1.

PTZ 机械镜头 （机械镜头可能有用到光学镜头）

EPTZ 电子镜头

电子镜头对图像缩放会影响像素，而光学镜头不会

mtk芯片 (联发科芯片)

淡入淡出，一张图片变淡，一张图片变深

4就是for

2就是to

vl\_cam\_resultQ 文件下有很多关键操作

YLakaze （AKAZE局部匹配）

astBox4Body检测出来的躯干

AIA是 鑫哥那边的，我们要把AIA的信息发给鑫哥

vdo\_eptz\_interface.cpp下的东西已经不用了，用的是vl\_pri.hpp的函数（API）

VDO\_RECT2\_S stRectBgMax; //电子镜头摄像头最大视野

VDO\_RECT2\_S stRectInner;//依据每个人脸区域求最上最下最左最右的那个框

VDO\_RECT2\_S stRectAllView;//, 每个人脸区域经过扩展求最上最下最左最右的那个框

VDO\_RECT2\_S stFinalView//感觉是摄像头最后的视野

后缀有4MS就是给多流用的东西

#define OSA\_align(value, align) ((( (value) + ( (align) - 1 ) ) / (align) ) \* (align) )

UInt32 uiStride = OSA\_align(u16Width , 16);//经过数学关系将参1向上取整，调整为16的整数倍

UInt32 uiHeight = OSA\_align(u16Height, 4); //经过数学关系将参1向上取整调整为4的整数倍

cam\_resultQ类下的私有变量Bool m\_bAudioSendMsg//为true，则是VT模式让它调整，如果为false，则是AF模式让他调整

VDOD\_JUDGE\_S m\_stJudgeR;//存着实时AF的框选信息

VDOD\_JUDGE\_S m\_stJudgeV; //存着AF的框选信息

VDOD\_JUDGE\_S m\_stJudgeA; //存着VT模式的框选信息

VCS\_SENSOR\_IMX383 //机械镜头

VCS\_SENSOR\_IMX274 //电子镜头

stElemFoc.aflFaceAngle [uiBodyId] //音频角度

stElemFoc.aflFaceAngleL[uiBodyId] //音频角度

stElemFoc.aflFaceAngleR[uiBodyId] //音频角度

保留区感觉就是防止有移动的人在框的边缘一直走动而使框选框不稳定

sys\_get\_userenv （env环境）

git clone … //就是先从远程克隆下了整个仓库，后面的一切基础都是这个

git fetch //更新提交记录

git lg --all//查看所有提交记录

git checkout … //切换到指定分支下（只要切换了本地的代码就改变了）

Beyond Compare 4 红色就是不同文件，紫色代表有新增文件

PTZ\_WITH\_AUDIO\_E = 0, //VT模式使用电子镜头

PTZ\_WITH\_AUDIO\_M, //VT模式使用机械镜头

PTZ\_WITH\_VIDEO\_E, //AF模式使用电子镜头

PTZ\_WITH\_VIDEO\_M //AF模式是哦用机械镜头

宽中紧，人头两旁区域预留大小有（宽/中/紧）三种状态，宽就是人头两旁预留区域多

套接字除了可以实现⽹络间不同主机间的通信外，还可以实现同⼀主机的不同进程间的通信。我们知道两个进程如果需要进行通讯最基本的一个前提是能够使用唯一的标识来标志一个进程，在本地进程通讯中我们可以使用PID来唯一标识一个进程，但PID只在本地唯一，所以就可以利用ip地址+协议+端口号来唯一标识网路中的一个进程

分辨率和屏幕尺寸的关系：

分辨率3840\*2160，宽度有3840个像素，高度有2160个像素，这些像素均匀的分布在屏幕上。我们得到的x,y坐标其实也就是对应到在哪个像素点上）

躯干图像分辨率：1024\*576

人头图像分辨率 1920\*1080

VDO\_IVE\_IMAGE\_S 结构体变量代表的就是一张图像，结构体里有一些指针用来指向图像的地址还有一些图像宽度之类东西。

CC\_BUFFER\_ELEM\_S结构体里有三个VDO\_IVE\_IMAGE\_S 结构体变量指针（有存灰度图像的指针、RGB格式的躯干图像、RGB格式人头图像）还有图像的一些时间（u64TimeStamp）、序号之类的东西

