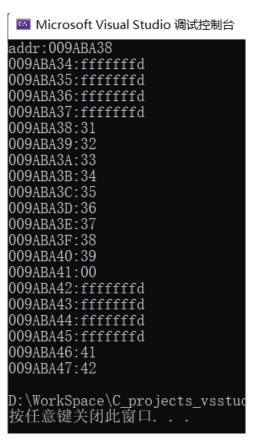
# §. 关于动态内存申请后越界访问的深度讨论

★ 如何判断动态申请越界(C方式, 注意源程序后缀为.c)

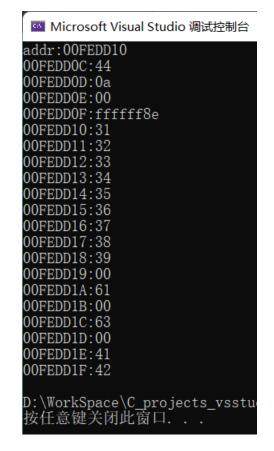
```
在VS2022的x86/Debug模式下运行:
#define <u>CRT_SECURE_NO_WARNINGS</u>
#include <stdio.h>
                                                     1、①②③全部注释,观察运行结果
                                                     2、①放开,②③注释,观察运行结果
#include <stdlib.h>
                                                     3、①③放开,②注释,观察运行结果
#include <string.h>
                                                     4、①②③全部放开,观察运行结果
int main()
                                                     结论: VS的Debug模式是如何判断
                                                          动态申请内存访问越界的?
   char *p;
   p = (char *) malloc(10 * sizeof(char));
                                                     再观察下面四种环境下的运行结果:
   if (p == NULL)
                                                        VS2022 x86/Release
      return -1:
                                                        Dev 32bit-Debug
   strcpy(p, "123456789");
                                                        Dev 32bit-Release
  p[10] = 'a'; //此句越界
                                                        Linux
   p[14] = 'A': //此句越界
                                                     每种讨论的结果可截图+文字说明,
   p[15] = 'B'; //此句越界
                                                     如果几种环境的结果一致,用一个
  p[10] = '\xfd'; //此句越界
                                                     环境的截图+文字说明即可(可加页)
   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<mark>越界读</mark>
      printf("%p:%02x\n", (p+i), p[i]);
  free(p);
   return 0:
```

### 1、①②③全部注释,观察运行结果

### VS2022的x86/Debug



#### VS2022 x86/Release



### Dev 32bit-Debug



#### D:\WorkSpace\C projects vs

```
addr:001F1630
001F162C:ffffffd0
001F162D:49
001F162E:00
001F162F:0e
001F1630:31
001F1631:32
001F1632:33
001F1633:34
001F1634:35
001F1635:36
001F1636:37
001F1637:38
001F1638:39
001F1639:00
001F163A:4e
001F163B:00
001F163C:55
001F163D:53
001F163E:41
001F163F:42
Process exited after 0.04
```

请按任意键继续. . . 🗕

### 1、①②③全部注释,观察运行结果

#### Dev 32bit-Release

### D:\WorkSpace\C\_projects\_vs: addr:00BC1630 00BC162C:ffffffbe 00BC162D:ffffffa2 00BC162E:00 00BC162F:0e 00BC1630:31 00BC1631:32 00BC1632:33 00BC1633:34 00BC1634:35 00BC1635:36 00BC1636:37 00BC1637:38 00BC1638:39 00BC1639:00 00BC163A:4e 00BC163B:00 00BC163C:55 00BC163D:53 00BC163E:41 00BC163F:42 Process exited after 0.01 清按任意键继续. . . \_

#### Linux



[u1850772@101080 ~	·]\$	./run
addr:0x211a2a0		
0x211a29c:00		
0x211a29d:00		
0x211a29e:00		
0x211a29f:00		
0x211a2a0:31		
0x211a2a1:32		
0x211a2a2:33		
0x211a2a3:34		
0x211a2a4:35		
0x211a2a5:36		
0x211a2a6:37		
0x211a2a7:38		
0x211a2a8:39		
0x211a2a9:00		
0x211a2aa:00		
0x211a2ab:00		
0x211a2ac:00		
0x211a2ad:00		
0x211a2ae:41		
0x211a2af:42		
[u1850772@101080 ~	·]\$	

越界写 可以写入 不同编译条件下没大 的区别 都可以读到越界写入 的内容

### 越界读

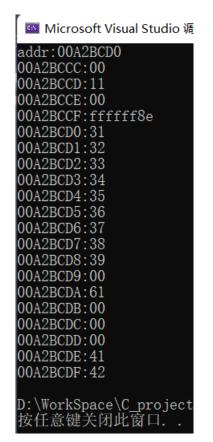
读取到的内容和编译 条件有关 Vs Debug 是固定值 Vs Release 是随机值 Dev Debug 是随机值 Dev Release 是随机值 Linux 是 0

### 2、①放开,②③注释,观察运行结果

### VS2022的x86/Debug

#### Microsoft Visual Studio 训 addr:00BDBE90 00BDBE8C:ffffffd 00BDBE8D:ffffffd 00BDBE8E:ffffffd 00BDBE8F:ffffffd 00BDBE90:31 00BDBE91:32 00BDBE92:33 00BDBE93:34 00BDBE94:35 00BDBE95:36 00BDBE96:37 00BDBE97:38 00BDBE98:39 00BDBE99:00 00BDBE9A:61 00BDBE9B:ffffffd 00BDBE9C:ffffffd 00BDBE9D:ffffffd 00BDBE9E:41 00BDBE9F:42 D:\WorkSpace\C\_project 按任意键关闭此窗口. .

#### VS2022 x86/Release



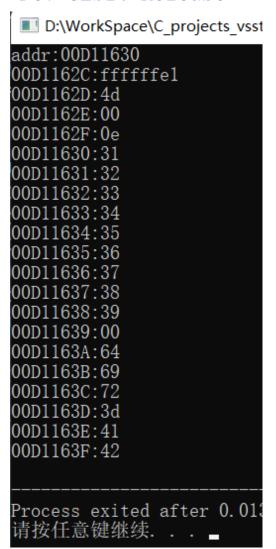
### Dev 32bit-Debug



```
D:\WorkSpace\C projects vsst
addr:00D01630
00D0162C:ffffffc9
00D0162D:ffffff97
00D0162E:00
00D0162F:0e
00D01630:31
00D01631:32
00D01632:33
00D01633:34
00D01634:35
00D01635:36
00D01636:37
00D01637:38
00D01638:39
00D01639:00
00D0163A:64
00D0163B:69
00D0163C:72
00D0163D:3d
00D0163E:41
00D0163F:42
Process exited after 0.033
请按任意键继续. . .
```

### 2、①放开,②③注释,观察运行结果

#### Dev 32bit-Release



#### Linux



[u1850772@101080 ~]\$ ./run addr:0x19662a0 0x196629c:00 0x196629d:00 0x196629e:00 0x196629f:00 0x19662a0:31 0x19662a1:32 0x19662a2:33 0x19662a3:34 0x19662a4:35 0x19662a5:36 0x19662a6:37 0x19662a7:38 0x19662a8:39 0x19662a9:00 0x19662aa:61 0x19662ab:00 0x19662ac:00 0x19662ad:00 0x19662ae:41 0x19662af:42 [u1850772@101080 ~]\$

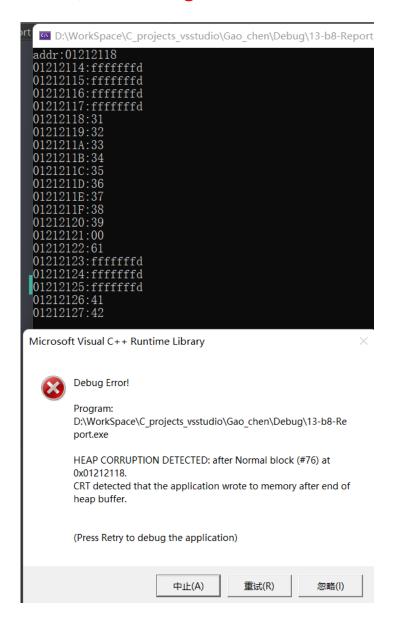
越界写 可以写入 不同编译条件下没大 的区别 都可以读到越界写入 的内容

### 越界读

读取到的内容和编译 条件有关 Vs Debug 是fffffffd Vs Release 是随机值 Dev Debug 是随机值 Dev Release 是随机值 Linux 是 0

