# Cppchaeck分析报告

1120132043 于伟平

### 特点简介

#### 功能介绍

Cpppcheck主要检测C语言或c++编写的程序中编译器不能发现的缺陷，是更严格的编译器。Cppcheck能发现很多错误，但不能发现所有错误。

#### 规则定义

1. error：出现的错误
2. warning：为了预防bug防御性编程建议信息
3. style：编码格式问题（没有使用的函数、多余的代码等）
4. portablity：移植性警告。该部分如果移植到其他平台上，可能出现兼容性问题
5. performance：建议优化该部分代码的性能
6. information：一些有趣的信息，可以忽略不看的。

#### 详细扫描规则

|  |  |
| --- | --- |
| 规则 | 描述 |
| arrayIndexOutOfBounds | 数组下标越界 |
| arrayIndexThenCheck | 数组越界访问 |
| bufferAccessOutOfBounds | 缓冲区访问越界 |
| CastIntegerToAddressAtReturn | 把整数返回地址 |
| catchExceptionByValue | 函数返回的整数指针类型是不可移植的 |
| charArrayIndex | 类型用作数组索引，如果该值可以超过127将会有一个缓冲区下溢 |
| clarifyCalculation | 可疑的计算；不明确计算的优先级 |
| clarifyCondition | 可疑的条件 |
| clarifyStatement | 可疑的声明 |
| class\_X\_Y | 信息处理，You can use -I or --include to add handling of this code. |
| compareBoolExpressionWithInt | 布尔表达式与整形比较 |
| comparisonError | 比较错误 |
| comparisonOfBoolWithInt | 布尔表达式与整形比较 |
| comparisonOfFuncReturningBoolError | 比较函数返回布尔值的错误 |
| ConfigurationNotChecked | 配置不检查 |
| constStatement | 常量的声明 |
| copyCtorPointerCopying | 指针指向分配的内存,复制构造函数,而不是分配新的内存。 |
| cstyleCast | c语言风格的指针铸造 |
| deallocDealloc | 回收一个分配指针 |
| deallocuse | dealloc函数的使用 |
| duplicateBreak | 重复 |
| duplicateExpression | 连续的 return, break, continue, goto or throw statements 是没有必要的.第二个语句无法执行,因此应该被删除。 |
| duplInheritedMember | 成员变量的名称与父类的相同 |
| eraseDereference | 迭代器中，无效后元素指向被删除。非关联化或比较它与另一个迭代器是无效的操作。 |
| exceptRethrowCopy | 重新抛出了异常,没有不必要的复制 |
| exceptThrowInDestructor | 析构函数中抛出异常。 |
| fflushOnInputStream | fflus调用输入流，结果是未定义的 |
| incorrectLogicOperator | 不正确的逻辑操作符 |
| incorrectStringBooleanError | 不正确的字符串逻辑 |
| incorrectStringCompare | 不正确的字符串比较 |
| insecureCmdLineArgs | 长命令行参数可能使缓冲区溢出。 |
| invalidPointerCast | 不兼容的二进制表示 |
| invalidPrintfArgType\_int | 无效的整形输出格式 |
| invalidPrintfArgType\_s | 无效的字符串输出格式 |
| invalidPrintfArgType\_sint | 无效的sint输出格式 |
| invalidPrintfArgType\_uint | 无效的uint输出格式 |
| invalidscanf | 字段没有字符长度限制 |
| IOWithoutPositioning | 读和写操作没有调用Positioning函数 |
| memleak | 内存泄露 |
| memleakOnRealloc | 常见的Realloc错误：调用失败的空值没有释放 |
| memsetClass | 使用memset类包含一个虚拟方法是不安全的,因为构造函数、析构函数和拷贝操作符调用omitted。这些都是必要的对于这个non-POD确保创建一个有效的对象类型。 |
| memsetValueOutOfRange | memset()参数并不适合一个无符号字符 |
| memsetZeroBytes | memset()调用0字节的tmp |
| mismatchAllocDealloc | 分配和回收失败 |
| multiCondition | 表达式总是错误的 |
| noConstructor | 没有构造函数 |
| noCopyConstructor | 没有拷贝构造函数 |
| nullPointer | 空指针 |
| obsoleteFunctionsasctime | 过时的函数asctime调用。推荐使用strftime函数。 |
| obsoleteFunctionsgets | 过时的函数被调用。推荐使用fgets函数。 |
| operatorEq | 操作符不符合C++的标准 |
| operatorEqToSelf | 使用运算符“=”检查自己的作业时应该避免动态内存的问题。 |
| operatorEqVarError | 错误的使用=赋值成员变量 |
| oppositeInnerCondition | 相反的内在条件 |
| passedByValue | 在c++中，函数参数strDir应该以引用的方式传递。 |
| pointerLessThanZero | 指针小于零是无意义的 |
| postfixOperator | 前缀+ + / -基本类型操作是低效率的 |
| publicAllocationError | 在公共函数可能泄漏。缓冲区的指针不收回之前分配。 |
| redundantAssignInSwitch | Switch中冗余分配值 |
| redundantAssignment | 冗余分配：变量是重新分配一个被使用旧的值。 |
| redundantCopy | 冗余拷贝：缓冲路径写入之前旧的内容已经被使用。 |
| redundantIfRemove | 冗余检查然后再删除它 |
| resourceLeak | 资源泄露 |
| returnLocalVariable | 返回局部变量 |
| selfAssignment | 自动赋值 |
| sizeofCalculation | 内部计算sizeof函数 |
| sizeofwithnumericparameter | 可疑的使用sizeof数字常数作为参数。 |
| sprintfOverlappingData | 重叠数据 |
| stlcstrParam | c\_str()的结果传递给一个函数,采用string作为参数。是缓慢和冗余。 |
| stlcstrReturn | 返回的结果c\_str()函数,它返回string是缓慢和冗余。 |
| stlOutOfBounds | 字符串越界 |
| stlSize | 字符串的长度限制 |
| strncatUsage | strncat附加在max三参数数量的字符。安全的方式使用strncat是计算剩余空间缓冲区和使用它作为第三个参数。 |
| strPlusChar | 不寻常的指针算术。一个char类型的值添加到字符串文字。 |
| syntaxError | 语法错误 |
| thisSubtraction | 可疑的指针减法。 |
| unassignedVariable | 未赋值的变量 |
| uninitMemberVar | 在构筑函数中没有赋值成员变量 |
| uninitStructMember | 未初始化结构体成员 |
| uninitvar | 未初始化变量 |
| unnecessaryQualification | 是不必要的或是由许多编译器造成的错误 |
| unreachableCode | 执行不到的代码 |
| unreadVariable | 未读的变量 |
| unsignedLessThanZero | 无符号变量永远不会是负的，所以它是无意义的检查 |
| unusedAllocatedMemory | 未使用分配的内存 |
| unusedPrivateFunction | 未使用的私有函数 |
| unusedStructMember | 未使用结构体成员 |
| unusedVariable | 未使用的变量 |
| useAutoPointerArray | 数组操作使用自由的指针 |
| useInitializationList | 构造函数体内的分配变量。考虑初始化列表中进行初始化。 |
| uselessAssignmentPtrArg | 无用的参数 |
| variableScope | 变量值的范围太大 |
| wrongPrintfScanfArgNum | 函数缺少参数 |

