从 BMP 到 TIFF

开始前你应该知道的

以往做实验，许多同学不知道如何下手。在这种不确定中挣扎一段时间之后，猛然发现 deadline 到了，便急急忙忙地上网找一段不知能否编译通过的代码，或者将同学写好的代码（或许也是在网上找的！）匆匆忙忙地修改一番，直接交了上去。这次实验，你将会经历前所未有的透彻感受。按照本文认真做完实验，你会发现格式转换实际上比想象中的要困难许多。即便是知道原理的同学，也可能没有考虑过实现的时候有多少细节需要处理。当然，这次实验你需要花的时间可能较长，如果要跟着本文做下去，请一定要有耐心。希望这本手册能让读者不仅对实验本身的内容有透彻的认识，我也希望它能对你以后如何完成实验起到些微启发作用。

本次实验要做什么

将一副 BMP 格式的图片转换为 TIFF 格式的图片。要做出的程序应该这样工作：用户输入 BMP 图片的路径并指定输出路径后，程序应该能输出一张 TIFF 图片到指定的位置。

本次实验怎么做

我们按照自底向上的方式，从基本概念入手，逐步清晰概念。主要步骤如下：

1. 了解文件
2. 了解图片
3. 学习 BMP 格式
4. 学习 TIFF 格式
5. 学习 C++的文件操作

本次实验如何下手

从分别弄清楚两种文件的细节开始。

文件是什么

文件是一个大家每天使用电脑都要接触的概念。最常见的文件（同时也是本次实验涉及的一种文件）就是二进制文件（binary files）。它被称作二进制文件，意味着这些文件的实质实际上就是一长串二进制数字。这些二进制数字本身没有任何意义，我们拿着一长串 0 和 1 也不知道从哪里断句。文件之所以有意义，之所以能够用特定的程序打开，就是因为这些程序采用了一定的规则来读取这些文件。这里的规则也就是解释文件的方法。这种规则通常指定了一串给定的二进制数中，开头若干个位为一段，以及它们代表的意思；往

后数一定的位数，又是什么意思……一句话概括起来，也就是：文件的实质是二进制数字串，我们可以按照一定的规则去读取它。

为了让读者对文件有一个直观的认识，我们一起来模拟一下文件这个东西。我们要模拟的是一个存储成绩单的文件。事实上，这种文件在有了现代数据库以后看起来非常落后。不过在此我们仅仅需要模拟文件的概念，使用一下这种古老的技术也是合理的。下面我们就开始对这种成绩单文件进行模拟。

想想，我们的成绩单一般包括学生姓名、学号、年级、班级和打印时间，然后就是成绩单的主体部分了。我们按照这个顺序将这些信息存储到文件里面。到目前为止，我们对于文件的概念就是下图这个样子：

姓名	学号	年级	班级	时间	主体

你没看错，文件就是这个样子的，非常简单的线性结构（现在只需理解到这个地步）。不过我们觉得好像这个文件的最后部分——主体，看起来太微不足道了。照常理来说，主体应该是占成绩单的大部分体积的。为了让我们的成绩单不会只能记一个连属于哪个科目都不知道的成绩，我们得细想一下主体部分怎么细化。很简单，将它扩充成下图就可以了

