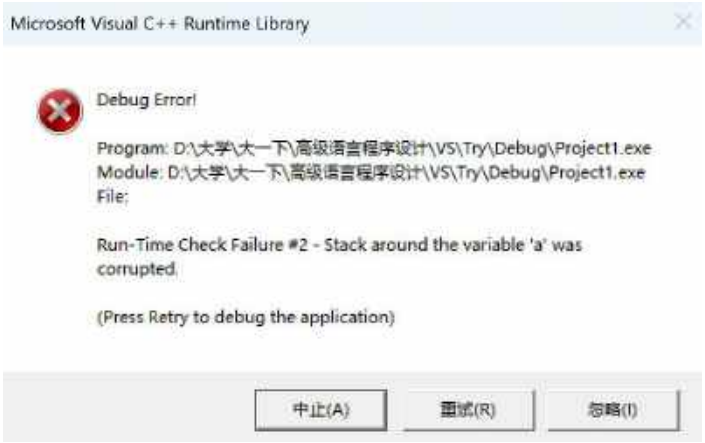
但动态内存退出代码为一个很大的负数

§.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）

1、数组用 char a[10]; 形式

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

int main()
{
    char a[10];
    char* p = a;
    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]);

    return 0;
}
```



p[10] = 'a'; p[10] = '\xfd';越界写入，程序出现弹窗，
与动态申请的情况4（全部放开）不相似

VS2022 x86/Debug环境下运行

3.①②放开，观察运行结果

```
addr:0093F760
0093F75C:ffffffcc
0093F75D:ffffffcc
0093F75E:ffffffcc
0093F75F:ffffffcc
0093F760:31
0093F761:32
0093F762:33
0093F763:34
0093F764:35
0093F765:36
0093F766:37
0093F767:38
0093F768:39
0093F769:00
0093F76A:fffffffd
0093F76B:ffffffcc
0093F76C:ffffffcc
0093F76D:ffffffcc
0093F76E:41
0093F76F:42

D:\大学\大一下\高级
出，代码为 3。
```

§.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）

1、数组用 char a[10]; 形式
VS2022 x86/Release
环境下运行：

程序均正常运行输出，
但p[14] = 'A';p[15] = 'B';
这两句越界没有写入

1、①②全部注释，
观察运行结果

```
addr:0073F834
0073F830:ffffffd8
0073F831:ffffff9d
0073F832:ffffffb6
0073F833:00
0073F834:31
0073F835:32
0073F836:33
0073F837:34
0073F838:35
0073F839:36
0073F83A:37
0073F83B:38
0073F83C:39
0073F83D:00
0073F83E:05
0073F83F:77
0073F840:23
0073F841:13
0073F842:fffffffe1
0073F843:00

D:\大学\大一下\高级
退出，代码为 0。
```

2、①放开，②注释，
观察运行结果

```
addr:005DFAA0
005DFA9C:38
005DFA9D:fffffff0
005DFA9E:ffffffca
005DFA9F:00
005DFAA0:31
005DFAA1:32
005DFAA2:33
005DFAA3:34
005DFAA4:35
005DFAA5:36
005DFAA6:37
005DFAA7:38
005DFAA8:39
005DFAA9:00
005DFAAA:61
005DFAAB:77
005DFAAC:25
005DFAAD:13
005DFAAE:77
005DFAAF:00

D:\大学\大一下\高级
退出，代码为 0。
```

3、①②放开，
观察运行结果

```
addr:008FF894
008FF890:ffffffd8
008FF891:ffffff9d
008FF892:ffffffb9
008FF893:00
008FF894:31
008FF895:32
008FF896:33
008FF897:34
008FF898:35
008FF899:36
008FF89A:37
008FF89B:38
008FF89C:39
008FF89D:00
008FF89E:fffffffd
008FF89F:77
008FF8A0:25
008FF8A1:13
008FF8A2:3b
008FF8A3:00

D:\大学\大一下\高级
退出，代码为 0。
```

§.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）

1、数组用 char a[10]; 形式
Dev 32bit-Debug
Dev 32bit-Release
Linux环境下运行：

程序均正常运行输出，
情况2/3很多数据没有正确写入

1、①②全部注释，观察运行结果

```
addr:0065FEBE
0065FEBA:65
0065FEBB:00
0065FEBC:40
0065FEBD:16
0065FEBE:31
0065FEBF:32
0065FEC0:33
0065FEC1:34
0065FEC2:35
0065FEC3:36
0065FEC4:37
0065FEC5:38
0065FEC6:39
0065FEC7:00
0065FEC8:ffffffbe
0065FEC9:fffffffe
0065FECA:65
0065FECB:00
0065FECC:0e
0065FECD:00
```

2、①放开，②注释，观察运行结果