Alg是算法用的，其分辨率大小是1024\*576

IWB 大的平板电脑

Rect 矩形框

zoom缩放大小

CC感觉就是一个集合的意思

一帧里面可能有有一个或一个以上的人，每个人占容器里一个元素std::vector<CC\_RECT\_SLICE\_S> vt\_stSlice; /\*\* 存放独区 \*\*/

vt 语音追踪

<YLprincipalPtsRefine.h> 感觉是镜头设置，误差调整的一些参数

<YLprincipalPtsRefine4Iwb.h>

VDO\_TRACK\_PARAM 视频追踪参数的一个结构体

VDO\_CAMERA\_ATTR\_S 感觉是设置摄像头的各种参数（电子镜头、机械镜头）

重区：两帧重复

静区：很多帧重复

独区没有重复

CC\_BBOXES\_S结构体感觉是一些框设置大小的东西

<utils/Log.h> 安卓日志打印的东西

3840,2160是4K

1280,720是720P

854,480是480P

char acParamPernFile[] = "/phone/config/a.yaml"; //躯干模型参数

char acParamHeadFile[] = "/phone/config/b.yaml";//人头模型参数

char acParamPrecFile[] = "/phone/config/c.yaml";//物体模型参数

char acModlePernFile[] = "/phone/config/a.dlc"; //躯干模型

char acModleHeadFile[] = "/phone/config/b.dlc";//人头模型

char acModlePrecFile[] = "/phone/config/c.dlc";//物体模型

物体模型就是防止误检的，比如说检测到一个躯干的置信度太低，就会送进物体检测，检测看它是不是物体，如果是物体，那它就不是人，不是物体就是人

USE\_DETECT\_ONLY = 0, /\* only detect alg \*/只检测

USE\_DETECT\_AND\_TRACK = 1, /\* detect and track alg \*/检测加追踪

USE\_DETECT\_AND\_LUMA\_STAT = 2, /\* detect and luma alg \*/ 检测并且加美颜的

UInt64 u64CreateTs; /\*\* 人体出现时间 \*\*/

VDO\_RECT2\_S stRgnBody; /\*\* 人体区域 \*\*/

VDO\_RECT2\_S stRgnFace; /\*\* 人体区域对应的人脸或人头区域 \*\*/

VDO\_RECT2\_S stRgnFaceMin;

UInt32 uiBodyId ; /\*\* body索引 \*\*/

UInt32 uiFrameId; /\*\* frame索引 \*\*/

Flt32 flScore;

Flt32 flAngle; /\*\* 人相对音频中心点的角度 \*\*/

Flt32 flAngleL; /\*\* 人相对音频中心点的角度 \*\*/

Flt32 flAngleR; /\*\* 人相对音频中心点的角度 \*\*/

Flt32 flVideoAngle; /\*\* 相对于定焦镜头的视频角 \*\*/

Flt32 flDistance; /\*\* 视频预测距离 \*\*/

Flt32 flHypotenuse; /\*\* 视频位置相对mic距离 \*\*/

Bool bBodyFaceMatchFlag; /\*\* 人体匹配人脸是否成功标志位 \*\*/

编译在对应文件夹下输入 mm

\_win不用了  
win32 win64啥的已经不用了

nce：算法模型的一系列推理平台，通过神经网络啥的，丢进去一张图片就会返回一个框

class cam\_bufferQ:public vdo\_elemQ<CC\_BUFFER\_ELEM\_S>

vdo\_elemQ类里用到了模板，<>里传的就是指定模板的类型为CC\_BUFFER\_ELEM\_S。

[dst](https://www.baidu.com/s?wd=dst&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1Y1ujRvm1mdmWw-rjIhPhmd0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3En164P1nLn1c)是destination的缩写，表目的  
[src](https://www.baidu.com/s?wd=src&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1Y1ujRvm1mdmWw-rjIhPhmd0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3En164P1nLn1c)是source的缩写，表源

2.