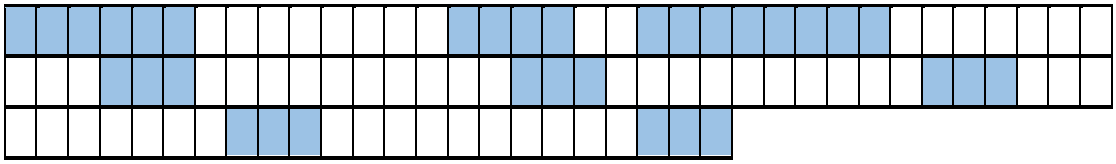
姓名	学号	年级	班级	时间	科目	成绩

但是这样的缺点就是只能记录一科成绩，这显然不是我们想要的。为此，我们将上面这“条”文件加长，将其最后两个部分多复制几个：

姓名	学号	年级	班级	时间	科目 1	成绩 1
科目 2	成绩 2	科目 3	成绩 3	科目 4	成绩 4	科目 5
成绩 5						

看，我们的文件开始变长了。要注意的是这并不是一个二维表，而是一个从左到右从上到下的线性一维表。再次提醒大家这一点，文件是个一维的串，而不是二维的表。

我们进一步细化这个文件，以求将这个文件模拟得更接近计算机上的文件。现在，这个文件中的每一段都还只是一个整体，但实际上肯定不是一个整体。比如学号，肯定是由一串数字组成的，那学号这一段就需要一定数量的位来存储。不只是学号，姓名中的字符在转为编码后，也应当是一串数组。此时，我们就理所应当地为上图中的每一段都分配一定数量的位，来保存信息。这里我们假设姓名在转为 ASCII 后不超过 6 个十进制数位（这里举十进制的例子是为了不让进制转换分散读者的注意力，因为这一节讨论的主要目的在于说明文件的概念），学号不超过 8 个数字，年级是 4 个数字，班级是 2 个数字，时间为 8 个数字，科目在转成 ASCII 后不超过 10 个数字，成绩不超过 3 个数字。那么到目前为止，我们的表就要发生很大的变化了，应该如下所示：



请不要被它吓到了，我解释一下，图中的每一次颜色变化（蓝到白或者白到蓝）都对应上一张图中的段的变化。即这张图的第一块 6 位灰色区域代表上一张图中的姓名段，接下来的 8 位白色区域就是学号。接下来的灰色区域即年级……

为了更清楚的说明文件的结构，以及更逼真地模拟文件概念，我们为上图的每一个数字位都从左到右，从上到下编序号，编完如下：

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
■	■	■	■	■	■									■	■
16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
■	■			■	■	■	■	■	■	■	■				
32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
						■	■	■							
48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
			■	■	■										
64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79
■	■	■											■	■	■
80	81	82	83	84	85	86	87	88	89	90	91	92			
										■	■	■			

上图中各个数位代表的内容如下表所示：

位编号	大小	内容
0 到 5	5 位	姓名的编码
6 到 13	8 位	学号
14 到 17	4 位	年级
18 到 19	2 位	班级
10 到 27	8 位	时间
28 到 37	10 位	课程名 1
38 到 40	3 位	成绩 1
41 到 50	10 位	课程名 2
51 到 53	3 位	成绩 2
54 到 63	10 位	课程名 3
64 到 66	3 位	成绩 3
67 到 76	10 位	课程名 4
77 到 79	3 位	成绩 4
80 到 89	10 位	课程名 5
90 到 92	3 位	成绩 5

好了，我们用于存储成绩单的文件大致已经完成了，下面我们放入一些测试数据试试。我们首先用传统成绩单的方式列出测试数据：

姓名	Tom	学号	20121993	年级	2012 级
班级	1 班	时间	2012 年 9 月 10 日		
Math			90		
Eng			100		

Bio	70
Music	85
Lab	95

现在我们要用刚才我们亲手炮制的文件格式来将这个成绩单存储起来。但是现在最明显的问题是，我们的文件里只能存储数字。那么我们首先应该将表中的非数字信息转化为数字。第一，姓名是字母。我们必须将其采用某种编码方式转换为数字。这里我们使用ASCII 举例子。“Tom”这三个字母的大写形式（我们仅采用大写形式，因为大写字母的ASCII 编码是两位十进制数，小写的是三位十进制数。采用大写可以节省空间）转换成ASCII 码为 84 79 77 这三个数。

接下来学号本来就是数字，直接存入即可。年级中的“级”字没有包含任何必要信息，没有必要把它存进文件。我们只需存储“2012”这个四位十进制数就可以了。同样，班级信息我们也只需存储数字。但是要注意，文件的每个段的大小规定的是最大长度，存入信息如果没有这么多位，则需要用 0 补齐。也就是说，“1 班”这个信息存入文件后应该是“01”。同样，去掉时间中的“年”、“月”，再将“9”补齐为“09”后，存入文件的应该是“20120910”。

主体部分中的课程名也需要转成 ACSII 码。转换并补齐后的结果如下表：

课程名	ASCII 码
MATH	00 77 65 84 72
ENG	00 00 69 78 71
BIO	00 00 66 73 79
MUSIC	77 85 83 73 67
LAB	00 00 76 65 66

最后，再考虑成绩，它们全是数字，直接补齐后存储就行了。于是我们就得到了下面的文件：

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
8	4	7	9	7	7	2	0	1	2	1	9	9	3	2	0
16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
1	2	0	1	2	0	1	2	0	9	1	0	0	0	7	7
32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
6	5	8	4	7	2	0	9	0	0	0	0	0	6	9	7
48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
8	7	1	1	0	0	0	0	0	0	6	6	7	3	7	9
64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79
0	7	0	7	7	8	5	8	3	7	3	6	7	0	8	5
80	81	82	83	84	85	86	87	88	89	90	91	92			
0	0	0	0	7	6	6	5	6	6	0	9	5			