### 3、①③放开,②注释,观察运行结果

### VS2022的x86/Debug



#### VS2022 x86/Release

Microsoft Visual Studio 调试控制: addr:00902020 0090201C:ffffffd0 0090201D:57 0090201E:00 0090201F:0e 00902020:31 00902021:32 00902022:33 00902023:34 00902024:35 00902025:36 00902026:37 00902027:38 00902028:39 00902029:00 0090202A:61 0090202B:00 0090202C:00 0090202D:00 0090202E:41 0090202F:42 D:\WorkSpace\C\_projects\_vss 按任意键关闭此窗口. . . **\_** 

### Dev 32bit-Debug



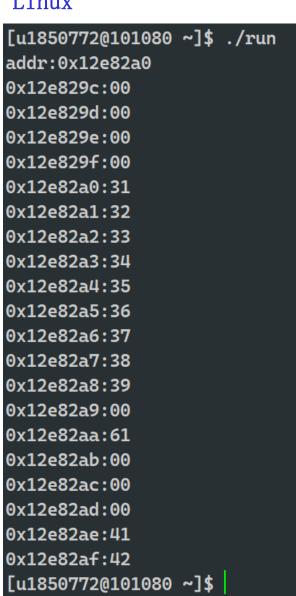
```
D:\WorkSpace\C projects vsstudio\G
addr:00D01630
00D0162C:48
00D0162D:ffffffb5
00D0162E:00
00D0162F:0e
00D01630:31
00D01631:32
00D01632:33
00D01633:34
00D01634:35
00D01635:36
00D01636:37
00D01637:38
00D01638:39
00D01639:00
00D0163A:64
00D0163B:69
00D0163C:72
00D0163D:3d
00D0163E:41
00D0163F:42
Process exited after 0.01436 s
请按任意键继续. . .
```

### 3、①③放开,②注释,观察运行结果

#### Dev 32bit-Release

```
■ D:\WorkSpace\C projects vsst
addr:00C21630
00C2162C:ffffffdf
00C2162D:ffffffd0
00C2162E:00
00C2162F:0e
00C21630:31
00C21631:32
00C21632:33
00C21633:34
00C21634:35
00C21635:36
00C21636:37
00C21637:38
00C21638:39
00C21639:00
00C2163A:64
00C2163B:69
00C2163C:72
00C2163D:3d
00C2163E:41
00C2163F:42
Process exited after 0.010
请按任意键继续...
```

#### Linux





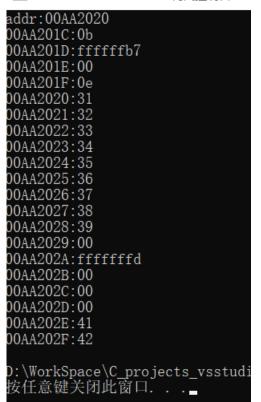
### 4、①②③全部放开,观察运行结果

### VS2022的x86/Debug

#### Microsoft Visual Studio 调试控制台 addr:01512310 0151230C:fffffffd 0151230D:fffffffd 0151230E:fffffffd 0151230F:ffffffd 01512310:31 01512311:32 01512312:33 01512313:34 01512314:35 01512315:36 01512316:37 01512317:38 01512318:39 01512319:00 0151231A:fffffffd 0151231B:ffffffd 0151231C:ffffffd 0151231D:fffffffd 0151231E:41 0151231F:42 D:\WorkSpace\C projects vsstudio\Gao 按任意键关闭此窗口. . .

#### VS2022 x86/Release

Microsoft Visual Studio 调试控制台



### Dev 32bit-Debug



D:\WorkSpace\C projects

```
addr:00971630
0097162C:ffffffe9
0097162D:67
0097162E:00
0097162F:0e
00971630:31
00971631:32
00971632:33
00971633:34
00971634:35
00971635:36
00971636:37
00971637:38
00971638:39
00971639:00
0097163A:64
0097163B:69
0097163C:72
0097163D:3d
0097163E:41
0097163F:42
Process exited after 0.
```

请按任意键继续. . .

### 4、①②③全部放开,观察运行结果

#### Dev 32bit-Release

```
D:\WorkSpace\C_projects vss
addr:00851630
0085162C:0a
0085162D:46
0085162E:00
0085162F:0e
0851630:31
00851631:32
0851632:33
0851633:34
00851634:35
00851635:36
0851636:37
00851637:38
00851638:39
00851639:00
0085163A:64
0085163B:69
0085163C:72
0085163D:3d
0085163E:41
0085163F:42
Process exited after 0.01
清按任意键继续...
```

#### Linux



```
[u1850772@101080 ~]$ ./run
addr:0x14082a0
0x140829c:00
0x140829d:00
0x140829e:00
0x140829f:00
0x14082a0:31
0x14082a1:32
0x14082a2:33
0x14082a3:34
0x14082a4:35
0x14082a5:36
0x14082a6:37
0x14082a7:38
0x14082a8:39
0x14082a9:00
0x14082aa:fffffffd
0x14082ab:00
0x14082ac:00
0x14082ad:00
0x14082ae:41
0x14082af:42
[u1850772@101080 ~]$
```

### 结论: VS的Debug模式如何判断动态内存访问越界

在没有自己定义申请内存的回收时 靠系统自己回收, 是不会进行动态申请访问内存越界的判断 在都不注释的情况下,尝试给p[11]和p[12] 或者p[...](越界范围) 均会报错.

在自己定义申请内存的回收时 Vs会检查当初分配的内存空间 查看是否越界.

VS DEBUG模式下,它应该是动态申请内存时, 对其中的数据(如10个数据单位) 以及之后的若干个数据完成初始化 还有之前的几个单位数据进行初始化 后面回收时,对应检查这么多单位



### 结论: 其他模式如何判断动态内存访问越界

## Vs Release 模式

不会对动态申请内存进行越界检查 原因可能在申请内存的未对齐初始化. 其不能够知道哪些数据是被非法访问的

### Dev Debug/Release模式

不会对动态申请内存进行越界检查申请内存时会初始化

### Linux 模式

不会对动态申请内存进行越界检查申请内存时会初始化为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)

```
在VS2022的x86/Debug模式下运行:
#define <u>CRT_SECURE_NO_WARNINGS</u>
#include <iostream>
                                                    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
#include <cstring>
                                                    3、①③放开,②注释,观察运行结果
using namespace std;
                                                    4、①②③全部放开,观察运行结果
int main()
                                                    结论: VS的Debug模式是如何判断
                                                         动态申请内存访问越界的?
   char *p;
   p = new(nothrow) char[10];
                                                    再观察下面四种环境下的运行结果:
   if (p == NULL)
                                                        VS2022 x86/Release
      return -1:
                                                        Dev 32bit-Debug
   strcpy(p, "123456789");
                                                        Dev 32bit-Release
  p[10] = 'a'; //此句越界
                                                        Linux
   p[14] = 'A': //此句越界
                                                    每种讨论的结果可截图+文字说明,
   p[15] = 'B'; //此句越界
                                                    如果几种环境的结果一致,用一个
   p[10] = '\xfd'; //此句越界
                                                    环境的截图+文字说明即可(可加页)
   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;
   delete[]p:
   return 0:
```

### 1、①②③全部注释,观察运行结果

### VS2022的x86/Debug

#### Microsoft Visual Studio 调试控制台 addr:00BF0A48 OOBFOA44:fffffffd OBFOA45:ffffffd 00BF0A46:ffffffd OOBFOA47:fffffffd 00BF0A48:31 00BF0A49:32 00BF0A4A:33 00BF0A4B:34 00BF0A4C:35 00BF0A4D:36 00BF0A4E:37 00BF0A4F:38 00BF0A50:39 00BF0A51:0 00BF0A52:ffffffd OOBFOA53:ffffffdd 00BF0A54:ffffffd 00BF0A55:ffffffd 00BF0A56:41 00BF0A57:42 D:\WorkSpace\C\_projects\_vsstudio 按任意键关闭此窗口. . . **\_**

#### VS2022 x86/Release

### Microsoft Visual Studio 调试控制台addr:0112ADD0 0112ADCC:0

0112ADCC:0 0112ADCD:27 0112ADCE:0 0112ADCF:ffffff8e 0112ADD0:31 0112ADD1:32 0112ADD2:33 0112ADD3:34 0112ADD4:35 0112ADD5:36 0112ADD6:37 0112ADD7:38 0112ADD8:39 0112ADD9:0 0112ADDA:0 0112ADDB:0 0112ADDC:0 0112ADDD:0 0112ADDE:41 0112ADDF:42 D:\WorkSpace\C\_projects\_vsstudio 按任意键关闭此窗口. . .

