C++编码规范

新能源2022年新员工培训

梁凯

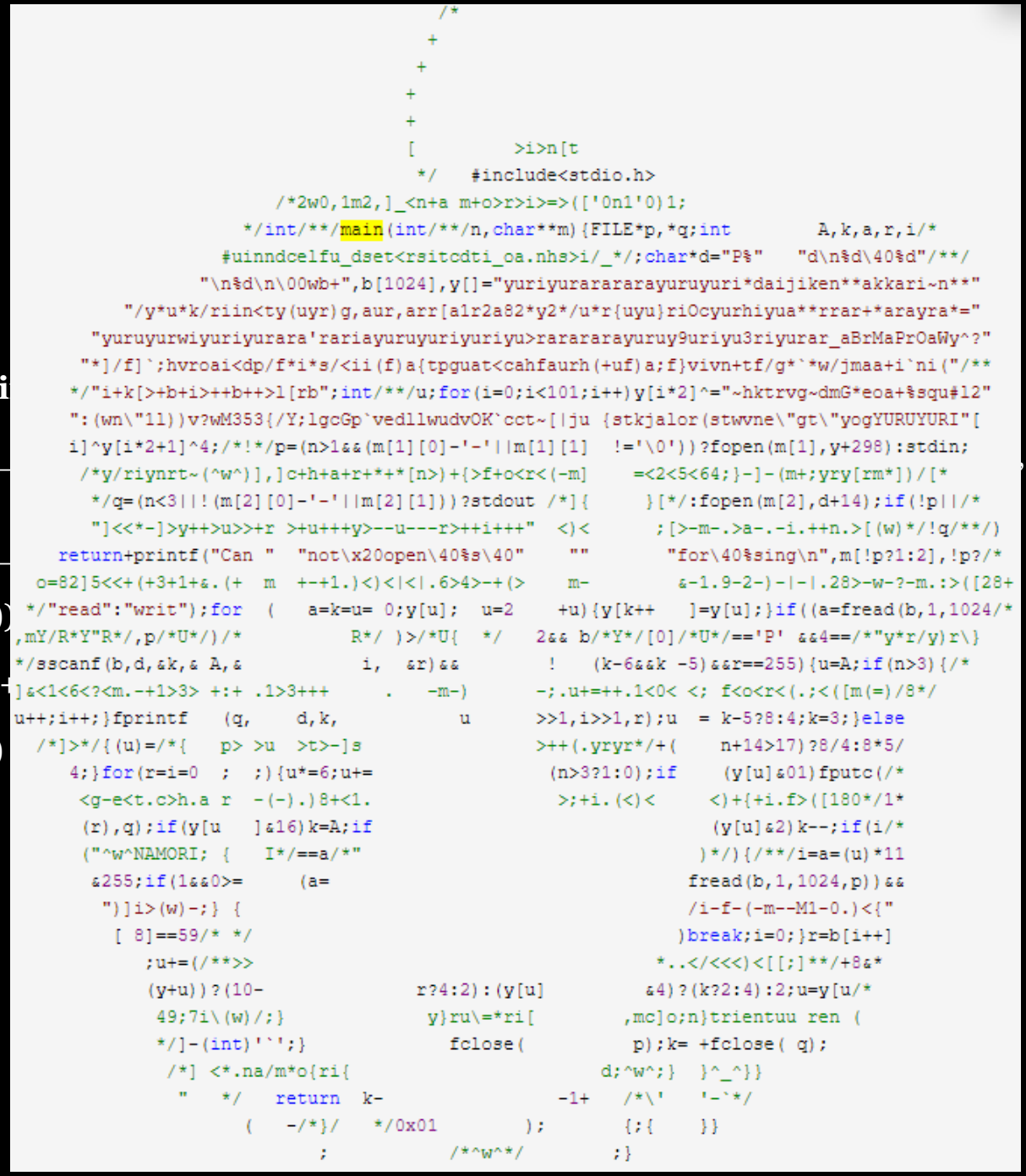
背景调查

- C/C++编程背景
  - 总人数
  - 学习过C/C++的人数
  - 了解C++98/03、C++11/14、C++17
  - 按照某种编码规范编程的人数
  - 知道或遵守过哪些编程规范

代码阅读

- 下面代码是什么功能

```
void _(int __, int ____, int _____, i

  _((____ / __) <= _____) ? _(__,__
_) :

   ((____ % __)==(____ / __) && !__

   _(__,____+_____,_____,_____)

   _(__,____+_____,_____,_____)

   _(__,____+_____,_____,_____)

}

int main() {

   _(100,0,0,1);

}
```



