



Systems Thinking

Data Abstractions, Control Abstractions

数据抽象与控制抽象

zxu@ict.ac.cn

zhangjialin@ict.ac.cn

Outline

- What is systems thinking?
- Three objectives of systems thinking
- Abstraction
 - What is abstraction?
 - The COG properties of abstraction
 - One abstraction for many scenarios
 - **Data abstractions 数据抽象**
 - Representing numbers, characters
 - Review of bit, byte, character, integer, array, and slice, as well as struct
 - Pointers, files
 - **Control abstractions 控制抽象**
 - Precedence, sequence, selection, loop
 - Function and the four segments of text, data, stack, and heap
- Modularization
- Seamless transition

These slides acknowledge sources for additional data not cited in the textbook

快排实验凝聚了前人智慧

- 什么是**抽象**？系统的本质精髓
 - 希望同学们初步领悟算法、程序、系统三个概念的异同
 - 实例：**quicksort**算法、班级快排程序、班级快排计算机
 - 不同于中学编写快速排序的程序
- Enable people to follow the way, without them having to understand [the internals of] it. **民可使由之不可使知之**
— 孔子《论语·泰伯篇》 Confucius (551–479 BCE)
- 算法是计算过程（步骤序列）的一组规则，满足五点：
有穷性、**确定性**（**明确性**）、输入、输出、**可行性**
— Don Knuth 高德纳，1968
- Inside every large program is **a small program** struggling to get out.
— Tony Hoare, 1970
- Inside every large program, there is ~~a small program~~ **an algorithm** trying to get out.
Here, algorithm also means specification or high-level design of a system
— Leslie Lamport, 2018



快排实验凝聚了前人智慧

- 什么是抽象？系统的本质精髓
 - 领悟算法、程序、系统三个概念的异同
 - 实例：**quicksort**算法、班级快排程序、班级快排计算机
- Inside every large program is **a small program** struggling to get out.
— Tony Hoare, 1970
- 1959年莫斯科州立大学访问学生时产生快排思想；
1960年在Elliott Brothers公司为Elliott 803计算机开发排序程序时，发明并实现快排算法

“**Very difficult to explain**” 很难向他人说明快排算法

1961年发表，非常简洁易懂（只有半页，6行伪代码）

Hoare, C. A. R. (1961). "Algorithm 64: Quicksort". Comm. ACM. 4 (7): 321

为什么？

— Tony Hoare, 1980图灵奖演说：皇帝的旧衣

```
ALGORITHM 64
QUICKSORT
C. A. R. HOARE
Elliott Brothers Ltd., Borehamwood, Hertfordshire, Eng.

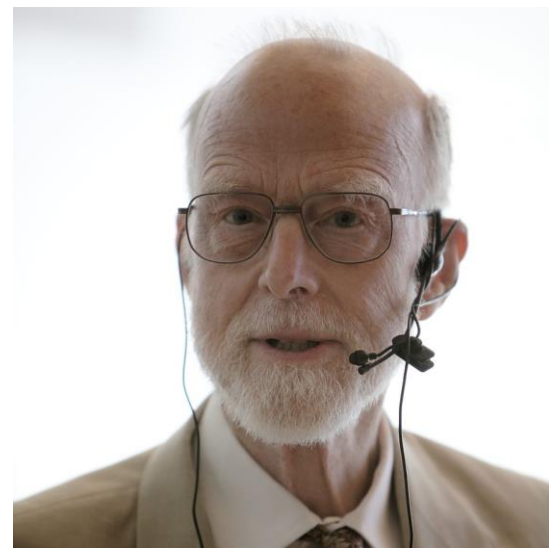
procedure quicksort (A,M,N); value M,N;
    array A; integer M,N;
comment Quicksort is a very fast and convenient method of
sorting an array in the random-access store of a computer. The
entire contents of the store may be sorted, since no extra space is
required. The average number of comparisons made is  $2(M-N) \ln(N-M)$ ,
and the average number of exchanges is one sixth this
amount. Suitable refinements of this method will be desirable for
its implementation on any actual computer;
begin    integer I,J;
        if M < N then begin partition (A,M,N,I,J);
            quicksort (A,M,J);
            quicksort (A, I, N)
        end
    end    quicksort
```

与教科书版本
基本一样

```
ALGORITHM 65
FIND
C. A. R. HOARE
Elliott Brothers Ltd., Borehamwood, Hertfordshire, Eng.

procedure find (A,M,N,K); value M,N,K;
    array A; integer M,N,K;
comment Find will assign to A [K] the value which it would
have if the array A [M:N] had been sorted. The array A will be
partly sorted, and subsequent entries will be faster than the first;
```

Communications of the ACM 321



系统思维旅程（抽象一点不“抽象”）

- 软件（抽象）
 - 数据抽象
 - 比特、字节、字符、整数、数组、切片；指针、**struct**
 - 文件
 - 控制抽象
 - 运算优先级、语句顺序、条件选择语句、**for**循环、函数调用（递归）
- 硬件（模块化，突出信息隐藏原理的抽象）
 - 指令流水线
 - 时序电路
 - 一般时序电路、串行加法器
 - 存储单元、触发器
 - 组合电路
 - 加法器、加法减法器
 - 门电路
 - 半导体电路

系统思维旅程（抽象一点不“抽象”）

- 软件（抽象）
 - 数据抽象
 - 比特、字节、字符、整数、数组、切片；指针、struct
 - 文件
 - 控制抽象
 - 运算优先级、语句顺序、条件选择语句、for循环、函数调用（递归）
- 硬件（模块化，突出信息隐藏原理的抽象）
 - 指令流水线
 - 时序电路
 - 一般时序电路、串行加法器
 - 存储单元、触发器
 - 组合电路
 - 加法器、加法减法器
 - 门电路
 - 半导体电路

请参见

- 中文教科书6.7
 - 英文教科书8.2 Appendix
- SEP课程资源中的
CS101-Chapters 5-7, Appendix, Index.pdf
Golang Programming Notes.pdf

旅程特点：

每一步清晰；
高级抽象如何由低级抽象组合而成；
旅程终点，知道计算机是如何运作的
（例如， $X=Y+Z$ 语句）。

为**信息隐藏**和**个人作品**两个实验做准备

- 信息隐藏（半天~1天）
- 个人作品（半天~3天）

3.4 Data abstractions

- Review data abstractions here in one place
 - To be more systematic
 - To understand why
- Representing numbers, characters
- Review of bit, byte, character, integer, array, slice
- Pointers, files
- 梳理数据抽象：在一个知识单元内比较各种数据类型
 - 字节、整数；比特、字符
 - 数组、切片
 - 指针、文件