Windows DLL编程（DLL动态链接技术）

\_\_declspec(dllexport)导出到dll

\_\_declspec(dllimport)从dll导入

fscanf 函数原型为 int fscanf(FILE \* stream, const char \* format, [argument...]); 其功能为根据数据格式(format)，从输入流(stream)中读入数据，存储到argument中。

%hd是一个短整数

%ld是一个长整数

%x的意思是以十六进制数形式输出整数

#define QUOTE(name) #name #name的作用就是把#后面的看成[字符串](https://so.csdn.net/so/search?q=%E5%AD%97%E7%AC%A6%E4%B8%B2&spm=1001.2101.3001.7020" \t "_blank)

atoi(s)函数用于把一个[字符串](https://so.csdn.net/so/search?q=%E5%AD%97%E7%AC%A6%E4%B8%B2&spm=1001.2101.3001.7020" \t "_blank)转换成一个整型数据，

例如：b = aito（“12345.67”） 现在就等于12345

property\_get("vendor.camera.calibration.mode", szProp, "0");

每个属性都有一个名称和值，他们都是字符串格式。属性被大量使用在Android系统中，用来记录系统设置或进程之间的信息交换。属性是在整个系统中全局可见的。每个进程可以get/set属性。

DOA波达方向定位技术

DOA，是电子、通信、雷达、声呐等研究领域的行业内用语，通过处理接收到的回波信号，获取目标的距离信息和方位信息。

min\_element(begin, end);

参数

begin : 序列起始地址（迭代器）

end : 序列结束地址（迭代器）

返回值 : 序列中最小元素地址（迭代器）

max\_element(begin, end);

参数

begin : 序列起始地址（迭代器）

end : 序列结束地址（迭代器）

返回值 : 序列中最大元素地址（迭代器）

distance( begin, end)

参数

begin : 序列起始地址（迭代器）

end : 序列结束地址（迭代器）

返回值 :返回两个迭代器范围内包含的元素个数

advance (InputIterator& it, Distance n)

参数

it：迭代器

n：为一个常数

函数作用：用于将迭代器前进（或者后退）指定长度的距离

accumulate(vec.begin() , vec.end() , 42);

accumulate带有三个形参：头两个形参指定要累加的元素范围，第三个形参则是累加的初值。

Int8 szProp[PROPERTY\_VALUE\_MAX];

memset(szProp, 0, sizeof(szProp));

property\_get("iwb.removestill", szProp, "0");

Int32 iIsOpenRemoveStill = atoi(szProp);

iwb.removestill这相当于一个全局变量，调试的时候可以手动设置这个全局变量，"0"表示默认设置这个全局变量是0，

[\_\_DATE\_\_](https://links.jianshu.com/go?to=https%3A%2F%2Fwww.codersrc.com%2Farchives%2F9308.html) 当前日期，一个以 “MMM DD YYYY” 格式表示的字符串常量。

\_\_TIME\_\_ 当前时间，一个以 “HH:MM:SS” 格式表示的字符串常量。

[\_\_FILE\_\_](https://links.jianshu.com/go?to=https%3A%2F%2Fwww.codersrc.com%2Farchives%2F9281.html) 这会包含当前文件名，一个[字符串](https://links.jianshu.com/go?to=https%3A%2F%2Fwww.codersrc.com%2Farchives%2F7815.html" \t "_blank)常量。

[\_\_LINE\_\_](https://links.jianshu.com/go?to=https%3A%2F%2Fwww.codersrc.com%2Farchives%2F9270.html) 这会包含当前行号，一个[十进制](https://links.jianshu.com/go?to=https%3A%2F%2Fwww.codersrc.com%2Farchives%2F8488.html" \t "_blank)常量。