```
addr:0065FE61
0065FE5D:7c
0065FE5E:ffffff83
0065FE5F:77
0065FE60:01
0065FE61:00
0065FE62:00
0065FE63:00
0065FE64:20
0065FE65:ffffff96
0065FE66:ffffff87
0065FE67:77
0065FE68:26
0065FE69:7c
0065FE6A:ffffff83
0065FE6B:77
0065FE6C:ffffffa9
0065FE6D:ffffffe0
0065FE6E:ffffffdf
0065FE6F:4d
0065FE70:04
```

3、①②放开，观察运行结果

```
addr:0065FE61
0065FE5D:7c
0065FE5E:ffffff83
0065FE5F:77
0065FE60:01
0065FE61:00
0065FE62:00
0065FE63:00
0065FE64:20
0065FE65:ffffff96
0065FE66:ffffff87
0065FE67:77
0065FE68:26
0065FE69:7c
0065FE6A:ffffff83
0065FE6B:77
0065FE6C:ffffffe8
0065FE6D:2a
0065FE6E:14
0065FE6F:ffffffdf
0065FE70:04
```

§.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）

2、数组用 int a[10]; 形式

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

int main()
{
    int a[10];
    int i;
    for (i = 0; i < 10; i++) {
        a[i] = i + 1;
    }
 ① a[10] = 10;     //此句越界
    a[14] = 20;     //此句越界
    a[15] = 30;     //此句越界
 ② a[10] = '\xfd'; //此句越界
    printf("addr:%p\n", a);
    for (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", (a + i), a[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

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）

2、数组用 int a[10]; 形式

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

int main()
{
    int a[10];
    int i;
    for (i = 0; i < 10; i++) {
        a[i] = i + 1;
    }
 ① //a[10] = 10;     //此句越界
    a[14] = 20;     //此句越界
    a[15] = 30;     //此句越界
 ② //a[10] = '\xfd'; //此句越界
    printf("addr:%p\n", a);
    for (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", (a + i), a[i]);

    return 0;
}
```

a[14] = 20;a[15] = 30;越界写入，程序正常运行，
与动态申请的情况1（全部注释）相似

VS2022 x86/Debug环境下运行

1、①②全部注释，观察运行结果

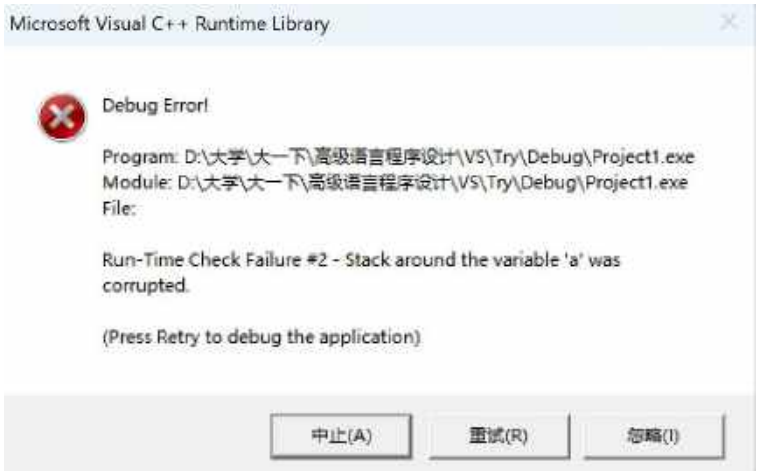
```
addr:012FFEA8
012FFE98:cccccccc
012FFE9C:fffffffd
012FFEA0:cccccccc
012FFEA4:cccccccc
012FFEA8:01
012FFEAC:02
012FFEB0:03
012FFEB4:04
012FFEB8:05
012FFEBC:06
012FFEC0:07
012FFEC4:08
012FFEC8:09
012FFECC:0a
012FFED0:cccccccc
012FFED4:1adc536a
012FFED8:12ffef8
012FFEDC:f866e3
012FFEE0:14
012FFEE4:1e

D:\大学\大一下\高
出，代码为 0。
```

§.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）

2、数组用 int a[10]; 形式

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

int main()
{
    int a[10];
    int i;
    for (i = 0; i < 10; i++) {
        a[i] = i + 1;
    }
 ① a[10] = 10;     //此句越界
    a[14] = 20;     //此句越界
    a[15] = 30;     //此句越界
 ② //a[10] = '\xfd'; //此句越界
    printf("addr:%p\n", a);
    for (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", (a + i), a[i]);

