中国科学院大学
University of Chinese Academy of Sciences

信息隐藏

zxu@ict.ac.cn
zhangjialin@ict.ac.cn

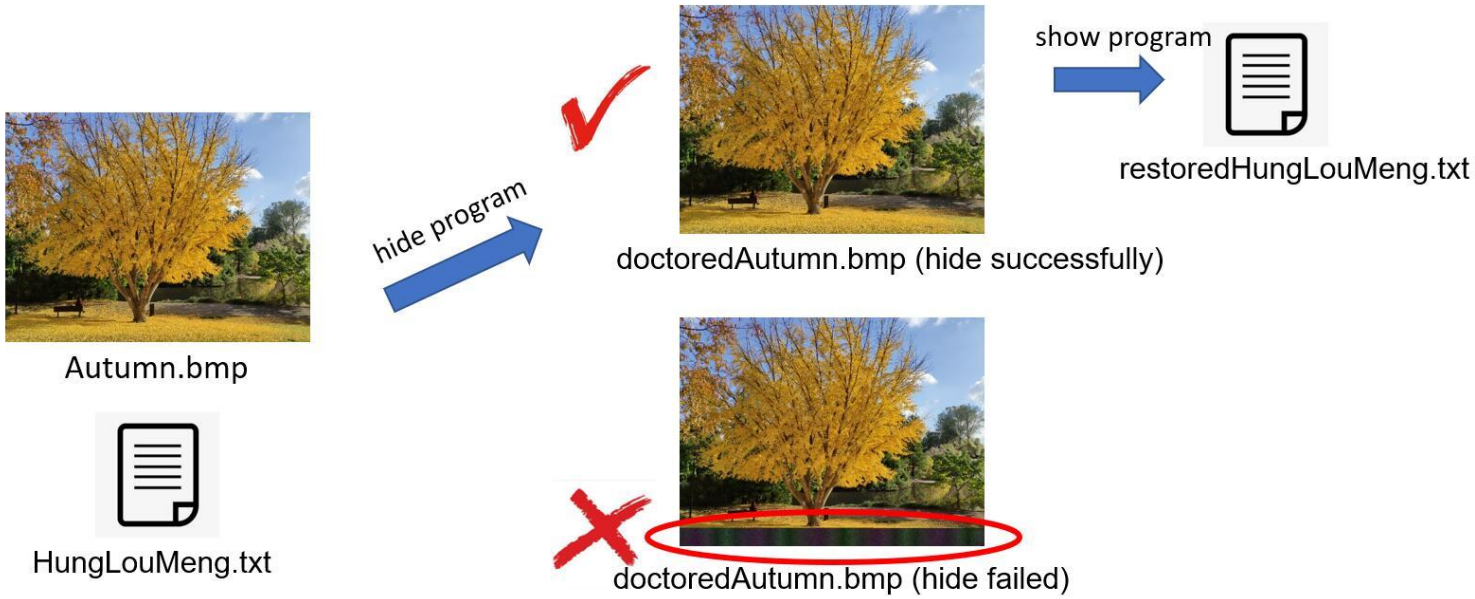
信息隐藏实验安排

周次	日期	课程内容	作业提交内容	截止时间
11	5月16日	实验要点、示例、信息隐藏框架	第一次代码	5月23日 10:00
12	5月23日	小组讨论、code review、答疑	第二次代码	5月30日 10:00

最终成绩取 2 次实验最高分
请关注网页上的实验要求，每次提交内容会有变化

目标

- 开发hide-0.go：把文本文件的内容隐藏到24位BMP图片中
  - 修改后的图片和原图片相比，不能有明显的视觉差异
- 开发show-0.go：从修改后的BMP图片中恢复文本文件的内容
  - 恢复的文本内容应与原文本内容完全相同
- 程序能在其他文本文件和24位BMP图片上执行隐藏和恢复操作



实验材料下载

- https://course.things.ac.cn:10088/exp/text_hider

资料下载

下载project.zip后解压，并上传至编程开发平台

实验要求

1. 本次提交要求详情请见信息隐藏实验课件
    1. 从图像的第218字节开始隐藏，先隐藏文本长度，再隐藏文本内容，每次修改图像像素字节的最低1位。
    2. 例如要把'H' = 01001000隐藏到像素字节切片的p[i: i+4], 需要把'H'的第0和1位（00），第2和3位（10），第4和5位（00）和第6和7位（01）依次隐藏到p[i], p[i+1], p[i+2], p[i+3]的最低2位。

第1次提交 **作业正在进行中:**

请提交

- 隐藏程序 选择文件 未选择文件
- 恢复程序 选择文件 未选择文件

提交

重要提醒

- 详细阅读中文教材第5.2节后，再开展实验
- 遵循良好的编码习惯
- 独立完成作业（抄袭会被判为0分）

如何在计算机中表示图片?
（内存或文件）

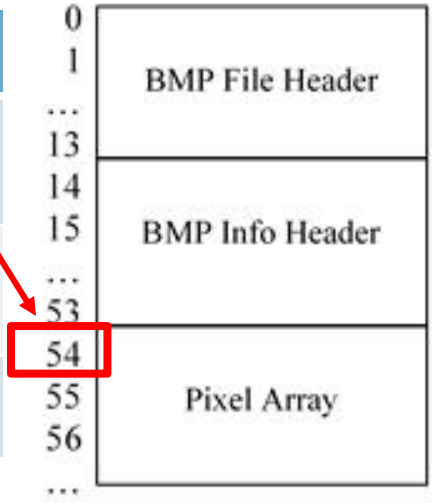
通过字节切片：一组元素构成的切片，其中每个元素是字节类型（**uint8**）

字节切片中保存元数据和像素

BMP图片格式

- 本实验中BMP图片的每个像素用3字节编码（24位）
- 图片像素阵列的起始地址在54字节

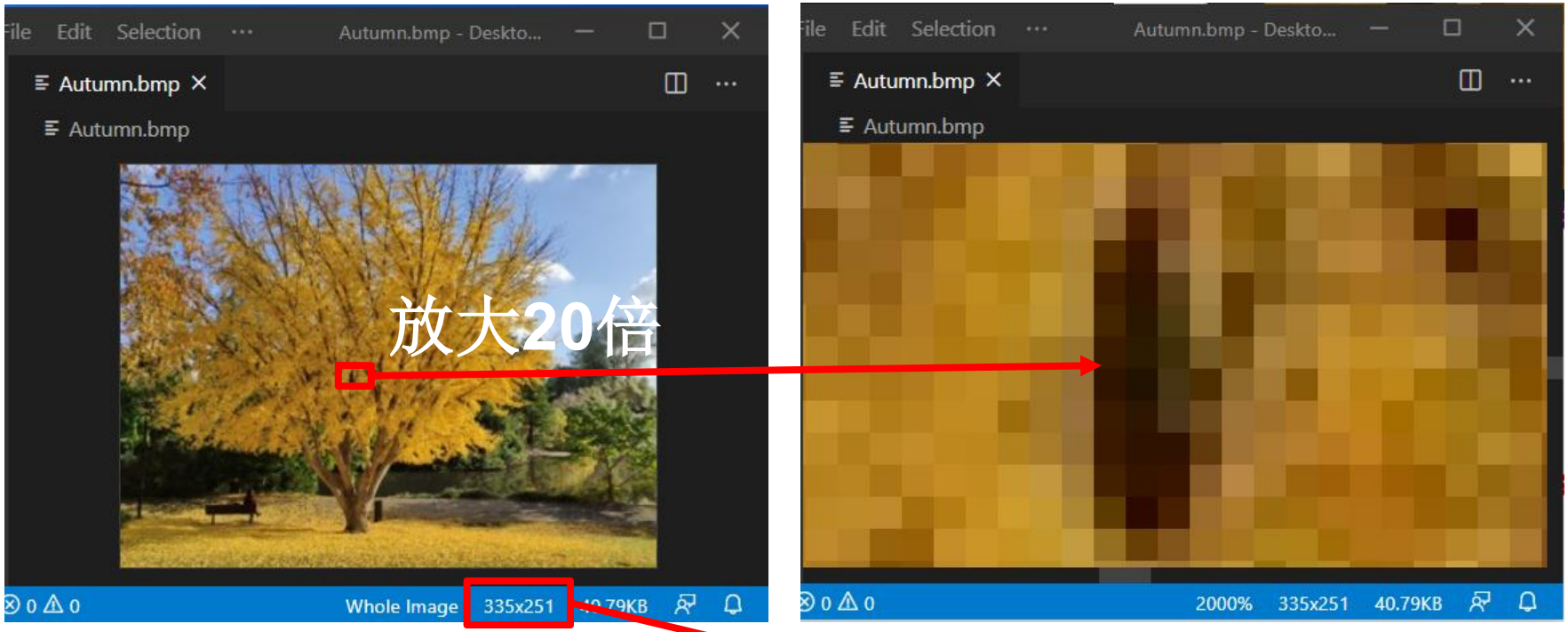
结构名称	大小	作用
File Header	14 字节	表明是BMP文件，存储BMP文件的通用信息，例如图片大小
BMP Info Header	40 字节	存储BMP文件的详细信息，例如该图片的宽和高
Pixel Array	3*宽*高 字节	像素阵列



