IFR2019 控制组 编程规范 1.0

1 第一章 概述

1.1 术语定义

Pascal 大小写

将标识符的首字母和后面连接的每个单词的首字母都大写。可以对三字符或更多字符的标识符使用 Pascal 大小写。例如:

BackColor

Camel 大小写（骆驼式）

标识符的首字母小写，而每个后面连接的单词的首字母都大写。例如:

backColor

1.2 文件命名组织

1.2.1 文件命名

文件名全小写。例如 usart1_remote.c

1.2.2 文件注释

在每个文件头必须包含以下注释说明

```
/*-------------------------------------------------------*作者:
* 文件名:
*创建时间:
*最后修改时间:
* 文件功能描述:
-------------------------------------------------------*/
```

文件功能描述只需简述，具体详情在函数的注释中描述。

2 第二章 代码外观

2.1 列宽

代码列宽控制在 80 字符左右，方便阅读。

2.2 换行

当表达式超出或即将超出规定的列宽，遵循以下规则进行换行

1、在逗号后换行。

2、 在操作符前换行。

3、规则 1 优先于规则 2。

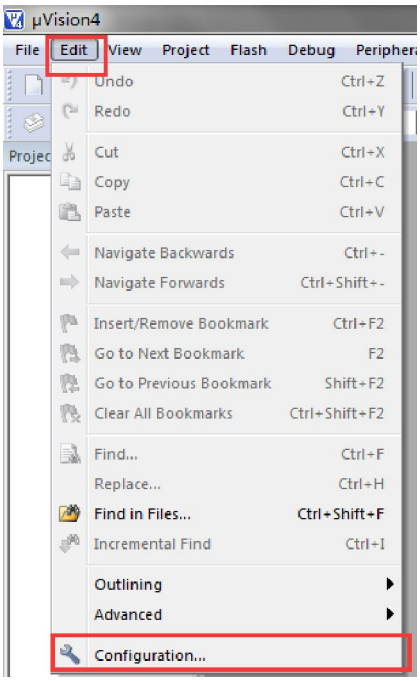
当以上规则会导致代码混乱的时候自己采取更灵活的换行规则。

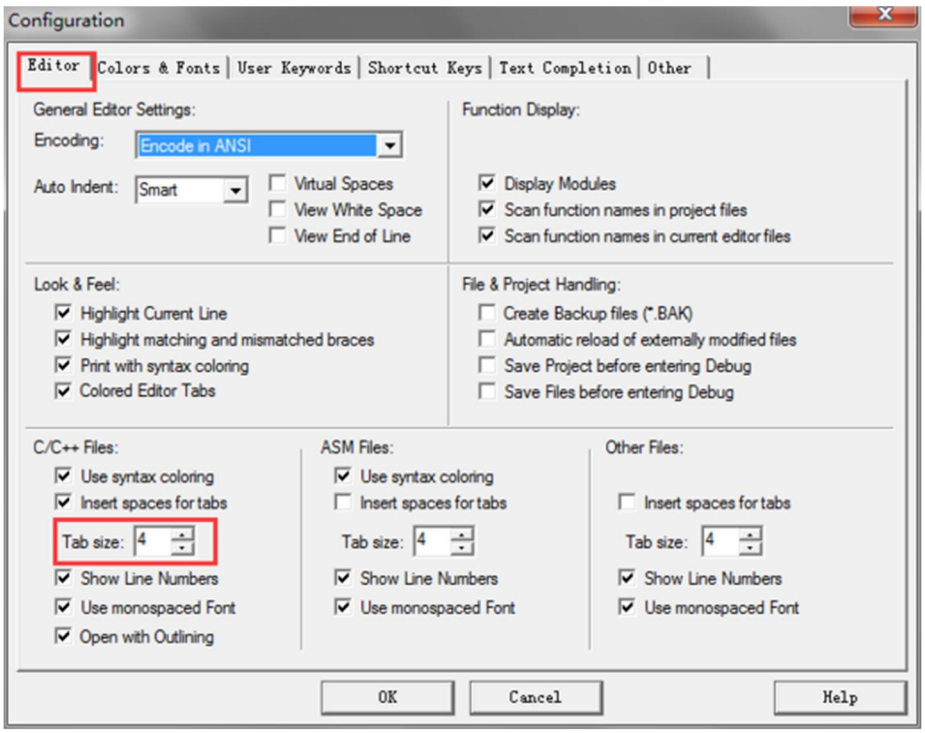
2.3 缩进

缩进应该是每行一个 Tab(4 个空格)，不要在代码中使用 Tab 字符。

把使用的文本编辑器的 Tab 调成 4 个空格，这样按一下 tap 就会自动缩进 4 个字符

设置教程:





2.4 空格

在以下情况中要使用到空格

1.除了 . 之外，所有的二元操作符都应用空格与它们的操作数隔开。一元操作符、++及--与操作 数间不需要空格。如

```
a += c + d;
a = (a + b) / (c * d);
while (d++ = s++)
{
    n++;
}
PrintSize("size is " + size + "\n");
```

2.5 花括号 - {}

1、 左花括号“{”放于关键字或方法名的下一行并与之对齐。如

```
if (condition)
```

```
{
}
public int Add(int x, int y)
{
}
```

2、 左花括号“{”要与相应的右花括号“}”对齐。

3、 左花括号“{”单独成行，不与任何语句并列一行。

4、 if、while、do 语句后一定要使用{}，即使{}号中为空或只有一条语句。

如

```
if (somevalue == 1)
{
    omevalue = 2;
}
```

5、 右花括号“}”后建议加一个注释以便于方便的找到与之相应的 {。如

```
while (1)
{
    if (valid)
    {
    } // if valid
    else
    {
    } // not valid
} // end forever
```

3 第三章 程序注释

3.1 注释概述

1、修改代码时，总是使代码周围的注释保持最新。

2、在每个例程的开始，提供标准的注释样本以指示例程的用途、假设和限制很有帮助。注释样本应该是解释它为什么存在和可以做什么的简短介绍.

3、避免在代码行的末尾添加注释，而是在行前用注释进行注释，如:

```
/*下面的语句有什么用呢*/
A=a++;
```

在批注变量声明时，行尾注释是合适的；在这种情况下，将所有行尾注释在公共制表位处对齐。

4 、避免杂乱的注释，如一整行星号。而是应该使用空白将注释同代码分开。

5 、在编写注释时使用完整的句子。注释应该阐明代码，而不应该增加多义性。

6 、在编写代码时就注释，因为以后很可能没有时间这样做。

10 、避免多余的或不适当的注释，如幽默的不主要的备注。

12、 注释代码中不十分明显的任何内容。

13 、为了防止问题反复出现，对错误修复和解决方法代码总是使用注释，尤其是在团队环境中。

14 、对由循环和逻辑分支组成的代码使用注释。这些是帮助源代码读者的主要方面。

15 、在整个应用程序中，使用具有一致的标点和结构的统一样式来构造注释。

16 、用空白将注释同注释分隔符分开。在没有颜色提示的情况下查看注释时，这样做会使注释很明显且容易被找到。

17 、为了是层次清晰，在闭合的右花括号后注释该闭合所对应的起点。

```
namespace Langchao.Procument.Web
{
} // namespace Langchao.Procument.Web
```

3.2 单行注释

该类注释用于

1 方法内的代码注释。如变量的声明、代码或代码段的解释。注释

```
// 注释语句
```

private int number;

2 方法内变量的声明或花括号后的注释，注释示例:

if (1 == 1) // always true

{

statement;

} // always true

3.3 函数前的注释

*/***

** 可编辑文本框选项左移*

** 功能描述 ：按向左键时，可编辑文本框的编辑位向左移一位。*

** 输入参数 : pbox 指向可编辑文本框的指针*

** 返回参数 ：无*

** 作 者 : Liy-tj*

**测试通过时间： 12/7/2008*

***/*

*void EditTxtBoxLeftGUI(EDIT_TXT_BOX *pbox)*

{

… …

}

4 第四章 声明

4.1.1 每行声明数

一行只建议作一个声明:如

int level; //推荐

int size; //推荐

int x, y; //不推荐

4.1.2 初始化

在变量声明时就对其做初始化。如 *int a=0;*

4.1.3 位置

变量建议置于块的开始处，不要总是在第一次使用它们的地方做声明。

如

```
void MyMethod()
{
    int int1 = 0; // beginning of method block
    if (condition)
    {
        int int2 = 0; // beginning of "if" block
        ...
    }
}
```