如何表示并操作各种数据

- 数据：数字符号组合（英文版教科书第一章）
- 基础知识（英文版教科书第二章）
 - 整数、数组、切片
 - ASCII字符、符号串
- 进阶知识入门介绍
 - 如何表示并操作各种数据类型？
 - 任意长度的整数：斐波那契数 $F(100)=$
 - 全球字符（the world's text）
 - 实数：圆周率 π
 - 比特：信息隐藏实验中，隐藏字节到像素数组
 - 文件：信息隐藏实验中，创建、读、写文件
 - 系统如何支持数据类型？
 - 寻址模式、指针（pointer）、结构（struct）

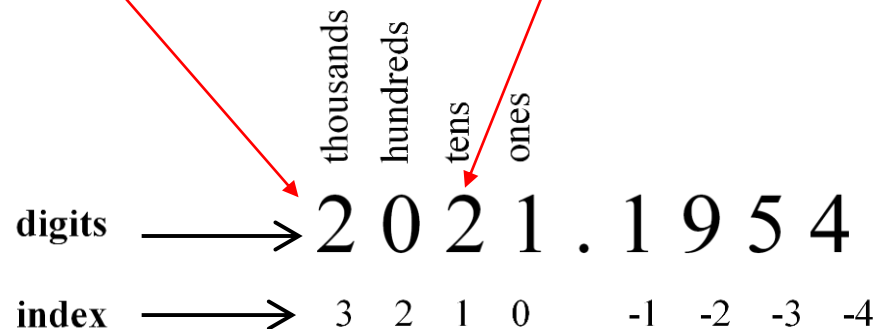
3.4.1 Positional number systems 位值计数法

- Computers (and humans) use positional number systems
- Why?
 - What happens if we use Roman numerals? 罗马计数法: 非位值计数法
 - a non positional number system, i.e., position independent system
 - Q: MMXXI – MCMLIV = ?
 - Value of MMXXI = M + M + X + X + I = 1000+1000+10+10+1=2021
 - Value of MCMLIV = M + CM + L + IV = 1000+900+50+4=1954
 - Note that CM and IV are short-hand symbols, e.g., IV = IIII = I+I+I+I
 - A: Use decimal, 2021-1954 = 67
 - Do MMXXI – MCMLIV = LXVII without decimal arithmetic
 - How about
 - MMXXI + MCMLIV = ?
 - MMXXI × MCMLIV = ?
 - MMXXI ÷ MCMLIV = ?

Roman	M	D	C	L	X	V	I	IV	IX	XL	XC	CD	CM
Decimal	1000	500	100	50	10	5	1	4	9	40	90	400	900

Positional number systems

- Positional number systems make arithmetic much easier
- Consider decimal number $a = 2021.1954$
- Value $a = \sum_{i=-4}^3 (a_i \times 10^i)$, where
 - 10 is **base** 基, i is **index** 索引, $\{0,1,2,3,4,5,6,7,8,9\}$ is **digit set** 数字集
- The key point: The value of a digit depends on both the digit and the position (index) of the digit 数的值由数字和数字所在位置决定
 - The first 2 is at position 3, and the second 2 is at position 1
 - The first 2 represents $a_3 \times 10^3 = 2 \times 10^3 = 2000$
 - The second 2 represents $a_1 \times 10^1 = 2 \times 10^1 = 20$



Examples of positional number systems

- Consider any n-digit number $a = \sum_{i=0}^{n-1} (a_i \times b^i)$, where
 - b is the **base**, i is the **index**, $\{0, \dots, b-1\}$ is the **digit set**
 - i is also called **position**
- There are three positional number systems often used
- Binary 如, $01000001 = 1 \times 2^6 + 1 \times 2^0 = 64 + 1 = 65$
 - $a = \sum_{i=0}^{n-1} (a_i \times 2^i)$, $b = 2$, digit set = $\{0, 1\}$ 注: 如有小数, 可 $i < 0$
 - Used in computers. This is what computers can understand
- Decimal 如, $65 = 6 \times 10^1 + 5 \times 10^0 = 60 + 5 = 65$
 - $a = \sum_{i=0}^{n-1} (a_i \times 10^i)$, $b = 10$, digit set = $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$
 - Used by humans and high-level language programs
- Hexadecimal 如, $41 = 4 \times 16^1 + 1 \times 16^0 = 64 + 1 = 65$
 - $a = \sum_{i=0}^{n-1} (a_i \times 16^i)$, $b = 16$,
digit set = $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F\}$
 - Used by programmers and high-level language programs

Are there other positional number systems?

- Radically different
 - Can we use an irrational number as the base, e.g., π ?
- Yes
- Examples
 - Bergman's number system (the τ number system)
 - Can represent some irrational numbers exactly in finite digits
 - Created by George Bergman, a 12-year junior high school student
 - Base $b = \tau = (1 + \sqrt{5})/2 \approx 1.6180339$, digit set = $\{0, 1\}$
 - $14 = \mathbf{100100.110110} = \tau^5 + \tau^2 + \tau^{-1} + \tau^{-2} + \tau^{-4} + \tau^{-5}$
 - Fibonacci number system
 - Fibonacci numbers 1, 2, 3, 5, 8, ... as positional weights
 - digit set = $\{0, 1\}$
 - $14 = \mathbf{11001} = 1 \times 8 + 1 \times 5 + 0 \times 3 + 0 \times 2 + 1 \times 1$

Five positional number systems

Decimal	Hexadecimal	Binary	The τ Number System	FNS
$10^1 10^0$	16^0	$2^3 2^2 2^1 2^0$	$\tau^5 \tau^4 \tau^3 \tau^2 \tau^1 \tau^0 \tau^{-1} \tau^{-2} \tau^{-3} \tau^{-4} \tau^{-5} \tau^{-6}$	8 5 3 2 1
0	0	0000	0	00000
1	1	0001	1	00001
2	2	0010	10.01	00010
3	3	0011	100.01	00100
4	4	0100	101.01	00101
5	5	0101	1000.1001	01000
6	6	0110	1010.0001	01001
7	7	0111	10000.0001	01010
8	8	1000	10001.0001	10000
9	9	1001	10010.0101	10001
10	A	1010	10100.0101	10010
11	B	1011	10101.0101	10100
12	C	1100	100000.101001	10101
13	D	1101	100010.001001	11000
14	E	1110	100100.110110	11001
15	F	1111	100101.001001	11010

***3.4.2 IEEE 754 浮点表示法

为什么获图灵奖？

Use floating-point numbers to represent reals

- 有同学在个人作品实验使用

- How to represent $\pi \approx 3.1415927$?