#### Dev 32bit-Debug



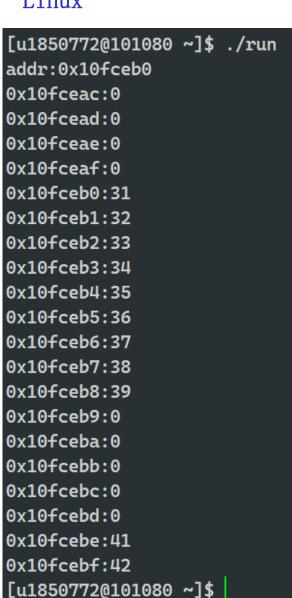
```
D:\WorkSpace\C projects \u00e4
addr:0xe76d70
0xe76d6c:67
0xe76d6d:ffffff93
0xe76d6e:0
0xe76d6f:e
0xe76d70:31
0xe76d71:32
0xe76d72:33
0xe76d73:34
0xe76d74:35
0xe76d75:36
0xe76d76:37
0xe76d77:38
0xe76d78:39
0xe76d79:0
0xe76d7a:0
0xe76d7b:0
0xe76d7c:0
0xe76d7d:0
0xe76d7e:41
0xe76d7f:42
Process exited after 0.
请按任意键继续...
```

### 1、①②③全部注释,观察运行结果

#### Dev 32bit-Release

```
■ D:\WorkSpace\C_projects_vsstudio\Gao_c
addr:0x1d6d70
0x1d6d6c:fffffffe
0x1d6d6d:ffffff96
0x1d6d6e:0
0x1d6d6f:e
0x1d6d70:31
0x1d6d71:32
0x1d6d72:33
0x1d6d73:34
0x1d6d74:35
0x1d6d75:36
0x1d6d76:37
0x1d6d77:38
0x1d6d78:39
0x1d6d79:0
0x1d6d7a:0
0x1d6d7b:0
0x1d6d7c:0
0x1d6d7d:0
0x1d6d7e:41
0x1d6d7f:42
Process exited after 0.03037 secor
请按任意键继续. . .
```

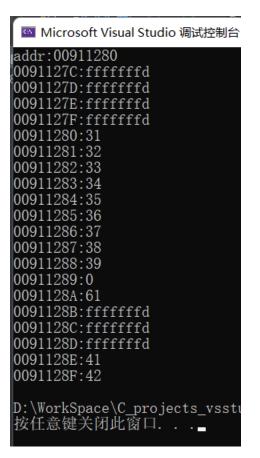
#### Linux



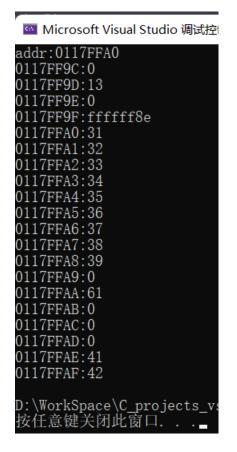


### 2、①放开,②③注释,观察运行结果

### VS2022的x86/Debug



#### VS2022 x86/Release



### Dev 32bit-Debug



■ D:\WorkSpace\C\_projects\_vss

```
addr:0xd86d70
0xd86d6c:ffffff87
0xd86d6d:5b
0xd86d6e:0
0xd86d6f:e
0xd86d70:31
0xd86d71:32
0xd86d72:33
0xd86d73:34
0xd86d74:35
0xd86d75:36
0xd86d76:37
0xd86d77:38
0xd86d78:39
0xd86d79:0
0xd86d7a:0
0xd86d7b:0
0xd86d7c:0
0xd86d7d:0
0xd86d7e:41
0xd86d7f:42
Process exited after 0.02
请按任意键继续. . .
```

### 2、①放开,②③注释,观察运行结果

#### Dev 32bit-Release

```
■ D:\WorkSpace\C projects vsstuc
addr:0xde6d70
0xde6d6c:ffffffa4
0xde6d6d:ffffffb9
0xde6d6e:0
0xde6d6f:e
0xde6d70:31
0xde6d71:32
0xde6d72:33
0xde6d73:34
0xde6d74:35
0xde6d75:36
0xde6d76:37
0xde6d77:38
0xde6d78:39
0xde6d79:0
0xde6d7a:0
0xde6d7b:0
0xde6d7c:0
0xde6d7d:0
0xde6d7e:41
0xde6d7f:42
Process exited after 0.0196
请按任意键继续. . . 🗕
```

#### Linux



```
addr:0x1ceaeb0
0x1ceaeac:0
0x1ceaead:0
0x1ceaeae:0
0x1ceaeaf:0
0x1ceaeb0:31
0x1ceaeb1:32
0x1ceaeb2:33
0x1ceaeb3:34
0x1ceaeb4:35
0x1ceaeb5:36
0x1ceaeb6:37
0x1ceaeb7:38
0x1ceaeb8:39
0x1ceaeb9:0
0x1ceaeba:61
0x1ceaebb:0
0x1ceaebc:0
0x1ceaebd:0
0x1ceaebe:41
0x1ceaebf:42
[u1850772@101080 ~]$
```

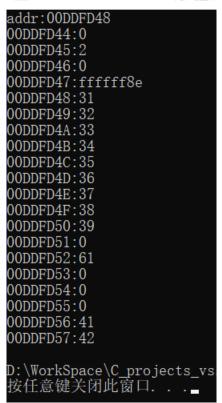
### 3、①③放开,②注释,观察运行结果

### VS2022的x86/Debug



#### VS2022 x86/Release

Microsoft Visual Studio 调试控制



#### Dev 32bit-Debug



```
D:\WorkSpace\C_projects_vsstu
```

```
addr:0xde6d70
0xde6d6c:69
0xde6d6d:6
0xde6d6e:0
0xde6d6f:e
0xde6d70:31
0xde6d71:32
0xde6d72:33
0xde6d73:34
0xde6d74:35
0xde6d75:36
0xde6d76:37
0xde6d77:38
0xde6d78:39
0xde6d79:0
0xde6d7a:0
0xde6d7b:0
0xde6d7c:0
0xde6d7d:0
0xde6d7e:41
0xde6d7f:42
Process exited after 0.030
请按任意键继续...
```