#### 缺陷

1. 不支持指针传递
2. 识别缓冲区精度不高
3. 内存操作函数检查的覆盖面不够及一些内存空间操作检查的不好

### 六类错误定义解析及实例

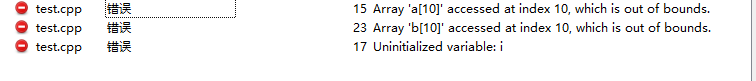
以下是我测试了大量代码后得到的六类等级的漏洞的部分错误原因

#### 错误

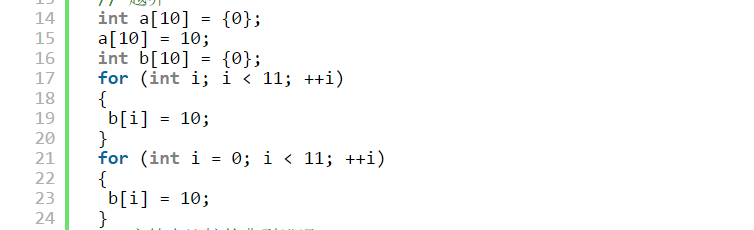
1. 内存泄漏:如果一个内存已被分配，使用后未释放内存，在其他路径可能会在没有释放该内存的情况下直接重新分配该内存，导致内存泄漏。
2. 数组访问越界：程序中存在可能会使数组下标越界的变量
3. 未初始化变量
4. 可能引用了空指针
5. 指针清零后未释放