- A simple way: converting whole and fraction into binary

- $3.1415927 = 11.0010010000111111011011$

- Another way (scientific notation): $3.1415927 =$

- May not be unique:

- $3.1415927 = 31415927 \times 10^{-7} = 0.31415927 \times 10^1 = \dots$

指数
Exponent



$$= 3.1415927 \times 10^0$$



Significant 尾数

- Key innovations of IEEE 754

- *Normalized significant* to guarantee representation uniqueness

- Default left-most 1: assume one and only one 1-digit before the binary point
- Since it is default, the left-most 1-bit can be omitted, saving one bit

- *Biased exponent* to speed up exponent comparison

- *Special values* for unusual

- Infinities ($\pm\infty$),
- Subnormal (underflow) values
- Not a Number (NaNs, such as trying to find $\sqrt{-5}$)

$$\begin{aligned} \pi &\approx 3.1415927 \times 10^0 \approx 1.5707964 \times 2^1 \\ &\approx +1.10010010000111111011011 \times 2^{00000001}; \\ &\rightarrow +.10010010000111111011011 \times 2^{00000001}; \quad \text{omit default left-most 1} \end{aligned}$$

Sign Exponent Significant

$$= \mathbf{0}1000000010010010000111111011011; \text{ the IEEE 754 representation}$$

Floating-point numbers are approximate values

- How to test the equality of two floating-point numbers?

> go run ./testPoint123.go

0.1+0.2 == 0.3

0.1+0.2 != 0.3

0.1+0.2 == 0.3

>

- To test if (X+Y) is equal to Z
 - Don't use $(X+Y) == Z$
 - Test if the absolute value of the difference is less than epsilon, i.e., use $\text{Abs}(X+Y-Z) < 10^{-12}$

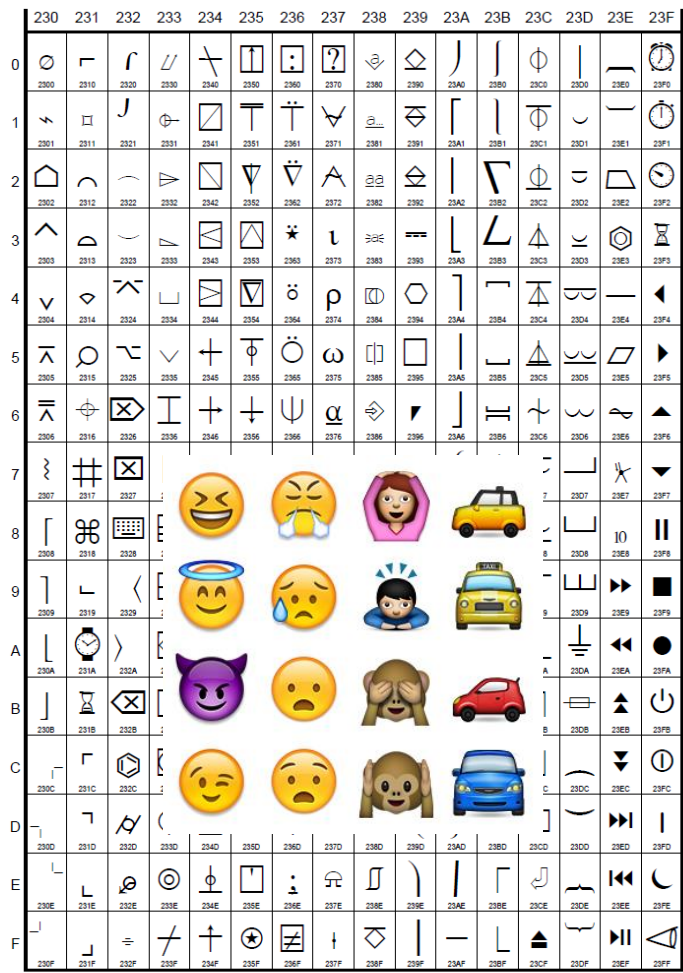
There may be a small difference between 0.1+0.2 and 0.3

```
package main // testPoint123.go
import "fmt"
import "math"
func main() {
    if 0.1 + 0.2 == 0.3 {
        fmt.Println("0.1+0.2 == 0.3")
    } else {
        fmt.Println("0.1+0.2 != 0.3")
    }
    X := 0.1 // var X float64 = 0.1
    Y := 0.2
    Z := 0.3
    if X + Y == Z {
        fmt.Println("0.1+0.2 == 0.3")
    } else {
        fmt.Println("0.1+0.2 != 0.3")
    }
    if math.Abs(X+Y - Z) < math.Pow(10, -12) {
        fmt.Println("0.1+0.2 == 0.3")
    } else {
        fmt.Println("0.1+0.2 != 0.3")
    }
}
```

3.4.3 ASCII, Unicode, UTF-8

表示字符

- ASCII encodes English characters, using 7 bits
- Unicode encodes the world's characters
 - Using 0000~FFFF₁₆, 10000~10FFFF₁₆
 - More than 1 million code points
 - Some combinations reserved
 - A character's Unicode encoding needs 16 or 32 bits
- UTF-8 (Unicode Transformation Format – 8-bit)
 - A variable-width character encoding implementing Unicode
 - Historical standard; most widely used in Internet
 - 注意：个人作品使用UTF-8



The Unicode Standard 13.0, Copyright © 1991-2020 Unicode, Inc. All rights reserved.