实际上，现在的BMP格式与本实验中的图片格式不完全一致，感兴趣的同学可自行搜索

图片由像素构成

- 一个像素（pixel，picture element）表示图片中的一个点

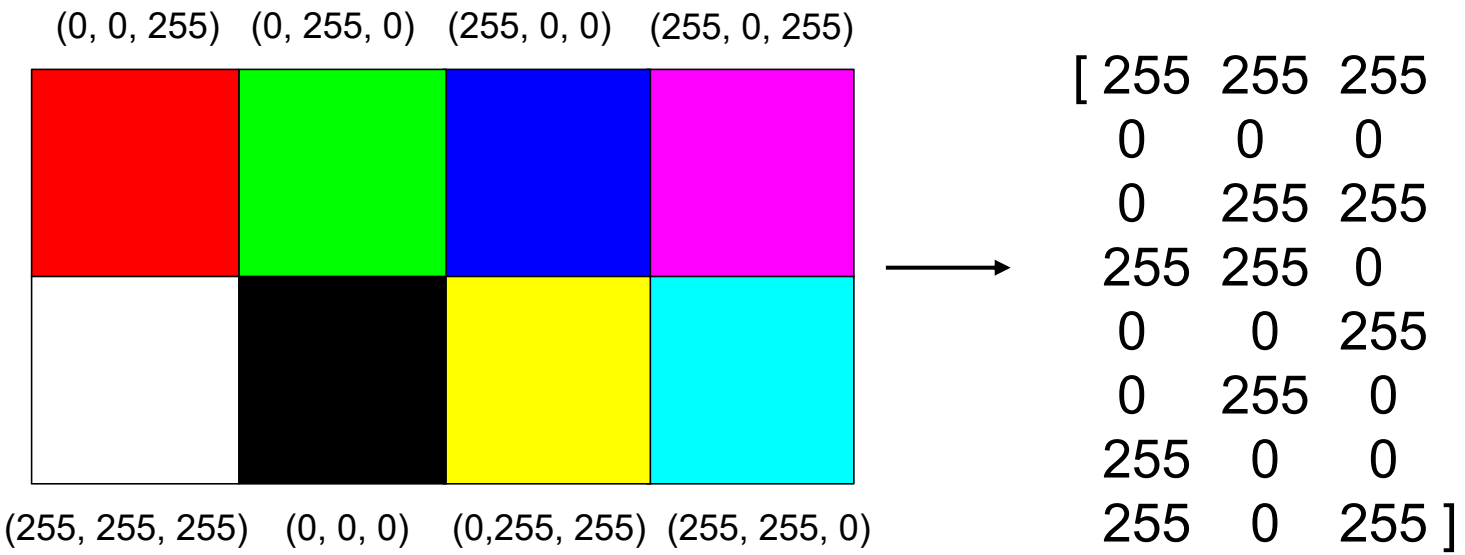


用VSCode打开的Autumn.bmp

335 x 251 像素

使用RGB编码像素

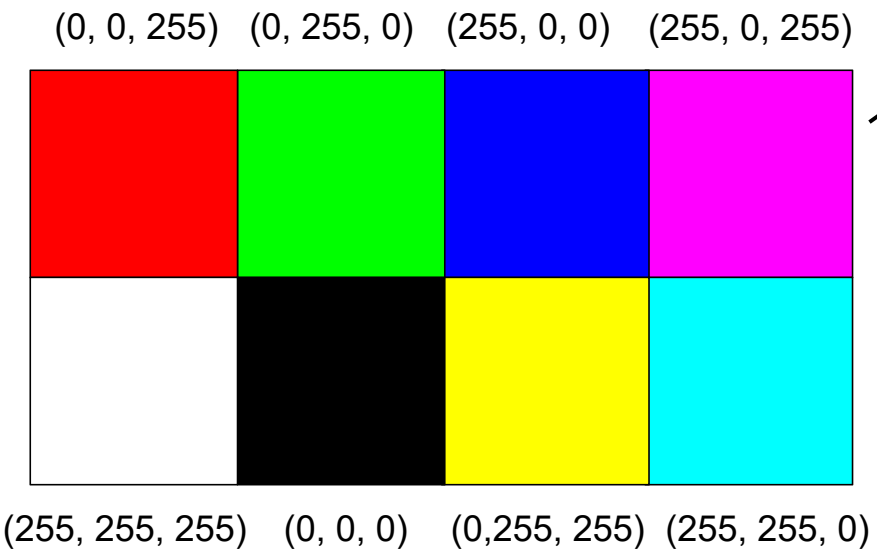
- 每个像素使用三元组(b,g,r)编码
  - b、g、r 是uint8 (0 - 255)类型的整数
  - 分别表示蓝色、绿色和红色的值
- 示例：
  - 一个有2x4个像素的图片，需要2x4x3=24 字节存储像素数据

