

前言：

上一次被班主任查水表的时候，班主任提到了可以写一份 C/C++ 编程时常见或新奇或棘手的错误，分享给大家。响应班主任的号召，于是就有了这个神奇的文档。

文档会不定期更新，更新频率与大家发掘新奇错误的频率成正比（确信）。

如果您遇到了什么新奇的错误，欢迎分享！

1. 请检查你有没有写什么奇怪（？）的内容：

比如：

是不是很喜欢 steam？

```
#include <iosteam>
```

```
#include <fsteam>
```

是不是喜欢吃面？

```
int mian()  
{  
    cout<<"烫烫烫";  
    return 0;  
}
```

手有时候也会抽筋？

```
2 using naemspace std;
```

```
7 retrun 0;
```

2. 有时候退格修改代码内容的时候，会误伤到大括号，于是：

```
int main()  
{  
    int cnt=0;  
    for(int i=0;i<10;++i)  
    {  
        cnt+=i;  
        cout<<cnt;  
    }  
    return 0;  
}
```

```
[Error] expected unqualified-id before 'return'
```

```
[Error] expected declaration before '}' token
```

有的时候报错会很迷，这取决于你哪里少了括号。

3. 变量未定义:

```
int main()
{
    for(int i=0;i<10;++i)
    {
        cnt+=i;
        cout<<cnt;
    }
    return 0;
}
```

In function 'int main()':

[Error] 'cnt' was not declared in this scope

4. 请给变量赋对应的值（这是一个真实的案例）:

```
1 #include <iostream>
2 #include <cstring>
3 using namespace std;
4 int main()
5 {
6     string str;
7     str=0;
8     return 0;
9 }
```

In function 'int main()':

[Error] ambiguous overload for 'operators=' (operand types are 'std::string [aka std::basic_string<char>]' and 'int')

[Note] candidates are:

In file included from Dev-Cpp/MinGW64/lib/gcc/x86_64-w64-mingw32/4.9.2/include/c++/string

from Dev-Cpp/MinGW64/lib/gcc/x86_64-w64-mingw32/4.9.2/include/c++/bits/locale_classes.h

from Dev-Cpp/MinGW64/lib/gcc/x86_64-w64-mingw32/4.9.2/include/c++/bits/ios_base.h

from Dev-Cpp/MinGW64/lib/gcc/x86_64-w64-mingw32/4.9.2/include/c++/ios

from Dev-Cpp/MinGW64/lib/gcc/x86_64-w64-mingw32/4.9.2/include/c++/ostream

from Dev-Cpp/MinGW64/lib/gcc/x86_64-w64-mingw32/4.9.2/include/c++/iostream

from t.cpp

[Note] std::basic_string<_CharT, _Traits, _Alloc>::operator=(const std::basic_string<_CharT, _Traits, _Alloc>&) [with _CharT = char, _Traits = std::char_traits<char>, _Alloc = std::allocator<char>]

[Note] std::basic_string<_CharT, _Traits, _Alloc>::operator=(const _CharT*) [with _CharT = char, _Traits = std::char_traits<char>, _Alloc = std::allocator<char>]

[Note] std::basic_string<_CharT, _Traits, _Alloc>::operator=(_CharT) [with _CharT = char, _Traits = std::char_traits<char>, _Alloc = std::allocator<char>]

[Note] std::basic_string<_CharT, _Traits, _Alloc>::operator=(std::basic_string<_CharT, _Traits, _Alloc>&&) [with _CharT = char, _Traits = std::char_traits<char>, _Alloc = std::allocator<char>]

```
string str=0;
```

```
return 0;
```

terminate called after throwing an instance of 'std::logic_error'
what(): basic_string::_S_construct null not valid

Process exited after 2.195 seconds with return value 3
请按任意键继续. . .

初始化时赋值不对，有可能会收到惊喜。

5. 有一些库函数调用之前别忘了加上头文件：

```
1  #include <iostream>
2  using namespace std;
3  int main()
4  {
5      char a[10];
6      memset(a,0,sizeof(a));
7      return 0;
8  }
```

In function 'int main()':

[Error] 'memset' was not declared in this scope

memset()函数在 string.h 和 cstring 中，请务必带上#include <string.h>或#include <cstring>

6. char 数组用于存储字符串时，请务必在尾巴上留足空间，并且赋值为'\0'，否则在输出字符串时容易出现溢出错误：

```
Process exited after 3.98 seconds with return value 3221225725
```

程序很长一段时间都没有反应，最后还给你一个 SIGSEGV 段错误返回值，这就是溢出的下场。

7. const 类型的变量必须要赋初值，有了初值之后就不能再被更改：

```
4  int main()
5  {
6      const int i;
7
8      return 0;
9  }
```

