

C 语言中的数组访问越界：产生原因、预防和检测

胡雨晨

2024.12.10

摘要

在 C 语言中，数组访问越界行为是指执行计算机代码时，这种代码在当前程序状态下的行为访问了不属于该数组的内存，这样的行为在 C 语言标准中是未规定的，会造成代码运行错误，严重时可能造成整个程序崩溃。但 IDE 并不会对这样的错误进行报错，再由于当代码极其复杂的时候，这些行为往往极其隐蔽，这就给承担着保证代码准确性的程序员带来了极大的困扰。基于此，本文将探究数组越界导致代码运行错误的原因以及 IDE 为什么不会识别这样的错误，并从代码优化和工具使用两个方面为程序员提出改进方法。

1 引言

在 C 语言中，访问越界 (Out-of-Bounds Access) 是指程序试图访问的内存位置超出了其合法或已分配的范围。数组越界是其常见的一种类型。例如图1，就在对 `arr[5]` 赋值时出现了访问越界问题。

这样的代码尽管具有访问越界问题，但仍然可以在 IDE 中运行，本文会探究其背后的原因，以及如何预防或检测这样的问题。

2 数组越界

在 C 语言中，对于一个声明为 `array[n]` 的数组，它的下标取值为从 0 到 `n-1` 的整数，当下标值超出该范围时，编译器试图访问的内存位置出现错误，就可能造成程序崩溃。

```
1 int arr[5]={0};//初始化
2 arr[0]=3;//正常访问
3 arr[1]=4;//正常访问
4 arr[2]=18;//正常访问
5 arr[3]=9;//正常访问
6 arr[4]=100;//正常访问
7 arr[5]=7;//访问越界
```

图 1: 定义数组

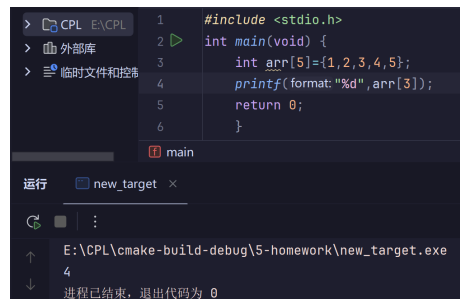
数组越界错误不会被 IDE 报错，但它造成的后果是极为隐蔽并可能造成代码崩溃的。接下来将分别介绍这种行为的错误原因和与为什么会出现在 IDE 的不识别。

2.1 错误原因

我们先来看两段输出，如图2和3，它们分别展示了在正确访问数组和越界访问数组的情况下的输出：

可见，在试图访问 `arr[5]` 并输出它的时候，尽管 IDE 并未报错，但它输出的结果是一个未知内存空间中储存的数据，这往往不是程序员想要的。会出现这样的情况，是因为在代码的第 3 行，对数组进行声明时，只申请了 5×4 个字节大小的内存空间并向其中储存数据。因此，正确地访问 `arr[3]` 所在的内存地址并输出其中的数据是可行的，而访问 `arr[5]` 就是程序试图访问的内存位置超出了其合法或已分配的范围的行为，会造成输出错误。

在更加复杂的代码中，这样的行为甚至会造成程序崩溃。



```
1 #include <stdio.h>
2 int main(void) {
3     int arr[5]={1,2,3,4,5};
4     printf(format: "%d", arr[3]);
5     return 0;
6 }
```

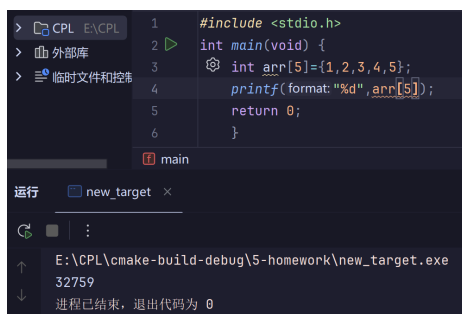
运行 new_target ×

E:\CPL\cmake-build-debug\5-homework\new_target.exe

4

进程已结束，退出代码为 0

图 2: 正确输出



```
1 #include <stdio.h>
2 int main(void) {
3     int arr[5]={1, 2, 3, 4, 5};
4     printf(format:"%d",arr[5]);
5     return 0;
6 }
```

运行 new_target x

E:\CPL\cmake-build-debug\5-homework\new_target.exe
32759
进程已结束，退出代码为 0

图 3: 错误输出

2.2 IDE 的不识别

如上所示的问题是一个非常典型的编程错误，但 IDE 并未对其报错，反而让它“顺利”地通过了编译。接下来本文将从内存访问机制和 C 语言的设计哲学两个角度分析为什么 IDE 不会报错。

内存访问机制 在 C 语言中，数组拥有一串连续的内存空间，指针与数组紧密相关。对数组元素的访问实质上是在对指针进行算术运算。例如 `arr[i]` 等价于索引数组首元素所在地址 `+i` 后所表示的地址所储存的元素。因此，越界访问时计算出的地址虽然超出了分配给数组的内存空间范围，但在硬件层面该地址存在，这仍然是一个可执行的操作。