Symbol	Description	ASCII	Unicode	UTF-8	Bytes needed by UTF-8
T	English capital letter T	0X54	U+0054	0X54	1
Ω	Greek letter Omega	N/A	U+03A9	0XCEA9	2
€	The Euro sign	N/A	U+20AC	0XE282AC	3
志	A Chinese character	N/A	U+5FD7	0XE5BF97	3
⦿	A Gothic letter	N/A	U+10348	0XF0908D88	4

3.4.4 Review of bit, byte, character, integer, array, slice

- Use a program to understand the basics in one place
请写一个Go程序，打印出各种数据类型的值

```
X := byte(63)                // X is a byte variable. What if changed to X:=63?
fmt.Printf("Decimal: %d\n", X) // Decimal: 63
fmt.Printf("Hex: %X\n", X)    // Hex: 3F
fmt.Printf("Character: %c\n", X) // Character: ?
fmt.Printf("Binary: %b\n", X)  // Binary: 111111; X是8比特字节，事实上是00111111，忽略了leading 0's
```

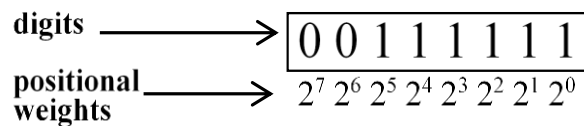
```
var S [5]byte = [5]byte{'h','e','l','l','o'}
// S=[104, 101, 108, 108, 111]
```

```
var byteSlice []byte = S[1:4]
// byteSlice=[101, 108, 108]
// slice is built from array
```

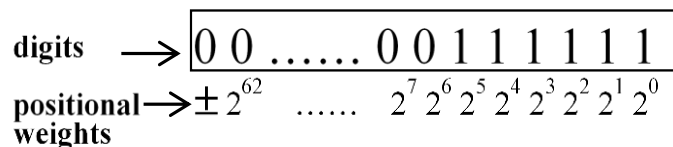
切片是动态数组

```
fmt.Println("array S = ", S)
// Array S = [104 101 108 108 111]
fmt.Println("byteSlice = ", byteSlice)
// byteSlice = [101 108 108]
```

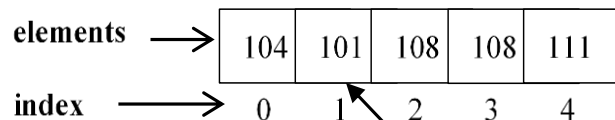
A byte variable X
by X:=byte(63)



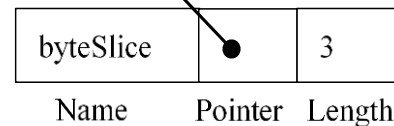
An int variable X
generated by X:=63



An array S generated
by var S [5]byte =
[5]byte{'h','e','l','l','o'}



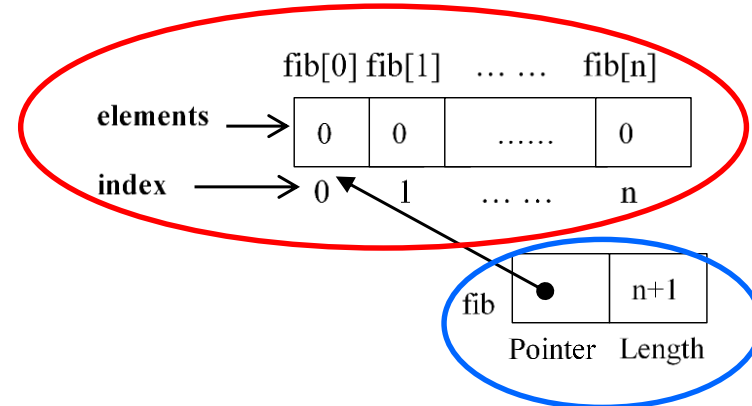
A slice byteSlice
generated from array S
var byteSlice []byte = S[1:4]



Remarks on Golang

- **byte** and **int** are the types most often used
 - Use **int** to represent an integer
 - Use **int** to represent a Unicode character, assuming leading 0's
 - '?' in previous slide
 - Use **byte** (which is the same as uint8) to represent an ASCII character and a small natural number
- A string is a read-only slice of bytes
 - Normally use **string** to represent a character string
- A **slice** can be generated in two ways
 - From an existing array, e.g.,
 - `var byteSlice []byte = S[1:4]`
 - Using the **make** function
 - `fib := make([]int, n+1)`
creates a **slice** and assigns to fib
 - `fib[i]` accesses the *i*th element of the array, which is initialized to 0. Also, `len(fib)=n+1`

Underlying array has $n+1$ elements



3.4.6 How to do bit operations?

- Answer: Operate on the byte or int variable containing the bit
 - Through some mask mechanism
- Example 1: Inverting the rightmost bit of a byte
 - Input: 00111111¹; Output: 00111110⁰
- Code explained with the corresponding operations

```
x := byte(63)      // assign 63=00111111 to variable x
v := ^x            // bitwise NOT of x, i.e., v=11000000
v = v & 0x1        // bitwise AND to retain the right-most bit of v
x = x & 0xFE       // bitwise AND to clear the right-most bit of x
x = x | v          // bitwise OR to get the final result
```

Mask
mechanism

```
x = 001111111
v =  $\overline{0}\overline{0}\overline{1}\overline{1}\overline{1}\overline{1}\overline{1}\overline{1}$  = 110000000
v = 11000000 & 00000001 = 000000000
x = 00111111 & 11111110 = 001111100
x = 00111110 | 00000000 = 001111100
```

Given input
Bitwise NOT
Bitwise AND
Bitwise AND
Bitwise OR

Replacing the least significant 2 bits of a byte

(used in Project Text Hider)

- Input: 00111111 = 63, 00101010 = 42; Output: 00111110
- 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
                    // 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	= 00000010	Bitwise AND
x = 00111111 & 11111100	= 00111100	Bitwise AND
x = 00111100 00000010	= 00111110	Bitwise OR

