



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

CS101

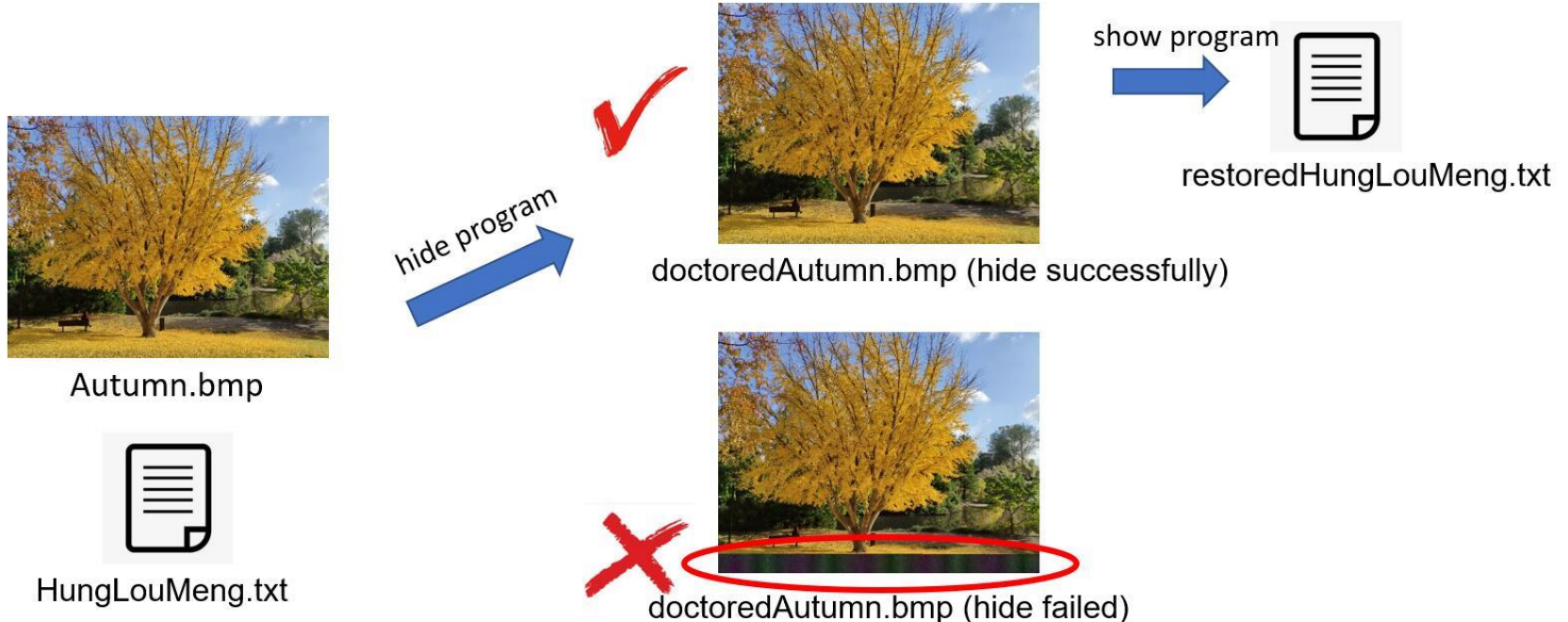
Lab Slides Text Hider

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

Objective

Read textbook 7.2

- Develop hide-0.go: Hide the content of a text file in a 24-bit BMP image file
 - The doctored image file shows no visible difference from the original image file
- Develop show-0.go: Recover the content of the text file from the doctored BMP image file.
 - The recovered text file should have contents identical to the original image file
- The two programs should work for other text and 24-bit BMP image files.
- Good programming practice
- Independent work



**How to represent an image in a computer
(in memory or in a file)?**

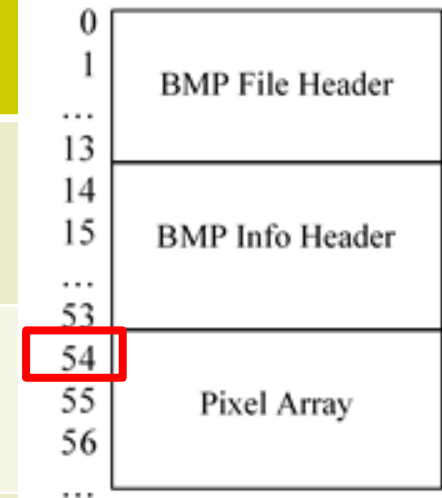
As a byte slice:
a slice of elements, where each element is of
byte type, i.e., uint8