以上错误较为严重，均可能导致程序崩溃

实例：



测试代码片段：



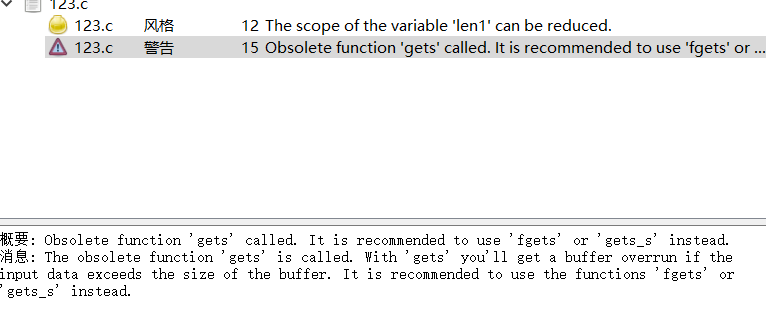
15行a[]数组最大下标为9，a[10]导致数组越界。23行同理

17行变量i没有初始化。以上错误皆可能导致系统崩溃。

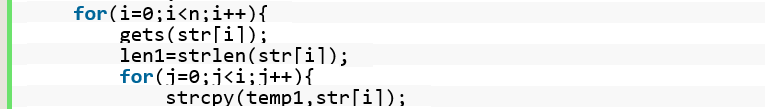
#### 警告

1. 使用了过时或不符合规范的的函数。
2. 不同类型之间赋值时没有使用转换。
3. 多余的条件判断

实例：



测试代码片段：

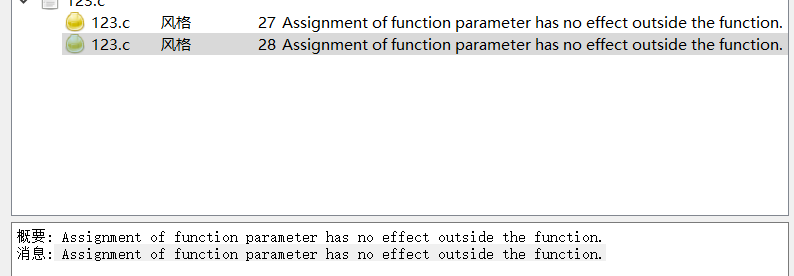


使用了过时的函数gets（），建议改为fgets或gets\_s函数

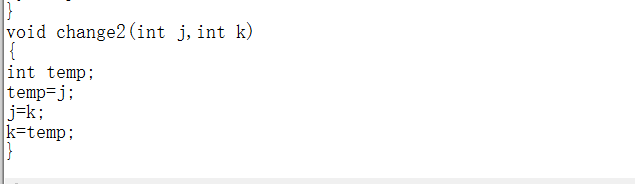
#### 风格

1. 定义的变量未使用
2. 变量的作用域可以缩小
3. 使用了不必要的break、continue、goto、throw语句
4. 使用了多余的else或if语句
5. 函数内参数的赋值在函数外无影响