    return 0;
}
```



a[10] = 10; 越界写入，程序出现弹窗，退出代码为3

与动态申请的情况3（只有//p[10] = '\xfd';注释）相似，但动态内存退出代码为一个很大的负数

VS2022 x86/Debug环境下运行

2.①放开，②注释，观察运行结果

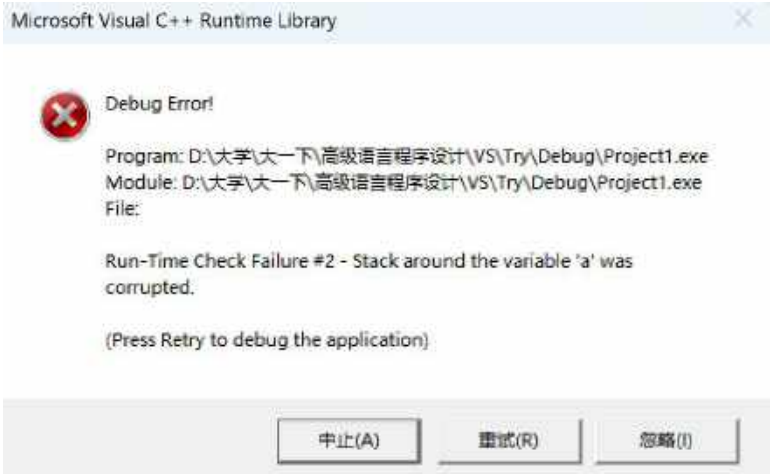
```
addr:00EBFE58
00EBFE48:cccccccc
00EBFE4C:fffffffd
00EBFE50:cccccccc
00EBFE54:cccccccc
00EBFE58:01
00EBFE5C:02
00EBFE60:03
00EBFE64:04
00EBFE68:05
00EBFE6C:06
00EBFE70:07
00EBFE74:08
00EBFE78:09
00EBFE7C:0a
00EBFE80:0a
00EBFE84:876d124c
00EBFE88:ebfea8
00EBFE8C:f666e3
00EBFE90:14
00EBFE94:1e

D:\大学\大一下\高级
出，代码为 3。
```

§.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）

2、数组用 int a[10]; 形式

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

int main()
{
    int a[10];
    int i;
    for (i = 0; i < 10; i++) {
        a[i] = i + 1;
    }
 ① a[10] = 10;     //此句越界
    a[14] = 20;     //此句越界
    a[15] = 30;     //此句越界
 ② //a[10] = '\xfd'; //此句越界
    printf("addr:%p\n", a);
    for (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", (a + i), a[i]);

    return 0;
}
```



VS2022 x86/Debug环境下运行

3.①②放开，观察运行结果

```
addr:00BBF6CC
00BBF6BC:cccccccc
00BBF6C0:fffffffd
00BBF6C4:cccccccc
00BBF6C8:cccccccc
00BBF6CC:01
00BBF6D0:02
00BBF6D4:03
00BBF6D8:04
00BBF6DC:05
00BBF6E0:06
00BBF6E4:07
00BBF6E8:08
00BBF6EC:09
00BBF6F0:0a
00BBF6F4:fffffffd
00BBF6F8:49423b02
00BBF6FC:bbf71c
00BBF700:2c66e3
00BBF704:14
00BBF708:1e

D:\大学\大一下\高级
出，代码为 3。
```

a[10] = 10;a[10] = '\xfd';越界写入，程序出现弹窗，
与动态申请的情况4（全部放开）不相似

§.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）

2、数组用 int a[10]; 形式
VS2022 x86/Release
环境下运行：

程序均正常运行输出，
但 a[14] = 20;a[15] = 30;
这两句越界没有写入

1、①②全部注释，
观察运行结果

```
addr:00EFF968
00EFF958:7211c
00EFF95C:7211c
00EFF960:1222820
00EFF964:1229dd8
00EFF968:01
00EFF96C:02
00EFF970:03
00EFF974:04
00EFF978:05
00EFF97C:06
00EFF980:07
00EFF984:08
00EFF988:09
00EFF98C:0a
00EFF990:df595146
00EFF994:eff9dc
00EFF998:712a9
00EFF99C:01
00EFF9A0:1229dd8
00EFF9A4:1222820

D:\大学\大一下\高级
出，代码为 0。
```

2、①放开，②注释，
观察运行结果

```
addr:009BFDC0
009BFDB0:8b211c
009BFDB4:8b211c
009BFDB8:e42820
009BFDBC:e49dd8
009BFDC0:01
009BFDC4:02
009BFDC8:03
009BFDCC:04
009BFDD0:05
009BFDD4:06
009BFDD8:07
009BFDDC:08
009BFDE0:09
009BFDE4:0a
009BFDE8:0a
009BFDEC:6b427632
009BFDF0:9bfe38
009BFDF4:8b12b7
009BFDF8:01
009BFDFC:e49dd8

D:\大学\大一下\高级语
退出，代码为 0。
```

3、①②放开，
观察运行结果

```
addr:00B5FB28
00B5FB18:60211c
00B5FB1C:60211c
00B5FB20:d82820
00B5FB24:d89dd8
00B5FB28:01
00B5FB2C:02
00B5FB30:03
00B5FB34:04
00B5FB38:05
00B5FB3C:06
00B5FB40:07
00B5FB44:08
00B5FB48:09
00B5FB4C:0a
00B5FB50:fffffffd
00B5FB54:ca458b88
00B5FB58:b5fba0
00B5FB5C:6012b7
00B5FB60:01
00B5FB64:d89dd8

D:\大学\大一下\高级
退出，代码为 0。
```