In function 'int main()':

[Error] uninitialized const 'i' [-fpermissive]

（你敢不赋初值？）

```
4  int main()
5  {
6      const int i=1;
7      i=0;
8      return 0;
9  }
```

In function 'int main()':

[Error] assignment of read-only variable 'i'

（你敢乱改？）

8. malloc 之后得到的空间一定要 free 掉，有一个 malloc 就得有一个 free 对应，否则会造成内存泄漏，C++中 new 和 delete 操作同理：

```
3 struct list
4 {
5     int elem;
6     list* next;
7 };
8 int main()
9 {
10     list* head=new list;
11     head->next=new list;
12     delete head;
13     return 0;
14 }
```

这个例子里我们看到，head 被分配了一个 list 的空间，而 head->next 也被分配了一个 list 的空间，那么直接 delete head，head->next 指向的那个空间是不是也会被一起清空？

答案是不会，这个空间是固定被分配的，delete head 时，只是存有 head->next 值的空间被释放了，而原来 head->next 指向的空间仍然在内存中，这种操作将导致内存泄露。

所以有一个 new（或 malloc），就得有一个 delete（或 free）。

9. &是什么运算符？按位与？引用？取址？

&运算符的作用要看具体的地方，这个运算符具有二义性（三义性？），所以容易被搞混。

```
3 int main()
4 {
5     int a=1,b=2;
6     cout<<(a & b);
7     return 0;
8 }
```

在&两侧有 int 相关的变量时，&的作用是位运算符与。

```
3 void this_is_a_function(string &str)
4 {
5     /*statements*/
6     return;
7 }
```

引用存在于声明函数时的参数表中，只有在声明函数传参的时候，&才是引用。

被引用的变量在该函数中若被更改了值，那么原来被引用进来的原变量的值也会改变。

```

3 void this_is_a_function(int* n)
4 {
5     /*statements*/
6     return;
7 }
8 int main()
9 {
10    int a=1;
11    this_is_a_function(&a);
12    cout<<&a;
13    return 0;
14 }

```

```

this_is_a_fun
cout<<&a; 0x70felc

```

在给参数表里有指针传参的函数传入参数时，有可能会使用到取址符&，在调用函数，获取变量地址时，&单独连接一个变量名，这时候&是取址符，获得变量所在的内存地址。（图中调用函数时，传入了a的地址，第二句cout输出的是a的地址）
三者一定要区分开，不能搞混！

10. 如果你写的是C语言程序：

```

3 void this_is_a_function(int n,int a,int b);
4 void this_is_a_function(int n)
5 {
6     /*statements*/
7     return;
8 }

```

提前声明的函数如果参数数量和之后定义的函数参数数量不一致，是会报错的哦！
（C++中这就重载变成两个函数了，也要注意）

11. 在C++中：

```

3 class A
4 {
5     public:
6         void print(int m);
7 };
8 void A::print()
9 {
10    return;
11 }

```

```

[Error] prototype for 'void A::print()' does not match any in class 'A'
[Error] candidate is: void A::print(int)

```

如果你在外写A类的成员函数，请把参数数量和类型格式一一对应，否则报错，一般格式如上图。

12. 调用函数时，参数数量和类型一定要对应哦！

```
3 void print(int a,int b,int c)
4 {
5     return;
6 }
7 int main()
8 {
9     print(1);
10    return 0;
11 }
12
```

In function 'int main()':

[Error] too few arguments to function 'void print(int, int, int)'

[Note] declared here

参数过少的报错类似于上图所示。

```
3 void print(int a,int b,int c)
4 {
5     return;
6 }
7 int main()
8 {
9     print(1,2,3,4);
10    void print (int a, int b, int c)
11 }
12
```

In function 'int main()':

[Error] too many arguments to function 'void print(int, int, int)'

[Note] declared here

参数过多的报错类似于上图所示。

```
3 void print(int a,int b,int c)
4 {
5     return;
6 }
7 int main()
8 {
9     print(1,2,"hello");
10    return 0;
11 }
12
```

In function 'int main()':

[Error] invalid conversion from 'const char*' to 'int' [-fpermissive]

参数格式不匹配的报错类似于上图所示。

13. 请使用在这一句语句之前已经被声明的内容：

```
3 void f1()  
4 {  
5     f2();  
6     return;  
7 }  
8 void f2()  
9 {  
10     return;  
11 }
```

In function 'void f1()':

