C++语言编码规范

Programming
Your Future

东软IT人才实训中心
hubr@neusoft.com

目的与目标

- 规范部门内软件开发和设计风格，保证所有开发人员写出风格一致的代码
- 增强代码的健壮性、可读性和可维护性
- 通过人为以及自动的方式对最终软件应用质量标准
- 减少程序的潜在错误

时间：4 学时

教学方法：讲授PPT

+案例分析

课程概述

- 1. 工程的目录结构
- 2. 文件
- 3. 程序格式—排版、注释
- 4. 类
- 5. 变量
- 6. 常量
- 7. 数据类型
- 8. 宏及预处理
- 9. 表达式以及控制语句
- 10. 其他
- 11. 内存管理
- 12. 调试与维护

1 工程的目录结构

- 一个工程应该划分为若干个目录，目录应该是模块划分的体现。如：

  Project →ModuleName1
  →ModuleName2
  → ……
  →ModuleNamen
  → Include

- 其中include目录下应该放置公共的头文件。
- 目录名称要有意义；

2 文件

- 2.1 文件的结构
- 文件的种类有两种：源文件（*.cpp）和头文件（*.h）。
- 源文件中一般应该包括的内容：
  - 本文件需要包含的头文件；
  - 只在本文件内部使用的（对外部隐藏的）类型；
  - 只在本文件内部使用的（对外部隐藏的）常量；
  - 只在本文件内部使用的（对外部隐藏的）宏定义；
  - 只在本文件内部使用的函数原型说明；
  - 只在本文件内部使用的（对外部隐藏的）变量的声明；
  - 函数的代码实现。

2.1 文件的结构

- 头文件中应该包含的内容
  - 提供给外部参照的类型；
  - 提供给外部参照宏定义；
  - 提供给外部参照（全局）函数原型说明；

- 文件划分的准则
  - 高内聚；
  - 低耦合；

2.2 文件的命名及文件的长度

- 使用英语命名，尽量选用可以发音和有意义的名字。
- 源文件和头文件名称应该与此源文件的主要函数名称相同（相近）或类名相同。
- 一个文件的长度不要超过2000行代码。

2.3 源文件

- 一般说来，源文件的整体构成包括以下内容（有时可能不存在某些部分）。具体情况请参见源文件模板。
  - 源文件的注释框
  - 包含文件部分
  - 宏定义部分
  - 类型定义部分
  - 结构体定义部分
  - 全局变量定义部分
  - 文件static变量定义部分
  - 函数原型声明
  - 调试开关定义部分
  - 函数定义部分

2.4 头文件

- 一般说来，头文件的整体构成包括以下内容（有时可能不存在某些部分）。具体情况请参见头文件模板。
  - 头文件的注释框
  - 包含文件部分
  - 宏定义部分
  - 类型定义部分
  - 结构体/类定义部分
  - 函数原型声明

2.4 头文件(续)

- 头文件应采用#ifndef/#define/#endif的方式来防止多次被包含，其中使用的宏名为：“__”+“文件名”+“_”+“扩展名”，文件名与扩展名均为大写。
- 对头文件的包含应该采用相对路径，而不采用绝对路径。
- 为了减少执行程序的大小，不应包含不需要的头文件。
- 头文件中不要定义变量。

3 程序格式—排版、注释

- 3.1 排版
- 3.1.1 分行
  - 较长的语句、表达式等要分成多行书写，一行程序不要超过81列；

如：

```
// Calculate the interval timeout for the read.
if (timeoutsForIrp.ReadIntervalTimeout &&
    (timeoutsForIrp.ReadIntervalTimeout !=
MAXULONG))
{  //Process
    }
```

3.1.1 分行

- 不允许把多个短语句写在一行中，即一行只写一条语句。
- 若函数的参数较长，则要进行适当的划分。划分的原则：整齐、美观、符合其他编码规范。

  如：

```
// set local read event
KeSetEvent(
                &SerialGlobals.Serial_Data[ fdoData->InID ].CanReadEventTimeout,
                IO_NO_INCREMENT,
                FALSE
                );
```

3.1.1 分行(续)

- 在两个函数定义之间必须使用空行分开。
- 在函数定义中，用空行将代码按逻辑片断划分。例如两个并列的判断语句之间就可以用空行来区分；变量的定义、变量的初始化以及真正的处理语句之间就可以用空行来区分。
- 意义不一致的宏定义之间、全局变量之间、函数声明之间需要添加空行。
- 如果函数参数不只有一个，则每个参数的定义各占一行，每行前有一个制表符的缩进。
- 最后一个参数的定义后要紧跟右括号。
- 定义函数体的左右大括号要各自独占一行。