The byte slice contains metadata and pixels

BMP file format

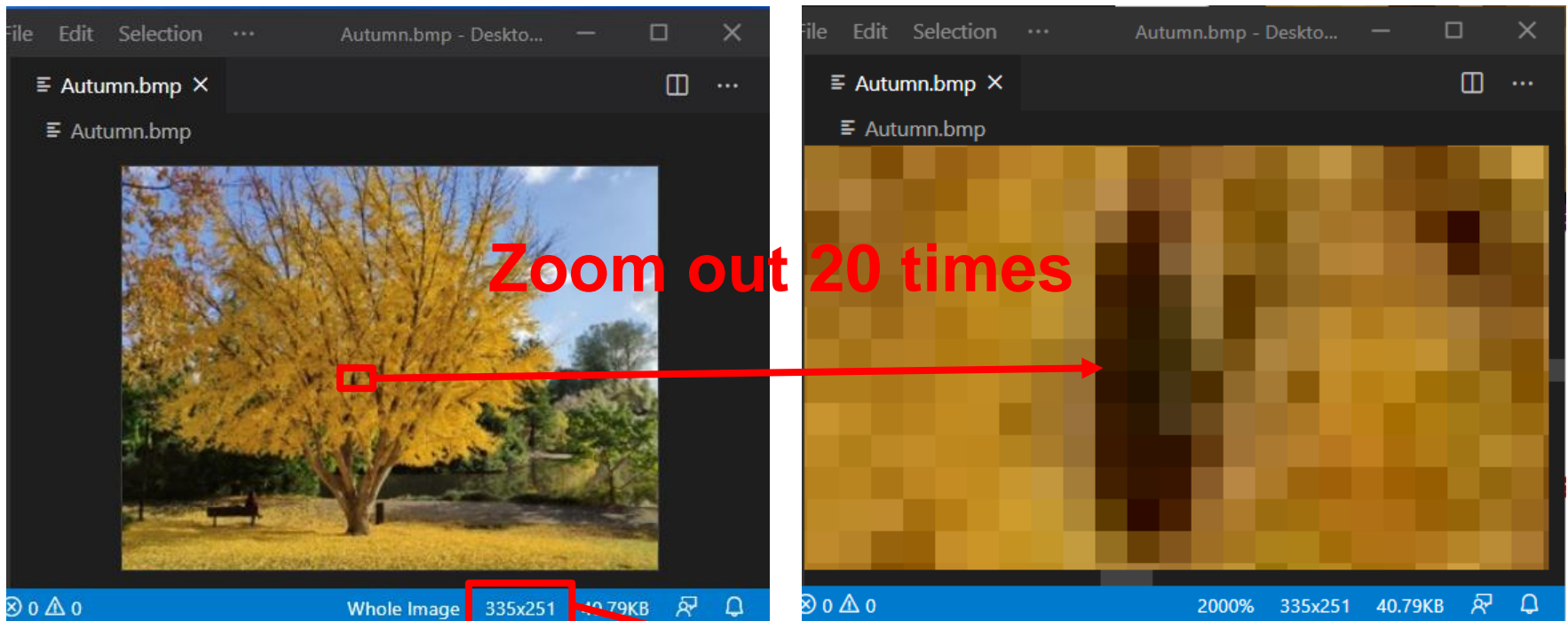
- In the experiment, each pixel of the BMP file is encoded in 3 bytes (24 bits).
- The pixel array for actual image data starts at address byte 54.

Structure name	Size	Purpose
File Header	14 bytes	General information about the BMP file, such as file size of the image.
BMP Info Header	40 bytes	Detailed information of the BMP file, such as height and width of the image.
Pixel Array	3*Height*Width	The actual values of the pixels