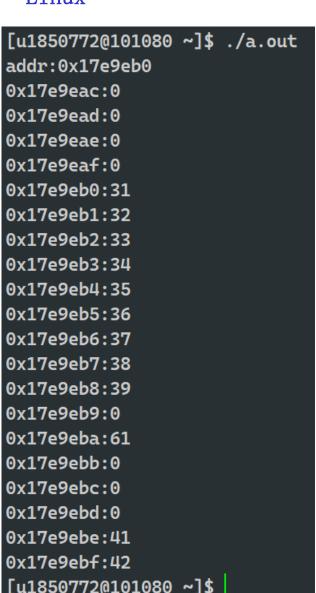
### 3、①③放开,②注释,观察运行结果

#### Dev 32bit-Release

#### ■ D:\WorkSpace\C projects vss

```
addr:0xe36d70
0xe36d6c:ffffffc9
Oxe36d6d:ffffffaa
0xe36d6e:0
0xe36d6f:e
0xe36d70:31
0xe36d71:32
0xe36d72:33
0xe36d73:34
0xe36d74:35
0xe36d75:36
0xe36d76:37
0xe36d77:38
0xe36d78:39
0xe36d79:0
0xe36d7a:0
0xe36d7b:0
0xe36d7c:0
0xe36d7d:0
0xe36d7e:41
0xe36d7f:42
Process exited after 0.02
请按任意键继续...
```

#### Linux



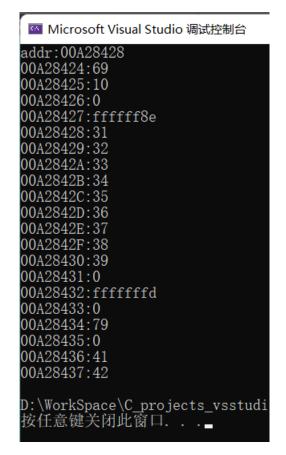


### 4、①②③全部放开,观察运行结果

### VS2022的x86/Debug

#### Microsoft Visual Studio 调试控制台 addr:009811B8 009811B4:ffffffd 009811B5:ffffffd 009811B6:ffffffd 009811B7:ffffffd 009811B8:31 009811B9:32 009811BA:33 009811BB:34 009811BC:35 009811BD:36 009811BE:37 009811BF:38 009811C0:39 009811C1:0 009811C2:fffffffd 009811C3:fffffffd 009811C4:ffffffd 009811C5:ffffffd 009811C6:41 009811C7:42 D:\WorkSpace\C\_projects\_vsst 按任意键关闭此窗口. . . \_

#### VS2022 x86/Release



### Dev 32bit-Debug

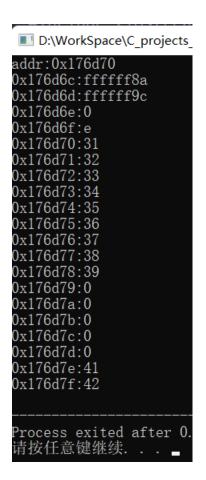


```
D:\WorkSpace\C project
addr:0xcf6d70
0xcf6d6c:ffffffe2
0xcf6d6d:ffffff9a
0xcf6d6e:0
0xcf6d6f:e
0xcf6d70:31
0xcf6d71:32
0xcf6d72:33
0xcf6d73:34
0xcf6d74:35
0xcf6d75:36
0xcf6d76:37
0xcf6d77:38
0xcf6d78:39
0xcf6d79:0
0xcf6d7a:0
0xcf6d7b:0
0xcf6d7c:0
0xcf6d7d:0
0xcf6d7e:41
0xcf6d7f:42
Process exited after
```

请按任意键继续. . .

### 4、①②③全部放开,观察运行结果

#### Dev 32bit-Release



#### Linux



```
addr:0x1f6beb0
0x1f6beac:0
0x1f6bead:0
0x1f6beae:0
0x1f6beaf:0
0x1f6beb0:31
0x1f6beb1:32
0x1f6beb2:33
0x1f6beb3:34
0x1f6beb4:35
0x1f6beb5:36
0x1f6beb6:37
0x1f6beb7:38
0x1f6beb8:39
0x1f6beb9:0
0x1f6beba:ffffffd
0x1f6bebb:0
0x1f6bebc:0
0x1f6bebd:0
0x1f6bebe:41
0x1f6bebf:42
[u1850772@101080 ~]$
```

### 结论: VS的Debug模式如何判断动态内存访问越界(同C方式)

在没有自己定义申请内存的回收时 靠系统自己回收, 是不会进行动态申请访问内存越界的判断 在都不注释的情况下,尝试给p[11]和p[12] 或者p[...](越界范围) 均会报错.

在自己定义申请内存的回收时 Vs会检查当初分配的内存空间 查看是否越界.

VS DEBUG模式下,它应该是动态申请内存时,对其中的数据(如10个数据单位)以及之后的若干个数据完成初始化还有之前的几个单位数据进行初始化后面回收时,对应检查这么多单位



### 结论: 其他模式如何判断动态内存访问越界

### Vs Release 模式同C方式

不会对动态申请内存进行越界检查 原因可能在申请内存的未对齐初始化. 其不能够知道哪些数据是被非法访问的

### Dev Debug/Release模式 同C方式

不会对动态申请内存进行越界检查申请内存时会初始化

### Linux 模式 同C方式

不会对动态申请内存进行越界检查申请内存时会初始化为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)

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
int main()
    char str[10] = "123456789";
   int num[10] = { 1,2,3,4,5,6,7,8,9,0};
#if 0
   str[14] = 'A'; //越界
   str[15] = 'B';
   str[16] = 'C';
   num[14] = 10;
   num[15] = 11;
   num[16] = 12;
#endif
    printf(" str:%p\t\t\tnum:%p\n", str,num);
    for (int i = -5; i \le 20; i++)
       if (i == 14)
            printf("---
        printf(
            " %p:%02x\t\t\t\t%p:%02x\n",
            (str + i), str[i],
            (num + i), num[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 x86Release 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)

## 1、取消越界写的结果

## VS2022的x86/Debug

### VS2022 x86/Release

## Dev 32bit-Debug



#### 亟 Microsoft Visual Studio 调试控制台

0076F9EF:ffffffcc	0076F9B0:00
0076F9F0:ffffffcc	0076F9B4:ccccccc
0076F9F1:ffffffcc	0076F9B8:fffffffd
0076F9F2:ffffffcc	0076F9BC:ccccccc
0076F9F3:ffffffcc	0076F9C0:ccccccc
0076F9F4:31	0076F9C4:01
0076F9F5:32	0076F9C8:02
0076F9F6:33	0076F9CC:03
0076F9F7:34	0076F9D0:04
0076F9F8:35	0076F9D4:05
0076F9F9:36	0076F9D8:06
0076F9FA:37	0076F9DC:07
0076F9FB:38	0076F9E0:08
0076F9FC:39	0076F9E4:09
0076F9FD:00	0076F9E8:00
0076F9FE:ffffffcc	0076F9EC:ccccccc
0076F9FF:ffffffcc	0076F9F0:ccccccc
0076FA00:ffffffcc	0076F9F4:34333231
0076FA01:ffffffcc	0076F9F8:38373635
0076FA02:ffffffcc	0076F9FC:cccc0039
0076FA03:ffffffcc	0076FA00:ccccccc
0076FA04:ffffffbf	0076FA04:73640ebf
0076FA05:0e	0076FA08:76fa28
0076FA06:64	0076FA0C:b42223
0076FA07:73	0076FA10:01
0076FA08:28	0076FA14:97ba68

#### Microsoft Visual Studio 调试控制台

str:00AFFB38 00AFFB33:00 00AFFB34:00 00AFFB36:00 00AFFB37:00 00AFFB38:31 00AFFB39:32 00AFFB38:33 00AFFB3B:34 00AFFB3B:35 00AFFB3B:36 00AFFB3F:38 00AFFB3F:38 00AFFB40:39 00AFFB41:00 00AFFB42:37 00AFFB43:00	num:00AFFB10 00AFFAFC:37109e 00AFFB00:00 00AFFB04:affb00 00AFFB08:affb00 00AFFB10:01 00AFFB10:01 00AFFB18:03 00AFFB18:03 00AFFB1C:04 00AFFB20:05 00AFFB20:05 00AFFB24:06 00AFFB28:07 00AFFB20:08 00AFFB30:09 00AFFB30:09 00AFFB30:09
00AFFB44:00	00AFFB3C:38373635
00AFFB44:ffffff85	00AFFB40:370039
00AFFB45:ffffffe6	00AFFB44:6126e685
00AFFB46:26	00AFFB48:affb90
00AFFB47:61	00AFFB4C:3712bd
00AFFB48:ffffff90	00AFFB50:01
00AFFB49:fffffffb	00AFFB54:e17f58
00AFFB4A:ffffffaf	00AFFB58:e188c0
00AFFB4B:00	00AFFB5C:6126e65d
00AFFB4C:ffffffbd	00AFFB60:371345