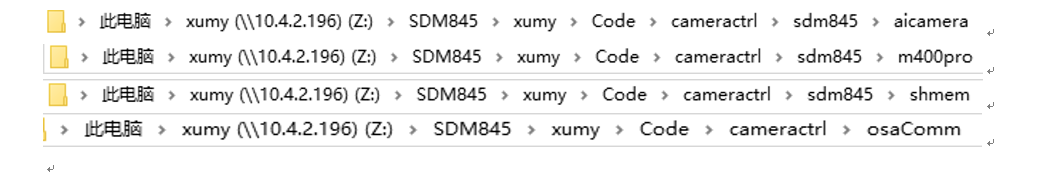
memcpy是内存拷贝，什么数据都可以

强转变量时，定义变量要考虑对应

比如：int \*retval;

(void \*\*)&retval //这边就是一个二级指针转二级指针

要编译的文件



setprop 变量名 值

虚函数是C++实现动态多态的一种方式，可以使得父类指针指向子类对象，调用虚函数时是调用子类的同名成员函数。

ADB（全称：Android Debug Bridge），ADB是android sdk里的一个工具, 用这个工具可以直接操作管理android模拟器或者真实的android设备。Android的初衷是用adb这样的一个工具来协助开发人员在开发android应用的过程中更快更好的调试apk，因此adb具有安装卸载apk、拷贝推送文件、查看设备硬件信息、查看应用程序占用资源、在设备执行shell命令等功能。

sprintf函数的格式：int sprintf( char \*[buffer](https://so.csdn.net/so/search?q=buffer&spm=1001.2101.3001.7020), const char \*format [, argument,...] );

除了前两个参数固定外，可选参数可以是任意个。buffer是[字符数组](https://so.csdn.net/so/search?q=%E5%AD%97%E7%AC%A6%E6%95%B0%E7%BB%84&spm=1001.2101.3001.7020" \t "_blank)名；format是格式化字符串

sprintf函数的功能与printf函数的功能基本一样，只是它把结果输出到指定的[字符串](https://so.csdn.net/so/search?q=%E5%AD%97%E7%AC%A6%E4%B8%B2&spm=1001.2101.3001.7020" \t "_blank)中了

fprintf函数的格式：int fprintf( FILE \*stream, const char \*format, [ argument ]…)

将结果输出到对应的流文件中。

实时线程有静态优先级的概念，里面拥有1-99个静态优先级，数字越到优先级越高

非实时线程没有静态优先级的概念

线程中支持三种调度策略：

抢占式调度策略，在同一静态优先级的情况下，抢占调度策略的线程一旦运行到便会一直抢占CPU资源，而其他同一 一直等到这个抢占式调度策略的线程退出才能被运行到（非实时线程会有一小部分资源分配到） SCHED\_FIFO

轮询式调度策略，在同一静态优先级的情况下，大家一起合理瓜分时间片，不会一直抢占CPU资源（非实时线程会有 分配到） SCHED\_\_RR

其他普通式调度策略，只能作用于非实时线程，由系统自动分配时间片，并且根据运行状态自动分配动态优先级 SCHED\_OTHER

static\_cast< 想要的类型> (原数据)， 是C++中的关键字。相当于C语言的强制类型转换

Wiki是一种在网络上开放且可供多人协同创作的[超文本系统](https://baike.baidu.com/item/%E8%B6%85%E6%96%87%E6%9C%AC%E7%B3%BB%E7%BB%9F/4145999" \t "_blank)

int pthread\_setaffinity\_np(pthread\_t thread, size\_t cpusetsize， const cpu\_set\_t \*cpuset); //设置线程在某个CPU核心上跑

int pthread\_getaffinity\_np(pthread\_t thread, size\_t cpusetsize, cpu\_set\_t \*cpuset);

