CppCheck核心类分析

TooSimple小组

目录

[CppCheck核心类分析 1](#_Toc502955324)

[概述 2](#_Toc502955325)

[CppCheck简介 2](#_Toc502955326)

[文档内容说明 3](#_Toc502955327)

[关键字 3](#_Toc502955328)

[功能分析 3](#_Toc502955329)

[Check类 3](#_Toc502955330)

[CheckAutoVariables类 4](#_Toc502955331)

[CheckBufferOverrun类 6](#_Toc502955332)

[CheckArrayOutOfBounds类 8](#_Toc502955333)

[Checkclass类 8](#_Toc502955334)

[CheckMemoryLeakInFunction类 10](#_Toc502955335)

[CheckMemoryLeakStructMember类 11](#_Toc502955336)

[CheckMemoryLeakInClass类 12](#_Toc502955337)

[CheckMemoryLeakNoVar类 13](#_Toc502955338)

[Cppcheck类 14](#_Toc502955339)

[Tokenizer类 16](#_Toc502955340)

[总结 18](#_Toc502955341)

[参考资料 18](#_Toc502955342)

# 概述

## CppCheck简介

CppCheck 是一个静态代码检查工具，支持C, C++ 代码。不同于C/C++编译器及其它分析工具，CppCheck只检查编译器检查不出来的bug，不检查语法错误，可以作为编译器的一种补充检查。执行的检查内容主要包括以下几项：

1. 自动变量检查；

2. 数组的边界检查；

3. class类检查；

4. 过期的函数，废弃函数调用检查；

5. 异常内存使用，释放检查；

6. 内存泄漏检查，主要是通过内存引用指针；

7. 操作系统资源释放检查，中断，文件描述符等；

8. 异常STL 函数使用检查；

9. 代码格式错误，以及性能因素检查；

本次成立项目组，主要目的是为了对CppCheck进行分析和二次开发，完善该程序的一些功能，本文档是对CppCheck核心类的相关分析，为后期二次开发打下基础。

## 文档内容说明

本文主要对CppCheck的核心类进行分析，主要是代码检查功能模块，包括数组是否越界、缓存区是否溢出、代码的简化和字符的转换等11个类的具体分析。详细请见功能分析。

## 关键字

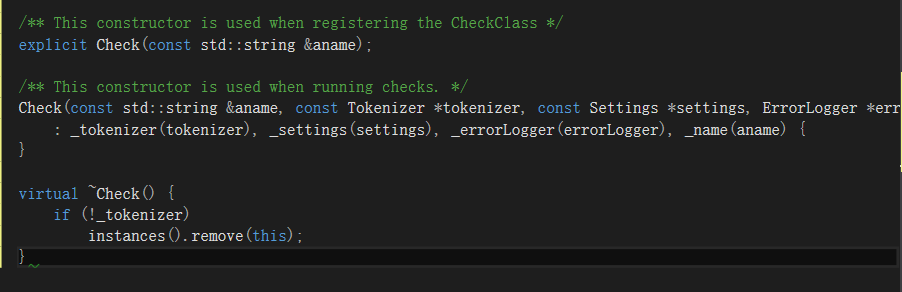
CppCheck，类，检查功能，缓存区溢出，方法。

# 功能分析

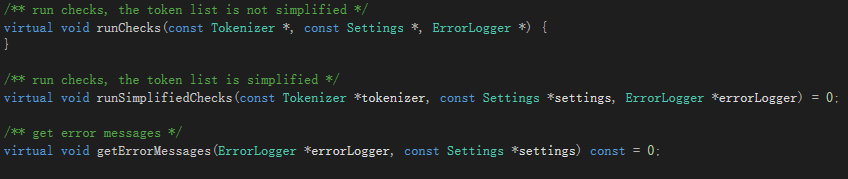
## Check类

Check类是CppCheck中的一个基类，类中定义了很多函数或方法，很多功能模块类都继承自这个基类：

首先，Check类中显示的构造函数和析构函数显示如下：



然后定义了一些方法来执行代码检查



这3个方法的作用分别是：

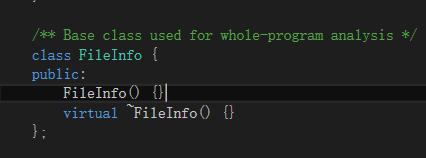
1. runChecks() 执行代码的检查；
2. runSimplifiedChecks() 对简化之后的代码开始执行检查；

