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

CS101

信息隐藏

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

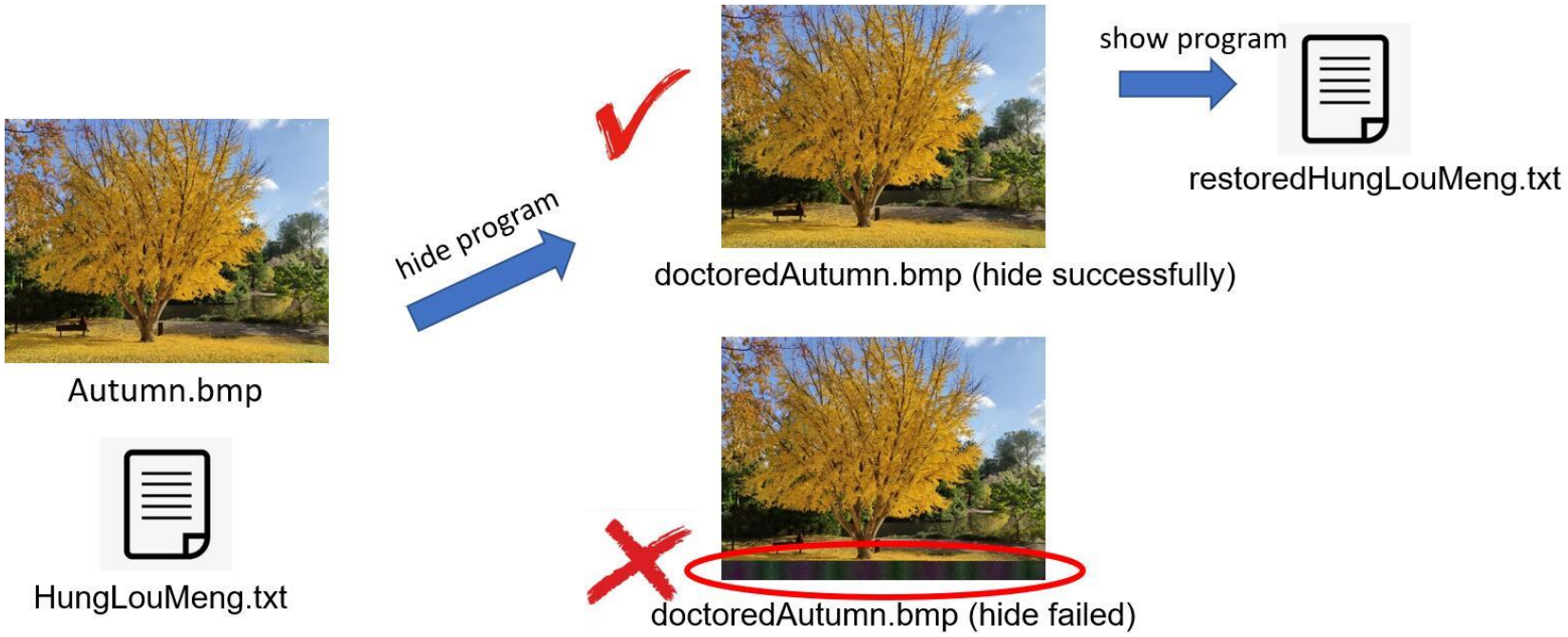
提纲

- 第一周：实验要点、示例、信息隐藏框架、答疑
- 第二、三周：小组讨论、Code Review、答疑

目标

阅读英文教科书7.2节

- 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
- 程序需要能在其他文本文件和24位BMP图片上执行隐藏和恢复操作
- 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）

