嵌入式应用代码走查CheckList

密级：内部公开

**目录**

[1. 编写目的 3](#_Toc481592406)

[2. 参考文献 3](#_Toc481592407)

[3. 走查CheckList 3](#_Toc481592408)

[3.1 程序版式及风格 3](#_Toc481592409)

[3.2 命名规则 4](#_Toc481592410)

[3.3 表达式与基本语句 4](#_Toc481592411)

[3.4 变量检查 5](#_Toc481592412)

[3.5 函数检查 5](#_Toc481592413)

[3.6 内存管理 7](#_Toc481592414)

[4. 修订记录 8](#_Toc481592415)

# 编写目的

文档描述了代码走查流程及代码走查CheckList，目的是使相关人员明确代码审查关注事项，规范代码编写风格及习惯，保证程序的健壮及稳定，最终提升产品质量。

# 参考文献

<<软件开发规范\_海康.pdf>>

<<C++ C编程指南.pdf>>

# 走查CheckList

## 程序版式及风格

|  |  |
| --- | --- |
| 审查项 | 结论（通过/不通过） |
| 头文件是否使用了 **ifndef/define/endif** 预处理块？ |  |
| 是否包含了无用的头文件，或者重复包含头文件？ |  |
| 头文件是否只做函数和变量的声明？ |  |
| 缩进的空格数为**4**个？ |  |
| 较长的语句分成多行书写？ |  |
| 一行只写一条语句？ |  |
| 函数的行数是否尽量少？（小于500行） |  |
| **if、for、while、do** 等语句自占一行，不论执行语句多少都要加“**{}**”？同一文件中{}的风格保持一致？ |  |
| 在定义变量（或参数）时，是否将修饰符 **\*** 和**＆** 紧靠变量名？ |  |
| 注释的内容是否清楚、明了，含义准确？注释是否准确、清晰并且必要？ |  |
| 源文件头部是否进行了注释，列出：版权说明、文件名、修改日志等？函数呢？类、结构、宏、枚举？关键代码？ |  |
| 类结构的 **public, protected, private** 顺序是否在所有的程序中保持一致？ |  |
| 宏定义中，参数加括号了吗？参数要加括号 |  |
| 同一工程文件代码风格是否一致？代码结构是否清晰？ |  |
| 文件的命名是否可以体现出该文件的作用？ |  |

## 命名规则

|  |  |
| --- | --- |
| 审查项 | 结论（通过/不通过） |
| 标识符的命名是否清晰、明了，有明确的含义？命名规则是否符合规范，能否表明变量类型、数据类型？比如：静态变量前用**s\_**，全局变量前用**g\_**，**c++**的成员变量前用**m\_**等。 |  |
| 是否用正确的反义词组命名具有互斥意义的变量或相反动作的函数等？ |  |

## 表达式与基本语句

|  |  |
| --- | --- |
| 审查项 | 结论（通过/不通过） |
| 如果代码行中的运算符比较多，是否已经用括号清楚地确定表达式的操作顺序？比如：**if ((a | b )&&( a & c))，尽量用()区分** |  |
| 是否编写太复杂或者多用途的复合表达式？比如：\* **stat\_poi ++ += 1,原则上拆分为\*stat\_poi += 1; stat\_poi++;** |  |
| 是否直接使用了不易理解的数字（魔数）？ |  |
| **switch** 语句是否有**default**分支？**case** 语句的结尾是否有加**break**？ |  |
| 使用 **goto** 语句时是否留下隐患？例如跳过了某些对象的构造、变量的初始化、重要的计算等。 |  |
| 是否循环体内工作量最小化？是否在多重循环中，将最忙的循环放在最内层？ |  |
| 如果循环体内存在逻辑判断，并且循环次数很大，是否已经将逻辑判断移到循环体的外面？ |  |
| 是否用乘法或其它方法代替除法，特别是浮点运算中的除法？ |  |