c.getErrorMessages() 这是获取报告的错误信息 ；

报告错误信息的静态方法：

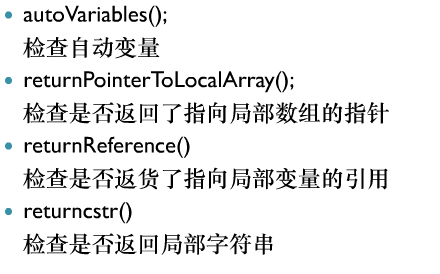


Check类中定义了一个FileInfo类，用来存储文件代码的各种信息，以便于检查代码中出现的错误：

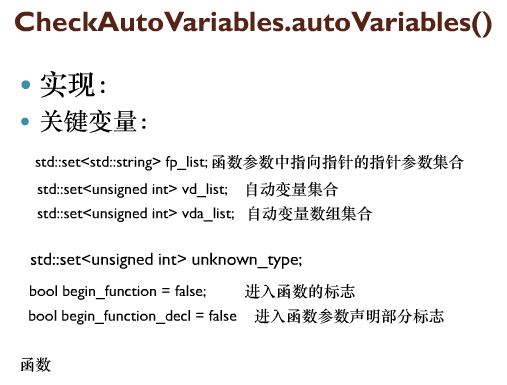


## CheckAutoVariables类

CheckAutoVariables类是Check的一个子类，主要功能是实现对自动变量的检查，类中主要定义了如图所示的几个方法：



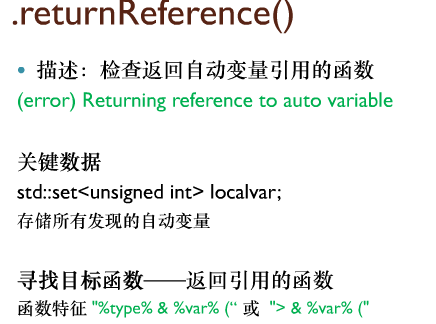
Auto Variables()方法的实现：



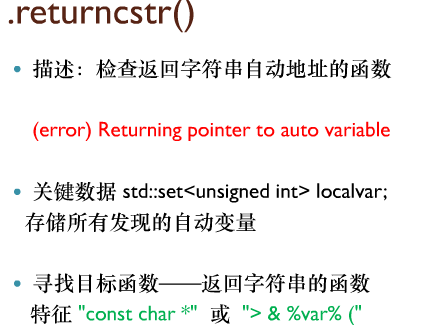
returnPointerToLocalArrary()方法检查程序中是否返回了局部的指针：