Note: 0x3 = 00000011; 0xFC = 00111100

The struct type

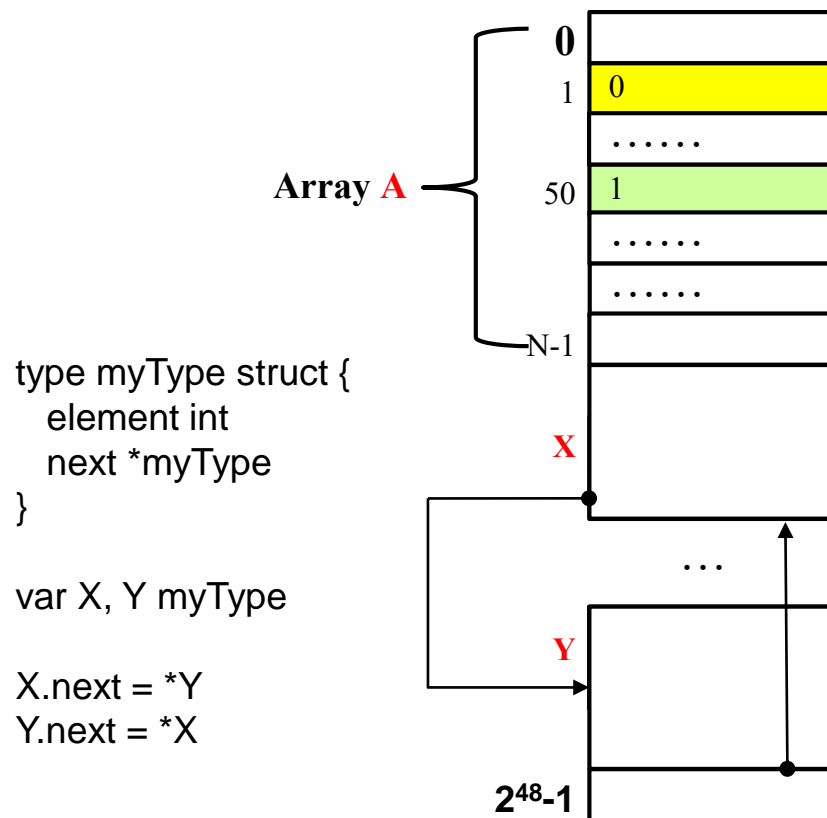
- Array is simple
 - Linear, consecutive arrangement of N elements of the same type
 - Example: `var A [5]byte` defines a byte array A of 5 elements
 - `A[0]`, `A[1]`, `A[2]`, `A[3]`, `A[4]` are all of type byte 数组：表示一组同构（同类）元素
- What if the elements are of different types? 不同类怎么办?
- Use **struct** 表示一组异构元素
 - A data structure with different types of elements
 - Elements are called fields
 - Use the **dot notation** to access fields, e.g.,
 - `JoanSmith.ID` accesses the student ID of Joan Smith, 不是 `JoanSmith[1]`
 - `FanWang.active = false` assigns the false value to the active field of student Fan Wang, indicating that he is not actively enrolled
 - 对比：数组使用index指向特定元素，如 `A[3]`

```
type Student struct {  
    name      string  
    ID        int  
    majorCode byte  
    active     bool  
    contact   string  
}  
var JoanSmith, FanWang Student
```

***3.4.7 Pointers and addressing modes

指针与寻址模式

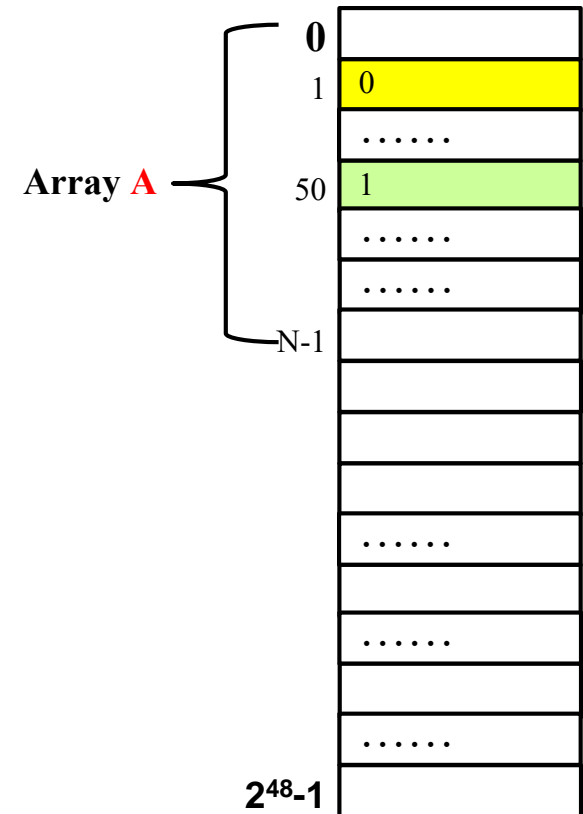
- Array is simple
 - Linear, consecutive arrangement of N elements of the same type
数组：一组顺序连续的同类数据
 - Example: a byte array A
 - Next element of A[i] is A[i+1]
 - If A[i] is at address 50, A[i+1] is at 51
 - What if not consecutive?
假如不连续怎么办?
- Pointer brings flexibility
 - Nonlinear arrangements, where the elements can jump around
 - Example: two variables X and Y connected by pointers
 - Indicated by the two arrows



The number 48 is due to the fact that Golang has a 48-bit virtual memory space

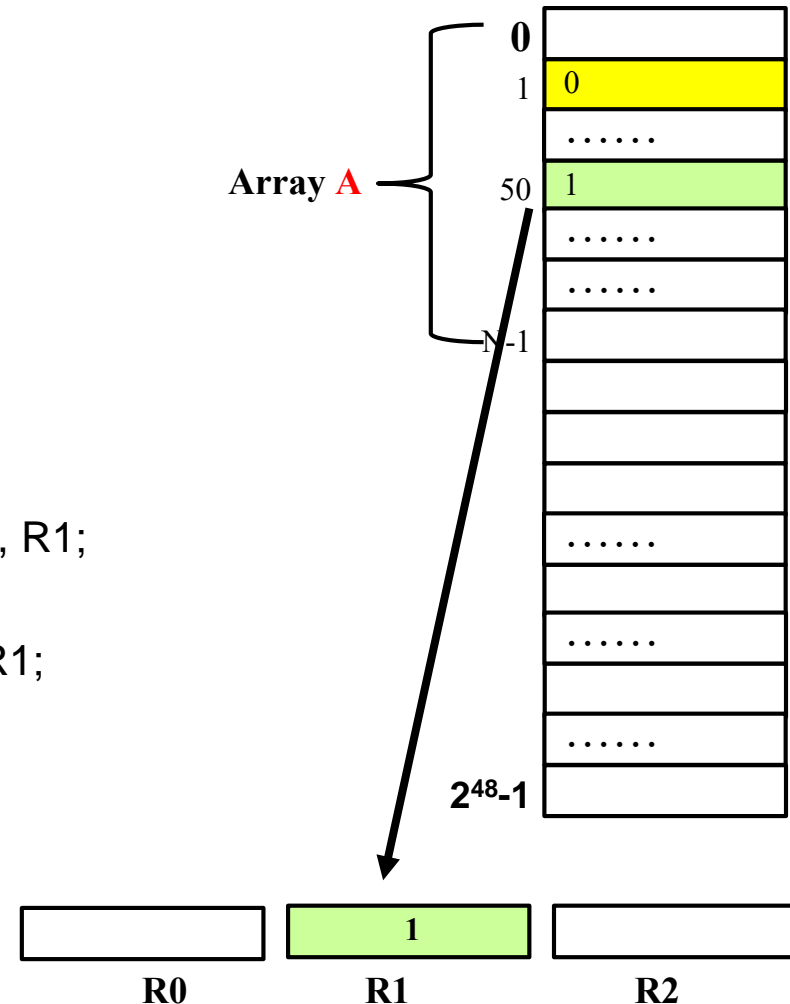
Contrasting three addressing modes

- Array is simple
 - Linear, consecutive arrangement
- Pointer brings flexibility
 - Nonlinear arrangements
- Pointers are supported by the *indirect addressing mode*
 - Contrasting three addressing modes
 - **Immediate addressing** 立即数寻址: MOV 50, R1;
 - 50 → R1, i.e., R1=50
 - No memory access