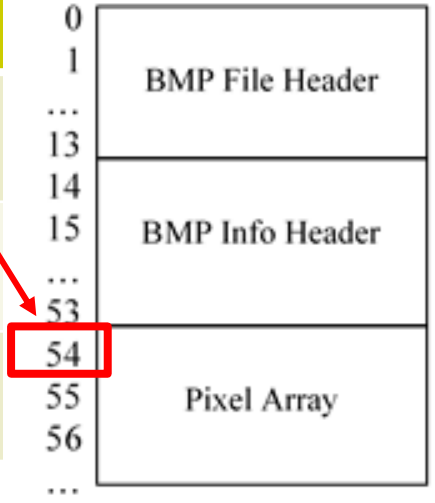
通过字节切片：一组元素构成的切片，其中每个元素是字节类型（uint8）

字节切片中保存元数据和像素

BMP图片格式

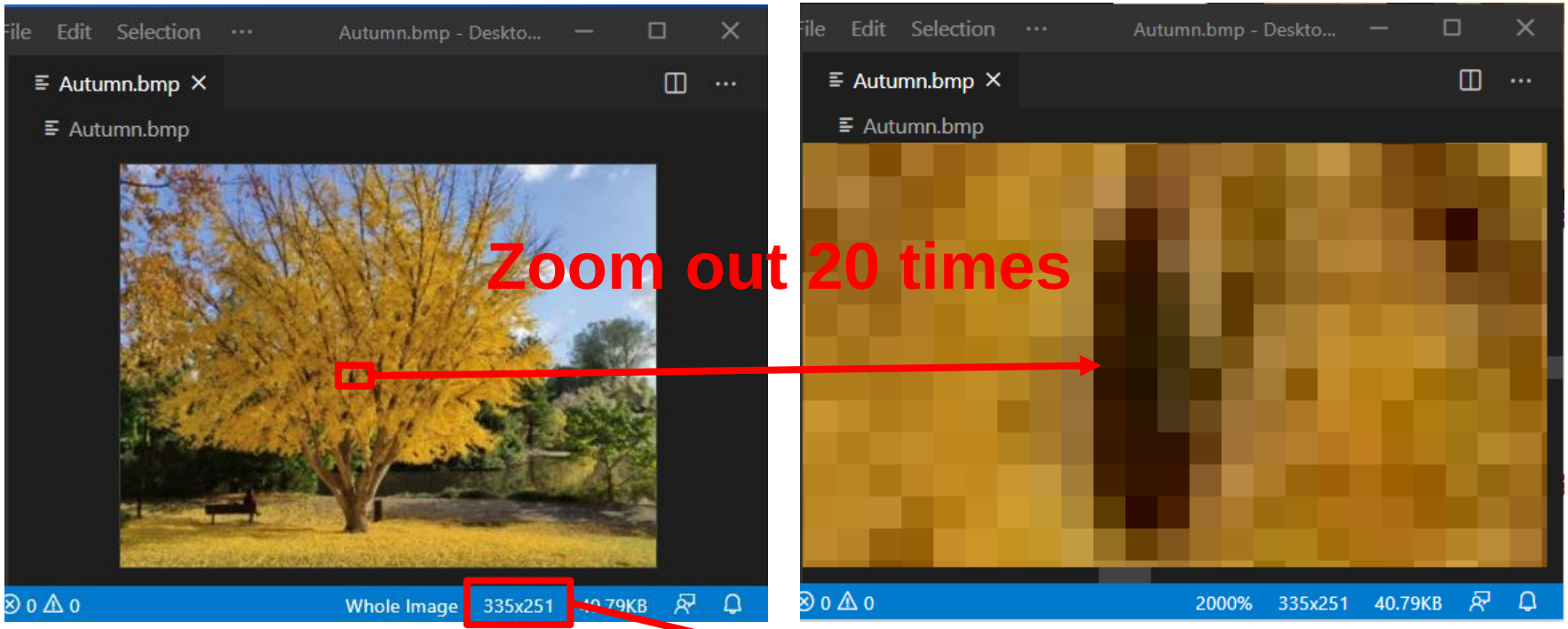
- 本实验中BMP图片的每个像素用3字节编码（24位）
- 图片像素阵列的起始地址在54字节

结构名称	大小	作用
File Header	14 bytes	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 bytes	存储BMP文件的详细信息，例如该图片的Height、Width
Pixel Array	3*Height* Width	像素阵列



