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。

## 用统计的办法找出图片的直方图，并用均衡化的算法去找到每种灰度对应改变成的新灰度。

统计出每种灰度出现的次数，用均衡化的算法：每1/256份就划分一次（大概），然后再按顺序对应灰度（tran数组）。注意保持递增和间距。具体算法参考ppt:dip\_3\_histogram slide 45。

## 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。利用上一问得到的tran数组改变每个像素的值。

## 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：

由于承接了上次用到的一些变量和类，下面先介绍第二次实验相较第一次实验新增的变量和函数：

## 类的新增变量。

变量有int histogram[256+5]、float bi[256+5]。

higtogram就是直方图，higtogram[x]=y指拥有第x灰度的像素出现了y次。

bi是直方图比率，即bi[x]=，就是ppt中提到的。

## 类的新增函数。

1. void init\_2()

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=8，即只考虑最简单的bmp灰度图，调色版满足b=g=r=offset的那种。同时用统计的办法，更新histogram数组得到直方图。然后计算出bi数组。

1. void trans\_2()

这次实验的核心函数。

主要核心思想是构造一个mytran[]数组，并且利用该数组来改变原图的像素。

mytran[]数组是用来记录原灰度变成的新灰度的值，mytran[x]=y表示原灰度x变成新灰度y。

mytran数组的求解是根据ppt dip\_3\_histogram slide 45 提炼得到的：

从小到大枚举旧灰度值i，同时：

维护一个bi数组的前缀和（变量now），和一个当前欲选择的新灰度（变量pre），一个已经使用的新灰度（变量uesd），一个当前暂存旧灰度的起始值（变量myst）。

其中now的更新就是now+=bi[i]。

pre的更新要满足pre/256 < now 而且pre要尽可能地大。

如果pre更新后不是used（没有被使用），说明当前的灰度前缀已经累积到一个可以被设置一个新灰度的程度了，于是就更新myst到i的旧灰度的mytran数组为pre就可以了。此时需要更新myst=i+1（myst表示的是当前还未分配的第一个旧灰度，现在旧灰度已经分配到i了，所以myst更新为下一个:i+1），和used=pre（used表示已经被使用的新灰度的最小值，当前pre灰度被使用了，并且pre总是下一个被检测的新灰度，所以更新used=pre）。

如果pre更新后是used（已被使用），说明当前的灰度前缀累积不够，需要继续累积。此时就继续累积就行了（下一个i）。

对于初始化的讨论：now=0 （因为前缀和的初始化一般为0），used=-1（0就是第一个新灰度了，而最开始第一个新灰度也没被使用），pre=0（从第一个新灰度开始检验），myst=0（从第一个旧灰度开始分配）。

对于变量和ppt表格中变量的对应关系：now对应ppt中的，pre对应，used、myst是两个状态记录变量（虽然表格中未给出，但是推导时会用到）。

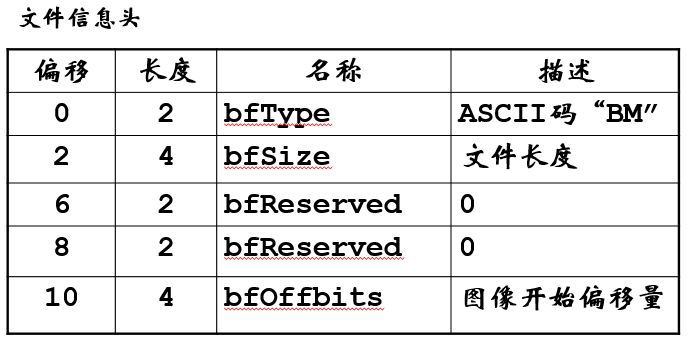
这样就得到了tran数组。

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。通过get(off,1) 就能得到该(j,i)像素的旧灰度值。

然后将旧灰度值用tran数组转换得到新灰度。再利用change函数修改灰度。change(off,1,23)就表示将(j,i)像素点的灰度值变为了23。

## 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)

第一次实验的核心函数，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

只考虑了bitcount为8的直方图均衡化，但是对于bitcount为4的应该也能实现功能，就是不知道会有什么其他问题出现了。

利用监视来找bug，特别是对于用工程来写程序的时候尤为重要。

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。