实例：



测试代码片段：

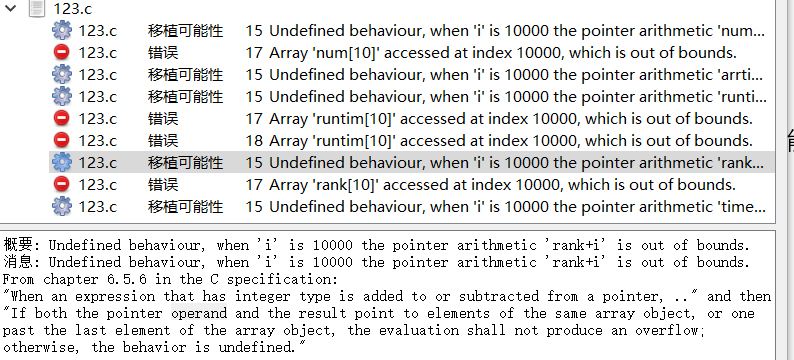


函数change2（）中j，k，temp的赋值无法对函数之外的内容造成影响，没有用处。

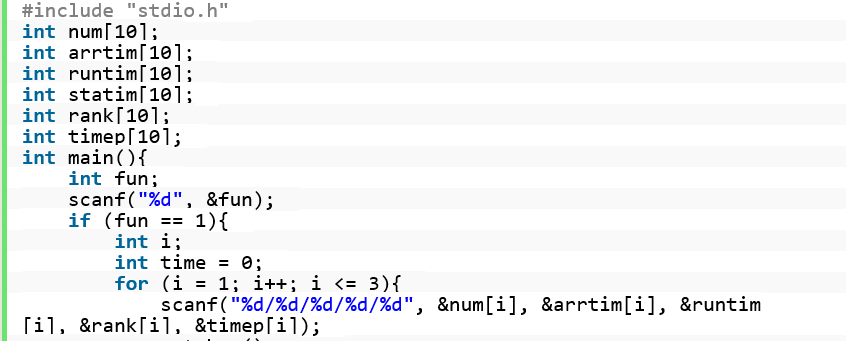
#### 移植可能行

程序移植到其他编译器上运行时可能会出现问题

实例1：

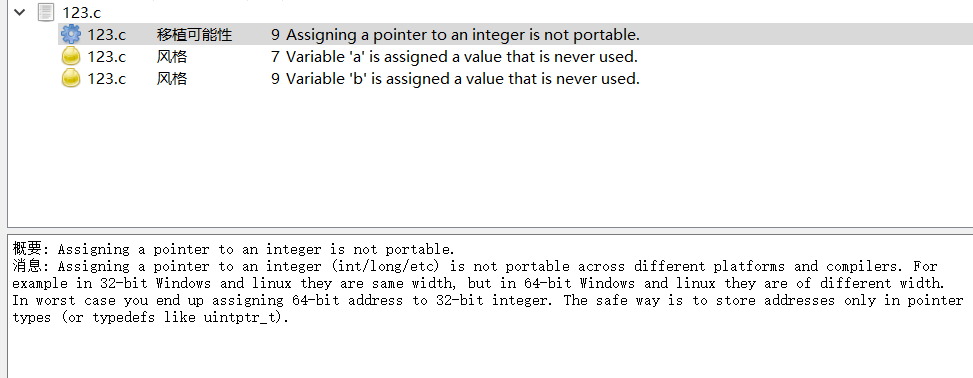


测试代码片段

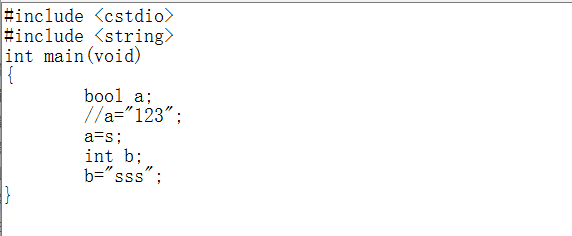


当fun=1时，定义的num、rank等数组不会越界，当fun不等于1时，行为未定义，不同编译器可能会出现不同的结果，可能出现移植可能性问题

实例2：



测试代码：

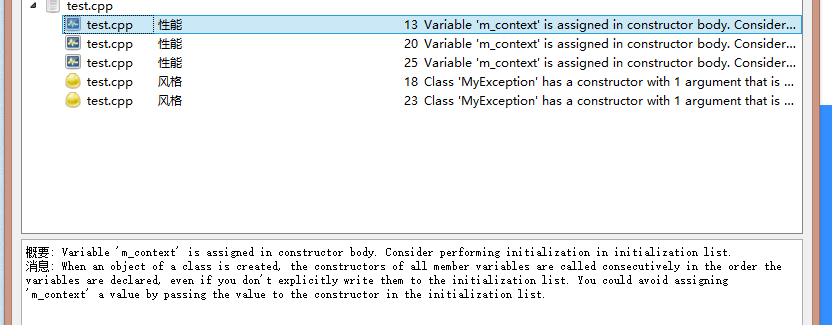


Int、long等数据类型在不同编译器下不同的长度引起的可移植性问题