图片由像素构成

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

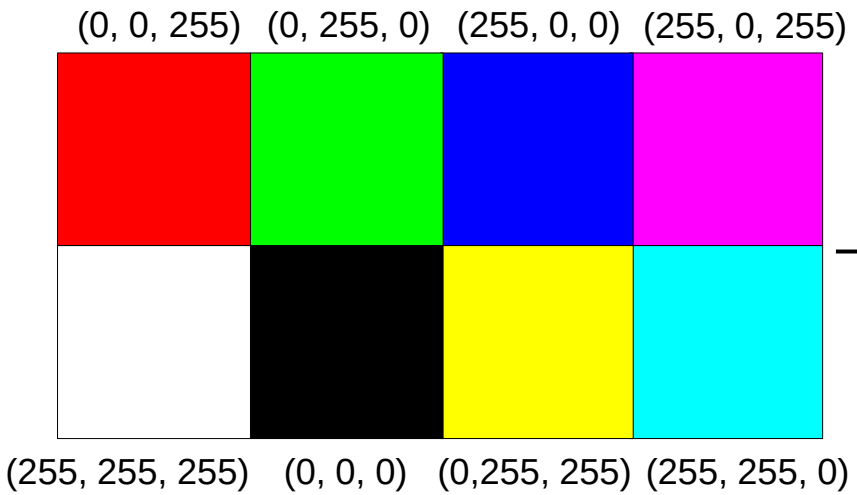


用VSCode打开的Autumn.bmp

335 x 251 pixels

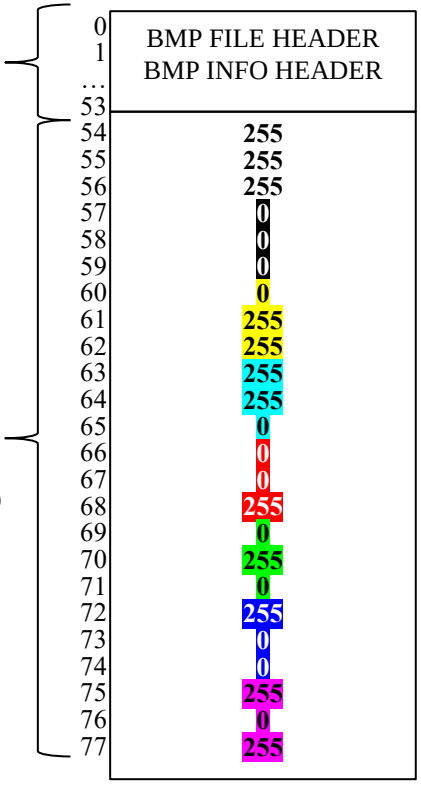
使用RGB编码像素

- 一个像素使用三元组 (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



从底向上、从左向右顺序存储三元组

[255 255 255
0 0 0
0 255 255
255 255 0
0 0 255
0 255 0
255 0 0
255 0 255]



该BMP图片一共有54+24 = 78 字节

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

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

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  "io/ioutil"
func main() {
        p, _ := ioutil.ReadFile("./panda.bmp")
        for i := 54; i < len(p); i++ {
                p[i] = 255 - p[i]
        }
        ioutil.WriteFile("./darkPanda.bmp", p, 0666)
}
```

ioutil.ReadFile

- Golang “io/ioutil” 包提供的函数
- 输入一个字符串格式的文件名
- 返回保存文件内容的字节切片

p, _ := ioutil.ReadFile(“./panda.bmp”)

- “./panda.bmp”：图片文件名
- 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 ioutil.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  "io/ioutil"
func main() {
        p, _ := ioutil.ReadFile("./panda.bmp")
        for i := 54; i < len(p); i++ {
                p[i] = 255 - p[i]
        }
        ioutil.WriteFile("./darkPanda.bmp", p, 0666)
}
```

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  "io/ioutil"
func main() {
        p, _ := ioutil.ReadFile("./panda.bmp")
        for i := 54; i < len(p); i++ {
                p[i] = 255 - p[i]
        }
        ioutil.WriteFile("./darkPanda.bmp", p, 0666)
}
```

p[54]的含义是什么？

- 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,…]

问题：如何把修改后的p保存到文件中？

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  "io/ioutil"
func main() {
        p, _ := ioutil.ReadFile("./panda.bmp")
        for i := 54; i < len(p); i++ {
                p[i] = 255 - p[i]
        }
        ioutil.WriteFile("./darkPanda.bmp", p, 0666)
}
```

