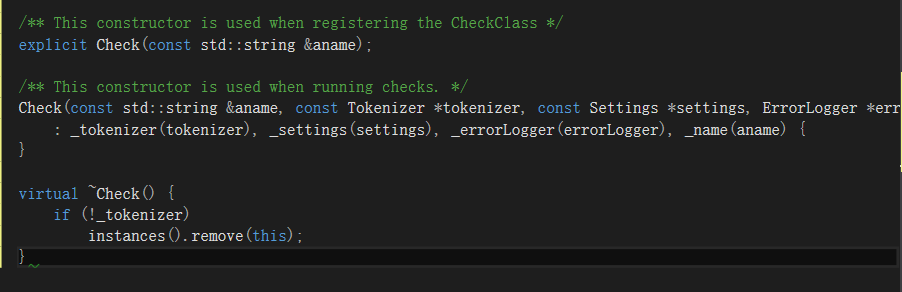
# 在这里，我们主要对几个核心类进行了分析，研究了这几个类的结构、作用及其实现的功能。

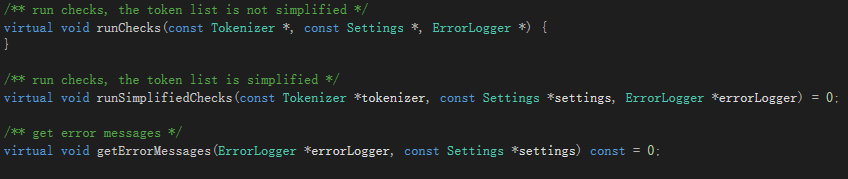
# Check类：

这是一个基类，然后在类中定义了很多函数或方法，在接下来的很多功能模块类都继承类这个基类：

关于这个类，首先是显示的构造函数和析构函数：



然后接下来定义了一些方法来执行代码检查



这3个方法分别的作用是：

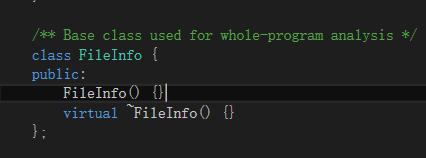
1. runChecks() 执行代码的检查
2. runSimplifiedChecks() 对简化之后的代码开始执行检查