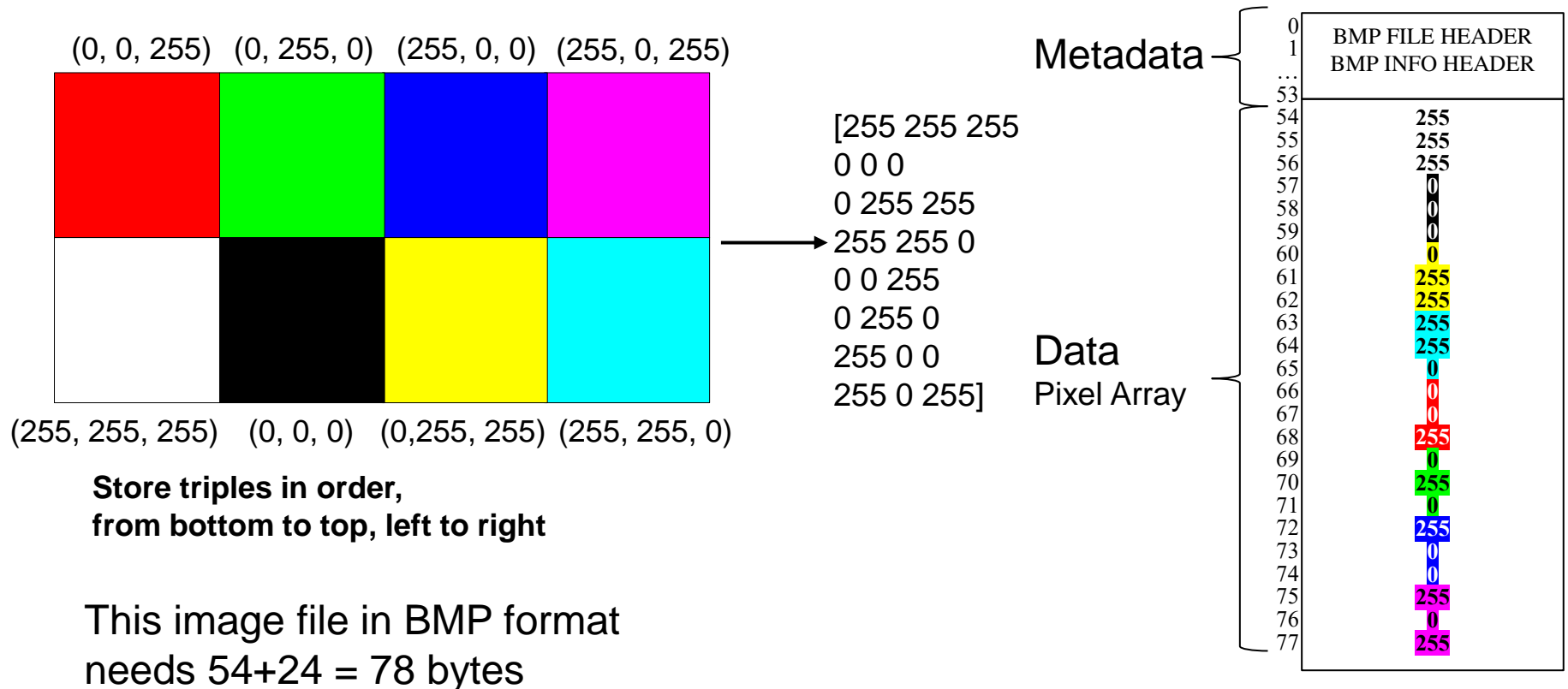
Images are made up of pixels

- A pixel (picture element) represents a point of an image
- Open Autumn.bmp with VS Code



Pixel encoded in RGB color model

- A pixel is encoded with a triple (b, g, r)
 - b, g, r are uint8 integers (0 - 255) representing blue, green and red color depths, respectively
- Example:
 - An image with 2x4 pixels, needing $2 \times 4 \times 3 = 24$ bytes



Example1: Invert the color of panda.bmp

- Algorithm
 - Read panda.bmp into variable p
 - pixel array starts at p[54]
 - For each v in pixel array, invert it with $(255 - v)$
 - Write the inverted pixel array to darkPanda.bmp



Invert the colors of each pixel



Example1: Invert the color of panda.bmp

- Algorithm
 - Read panda.bmp into variable p
 - pixel array starts at p[54]
 - For each v in pixel array, invert it with $(255 - v)$
 - Write the inverted pixel array to darkPanda.bmp

```
package main                // darkPanda.go
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)
}
```