避免不同层次间的变量重名，这样会导致很难发现的错误：

```
int count;
...
void MyMethod()
{
    if (condition)
    {
        int count = 0; // 避免
        ...
    }
    ...
}
```

5 第五章 命名规范▲

5.1 命名概述

名称应该说明“什么”而不是“如何”。可以使用 GetNextStudent()，而不

是 GetNextArrayElement()。

命名原则是:

以下几点是推荐的命名方法。

1、避免容易被主观解释的难懂的名称，如方面名 AnalyzeThis()，或者属性名 xxK8。这样的名称会导致多义性。

3、只要合适，在变量名的末尾加计算限定符（Avg、Sum、Min、Max、Index）以此来表示该值的意义。

4、在变量名中使用互补对，如 min/max、begin/end 和 open/close。

5、布尔变量名应该包含 Is，这意味着 Yes/No 或 True/False 值，如 fileIsFound。

6、在命名状态变量时，避免使用诸如 Flag 的术语。状态变量不同于布尔变量的地方是它可以具有两个以上的可能值。不是使用 documentFlag，而是使用更具描述性的名称，如 documentFormatType 或者 XxxState,如 WorkState.

7、即使对于可能仅出现在几个代码行中的生存期很短的变量，仍然使用有意义的名称。仅对于短循环索引使用单字母变量名，如 i 或 j。 可能的情况下，尽量不要使用原义数字或原义字符串，如 For i = 1 To 7。而是使用命名常数，如 For i = 1 To NUM_DAYS_IN_WEEK 以便于维护和理解。

8、在为宏定义命名的时候，都使用大写字母，并以下划线分隔每个单词，最后应相关模块名结束

如:

```
#define CLS_CMD_LCD 0x03
```

我们可以知道是 LCD 的“清除”命令。

5.2 大小写规则

下表汇总了大写规则，并提供了不同类型的标识符的示例。

标识符	大小写	示例
结构体类型	Pascal+Typedef	OneStructTypedef
枚举类型	Pascal+ Typedef	ErrorLevelTypedef
结构体	Pascal	OneStruct

枚举	Pascal	OneEnum
全局变量	Pascal	RedValue
函数	Pascal	WriteSrtSpi
局部变量	Camel	backColor
宏定义	全部大写	CLS_CMD_LCD
枚举内的值	全部大写	NORMAL_STATE

5.3 缩写

为了避免混淆和保证跨语言交互操作，请遵循有关区缩写的使用的下列规则:

1 不要使用计算机领域中未被普遍接受的缩写，如遇到要大家讨论后使用。

2 在适当的时候，使用众所周知的缩写替换冗长的词组名称。例如，用 UI 作为 User Interface 缩写，用 OLAP 作为 On-line Analytical Processing 的缩写。

3 在使用缩写时，对于超过两个字符长度的缩写请使用 Pascal 大小写或 Camel 大小写。例如，使用 HtmlButton 或 HTMLButton。

5.4 结构体

1、使用 Pascal 大小写。

2、用名词或名词短语命名类。

3、使用全称避免缩写，除非缩写已是一种公认的约定，如 URL、HTML

5.5 结构体成员

以下规则概述字段的命名指南:

1、使用 Camel 大小写。

2、拼写出成员中使用的所有单词。仅在开发人员一般都能理解时使用缩写。

名称不要使用大写字母。下面是正确命名的字段的示例。

```
Struct SampleClass
{
    char *url;
    char *destinationUrl;
}
```

5.6 枚举 (ENUM)

1 对于 Enum 类型和值名称使用全大写。因为枚举通常用来作为宏定义使用

2 少用缩写。

如：

```
/***********************--工作状态--**********************/
typedef enum
{
    PREPARE_STATE,      //上电后初始化状态 1s 钟左右
    CHECK_STATE,        //自检状态 在此阶段能够通过内核重启解决的问题发生时将会自动内核重启
    CALI_STATE,         //校准状态标定
    LOST_STATE,         //硬件帧率丢失
    NORMAL_STATE,       //正常输入状态 即底盘跟随云台模式
    WAIST_STATE,        //扭腰模式
    STOP_STATE,         //停止运动状态(由遥控器控制的保护状态)
    TAKEBULLET_STATE,   //取弹状态
    PROTECT_STATE,      //保护状态，在程序循环时执行，当检测到程序发生异常或者传感器异常时进入该模式
    ERROR_STATE,        //硬件错误状态，包括但不限于检测出传感器失效，数据失常等重启无法解决的问题
}WorkStateTypedef;
```