用ioutil.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

- 问题：0666的含义是什么？
  - 访问权限
    - 该文件拥有者、拥有者所在的组、其他用户读、写、执行该文件的权限
  - 所有用户均可读、可写、不可执行
    - -rw-rw-rw-
    - 0666 = 0 110 110 110

Owner			Group			Others		
r	w	e	r	w	e	r	w	e
1	1	-	1	1	-	1	1	-

```
package main
import  "io/ioutil"
func main() {
        p, _ := ioutil.ReadFile("./panda.bmp")
        for i := 54; i < len(p); i++ {
                p[i] = 255 - p[i]
        }
        ioutil.WriteFile("./darkPanda.bmp", p, 0666)
}
```

如果把255改成230会怎样？

产生无符号整数溢出。计算结果要属于 [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
- 230 - 240 = **(-10)** % 256 = **502**%256 = 246 （错误）
  - 注：(-10) = 111110110 = 502 （9位）

溢出了

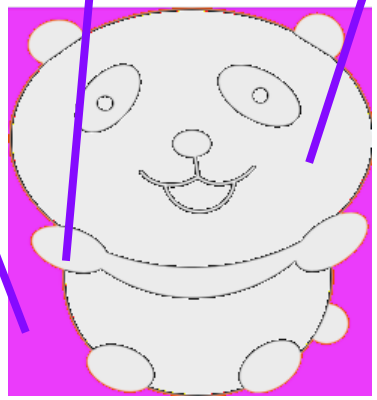
修改前的 **p**

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

修改后的 **p**

[…,246,54,230,…,230,230,230,…,231,231,231,…]

```
for i := 54; i < len(p); i++ {
        p[i] = 230 - p[i]
}
```



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？

产生无符号整数溢出。计算结果需要在 [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）

修改前的 **p**

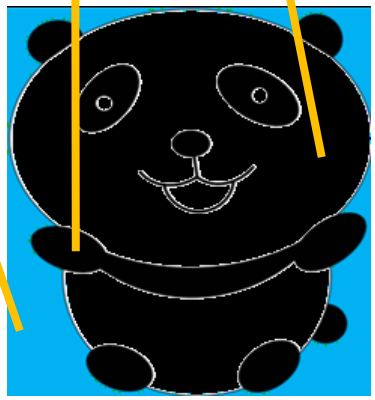
[..,240,176,0,…,0, 0, 0, …,255,255,255,…]

```
for i := 54; i < len(p); i++ {
        p[i] = 1 + p[i]
}
```

修改后的 **p**

Overflow

[…,241,177,1,…,1,1,1,…,0,0,0,…]



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

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
  - 问题：给定一个字节x，如何反转它的最低位？

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

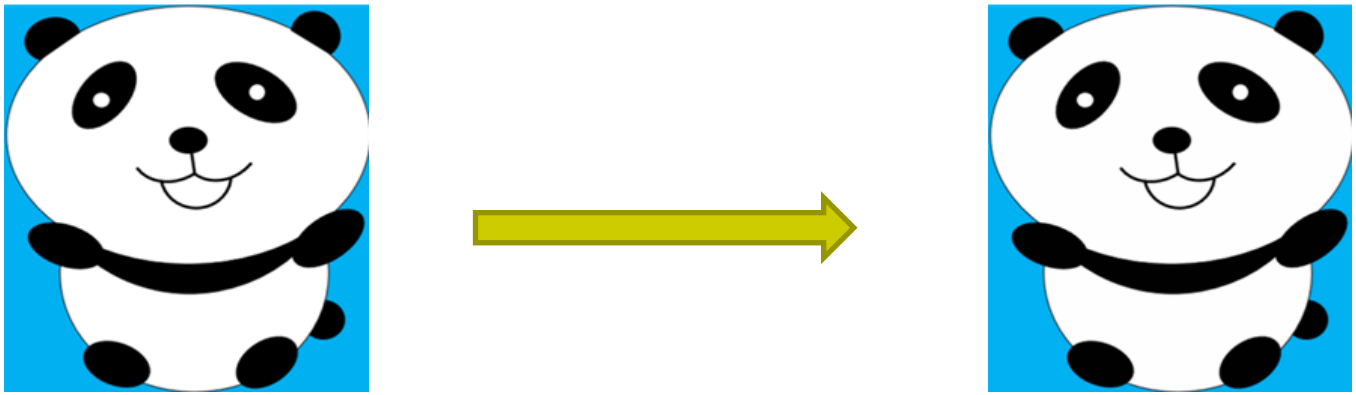
练习2：对于每个像素，反转每个字节的最高位

- 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
    - x = 240 = 11110000
    - 11110000 XOR 00000001 = 11110001