[Error] 'f2' was not declared in this scope

显示 f2 未被声明。

此时只需要在前面加上：

```
4 void f2();  
5  
6 void f1()  
7 {  
8     f2();  
9     return;  
10 }  
11 void f2()  
12 {  
13     return;  
14 }
```

或者写成：

```
4 void f2()  
5 {  
6     return;  
7 }  
8 void f1()  
9 {  
10     f2();  
11     return;  
12 }
```

就行了。

同理，在多文件联编的时候，也要注意声明顺序问题。

```
#ifndef __XXX_HPP__  
#define __XXX_HPP__  
  
#include "a.h"  
#include "b.h"  
#include "c.h"  
  
#endif
```

如上图，如果是这样的写法，则声明顺序是按照 a.h 里的内容先声明，然后是 b.h，接着是 c.h 这样来的。

14. 这个应该不用说了，a 的值不会被改的：

```
4 void f1(int n)
5 {
6     n=1;
7     return;
8 }
9 int main()
10 {
11     int a=0;
12     f1(a);
13     return 0;
14 }
```

f1 中的 n 是个局部变量，离开 f1 之后 n 就被销毁了。

15. 如果您的 if、else if、else、for、while、do while 下语句超过了一条，请使用大括号：

```
5 int main()
6 {
7     for(int i=0;i<10;++i)
8     {
9         int m=i;
10        cout<<m;
11    }
12    return 0;
13 }
```

（会报错 m 未定义哦）

16. 请不要在如何一个局部区域直接声明太大的空间，否则会出现堆栈溢出导致程序崩溃：

```
4 int main()
5 {
6     int c[1073741824];
7     return 0;
8 }
9
```

Process exited after 2.82 seconds with return value 3221225725
请按任意键继续. . .

这是某位同学在参加 csp 时遇到的悲惨事件。

17. 注意不要做出格的事情（???）：

```
5 int main()
6 {
7     char buf;
8     scanf("%s",&buf);
9     return 0;
10 }
```

不安全的读取操作会导致溢出甚至程序崩溃（或许还会导致更加严重的后果？）

18. 浅拷贝可能会导致一些问题:

```
class A
{
    public:
};
int main()
{
    A a;
    A b;
    a=b;
    return 0;
}
```

图里的当然不会出事,但是如果您的类中存在指针,并且指针已经被分配空间,请重载运算符=,写好拷贝构造函数,给另外一个类按照需求重新分配新空间,否则有可能导致一个类已经被析构之后且空间全释放之后,另一个类访问已经被释放的空间,这直接会引起程序崩溃。

19. 划重点: 导致程序崩溃的罪魁祸首:

SIGSEGV 段错误/无效内存引用, SIGTRAP 断点处停止运行, SIGFPE 运算异常错误。

导致系统返回 SIGSEGV 并且程序崩溃的情况有以下几种:

(1) 数组越界/连续空间访问越界(很少会引起崩溃,但是很危险,有时候人品不好就挂了)

```
5  int main()
6  {
7      char* c;
8      c=new char[100];
9      c[104]='a';
10     return 0;
11 }
```

(2) 已经释放的空间又被释放一遍,会导致 SIGTRAP:

```
5  int main()
6  {
7      char* c;
8      c=new char[10024];
9      delete []c;
10     delete []c;
11     return 0;
12 }
```


(5) 有时候，程序崩溃真的就是，你没有注意到指针赋错了值，或者指针被悬空了：

```
3 struct list
4 {
5     int elem;
6     list* next;
7 };
8 void visit(list* L)
9 {
10     while(L->next)
11     {
12         L=L->next;
13         cout<<L->elem;
14     }
15     return;
16 }
17 int main()
18 {
19     list* head;
20     head=new list;
21     visit(head);
22     return 0;
23 }
```

是不是忘了 head->next=NULL; ? ? ?

```
using namespace std;
struct list
{
    int elem;
    list* next;
};
void visit(list* L)
{
    while(L->next)
    {
        L=L->next;
        cout<<L->elem;
    }
    return;
}
int main()
{
    list* head;
    head=new list;
    visit(head);
    return 0;
}
```

一个悲伤的故事。（大多数情况下，这种不会出现 SIGSEGV，只会让你的输出更加“丰富多彩”）

某些函数的传出数值一般不能是引用，因为引用有可能会被回收的变量被引用，导致 SIGSEGV。但是有的情况下（尤其是面向对象编程时），传出数值不是对象成员的引用而是一

一个新变量，之后在对这个变量进行引用的时候，也会引用到被回收的内存，导致 SIGSEGV。

(6) 一定要注意边界条件，因为一不小心，你就内存越界了：
(一个鲜活的例子) 在某链表中：

```
while(ptr)
{
    ptr=ptr->next;
    if(ptr->data!=1024)
        cout<<ptr->data<<" ";
}
```