5.7 函数

以下规则概述方法的命名指南：

1 使用 Pascal 大小写。

2 函数的命名以描述性的动词开头，并在最后以引函数所操作的模块结束。如：

WriteByteUart0(char wrtByte)

以 Write 开头，以操作的模块 Uart0 结束。这样既不会冲突，也容易理解。

3 较短的函数可以不加下划线，名字较长的函数可以加下划线分割单词

函数名能表示这个函数干了什么，如

CAN2_Shoot_Bullet_SendMsg(s16 motor201,s16 motor202);

5.8 常量 (CONST)

以下规则概述常量的命名指南：

所有单词大写，多个单词之间用 "_" 隔开。 如

```
const string PAGE_TITLE = "Welcome";
```

6 第六章 语句

6.1 每行一个语句

每行最多包含一个语句。如

```
a++; //推荐
b--; //推荐
a++; b--; //不推荐
```

6.2 复合语句

复合语句是指包含"父语句{子语句;子语句;}"的语句，使用复合语句应遵循以下几点

1 子语句要缩进。

2 左花括号“{” 在复合语句父语句的下一行并与之对齐，单独成行。

3 即使只有一条子语句要不要省略花括号“ {}”。 如

```
while (d += s++)
{
    n++;
}
return myDisk.size();
return (size ? size : defaultSize);
```

6.3 IF、 IF-ELSE、IF ELSE-IF 语句

if、 if-else、if else-if 语句使用格式

```
if (condition)
{
    statements;
}
if (condition)
{
    statements;
}
else
{
    statements;
}

if (condition)
{
    statements;
}
```

```
else if (condition)
{
    statements;
}
else
{
    statements;
}
```

6.4 FOR

for 语句使用格式

```
for (initialization; condition; update)
{
    statements;
}
```

空的 for 语句（所有的操作都在 initialization、condition 或 update 中实现）使用格式

```
for (initialization; condition; update); // update user id
```

注意 1 在循环过程中不要修改循环计数器

2 对每个空循环体给出确认性注释。

6.5 WHILE 语句

while 语句使用格式

```
while (condition)
{
    statements;
}
```

空的 while 语句使用格式

```
while (condition);
```

6.6 DO - WHILE 语句

do - while 语句使用格式

```
do
{
    statements;
} while (condition);
```

6.7 SWITCH - CASE 语句

switch - case 语句使用格式

```
switch (condition)
{
    case 1:
        statements;
    break;
    case 2:
        statements;
    break;
    default:
        statements;
    break;
}
```

注意：

1、语句 switch 中的每个 case 各占一行。

2、语句 switch 中的 case 按字母顺序排列。

3、为所有 switch 语句提供 default 分支。

4、所有的非空 case 语句必须用 break; 语句结束。

6.8 GOTO 语句

goto 语句使用格式

goto Label1:

statements;

Lable1:

statements;