3.1.2 空格

- 单目操作符后不要加空格。
- 单目操作符前不要加空格。
- 赋值符号前后要有一个空格。
- 位逻辑运算符号后的ASCII码应该有一个空格。 *如：y = z | 0x0f;*
- 在条件表达式与括号间应该有一个空格。
- 在问号表达式（?）前后应该有一个空格。
- 在二元逻辑操作符前后应该有一个空格。
- 在比较操作符号前后应该有一个空格。
- 在“.”、“->”前后不应该有空格。
- 在数组名与“[”之间不应该有空格。
- “,”之后应该有空格。
- “；”之后应该有空格。
- “sizeof”与“（”之间不应该有空格；“（”与类型之间不应该有空格。

3.1.3 缩进

- 代码缩进要使用制表符，而不是直接使用空格。
- 对嵌套语句使用一个制表符----Tab（4个字符）的缩进。
- 尽可能保证缩进嵌套的层数不超过四层。

3.2 注释

- 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。
- 注释写在说明代码的上面。
- 注释的目的是解释代码的目的、功能和采用的方法，提供代码以外的信息，帮助读者理解代码，防止没必要的重复注释信息。
- 注释与前面的执行语句之间空一行。
- 源文件的开头要有注释，但每个业务/项目由于客户的要求不同而有不同的注释格式，这里对注释格式不作统一要求。
- 注释不要有嵌套。
- 应对不易理解的分枝条件表达式加注释。
- 不易理解的循环, 应说明出口条件。

4 类

4.1 关于类名

- 头文件为XXXX.h，源文件为XXXX.cpp，则类名必须为CXXXX。

4.2 声明顺序

- 无论成员函数还是成员变量，均按public、protected、private的顺序排列。
- 成员函数按照以下顺序进行排列：
  - 构造函数、析构函数
  - 其他函数
- 同类型的函数、变量在一起声明。
- 先声明成员函数，再声明成员变量。
- 定义与声明的顺序一致。

4.3 成员函数(1)

- 析构函数一定要为虚函数。
- 不改变成员变量的成员函数需要定义成const。
- 不要重新定义父类的非虚函数。
- 函数参数传递采用引用方式，而不采用值方式。
- 不要返回可写的成员变量的指针或者引用。
- 避免将成员函数的定义放在类的定义中。
- 一定要显示声明函数的返回值类型。
- 除非必要，不要使用函数重载。
- 不要重载或者隐藏非虚函数。

4.3 成员函数(2)

- 除了构造函数、析构函数外，其他函数必须都有函数说明。
- 说明中必须包含如下内容：
  - 名称(Function Name)：函数的名称。
  - 描述(Description)：函数功能的简单描述。
  - 作成日期(Created)：格式为YY/MM/DD的日期，如03/09/28。
  - 参数(Parameter)：函数参数的说明。
    - 参数说明的顺序依次为参数名称、参数类型、输入参数还是输出参数。
    - 如果函数没有任何参数，本项填写“VOID”。
  - 返回值类型(Return Code)：函数返回值的类型。
    - 如果函数不需要返回值，本项填写“VOID”。
  - 作者(Author)：函数的作者。

4.3 成员函数(3)

- 函数的声明要与函数定义一致（包括函数的类型及参数的类型、个数、顺序）。
- 注意控制参数的数量，一般来说不要超过7个，当参数过多时，应该考虑将参数定义为一个结构体，并且将结构体指针作为参数。
- 函数的大小不要过长，一般定为60行以内（除去注释，空行，变量定义，调试开关等）。
- 一个函数仅完成一件功能。
- 防止将函数的参数作为工作变量。
- 编写可重入函数时，若使用全局变量，则应通过信号量保护。

4.3 成员函数(4)

- 对于可能在中断处理程序中访问的全局变量，应该用关中断的方式进行保护。
- 回避函数的递归调用。
- 调用没有参数的函数时、不要忘了加括弧“( )”。
- 不提倡使用友元(friend)函数。
- 对函数尽可能使用Const。