#### D:\WorkSpace\C\_projects\_vsstudio\Gao\_chen\13-b8-test1\13-b8-test.exe

str:0065FEC2 0065FEBD:00 0065FEBE:00 0065FEBF:00 0065FEC0:50 0065FEC2:31 0065FEC3:32 0065FEC4:33 0065FEC5:34 0065FEC6:35 0065FEC7:36 0065FEC7:36 0065FEC8:37 0065FEC8:37 0065FEC8:39 0065FEC8:39 0065FECC:0a 0065FECC:0a	num:0065FE98 0065FE84:160 0065FE88:65ffcc 0065FE8C:7542dcd0 0065FE90:c758b556 0065FE94:fffffffe 0065FE98:01 0065FE9C:02 0065FEA0:03 0065FEA4:04 0065FEA8:05 0065FEAC:06 0065FEB0:07 0065FEB0:07 0065FEB8:09 0065FEB8:09 0065FEBC:00 0065FEC0:32311650 0065FEC4:36353433 0065FEC8:393837
0065FECE:00	0065FEC8:393837
0065FECF:00	0065FECC:0d
0065FED0:ffffffe4	0065FED0:1e15e4
0065FED1:15	0065FED4:45
0065FED2:1e	0065FED8:65ff68
0065FED3:00	0065FEDC:401386
0065FED4:45	0065FEE0:01
0065FED5:00	0065FEE4:1e15e0
0065FED6:00	0065FEE8:1e1918

### 1、取消越界写的结果

#### Dev 32bit-Release

#### Linux



#### D:\WorkSpace\C projects vsstudio\Gao chen\13-b8-test1\13-b8-test.exe

4	str:0064FEC2	num:0064FE98
111	0064FEBD:00	0064FE84:150
ı	0064FEBE:00	0064FE88:64ffcc
ı	0064FEBF:00	0064FE8C:7542dcd0
ı	0064FEC0:50	0064FE90:12bd431
ı	0064FEC1:16	0064FE94:fffffffe
ı	0064FEC2:31	0064FE98:01
ı	0064FEC3:32	0064FE9C:02
ı	0064FEC4:33	0064FEA0:03
ı	0064FEC5:34	0064FEA4:04
ı	0064FEC6:35	0064FEA8:05
ı	0064FEC7:36	0064FEAC:06
ı	0064FEC8:37	0064FEB0:07
L	0064FEC9:38	0064FEB4:08
ı	0064FECA:39	0064FEB8:09
ı	0064FECB:00	0064FEBC:00
ı	0064FECC:0a	0064FEC0:32311650
ı	0064FECD:00	0064FEC4:36353433
ı	0064FECE:00	0064FEC8:393837
ı	0064FECF:00	0064FECC:0d
ı	0064FED0:ffffffe4	0064FED0:bf15e4
ı	0064FED1:15	0064FED4:45
ı	0064FED2:ffffffbf	0064FED8:64ff68
ı	0064FED3:00	0064FEDC:401386
ı	0064FED4:45	0064FEE0:01
ı	0064FED5:00	0064FEE4:bf15e0
	0064FED6:00	0064FEE8:bf1918

***************************************	
str:0x7ffefe934762	num:0x7ffefe934730
0x7ffefe93475d:00	0x7ffefe93471c:7f77
0x7ffefe93475e:00	0x7ffefe934720:01
0x7ffefe93475f:00	0x7ffefe934724:00
0x7ffefe934760:40	0x7ffefe934728:4006b9
0x7ffefe934761:48	0x7ffefe93472c:00
0x7ffefe934762:31	0x7ffefe934730:01
0x7ffefe934763:32	0x7ffefe934734:02
0x7ffefe934764:33	0x7ffefe934738:03
0x7ffefe934765:34	0x7ffefe93473c:04
0x7ffefe934766:35	0x7ffefe934740:05
0x7ffefe934767:36	0x7ffefe934744:06
0x7ffefe934768:37	0x7ffefe934748:07
0x7ffefe934769:38	0x7ffefe93474c:08
0x7ffefe93476a:39	0x7ffefe934750:09
0x7ffefe93476b:00	0x7ffefe934754:00
0x7ffefe93476c:0a	0x7ffefe934758:4004f0
0x7ffefe93476d:00	0x7ffefe93475c:00
0x7ffefe93476e:00	0x7ffefe934760:32314840
0x7ffefe93476f:00	0x7ffefe934764:36353433
0x7ffefe934770:ffffffd0	0x7ffefe934768:393837
0x7ffefe934771:06	0x7ffefe93476c:0f
0x7ffefe934772:40	0x7ffefe934770:4006d0
0x7ffefe934773:00	0x7ffefe934774:00
0x7ffefe934774:00	0x7ffefe934778:478fdd85
0x7ffefe934775:00	0x7ffefe93477c:7f77
0x7ffefe934776:00	0x7ffefe934780:47a48ac8

### 越界读

读取到的内容和编译 条件有关

#### Str:

Vs Debug 是固定值 Vs Release 是随机值 Dev Debug 是随机值 Dev Release 是随机值 Linux 是 0

#### Int:

Vs Debug 是固定值 Vs Release 是随机值 Dev Debug 是随机值 Dev Release 是随机值 Linux 是 0

## 2、越界写的结果

## VS2022的x86/Debug

### VS2022 x86/Release

## Dev 32bit-Debug



D:\Work	:Space\C_projects_	vsstudio\Gao_chen\De	ebug\13-b8-test1.exe	
str:00A1 00A1FC13 00A1FC16 00A1FC16 00A1FC17 00A1FC18 00A1FC19 00A1FC1B 00A1FC1D 00A1FC1D 00A1FC1D 00A1FC1D 00A1FC20 00A1FC23 00A1FC23 00A1FC24 00A1FC25 00A1FC25 00A1FC25	FC18 :00 :00 :00 :00 :00 :00 :00 :00 :00 :35 :36 :37 :38 :39 :00 :41 :42		num:00A1FBE8 00A1FBD4:00 00A1FBD8:cccccc 00A1FBD0:fffffff 00A1FBE0:cccccc 00A1FBE4:cccccc 00A1FBE8:01 00A1FBE6:02 00A1FBF0:03 00A1FBF6:05 00A1FBF6:05 00A1FBF0:07 00A1FC00:07 00A1FC00:07 00A1FC00:00 00A1FC10:0a 00A1FC10:0a 00A1FC18:0c 00A1FC18:0c 00A1FC18:0c	fd cc
Microsof	Module: D:\WorkSpace\C_p File: Run-Time Check Facorrupted.	rojects_vsstudio\Gao_ch	en\Debug\13-b8-test1.exe en\Debug\13-b8-test1.exe the variable 'str' was	
		中止(A)	重试(R) 忽略(I)	

☑ Microsoft Visual Studio 调试控制台	
0100F763:00 0100F764:00 0100F765:00 0100F766:00 0100F767:00 0100F768:31 0100F768:32 0100F76A:33 0100F76B:34 0100F76B:35 0100F76B:36 0100F76B:37 0100F76B:38 0100F771:00 0100F771:00 0100F772:05 0100F773:77 0100F774:45 0100F775:13	0100F72C:f0109e 0100F730:00 0100F734:100f730 0100F738:100f730 0100F73C:124ba50 0100F740:01 0100F744:02 0100F748:03 0100F74C:04 0100F750:05 0100F750:05 0100F750:08 0100F760:09 0100F760:09 0100F768:34333231 0100F76C:38373635 0100F770:77050039 0100F774:f01345
0100F776:ffffffff0 0100F777:00 0100F778:ffffffd8 0100F779:15 0100F77A:fffffffeb 0100F77B:fffffffb4 0100F77C:fffffffc4 D:\WorkSpace\C_projects_vsstudio	0100F778:b4eb15d8 0100F77C:100f7c4 0100F780:f012bd 0100F784:01 0100F788:124ba50 0100F78C:1248b48 0100F790:b4eb1560 \Gao_chen\Release\13-b8-test1. exe