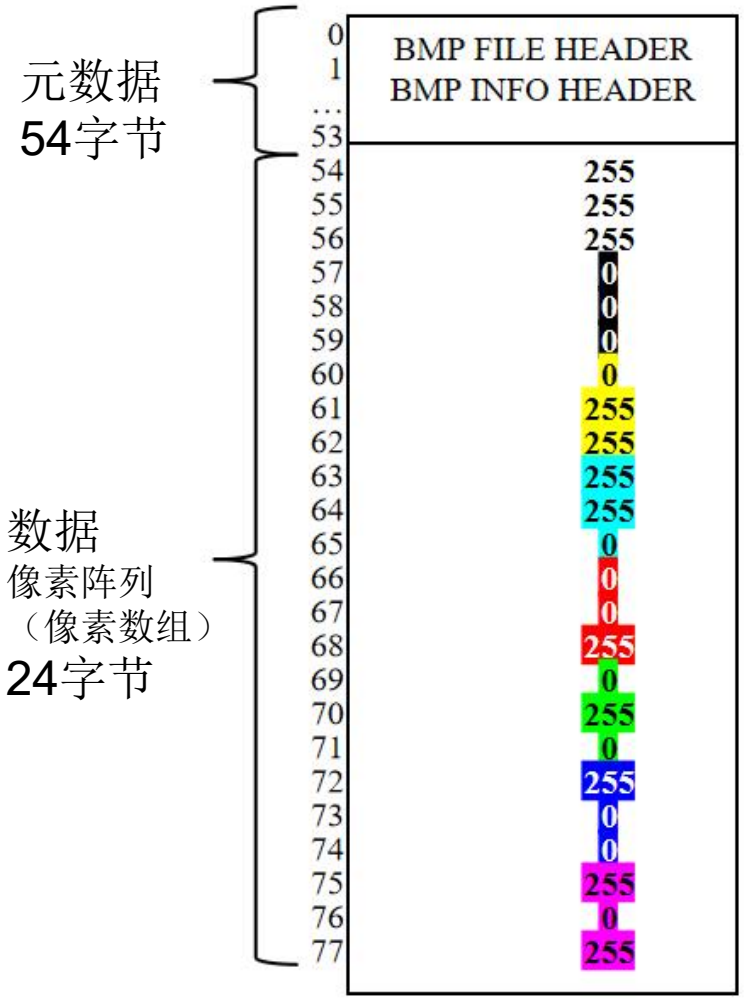


从底向上、从左向右顺序存储三元组

使用RGB编码像素

- 该BMP图片共有54+24 = 78字节
- 0-53字节为元数据，与实验无关，不需要修改
- 将信息隐藏在54字节起的像素数据中





示例1：反转颜色

- 程序：
  - 把panda.bmp读到变量p中
    - 像素阵列从 p[54]开始
  - 对于像素阵列每个元素的值v，计算255–v并赋给该元素
  - 把修改后的p写入到darkPanda.bmp中

反转每个像素的颜色

示例1：反转颜色

- 程序：
  - 把panda.bmp读到变量p中
    - 像素阵列从 p[54]开始
  - 对于像素阵列每个元素的值v，计算255-v并赋给该元素
  - 把修改后的p写入到darkPanda.bmp中

反转每个像素的颜色

问题: 如何把图片读取到变量中？
保存图片变量类型是什么？

如何把图片读取到变量中？

- 程序：
  - 把panda.bmp读到变量p中
    - 像素阵列从 p[54]开始
  - 对于像素阵列每个元素的值v，计算255-v并赋给该元素
  - 把修改后的p写入到darkPanda.bmp中

```
package main
import  "os"
func main() {
        p, _ := os.ReadFile("./panda.bmp")
        for i := 54; i < len(p); i++ {
                p[i] = 255 - p[i]
        }
        os.WriteFile("./darkPanda.bmp", p, 0644)
}
```

如何把图片读取到变量中？

- 程序：
  - 把panda.bmp读到变量p中
    - 像素阵列从 p[54]开始
  - 对于像素阵列每个元素的值v，计算255-v并赋给该元素
  - 把修改后的p写入到darkPanda.bmp中

os.ReadFile

- Go语言 “os” 包提供的函数
- 输入一个字符串格式的文件路径
- 返回保存文件内容的字节切片

```
p, _ := os.ReadFile("./panda.bmp")
```

- “./panda.bmp”：图片文件路径
- p：保存panda.bmp的字节切片类型的变量
- _：空白标识符，忽略返回值（此处忽略了 os.ReadFile的错误）

示例1：反转颜色

- 程序：
  - 把panda.bmp读到变量p中
    - 像素阵列从 p[54]开始
  - 对于像素阵列每个元素的值v，计算255-v并赋给该元素
  - 把修改后的p写入到darkPanda.bmp中

```
package main
import  "os"
func main() {
        p, _ := os.ReadFile("./panda.bmp")
        for i := 54; i < len(p); i++ {
                p[i] = 255 - p[i]
        }
        os.WriteFile("./darkPanda.bmp", p, 0644)
}
```

问题：
p[54]的含义是什么？

示例1：反转颜色

- 程序：
  - 把panda.bmp读到变量p中
    - 像素阵列从 p[54]开始
  - 对于像素阵列每个元素的值v，计算255-v并赋给该元素
  - 把修改后的p写入到darkPanda.bmp中

```
package main
import  "os"
func main() {
        p, _ := os.ReadFile("./panda.bmp")
        for i := 54; i < len(p); i++ {
                p[i] = 255 - p[i]
        }
        os.WriteFile("./darkPanda.bmp", p, 0644)
}
```

p[54]的含义:
图片最下一行最左侧像素的蓝色值，也是像素阵列的起始地址。

类似地，请思考 p[55] 、p[56]、p[57]的含义

示例1：反转颜色

修改前的 p （字节切片）

[..,240,176,0,…,0, 0, 0, …,255,255,255,…]

```
for i := 54; i <
len(p); i++ {
    p[i] = 255 - p[i]
}
```

修改后的 p

[…,15,79,255,…,255,255,255,…,0,0,0,…]

保存修改后的字节切片到文件

- 程序：
  - 把panda.bmp读到变量p中
    - 像素阵列从 p[54]开始
  - 对于像素阵列每个元素的值v，计算255-v并赋给该元素
  - 把修改后的p写入到darkPanda.bmp中

```
package main
import  "os"
func main() {
        p, _ := os.ReadFile("./panda.bmp")
        for i := 54; i < len(p); i++ {
                p[i] = 255 - p[i]
        }
        os.WriteFile("./darkPanda.bmp", p, 0644)
}
```

用**os.WriteFile**保存文件，参数表示：

1. 保存文件的路径；
2. 待保存的字节切片；
3. 保存文件的访问权限。

示例1：反转颜色

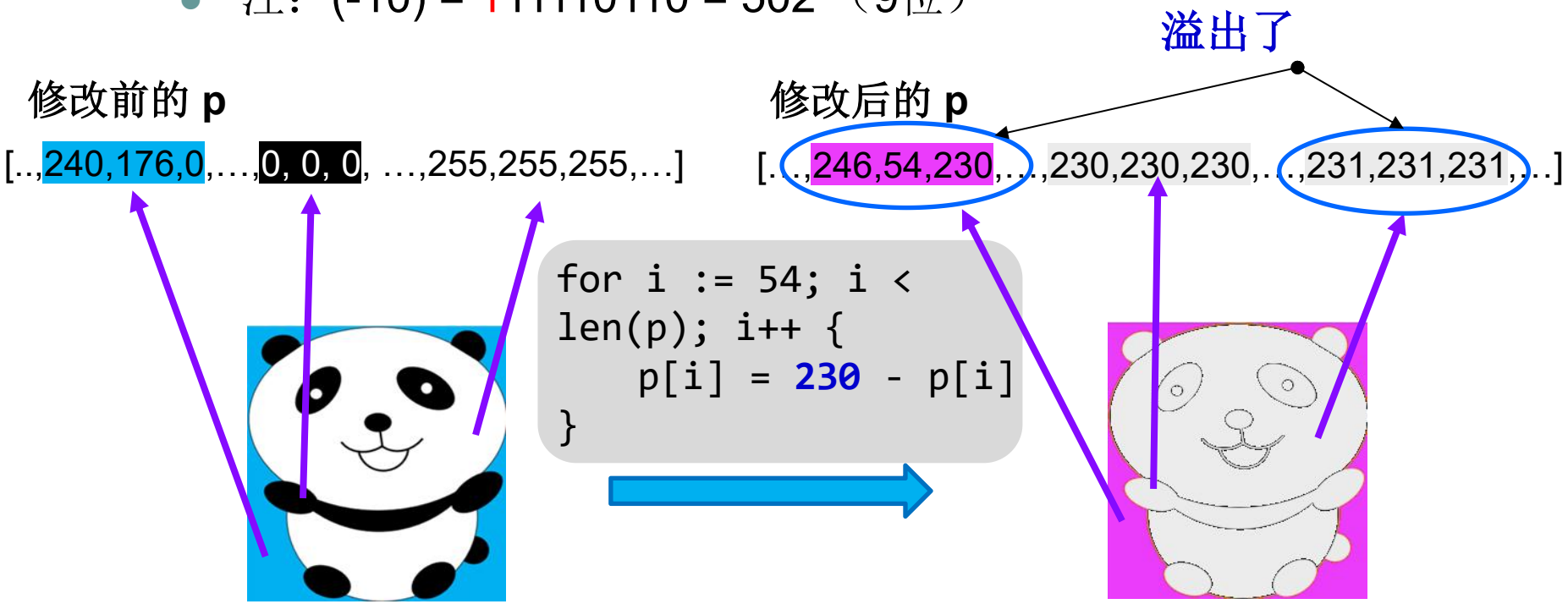
```
os.WriteFile("./darkPanda.bmp", p, 0644)
```