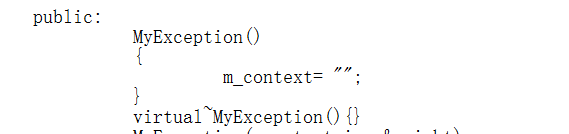
#### 性能

建议优化该部分代码的性能

实例：



测试代码片段：

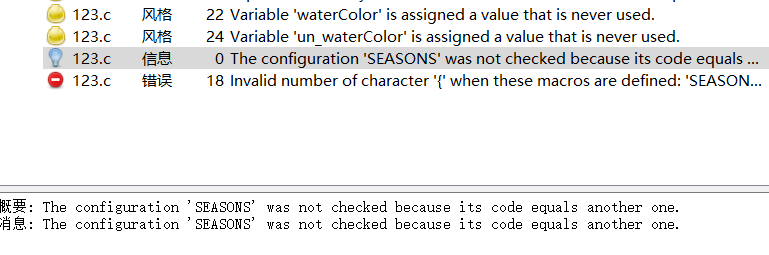


变量m\_context在一个构造函数中初始化，推荐在初始化列表中初始化，可以提高性能

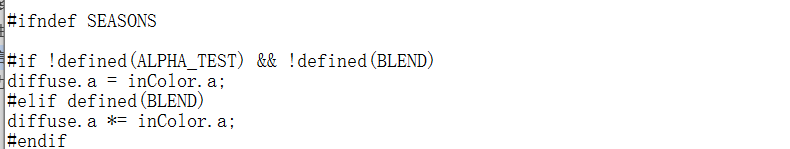
#### 信息

一般是可忽略信息，比如两个结构产生了相同的预处理结果，cppcheck只处理其中一个。

实例：



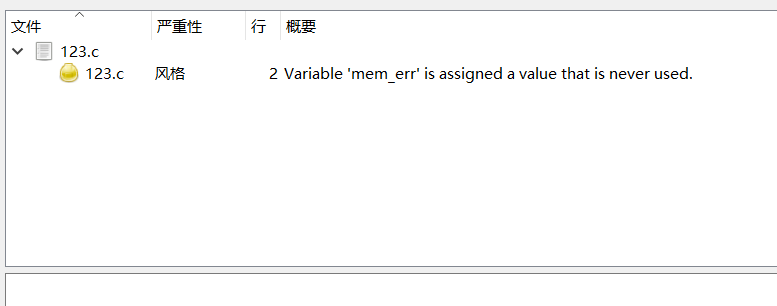
测试代码片段：



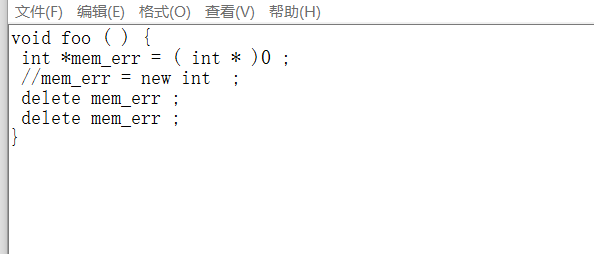
宏SEASONS在之前已经定义过，且产生与此次定义产生的结果相同，故cppcheck不处理此次定义