Example1: Invert the color of panda.bmp

● Algorithm

- Read panda.bmp into variable p
 - pixel array starts at p[54]
- For each v in pixel array, invert it with $(255 - v)$
- Write the inverted pixel array to darkPanda.bmp

```
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

- A function provided by Golang package "io/ioutil"
- Accepts a file name in string
- Returns the file content in byte slice

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

- `"./panda.bmp"`: image file name
- `p`: byte slice of panda.bmp
- `_`: blank identifier, ignore the returned value (errors of ioutil.ReadFile)

Example1: Invert the color of panda.bmp

- Algorithm

- Read panda.bmp into variable p
 - pixel array starts at p[54]
- For each v in pixel array, invert it with $(255 - v)$
- Write the inverted pixel array to darkPanda.bmp

```
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)
}
```

Example1: Invert the color of panda.bmp

Original p

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



Modified p

[..., 15, 79, 255, ..., 255, 255, 255, ..., 0, 0, 0, ...]



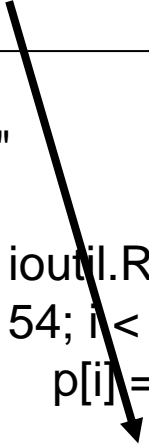
```
for i := 54; i < len(p); i++ {  
  p[i] = 255 - p[i]  
}
```



Example1: Invert the color of panda.bmp

● Algorithm

- Read panda.bmp into variable p
 - pixel array starts at p[54]
- For each v in pixel array, invert it with $(255 - v)$
- Write the inverted pixel array to darkPanda.bmp



```
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)
}
```

Example1: Invert the color of panda.bmp

- Question: What's the meaning of 0666?
 - Access permissions
 - Rights to read, write, and execute a file by the owner of the file, by the group the owner belonging to, and by other users
 - Every user can read and write, but cannot execute
- -rw-rw-rw-
- 0666 = 0110110110

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)
}
```

Example1: How about 230-v?

Could cause unsigned integer overflow. Result should be in $[0, 255]$

- The operations $+$, $-$, $*$, and \ll are computed modulo 2^n (n is the bit width of the unsigned integer's type).
 - From https://golang.org/ref/spec#Integer_overflow
- $230 - 240 = (-10) \% 256 = 502 \% 256 = 246$ (**wrong result**)
 - Note: $(-10) = 111110110 = 502$ (9 bits)

Original p

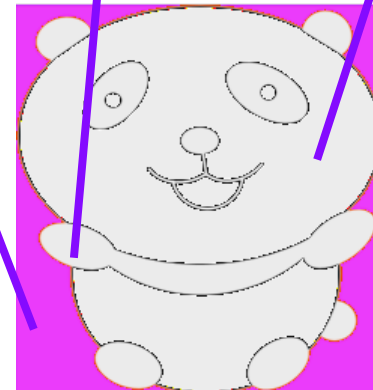
[..., 240, 176, 0, ..., 0, 0, 0, ..., 255, 255, 255, ...]



Modified p

[..., 246, 54, 230, ..., 230, 230, 230, ..., 231, 231, 231, ...]

Overflow



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



Example1: How about 1+v?

Could cause unsigned integer overflow. Result should be in $[0, 255]$

- The operations $+$, $-$, $*$, and \ll are computed modulo 2^n (n is the bit width of the unsigned integer's type).
 - From https://golang.org/ref/spec#Integer_overflow
- $1 + 255 = \textcolor{red}{(256)} \% 256 = 0$ (**wrong result**)
 - Note: $(256) = \textcolor{red}{1}00000000$ (9 bits)

Original p

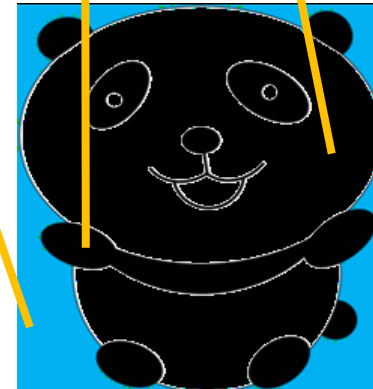
[..., 240, 176, 0, ..., 0, 0, 0, ..., 255, 255, 255, ...]



Modified p

[..., 241, 177, 1, ..., 1, 1, 1, ..., 0, 0, 0, ...]