到此，我想也不用多对这个最终形成的文件作出解释了。看到这里有些迷茫的读者可以往前面的步骤追溯，直到你能看懂的地方，然后再往下读，也许就能找回刚才遗漏的信息。

文件的读取

那么如果我拥有一个文件，我怎么读取里面的信息呢？显然，我们应当先对文件有详尽的了解。以上一节中我们亲手创造的文件为例，任何人如果想要读取这个文件，就必须知道这一串数字中的哪一段代表什么意思。这也就是前面提到过的“规则”，只要详细了解这个“规则”我们才可能从文件读出正确的信息。现在假设一种情况，即还存在另一种成绩单文件的存储格式，这个格式中名字的最大位数是 7 位。那么如果我们以这种和我们创造的文件不同的规则来读取上一节的成绩单，那我们会得到完全错位的结果。

所以我们在读取文件之前必须详细了解这种文件的存储方式，哪一段代表什么意思。这个例子中，我们已经对上一节的成绩单文件格式了如指掌了。那么下面我们就开始读取这个文件。

首先，拿到一个成绩单文件之后，打开它，我们看到的应该是这个样子：

8	4	7	9	7	7	2	0	1	2	1	9	9	3	2	0	1	2	0	1	2	0	1	2	0	9	1	0	0	0	7	7	6	5	8
4	7	2	0	9	0	0	0	0	0	6	9	7	8	7	1	1	0	0	0	0	0	0	0	6	6	7	3	7	9	0	7	0	7	7	8
5	8	3	7	3	6	7	0	8	5	0	0	0	0	7	6	6	5	6	6	0	9	5													

在我们不知道这是个什么文件的情况下，我们是无从下手的。因为我们不知道到底要按照哪种“规则”来解析这串数字。但是我们在上一节中明确规定了从第几位到第几位是什么意思，那么我们就可以明确地、准确地解释这串数字了。为了方便说明，我们将这串数字按照“规则”着色：

8	4	7	9	7	7	2	0	1	2	1	9	9	3	2	0	1	2	0	1	2	0	1	2	0	9	1	0	0	0	7	7	6	5	8
4	7	2	0	9	0	0	0	0	0	6	9	7	8	7	1	1	0	0	0	0	0	0	0	6	6	7	3	7	9	0	7	0	7	7	8
5	8	3	7	3	6	7	0	8	5	0	0	0	0	7	6	6	5	6	6	0	9	5													

然后，我们就应该能轻易读出这个文件的内容了：

位编号	含义	内容	转换结果 (若果需要)
0 到 5	姓名的编码	847977	TOM
6 到 13	学号	20121993	-
14 到 17	年级	2012	-
18 到 19	班级	01	-
10 到 27	时间	20120910	-
28 到 37	课程名 1	0077658472	MATH
38 到 40	成绩 1	090	
41 到 50	课程名 2	0000697871	ENG
51 到 53	成绩 2	100	
54 到 63	课程名 3	0000667379	BIO
64 到 66	成绩 3	070	

67 到 76	课程名 4	7795837367	MUSIC
77 到 79	成绩 4	085	
80 到 89	课程名 5	0000766566	LAB
90 到 92	成绩 5	095	

也就是说，当我们想要读取一个文件时，我们先要知道它的所有数位中的哪一段分别代表什么意思，以及这个值需要通过哪种编码转换才能得到有意义的值。

通过这两节的论述，在亲手建立和解析了一个十进制文件之后，我们应该非常明确文件这个概念了。

图片是什么

图片也是一个大家每天打开电脑几乎都要接触的概念。上人人网，能看同学分享的照片。上 QQ 能在群里面转发各种搞怪图片……这些都是图片。那么对于要做这次实验的我们，需要怎么理解图片这个概念呢？

首先，我们应当考虑，一张图片里存有哪些信息。一张图片上存有哪些信息之后，我才能根据这些信息完整地重现这张图呢？显然，第一我要将这个图片中的每一个像素点都存入图片。打个比方，如果我这张图片里面有一大片蓝色的天，那么图片中就应该存在大量的(255, 0, 0)这种颜色的点。其他还要存储什么信息我们马上会讲到。

第二，图片在计算机里面通常是一个二进制文件。那么既然它是二进制文件，那么就一定是一串二进制数字的序列。我们只要知道了一种类型的图片存储这些数字的“规则”，那么我们就能够解析这串数字，从而读取这个文件的所有信息。