§.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）

2、数组用 int a[10]; 形式
Dev 32bit-Debug
Dev 32bit-Release
Linux环境下运行：

程序均运行输出，
但退出代码不为0

1、①②全部注释，观察运行结果

```
addr:0065FEA4
0065FE94:65fea4
0065FE98:65fea4
0065FE9C:4014ef
0065FEA0:4015e0
0065FEA4:01
0065FEA8:02
0065FEAC:03
0065FEB0:04
0065FEB4:05
0065FEB8:06
0065FEBC:07
0065FEC0:08
0065FEC4:09
0065FEC8:0a
0065FECC:0a
0065FED0:3d
0065FED4:c61504
0065FED8:65ff68
0065FEDC:14
0065FEE0:1e

--------------------------------
Process exited after 2.963 seconds with return value 3221225477
```

2、①放开，②注释，观察运行结果

```
addr:0065FEA4
0065FE94:65fea4
0065FE98:65fea4
0065FE9C:4014ef
0065FEA0:4015e0
0065FEA4:01
0065FEA8:02
0065FEAC:03
0065FEB0:04
0065FEB4:05
0065FEB8:06
0065FEBC:07
0065FEC0:08
0065FEC4:09
0065FEC8:0a
0065FECC:0a
0065FED0:3d
0065FED4:c91504
0065FED8:65ff68
0065FEDC:14
0065FEE0:1e
```

3、①②放开，观察运行结果

```
addr:0065FEA4
0065FE94:65fea4
0065FE98:65fea4
0065FE9C:4014ef
0065FEA0:4015f0
0065FEA4:01
0065FEA8:02
0065FEAC:03
0065FEB0:04
0065FEB4:05
0065FEB8:06
0065FEBC:07
0065FEC0:08
0065FEC4:09
0065FEC8:0a
0065FECC:0a
0065FED0:3d
0065FED4:d91504
0065FED8:65ff68
0065FEDC:14
0065FEE0:1e
```

§.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）

1、数组用 char a[10]; 形式

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

int main()
{
    char a[10];
    char* p = a;
    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;

    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

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）

1、数组用 char a[10]; 形式

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

int main()
{
    char a[10];
    char* p = a;
    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;

    return 0;
}
```

p[14] = 'A';p[15] = 'B'; 越界写入，程序正常运行，
与动态申请的情况1（全部注释）相似

VS2022 x86/Debug环境下运行

1、①②全部注释，观察运行结果

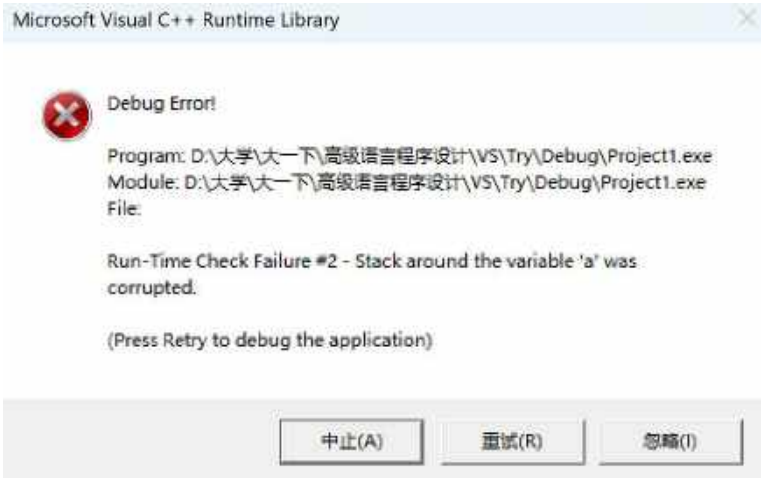
```
addr:00FAF8B8
00FAF8B4:ffffffcc
00FAF8B5:ffffffcc
00FAF8B6:ffffffcc
00FAF8B7:ffffffcc
00FAF8B8:31
00FAF8B9:32
00FAF8BA:33
00FAF8BB:34
00FAF8BC:35
00FAF8BD:36
00FAF8BE:37
00FAF8BF:38
00FAF8C0:39
00FAF8C1:0
00FAF8C2:ffffffcc
00FAF8C3:ffffffcc
00FAF8C4:ffffffcc
00FAF8C5:ffffffcc
00FAF8C6:41
00FAF8C7:42

D:\大学\大一下\高级
出，代码为 0。
```

§.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）

1、数组用 char a[10]; 形式

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

int main()
{
    char a[10];
    char* p = a;
    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;

    return 0;
}
```