Overflow



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



In-Class Exercises:

change the darkPanda.go program

- Exercise 1: invert the least significant bit of every byte of the pixel array
- Exercise 2: invert the most significant bit of every byte of the pixel array
- Recall: XOR operator
 - xor 1 (invert bit)
 - $1 \wedge 1 = 0$
 - $0 \wedge 1 = 1$
 - xor 0 (keep bit unchanged)
 - $1 \wedge 0 = 1$
 - $0 \wedge 0 = 0$
- Break to let the students do the exercises

In-Class Exercises

Exercise1: invert the least significant bit of a byte

Exercise2: invert the most significant bit of a byte

- Recall: XOR operator

- XOR 1 (invert bit)

- $1 \wedge 1 = 0$
- $0 \wedge 1 = 1$

- XOR 0 (keep bit unchanged)

- $1 \wedge 0 = 1$
- $0 \wedge 0 = 0$

- A byte XOR 00000001

- $240 = 11110000$
- $11110000 \text{ XOR } 00000001 = 11110001$

The higher bits are unchanged

The least significant bit inverted

Exercise1: invert the least significant bit of a byte

- Changing the least significant bit of every byte in the pixel array does not produce visible changes

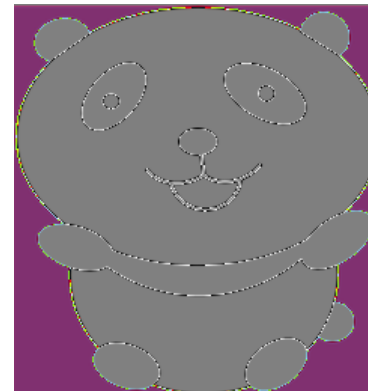
```
package main
import "io/ioutil"
func main() {
    p, _ := ioutil.ReadFile("./panda.bmp")
    for i := 54; i < len(p); i++ {
        p[i] = p[i] ^ 0x1
    }
    ioutil.WriteFile("./darkPanda.bmp", p, 0666)
}
```



Exercise1: invert the most significant bit of a byte

- Changing the most significant bit of every byte in the pixel array does produce visible change

```
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)
}
```



Example1 in good programming practice – part 1

- Use descriptive names
 - "headerSize"
- Avoid magic numbers
 - Define "maxDepth" instead of using "255" directly
- Put constant definitions up front

```
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
)
```

Example1 in good programming practice – part 2

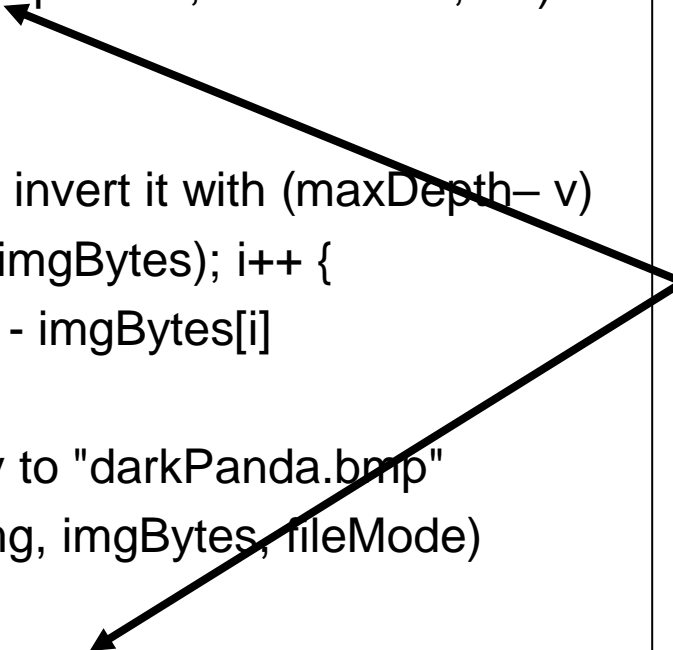
```
func main() {  
    // Read "panda.bmp"  
    imgBytes, err := ioutil.ReadFile(srcImg)  
    if err != nil {  
        fmt.Printf("read panda.bmp failed, err = %v\n", err)  
        os.Exit(1)  
    }  
    // For each v in pixel array, invert it with (maxDepth - v)  
    for i := headerSize; i 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

- Use comments to document the code
- Avoid repetitive code
 - For loop

Example1 in good programming practice – part 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("save image failed, err = %v\n", err)  
        os.Exit(1)  
    }  
}
```

If error occurs, you
need to fix the error
and retry



Example 2:

Replacing the least significant 2 bits of a byte

- Input: 001111**11**, 001010**10**; Output: 001111**10**
- Code explained with the corresponding operations

```
x := byte(63)    // assign 63=00111111 to variable x
v := byte(42)    // assign 42=00101010 to variable v
v = v & 0x3       // bitwise AND to retain the right-most 2 bits of v
x = x & 0xFC      // bitwise AND to clear the right-most 2 bits of x
                  // and retaining the leftmost 6 bits
x = x | v         // bitwise OR to get the final result
```

Mask
mechanism

x = 00111111		Given input
v = 00101010		Given input
v = 00101010 & 00000011	= 000000 10	Bitwise AND
x = 00111111 & 11111100	= 001111 00	Bitwise AND
x = 001111 00 000000 10	= 001111 10	Bitwise OR

Note: 0x3 = 000000**11**; 0xFC = 111111**00**

Hide program: Hide text in a BMP image

- **Input:** A text file and an image file.
- **Output:** A doctored image file.
- **Steps:**
 1. Get command line arguments
 - srcImage: input image name; srcTxt: input text name; destImage: output image name
 2. Read the input image into variable p // p for picture
 3. Read the input text into variable t // t for text
 4. Hide the length of the input text in the first 32 bytes of Pixel Array
 5. Hide the input text in variable p in the remaining bytes of the Pixel Array
 6. Write p to the output image

Hide program: Hide text in a BMP image

- **Input:** A text file and an image file.
- **Output:** A doctored image file.
- **Steps:**
 1. **Get command line arguments**
 - srcImage: input image name; srcTxt: input text name; destImage: output image name
 2. Read the input image into variable p // p for picture
 3. Read the input text into variable t // t for text
 4. Hide the length of the input text in the first 32 bytes of Pixel Array
 5. Hide the input text in variable p in the remaining bytes of the Pixel Array
 6. Write p to the output image



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

Get input image srcImage, input text srcTxt and output image destImage

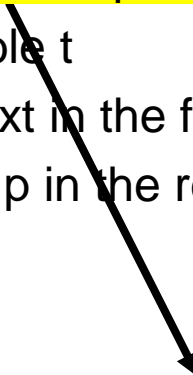
go run hide.go **-i Autumn.bmp** **-t HungLouMeng.txt** **-o doctoredAutumn.bmp**

```
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 name.")  
    flag.StringVar(&srcTxt, "t", "", "input text name.")  
    flag.StringVar(&destImage, "d", "", "output doctored image name.")  
}  
  
func main() {  
    // parse command line arguments  
    flag.Parse()  
    if srcImage == "" || srcTxt == "" || destImage == "" {  
        flag.PrintDefaults()  
        os.Exit(1)  
    }  
}
```

Get command line arguments using “flag” package (**Don't modify the code**).

Hide algorithm: Hide text in a BMP image

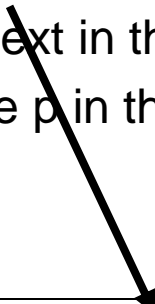
- **Input:** A text file and an image file.
- **Output:** A doctored image file.
- **Steps:**
 1. Get command line arguments
 - srcImage: input image name; srcTxt: input text name; destImage: output image name
 2. Read the input image into variable p // p for picture
 3. Read the input text into variable t // t for text
 4. Hide the length of the input text in the first 32 bytes of Pixel Array
 5. Hide the input text in variable p in the remaining bytes of the Pixel Array
 6. Write p to the output image



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

Hide algorithm: Hide text in a BMP image

- **Input:** A text file and an image file.
- **Output:** A doctored image file.
- **Steps:**
 1. Get command line arguments
 - srcImage: input image name; srcTxt: input text name; destImage: output image name
 2. Read the input image into variable p // p for picture
 3. Read the input text into variable t // t for text
 4. Hide the length of the input text in the first 32 bytes of Pixel Array
 5. Hide the input text in variable p in the remaining bytes of the Pixel Array
 6. Write p to the output image