这个地方就存在越界风险，如果在 `ptr=ptr->next` 时，`ptr->next` 是 `NULL`，那么 `ptr` 就指向 `NULL`，这时再调用 `ptr->data`，就出现了内存访问异常的问题，会导致程序崩溃。

(7) C++中不对应的内存释放会导致内存泄露或者崩溃：

```
5 int main()
6 {
7     char* c;
8     c=new char[10024];
9     delete c;
10    return 0;
11 }
```


申请连续空间，却只释放单个空间，导致无法挽回的内存泄露。

```
5 int main()
6 {
7     char* c;
8     c=new char;
9     delete []c;
10    return 0;
11 }
```

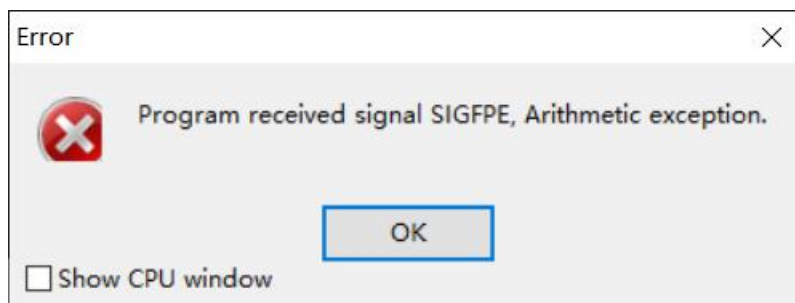
一般不会出问题，但是谁都有人品极差的时候……

(8) 如果结构体指针的值指向不正确的地址，那么取成员的时候也会出问题：

```
1 #include <iostream>
2
3 struct stu
4 {
5     int elem;
6 };
7
8 int main()
9 {
10     stu* a;
11     std::cout<<a->elem;
12     return 0;
13 }
```



(9) int 类型数据除以 0，对 0 取余或者浮点数取模运算，会导致：

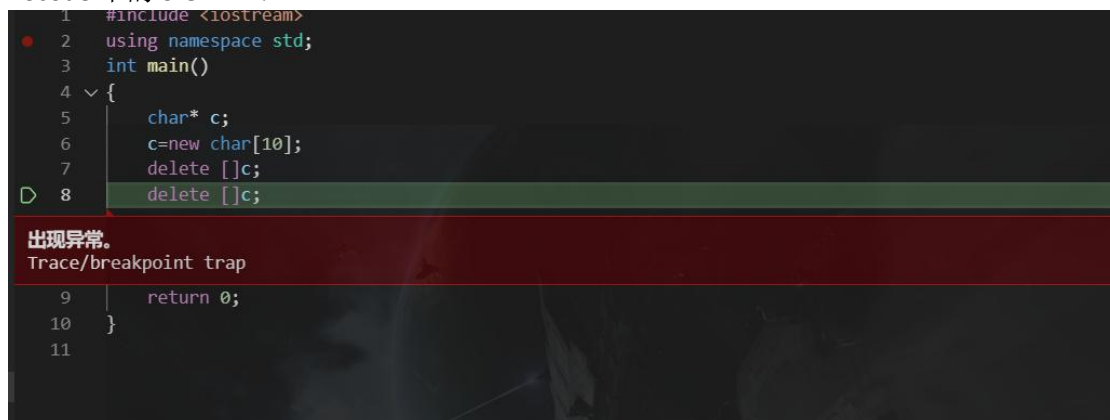


下面是 Dev-cpp 和 VScode 对 SIGSEGV、SIGTRAP、SIGFPE 的报错显示：

Dev-cpp 里，控制台返回值会告诉你是否出现了程序崩溃的问题（当然，程序突然没有输出的时候，可能事情已经糟了）：

3221226356	SIGTRAP	3221225477	SIGSEGV	3221225620	SIGFPE
------------	---------	------------	---------	------------	--------

VScode 中的 SIGTRAP:



VScode 中的 SIGSEGV:



VScode 中对 SIGFPE 的报错:

```
4  int main()
5  {
6      int a=1;
7      cout<<a%0;
8      cout<<a/0;
9      return 0;
10 }
```

出现异常。
Arithmetic exception

20. 释放空间不等于把指针赋值为 NULL:

```
4  int main()
5  {
6      int* a;
7      a=new int[10];
8      a=NULL;
9      return 0;
10 }
```

这样只是更改了 a 指针的值，您申请的空间仍然没有被释放，一直占用内存，造成内存泄露。