C 语言设计哲学 首先，C 语言的设计理念中重要的一条即为追求程序运行的高效性。而如果对于每一个数组访问操作都进行边界检查，将会带来极大的性能损耗，这在许多应用场景中都是不可接受的。基于此，为追求高效与灵活，C 语言将对内存的控制权交给了程序员，同时也赋予了他们保证不出现包括数组越界在内的错误（即保证内存使用正确）的义务。

其次，数组的访问越界往往出现在执行时的动态循环中，而 C 语言作为静态编译语言，它的编译器主要负责语法错误与静态语义的检查。对于数组访问越界这种动态错误，它的发生与否取决于程序的动态执行，因此 IDE 不会事先得知是否出错，自然无法进行编译错误的提示。

综上所述，C 语言的特点使其编译器不具有为数组越界行为报错的义务，相应地，这一义务被交给了作为程序员的人类。

3 改进方法

由上文可知，程序员由保证代码正确性的义务。但当代码逻辑复杂的时候，数组越界的错误往往是极其隐蔽的，人类不一定能快速、完整地发现它们。因此，下文将从如何通过代码优化来尽量避免该类错误和工具使用来较为高效地排查错误两方面提出改进方法。

3.1 代码优化

在编写代码时，程序员可以通过优化代码结构来规避数组越界行为。例如，在扫雷问题中，我们需要遍历每一个格子周围的所有格子。为避免检索边界格时的数组越界行为与过于繁琐的分类讨论，我们可以通过扩大一圈数组来实现目的。

以上只是一个非常简单的示例，在更复杂的问题中，我们也可以期望程序员通过优化算法来避免数组越界行为，在此不做讨论。

3.2 工具使用

由于数组越界对程序潜在的巨大危害，人类目前已发明了多种工具对数组越界行为进行检测。

传统地，数组越界的检测方法分为静态检测和动态检测。静态检测方法主要是通过人工分析，模型验证或形式化证明等方法来找出程序中可能存在的问题，如 LCLint, CSSV, BOON, ESC/Java 等，能主动检查所有代码，报告所有可疑越界点。动态检测方法主要是进行程序插桩，即对源代码进行访问和修改，在需要插桩的地方加入插桩代码，重新编译链接，检测程序的运行行为，从程序运行中发现错误。如 Purify, CCured 等都能通过插装函数，在代码中添加对越界行为的运行时检查。

然而，静态检查的误报率与漏报率都较高，并需耗时的人工复查和繁琐的源码标记；动态检查也存在减慢软件运行速度，代码受检率不高等问题。

因此，我们希望使用动静结合的方法来完成检验。新型的动静结合的符号执行方法应运而生。它通过符号化输入代替程序变量，执行时程序变量、语句和表达式会根据符号集模拟执行。符号执行通过探索不同的程序路径，为每个路径生成一组具体输入值来遍历程序的每一种可能，以检查程序中是否有错误。通过这样的模拟运行，就可以在不运行程序的条件下检查是否会出现数组越界。

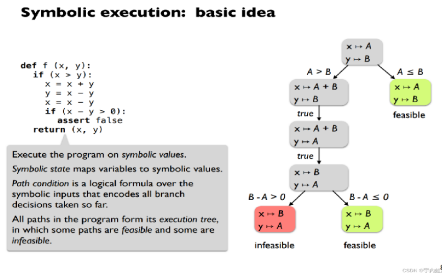


图 4: 符号执行方法举例

4 结论

本文聚焦于未定义行为中的数组越界行为, 探究了其会造成代码运行错误的原因, 并从内存访问机制与 C 语言本身的特点两个角度分析了为什么编译器不会对这种行为进行报错。基于此, 我们对程序员给出代码优化与工具使用这两种改进意见。前者聚焦于在代码编写的过程中, 希望程序员使用更加“安全”的代码或更加“高明”的算法来规避数组越界情况。而后者则聚焦于对代码的检查, 期望综合各有利弊的动态检查与静态检查方法, 以符号执行方法来帮助程序员检查复杂代码中的数组越界行为, 从而进行修改。

由于篇幅限制, 本文并未对算法优化与符号执行方法的具体原理展开讨论, 但它们实际上是非常有价值的命题, 希望有兴趣的读者可以自行探究。

参考文献

- [1]Wang J J.Jiang F.Zhang T.Detection method for memory overrun in multi-loop programs[J].Journal of the Graduate School of the Chinese Academy of Sciences,2010 27 (1): 117-126.
- [2]LI Wenming, CHEN Zhe, LI Xurong, et al. Runtime verification of array bounds overflow of C programs. Computer Engineering and Applications, 2015, 51 (11): 190-195.
- [3]Review of Symbolic Execution Technology and ApplicationsWU Hao1, ZHOU Shilong2, SHI Donghui1, LI Qiang21.School of Electronic and Information Engineering, Anhui Jianzhu University, Hefei 230601, China2.School of Electronic Countermeasures, National University of Defense Technology,

Hefei 230037, China