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不会有视觉差异

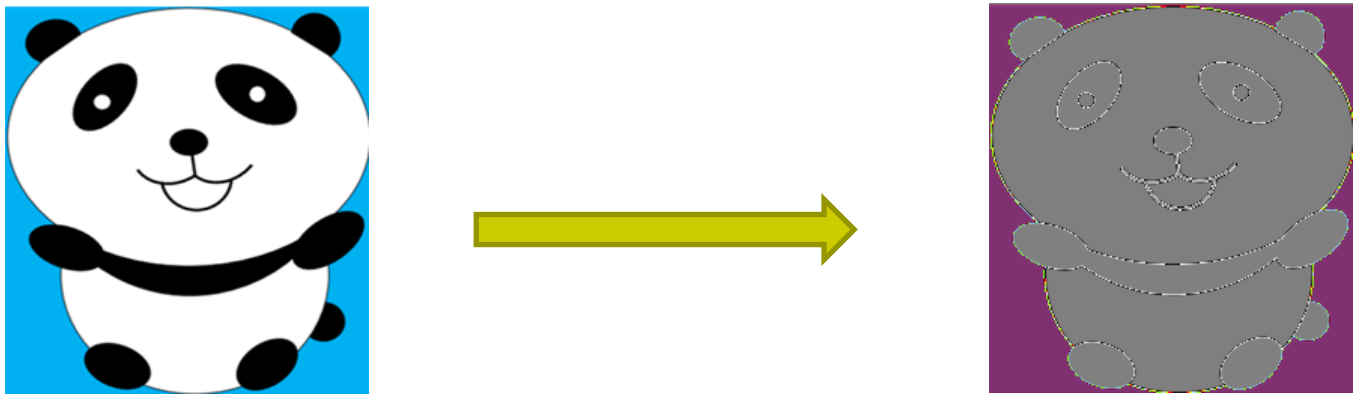
```
package main
import  "io/ioutil"
func main() {
        p, _ := ioutil.ReadFile("./panda.bmp")
        for i := 54; i < len(p); i++ {
                p[i] = p[i] ^ 0x1
        }
        ioutil.WriteFile("./darkPanda.bmp", p, 0666)
}
```



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  "io/ioutil"
func main() {
        p, _ := ioutil.ReadFile("./panda.bmp")
        for i := 54; i < len(p); i++ {
                p[i] = p[i] ^ 0x80
        }
        ioutil.WriteFile("./darkPanda.bmp", p, 0666)
}
```

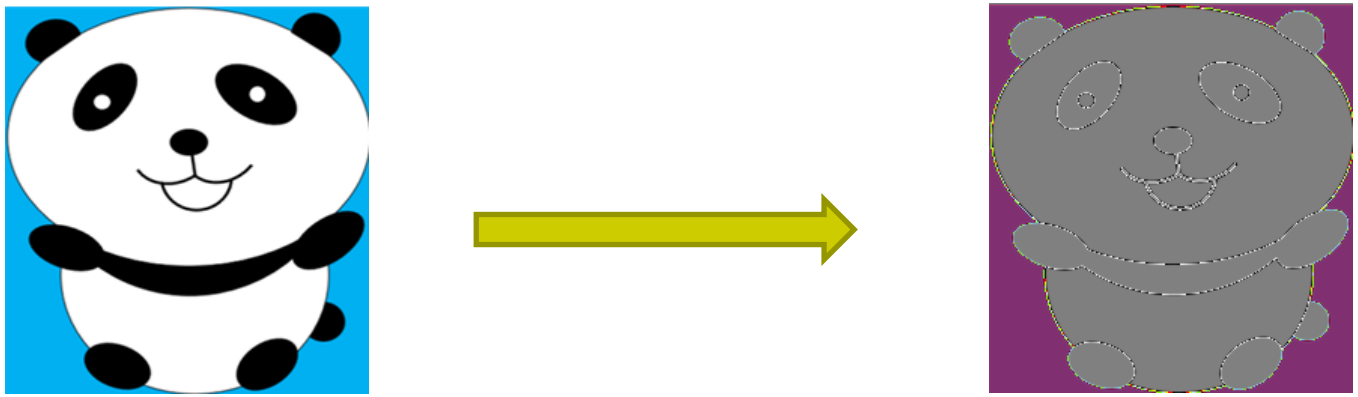


问题：如何反转每个字节的最低2位和最高2位？

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  "io/ioutil"
func main() {
        p, _ := ioutil.ReadFile("./panda.bmp")
        for i := 54; i < len(p); i++ {
                p[i] = p[i] ^ 0x80
        }
        ioutil.WriteFile("./darkPanda.bmp", p, 0666)
}
```