4.4 成员变量

- 成员变量名前一定要加前缀m_。
- 成员变量命名规则参照本规范中变量命名规则的说明。
- 静态成员变量的访问使用“::”而不使用“.”或者“->”。
- 成员变量的属性应该是Private。
- Const成员函数不要返回非Const的Handle(Pointer)。

```
class A {
public:
int* foo() const {
return m_pa; }
private:
int* m_pa;  };
```

```
void bar( )  {
const A a;
int* pa = a.foo();
*pa = 10; // modifies private data in
a!  };
```

- 同时，任何成员函数不要返回非Const的Handle(Pointer)。

5 变量

- 5.1变量命名
  - 标识符应该直观且可以拼读，可望文生义。
  - 程序中不要出现仅靠大小写区分的相似的标识符。
  - 要防止局部变量与全局变量同名。

5.2 变量定义与初始化

- 一行只作一个变量定义。但是象坐标x、y、z，纬度、经度等对同时进行定义有非常重要意义的在一行可进行多个变量定义。
- 严禁读取未经初始化的变量。
- 不要定义过大的局部变量，以免堆栈溢出，可以定义成static全局变量或从内存池中分配。
- Class类型的变量的初始化采用直接初始化，而不采用Copy初始化。

  例：

  String a1 = "Hello"; ***// avoid***

  String b1 = String("Hello"); ***// avoid***

  String c1("Hello"); ***// prefer***

5.3 变量的类型转换

- 不提倡变量间的类型转换。
- 如需要类型转换，请采用显形方式进行。
- 建议不要直接比较BOOL类型变量。
- 不要将有符号数强制转化为无符号数。
- 不要将浮点型指针强制转化为整型指针。

5.4 变量使用的注意事项

- 不要返回局部变量的地址。
- 用unsigned定义不可能为负数的变量。
- 变量的使用目的只能有一个，禁止以多个目的使用变量。
- 函数中避免存在不使用的变量。

6 常量

- 对于long、unsigned int、unsigned long类型的常量，使用L、U、UL等后缀（注意是大写字符）。

  例：

```
const unsigned int a = 0U;
const unsigned int b = 0u;
const unsigned int c = 0;
const long d = 0L;
const long e = 0l;
const long f = 0;
const unsigned long g = 0UL;
const unsigned long h = 0Ul;
const unsigned long i = 0;
```

6 常量(续)

- 对float、long double类型的变量，使用F、L的前缀（注意是大写字符）。

  例：

  const float PI = 3.1415F;

  const long double R = 0.003;

  const long double A = 0.0L;

  const long double Z = 0.0l;

7 数据类型

7.1 指针

- 当对指针进行“++”、“--”与“+=”、“-=”操作时，要注意:
  - （1）指针地址的改变是以所指向数据类型的大小为单位的。
  - （2）时刻留心指针所指向的地址是否越出所允许的范围（即是否越界）。
- pointer的间接参照的嵌套不要作成2层以上。

7.2 数组

- 字符数组的定义和初始化要考虑‘\0’。
  如错误的例子：
  char hello[5] = "Hello";

- 二维数组或多维数组的初始化不要遗漏{和}。
  如错误的例子：
  int x[2][2] = { 1, 2, 3, 4 };
  如正确的例子：
  int y[2][2] = { {1, 2}, {3, 4} };

7.2 数组(续)

- 结构体数组的每个元素初始化不要遗漏{和}。
- 对数组元素的初始化不要多写逗号“，”。
  如错误的例子：
  char ChArray[ ] = {‘A’, ‘B’, ‘C’, ‘D’, ‘E’,};
  如正确的例子：
  char ChArray[ ] = {‘A’, ‘B’, ‘C’, ‘D’, ‘E’};
- 编码时进行数组下标索引的操作前，要思考是否有溢出可能。

7.3 自定义数据类型－结构体、联合体、枚举

- 多个源文件使用的用户自定义类型要在头文件内定义。
- 在一个源文件内使用的用户自定义类型在源文件内相应的节中定义。
- 应尽量避免使用memcmp进行结构体/联合体的比较。

7.3.1 结构体

- 结构体名格式为tagXXXX_t。
- 结构体别名格式为XXXX_t。
- 成员变量名首字母要大写，前面不要加类型标志。示例：

```
typedef struct tagSearchListParam_t{
        LONG  ListKind;
        CHAR  PreCode[4];
} SearchListParam_t;
```

- 结构体成员的个数最大不超过20个。
- 结构体的嵌套深度的最大不要超过为3。
- 合理排列结构中成员的顺序，可节省空间并增加可理解性。结构中的各成员的顺序不同，结构的长度也不同。建议结构体中的成员按照由小到大的顺序排列。