VS2022 x86/Debug环境下运行

2.①放开，②注释，观察运行结果

```
addr:0058F7A8
0058F7A4:ffffffcc
0058F7A5:ffffffcc
0058F7A6:ffffffcc
0058F7A7:ffffffcc
0058F7A8:31
0058F7A9:32
0058F7AA:33
0058F7AB:34
0058F7AC:35
0058F7AD:36
0058F7AE:37
0058F7AF:38
0058F7B0:39
0058F7B1:0
0058F7B2:61
0058F7B3:ffffffcc
0058F7B4:ffffffcc
0058F7B5:ffffffcc
0058F7B6:41
0058F7B7:42

D:\大学\大一下\高级
出，代码为 3。
```

p[10] = 'a'; 越界写入，程序出现弹窗，退出代码为3

与动态申请的情况3（只有//p[10] = '\xfd';注释）相似，

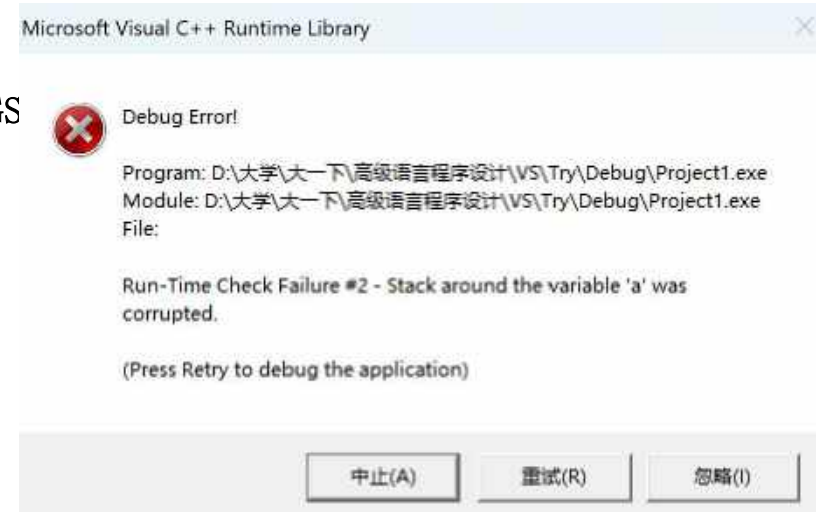
但动态内存退出代码为一个很大的负数

§.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）

1、数组用 char a[10]; 形式

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

int main()
{
    char a[10];
    char* p = a;
    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;

    return 0;
}
```



VS2022 x86/Debug环境下运行

3.①②放开，观察运行结果

```
addr:00D3F85C
00D3F858:ffffffcc
00D3F859:ffffffcc
00D3F85A:ffffffcc
00D3F85B:ffffffcc
00D3F85C:31
00D3F85D:32
00D3F85E:33
00D3F85F:34
00D3F860:35
00D3F861:36
00D3F862:37
00D3F863:38
00D3F864:39
00D3F865:0
00D3F866:fffffffd
00D3F867:ffffffcc
00D3F868:ffffffcc
00D3F869:ffffffcc
00D3F86A:41
00D3F86B:42

D:\大学\大一下\高级
出，代码为 3。
```

p[10] = 'a'; p[10] = '\xfd';越界写入，程序出现弹窗，
与动态申请的情况4（全部放开）不相似

§.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）

1、数组用 char a[10]; 形式
VS2022 x86/Release
环境下运行：

程序均正常运行输出，
但p[14] = 'A';p[15] = 'B';
这两句越界没有写入

1、①②全部注释，
观察运行结果

```
addr:008FFD3C
008FFD38:48
008FFD39:c
008FFD3A:fffffffb
008FFD3B:0
008FFD3C:31
008FFD3D:32
008FFD3E:33
008FFD3F:34
008FFD40:35
008FFD41:36
008FFD42:37
008FFD43:38
008FFD44:39
008FFD45:0
008FFD46:5
008FFD47:77
008FFD48:72
008FFD49:16
008FFD4A:1b
008FFD4B:0

D:\大学\大一下\高
出，代码为 0。
```

2、①放开，②注释，
观察运行结果

```
addr:008FF8FC
008FF8F8:48
008FF8F9:c
008FF8FA:fffffffc0
008FF8FB:0
008FF8FC:31
008FF8FD:32
008FF8FE:33
008FF8FF:34
008FF900:35
008FF901:36
008FF902:37
008FF903:38
008FF904:39
008FF905:0
008FF906:61
008FF907:ffffffff
008FF908:52
008FF909:ffffff8b
008FF90A:5
008FF90B:77

D:\大学\大一下\高
退出，代码为 0。
```

3、①②放开，
观察运行结果

```
addr:0135F80C
0135F808:ffffffc8
0135F809:10
0135F80A:7b
0135F80B:1
0135F80C:31
0135F80D:32
0135F80E:33
0135F80F:34
0135F810:35
0135F811:36
0135F812:37
0135F813:38
0135F814:39
0135F815:0
0135F816:fffffffd
0135F817:ffffffff
0135F818:52
0135F819:ffffff8b
0135F81A:5
0135F81B:77

D:\大学\大一下\高级
退出，代码为 0。
```

§.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）

1、数组用 char a[10]; 形式
Dev 32bit-Debug
Dev 32bit-Release
Linux环境下运行：

程序均正常运行输出，
情况2/3很多数据没有
正确写入

1、①②全部注释，
观察运行结果