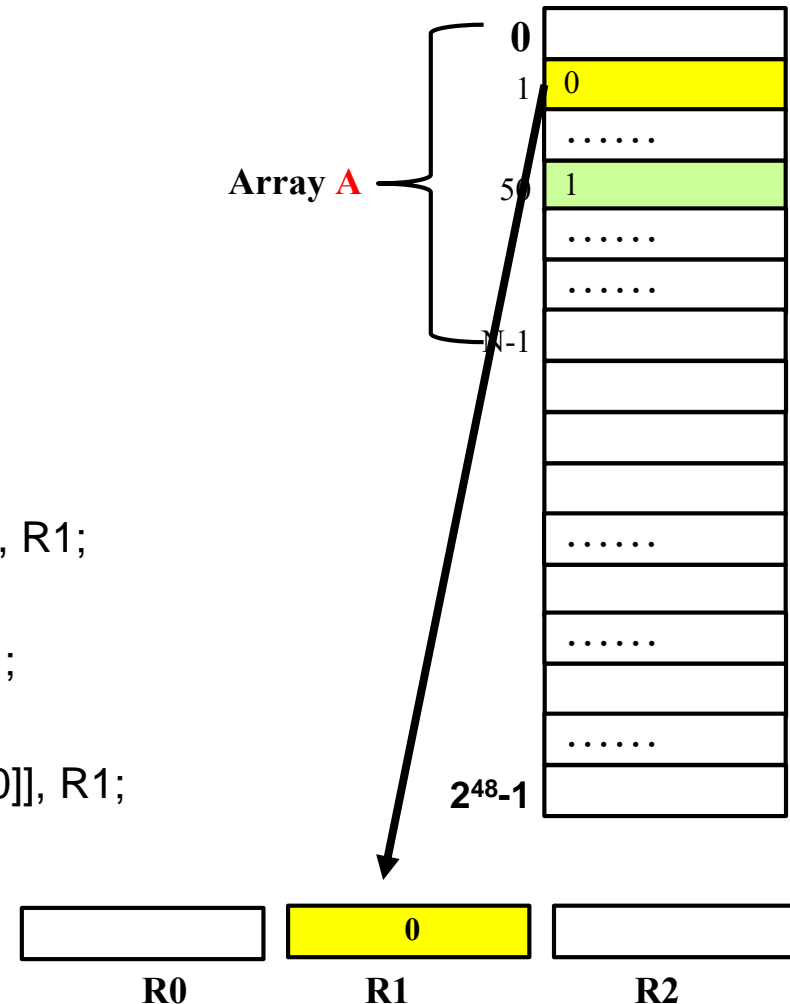
Contrasting three addressing modes

- Array is simple
 - Linear, consecutive arrangement
- Pointer brings flexibility
 - Nonlinear arrangements
- Pointers are supported by the *indirect addressing mode*
 - Contrasting three addressing modes
 - Immediate addressing 立即数寻址: `MOV 50, R1;`
 - `50` → R1, i.e., `R1=50`
 - **Direct addressing** 直接寻址: `MOV M[50], R1;`
 - `M[50]` → R1, i.e., `R1=1`
 - Often the common case



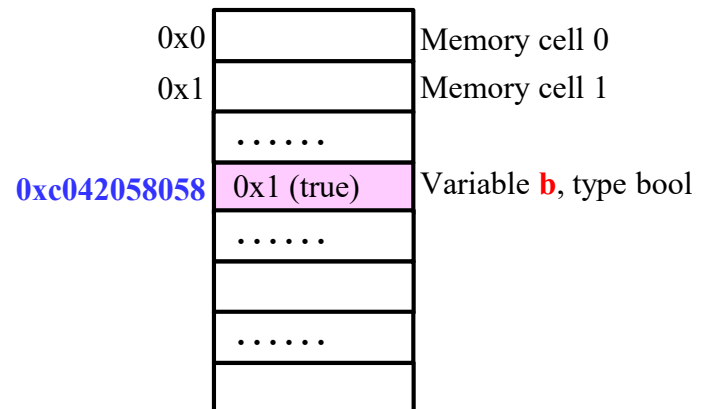
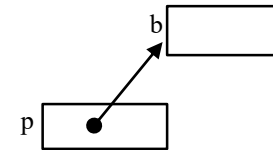
Contrasting three addressing modes

- Array is simple
 - Linear, consecutive arrangement
- Pointer brings flexibility
 - Nonlinear arrangements
- Pointers are supported by the *indirect addressing mode*
 - Contrasting three addressing modes
 - Immediate addressing 立即数寻址: `MOV 50, R1;`
 - `50` → R1, i.e., $R1=50$
 - Direct addressing 直接寻址: `MOV M[50], R1;`
 - `M[50]` → R1, i.e., $R1=1$
 - Indirect addressing 间接寻址: `MOV M[M[50]], R1;`
 - `M[M[50]]` → R1, i.e.,
 $M[M[50]]$ is $M[1]$, $R1=M[1]=0$
 - First get the address
 - Then access for the value



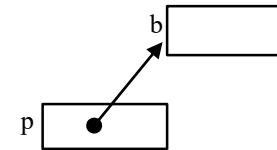
Step-by-step Illustration of pointers

```
func main() {  
    b := true           // Boolean variable b  
    p := &b              // p holds b's address  
    fmt.Println(p)       // Print b's address  
    fmt.Println(*p)      // Print b's value; dereference p  
    *p = false           // Modify b's value  
    fmt.Println(b)       // Print b's value  
    *p = !(*p)           // Use and modify b's value by negation  
    fmt.Println(b)  
}
```