### 误报漏报实例

#### 实例1

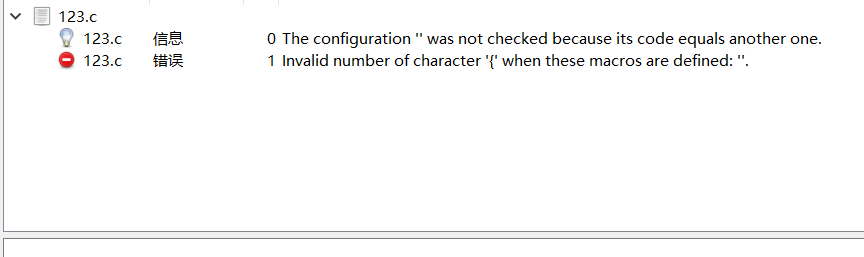


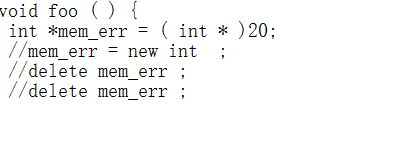
测试代码



尝试多次释放统一内存，属于漏报

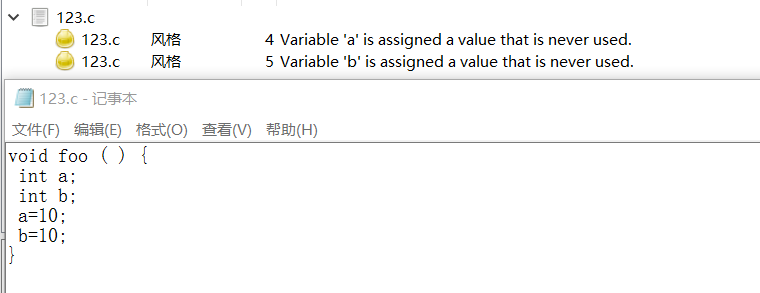
#### 实例2

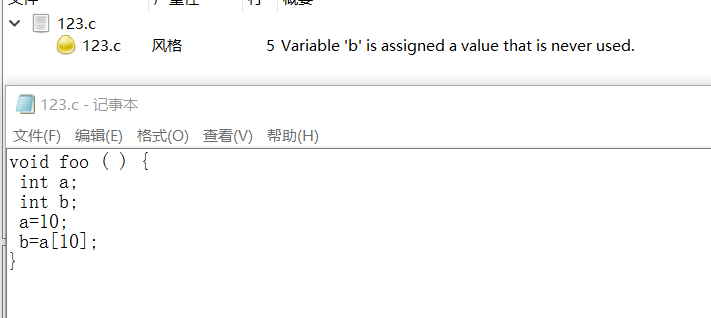




只有前括号出现时会有信息提示，但并不显示变量内容，估计为严重语法错误引起的误报

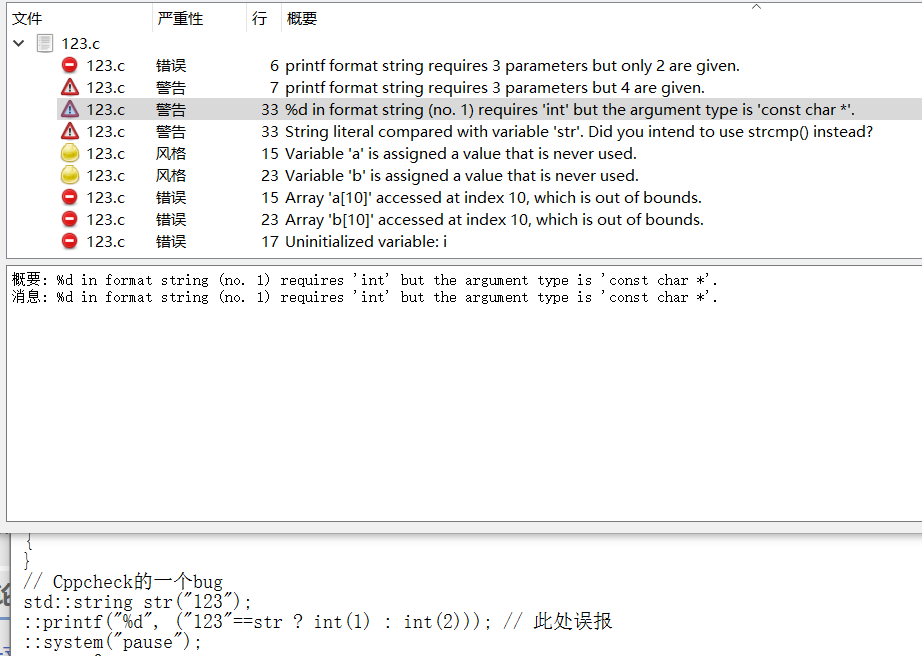
#### 实例3





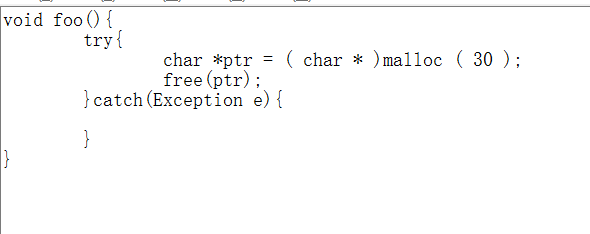
将数组a误认为int a，属于误报

#### 实例4



将简单表达式误认为是静态字符类型指针，属于误报

#### 实例5



异常时未释放ptr，未检测出内存泄露，属于漏报