- 问题：0644的含义是什么？
  - Go语言中，以0开头的数字被视为8进制数
  - 访问权限
    - 不同粒度：该文件拥有者、拥有者所在的组、其他用户
    - 不同操作：读r、写w、执行x 该文件的权限
  - 文件拥有者可读、可写、不可执行；拥有者所在的组和其他用户只可读
    - -rw-r--r--
    - 0644 = 0 110 100 100
    - 使用 `ls -l` 查看

拥有者			所在组			其他用户		
r	w	x	r	w	x	r	w	x
1	1	0	1	0	0	1	0	0

如果把255改成230会怎样？

- 可能产生无符号整数溢出
  - 字节byte的实际类型为uint8，范围为[0,255]
  - 无符号整数+、-、*和<<运算溢出时，模2^n作为最终结果（n是无符号数的位宽）
    - 参考 https://golang.org/ref/spec#Integer_overflow
  - 230 - 240 = **(-10)** % 256 = **502**%256 = 246 （错误）
    - 注：(-10) = 111110110 = 502 （9位）

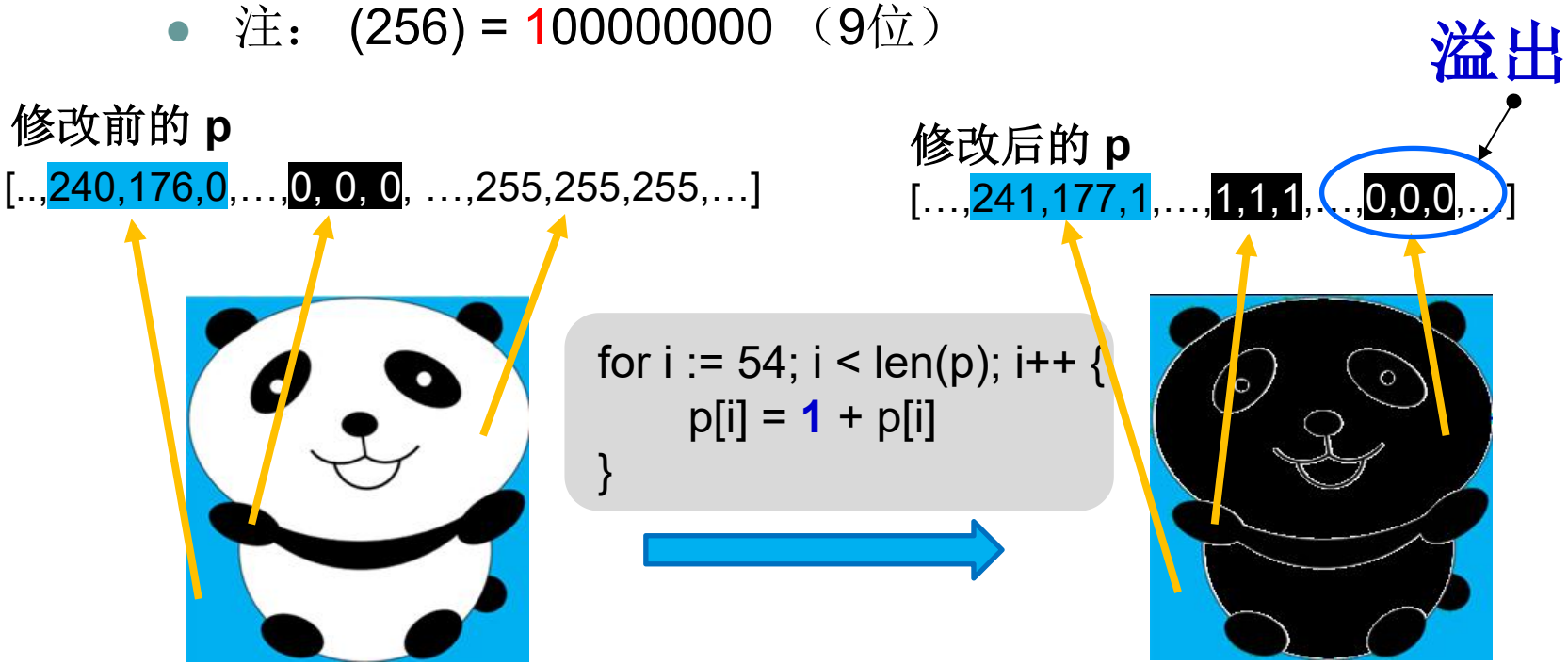


如果把p[i] = 255 - p[i]改成p[i] = p[i] + 1会怎样？

修改后的图片会产生明显的视觉差异吗？为什么？

如果改成p[i] = p[i] + 1会怎样？

- 可能产生无符号整数溢出
  - 字节byte的实际类型为uint8，范围为[0,255]
  - 无符号整数+、-、*和<<运算溢出时，模2^n作为最终结果（n是无符号数的位宽）
    - From https://golang.org/ref/spec#Integer_overflow
  - 1 + 255 = **(256)** % 256 = 0（错误）
    - 注： (256) = 100000000 （9位）



虽然每个字节只加1，但产生了溢出，图片视觉上产生明显差异

课堂练习：修改darkPanda.go

- 练习1：对于每个像素，反转每个字节的最低位
- 练习2：对于每个像素，反转每个字节的最高位
- 回顾：XOR 运算符
  - xor 1 （比特反转）
    - 1 ^ 1 = 0
    - 0 ^ 1 = 1
  - xor 0（保持不变）
    - 1 ^ 0 = 1
    - 0 ^ 0 = 0
- 接下来是同学的练习时间

XOR ^	1	0
1	0	1
0	1	0

课堂练习：修改darkPanda.go

- 练习1：对于每个像素，反转每个字节的最低位
- 练习2：对于每个像素，反转每个字节的最高位
- 回顾：XOR 运算符
  - xor 1 （比特反转）
    - 1 ^ 1 = 0
    - 0 ^ 1 = 1
  - xor 0（保持不变）
    - 1 ^ 0 = 1
    - 0 ^ 0 = 0
  - 给定一个字节x ，如何反转它的最低位？