规范背景

- 公司级别《Q/JC GL 0310-2020 C&C++编码规范》
  - 100条规则，5条建议
- 部门级别继续沿用《电网自动化事业部编码规范》
  - 15条规则，5条建议
- 规范参考
  - 林锐编著的《高质量 C++ 编程指南》
  - 陈世忠编著的《C++编码规范》
  - 华为公司的《编程规范与范例》

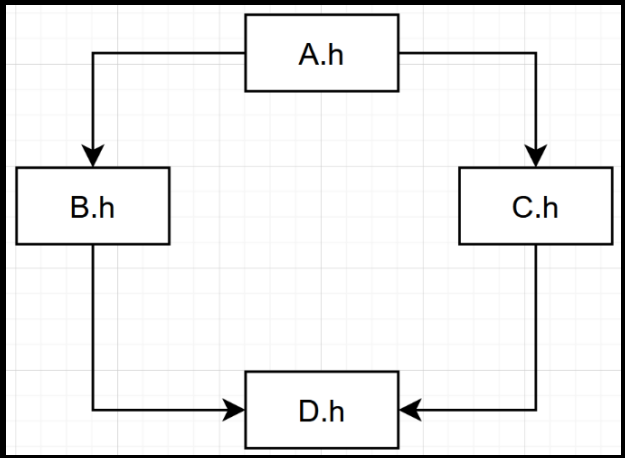
编码规范组织形式

- 1 文件名称及结构
- 2 程序的版式
- 3 命名规则
- 4 表达式和基本语句
- 5 函数设计
- 6 内存管理

- 规范分类
  - 规则，编程者必须遵守的
  - 建议，不做强制性要求，但推荐编程者采纳

文件名称及结构

【规则3-1】使用ifndef/define/endif预处理块防止头文件被重复引用



- 解决问题
  - 避免头文件被直接或间接地多次引用
  - 避免类型、变量、宏等重复定义或循环引用
  - 加快编译速度
- 代码编写
  - 使用预处理块，宏名<文件名>_H
  - 全部采用大写字母
  - endif处用注释标明该宏名称

```
// 版权和版本声明，此处省略

#ifndef  GRAPHICS_H
#define  GRAPHICS_H

#include <math.h>            // 引用头文件
...          // 你的代码

#endif    // GRAPHICS_H
```

文件名称及结构

【规则3-2】用 <> 格式引用依赖库的头文件，用 "" 格式引用自定义头文件

- 解决问题
  - 便于识别自定义文件
  - 加快编译速度
- 系统头文件
  - 标准库的头文件
  - 开发工具提供库的头文件
  - 第三方提供库的头文件
- 自定义头文件
  - 公司内基础库的头文件
  - 项目自身产生的头文件

```
#include <math.h>            // 引用系统头文件
…
#include “myheader.h”       // 引用自定义头文件
…
…            // 你的代码
```

程序的版式

【规则4-1】程序块要采用缩进风格编写，缩进采用Tab符，其长度为 4 个空格

- 解决问题
  - 可改进代码易读性
  - 避免使用空格造成缩进或多或缺的问题

程序的版式

【规则4-2】二元操作符前后应加空格，一元操作符与被操作数之间不加空格

- 二元操作符
  - 赋值操作符，如“=”、“+=”
  - 比较操作符，如“>=”、“<=”
  - 算术操作符，如“+”、“*”、“%”
  - 逻辑操作符，如“&&”、“||”
  - 位域操作符、如“<<”、“^”
- 一元操作符
  - 如“!”、“~”、“++”、“--”、“&”（地址运算符）

if (year >= 2000)
if(year>=2000)
if ((a>=b) && (c<=d))
if(a>=b&&c<=d)
	// 良好的风格
	// 不良的风格
	// 良好的风格(紧凑格式)
	// 不良的风格

for (i = 0; i < 10; i++)
for (i=0; i<10; i++)
for(i=0;i<10;i++)
for (i = 0 ; i < 10 ; i ++)
	// 良好的风格
	// 良好的风格(紧凑格式)
	// 不良的风格
	// 过多的空格

x = a < b ? a : b;
x = a<b ? a : b;
x=a<b?a:b;
	// 良好的风格
	// 良好的风格(紧凑格式)
	// 不良的风格

程序的版式

【规则4-3】if、for、while、do、switch、try、catch等语句自占一行，执行语句不得紧跟其后，并且都要加{}

【规则4-4】程序的分界符'{'和'}'应独占一行并且位于同一列，同时与引用它们的语句 左对齐

- 注意：不论执行语句有多少都要加{}，以防止书写失误

```
if (x < y)                  // 符合规则的写法
{
   …
}

if (x < y){                 // 违反规则的写法
   …
}
```