```
addr:0x78feae
0x78feaa:0
0x78feab:0
0x78feac:0
0x78fead:0
0x78feae:31
0x78feaf:32
0x78feb0:33
0x78feb1:34
0x78feb2:35
0x78feb3:36
0x78feb4:37
0x78feb5:38
0x78feb6:39
0x78feb7:0
0x78feb8:ffffffae
0x78feb9:fffffffe
0x78feba:78
0x78febb:0
0x78febc:e
0x78febd:0
```

2、①放开，②注释，
观察运行结果

```
addr:0x78fe61
0x78fe5d:fffffff0
0x78fe5e:4c
0x78fe5f:0
0x78fe60:ffffff80
0x78fe61:ffffffe7
0x78fe62:4c
0x78fe63:0
0x78fe64:4
0x78fe65:15
0x78fe66:ffffffd0
0x78fe67:0
0x78fe68:ffffffc8
0x78fe69:fffffffe
0x78fe6a:78
0x78fe6b:0
0x78fe6c:49
0x78fe6d:47
0x78fe6e:4c
0x78fe6f:0
0x78fe70:ffffff80
```

3、①②放开，
观察运行结果

```
addr:0x78fe61
0x78fe5d:fffffff0
0x78fe5e:4c
0x78fe5f:0
0x78fe60:ffffff80
0x78fe61:ffffffe7
0x78fe62:4c
0x78fe63:0
0x78fe64:4
0x78fe65:15
0x78fe66:ffffffe1
0x78fe67:0
0x78fe68:ffffffc8
0x78fe69:fffffffe
0x78fe6a:78
0x78fe6b:0
0x78fe6c:59
0x78fe6d:47
0x78fe6e:4c
0x78fe6f:0
0x78fe70:ffffff80
```

§.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）

2、数组用 int a[10]; 形式

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

int main()
{
    int a[10];
    int i;
    for (i = 0; i < 10; i++) {
        a[i] = i + 1;
    }
 ① a[10] = 10;     //此句越界
    a[14] = 20;     //此句越界
    a[15] = 30;     //此句越界
 ② a[10] = '\xfd'; //此句越界
    cout << "addr:" << hex << (void*)(a) << endl;
    for (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*)(a + i) << ":" << a[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

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）

2、数组用 int a[10]; 形式

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

int main()
{
    int a[10];
    int i;
    for (i = 0; i < 10; i++) {
        a[i] = i + 1;
    }
 ① //a[10] = 10;     //此句越界
    a[14] = 20;     //此句越界
    a[15] = 30;     //此句越界
 ② //a[10] = '\xfd'; //此句越界
    cout << "addr:" << hex << (void*)(a) << endl;
    for (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*)(a + i) << ":" << a[i] << endl;

    return 0;
}
```

a[14] = 20;a[15] = 30; 越界写入，程序正常运行，
与动态申请的情况1（全部注释）相似

VS2022 x86/Debug环境下运行

1、①②全部注释，观察运行结果

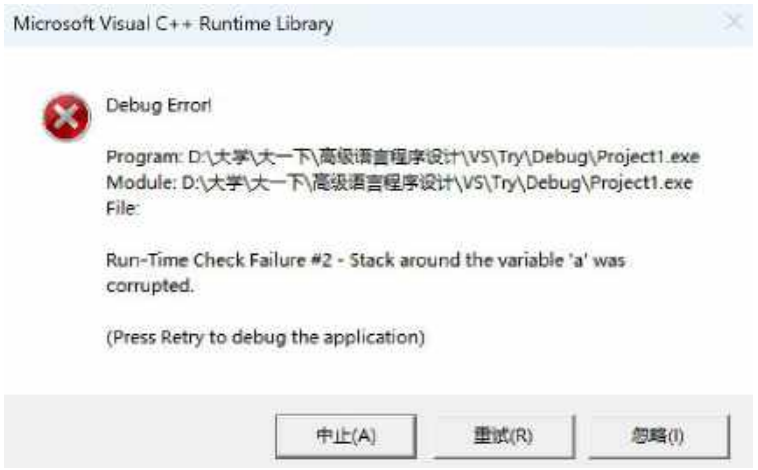
```
addr:00A6FA84
00A6FA74:cccccccc
00A6FA78:fffffffd
00A6FA7C:cccccccc
00A6FA80:cccccccc
00A6FA84:1
00A6FA88:2
00A6FA8C:3
00A6FA90:4
00A6FA94:5
00A6FA98:6
00A6FA9C:7
00A6FAA0:8
00A6FAA4:9
00A6FAA8:a
00A6FAAC:cccccccc
00A6FAB0:758e564a
00A6FAB4:a6fad4
00A6FAB8:bb66b3
00A6FABC:14
00A6FAC0:1e

D:\大学\大一下\高
出，代码为 0。
```