c.getErrorMessages() 这是获取报告的错误信息

然后还定义了一个报告错误信息的静态方法

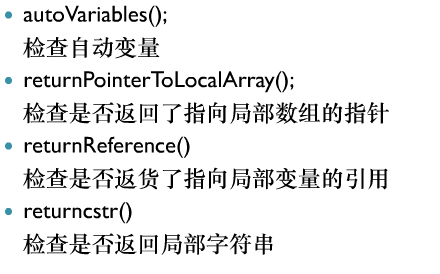


另外在Check类中定义了一个FileInfo类，用来存储文件代码的各种信息，以便于检查代码中出现的错误。

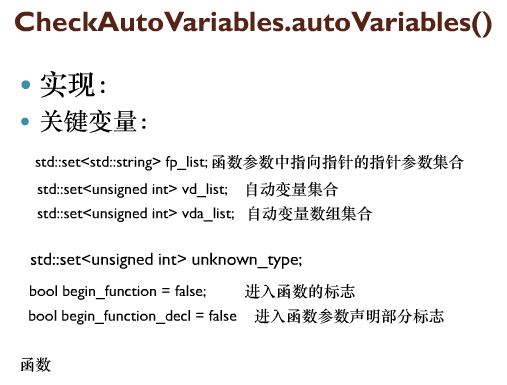


CheckAutoVariables类

这个类也是Check的子类，主要是为了实现对自动变量的检查，类中主要定义了下面几个方法。



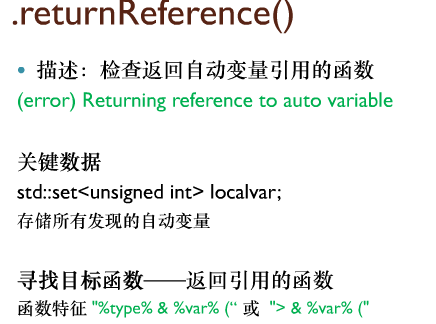
Auto Variables()的实现：



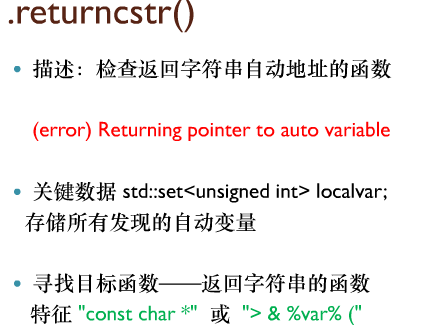
检查是否返回了局部的指针：



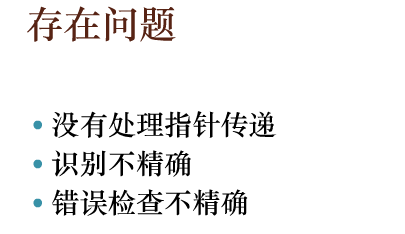
返回自动变量引用的函数：



检查返回字符串自动地址：



但是这个类还存在一些问题。

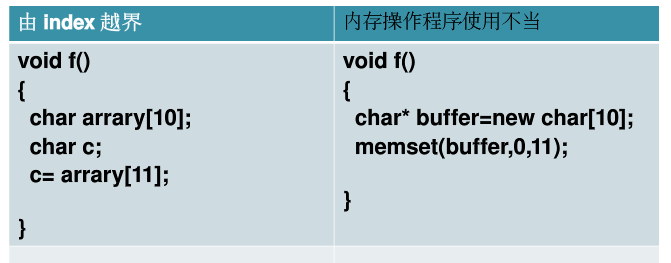


# CheckBufferOverrun类：

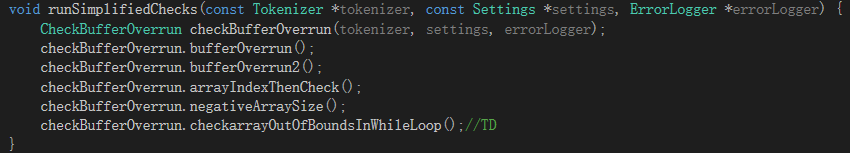
这个类继承于Check这个基类，用于检查缓冲区是否溢出。缓冲区溢出也就是说内存被分配到了被分配内存空间之外的内存空间，是因为计算机向缓冲区写入数据的时候超出了缓冲区的容量，从而导致了缓冲区溢出的问题，使得溢出的数据覆盖了其他内存空间的数据。

可能会导致程序运行失败，系统宕机，重新启动等后果。

可能的原因：

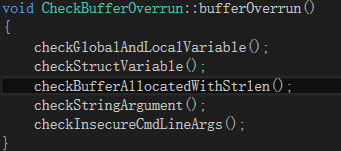


首先这个类继承了Check类的一些方法：



这是对简化后的代码进行检查，其中声明了6个方法。

checkBufferOverrun.bufferOverrun();



这个方法中检查了全局、局部变量、结构变量是否定义正确，然后检查了缓冲区和字符长度的分配是否有错，检查了字符串参数的使用是否有错，最后还检查了不牢靠的命令行参数。

checkBufferOverrun.bufferOverrun2();

这个方法检查了ast,符号库以及数据流。

checkBufferOverrun.arrayIndexThenCheck();

这个方法对数组及其索引是否有错。

checkBufferOverrun.negativeArraySize();

检查当定义数值为负数的数组的时候是否出错。

checkBufferOverrun.checkarrayOutOfBoundsInWhileLoop()