那么图片文件里面到底存储了哪些信息？这些信息又是如何被组织起来的？请继续看下一节的内容。

理解 BMP

有了上面的基础后，我们就能更为容易地理解图片文件了。

BMP 文件是最简单的图片格式，是其他图片格式的基础。BMP 文件也是按照一定规则，在数字串的不同段存储不同数据的。这里，我们举的例子是一张 4x4 的橘黄色纯色 BMP 图片。这个文件的数字序列为（这里你可以将附件中的 orange.bmp 用 UltraEdit 软件打开，即可查看该文件的数字序列）：

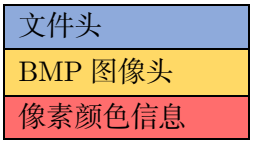
```
            0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00000000h: 42 4D 66 00 00 00 00 00 00 00 36 00 00 00 28 00 ; BMf.......6...(.
00000010h: 00 00 04 00 00 00 FC FF FF FF 01 00 18 00 00 00 ; ......üÿÿÿ......
00000020h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
00000030h: 00 00 00 00 00 00 1B 75 FC 1B 75 FC 1B 75 FC 1B ; .......uü.uü.uü.
00000040h: 75 FC 1B 75 FC 1B 75 FC 1B 75 FC 1B 75 FC 1B 75 ; uü.uü.uü.uü.uü.u
00000050h: FC 1B 75 FC 1B 75 FC 1B 75 FC 1B 75 FC 1B 75 FC ; ü.uü.uü.uü.uü.uü
00000060h: 1B 75 FC 1B 75 FC                               ; .uü.uü
```

这里，我们看到了前面不曾提到的十六进制数。需要说明的是，二进制文件本来是用二进制表示的，但是二进制数通常比较长（因为进制小），所以我们采用十六进制缩短表示这些数所需的长度。而且，真实的文件与前面我们亲手制造的文件有一个很大的不同，我们做的文件是以两个十进制位为一个数据单位组成的。真正的文件，比如上图的文件，其

组成单位是两个十六进制数，也就是 8 个二进制位，即一个字节。这一点我们要特别注意。

那么 BMP 文件是按照什么“规则”来排列这些字节的呢？

粗略地分，BMP 文件的数字序列大概由以下几个部分组成：



以上面那张图片为例，我们用下面的表格来具体地分析一下 BMP 图片中的每一字节代表的意义（注意与上图的颜色对应，另提醒读者这张图片的编码方式为小端模式）：

字节	意义	大小	值	解释
0x0000 到 0x0001	文件标志	2 字节	42 4D	这就是 ASCII 中的 BM 两个字母，代表这个文件存储了位图（Bit Map）
0x0002 到 0x0005	文件大小	4 字节	66 00 00 00	注意我们这张图的编码方式是小端模式，所以这个数字应当被解析为 0x00000066，也就是十进制的 102。
0x0006 到 0x0009	保留字节，永远都全为 0	4 字节	00 00 00 00	总是为 0
0x000A 到 0x000D	图像像素颜色信息的起始位置	4 字节	36 00 00 00	也就是像素颜色信息（本表红色部分）开始的地方为 0x00000036
0x000E 到 0x0011	BMP 图像头的大小	4 字节	28 00 00 00	也就是本表黄色部分所占字节数
0x0012 到 0x0015	图像宽度（单位:像素）	4 字节	04 00 00 00	也就是指本图宽 0x00000004 = 4 个像素
0x0016 到 0x0019	图像高度（单位:像素）	4 字节	FC FF FF FF	也就是指本图高 0x00000004 = 4 个像素
0x001A 到 0x001B	图像的调色板数量	2 字节	01 00	此图有一个调色板。但这个数据不起作用，因为此图不含调色板。只有每个像素占 8 位以下的图片才有调色板
0x001C 到 0x001D	图像每一个像素所占位数	2 字节	18 00	图像的每一个像素占 0x0018 = 24 位，也就是 3 个字节

0x001E 到 0x0021	图像数据的压缩方式	4 字节	00 00 00 00	这个值可以取从 0 到 6（十进制）中的某个值。每一个值代表一种压缩方式。此处为 0，说明不采用任何压缩。
0x0022 到 0x0025	像素颜色信息的总大小	4 字节	00 00 00 00	像素颜色区域中的原始数据大小
0x0026 到 0x0029	水平方向上，每米包含多少像素	4 字节	00 00 00 00	涉及到在屏幕上显示的时候，我们需要指定一米内的像素大小，也就是间接指定分辨率。但是我们这个例子中的图是一张设备无关位图，因此这个值为零
0x002A 到 0x002D	水平方向上，每米包含多少像素	4 字节	00 00 00 00	与上一项类似
0x003E 到 0x0031	调色板的颜色种数	4 字节	00 00 00 00	没有调色板，所以调色板的颜色总数也无从谈起，值为 0
0x0032 到 0x0035	0 表示所有颜色一样重要	4 字节	00 00 00 00	表示所有颜色同样重要
0x0036 到 0x0038	第一个像素的颜色	3 字节	1B 75 FC	也就是 RGB(27, 117, 252)
0x0039 到 0x003B	第二个像素的颜色	3 字节	1B 75 FC	与上一项类似
……				
0x0063 到 0x0065	第十六个像素的颜色	3 字节	1B 75 FC	与上一项类似

