Clean Code C

Why CC

童子军规，破窗理论
读写代码的比例10：1

代码与绘画

相同点：

都能搞出让人看不懂的东西
都需要识别美的东西
就算能看懂美丑，仍然不会写(画)，需要大量练习

简单设计4原则

1. 通过所有用例
2. 尽可能消除重复
3. 提升表达力
4. 使用更少的元素

实用主义CC

格式

重点关注空行

1. 通过空行来区分不同的逻辑片段
2. 同一个功能的变量定义和代码紧凑排布，

3. 初始化可以确定的变量，没有必要先给个无效值

```
DWORD RnluModuleGetCoreAndTaskInfo(VOID)
{
 BYTE                   ucLoop      = 0;
 const CHAR            *pbyTaskName;
 TRnluData             *ptRnluData  = NULL;
 TRnluModuleConfigInfo *ptCfgInfo   = NULL;
 ptRnluData  = (TRnluData *)RnluGetTaskDataArea();
 pbyTaskName = RnluGetSelfTaskName();
 ptRnluData->tRnluModuleCfg.pucTaskName       = pbyTaskName;
 ptRnluData->tRnluModuleCfg.wTaskModuleBitmap = 0;
 for (ucLoop = 0; ucLoop < RNLU_TASK_MAX_MODULE_NUM; ucLoop++)
 {
     ptCfgInfo = &g_atRnluModuleLocalConfigInfo[ucLoop];
 }
 RNLU_InitConfig_STATS_INC(dwRnluModuleGetCoreAndTaskInfo_success);
 return RNLU_SUCC;
}
```

紧凑后

```
DWORD RnluModuleGetCoreAndTaskInfo(VOID)
{
 const CHAR *pbyTaskName = RnluGetSelfTaskName();
 TRnluData *ptRnluData  = (TRnluData *)RnluGetTaskDataArea();

 ptRnluData->tRnluModuleCfg.pucTaskName = pbyTaskName;
 ptRnluData->tRnluModuleCfg.wTaskModuleBitmap = 0;

 for (BYTE  ucLoop = 0; ucLoop < RNLU_TASK_MAX_MODULE_NUM; ucLoop++)
 {
     TRnluModuleConfigInfo *ptCfgInfo   = NULL;
     ptCfgInfo = &g_atRnluModuleLocalConfigInfo[ucLoop];
 }
 RNLU_InitConfig_STATS_INC(dwRnluModuleGetCoreAndTaskInfo_success);
 return RNLU_SUCC;
}
```

其他格式请善用工具

IDE-format， CppLint

命名

babyName? 最重要的是规则达成共识，最好形成文档wiki。具体命名规则参加编码规范手册

统一命名原则的好处

1. 看起来赏心悦目
2. 模仿起来得心应手

统一缩略语

团队确定认可的缩略语，并保留语义。pre、proc、fwd、cap、rls。

统一语义

团队确定采用明确的词对应场景(get|fetch）、（mananger|controler|driver）、（delete/free/release）

名副其实

- 避免魔幻数字(为了好搜)
- 避免无意义的命名（为了好认）
- 读的出来的名字(为了好交流)

```
getRecdrdTasByName();
```

注释

不必要的注释

1. 准备删除的代码，
2. 英文翻译，
3. 搞怪的注释

```
// fuck MS
```

函数设计

回归函数的本质

函数命名

尽量描述做什么事(sendPdu)而不是什么时候调用(procWhenAttach)

函数参数个数不易过多

超过三个参数的函数有可能隐藏的信息是，这个函数可能做了很多的事情

函数入参的const说明

入参用 const 修饰， 出参不用。安全且明确，不需要额外通过在名字中嵌入in/out来区分

函数返回值是否有必要

根据函数用途，设计返回值

出参还是使用返回值

能用返回值就不用出参

static 修饰内部函数

目的是对函数的使用情况加以限制，便于编译器优化

数据结构

概念提取

为什么要有结构体？结构体是对内存的抽象描述

```
void init()
{
    BYTE* mem = (BYTE*)getMem("Share");
    MemInfo* memInfo = (MemInfo*)(mem);
    memInfo->idle = TRUE;
    Switch* switch = (Switch*)(mem + sizeof(MemInfo));
    switch->isOn = TRUE;
}
```

提取概念后

```
typedef struct ShareMem
{
    MemInfo memInfo;
    Switch switch;
}ShareMem;
void init()
{
    ShareMem* mem = (ShareMem*)getMem("Share");
    mem->switch.isOn = TRUE;
    mem->memInfo.idle = TRUE;
}
```

赋予意义

```
WORD32 cfgBrsTunnel(CfgInfo*);
WORD32 calcBuffSize(const BuffInfo*);
```

```
typedef WORD32 Status ;
typedef WORD32 BuffLen ;
Status cfgBrsTunnel(CfgInfo*);
BuffLen calcBuffSize(const BuffInfo*);
```

物理设计

为什么要有头文件？

头文件自满足

不自满足产生的问题

1. 单独包含编译不过
2. 不同包含顺序，可能产生二义性错误

头文件最小依赖

头文件依赖重产生的影响

1. 所有包含本头文件的c文件编译时间均变长，
2. 头文件修改波及重新编译范围变大，
3. 只包含接口声明

CC训练

代码知识卡片

类别：物理设计，函数设计，命名，数据结构，格式，注释(类别后续可以补充)
每个类别中需要考虑的点形成卡片，贴于墙上，供查阅和学习分享，早晨代码走查阶段抽空分享。

CC训练营

根据项目情况，周六(建议)，选取OJ平台上的一个题目，前期选取3-4张已分享的卡片，作为考察点。完成题目后，集体走查分享。