检查在执行while循环的时候是否出现了数组越界的问题。



这里声明了一个方法，为了检查指针是否造成了缓冲区溢出。

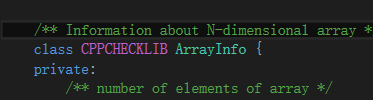


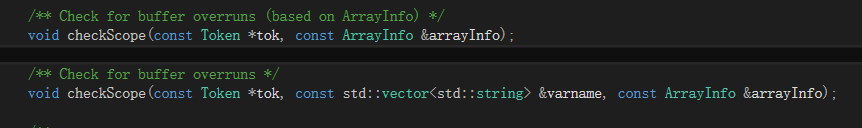
检查strncat的用法是否有错



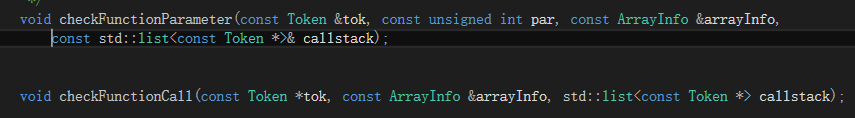
这个方法主要用来报告可能出现的缓冲区溢出的错误信息。

在这个类中定义了另一个类，用来存放数组的信息。

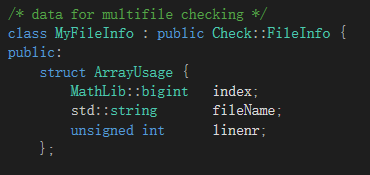




这个函数是对数组的全局进行检查，看数组在使用的过程中是否出错。

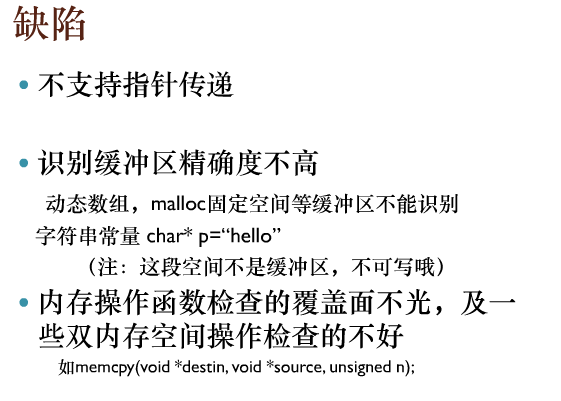


检查函数的参数以及调用是否正确。

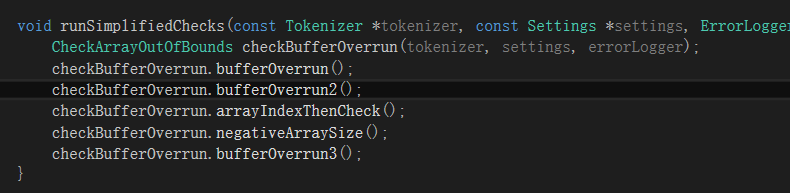


然后又定义了一个MyFileInfo类，这个类继承于基类FileInfo,这里存放了数组的各种信息以便于检查数组出错之后返回错误信息。

但是这个类的设计存在一些缺陷：

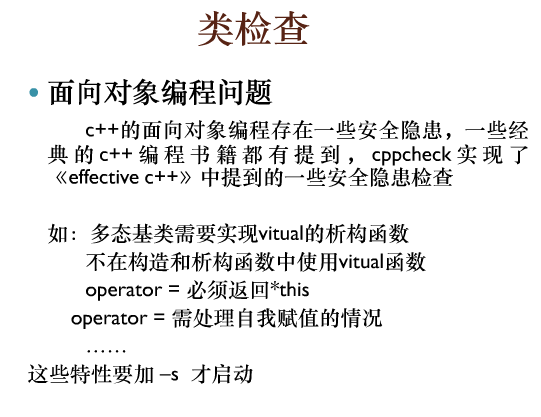


CheckArrayOutOfBounds类

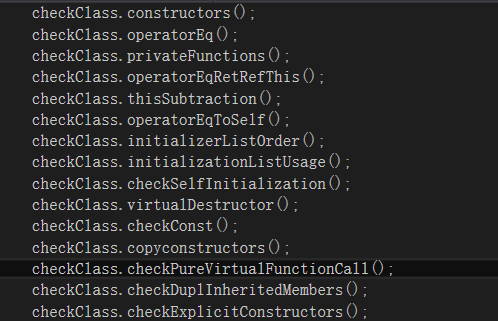


这个类主要用来判断数组越界的问题，同样是Check的子类，在类里面调用了很多CheckBufferOverrun类里面的方法，基本上都是CheckBufferOverrun里面的方法。