如果我们想要读取一张 BMP 图片中的信息，只需按照这张表，读取对应的字节即可。这里要提出的是，如果读者对表中的某些项目不理解，大可不必紧张。因为我们这个实验需要实现的只是从 BMP 到 TIFF，并不需要太复杂的信息。我们只需要知道一张 BMP 中的像素信息从哪里开始，每个像素多少位，然后一直读到文件末尾，我们就能得到最关键的像素颜色信息了。再需要知道高和宽，我们就能完全重现这张图片了。

深入理解 TIFF

相比 BMP，TIFF 文件格式我们要更为详细、深入地了解，因为我们要根据得到的像素颜色等信息创建这种文件。创建文件的过程是一个字节一个字节地操作的，我们需要精确掌控每一个字节。

然而，TIFF 文件格式和 BMP 相比有很大的不同，不是简单的线性结构。读者刚开始可能难以理解。不过在看完下面的描述之前，先不要打退堂鼓。

还记得我们在前面曾经亲手创造了一个成绩单文件吗？文件中每一项都被规定了最大长度，比如科目名段不能装下转换为 ASCII 后超过 5 个十进制位的名字。例子里面，英语（English）的科目名就不得不被简写为“Eng”。能不能想一个办法解决这个问题呢？如果我们需要存入“English”这个完整单词，那么我们应该这样做：

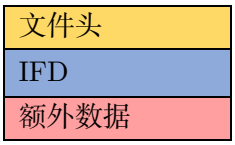
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
8	4	7	9	7	7	2	0	1	2	1	9	9	3	2	0
16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
1	2	0	1	2	0	1	2	0	9	1	0	0	0	7	7
32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
6	5	8	4	7	2	0	9	0	0	0	0	9	3	0	0
48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
0	1	4	1	0	0	0	0	0	0	6	6	7	3	7	9
64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79
0	7	0	7	7	8	5	8	3	7	3	6	7	0	8	5
80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95
0	0	0	0	7	6	6	5	6	6	0	9	5	6	9	7
96	97	98	99	100	85	86	87	88	89	90					
8	7	1	7	6	7	3	8	3	7	2					

发现了吗，表格最后多出了 14 位（红色部分），原来表中的第 41 到第 50 位（黄色部分）中的内容也发生了变化。这多出的 14 位的起始位置是 93，黄色部分的内容中也有 93 这个数字（前 5 位），同时也有 14 这个数字（后 5 位）。看到这里，读者应该明白了，要想存储超过某段长度的值，我们应该将这个值存放到文件的其他地方（比如文件末尾），然后在这个段中放入实际数据的开始位置和长度。这就有点像 C++中的指针了：在一个位数不够的段中存入指向实际数据的指针。

这就是 TIFF 与 BMP 的不同之处，也就是我们说 TIFF 是非线性文件的原因。为了让读者对 TIFF 文件有个清楚的认识，下面我们使用一张 4 x 4 的橘黄色 TIFF 图片来逐字节分析一下：

```
            0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00000000h: 4D 4D 00 2A 00 00 00 56 00 02 A0 02 00 04 00 00 ; MM.*...V.. .....
00000010h: 00 01 00 00 00 04 A0 03 00 04 00 00 00 01 00 00 ; ...... ........
00000020h: 00 04 00 00 00 00 FC 75 1B FC 75 1B FC 75 1B FC ; ......üu.üu.üu.ü
00000030h: 75 1B FC 75 1B FC 75 1B FC 75 1B FC 75 1B FC 75 ; u.üu.üu.üu.üu.üu
00000040h: 1B FC 75 1B FC 75 1B FC 75 1B FC 75 1B FC 75 1B ; .üu.üu.üu.üu.üu.
00000050h: FC 75 1B FC 75 1B 00 0D 01 00 00 03 00 00 00 01 ; üu.üu...........
00000060h: 00 04 00 00 01 01 00 03 00 00 00 01 00 04 00 00 ; ................
00000070h: 01 02 00 03 00 00 00 03 00 00 00 F8 01 03 00 03 ; ...........ø....
00000080h: 00 00 00 01 00 01 00 00 01 06 00 03 00 00 00 01 ; ................
00000090h: 00 02 00 00 01 11 00 04 00 00 00 01 00 00 00 26 ; ...............&
000000a0h: 01 12 00 03 00 00 00 01 00 01 00 00 01 15 00 03 ; ................
000000b0h: 00 00 00 01 00 03 00 00 01 16 00 03 00 00 00 01 ; ................
000000c0h: 00 04 00 00 01 17 00 04 00 00 00 01 00 00 00 30 ; ...............0
000000d0h: 01 1C 00 03 00 00 00 01 00 01 00 00 01 53 00 03 ; ............S..
000000e0h: 00 00 00 03 00 00 00 FE 87 69 00 04 00 00 00 01 ; .......þ‡i......
000000f0h: 00 00 00 08 00 00 00 00 00 08 00 08 00 08 00 01 ; ................
00000100h: 00 01 00 01                                     ; ....
```