//得到CPU核心上跑了那些线程

从函数名以及参数名都很明了，唯一需要点解释下的可能就是cpu\_set\_t这个结构体了。这个结构体的理解类似于select中的fd\_set，可以理解为cpu集，也是通过约定好的宏来进行清除、设置以及判断：

void CPU\_ZERO (cpu\_set\_t \*set); //初始化，设为空

void CPU\_SET (int cpu, cpu\_set\_t \*set); //将某个cpu加入cpu集中

void CPU\_CLR (int cpu, cpu\_set\_t \*set); //将某个cpu从cpu集中移出

int CPU\_ISSET (int cpu, const cpu\_set\_t \*set); //判断某个cpu是否已在cpu集中设置了

int prctl(int option, unsigned long arg2, unsigned long arg3, unsigned long arg4, unsigned long arg5);

prctl(PR\_SET\_NAME, “process\_name”)

第一个参数是操作类型，指定PR\_SET\_NAME，即设置进程名  
第二个参数是进程名字符串，长度至多16字节

IPC（Inter-[Process](https://baike.baidu.com/item/Process/1170280)[Communication](https://baike.baidu.com/item/%20Communication/20394231)，[进程间通信](https://baike.baidu.com/item/%E8%BF%9B%E7%A8%8B%E9%97%B4%E9%80%9A%E4%BF%A1/1235923" \t "_blank)

回调函数就是一个通过函数指针调用的函数。

纯虚函数是在声明虚函数时被“初始化”为0的函数。声明纯虚函数的一般形式是

virtual 函数类型 函数名 (参数表列) =0;

注意: ①纯虚函数没有函数体；②最后面的“=0”并不表示函数返回值为0，它只起形式上的作用，告诉编译系统“这是纯虚函数”; ③这是一个声明语句，最后应有分号。

纯虚函数只有函数的名字而不具备函数的功能，不能被调用。它只是通知编译系统: “在这里声明一个虚函数，留待派生类中定义”。在派生类中对此函数提供定义后，它才能具备函数的功能，可被调用。

在 C++ 中，每一个对象都能通过 this 指针来访问自己的地址，为 this 的目的总是指向“这个”对象，所以 this 是一个常量指针

this只能在成员函数中使用。

全局函数，静态函数都不能使用this。

实际上，成员函数默认第一个参数为T\*const register this。（隐藏的）

静态成员变量，它是所有对象所共享的，类内只是声明，类外实现定义。静态成员变量不属于对象，是属于类级别的。 当我们再次计算对象大小的时候，静态成员变量不纳入对象内存的。

注释一段代码

#if 0

#else 后面这段代码有效

#endif

#if 1 后面这段代码有效

#else

#endif

restrict关键字：用来限定指针变量，被该关键字限定的指针变量所指向的内存操作，必须由本指针完成

Csc是缩放的意思

union联合体

联合的成员是共用一块内存空间的，这样一个联合变量的大小，至少是最大成员的大小

睡眠和挂起是两种行为，阻塞则是一种状态。

操作系统中睡眠、阻塞、挂起的区别形象解释：  
        首先这些术语都是对于线程来说的。对线程的控制就好比你控制了一个雇工为你干活。你对雇工的控制是通过编程来实现的。  
        挂起线程的意思就是你对主动对雇工说：“你睡觉去吧，用着你的时候我主动去叫你，然后接着干活”。  
        使线程睡眠的意思就是你主动对雇工说：“你睡觉去吧，某时某刻过来报到，然后接着干活”。

        线程**阻塞**的意思就是，你突然发现，你的雇工不知道在什么时候没经过你允许，自己睡觉呢，但是你不能怪雇工，肯定你这个雇主没注意，本来你让雇工扫地，结果扫帚被偷了或被邻居家借去了，你又没让雇工继续干别的活，他就只好睡觉了。至于扫帚回来后，雇工会不会知道，会不会继续干活，你不用担心，雇工一旦发现扫帚回来了，他就会自己去干活的。因为雇工受过良好的培训。这个培训机构就是操作系统

git diff 1258e (需要对比的哈希值号)

fabsf 函数 返回浮点的绝对值

在计算联合体大小之前我们必须知道两个知识点：  
1.联合的大小至少是最大成员的大小  
2.当最大成员大小不是最大对齐数的整数倍的时候，就要对齐到最大对齐数的整数倍。