7.3.2 联合

- 联合名格式为tagXXXX_u。
- 联合别名格式为XXXX_u。
- 成员变量名首字母要大写，前面不要加类型标志。示例：

```
typedef tagSearchListRecAttrb_u{
                    SearchListRecAttrbGenre_t          Genre;
                    SearchListRecAttrbGenreR_t         GenreR;
                    SearchListRecAttrbAddressR_t       AdrsR;
                    SearchListRecAttrbCity_t           City;
} SearchListRecAttrb_u;
```

7.3.3 枚举

- 枚举名格式为tagXXXXEnum
- 枚举别名格式为XXXXEnum
- 成员名只能由大写字母、数字与下划线构成。示例：

```
typedef enum tagOpenMapFormModeEnum{
    OPENMAP_FORM_MAPFULL,
    OPENMAP_FORM_MAPMENU
} OpenMapFormModeEnum;
```

- 枚举类型的定义不要多写逗号“，”。
- 枚举类型成员数值的给定方式应该只指定第一个成员，或指定全部成员；不要只指定部分成员。

8 宏及预处理

- **8.1** 宏的命名
  - 宏名只能由大写字母、数字与下划线构成。
- **8.2** 宏定义的位置
  - 多个源文件使用的宏要在头文件内定义。
  - 在一个源文件内使用宏源文件内相应的节中定义。

8.3 宏定义以及使用

- 除非意义明确，代码中不要直接使用数值和字符串，而要使用宏。访问数组下标除外。
- 不要将宏NULL使用在指针以外的场合。
- 宏值一定要用小括号括起来。→对宏值的操作
- C++语言中不提倡使用宏函数。
- 使用宏时，宏的参数要括起来。
- 使用宏时，不允许参数发生变化。

8.3 宏定义以及使用(例)

如下用法可能导致错误。

```
#define  SQUARE( a )  ( ( a ) * ( a ) )
int a = 5 ;
int b ;
b = SQUARE( a++ ) ;  // 结果：a = 7，即执行了两次增1。
```

正确的用法是：

```
b = SQUARE( a ) ;
a ++ ;              // 结果：a = 6，即执行了一次增1。
```

使用宏时，宏的参数要括起来。

8.4 预处理

- 所有的预处理器命令从第一列开始写。但是在“#if”的嵌套内部的预处理器命令语句缩进。
- 用<>来包含系统或标准库文件，对于其他文件的包含使用“”。
- 对包含的头文件，请不要使用绝对路径。

9 表达式以及控制语句

- 9.1一般规定
- 在控制条件语句内不使用复杂的函数，要使用复杂函数返回的值作为控制条件使用时，另外定义表示函数的返回值的变量，并将其变量用于控制条件语句内。
- 在控制条件语句内不进行复杂的算术运算。可以另外定义变量并且将算术运算的结果赋值给这个变量，将此变量作为判断；
- 控制条件语句中不写使用超过2个的括号才可以正确表达的复杂的条件。在复杂的条件判断中，定义布尔变量，并在条件语句中使用；
- 当条件中包含数值范围的限制时，数值以从左至右往大的顺序写。
- 用“==”比较变量和常数是否相等时，把常数写在左边，以防止把“==”写成“=”。

9.2 运算符使用规则

- 对于复杂的运算，适当地加上括号来表示运算的优先顺序，即使是优先级高的运算加上扩号也没有坏处。

9.3 控制语句

- 9.3.1　控制语句的嵌套
  - 建议控制语句的嵌套不得超过6级。
  - 控制语句嵌套时，新的语句前应加一行空行。

9.3.2 If语句

- If语句独占一行，执行语句不得紧跟其后。
- 无论有多少条执行语句，都必须用大括号将执行语句括起来。
- 使用else if语句的时候，最后的else节一定要记述，避免else与if的匹配错误。
- if语句中记述多个条件时，在逻辑演算符“&&”, “||”的两侧的条件是用括弧”(”、”)”括起来。但是，条件是单项的时候可以不必用括弧括起。
- 需要用浮点数比较的时候，不要通过等号“==”和不等号“!=”来进行比较。

9.3.3 switch语句

- switch语句独占一行。
- 每个case独占一行，执行语句不得紧跟其后。
- 若某个case不需要break要给出确认性注释。
- 一定要提供default。
- 真假两种处理采用if语句，不采用switch语句。
- case语句的排列顺序是、从以下这样的排列顺序中选择有效的。
  - (1). 按频度多少的顺序，将比较频繁的处理放在前面，这样可以提高执行的效率；
  - (2). 从通常的case到例外的case；
  - (3). 数值顺序或字母顺序。