TIFF 文件大概由以下几个部分组成：



其中 IFD 代表图像文件目录（Image File Directory），我们这个全称摆在这里仅供有闲心研究名字的读者看。其实 IFD 就是一些由 12 字节大小的段组成的一长串描述信息。每一个构成 IFD 的这些 12 字节的块叫做一条 entry[1]。每一条 entry 具有固定的结构，请参考下图：

0	1	2	3	4	5	6	7	8	9	10	11
属性 ID		数据长度的代号		有多少个这样的数据				值或值所在的地址			

下面分别解释以上上图所示的 Entry 的各个段：

1. 属性 ID：位于第 0 到第 1 个字节。这个段的值代表了这一条 entry 所描述的到底是这张图的什么属性，具体有哪些属性可以被描述，这些属性的 ID 是什么，请参考下表（不完全，有很多，只列出了本实验相关的）：

属性 ID	属性名
01 00	宽度
01 01	高度
01 02	表示颜色的每一个组分（R、G 或者 B）所需要的位数
01 03	压缩模式
01 06	光测度说明（Photometric Interpretation），一个奇怪的概念。我们在创建新文件的时候，就用一个固定值就行了，这个固定值可以参考一张现成的 TIFF 图片。
01 11	Strip 偏移量
01 12	图片朝向
01 15	每个像素颜色组分数（如果有 R、G 和 B 三个组分，那么这个值就应该为 3。如果还有 Alpha 通道，那么应该为 4。当然，还存在更多的情况。最普通的就是 3）
01 16	每个 strip 的行数

01 17	Strip 的字节数
01 1C	颜色组分存储方式
01 53	组分解释方式
87 69	EXIF 信息

2. 数据长度的代号：位于第 2 到第 3 个字节。里面填写的是一个代号，根据代号我们可以查到用于描述这个属性的一个数据所占字节数；比如，1 代表该数据为 BYTE 类型，占 1 字节。具体的代号意义参见下表：

代号	意义	大小
1	BYTE	1 字节
2	ASCII	8 字节
3	SHORT	2 字节
4	LONG	4 字节
5	RATIONAL	8 字节

3. 有多少个这样的数据：位于第 4 到第 7 个字节。描述了这个属性需要多少个这么长的数据；
4. 值或值所在的地址：位于第 8 到第 11 个字节。设根据数据长度代号所查询到的大小为 x 字节，共有 n 个这样的数据，那么这条 entry 所描述的属性的数据的所需字节数应为 nx。如果 nx 的大小小于 4 字节，那么这本段就直接存入这个数据。如果 nx 的大小不小于 4 字节，那么就把数据存到文件的另外的地方，并将其首字节的位置存入本段。存入本段时也有一定规则：如果这个值只占 1 或 2 个字节，则放入较高的 2 个字节中；如果占 3 或 4 个字节，则放入较低的 2 个字节中。

了解了 TIFF 的各项关键信息后，下面我们来分析上图中的每一个字节，并对其着色分类：

字节	意义	大小	值	解释
0x0000 到 0x0001	编码类型	2 字节	4D 4D	表明该文件是采用大端或者小端的编码方式。4D 4D 代表大端模式，49 49 代表小端模式
0x0002 到 0x0003	预留字节	2 字节	00 2A	一个表明文件类型的值，每个文件都一样，不用管它
0x0004 到 0x0007	IFD 的开始位置	4 字节	00 00 00 56	也就是说，IDF 的开始位置在 0x56 处
0x0008 到 0x0025	EXIF 信息	29 字节	略	图片的 EXIF 信息，如曝光时间、拍摄日期等。可以预料到，后面将会有某个段指向这 29 个字节
0x0026 到 0x0055	像素颜色	48 字节	略	图片中的橘黄色就是靠这些颜色显示出来的