XOR ^	1	0
1	0	1
0	1	0

课堂练习：修改darkPanda.go

- 练习1：对于每个像素，反转每个字节的最低位
- 练习2：对于每个像素，反转每个字节的最高位
- 回顾：XOR 运算符
  - xor 1 （比特反转）
    - 1 ^ 1 = 0
    - 0 ^ 1 = 1
  - xor 0（保持不变）
    - 1 ^ 0 = 1
    - 0 ^ 0 = 0
  - 给定一个字节x，如何反转它的最低位？
    - XOR 00000001
    - 11110000 XOR 00000001 = 11110001 (x = 240 = 11110000)

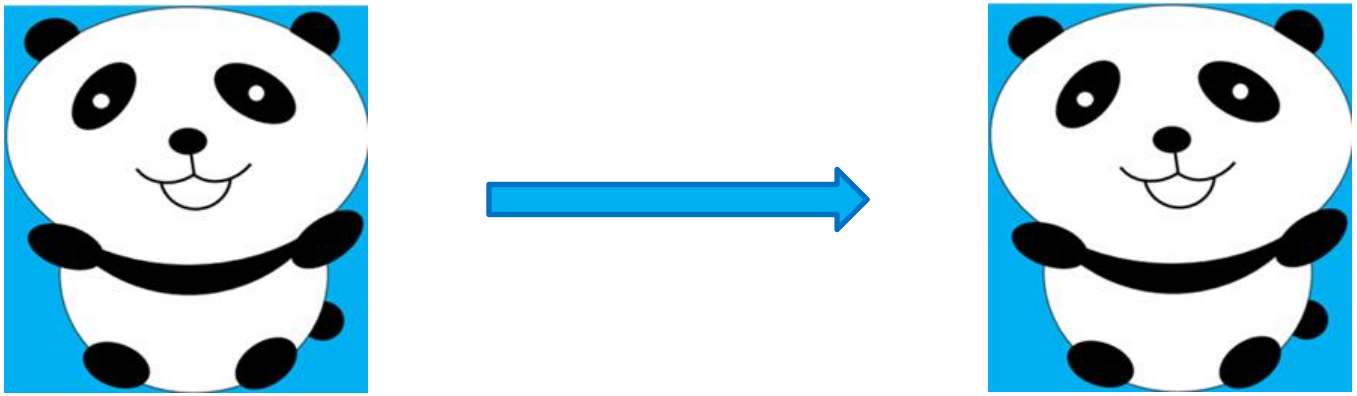
XOR ^	1	0
1	0	1
0	1	0

高位保持不变 最低位反转

练习1：对于每个像素，反转每个字节的最低位

- 对于像素阵列，修改每个字节的最低位几乎没有视觉差异

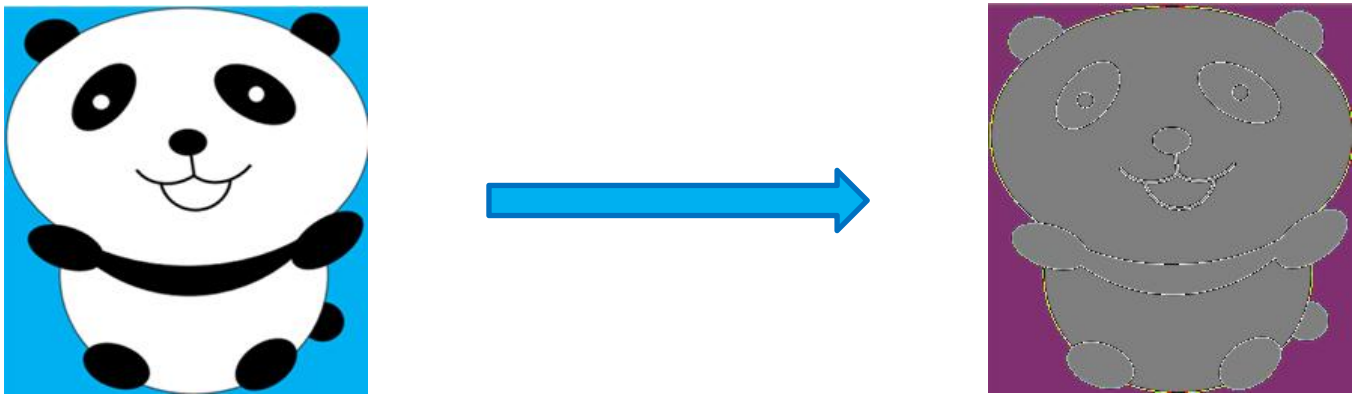
```
package main
import  "os"
func main() {
        p, _ := os.ReadFile("./panda.bmp")
        for i := 54; i < len(p); i++ {
                p[i] = p[i] ^ 0x1
        }
        os.WriteFile("./darkPanda.bmp", p, 0644)
}
```



练习2：对于每个像素，反转每个字节的最高位

- 对于像素阵列，修改每个字节的最高位会有视觉差异

```
package main
import  "os"
func main() {
        p, _ := os.ReadFile("./panda.bmp")
        for i := 54; i < len(p); i++ {
                p[i] = p[i] ^ 0x80
        }
        os.WriteFile("./darkPanda.bmp", p, 0644)
}
```



问题：

- 如何反转每个字节的最低2位？
- 如何反转每个字节的最高2位？

问题：

- 如何反转每个字节的最低2位？
- XOR 0x3
- 如何反转每个字节的最高2位？
- XOR 0xc0

示例1：良好编码习惯-1

- 使用清晰易懂的变量名
  - "headerSize"
- 避免魔数
  - 使用 “maxDepth” 替代魔数"255"
- 把常数定义放到前面

```
package main
import (
    "fmt"
    "os"
)
const (
    headerSize = 54                 // standard size of header
    fileMode   = 0644               // Access permissions
    maxDepth   = 255                // maximum color depth values
    srcImg     = "./panda.bmp"      // input image name
    destImg    = "./darkPanda.bmp"  // output image name
)
```

示例1：良好编码习惯-2

```
func main() {
    // Read "panda.bmp"
    imgBytes, err := os.ReadFile(srcImg)
    if err != nil {
        fmt.Printf(“read panda.bmp failed, err = %v\n", err)
        os.Exit(1)
    }
    // For each v in pixel array, invert it with (maxDepth- v)
    for i := headerSize; i < len(imgBytes); i++ {
        imgBytes[i] = maxDepth - imgBytes[i]
    }
    // save modified pixel array to "darkPanda.bmp"
    err = os.WriteFile(destImg, imgBytes, fileMode)
    if err != nil {
        fmt.Printf(“write image failed, err = %v\n", err)
        os.Exit(1)
    }
}
```

使用注释解释代码

避免重复代码

For loop

示例1：良好编码习惯-2

```
func main() {
    // Read "panda.bmp"
    imgBytes, err := os.ReadFile(srcImg)
    if err != nil {
        fmt.Printf("read panda.bmp failed, err = %v\n", err)
        os.Exit(1)
    }
    // For each v in pixel array, invert it with (maxDepth- v)
    for i := headerSize; i < len(imgBytes); i++ {
        imgBytes[i] = maxDepth - imgBytes[i]
    }
    // save modified pixel array to "darkPanda.bmp"
    err = os.WriteFile(destImg, imgBytes, fileMode)
    if err != nil {
        fmt.Printf("write image failed, err = %v\n", err)
        os.Exit(1)
    }
}
```