## 变量检查

|  |  |
| --- | --- |
| 审查项 | 结论（通过/不通过） |
| 是否一定要使用全局变量？若需要，可否加static修饰？ |  |
| 当向全局变量传递数据时，是否做了合法性检查，防止赋予不合理的值或越界等现象发生？ |  |
| 是否有局部变量与全局变量同名？ |  |
| 是否有未经初始化的变量作为右值？变量的初始化或缺省值有错误吗？ |  |
| 使用的数据类型是否利于程序的移植？可用HPR开头的海康变量标准。 |  |
| 是否考虑网络通信时的字节顺序？或不同**CPU**间的字节对齐？ |  |
| 是否有数据运算溢出的可能性？比如数组大小是否足够，数据访问是否越界，变量范围是否足够表示数据，数据在运算是是否发生了上溢或下溢错误？ |  |
| **C++**中，是否用**const**定义常量？ |  |
| 指针变量检查时，是否直接与“**NULL**”比较？ |  |
| 变量的精度够吗？存在不同数据类型的赋值及比较吗？ |  |

## 函数检查

|  |  |
| --- | --- |
| 检查项 | 结论（通过/不通过） |
| 函数的名字是否清晰的定义了它的目标以及函数所做的事情？ |  |
| 函数是否高内聚只做一件事情，并做好？是否有优化空间？ |  |
| 函数是否需要返回值？若需要，返回值类型是否合理？ |  |
| 函数的参数是否书写完整？函数的参数是否合理，能否尽量控制在**5**个以内？是否有对函数参数做合法性检查？ |  |
| 是否有将函数参数作为工作变量？ |  |
| 若参数是指针，且仅作输入用，是否有在类型前加**const**？ |  |
| 若输入参数以值传递的方式传递对象，**C++**语言时，是否用“**const &**”方式传递，以省去临时对象的构造和析构过程，提高效率？ |  |
| 是否有重复代码？或者相似的代码可以分离出来？ |  |
| 使用滥用了 **assert**？ 例如混淆非法情况与错误情况，后者是必然存在的并且是一定要作出处理的。 |  |
| **return** 语句是否返回指向“栈内存”的“指针”或者“引用”？ |  |
| 文件句柄的打开与关闭是否配对？ |  |
| 构造函数中是否遗漏了某些初始化工作？ |  |
| 析构函数中是否遗漏了某些清除工作？ |  |
| 是否一定要使用锁？锁住的区域是否合理？加锁和解锁是否对应？有无死锁可能？ |  |
| 可重入函数是否使用了全局变量或者静态局部变量？若使用是否有保护措施？ |  |
| 错误处理问题：   1. 忘记进行错误处理吗？ 2. 错误处理程序块一直没有机会被运行？ 3. 错误处理程序块本身就有毛病吗？如报告的错   误与实际错误不一致，处理方式不正确等等。  4， 错误处理程序块是“马后炮”吗？如在被它被调用之前软件已经出错。 |  |
| 函数是否有调测开关？调测开关是否有相应的等级及类型？开关打开时，是否输出需要的信息？ |  |

## 内存管理

|  |  |
| --- | --- |
| 审查项 | 结论（通过/不通过） |
| **malloc** 语句是否正确无误？例如字节数是否正确？  类型转换是否正确？ |  |
| 用 **malloc** 或**new** 申请内存之后，是否立即检查指针值是否为**NULL**？（防止使用指针值为NULL 的内存） |  |
| 是否将 **malloc/free** 和 **new/delete** 混淆使用？ |  |
| 是否忘记为数组和动态内存赋初值？（防止将未被初始化的内存作为右值使用） |  |
| 数组或指针的下标是否越界？是否有读写越界？ |  |
| 动态内存的申请与释放是否配对？（防止内存泄漏） |  |
| 是否修改“指向常量的指针”的内容？ |  |
| 是否出现野指针？例如   1. 指针变量没有被初始化。 2. 用**free** 或**delete** 释放了内存之后，忘记将指针设置为**NULL**。 |  |

# 修订记录

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **序号** | **变更时间** | **版本** | **变更人** | **审批人** | **变更说明** |
| 1 | 2017-05-2 | V1.0.0 | 苏鹏东 | 金静阳 | 新增 |