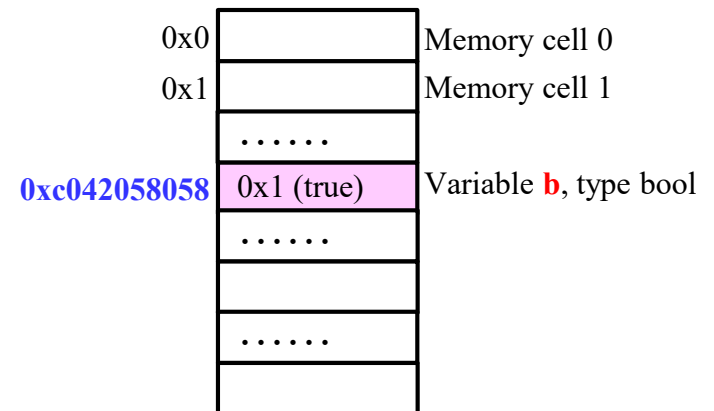
Step-by-step Illustration of pointers

```
func main() {  
    b := true           // Boolean variable b  
    p := &b             // p holds b's address  
    fmt.Println(p)       // Print b's address  
    fmt.Println(*p)      // Print b's value; dereference p  
    *p = false           // Modify b's value  
    fmt.Println(b)       // Print b's value  
    *p = !(*p)           // Use and modify b's value by negation  
    fmt.Println(b)  
}
```



1-minute Quiz:

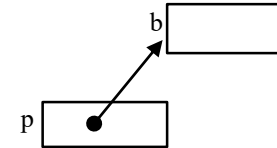
What is the output of the final statement?



Step-by-step Illustration of pointers

- 理解指针的一个好办法是运行实例程序

```
func main() {  
    b := true           // Boolean variable b  
    p := &b             // p holds b's address  
    fmt.Println(p)      // Print b's address  
    fmt.Println(*p)     // Print b's value  
    *p = false          // Modify b's value  
    fmt.Println(b)      // Print b's value  
    *p = !(*p)          // Use and modify b's value by negation  
    fmt.Println(b)  
}
```



```
> go run ./pointer.go  
0xc042058058  
true  
false  
true  
>
```

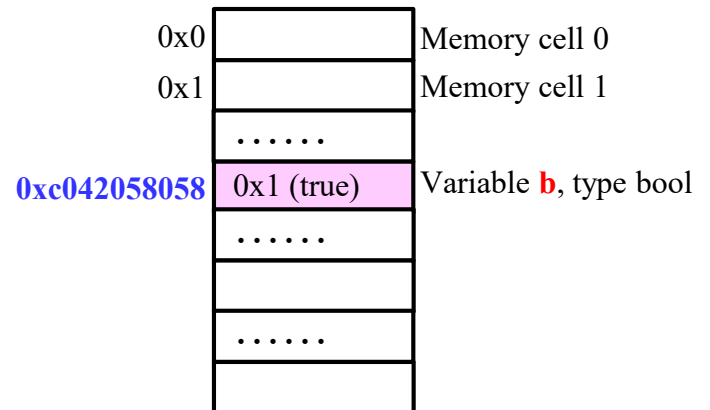
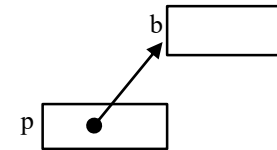


Illustration of pointers

```
func main() {  
    b := true           // Boolean variable b  
    p := &b             // p holds b's address  
    fmt.Println(p)      // Print b's address  
    fmt.Println(*p)     // Print b's value  
    *p = false          // Modify b's value  
    fmt.Println(b)      // Print b's value  
    *p = !(*p)          // Use and modify b's value by negation  
    fmt.Println(b)  
}
```



```
> go run ./pointer.go  
0xc042058058  
true  
false  
true  
>
```

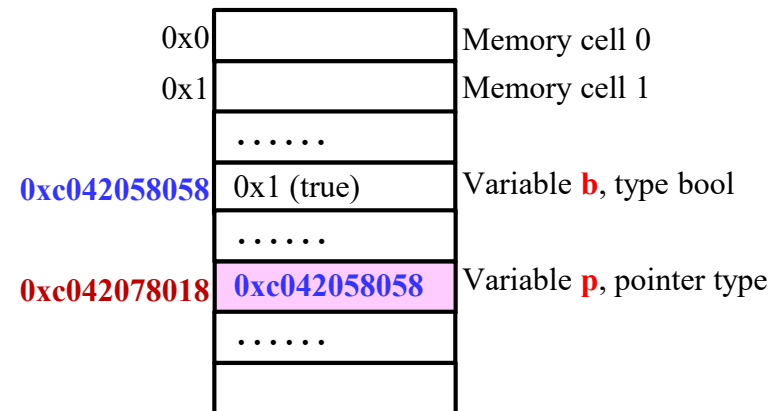


Illustration of pointers

```
func main() {  
    b := true           // Boolean variable b  
    p := &b             // p holds b's address  
    fmt.Println(p)      // Print b's address  
    fmt.Println(*p)     // Print b's value  
    *p = false          // Modify b's value  
    fmt.Println(b)      // Print b's value  
    *p = !(*p)          // Use and modify b's value by negation  
    fmt.Println(b)  
}
```

> go run ./pointer.go

0xc042058058

true

false

true

>

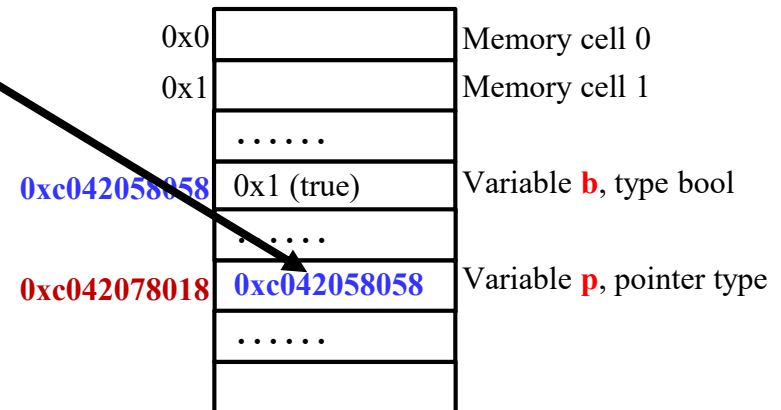
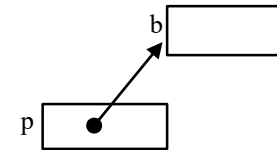
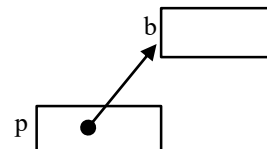


Illustration of pointers

```
func main() {  
    b := true           // Boolean variable b  
    p := &b             // p holds b's address  
    fmt.Println(p)      // Print b's address  
    fmt.Println(*p)     // Print what *p holds, i.e., b's value 间接寻址  
    *p = false          // Modify b's value  
    fmt.Println(b)      // Print b's value  
    *p = !(*p)          // Use and modify b's value by negation  
    fmt.Println(b)  
}
```



```
> go run ./pointer.go  
0xc042058058  
true  
false  
true  
>
```