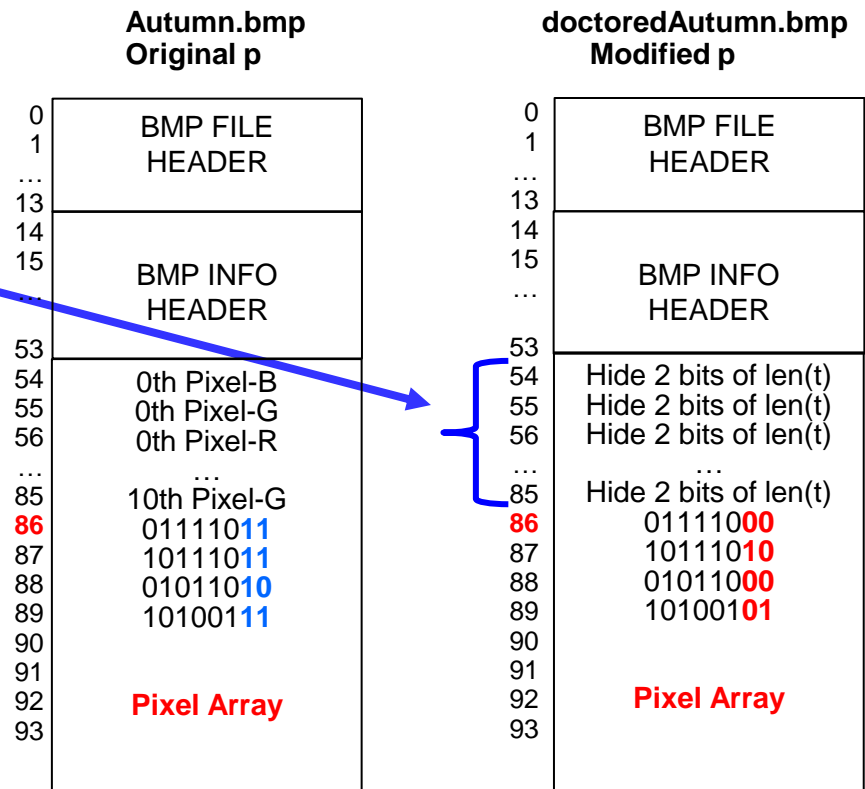
```
t, err := ioutil.ReadFile(srcTxt)
if err != nil {
    fmt.Printf("Read text file failed, err = %v\n", err)
    os.Exit(1)
}
```

Hide algorithm: Hide text in a BMP image

- **Input:** A text file and an image file.
- **Output:** A doctored image file.
- **Steps:**
 1. Get command line arguments
 - srcImage: input image name; srcTxt: input text name; destImage: output image name
 2. Read the input image into variable p // p for picture
 3. Read the input text into variable t // t for text
 4. Hide the length of the input text in the first 32 bytes of Pixel Array
 5. Hide the input text in variable p in the remaining bytes of the Pixel Array
 6. Write p to the output image

How to hide the length of a text file in a picture?

- t is the byte array of the input text
- p is the byte array of the input image
- Length $\text{len}(t)$ is a 64-bit integer
 - Hide every 2 bits in a byte of p
 - Need 32 bytes
 - $S = 54, T = 32$
- $\text{modify}(\text{len}(t), p[S:S+T], T)$
to hide $\text{len}(t)$ in $p[54:86]$



How to hide the length of a text file in a picture?

- t is the byte array of the input text
- p is the byte array of the input image
- Length $\text{len}(t)$ is a 64-bit integer



- Hide every 2 bits in a byte of p
- Need 32 bytes
- $S = 54, T = 32$
- $\text{modify}(\text{len}(t), p[S:S+T], T)$
to hide $\text{len}(t)$ in $p[54:86]$

```
func modify(txt int, pix []byte, size int) {
```

```
    for i := 0; i < size; i++ {
```

```
        replace last 2 bits of pix[i]  
        with the last 2 bits of txt
```

```
        repeat with the next 2 bits of txt
```

```
    }
```

```
}
```

See example 2 in this lecture

Autumn.bmp
Original 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
Modified p

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

How to hide the length of a text file in a picture?

- t is the byte array of the input text
- p is the byte array of the input image
- Length $\text{len}(t)$ is a 64-bit integer



- Hide every 2 bits in a byte of p
- Need 32 bytes
- $S = 54, T = 32$

- $\text{modify}(\text{len}(t), p[S:S+T], T)$
to hide $\text{len}(t)$ in $p[54:86]$

```
func modify(txt int, pix []byte, size int) {
```

```
    for i := 0; i < size; i++ {
```

```
        replace last 2 bits of pix[i]  
        with the last 2 bits of txt
```

```
        repeat with the next 2 bits of txt
```

```
    }
```

```
}
```