6.9 表达式

1 避免在表达式中用赋值语句

3 避免对浮点类型做等于或不等于判断

7 第七章 补充

7.1 源程序文件组织方式

文件夹名:除库文件外全大写

KEIL: 编译环境

KEIL

STM32F4MCU：STM32 芯片相关文件夹

CORE:启动文件

FWLIB: 核心文件

inc: stm32 相关头文件

src: stm32 相关 C 文件

ROBOT：机器人相关文件夹。

APP：应用文件 chassis.c

BSP: 机器人初始化文件 led.c usart.c can.c 该处只放初始化和中断服务函数

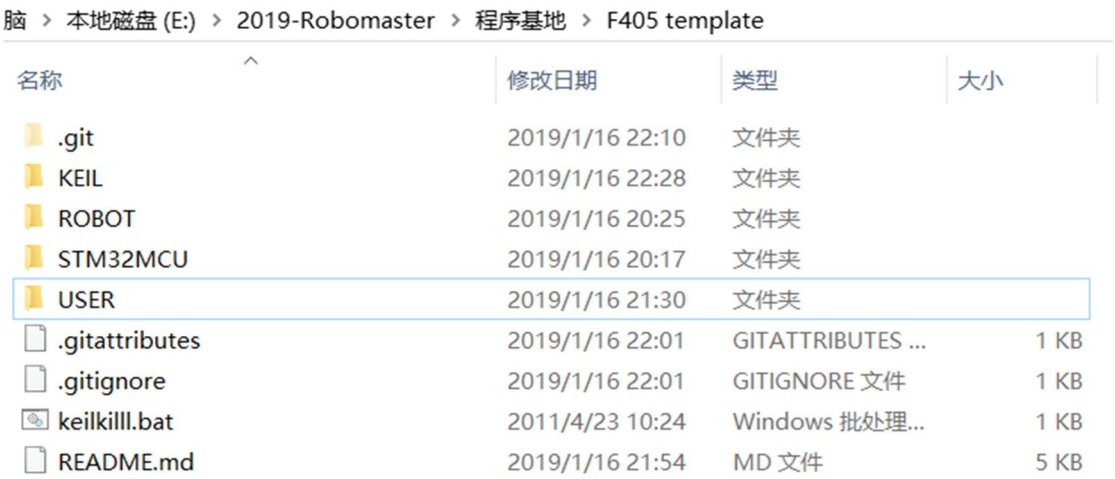
ANALYSIS:机器人底层数据解析文件

ALGO: 通用算法文件 filter.c pid_general.c

USER:主函数文件夹

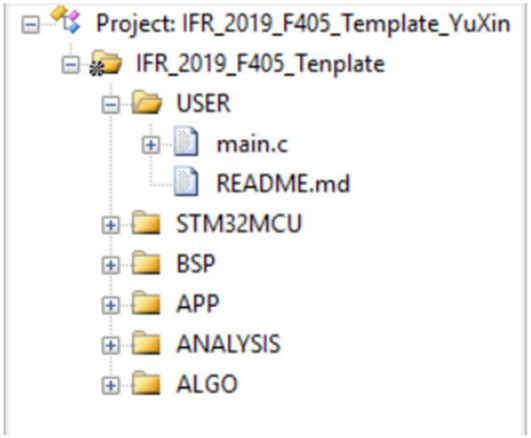
工程根目录下新建 redeme.md 文件用来对程序进行说明

示例:



7.2 工程内文件夹组织规范

示例:



详情参考: https://github.com/yx19981001/IFR2019-Progra_standard-F405_template

7.3 文件内部附加说明

1)所有.C 文件对应自己的头文件(#include)

如:chassis.c 只包含 chassis.h 文件

gpio.c 只包含 gpio.h

所有自己定义的.h 文件都只包含 main.h

main.h bsp.h

```
#ifndef __BSP_H__
#define __BSP_H__

#include "main.h"
#include "can1.h"
#include "can2.h"
#include "pwm.h"
#include "gpio.h"
#include "usart1_remote.h"
#include "heartbeat.h"

void BSP_Init(void);

#endif
```

main.h bsp.h

```
#ifndef __MAIN_H__
#define __MAIN_H__

#include "stm32f4xx.h"

#include "main.h"
#include  "bsp.h"
#include  "chassis.h"
#include "yun.h"
#include  "fire.h"

#include "delay.h"
#include "filter.h"
#include "pid.h"
#include "keyboard.h"
#include "gpioctrl.h"
#include "can1_analysis.h"
#include "can2_analysis.h"
#include "remote_analysis.h"

#include "mpu6050IIC.h"
#include "mpu6050_it.h"
#include "mpu6050_process.h"

#include <string.h>
#include <stdio.h>
#include <math.h>
```

2).h 文件中放对应文件的宏定义以及结构体类型声明和函数声明。

.c 文件中放变量；函数；结构体定义以及全局的变量和结构体等（extern）

3).h 文件中对于避免重复引用所使用的#ifndef 后面的标识符统一采用 __XXXX_H__的形式 XXXX 为文件名的全大写

如：__MAIN_H__

__CHASSIS_H__

4)注意注释的添加

对于函数注释：是什么？

```
/*************************************************
函数名称：Chassis_Move_Remote
函数功能：遥控器控制底盘移动
函数参数：无
函数返回值：无
描述：由遥控器通道1；2；3得到底盘目标移动数据
*************************************************/
void Chassis_Move_Remote(void)

{
```

对于算法的注释：为什么？