0x0056 到 0x0057	IFD Entry 条数	2 字节	00 0D	即后面有多少个 12 字节的块（每一块称为 IFD Entry）
0x0058 到 0x0063	Entry 1	12 字节	01 00 00 03 00 00 00 01 00 04 00 00	01 00 即说明这个 IFD entry 描述了图像宽度； 00 03 指的是 BYTE，也就是一个数据要占 2 个字节； 00 00 00 01 是指需要用一个 SHORT 类型的数据来描述这个属性值； 00 04 00 00 指的是值为 4，即宽度为 4 像素
0x0064 到 0x006F	Entry 2	12 字节	01 01 00 03 00 00 00 01 00 04 00 00	01 01 说明此 IFD entry 描述的是图像高度； 00 03 指 BYTE，表明这个数据要占 2 字节； 00 00 00 01 表明需要用一个这样的类型的数据来描述这个属性； 00 04 00 00 说明这个属性的值为 4，即高度为 4 像素
0x0070 到 0x007B	Entry 3	12 字节	01 02 00 03 00 00 00 03 00 00 00 F8	01 02 说明此 IFD entry 描述的是每一个颜色的组分所需位数； 00 03 代表数据类型为 SHORT，占 2 个字节； 00 00 00 03 代表有三个这样类型的数据； 00 00 00 F8 即该属性的实际值。也就是说颜色的每个分量（R、G 或者 B）需要占 3 个字节
0x007C 到 0x0087	Entry 4	12 字节	01 03 00 03 00 00 00 01 00 01 00 00	01 03 指压缩模式； 00 03 指 SHORT，也就是需要 2 个字节来存储这个属性值； 00 00 00 01 说明需要 1 个 2 字节值来存储 00 01 即这个属性值，表明采用 1 这种压缩方式
0x0088 到 0x0093			01 06 00 03 00 00 00 01 00 02 00 00	01 06 指光测度说明； 可以不用管这个属性。稍后我们生成 TIFF 文件的时候就直接使用这 12 个字节的值就可以了

0x0094 到 0x009F			01 11 00 04 00 00 00 01 00 00 00 26	01 11 指 strip 偏移量。不用关心这个属性，生成文件的时候直接使用这个值就行了
0x00A0 到 0x00AB			01 12 00 03 00 00 00 01 00 01 00 00	01 12 指图片的朝向； 00 03 指 SHORT； 00 00 00 01 指需要一个 SHORT 类型的数据； 00 01 指 1，也就是水平朝向
0x00AC 到 0x00B7			01 15 00 03 00 00 00 01 00 03 00 00	01 15 指每个像素颜色的组分数； 00 03 指数据类型为 SHORT； 00 03 即说明，每个像素的颜色有三个分量
0x00B8 到 0x00C3			01 16 00 03 00 00 00 01 00 04 00 00	01 16 指每个 strip 的行数； 不用关心这个属性，生成文件的时候直接使用这个值即可
0x00C4 到 0x00CF			01 17 00 04 00 00 00 01 00 00 01 53	01 17 指 strip 的字节数； 不用关心这个属性，生成文件的时候直接使用这个值即可
0x00D0 到 0x00DB			01 1C 00 03 00 00 00 01 00 01 00 00	01 1C 指颜色组分的存储方式； 直接使用这个值即可，在新生成的图片中无需改变
0x00DC 到 0x00E7			01 53 00 03 00 00 00 03 00 00 00 FE	01 53 是指组分的解释方式； 无需关心
0x00E8 到 0x00F3			87 69 00 04 00 00 00 01 00 00 00 08	B7 69 指 EXIF 信息； 00 04 指 LONG，需要 4 个字节； 4 字节 x 1 个 = 4，则需要转存到文件的另一个地方，然后将那个地方的起始位置存入此处。这里，0x00000008 就是那一段数据的其实地址，也就是说，文件的第 8 个字节开始，就是本图片的 EXIF 信息。

理解文件操作

如果你知道 ifstream、ofstream 是什么、怎么用，那么你就可以跳过这一节。对于尚不清楚在 C++中如何操作文件的读者，请认真阅读本节。这一节除了对本实验有用处，在其他涉及到读写文件的方面都会使你受益。

C++中的文件操作主要利用 fstream 头文件中定义的两个文件类：ifstream 和 ofstream。ifstream 在读入文件时使用，ofstream 在输出文件时使用。也就是说，在我们这个 BMP 到 TIFF 转换的实验中，BMP 需要用 ifstream 打开，TIFF 文件需要使用 ofstream 创建。