D:\WorkSpace\C_projects_vsstudio\Gao_chen\13-b8-test1\13-b8-test.exe		
0065FEBE:00	0065FE88:65ffcc	
0065FEBF:00	0065FE8C:7542dcd0	
0065FEC0:70	0065FE90:fe657106	
0065FEC1:16	0065FE94:fffffffe	
0065FEC2:31	0065FE98:01	
0065FEC3:32	0065FE9C:02	
0065FEC4:33	0065FEA0:03	
0065FEC5:34	0065FEA4:04	
0065FEC6:35	0065FEA8:05	
0065FEC7:36	0065FEAC:06	
0065FEC8:37	0065FEB0:07	
0065FEC9:38	0065FEB4:08	
0065FECA:39	0065FEB8:09	
0065FECB:00	0065FEBC:00	
0065FECC:0a	0065FEC0:32311670	
0065FECD:00	0065FEC4:36353433	
0065FECE:00	0065FEC8:393837	
0065FECF:00	0065FECC:0d	
0065FED0:0a	0065FED0:0a	
0065FED1:00	0065FED4:0b	
0065FED2:00	0065FED8:0c	
0065FED3:00	0065FEDC:401386	
0065FED4:0b	0065FEE0:01	
0065FED5:00	0065FEE4:cc15e0	
0065FED6:00	0065FEE8:cc1918	
Process exited after 0.06385 書按任音键继续	seconds with return value 0	

### 2、越界写的结果

#### Dev 32bit-Release

### Linux



str:0064FEC2 0064FEBD:00 0064FEBE:00 0064FEC0:70 0064FEC1:16 0064FEC2:31 0064FEC3:32 0064FEC4:33 0064FEC5:34 0064FEC6:35 0064FEC7:36 0064FEC8:37 0064FEC8:37 0064FEC9:38 0064FEC3:39 0064FECB:00 0064FECC:0a 0064FECC:0a 0064FECC:00 0064FECC:00 0064FECE:00 0064FECE:00	num:0064FE98 0064FE84:150 0064FE88:64ffcc 0064FE8C:7542dcd0 0064FE90:229e643e 0064FE94:fffffffe 0064FE98:01 0064FE9C:02 0064FEA0:03 0064FEA0:03 0064FEA0:06 0064FEB0:07 0064FEB0:07 0064FEB0:07 0064FEB0:07 0064FEB0:07	[u1850772@101080 ~]\$ ./a str:0x7ffd8e3b8312 0x7ffd8e3b830d:00 0x7ffd8e3b830e:00 0x7ffd8e3b830f:00 0x7ffd8e3b8310:fffffff0 0x7ffd8e3b8311:ffffff83 0x7ffd8e3b8312:31 0x7ffd8e3b8312:31 0x7ffd8e3b8313:32 0x7ffd8e3b8315:34 0x7ffd8e3b8315:34 0x7ffd8e3b8316:35 0x7ffd8e3b8317:36 0x7ffd8e3b8317:36 0x7ffd8e3b8318:0a 0x7ffd8e3b831a:00 0x7ffd8e3b831a:00 0x7ffd8e3b831c:0a 0x7ffd8e3b831c:0a 0x7ffd8e3b831d:00 0x7ffd8e3b831d:00 0x7ffd8e3b831d:00	num:0x7ffd8e3b82e0 0x7ffd8e3b82cc:7f95 0x7ffd8e3b82d0:01 0x7ffd8e3b82d4:00 0x7ffd8e3b82d8:4006da 0x7ffd8e3b82ec:00 0x7ffd8e3b82e0:01 0x7ffd8e3b82e4:02 0x7ffd8e3b82e8:03 0x7ffd8e3b82ec:04 0x7ffd8e3b82ec:04 0x7ffd8e3b82f0:05 0x7ffd8e3b82f0:05 0x7ffd8e3b82f0:06 0x7ffd8e3b82f2:08 0x7ffd8e3b82f2:08 0x7ffd8e3b8300:09 0x7ffd8e3b8300:09 0x7ffd8e3b8300:00 0x7ffd8e3b8300:00 0x7ffd8e3b8300:00
0064FED0:0a 0064FED1:00 0064FED2:00 0064FED3:00 0064FED4:0b 0064FED5:00 0064FED6:00	0064FED0:0a 0064FED4:0b 0064FED8:0c 0064FEDC:401386 0064FEE0:01 0064FEE4:ca15e0 0064FEE8:ca1918	0x7ffd8e3b8320:0c 0x7ffd8e3b8320:0c 0x7ffd8e3b8322:00 0x7ffd8e3b8322:00 0x7ffd8e3b8323:00 0x7ffd8e3b8324:00 0x7ffd8e3b8325:00 0x7ffd8e3b8325:00	0x7ffd8e3b8318:0a 0x7ffd8e3b8318:0a 0x7ffd8e3b831c:0f 0x7ffd8e3b8320:0c 0x7ffd8e3b8324:00 0x7ffd8e3b8328:77084d85 0x7ffd8e3b832c:7f95 0x7ffd8e3b8330:771cfac8

### 越界写

#### Char:

Vs DEBUG 会中断退 出,结束进程 VS Release 会阻止写 入,不报错提示 Dev Debug 不会写入 Dev Release 不会写入 正常退出 Linux 不会写入

Vs DEBUG 会中断退

#### Int:

出,结束进程 VS Release 正常写入, 不报错提示 Dev Debug 正常写入 Dev Release 写入 正常退出 Linux 写入并正常退出 结论: VS的Debug模式如何判断动态内存访问越界(同C方式)



### 结论:与动态内存申请的相似

在int[]与char[]都与动态内存申请相似,申请一段内存时,或是定义一段数组,该区域前后都要初始化,检查越界时,看越过区域的值与初始值是否相同,不同就是越界了。但不同定义方式不同,如int[]其前后区域的一些信息仿佛是在声明该区域是int型的数组,char[]型数组前后都是相同的值,且都是在p[14]及之后位置,越界就不管了,前面几个位置应该是相关的一些信息。

### 结论: 其他模式如何判断动态内存访问越界



### Vs Release 模式

结论:与动态内存申请的相似,但也有不同的地方

在int[]与char[]都与动态内存申请相似,申请一段内存时,或是定义一段数组,该区域前后都不会初始化,不检查越界,或是说,当在越界区域写时,写不进入.