§.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）

2、数组用 int a[10]; 形式

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

int main()
{
    int a[10];
    int i;
    for (i = 0; i < 10; i++) {
        a[i] = i + 1;
    }
 ① a[10] = 10;     //此句越界
    a[14] = 20;     //此句越界
    a[15] = 30;     //此句越界
 ② //a[10] = '\xfd'; //此句越界
    cout << "addr:" << hex << (void*)(a) << endl;
    for (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*)(a + i) << ":" << a[i] << endl;

    return 0;
}
```



a[10] = 10; 越界写入，程序出现弹窗，退出代码为3

与动态申请的情况3（只有//p[10] = '\xfd';注释）相似，

但动态内存退出代码为一个很大的负数

VS2022 x86/Debug环境下运行

2.①放开，②注释，观察运行结果

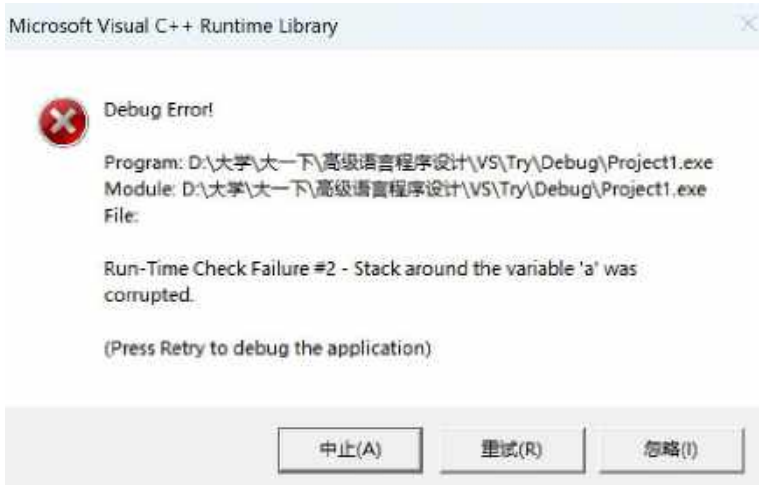
```
addr:00EFFCD4
00EFFCC4:cccccccc
00EFFCC8:fffffffd
00EFFCCC:cccccccc
00EFFCD0:cccccccc
00EFFCD4:1
00EFFCD8:2
00EFFCDC:3
00EFFCE0:4
00EFFCE4:5
00EFFCE8:6
00EFFCEC:7
00EFFCF0:8
00EFFCF4:9
00EFFCF8:a
00EFFCFC:a
00EFFD00:41514a5b
00EFFD04:effd24
00EFFD08:5c66b3
00EFFD0C:14
00EFFD10:1e

D:\大学\大一下\高级
出，代码为 3。
```

§.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）

2、数组用 int a[10]; 形式

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

int main()
{
    int a[10];
    int i;
    for (i = 0; i < 10; i++) {
        a[i] = i + 1;
    }
 ① a[10] = 10;     //此句越界
    a[14] = 20;     //此句越界
    a[15] = 30;     //此句越界
 ② a[10] = '\xfd'; //此句越界
    cout << "addr:" << hex << (void*)(a) << endl;
    for (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*)(a + i) << ":" << a[i] << endl;

    return 0;
}
```



a[10] = 10;a[10] = '\xfd';越界写入，程序出现弹窗，
与动态申请的情况4（全部放开）不相似

VS2022 x86/Debug环境下运行

3.①②放开，观察运行结果

```
addr:003DFAE4
003DFAD4:cccccccc
003DFAD8:fffffffd
003DFADC:cccccccc
003DFAE0:cccccccc
003DFAE4:1
003DFAE8:2
003DFAEC:3
003DFAF0:4
003DFAF4:5
003DFAF8:6
003DFAFC:7
003DFB00:8
003DFB04:9
003DFB08:a
003DFB0C:fffffffd
003DFB10:114dfeca
003DFB14:3dfb34
003DFB18:d966b3
003DFB1C:14
003DFB20:1e

D:\大学\大一下\高级
出，代码为 3。
```

§.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）

2、数组用 int a[10]; 形式
VS2022 x86/Release
环境下运行：

程序均正常运行输出，
但 a[14] = 20;a[15] = 30;
这两句越界没有写入

1、①②全部注释，
观察运行结果

```
addr:00D9FA2C
00D9FA1C:ae1300
00D9FA20:13b1808
00D9FA24:13b10c8
00D9FA28:77061780
00D9FA2C:1
00D9FA30:2
00D9FA34:3
00D9FA38:4
00D9FA3C:5
00D9FA40:6
00D9FA44:7
00D9FA48:8
00D9FA4C:9
00D9FA50:a
00D9FA54:99993f8e
00D9FA58:ae168d
00D9FA5C:d9faa4
00D9FA60:ae1605
00D9FA64:1
00D9FA68:13b10c8

D:\大学\大一下\高级
退出，代码为 0。
```