使用c语言编写的文件要用以下格式框起来

#ifdef \_\_cplusplus

extern "C"{

#endif

xxxxx //C语言的n个函数实现

#ifdef \_\_cplusplus

}

#endif

原因是在C++的环境中\_\_cplusplus这个宏定义会自动被定义，其作用就可以将此块被框起来的代码按C语言的方式去编译

派是弧度制,180度是角度制,派／180,表示每角度等于多少弧度.那30\*π/180表示30度的角等于π/6

malloc分配得到的地址也可以用下标来进行访问：

uint64\_t \*chunk = (uint64\_t \*)malloc(65536);

printf("chunk%u\n",chunk[8191]);

YUV是编译true-color颜色空间（color space）的种类，Y’UV, YUV, YCbCr，YPbPr等专有名词都可以称为YUV，彼此有重叠。“Y”表示明亮度（Luminance或Luma），也就是灰阶值，“U”和“V”表示的则是色度（Chrominance或Chroma），作用是描述影像色彩及饱和度，用于指定像素的颜色。

YUV 是一种彩色编码系统，主要用在视频、图形处理流水线中(pipeline)。相对于 RGB 颜色空间，设计 YUV 的目的就是为了编码、传输的方便，减少带宽占用和信息出错。

YUV 格式通常有两大类:打包(packed)格式和平面(planar)格式。前者将 YUV 分量存放在同一个数组中, 通常是几个相邻的像素组成一个宏像素(macro-pixel);而后者使用三个数组分开存放 YUV 三个分量,就像是一个三维平面一样。

一、YUV420、YUV420P和YUV420SP区别

YUV420： 即打包格式的YUV420

YUV420P: 即YUV420 Planar，Y\U\V数据是分开存放的

YUV420SP:即YUV420 semi planar, 这个格式的数据量跟YUV420 Planar的一样，但是U、V是交叉存放的.

NV12 属于 YUV420SP 格式。两个平面，分别存储 Y 分量 和 UV 分量。其中 UV 分量共用一个平面并且以 U, V, U, V 的顺序交错排列。每四个 Y 分量共享一组 UV 分量，

NV21 属于 YUV420SP，与 NV12 几乎一致，区别是 UV 平面中 U 与 V 的排列顺序颠倒，以 V, U, V, U 的顺序交错排列，

RGB565 是16位的，2个字节，5+6+5，第一字节的前5位是R，后三位+第二字节前三位是G，第二字节后5位是B。

RGB555 也是16位的，2个字节，RGB各5位，有1位未用。

RGB888 是24位的，3个字节。

U8C1 就是RGB的一个通道，灰度图

U8C3 就是RGB的三个通道

用于检测的神经网络算法是YOLO

git reset --hard 索引值

cin.clear(); /\*\* 清除输入错误标志 \*\*/比如要输入一个整数int型，但是输入了一个字符，这时候cin流中就会置一个错误标志，再输入其他数据都会受到影响，简单的理解，流错了，咋弄都白扯（典型的错误就是你输错之后，如果是循环输入的，后面都不提示你输入了，死循环）

cin.ignore(1024, '\n'); cin.ignore()函数中有两个参数，分别为数值型的a 和 字符型的 ch ，即cin.ignore( a, ch )。它表示从输入流 cin 中提取字符，提取的字符被忽略，不被使用。

文件操作整理：

FILE \* fopen ( const char \* filename, const char \* mode );

参数：

filename：文件名

mode：打开方式 “rb“打开二进制文件只读

函数作用：打开文件



int fclose ( FILE \* stream );

参数：

stream：文件名

函数作用：关闭文件

size\_t fread( void \*buffer, size\_t size, size\_t count, FILE \*stream );

参数：

buffer:存放读取文件内容的地址

size:读取数据块的字节数

count:要读取的数据块数

stream:文件指针

函数作用：读文件

size\_t fwrite( const void \*buffer, size\_t size, size\_t count, FILE \*stream );

参数：

buffer:存放需写入文件内容的地址

size:写入数据块的字节数

count:要写入的数据块数

stream:文件指针

int fseek( FILE \*stream, long offset, int origin );