当程序报错时，尝试修复错误并重试

示例2：替换一个字节的最低2位

- 输入： 00111111, 00101010; Output: 00111110
- 代码及对应的解释

```
x := byte(63)  // 把 63=00111111 赋值给变量x
v := byte(42)  // 把 42=00101010 赋值给变量v
v = v & 0x3    // 按位与，仅保留v的最低2位
x = x & 0xFC   // 按位与，清除x的最低2位，保留最高6位
x = x | v      // 按位或，得到最终结果

x = 00111111                                    给定输入
v = 00101010                                    给定输入
v = 00101010 & 00000011       = 00000010        按位与
x = 00111111 & 11111100       = 00111100        按位与
x = 00111100 | 00000010       = 00111110        按位或

注：0x3 = 00000011；0xFC = 11111100
```

Mask mechanism
掩码机制

隐藏程序：把文本隐藏到BMP图片

- **输入：一个文本文件、一个图片文件**
- **输出：一个修改后的图片文件**
- **步骤：**

1. 获取命令行参数
   - srcImage：输入图片文件名；
   - srcTxt：输入文本文件名；
   - destImage：输出图片文件名
2. 把输入图片的内容读到变量p　　// p for picture
3. 把输入文本的内容读到变量t　　// t for text
4. 把文本的长度隐藏在变量p像素部分的前32字节
5. 把文本的内容隐藏在变量p像素部分的剩余字节
6. 把变量p保存到输出图片

隐藏程序：把文本隐藏到BMP图片

- 输入：一个文本文件、一个图片文件
- 输出：一个修改后的图片文件
- 步骤：

1. 获取命令行参数
   - srcImage：输入图片文件名；
   - srcTxt：输入文本文件名；
   - destImage：输出图片文件名
2. 把输入图片的内容读到变量p　　// p for picture
3. 把输入文本的内容读到变量t　　// t for text
4. 把文本的长度隐藏在变量p像素部分的前32字节
5. 把文本的内容隐藏在变量p像素部分的剩余字节
6. 把变量p保存到输出图片

```
go run hide-0.go -i Autumn.bmp -t HungLouMeng.txt -d doctoredAutumn.bmp
```

如何获取输入图片名、输入文本名和输出图片名？

```
go run hide-0.go -i Autumn.bmp -t HungLouMeng.txt -d doctoredAutumn.bmp
```

```
var (
    srcImage  string // input image name
    srcTxt    string // input text name
    destImage string // output doctored image name
)

// init sets command line arguments
func init() {
    // DON'T modify this function!!!
    flag.StringVar(&srcImage, "i", "", "input image path.")
    flag.StringVar(&srcTxt, "t", "", "input text path.")
    flag.StringVar(&destImage, "d", "", "output doctored image path.")
}
func main() {
    // parse command line arguments
    flag.Parse()
    if srcImage == "" || srcTxt == "" || destImage == "" {
        flag.PrintDefaults()
        os.Exit(1)
    }
```

使用“**flag**”包获取命令行参数（请勿修改以上代码）

隐藏程序：把文本隐藏到BMP图片

- **输入：一个文本文件、一个图片文件**
- **输出：一个修改后的图片文件**
- **步骤：**

1. 获取命令行参数
   - srcImage：输入图片文件名；
   - srcTxt：输入文本文件名；
   - destImage：输出图片文件名
2. 把输入图片的内容读到变量p　　// p for picture
3. 把输入文本的内容读到变量t　　// t for text
4. 把文本的长度隐藏在变量p像素部分的前32字节
5. 把文本的内容隐藏在变量p像素部分的剩余字节
6. 把变量p保存到输出图片

```
p, err := os.ReadFile(srcImage)
if err != nil {
    fmt.Printf("Read image file failed, err = %v\n", err)
    os.Exit(1)
}
```

隐藏程序：把文本隐藏到BMP图片

- **输入：一个文本文件、一个图片文件**
- **输出：一个修改后的图片文件**
- **步骤：**

1. 获取命令行参数
   - srcImage：输入图片文件名；
   - srcTxt：输入文本文件名；
   - destImage：输出图片文件名
2. 把输入图片的内容读到变量p // p for picture
3. 把输入文本的内容读到变量t // t for text
4. 把文本的长度隐藏在变量p像素部分的前32字节
5. 把文本的内容隐藏在变量p像素部分的剩余字节
6. 把变量p保存到输出图片

```
t, err := os.ReadFile(srcTxt)
if err != nil {
    fmt.Printf("Read text file failed, err = %v\n", err)
    os.Exit(1)
}
```

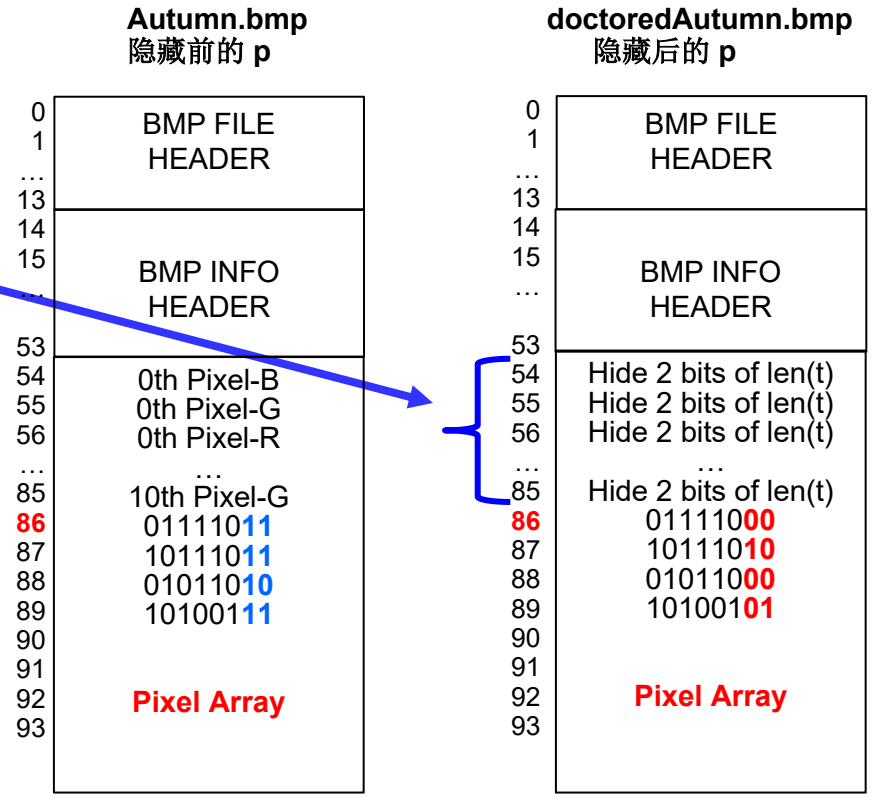
隐藏程序：把文本隐藏到BMP图片

- **输入：一个文本文件、一个图片文件**
- **输出：一个修改后的图片文件**
- **步骤：**

1. 获取命令行参数
   - srcImage：输入图片文件名；
   - srcTxt：输入文本文件名；
   - destImage：输出图片文件名
2. 把输入图片的内容读到变量p　　// p for picture
3. 把输入文本的内容读到变量t　　// t for text
4. 把文本的长度隐藏在变量p像素部分的前32字节
5. 把文本的内容隐藏在变量p像素部分的剩余字节
6. 把变量p保存到输出图片

如何把文本文件的长度隐藏到图片？

- t是输入文本的字节切片
- p是输入图片的字节切片
- len(t) 是64位整数
  - 每2位隐藏到p的1个字节中
  - 需要32个字节
  - S = 54, T = 32
- modify(len(t), p[S:S+T], T)
  把len(t) 隐藏到 p[54:86]