9.3.4 for语句

- for语句独占一行，执行语句不得紧跟其后。
- 无论有多少条执行语句，都必须用大括号将执行语句括起来。
- 空的循环体要给出确认性注释。
- 若无特殊情况，循环计数要从0开始，不要从1开始。结束条件要用“<”，不要用“<=”。
- Loop记数器在Loop内不强制进行变更（即Loop记数器的数值只在增量表达式中修改）。
- 尽量避免在while/for循环体中申请或释放内存，以防止产生内存碎片。
- 在For循环中，禁止使用逗号演算子“,”。

9.3.5 while语句

- while语句独占一行，执行语句不得紧跟其后。
- 无论有多少条执行语句，都必须用大括号将执行语句括起来。
- 空的循环体要给出确认性注释。

9.3.6 return语句

- Return语句的返回值的类型和已声明的类型是同样的。
- void型的函数时、Return语句不应该有返回值。
- 建议一个函数一条返回语句。

- 9.3.7 其他语句
  - 禁止使用goto语句。

10 其他

- 10.1 关于位运算
  - 位运算降低代码的可读性，使用位运算时，写上详细注释。
  - 位演算不用带符号的(“signed”)整数来进行。

- 10.3 Miscellaneous
  - 所有标识符不超过31字符。
  - 不得遗留“永远不会用到”的代码。
  - 不允许存在空语句。
  - 避免在布尔表达式中使用赋值语句。

10.3 关于性能

- 不要在循环体内定义对象。
- 不要在循环体内申请内存。
- 使用 += ， -=，〉>>=, 等复合运算符，而不使用 A = A+1…等这样的操作（减少临时对象）
- 不应该在循环体执行的语句，不要放在循环体中执行。

  例：

```
for( i=0; i<1000; i++ ){
        GetLocalHostName( hostname );
                                ...
      }
```

- GetLocalHostName的意思是取得当前计算机名，在循环体中，它会被调用1000次啊。这是 多么的没有效率的事啊。应该把这个函数拿到循环体外，这样只调用一次，效率得到了很 大的提高。

11 内存管理

- 11.1 共通规则
  - C++语言中用new分配内存，用delete释放内存；
  - 不要用malloc、calloc、realloc分配内存，不要用free释放内存。
  - 分配内存后要立刻判断指针是否为NULL。
  - 原则上谁申请谁释放。
  - 指针指向的内存被释放后，该指针要立刻赋值为NULL。
  - 不要强制引用或指针指向尺寸不同的目标。

11.2 new和delete

- 数组的分配与释放要使用new[]和delete[]。
- new与delete要成对使用。
- New[]与delete[]要成对使用。
- delete完成之后，请将指向内存的指针置为0。
- 如果重载了New，那么一定要重载Delete。

12 调试与维护

- 12.1 调试
  - 增加调试语句应该放在调试开关内，如
  - ＃ifdef DEBUG_FUNC1 // Modified by XXX on 05-10-10
    调试代码1；
    ……
    ＃endif
  - 调试开关名称的形式应为DEBUG_XXXX, 开关的说明应该在文件头里记述。
  - 定义调试开关的位置应该是相关的头文件里（影响到多个CPP）或者CPP文件（影响到一个CPP）的前部。

12.2 维护

- 在代码正式Release后，再修改代码应该属于维护的代码，对于维护代码的处理请遵守以下规则：
- 对于修改Bug而改动代码，请对改动的代码进行如下处理：

```
#ifdef BUG_XXXX    //  Modified by XXX  on  05-10-10
    修改后代码；
    #else
    原来代码；
    #endif  //  BUG_XXXX
```

12.2 维护(续)

- 对于其他情况而改动代码，请对改动的代码进行如下处理：

```
#ifdef  OTHER_XXXX    //  Modified by XXX  on  05-10-10
    修改后代码；
    #else
    原来代码；
    #endif  //  OTHER _XXXX
```

- 维护标记名称的形式应为BUG_XXXX或OTHER _XXXX, 维护标记名称的说明应该在文件头里记述。
- 定义维护标记名称的位置应该是相关的头文件里（影响到多个CPP）或者CPP文件（影响到一个CPP）的前部。

Neusoft

Beyond Technology