程序的版式

【建议4-1】长表达式要在低优先级操作符处拆分成新行，操作符放在新行之首（以便突出操作符）。拆分出的新行要进行适当的缩进，使排版整齐，语句可读

- 缩进形式
  - 在新行相对于上一行缩进一级
  - 在函数的 '(' 后对齐（这是VC编辑器自动对齐的方式）

```
// 在低优先级操作符处拆分，相对上一行缩进对齐
if ((long_variable1 >= long_variable12)
    && (long_variable3 <= long_variable14)
    && (long_variable5 <= long_variable16))
{
    doSomething();
}
// 在下一个函数参数处拆分，在函数的 '(' 后对齐
CMatrix CMultiplyMatrix(CMatrix leftMatrix,
                        CMatrix rightMatrix);
for (long_initialization;
     long_condition;
     long_update)
{
    doSomething();
}
```

程序的版式

【规则4-5】数据结构声明(包括类、结构、联合、枚举等)、全局函数/变量、静态函数/变量、类成员函数/变量，必须加以注释

- 注释的原则
  - 应有助于对程序的阅读理解
  - 加在该加的地方
  - 注释语言必须准确、易懂、简洁

```
// 矩形类
// 本类定义了矩形的坐标、宽高等属性及设置方法，也实现了矩形的绘制。
class CRect : public CShape
{
public:
    // 设置坐标
    // 输入参数：x 左上角横坐标，y 左上角纵坐标，width 矩形宽度，
height 矩形高度
    void setRect(float x, float y, float width, float height);
    ...
private:
    // 宽度
    float m_width;
    ...
};
```

程序的版式

【建议4-2】边写代码边注释，修改代码同时修改相应的注释，以保证注释与代码的一致性；不再有用的注释要删除。

【规则4-6】程序中的无用代码直接删除，只保留最新的代码和注释；

- 注意：不要指望回过头再写注释
- 可通过SVN进行源代码版本管理，查看代码进化过程

命名规则

【规则5-1】变量命名时，全局变量加前缀g_（表示global），类数据成员加前缀m_（表示member），静态变量加前缀s_（表示static）

```
// 全局变量
int g_howManyPeople = 100;
// 静态变量
static int s_initValue = 10;

enum EShape
{
  EShape_Rect,
  EShape_Circle,
  ...
};
```

```
class CRect
{
public:
  void setValue(int width, int height);
private:
  int m_width;                    // 成员变量
  int m_height;                   // 成员变量
};

void CRect::setValue((int width, int height)
{
  m_width = width;                // 加前缀后不会和参数发生冲突
  m_height = height;
}
```

命名规则

【规则5-2】函数、变量和参数用小写字母开头的单词或词组命名，使用词组时，除第一个单词外，后续单词首字母大写；宏定义全用大写的字母，用下划线分隔单词；枚举值用枚举类型（或其缩写）加枚举名的方式定义，中间以下划线分隔；名称应尽量具备自注释性

【规则5-3】类名加前缀C（表示class），结构名加前缀S（表示struct），枚举名加前缀E（表示enum），联合体加前缀U（表示union ）

```
int g_howManyPeople = 100;
static int s_initValue = 10;
#define MAX_LENGTH 100

enum EShape
{
  EShape_Rect,
  EShape_Circle,
  ...
};
```

```
class CStudent    // 类名称
{
...
};

struct SRect      // 结构名称
{
...
};
```

```
union USimple    // 联合名称
{
...
};
```

表达式和基本语句

【规则6-1】变量比较时，浮点变量不可用“==”或“!=”与任何数字比较；指针变量用“==”或“!=”与NULL比较，而且 NULL 作为左值进行判断；布尔变量不可直接与true、false或者 1、0 进行比较；整形变量需要使用==或!=与0直接比较，不能直接做布尔量判定；变量与常量比较时常量作为左值

- 说明
  - 浮点数都有精度限制，采用实际允许的最小误差转换为>=或<=的形式
  - 指针变量不能直接应用于条件语句
  - 防止书写错误为=NULL，要将NULL作为左值判断
  - 布尔变量零值表示false，而任何非零值都为true

```
// 浮点数允许误差
#define EPSILON 1.0e-06F
// 判断浮点数相等
if ( (x >= y - EPSILON) && (x <= y + EPSILON))

// 判断指针为空
if (NULL == pointer)

// 判断布尔值为TRUE
if (bContinue)

// 判断整数为0
if (0 == nCount)
```

表达式和基本语句

【规则6-2】相同内容的case可以合并，其他情况下每个case语句的结尾需要加break防止重叠；不能缺少default分支，default分支也要加break

- 说明
  - 增加default分支，避免case 没有完全覆盖所有枚举值产生编译警告
  - default分支插入断言或输出调试信息，帮助程序员得到提示

```
switch(EShape)
{
  case EShape_Rect:
    ...
    break;
  // 圆形与椭圆处理方式一致
  case EShape_Circle:
  case EShape_Ellipse:
    ...
    break;
  default:
    // 错误情况可用断言调试
    assert(FALSE && “不应该执行到此处”);
    break;
}
```