问题：如何反转每个字节的最低2位和最高2位？ XOR 0x3和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"
    "io/ioutil"
    "os"
)
const (
    headerSize = 54                  // standard size of header
    fileMode   = 0666                // Access permissions
    maxDepth   = 255                 // maximum color depth values
    srcImg     = "./panda.bmp"       // input image name
    destImg    = "./darkPanda.bmp" // output image name
)
```

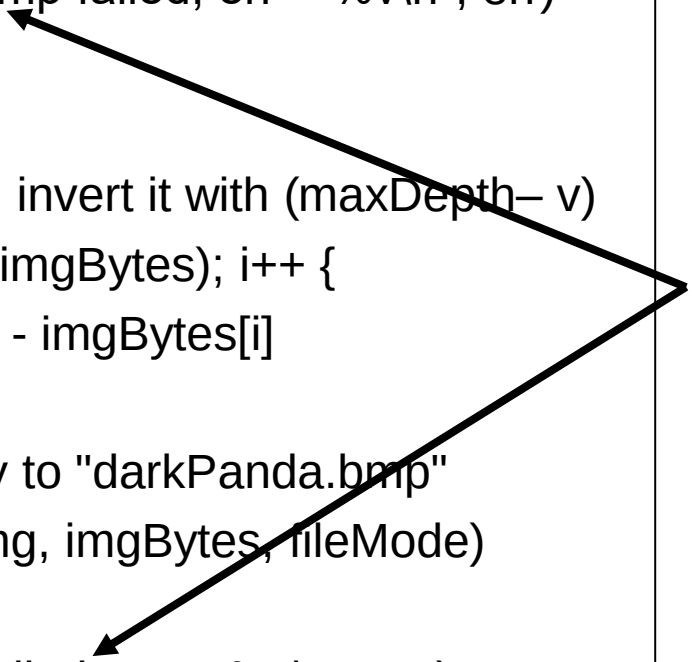
示例1：良好编码习惯-2

```
func main() {
    // Read "panda.bmp"
    imgBytes, err := ioutil.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 = ioutil.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 := ioutil.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 = ioutil.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
```

Mask mechanism
掩码机制

```
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

隐藏程序：把文本隐藏到BMP图片

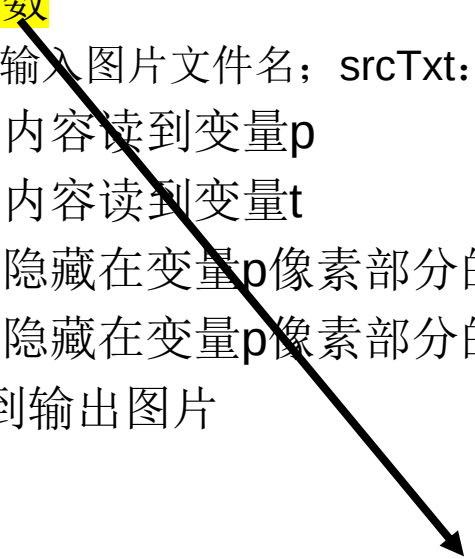
- **输入：一个文本文件、一个图片文件**
- **输出：一个修改后的图片文件**
- **步骤：**