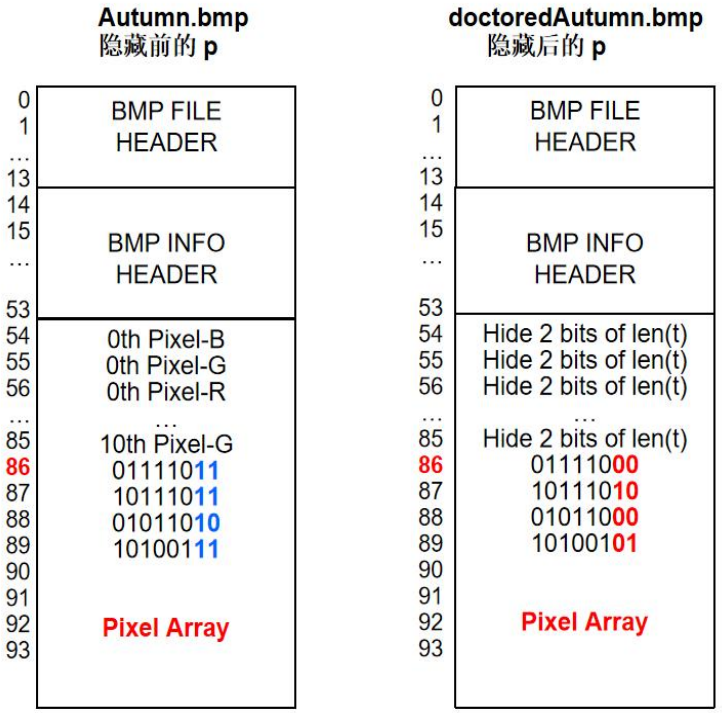


如何把文本文件的长度隐藏到图片？

- t是输入文本的字节切片
- p是输入图片的字节切片
- len(t) 是64位整数
  - 每2位隐藏到p的1个字节中
  - 需要32个字节
  - S = 54, T = 32
- modify(len(t), p[S:S+T], T)
  把len(t) 隐藏到 p[54:86]

```
func modify(value int, pix []byte, size int) {
    for i := 0; i < size; i++ {
        replace last 2 bits of pix[i]
             with the last 2 bits of value
        repeat with the next 2 bits of value
    }
}
```

请参考本课件的示例2

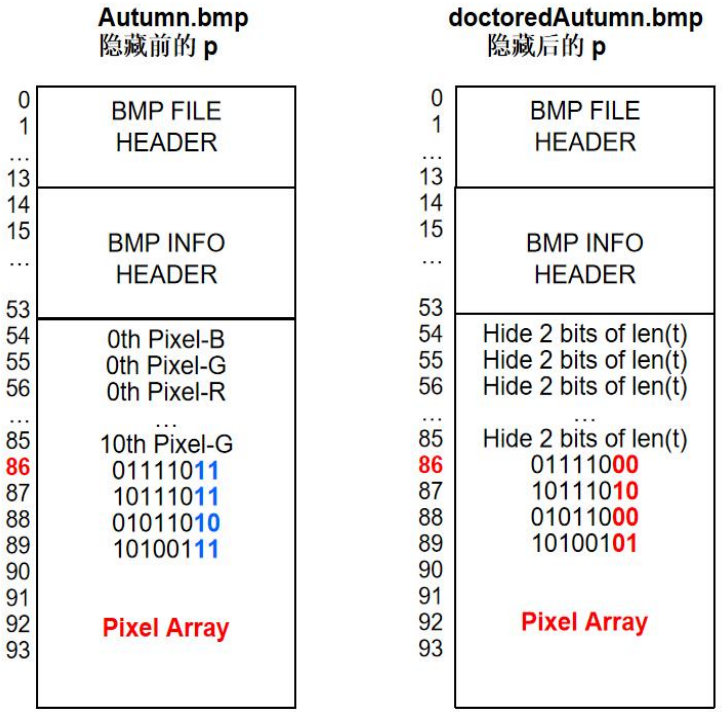


如何把文本文件的长度隐藏到图片？

- t是输入文本的字节切片
- p是输入图片的字节切片
- len(t) 是64位整数
  - 每2位隐藏到p的1个字节中
  - 需要32个字节
  - S = 54, T = 32
- modify(len(t), p[S:S+T], T)
  把len(t) 隐藏到 p[54:86]

```
func modify(value int, pix []byte, size int) {
    for i := 0; i < size; i++ {
        replace last 2 bits of pix[i]
             with the last 2 bits of value
        repeat with the next 2 bits of value
    }
}
```

使用右移：把value右移2位



隐藏程序：把文本隐藏到BMP图片

- **输入：一个文本文件、一个图片文件**
- **输出：一个修改后的图片文件**
- **步骤：**

1. 获取命令行参数
   - srcImage：输入图片文件名；
   - srcTxt：输入文本文件名；
   - destImage：输出图片文件名
2. 把输入图片的内容读到变量p　　// p for picture
3. 把输入文本的内容读到变量t　　// t for text
4. 把文本的长度隐藏在变量p像素部分的前32字节
5. 把文本的内容隐藏在变量p像素部分的剩余字节
6. 把变量p保存到输出图片

```
for i := 0; i < len(t); i++ {
        offset := S + T + C*i
        modify(int(t[i]), p[offset:offset+C], C)
}
```

如何把文本文件的内容隐藏到图片？

- t是输入文本的字节切片
- p是输入图片的字节切片
- t[0]是要隐藏的第1个字符 'H' = 72
- `modify(int(t[0]), p[S+T:S+T+C], C)`
  其中
  - t[0] = 'H' = 72 = **01001000**
  - S = 54, T = 32, C = 4
  - p[S+T:S+T+C] 为 p[86:90]

Original p[86:90]

86	01111011
87	10111011
88	01011010
89	10100111

Modified p[86:90]

	01111000
	10111010
	01011000
	10100101

Autumn.bmp
隐藏前的 p

0 1 … 13	BMP FILE HEADER
14 15 … 53	BMP INFO HEADER
54 55 56 … 85 86 87 88 89 90 91 92 93	0th Pixel-B 0th Pixel-G 0th Pixel-R … 10th Pixel-G 01111011 10111011 01011010 10100111 Pixel Array

doctoredAutumn.bmp
隐藏后的 p

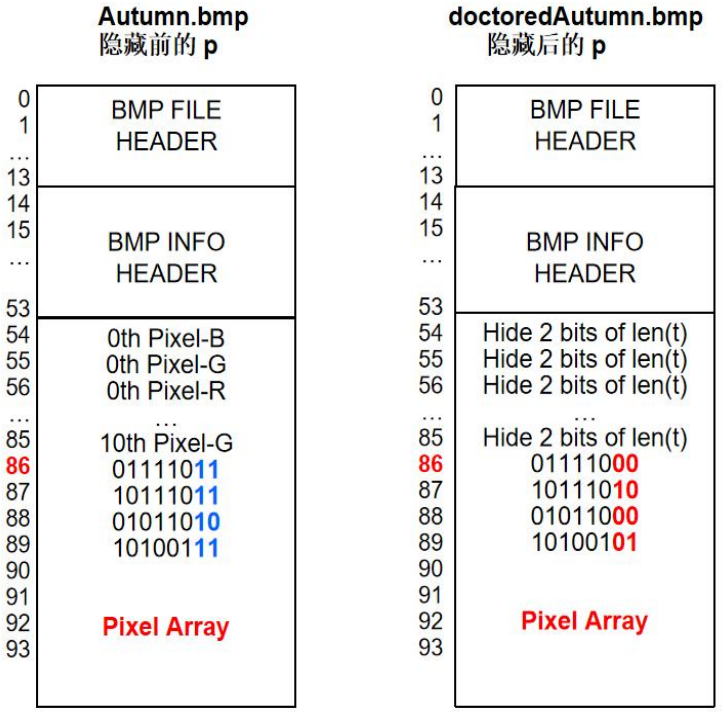
0 1 … 13	BMP FILE HEADER
14 15 … 53	BMP INFO HEADER
54 55 56 … 85 86 87 88 89 90 91 92 93	Hide 2 bits of len(t) Hide 2 bits of len(t) Hide 2 bits of len(t) … Hide 2 bits of len(t) 01111000 10111010 01011000 10100101 Pixel Array