表达式和基本语句

【建议6-1】注意运算符的优先级。如果代码行中的运算符比较多，应使用括号确定表达式的操作顺序，避免使用默认的优先级

- 说明
  - 防止产生歧义并提高可读性

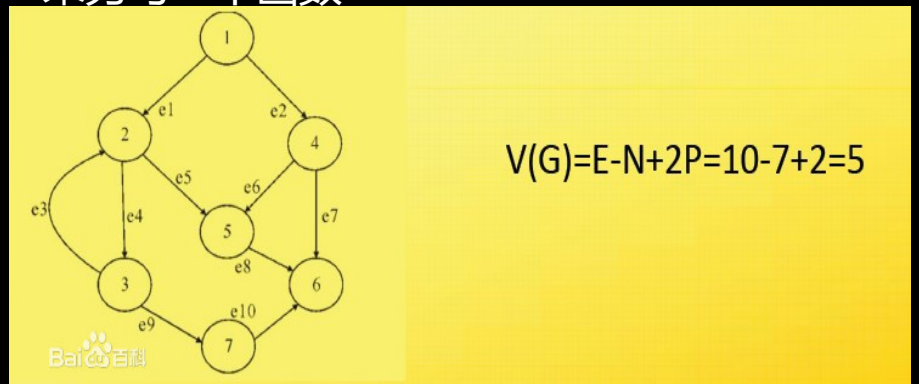
```
// 不良的风格
word = high << 8 | low
if (a | b && a & c)
```

```
// 使用括号保证优先进行比较，而后进行逻辑与计算
if ((long_variable1 >= long_variable12)
    && (long_variable3 <= long_variable14)
    && (long_variable5 <= long_variable16))
{
    doSomething();
}
```

函数设计

【建议7-1】函数的功能要单一，不要设计多用途的函数；函数体的规模要小，平均代码行数≤100，超出比例不能超过0.5%，建议函数体最好控制在50行以内；函数的圈复杂度平均小于10，超出比例不能超过0.5%

- 带来问题
  - 阅读起来很不方便
  - 程序的可读性降低
  - 函数的设计往往存在问题
- 解决方法
  - 将相对独立或重复性的部分拿出来另写一个函数



- 圈复杂度
  - 用来衡量一个模块判定结构的复杂程度，数量上表现为独立线性路径条数，即合理的预防错误所需测试的最少路径条数。
- 举例
  - 函数不包含控制流语句，圈复杂度为1
  - 如果仅包含一个if语句，且if语句仅有一个条件，圈复杂度为2
  - 如果包含两个嵌套的if语句，或是一个if语句有两个条件，圈复杂度为3

函数设计

【建议7-2】对非基本类型应尽可能使用引用（首选）或指针来传递参数；参数用作输出时应尽量使用引用而不是指针；尽可能用const修饰函数的参数

- 非基本类型

  自定义的类(class)、结构(struct)和联合(union)

- 解决问题
  - 有利于提高效率
  - 有利于函数支持派生类
  - 使代码显得更自然
  - 防止调用者传入错误的指针值而引起系统崩溃
  - 加上const修饰，以防止函数中意外地将其修改

```
// func函数
// 输入参数：input 输入
// 输出参数：output 输出
// 返回值：错误码
int func(const CObject& input, CObject& output);
```

内存管理

【规则8-1】在定义变量的同时初始化该变量；数组、动态分配内存亦应即赋初值

- 解决问题
  - 如果变量的引用处和其定义处相隔比较远，变量的初始化很容易被忘记。
  - 如果引用了未被初始化的变量，可能会导致程序错误。
  - 有助于减少隐患。

```
int width = 10;      // 定义并初始化width
char *p = NULL;    // 指针变量初始化为空，防止指向非法地址

int array[10];       // 定义数组
// 初始化数组为0
memset((void*)array, 0, sizeof(int)*10);

int* pArray = NULL;          // 定义数组指针
pArray = new int[10];        // 动态分配内存
// 初始化数组为0
memset((void*)pArray , 0, sizeof(int)*10);
```

内存管理

【规则8-2】用free或delete释放了内存之后，立即将指针设置为NULL，防止产生“野指针”

- 说明
  - 内存释放以后已经被回收而不可访问
  - 内存可能被重新分配给其他对象，可导致严重的数据错误

```
CRect* pRect = new CRect;
...
delete pRect;
pRect = NULL;     // 内存释放后置为NULL
```

学习建议

- 熟练掌握《部门C++编码规范》
  - 15条规则，5条建议
- 提升学习《Q/JC GL 0310-2020 C&C++编码规范》
  - 100条规则，5条建议

谢谢大家的学习！

提问交流...