C++里最适合存储字节的数据类型是 unsigned char，它可以表示 0x00 到 0xFF（也就是十进制的 0 到 255）。那么存储一串字节的方法就是使用 char 的数组。下面我们来看看具体要读入或者输出文件应该怎么操作。

打开二进制文件可采用下面的方式：

```
#include <fstream>
using namespace std;

int main() {
    ifstream f(" file path here ", ios::in | ios::binary);
    if (f) {
        unsigned char buffer[100];
        f.read(buffer, 100);
        f.close();
    }
    else {
        // do something when the file failed to open
    }
}
```

输出二进制文件采用下面的方式：

```
#include <fstream>
using namespace std;

int main() {
    ofstream f(" file path here ", ios::out | ios::binary);
    unsigned char buffer[100];
    f.write(buffer, 100);
    f.close();
}
```

了解模块化思想

这个实验如果不按照软件工程中的模块化思想，我们也能做出同样效果的作品来。但是，为了软件在将来我们想要修改时比较容易（不会动一发牵全身），或者想要将其中的某一部分（如图片信息抽取）拿出来给另一个程序用，我们就不能毫无章法地编写代码。

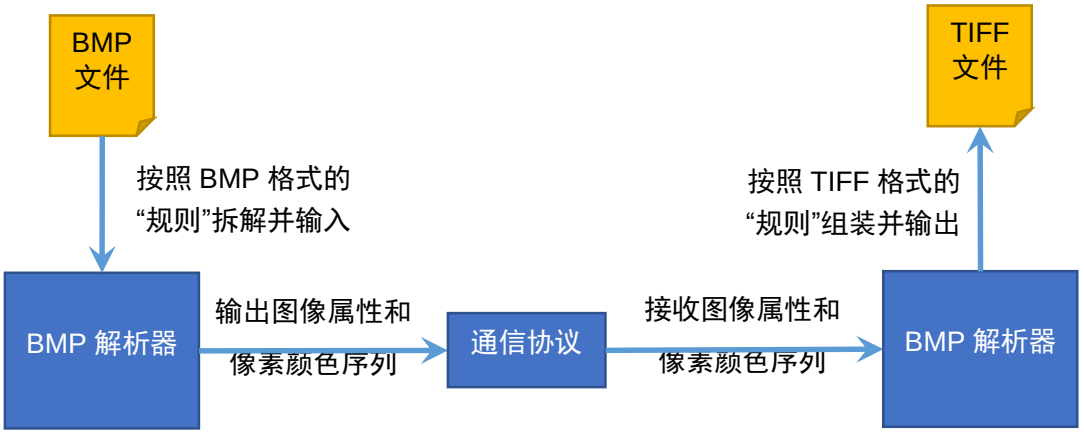
这里我们需要遵从模块化思想：把软件分割成几个模块，模块之间互不了解对方的内部实现细节，两者的通信通过接口进行。这样能保证我们以后在修改某处代码的时候，不会影响到全部代码。

我们首先应当把目标搞清楚：将一个 BMP 文件转换为 TIFF 文件。根据这个目标，我们采用分析法，得出我们解决这个问题的方案。首先看，我们最终要的是一张 TIFF 图片。要输出这样一张图片，我们必须知道图像的像素序列和多少像素构成一行（即宽度）等图片属性，然后将这些属性按照 TIFF 格式的要求，组装成一个字节序列，最后输出到一个文件中。要想得到这张图片的像素序列和宽度等信息的办法就是去解析源文件 BMP 格式。要解析 BMP 格式，我们必须先读入 BMP 格式的文件。

也就是说，我们应当按照下面的步骤完成图像文件的转换：

1. 从指定路径读入 BMP 格式的文件，获取其字节串；
2. 根据“规则”读取字节串中的各个段；
3. 解析这些段，获得图片的各种属性；
4. 将这些属性重新按照 TIFF 格式的要求组装成一个字节序列；
5. 将组装好的序列输出到指定路径的文件中。

那么，我们就应该按照下图的结构设计我们的软件：



那么显然，我们要做的工作就是写出两个工具类来：一个用于读入 BMP 文件并解析出该文件的属性和像素的颜色序列，另一个用于接收属性和像素序列并将其重新组合成一个 TIFF 文件输出。现在我们已经完全准备好了。

从开头一直坚持看到这里的读者，一定很有毅力。你花的时间不会被白费！在了解这么多前期准备知识后，我们就可以正式开始动手了！

参考资料

[1] ^维基百科条目：EXIF
[2] 未完待续