Checkclass类



Checkclass这个类继承与Check这个类，主要作用是对定义的类进行检查，定义了下列的一些方法。





首先，constructors()这个方法是为了检查是否有构造器，是否有需要初始化的变量没有初始化。



这个方法用来检查操作符使用是否正确。



检查私有函数的使用是否正确。



检查返回的标识是否正确。



检查this的用法是否正确。



检查operator处理自我赋值的情况。



检查列表顺序是否初始化了。



检查列表一些使用是否初始化了。



检查变量是否自我初始化了。



检查隐式构造器是否正确使用。



检查const常量的定义。



检查构造器的复制是否正确。



检查纯虚函数的使用是否正确。



检查是否以相同的名称隐藏继承的成员。



检查显示构造函数的使用是否正确。

# Checkmemoryleak 检查是否有内存泄露

内存泄露是指在程序运行时，动态申请的内存（堆）使用过后没有释放，导致一些内存永远得不到回收

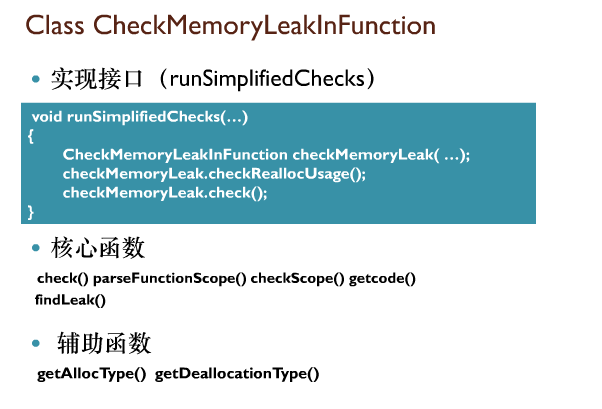
此项功能是CPPCHECK的“最优价值”检查功能，实现代码最长。

其中包含四个类：

**CheckMemoryLeakInFunction**

功能：以函数为单位的检查，参数、局部变量、指针的检查



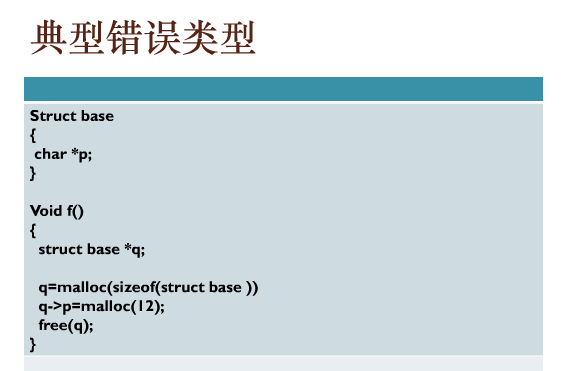


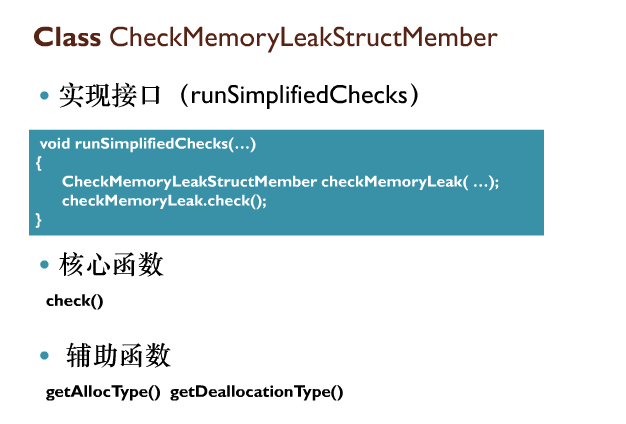
-----------------------------------------------------------------------------------------------------------

