算法流程图

## 读入图片，得到图片基本信息：

二进制读入图片到数组。通过ppt上给出的一系列关键位置的offset得到图片的基本信息。最重要的是bitcount和图片宽度。

## 算出一行的真正占用的字节数：

用公式 hang=(bitcount\*宽度+31)/32\*4得到一行真正占得的字节数。

## 用循环去找到每个所需修改的像素的位置。

根据像素下表、图片高度和每一行的字节数计算出像素所在数组的偏移量。

## 利用k值更改像素的值。

个人rgb的修改方法是之间每个单项\*(1+k)，然后不超过255。

## 保存图片

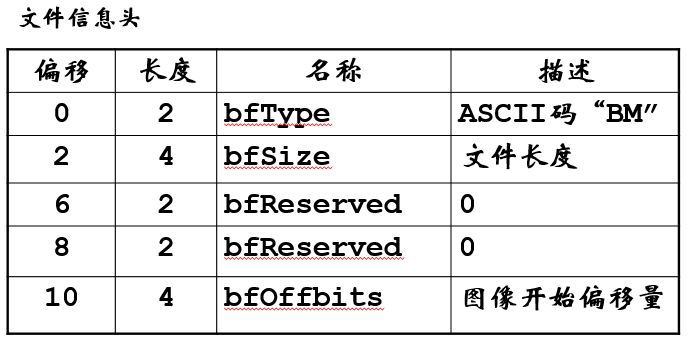
二进制输出到文件。

功能说明

使用了visual studio 2015 中的MFC图形化界面设计，而关于bmp图像的处理则专门集成到了mytran类（文件mytran.h）中。下面详细介绍一下这个类：

## 类的变量。

主要的变量有cnt、yu[maxn]、yu2[maxn]、bfoff、kuan、gao、bitcount、hang、bfsize和st。而常量有bfsize\_pos、bfoffbits\_pos、bisize\_pos、kuan\_pos、gao\_pos、bitcount\_pos和bmicolors。其中的常量都是bmp图片关键信息在信息头的位置偏移量。由ppt的这些表格得出：



每个变量的含义如下：

cnt：当前文件的总字节数，读入文件时用于计数。

yu[maxn]：按字节小端序储存当前的图片，由于bmp是小端序，所以每个字节内部是反着的。

yu2[maxn]:按字节小端序储存当前修改后的图片。

1. bfoff:这个是图片信息开始处的偏移量，用get函数和bfoffbits\_pos共同得到。 它的值=信息头大小+调色板大小。其中信息头大小固定54，调色板大小随bitcount变化。
2. kuan:这个是图片宽度，用get函数和kuan\_pos得到。
3. gao:这个是图片的高度，用get函数和gao\_pos得到。
4. bitcount:这个是bitcount，即每一个像素为所占的字，用get函数和bitcount\_pos得到，其中我只考虑了最简单的情况即bitcount=24的情况。
5. hang:这个是每一行真正所占的字节数。利用bitcount和kuan结合公式得到。
6. bfsize:这个是文件的大小，用get函数和bfsize\_pos得到。它的值应该和cnt一致。
7. st:这个是图像信息开始的offset，就是bfoff。

## 类的主要函数

### void init()

初始化函数，在读取完文件后，用get函数去更新能直接得到的变量。并且用公式hang=(bitcount\*kuan+31)/32\*4算出变量hang。要求bitcount=24，即只考虑最简单的bmp图，rbg表示的那种。

### long long get(long long st, long long si)

用于从图像的二进制数据中按字节读取。就是读取yu数组中第st位到st+si-1位的数据，并且将原本的小端序转换成为正常的顺序。最后如果si>1，则合并到一个long long 中返回。如果si=1，就直接返回。

### void change(long long st,long long si, long long tmp)

用于将二进制数据写入到新图像中。将正常序的tmp写入到yu2数组中的第st位到st+si-1位中。会把tmp按字节分开，对应字节中用小端序写入。

### void open(CString tpath)

用于打开图像文件，从tpath读取bmp文件到yu数组，同时同步到yu2数组。利用了cnt来记序。由于限制了数组大小maxn的大小为1200000（可以更大），所以过大的图片文件会弹出提示框。

### bool save(CString tpath, int len)

用于储存文件，将yu2数组中的0到len-1的数据写入到tpath文件中。写回成功返回true，如果遇到并未打开文件、文件写回错误的情况，会返回false。

### void trans\_3\_3(float k)

这次实验的核心函数，k是调亮的系数，是亮度增大k倍的意思。

其中要确定更改的对象是右上角，对应的图片矩阵的下标(j,i)的范围就是：

kuan/2<j<kuan , 0<i<gao/2。（bmp的坐标系从左下角(0,0)开始）。

对于每一个下标(j,i)要通过，off=st+i\*hang+j\*3来得到改像素在yu数组中的起始下标。然后off+0为该像素第一个字节的下标，表示的是蓝色，off+1表示该像素第二个的字节，表示的是绿色，off+3表示该像素第三个字节的下标，为红色。

通过get(off,1) 就能得到该(j,i)像素的b值，get(off+1,1)就能得到该像素的g值。

然后将r、g、b值都乘以系数(1+k)来表示新的rgb值，并且通过cahnge函数写入到yu2的对应位置上。change(off,1,23)就表示将(j,i)像素点的b值变为了23。

## 实验反思

在预设yu数组大小的时候，发现过大的数组(>50000)不能存入，会报stack over flaw 的错误。后来将yu数组变成static的静态数组就可以设置得很大了。

亮度的调整，本来想先把rgb转换成yuv，然后通过调整亮度系数y来进行亮度调整，但是效果太差（基本全白），所以改回了现在的做法，效果还不错。

对于其他bitcount的亮度调整的思考：想到如果是带调色板的bmp文件，会有一点难以解决，就是在更改1/4的图像亮度时难以兼顾其他3/4的拥有相同像素值的像素。就是说如果改了1/4图像中的红色的亮度，那么不能保证其他3/4的图像中的红色不变。解决这个问题涉及到对调色板的调整。一个思路是放弃调色板，将所有bmp图像全变成bitcount=24的图片，然后同一解决。我难以解决这个问题，于是放弃了带调色板的bmp图像的亮度调整。

另外MFC窗口化界面，大多参考了一些教程和之前学习课程的内容。