returnReference()是返回自动变量引用的函数：



Returncstr()方法检查返回字符串自动地址：



存在问题：

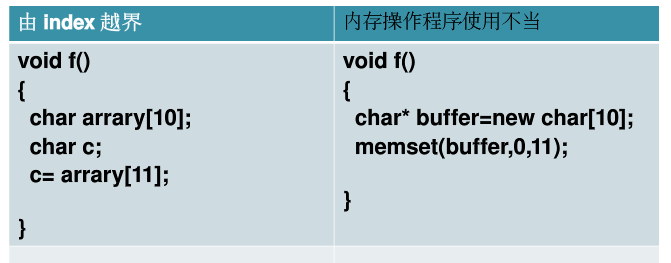
* 没有处理指针传递；
* 识别不精确；
* 错误检查不精确；

## CheckBufferOverrun类

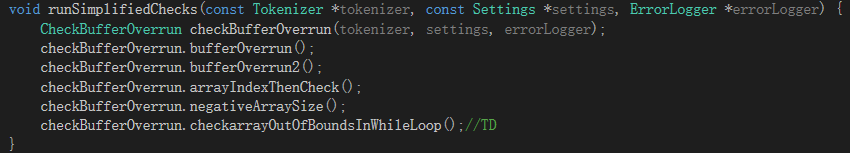
CheckBufferOverrun类继承于Check基类，主要功能是检查缓冲区是否溢出。缓冲区溢出是指程序运行时，存在内存被分配到了特定内存空间之外的地址，其中特定内存空间是指程序本身被分配的区域。这种问题出现是因为计算机向缓冲区写入数据的时候超出了缓冲区的容量，从而导致了缓冲区溢出的问题，使得溢出的数据覆盖了其他内存空间的数据。

缓存区溢出问题可能会导致程序运行失败，系统宕机，重新启动等后果。

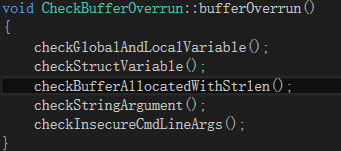
可能导致缓存区溢出的原因：



CheckBufferOverrun类继承了Check类的一些方法，主要功能是对简化后的代码进行检查，其中声明了6个方法：



checkBufferOverrun.bufferOverrun()方法主要功能是检查全局变量、局部变量、结构变量是否定义正确，缓冲区和字符长度的分配是否有错，字符串参数的使用是否有错，以及不牢靠的命令行参数，反复的具体定义如下：



checkBufferOverrun.bufferOverrun2()方法检查ast,符号库以及数据流。

checkBufferOverrun.arrayIndexThenCheck()方法检查数组及其索引是否有错。

checkBufferOverrun.negativeArraySize()方法检查当定义数值为负数的数组的时候，是否出错。

checkBufferOverrun.checkarrayOutOfBoundsInWhileLoop()方法检查在执行while循环的时候是否出现了数组越界的问题。

pointerOutOfBoundsError()方法检查指针是否造成了缓冲区溢出。



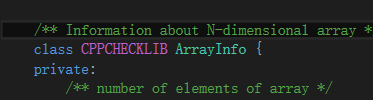
strncatUsageError()方法检查strncat的用法是否有错。



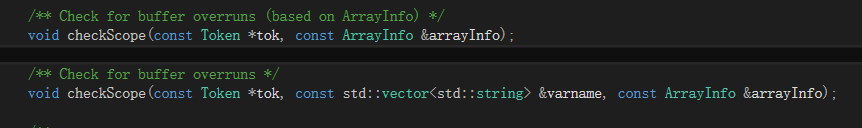
possibleBufferOverrunError()方法主要用来报告可能出现的缓冲区溢出的错误信息。



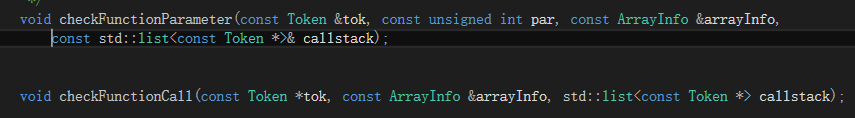
本类中定义了ArraryInfo类，用来存放数组的信息。



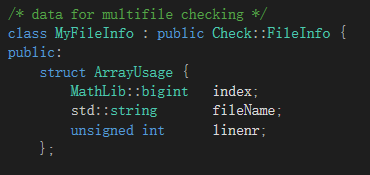
checkScope()函数是对数组的全局进行检查，看数组在使用的过程中是否出错。