参数：

stream：文件指针

offset：偏移量。（可正可负）

origin:从哪个位置开始计算偏移量。位置可取3种值：文件首部、当前位置和文件尾部，实际表示时分别对应值0、1、2，或常量SEEK\_SET、SEEK\_CUR、SEEK\_END

int fflush(FILE \*stream)

参数：

stream：文件指针

函数作用：可以实时将缓冲区中的数据写入磁盘中

不要在switch语句内定义变量 //提示错误jump to case label [-fpermissive]

FIXME：说明此标识处有错误，需要修正

视频编码技术：

H.264 = SVC（可伸缩部分） + AVC（不可伸缩部分）  
SVC 在视频编码过程中将视频分为多个层（核心层、中间层和扩展层）。上层依赖于下层。上层越高，视频越清晰，视频越低，视频越模糊。

在带宽较差的情况下，只能传输底层（核心层）。如果带宽足够，则可以将所有数据传输到终端。

string是对象，不是一段[内存](https://so.csdn.net/so/search?q=%E5%86%85%E5%AD%98&spm=1001.2101.3001.7020" \t "_blank)，不适合放在结构体中。

对于string、列表、容器是不能使用memset的

fread函数用于从文件流中读取数据，其函数原型为：

size\_t fread(void\* buffer, size\_t size, size\_t count, FILE\*stream);

【参数设置】

1) buffer为接收数据的地址，对于fread来书是要读出数据的地址，即数据保存的地址

2) size是要读出内容的单字节数。

3) count是要进行读出size字节的数据项的个数。

4) stream为目标文件指针。

H264 I帧 P帧 B帧

I帧：帧内编码帧，帧表示关键帧，你可以理解为这一帧画面的完整保留；解码时只需要本帧数据就可以完成（因为包含完整画面）

P帧：前向预测编码帧。P帧表示的是这一帧跟之前的一个关键帧（或P帧）的差别，解码时需要之前缓存的画面叠加上本帧定义的差别，生成最终画面。（也就是差别帧，P帧没有完整画面数据，只有与前一帧的画面差别的数据）

B帧：双向预测内插编码帧。B帧是双向差别帧，也就是B帧记录的是本帧与前后帧的差别（具体比较复杂，有4种情况，但我这样说简单些），换言之，要解码B帧。不仅要取得之前的缓存画面，还要解码之后的画面，通过前后画面的与本帧数据的叠加取得最终的画面。B帧压缩率高，但是解码时CPU会比较累。

字符串比较（char类型）

函数原型

extern int strcmp(const char \*s1, const char \*s2)

当s1 < s2 返回值：负数

当s1 = s2 返回值：0

当s1 > s2 返回值：正数

正负数在if语句中均为真

枚举类型的初始化：后一个会自动比前一个名称大1，也就是CC\_STRATEGY\_LEFT\_FIRST是1，CC\_STRATEGY\_RIGHT\_FIRST是2以此类推

typedef enum tagCcStrategeyEn /\* 左右先\*/

{

CC\_STRATEGY\_NONE = 0,

CC\_STRATEGY\_LEFT\_FIRST ,

CC\_STRATEGY\_RIGHT\_FIRST ,

CC\_STRATEGY\_MIDDLE\_FIRST ,

CC\_STRATEGY\_MAX

} CC\_STRATEGY\_E;

3.

<cutils/properties.h>有什么作用

MKIT是什么意思

Android Studio推荐使用CMakeLists.txt，对于已经使用Android.mk写的编译文件，转换为CMakeLists.txt即可。



getprop android.cc.track.speed //追踪速度

getprop android.cc.talk.speed //对讲追踪速度

写代码注意点：

1. 如果容器内存的数据类型是指针，其有在堆区分配空间，在析构函数除了要clear()还要清空指针指向的内存空间（使用delete）
2. 变量使用前要记得初始化
3. 进行逻辑/算术运算的时候，变量类型需要一致，例如： (rand() % 401 + 600) / 10.0f / accumulate(score.begin(), score.end(), 0.0f); / average = sum / (double)(score.size());