2、①放开，②注释，
观察运行结果

```
addr:004FFEA8
004FFE98:f813d0
004FFE9C:f81300
004FFEA0:7c1a48
004FFEA4:7c5e70
004FFEA8:1
004FFEAC:2
004FFEB0:3
004FFEB4:4
004FFEB8:5
004FFEBC:6
004FFEC0:7
004FFEC4:8
004FFEC8:9
004FFECC:a
004FFED0:a
004FFED4:f9c98449
004FFED8:4fff20
004FFEDC:f81605
004FFEE0:1
004FFEE4:7c5e70

D:\大学\大一下\高级
退出，代码为 0。
```

3、①②放开，
观察运行结果

```
addr:00FBFDA0
00FBFD90:eb13d0
00FBFD94:eb1300
00FBFD98:10e1808
00FBFD9C:10e10c8
00FBFDA0:1
00FBFDA4:2
00FBFDA8:3
00FBFDAC:4
00FBFDB0:5
00FBFDB4:6
00FBFDB8:7
00FBFDBC:8
00FBFDC0:9
00FBFDC4:a
00FBFDC8:fffffffd
00FBFDCC:a6d05e0
00FBFDD0:eb168d
00FBFDD4:fbfe1c
00FBFDD8:eb1605
00FBFDDC:1

D:\大学\大一下\高级
退出，代码为 0。
```

§.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）

2、数组用 int a[10]; 形式
Dev 32bit-Debug
Dev 32bit-Release
Linux环境下运行：

程序均正常运行输出

1、①②全部注释，观察运行结果

```
addr:0x78fe94
0x78fe84:4cf007
0x78fe88:78ffcc
0x78fe8c:7781e170
0x78fe90:187f2ada
0x78fe94:1
0x78fe98:2
0x78fe9c:3
0x78fea0:4
0x78fea4:5
0x78fea8:6
0x78feac:7
0x78feb0:8
0x78feb4:9
0x78feb8:a
0x78febc:a
0x78fec0:40bc00
0x78fec4:78fee0
0x78fec8:78ff68
0x78fecc:14
0x78fed0:1e
```

2、①放开，②注释，观察运行结果

```
addr:0x78fe94
0x78fe84:4cf007
0x78fe88:78ffcc
0x78fe8c:7781e170
0x78fe90:eecedd8
0x78fe94:1
0x78fe98:2
0x78fe9c:3
0x78fea0:4
0x78fea4:5
0x78fea8:6
0x78feac:7
0x78feb0:8
0x78feb4:9
0x78feb8:a
0x78febc:a
0x78fec0:40bc10
0x78fec4:78fee0
0x78fec8:78ff68
0x78fecc:14
0x78fed0:1e
```

3、①②放开，观察运行结果

```
addr:0x78fe94
0x78fe84:4cf007
0x78fe88:78ffcc
0x78fe8c:7781e170
0x78fe90:1533bf1
0x78fe94:1
0x78fe98:2
0x78fe9c:3
0x78fea0:4
0x78fea4:5
0x78fea8:6
0x78feac:7
0x78feb0:8
0x78feb4:9
0x78feb8:a
0x78febc:a
0x78fec0:40bc10
0x78fec4:78fee0
0x78fec8:78ff68
0x78fecc:14
0x78fed0:1e
```

§.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

答：内存越界是指程序访问了未分配给它的内存区域，可能发生在静态数组或者动态分配的内存块中。由于这些内存区域的分配由操作系统管理，一旦超出界限，后果不可预测，可能覆盖其他重要的数据或导致程序崩溃。

不同环境下的表现：在 Debug 模式下，很多越界问题会通过内存填充、保护机制等被检测到。然而，在Release模式下，这些保护通常会被移除，因此越界行为可能不被察觉，甚至不会引发崩溃，但仍然可能导致数据的不可预测更改或崩溃。

编译器与调试器如何防范越界：

1. Debug 模式：调试模式下，编译器通常会加入额外的内存检查和保护。比如VS的Debug模式会在内存块的两端加上“哨兵”或保护字节，确保如果写入越界，这些保护字节会被修改，触发调试器警报。

2. 内存填充：在动态分配和释放内存时，Debug 模式往往会用特殊字节（如0xCC或0xFD）填充未使用或释放的内存区域，以检测对未初始化或释放后内存的访问。

3. 动态分配的内存块管理：动态内存管理器如malloc和free会维护分配的内存块信息，在释放内存时会检查是否存在非法的访问行为。

为了尽量防范内存越界问题，应该养成良好的习惯：

1. 仔细管理数组与内存大小：在定义数组或申请动态内存时，确保准确计算所需大小，避免操作超过分配的内存区域。
2. 使用安全的函数：尽量使用安全版本的函数，如 `strncpy` 代替 `strcpy`，以避免字符串拷贝时发生越界。
3. 边界检查：在操作数组时，检查访问的索引是否在合法范围内。
4. 养成良好的释放内存习惯：在使用malloc或其他内存分配函数后，确保及时free掉分配的内存，以避免内存泄漏，并且避免在free之后继续访问该内存区域。