如何把文本文件的内容隐藏到图片？

- t是输入文本的字节切片
- p是输入图片的字节切片
- 隐藏t[i]

```
for i:=0; i<len(t); i++{
    offset := S+T+C*i
    modify(int(t[i]), p[offset:offset+C], C)
}
```

- 每次迭代把 t[i] 隐藏到 `p[S+T+(i*C):S+T+(i*C)+C]`
  - 其中S = 54, T = 32, C = 4
- 即 t[i] 隐藏到 `p[86+(i*4)], p[86+(i*4)+1], p[86+(i*4)+2], p[86+(i*4)+3]`
- 例如t[1]='A' 隐藏到p[90:94]



隐藏程序：把文本隐藏到BMP图片

- **输入：一个文本文件、一个图片文件**
- **输出：一个修改后的图片文件**
- **步骤：**

1. 获取命令行参数
   - srcImage：输入图片文件名；
   - srcTxt：输入文本文件名；
   - destImage：输出图片文件名
2. 把输入图片的内容读到变量p　　// p for picture
3. 把输入文本的内容读到变量t　　// t for text
4. 把文本的长度隐藏在变量p像素部分的前32字节
5. 把文本的内容隐藏在变量p像素部分的剩余字节
6. 把变量p保存到输出图片

```
err = os.WriteFile(destImage, p, 0644)
if err != nil {
        fmt.Printf("Write doctored image failed, err = %v\n", err)
        os.Exit(1)
}
```

如何检查程序的执行结果？

- 完成hide-0.go和show-0.go
- 执行并查看结果

```
> go run hide-0.go -i Autumn.bmp -t HungLouMeng.txt -d doctoredAutumn.bmp
> display doctoredAutumn.bmp
> go run show-0.go -i doctoredAutumn.bmp -t restoredHungLouMeng.txt
> diff HungLouMeng.txt restoredHungLouMeng.txt
```

- 比较原始图片和修改后的图片（无视觉差异）

Autumn.bmp

doctoredAutumn.bmp

- 比较原始文本和恢复的文本（如下图使用diff命令比较，命令行无任何输出）

```
/cs101/code > diff HungLouMeng.txt restoredHungLouMeng.txt
/cs101/code >
```

hide-0.go – part 1

```go
package main

import (
    "flag"
    "fmt"
    "os"
)

const (
    S = 54 // standard size of bmp headers
    T = 32 // number of bytes needed to hide the text length
    C = 4  // number of bytes needed to hide a character
)

// modify hides an integer to a byte slice
func modify(value int, pix []byte, size int) {
    for i := 0; i < size; i++ {
        // TODO: write your code here
        // replace last 2 bits of pix[i] with the last 2 bits of value
        // the next iteration repeats with the next 2 bits of value
    }
}

var (
    srcImage  string // input image name
    srcTxt    string // input text name
    destImage string // output doctored image name
)

// init sets command line arguments
func init() {
    // DON'T modify this function!!!
    flag.StringVar(&srcImage, "i", "", "input image name")
    flag.StringVar(&srcTxt, "t", "", "input text name")
    flag.StringVar(&destImage, "d", "", "output doctored image name")
}
```

完成“modify”函数

请关注平台提交页面你的起始隐藏位数和每次修改像素字节的最低位数，PPT中展示的是54位和2位

不要修改“init”函数

hide-0.go – part 2

```go
func main() {
    // parse command line arguments
    flag.Parse()
    if srcImage == "" || srcTxt == "" || destImage == "" {
        flag.PrintDefaults()
        os.Exit(1)
    }
    // read input image to a byte slice p
    p, err := os.ReadFile(srcImage)
    if err != nil {
        fmt.Printf("Read image file failed, err = %v\n", err)
        os.Exit(1)
    }
    // read input text to a byte slice t
    t, err := os.ReadFile(srcTxt)
    if err != nil {
        fmt.Printf("Read text file failed, err = %v\n", err)
        os.Exit(1)
    }
    // check if the text is too big
    if T+len(t)*C > len(p[S:]) {
        fmt.Println("The text file is too big")
        os.Exit(1)
    }
    // save the text length to p
    modify(len(t), p[S:S+T], T)
    // save the content of text to p
    for i := 0; i < len(t); i++ {
        offset := S + T + C*i
        modify(int(t[i]), p[offset:offset+C], C)
    }
    // write the modified p to destImage
    err = os.WriteFile(destImage, p, 0644)
    if err != nil {
        fmt.Printf("Write doctored image failed, err = %v\n", err)
        os.Exit(1)
    }
}
```

不要修改该代码块

show-0.go

```go
package main

import (
    "flag"
    "os"
)

const (
    S = 54 // standard size of header
    T = 32 // number of bytes needed to hide the text length
    C = 4  // number of bytes needed to hide a character
)

var (
    image string // input doctor image name
    txt   string // output text name
)

// init sets command line arguments
func init() {
    // DON'T modify this function!!!
    flag.StringVar(&image, "i", "", "input image name")
    flag.StringVar(&txt, "t", "", "output text name")
}

func main() {
    // parse command line arguments
    flag.Parse()
    if image == "" || txt == "" {
        flag.PrintDefaults()
        os.Exit(1)
    }
    // TODO: write your code here
}
```

可以在此定义新函数

不要修改该代码块

评分标准

- 采用5档分制，如下表所示

分数	评判标准
4	独立完成且正确，少量使用他人的素材（有标注致谢），有创意
3	独立完成且大部分正确，少量使用他人的素材（有标注致谢）
2	完成且基本正确（如代码运行但结果有错），并标注致谢了使用他人的素材
1	截止时间前提交，但有明显错误（如代码不能运行）或缺失50%标注致谢
0	未在截止时间前提交，或抄袭

- 鼓励主动学习，惩罚抄袭。
  - 鼓励提问与交流，使用了他人的素材（如数据、代码等）必须显式地标注致谢。
  - 可随意使用教学团队提供的材料，包括助教提供的帮助和回答，不需要显式地标注致谢。

关键时间节点

- 在本幻灯片中，从54字节起隐藏，每次隐藏到字节的低2位
- 提交作业时，每位同学开始隐藏的字节数与隐藏到字节的最低位数均不同，请关注平台提交页面具体要求
- 2025.5.23　上午10:00　第一次代码提交截止时间
- 2025.5.30　上午10:00　第二次代码提交截止时间

- 最终取2次最高分

实验要求

1. 本次提交要求详情请见信息隐藏实验课件
    1. 从图像的第218字节开始隐藏，先隐藏文本长度，再隐藏文本内容，每次修改图像像素字节的最低1位。
    2. 例如要把'H' = 01001000隐藏到像素字节切片的p[i: i+4], 需要把'H'的第0和1位（00），第2和3位（10）,第4和5位（00）和第6和7位（01）依次隐藏到p[i], p[i+1], p[i+2], p[i+3]的最低2位。

代码提交

- https://course.things.ac.cn:10088/exp/text_hider