**CheckMemoryLeakStructMember**

功能：在一个函数内检查成员结构体被释放（返回）时，其成员不释放空间。

返回错误：（error）Memory leak:STRUCT.MEMBER 中间点表示成员。





---------------------------------------------------------------------------------------------------------------------

**CheckMemoryLeakInClass**

检查：

1.成员变量在构造函数中赋空间，析构函数中没有被释放

2.成员变量在函数（所有函数，不单单是一个函数）中分配空间，与释放空间函数不成匹配（malloc-free,new-delete）出现

算法描述：两个变量：Alloc,Dealloc

一旦对成员变量分配空间，Alloc被置值，如果发现对变量释放空间函数，Dealloc置值，并立即对比，是否是同一类内存申请释放函数，如果不是，则立即报dismatch错，遍历完所有类的函数。

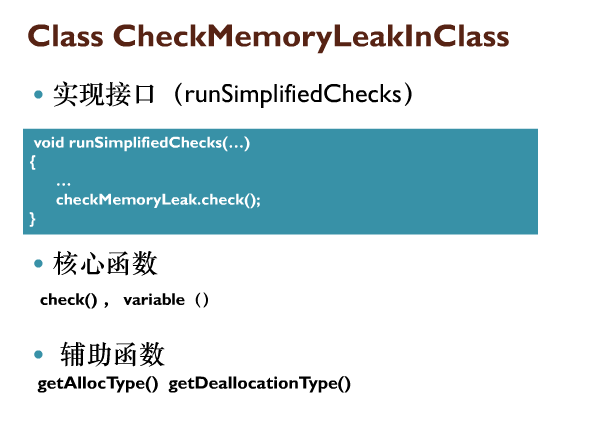


错误输出：

1.存在有Alloc无Dealloc的情况：（error）Memory leak:CLASSNAME::MEMBER

2.发现不匹配：（error）Mismatching allocation and deallocation CLASSNAME::MEMBER

3.检查构造函数，析构函数中是否成对（申请、释放）（如果构造中有内存申请，析构中必须有释放）：（error）Memory leak:CLASSNAME::MEMBER



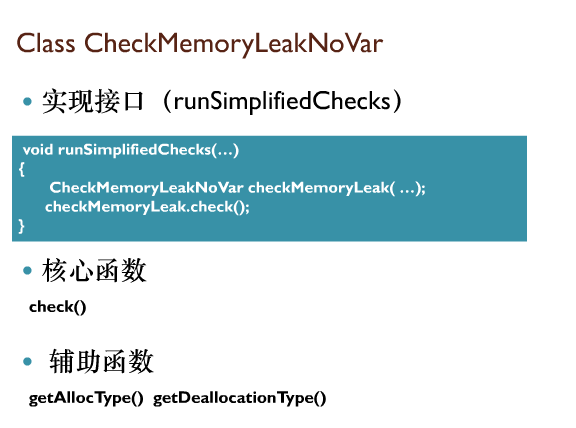
----------------------------------------------------------------------------------------------------------------------

**CheckMemoryLeakNoVar**

功能：检查参数部分有无内存泄露

报错信息：(error)Allocation with FUNCNAME,FUNCNAME doesn’t release it.