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

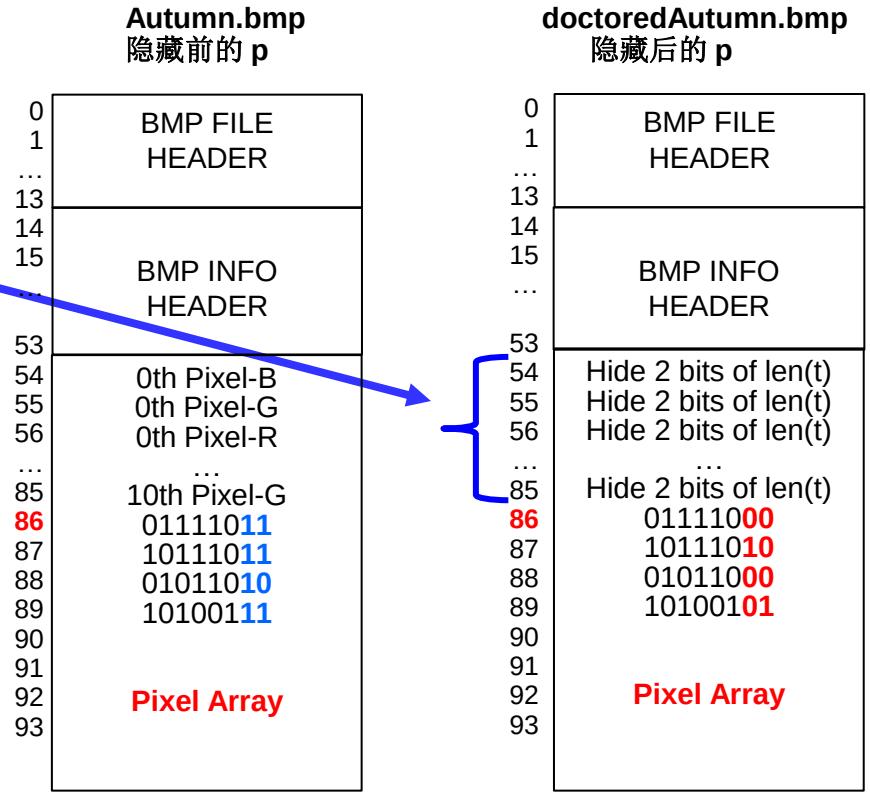
隐藏程序：把文本隐藏到BMP图片

- **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**

Autumn.bmp
隐藏前的 p

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

doctoredAutumn.bmp
隐藏后的 p

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

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

Autumn.bmp
隐藏前的 p

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

doctoredAutumn.bmp
隐藏后的 p

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

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]保存第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 is 4
  - p[S+T:S+T+C] is 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	BMP FILE HEADER
1	
…	
13	
14	BMP INFO HEADER
15	
…	
53	
54	0th Pixel-B
55	0th Pixel-G
56	0th Pixel-R
…	…
85	10th Pixel-G
86	01111011
87	10111011
88	01011010
89	10100111
90	
91	
92	**Pixel Array**
93	

doctoredAutumn.bmp
隐藏后的 p

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

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]，i 从0 到len(t)

```
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]

Autumn.bmp
隐藏前的 p

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

doctoredAutumn.bmp
隐藏后的 p

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

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 = ioutil.WriteFile(destImage, p, 0666)
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"
    "io/ioutil"
    "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

不要修改“init” 函数

hide-0.go – part 2

```
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 := ioutil.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 := ioutil.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)
    }
```

不要修改该代码块

hide-0.go – part 3

```
    // 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)
    }
    // save the modified p to destImage
    err = ioutil.WriteFile(destImage, p, 0666)
    if err != nil {
        fmt.Printf("Write doctored image failed, err = %v\n", err)
        os.Exit(1)
    }
}
```

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
}
```