### Dev Debug/Release模式

Int和Char 写入方式有所不同,Int可以写入,但是Char 不能写入,程序可以正确退出

### Linux 模式 同C方式

Linux 越界int能写入,Char可以正常写入,这点和Dev 相似,因为都是gcc编译器.程序可以正确退出

# §. 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)

```
#define _CRT_SECURE_NO_WARNINGS
∃#include<iostream>
#include<iomanip>
 using namespace std;
∃int main()
     char str[10] = "123456789";
     int num[10] = { 1,2,3,4,5,6,7,8,9,0 };
∃#if 0
     str[14] = 'A'; //越界
     str[15] = 'B';
     str[16] = 'C';
    num[14] = 10;
    num[15] = 11;
     num[16] = 12;
 #endif
     cout << "str:" << hex << (void*)(str);</pre>
     cout << "\t\t";</pre>
     cout << "num:" << hex << (void*)(num);</pre>
     cout << endl:
     for (int i = -5; i \le 20; i++)
         if (i == 14)cout << "-----
         cout << hex << (void*)(str + i) << ":" <<int(str[i]);</pre>
         cout << "\t";
         cout << hex << (void*)(num + i) << ":" << num[i];</pre>
         cout << endl;</pre>
     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 x86Release 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)

### 1、取消越界写的结果

#### Dev 32bit-Release

#### Linux



#### D:\WorkSpace\C projects vsstudio\Gao chen\13-b8-test1\13-b8-test.exe

4	str:0064FEC2	num:0064FE98
111	0064FEBD:00	0064FE84:150
ı	0064FEBE:00	0064FE88:64ffcc
ı	0064FEBF:00	0064FE8C:7542dcd0
ı	0064FEC0:50	0064FE90:12bd431
ı	0064FEC1:16	0064FE94:fffffffe
ı	0064FEC2:31	0064FE98:01
ı	0064FEC3:32	0064FE9C:02
ı	0064FEC4:33	0064FEA0:03
ı	0064FEC5:34	0064FEA4:04
ı	0064FEC6:35	0064FEA8:05
ı	0064FEC7:36	0064FEAC:06
ı	0064FEC8:37	0064FEB0:07
L	0064FEC9:38	0064FEB4:08
ı	0064FECA:39	0064FEB8:09
ı	0064FECB:00	0064FEBC:00
ı	0064FECC:0a	0064FEC0:32311650
ı	0064FECD:00	0064FEC4:36353433
ı	0064FECE:00	0064FEC8:393837
ı	0064FECF:00	0064FECC:0d
ı	0064FED0:ffffffe4	0064FED0:bf15e4
ı	0064FED1:15	0064FED4:45
ı	0064FED2:ffffffbf	0064FED8:64ff68
ı	0064FED3:00	0064FEDC:401386
ı	0064FED4:45	0064FEE0:01
ı	0064FED5:00	0064FEE4:bf15e0
	0064FED6:00	0064FEE8:bf1918

***************************************	
str:0x7ffefe934762	num:0x7ffefe934730
0x7ffefe93475d:00	0x7ffefe93471c:7f77
0x7ffefe93475e:00	0x7ffefe934720:01
0x7ffefe93475f:00	0x7ffefe934724:00
0x7ffefe934760:40	0x7ffefe934728:4006b9
0x7ffefe934761:48	0x7ffefe93472c:00
0x7ffefe934762:31	0x7ffefe934730:01
0x7ffefe934763:32	0x7ffefe934734:02
0x7ffefe934764:33	0x7ffefe934738:03
0x7ffefe934765:34	0x7ffefe93473c:04
0x7ffefe934766:35	0x7ffefe934740:05
0x7ffefe934767:36	0x7ffefe934744:06
0x7ffefe934768:37	0x7ffefe934748:07
0x7ffefe934769:38	0x7ffefe93474c:08
0x7ffefe93476a:39	0x7ffefe934750:09
0x7ffefe93476b:00	0x7ffefe934754:00
0x7ffefe93476c:0a	0x7ffefe934758:4004f0
0x7ffefe93476d:00	0x7ffefe93475c:00
0x7ffefe93476e:00	0x7ffefe934760:32314840
0x7ffefe93476f:00	0x7ffefe934764:36353433
0x7ffefe934770:ffffffd0	0x7ffefe934768:393837
0x7ffefe934771:06	0x7ffefe93476c:0f
0x7ffefe934772:40	0x7ffefe934770:4006d0
0x7ffefe934773:00	0x7ffefe934774:00
0x7ffefe934774:00	0x7ffefe934778:478fdd85
0x7ffefe934775:00	0x7ffefe93477c:7f77
0x7ffefe934776:00	0x7ffefe934780:47a48ac8

### 越界读

读取到的内容和编译 条件有关

#### Str:

Vs Debug 是固定值 Vs Release 是随机值 Dev Debug 是随机值 Dev Release 是随机值 Linux 是 0

#### Int:

Vs Debug 是固定值 Vs Release 是随机值 Dev Debug 是随机值 Dev Release 是随机值 Linux 是 0

### 2、越界写的结果

### VS2022的x86/Debug

D:\WorkSpace\C projects vsstudio\Gao chen\Debug\13-b8-test2.exe

```
str:00F9F9A8
                        num:00F9F978
00F9F9A3:ffffffcc
                        00F9F964:0
                        00F9F968:ccccccc
00F9F9A4:ffffffcc
                        00F9F96C:fffffffd
00F9F9A5:fffffcc
00F9F9A6:ffffffcc
                        00F9F970:ccccccc
                        00F9F974:ccccccc
00F9F9A7:ffffffcc
00F9F9A8:31
               00F9F978:1
00F9F9A9:32
                00F9F97C:2
0F9F9AA:33
               00F9F980:3
00F9F9AB:34
                00F9F984:4
0F9F9AC:35
                00F9F988:5
00F9F9AD:36
                00F9F98C:6
0F9F9AE:37
                00F9F990:7
00F9F9AF:38
                00F9F994:8
               00F9F998:9
00F9F9B0:a
00F9F9B1:0
                00F9F99C:0
00F9F9B2:0
                00F9F9A0:ccccccc
00F9F9B3:0
                00F9F9A4:ccccccc
               00F9F9A8:34333231
00F9F9B4:b
00F9F9B5:0
                00F9F9AC:38373635
00F9F9B6:0
                00F9F9B0:a
0F9F9B7:0
                00F9F9B4:b
00F9F9B8:c
                00F9F9B8:c
0F9F9B9:0
                00F9F9BC:f9f9dc
00F9F9BA:0
                00F9F9C0:1838f3
00F9F9BB:0
                00F9F9C4:1
OF9F9BC:ffffffdc
                        00F9F9C8:157efa0
```

Microsoft Visual C++ Runtime Library



Program:

 $\label{lem:condition} D:\WorkSpace\C_projects\_vsstudio\Gao\_chen\Debug\13-b8-test2.exe\\ Module:$ 

D:\WorkSpace\C\_projects\_vsstudio\Gao\_chen\Debug\13-b8-test2.exe

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

(Press Retry to debug the application)

中止(A) 重试(R)

忽略(I)

#### VS2022 x86/Release

#### ™ Microsoft Visual Studio 调试控制台

	, 2,- 43-	
str:00F3F9BC		num:00F3F994
00F3F9B7:0	00F3F980	:9c21c5
00F3F9B8:0	00F3F984	:4
00F3F9B9:0	00F3F988	:f3f9d0
00F3F9BA:0	00F3F98C	:9c1162
00F3F9BB:0	00F3F990	:13582e0
00F3F9BC:31	00F3F994	:1
00F3F9BD:32	00F3F998	:2
00F3F9BE:33	00F3F99C	:3
00F3F9BF:34	00F3F9A0	:4
00F3F9C0:35	00F3F9A4	:5
00F3F9C1:36	00F3F9A8	:6
00F3F9C2:37	00F3F9AC	: 7
00F3F9C3:38	00F3F9B0	:8
00F3F9C4:39	00F3F9B4	:9
00F3F9C5:0	00F3F9B8	:0
00F3F9C6:5	00F3F9BC	:34333231
00F3F9C7:77	00F3F9C0	:38373635
00F3F9C8:34	00F3F9C4	:77050039
00F3F9C9:17	00F3F9C8	:9c1734
		00F3F9CC:d0a0971b
00F3F9CB:0		
00F3F9CC:1b		
00F3F9CD:ffffff9		
		00F3F9DC:13582e0
00F3F9CF:ffffffd		00F3F9E0:1358c48
00F3F9D0:18	UUF3F9E4	:: duau9497

#### Dev 32bit-Debug



#### D:\WorkSpace\C projects vsstudio\Gao chen\13-b8-test2\13-b8-test2.exe