# Cppcheck

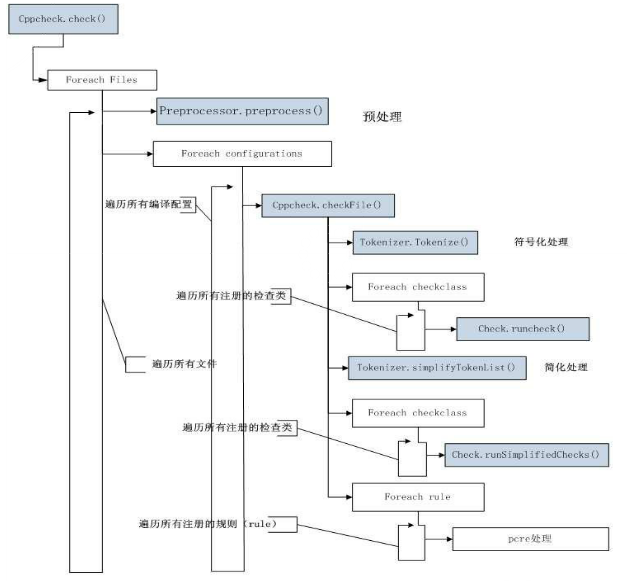
主干类，核心类

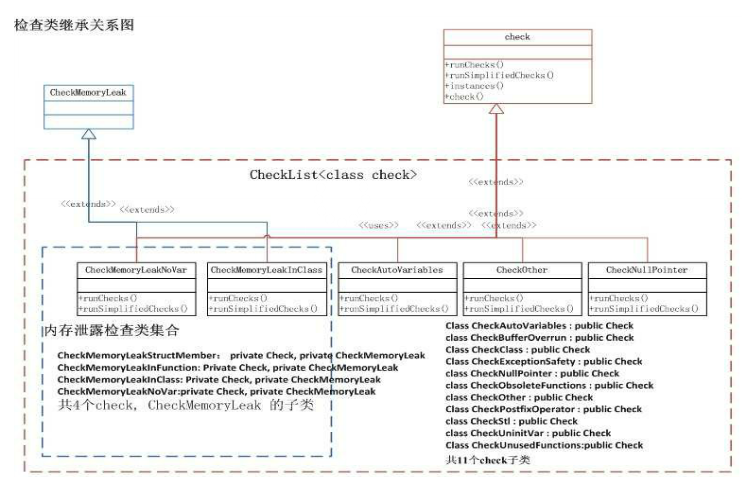
核心函数：

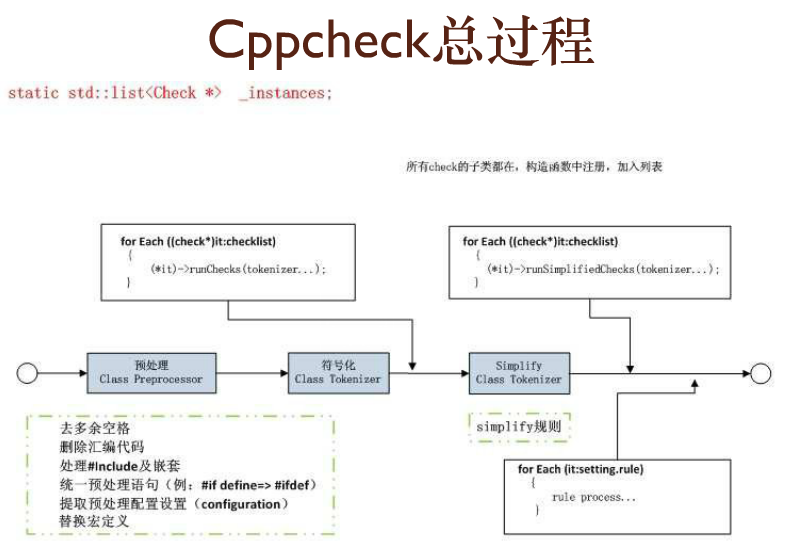
Check() 处理入口，在此函数对输入代码进行初步分析处理，最后将代码传递给CheckFile()。