CheckFunctionParameter()方法检查函数的参数以及调用是否正确。



本类中定义了MyFileInfo类，这个类继承于基类FileInfo，用来存放数组的各种信息以便于检查时，数组出错之后返回错误信息。

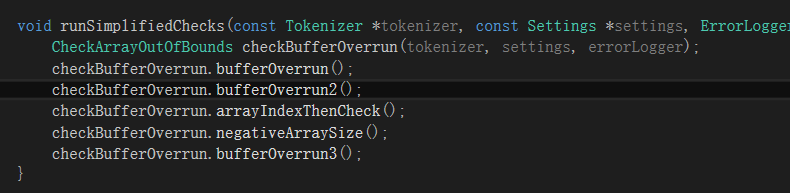


存在问题：

* 不支持指针传递；
* 识别缓冲区精度不高；
* 内存操作函数检查的覆盖度不够，双内存空间操作检查的不好；

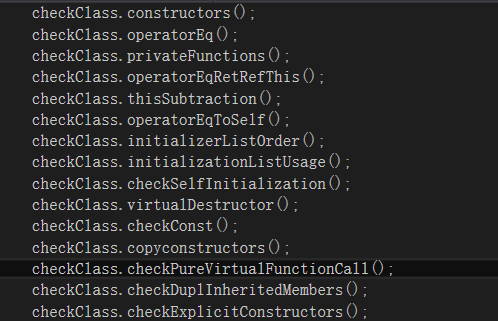
## CheckArrayOutOfBounds类

CheckArrayOutOfBounds类主要用来判断数组是否越界的问题，是Check类的子类，同时类中调用了大量CheckBufferOverrun类的方法



## Checkclass类

Checkclass类继承于Check类，主要作用是对定义的类进行检查，其中包含下列方法：



constructors()方法检查是否有构造器，是否有需要初始化的变量没有初始化。

operatorEq()方法检查操作符使用是否正确。



privateFunctions()检查私有函数的使用是否正确。



operatoeEqRetRefThis()方法检查返回的标识是否正确。



thisSubtraction()方法检查this的用法是否正确。



operatorEqToSelf()方法检查operator处理自我赋值的情况。



initializerListOrder()方法检查列表顺序是否初始化。



initializationListUsage()方法检查列表使用是否初始化。



checkSelfInitialization()方法检查变量是否自我初始化。



virtualDestructor()方法检查隐式构造器是否正确使用。