```
0x77feaf:0
                0x77fe7c:40bd50
0x77feb0:46
                0x77fe80:77fe6c
0x77feb1:0
                0x77fe84:0
0x77feb2:31
                0x77fe88:1
0x77feb3:32
                0x77fe8c:2
0x77feb4:33
                0x77fe90:3
0x77feb5:34
                0x77fe94:4
0x77feb6:35
                0x77fe98:5
0x77feb7:36
                0x77fe9c:6
0x77feb8:37
                0x77fea0:7
0x77feb9:38
                0x77fea4:8
0x77feba:39
                0x77fea8:9
0x77febb:0
                0x77feac:0
0x77febc:a
                0x77feb0:32310046
0x77febd:0
                0x77feb4:36353433
0x77febe:0
               0x77feb8:393837
               0x77febc:d
0x77febf:0
0x77fec0:a
                0x77fec0:a
0x77fec1:0
                0x77fec4:b
0x77fec2:0
                0x77fec8:c
0x77fec3:0
               0x77fecc:401386
               0x77fed0:46
0x77fec4:b
0x77fec5:0
               0x77fed4:dc158c
0x77fec6:0
                0x77fed8:dc15e4
Process exited after 2.655 seconds with return value 3221225477
请按任意键继续.
```

### 2、越界写的结果

#### Dev 32bit-Release

#### Linux

Dev 32bit Release		DIII	Liliux	
■ 洗择 D:\Works	Space\C projects vsstudio\Gao chen\13-b8-test2\13-b8-te	est2 exe [u1850772@101080 ~]\$ g+	+ -Wall -o a 13-b8-test2.	
		[u1850772@101080 ~]\$ ./		
0x77feae:0 0x77feaf:0	0x77fe78:0 0x77fe7c:40bd50	str:0x7ffe44354c12	num:0x7ffe44354b	
0x77feb0:46	0x77fe80:77fe6c	0x7ffe44354c0d:0	0x7ffe44354bcc:0	
x77feb1:0	0x77fe84:0	0x7ffe44354c0e:0	0x7ffe44354bd0:44354c20	
x77feb2:31	0x77fe88:1	0x7ffe44354c0f:0	0x7ffe44354bd4:7ffe	
x77feb3:32	0x77fe8c:2	0x7ffe44354c10:fffffff0		
x77feb4:33	0x77fe90:3	0x7ffe44354c10:77777770	0x7ffe44354bdc:0	
x77feb5:34 x77feb6:35	0x77fe94:4 0x77fe98:5			
x77feb7:36	0x77fe9c:6	0x7ffe44354c12:31	0x7ffe44354be0:1	
x77feb8:37	0x77fea0:7	0x7ffe44354c13:32	0x7ffe44354be4:2	
x77feb9:38	0x77fea4:8	0x7ffe44354c14:33	0x7ffe44354be8:3	
x77feba:39	0x77fea8:9	0x7ffe44354c15:34	0x7ffe44354bec:4	
x77febb:0	0x77feac:0	0x7ffe44354c16:35	0x7ffe44354bf0:5	
x77febc:a x77febd:0	0x77feb0:32310046 0x77feb4:36353433	0x7ffe44354c17:36	0x7ffe44354bf4:6	
x77feba.0	0x77feb8:393837	0x7ffe44354c18:a	0x7ffe44354bf8:7	
x77febf:0	0x77febc:d	0x7ffe44354c19:0	0x7ffe44354bfc:8	
		0x7ffe44354c1a:0	0x7ffe44354c00:9	
x77fec0:a	0x77fec0:a	0x7ffe44354c1b:0	0x7ffe44354c04:0	
x77fec1:0	0x77fec4:b	0x7ffe44354c1c:a	0x7ffe44354c08:400840	
)x77fec2:0 )x77fec3:0	0x77fec8:c 0x77fecc:401386	0x7ffe44354c1d:0	0x7ffe44354c0c:0	
x77fec4:b	0x77fed0:46	0x7ffe44354c1e:0	0x7ffe44354c10:32314cf0	
x77fec5:0	0x77fed4:c7158c	0x7ffe44354c1f:0	0x7ffe44354c14:36353433	
x77fec6:0	0x77fed8:c715e4	0X/FTE44334CIF.0		
		0x7ffe44354c20:c	0x7ffe44354c18:a	
rocess exite	d after 2.25 seconds with return value 322	1225471 <b>0x7ffe44354c21:0</b>	0x7ffe44354c1c:f	
<b>圭拉</b> 红音键	<del>,</del>	0x7ffe44354c22:0	0x7ffe44354c20:c	
		0x7ffe44354c23:0	0x7ffe44354c24:0	
		0x7ffe44354c24:0	0x7ffe44354c28:e7958d85	
		0x7ffe44354c25:0	0x7ffe44354c2c:7f63	
		0x7ffe44354c26:0	0x7ffe44354c30:e833f990	
		0X/TTE44354C26:0	0X/TTE44354C30:E633T990	



### 越界写

#### Char:

Vs DEBUG 会中断退出,结束进程 VS Release 会阻止写入,不报错提示 Dev Debug 不会写入 并exit(错误代码) Dev Release 不会写入 正常退出 Linux 不会写入 Int:

Vs DEBUG 会中断退出,结束进程 VS Release 正常写入, 不报错提示 Dev Debug 正常写入 并exit(错误代码) Dev Release 写入 正常退出 Linux 不能写入段错误

### 结论: VS的Debug模式如何判断动态内存访问越界(同C方式)

### 结论: 与动态内存申请的相似

在int[]与char[]都与动态内存申请相似,申请一段内存时,或是定义一段数组,该区域前后都要初始化,检查越界时,看越过区域的值与初始值是否相同,不同就是越界了。但不同定义方式不同,如int[]其前后区域的一些信息仿佛是在声明该区域是int型的数组,下一页有图片展示,char[]型数组前后都是相同的值,且都是在p[14]及之后位置,越界就不管了,前面几个位置应该是相关的一些信息。

在int[]与char[]都与动态内存申请相似,申请一段内存时,或是定义一段数组,该区域前后都要初始化,检查越界时,看越过区域的值与初始值是否相同,不同就是越界了。但不同定义方式不同,如int[]其前后区域的一些信息仿佛是在声明该区域是int型的数组,下一页有图片展示,char[]型数组前后都是相同的值,且都是在p[14]及之后位置,越界就不管了,前面几个位置应该是相关的一些信息。



### 结论: 其他模式如何判断动态内存访问越界

1 A90 P APP

Vs Release 模式 同 C 方式 结论:与动态内存申请的相似,但也有不 同的地方

在int[]与char[]都与动态内存申请相似,申请一段内存时,或是定义一段数组,该区域前后都不会初始化,不检查越界,或是说,当在越界区域写时,写不进入.

### Dev Debug/Release模式 与C方式不同

都不能非法写入,虽然编译无问题,但是运行时C++会非正常退出,exit不是0,说明其中有判断写入的越界,导致从某个库函数内exit(错误返回码) C可以正常退出

### Linux 模式 与C++方式不同

Linux 越界会产生段错误(核心已转移)而且不能非 法写入

# §. 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

内存越界,就是访问了超去用户自身原来声明需要的内存单元的情况,使用了不合法的内存。它如果只是读的话,影响还是比较小的,如果改变了其中的数据,就会有很多可怕的后果。 1.操作系统/编译器在防范越界时,大概是这样的,如果用户需要一定的内存,在它定义或是动态申请之后,我进行反馈,我申请一段更大的内存,并将用户需要的内存部分,放在中间,对其初始化,设定一个很特殊的初始值。用户使用的时候,大多数情况,都应该是连续使用的,我对其进行检查,如果规定部分的前后,内存部分不为初始值,那用户肯定是使用了非法部分就提示越界。

2.不同的编译器之前对越界的处理有区别,且动态内存申请与数组、数组的类型、不同的语言会产生不同的越界判断处理情况。但总的思想是相同的,即上述的思想。如果在设计编译器的时候,考虑对所定义索引加以判断,可以减少部分越界的情况,但会使编译器更为复杂。3.平时的话,敲代码的时候,考虑清除,需要多大的内存,申请完毕后,在使用这些内存的时候,多考虑会不会有越界的情况。或许,可以考虑多申请或是定义一个或是若干的内存单元用作纠错、提示。就如前段时间敲大作业的时候,那个提示输出色块中间是什么的结构体,最后一个结构体表示结束。