CheckFile() 分析一个代码文件，将代码流做进一步分析，做字符化、简化处理分析后代码，报告错误。

两个函数的实现：







# Tokenizer代码解析成符号

主要函数：

setTimerResults() 设置定时器结果；

isC() 是C语言代码；

isCPP() 是C++语言代码；

IsScopeNoReturn() 检验内部范围是否以无返回值结尾；

Tokenize() 代码处理；

tokenizeCondition() 标记条件；

setVarId() 设置变量ID；

simplifyTokenList1() 简化处理链1；

SimplifyNamelessRValueReferences() 简化无名R值引用；

simplifyTokenList2()简化处理链2；

eraseDeadCode() 删除无效代码；

simplifyMulAndParens() 简化乘法和括号；

sizeOfType() 计算给定类型的大小值；

isFunctionParameterPassedByValue() 试图确定函数按值传递的参数是通过查看函数声明；

getErrorMessages() 得到错误信息；

simplifyAssignmentInFunctionCall() 简化任务功能；

simplifyAssignmentBlock() 简化作业块；

simplifyCalculations() 简化计算；

simplifyOffsetPointerDereference() 简化非关联化，一个指针抵消一个数字；

simplifyOffsetPointerReference() 简化引用指针偏移量；

arraySize() 在没有给出的地方插入数组大小；

simplifyLabelsCaseDefault() 简化标签和案例、违约的语法；

simplifyCaseRange() 简化范围，gcc扩展；

removeMacrosInGlobalScope() 在全部范围内删除宏；

removeMacroInClassDef() 在类定义中删除未定义的宏；

removeMacroInVarDecl() 删除未知的宏变量声明；

removeRedundantAssignment() 删除冗余分配；

simplifyRealloc() 简化realloc使用；

sizeofAddParentheses() 添加括号运算符；

simplifySizeof() 将sizeof替换为合适大小；

simplifyVarDecl 简化变量声明；

simplifyInitVar() 简化变量初始化；

simplifyPlatformTypes() 平台依赖类型转换为标准类型；

simplifyStdType() 将复合的定义简化为单字；

simplifyConstTernaryOp() 简化表达式中三元运算；

simplifyCompoundAssignment() 简化复合作业；

simplifyPointerConst() 简化“\*常数”为单纯常数；

simplifyStaticConst() 简化静态常量；

simplifyIfAndWhileAssign() 简化if、while语句中的分配；

simplifyVariableMultipleAssign() 简化多个作业；

simplifyCAlternativeTokens() 简化C语言替代字符；

simplifyComma() 在可能的情况下简化逗号、分号；

simplifyAddBraces() 为if、for等语句体添加括号；

simplifyAddBracesToCommand为if、for等语句体添加括号；

simplifyAddBracesPair为单个if、for等语句添加双括号；

simplifyTypedef() 简化类型定义；

simplifyCasts() 简化类型转换；

simplifyUndefinedSizeArray() 变更多个数组和指针；

simplifyKnownVariables() 简化已知变量；

simplifyKnownVariablesGetData 从已分配变量取得数据；

simplifyKnownVariablesSimplify 已分配变量化简；

simplifyEmptyNamespaces() 简化空命名空间，主要指C++代码；

simplifyFlowControl() 简化控制流代码的冗余；

simplifyNestedStrcat() 扩大嵌套的strcat()调用；

elseif() 简化if、else语句；

simplifyConditions() 简化条件；

removeRedundantConditions() 删除冗余代码；

removeRedundantFor() 删除冗余for循环；

removeRedundantSemicolons() 删除冗余分号；

simplifyFunctionReturn() 简化返回功能；

simplifyStructDecl() 结构简化；

simplifyRedundantParentheses() 删除冗余括号；

simplifyCharAt()

simplifyReference() 简化引用；

simplifyFunctionParameters() 简化功能参数；

simplifyTemplates() 简化模板；

simplifyMathFunctions 简化数学功能；

simplifyMathExpressions 简化数学表达；

isFunctionHead 检查令牌是否指向函数头；

simplifyWhile0() 简化while（0）；

simplifyErrNoInWhile() 简化不再在循环内的情况；

simplifyFuncInWhile() 简化循环内；

simplifyEnum() 用恒定值代替枚举；

simplifyStd() 在一些情况下移除std；

simplifyPointerToStandardType() 将指针简化为标准形式，只在C语言中；

simplifyFunctionPointers() 简化功能指针；

removeExceptionSpecifications() 删除异常规范；

cppcheckError 发送错误信息；

createLinks() 为令牌设置链接；

createLinks2() 在有and的情况下实现令牌链接；

syntaxError() 语法错误；