不要修改该代码块

评分标准

- 采用5档分制，如下表所示。
- 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。

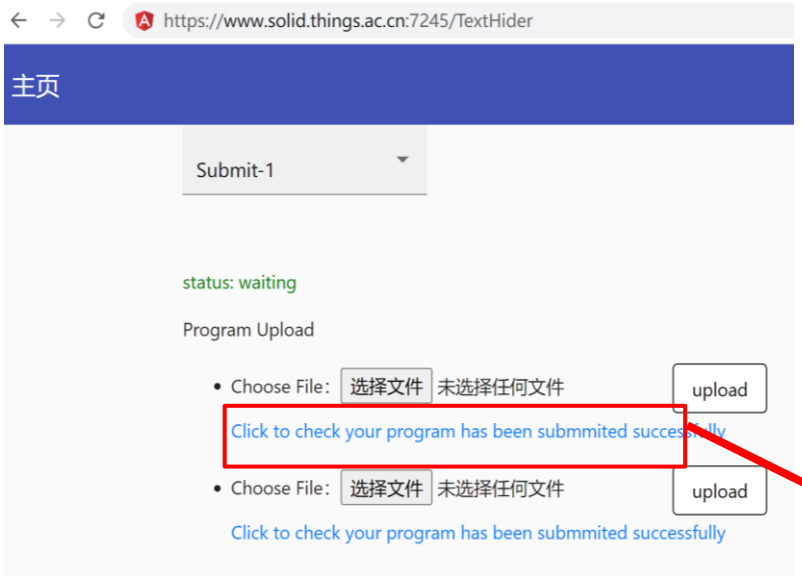
分数	评判标准
4	独立完成且正确，少量使用他人的素材（有标注致谢），有创意
3	独立完成且大部分正确，少量使用他人的素材（有标注致谢）
2	完成且基本正确（如代码运行但结果有错），并标注致谢了使用他人的素材
1	截止时间前提交，但有明显错误（如代码不能运行）或缺失 50%标注致谢
0	未在截止时间前提交，或抄袭

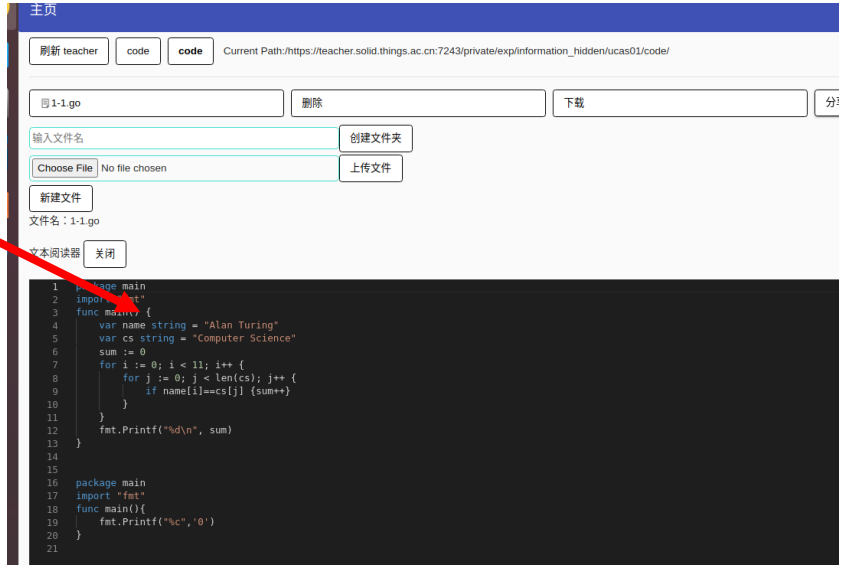
框架代码下载：
https://teacher.solid.things.ac.cn:7243/public/materials/TextHider/project.zip

实验代码提交：https://www.solid.things.ac.cn:7245/TextHider

代码提交

- https://www.solid.things.ac.cn:7245/TextHider





如何检查是否提交成功：点击红框中的链接，如果能显示代码（如右图），则提交成功。

关键时间节点

- 2022.5.13　中午12:00　第一次代码提交截止时间
- 2022.5.20　中午12:00　第二次代码提交截止时间
- 2022.5.27　中午12:00　第三次代码提交截止时间

- 最终取三次最高分