checkConst()方法检查const常量的定义。



copyconstructors()方法检查构造器的复制是否正确。



checkPureVirtualFunctionCall()方法检查纯虚函数的使用是否正确。



checkDup1InheritedMembers()方法检查是否以相同的名称隐藏继承的成员。



checkExplicitConstructors()方法检查显示构造函数的使用是否正确。

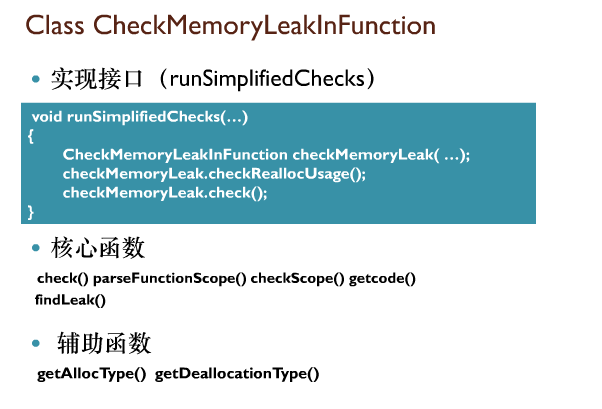


## CheckMemoryLeakInFunction类

功能：以函数为单位的检查，参数、局部变量、指针的检查。

典型错误如下所示：

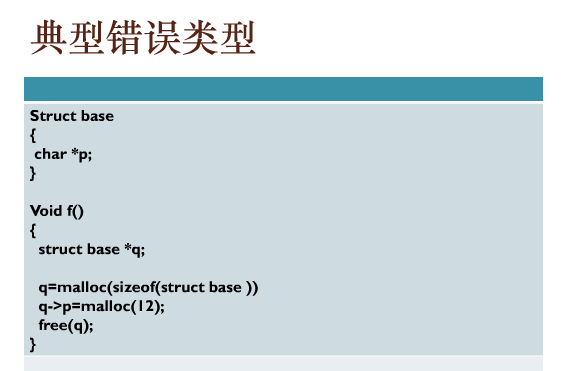


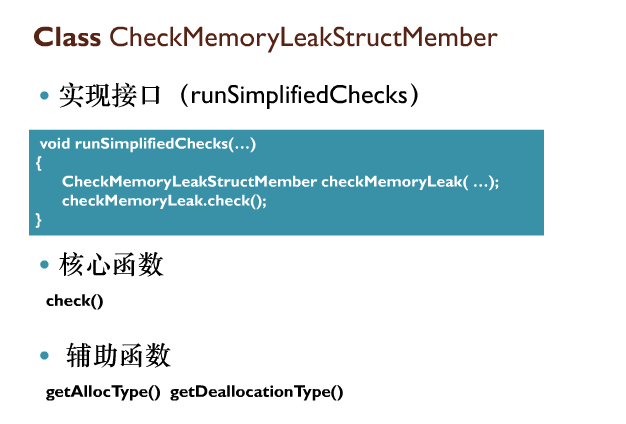


## CheckMemoryLeakStructMember类

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类

功能：

1.检查成员变量在构造函数中赋空间，析构函数中没有被释放

2.检查成员变量在函数（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。

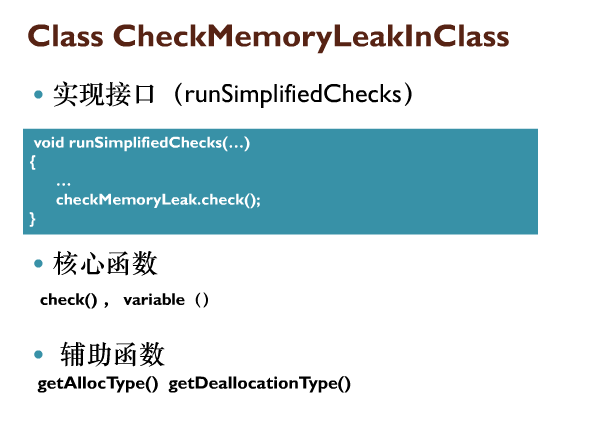


错误输出：

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。

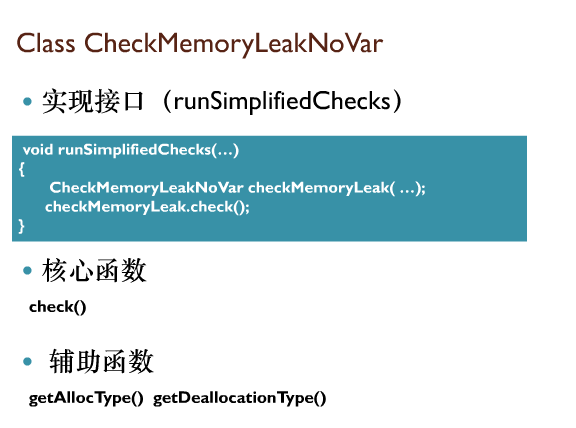


## CheckMemoryLeakNoVar类

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.