Use right shift: shift txt right 2 bit

Autumn.bmp
Original p


0	BMP FILE
1	HEADER
...	
13	
14	BMP INFO
15	HEADER
...	
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
Modified p

0	BMP FILE
1	HEADER
...	
13	
14	BMP INFO
15	HEADER
...	
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	

Hide algorithm: Hide text in a BMP image

- **Input:** A text file and an image file.
- **Output:** A doctored image file.
- **Steps:**
 1. Get command line arguments
 - srcImage: input image name; srcTxt: input text name; destImage: output image name
 2. Read the input image into variable p // p for picture
 3. Read the input text into variable t // t for text
 4. Hide the length of the input text in the first 32 bytes of Pixel Array
 5. Hide the input text in variable p in the remaining bytes of the Pixel Array
 6. Write p to the output image



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

How to hide the contents of a text file in a picture?

- t is the byte array of the input text
- p is the byte array of the input image
- $t[0]$ holds the 1st character 'H' = 72
- $\text{modify}(\text{int}(t[0]), p[S+T:S+T+C], C)$

where

- $t[0]$ is '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	011110 11
87	101110 11
88	010110 10
89	101001 11



Modified $p[86:90]$

	011110 00
	101110 10
	010110 00
	101001 01



Autumn.bmp
Original 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	011110 11
87	101110 11
88	010110 10
89	101001 11
90	Pixel Array
91	
92	
93	
93	

doctoredAutumn.bmp
Modified p

0	BMP FILE HEADER
1	
...	
13	
14	BMP INFO HEADER
15	
...	
53	
54	Hide 2 bits of $\text{len}(t)$
55	Hide 2 bits of $\text{len}(t)$
56	Hide 2 bits of $\text{len}(t)$
...	...
85	Hide 2 bits of $\text{len}(t)$
86	011110 00
87	101110 10
88	010110 00
89	101001 01
90	Pixel Array
91	
92	
93	
93	

How to hide the contents of a text file in a picture?

- t is the byte array of the input text
- p is the byte array of the input image
- To hide all $t[i]$, $i = 0$ to $\text{len}(t)$

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

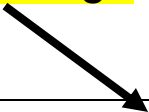
- Each iteration hides $t[i]$ in $p[S+T+(i*C):S+T+(i*C)+C]$
 - Where $S = 54$, $T = 32$, $C = 4$
- That is, $t[i]$ is hidden in $p[86+(i*4)]$, $p[86+(i*4)+1]$, $p[86+(i*4)+2]$, $p[86+(i*4)+3]$
- E.g., $t[1]='A'$, is hidden in $p[90:94]$



Autumn.bmp Original p		doctoredAutumn.bmp Modified p	
0	BMP FILE	0	BMP FILE
1	HEADER	1	HEADER
...		...	
13		13	
14	BMP INFO	14	BMP INFO
15	HEADER	15	HEADER
...		...	
53		53	
54	0th Pixel-B	54	Hide 2 bits of len(t)
55	0th Pixel-G	55	Hide 2 bits of len(t)
56	0th Pixel-R	56	Hide 2 bits of len(t)
...		...	
85	10th Pixel-G	85	Hide 2 bits of len(t)
86	01111011	86	01111000
87	10111011	87	10111010
88	01011010	88	01011000
89	10100111	89	10100101
90		90	
91		91	
92	Pixel Array	92	Pixel Array
93		93	

Hide algorithm: Hide text in a BMP image

- **Input:** A text file and an image file.
- **Output:** A doctored image file.
- **Steps:**
 1. Get command line arguments
 - srcImage: input image name; srcTxt: input text name; destImage: output image name
 2. Read the input image into variable p // p for picture
 3. Read the input text into variable t // t for text
 4. Hide the length of the input text in the first 32 bytes of Pixel Array
 5. Hide the input text in variable p in the remaining bytes of the Pixel Array
 6. Write p to the output image



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

Check results

- Write the complete hide-0.go and show-0.go
- Execute and display result

```
> 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
```

- Original image and doctored image



Autumn.bmp



doctoredAutumn.bmp

- Original text and restored text. The following picture shows the correct result

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

hide-0.go – part 1

```
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
```

Complete "modify" function

```
// modify hides an integer to a byte slice
func modify(txt 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 txt
        // the next iteration repeats with the next 2 bits of txt
    }
}
```

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

DON'T modify "init" function

```
// 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")
}
```


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)  
    }  
}
```

DON'T modify this block

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

```
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
}
```

DON'T modify this block