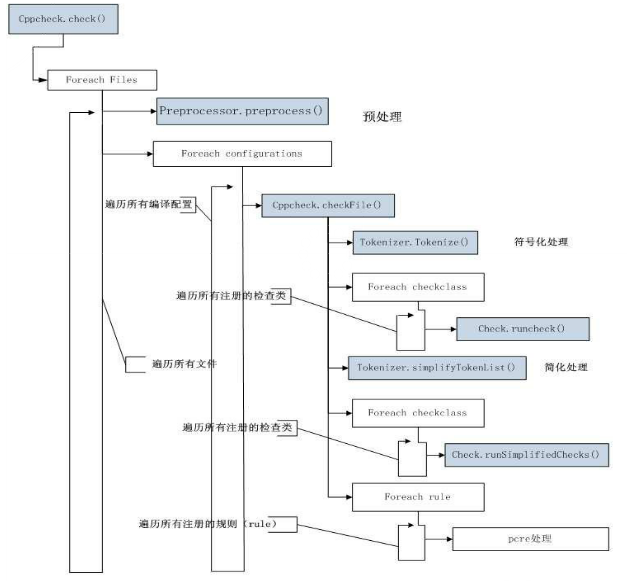
## Cppcheck类

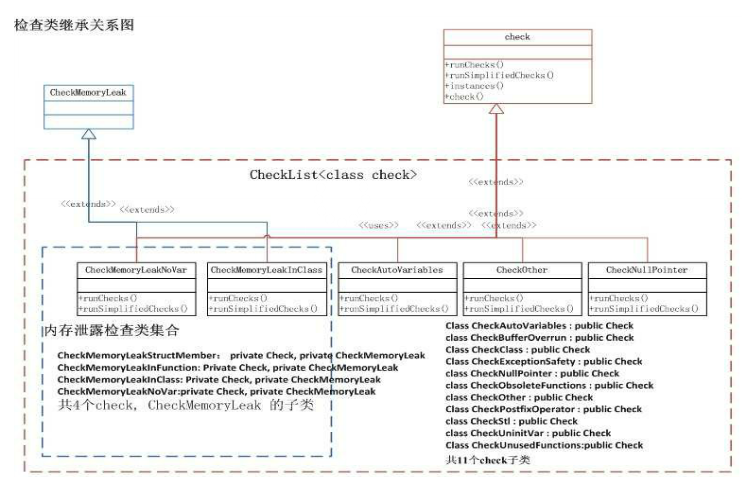
Cppcheck类是整个程序的一个主干类，它的核心函数主要是：

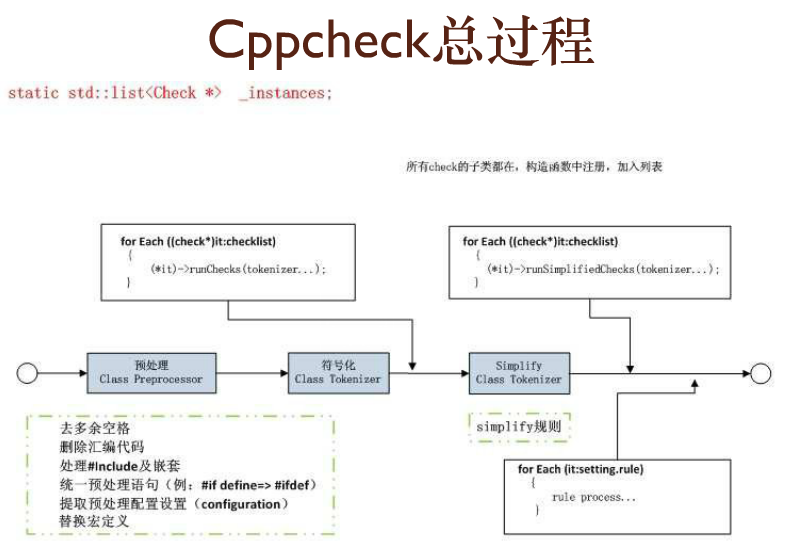
Check() 处理入口，在此函数对输入代码进行初步分析处理，最后将代码传递给CheckFile()。

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。

两个函数的实现方法：







## Tokenizer类

功能：将代码解析成符号。

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；

# 总结

在对CppCheck的代码分析过程中，本小组结束了前两周的工作，基本目标已经完成，对CppCheck的代码有了比较深刻的认识。在对CppCheck目前存在的问题分析、讨论，以及对小组成员的能力综合考虑之后，小组成员已经初步明确了对CppCheck的代码进行二次开发的方向，准备着手改进CppCheck的部分功能，完善现有的无法检查出来的部分功能，将不支持指针传递等问题进行深入研究和改良。

# 参考资料

1. <http://blog.csdn.net/e5max/article/details/11489137>
2. http://wenku.baidu.com/view/7d9804c7bb4cf7ec4afed0aa.html
3. http://wenku.baidu.com/view/018b5a35a32d7375a41780ab.html