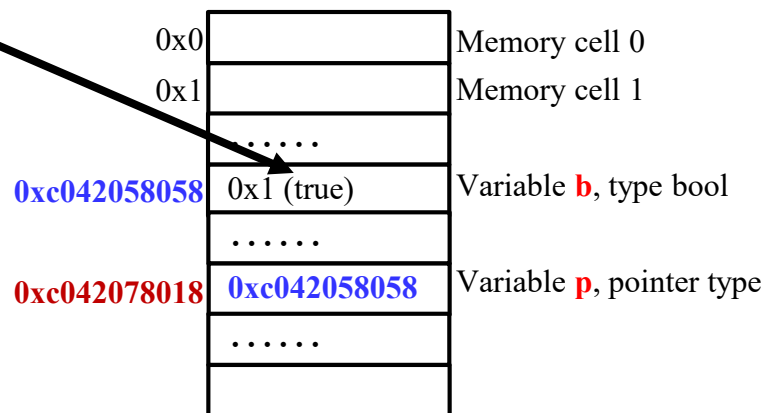
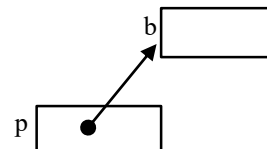


Illustration of pointers

```
func main() {  
    b := true           // Boolean variable b  
    p := &b             // p holds b's address  
    fmt.Println(p)      // Print b's address  
    fmt.Println(*p)     // Print b's value  
    *p = false          // Modify b's value 间接寻址  
    fmt.Println(b)      // Print b's value  
    *p = !(*p)          // Use and modify b's value by negation  
    fmt.Println(b)  
}
```



```
> go run ./pointer.go  
0xc042058058  
true  
false  
true  
>
```

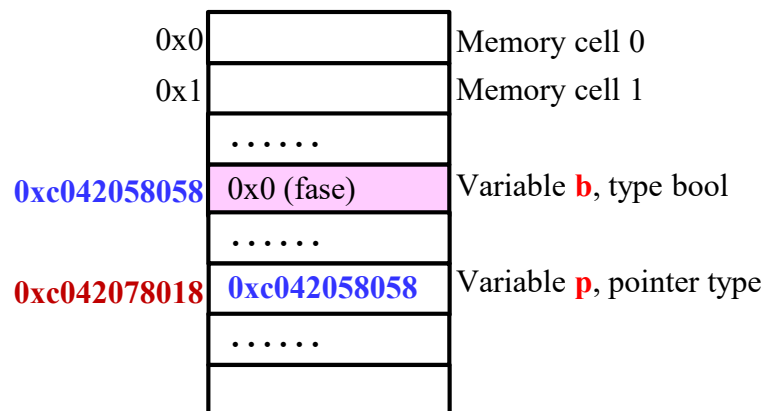
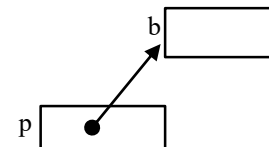


Illustration of pointers

```
func main() {  
    b := true           // Boolean variable b  
    p := &b            // p holds b's address  
    fmt.Println(p)      // Print b's address  
    fmt.Println(*p)     // Print b's value  
    *p = false          // Modify b's value  
    fmt.Println(b)     // Print b's value  
    *p = !(*p)          // Use and modify b's value by negation  
    fmt.Println(b)  
}
```



```
> go run ./pointer.go  
0xc042058058  
true  
false  
true  
>
```

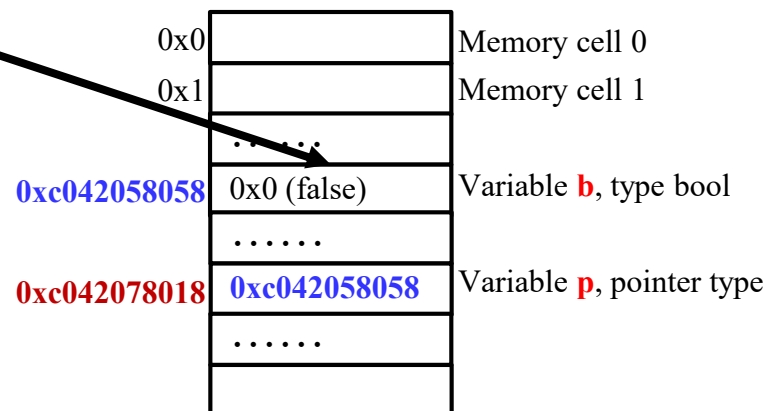
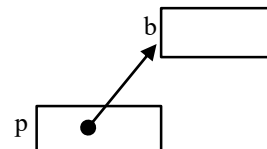


Illustration of pointers

```
func main() {  
    b := true           // Boolean variable b  
    p := &b             // p holds b's address  
    fmt.Println(p)      // Print b's address  
    fmt.Println(*p)     // Print b's value  
    *p = false          // Modify b's value  
    fmt.Println(b)      // Print b's value  
    *p = !(*p)          // Use and modify b's value by negation  
    fmt.Println(b)  
}
```



```
> go run ./pointer.go  
0xc042058058  
true  
false  
true  
>
```

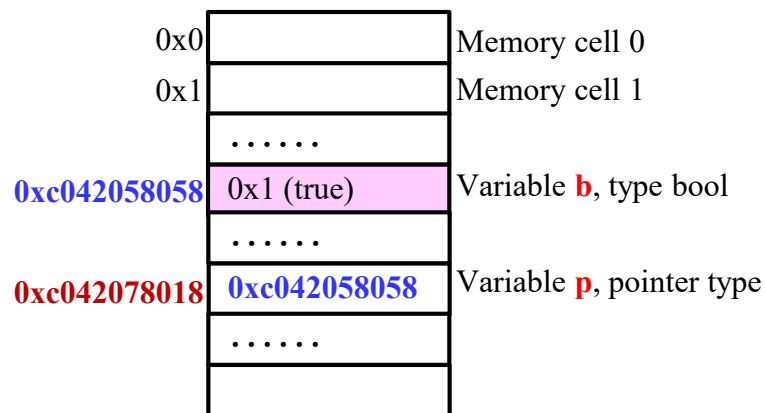
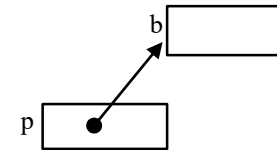
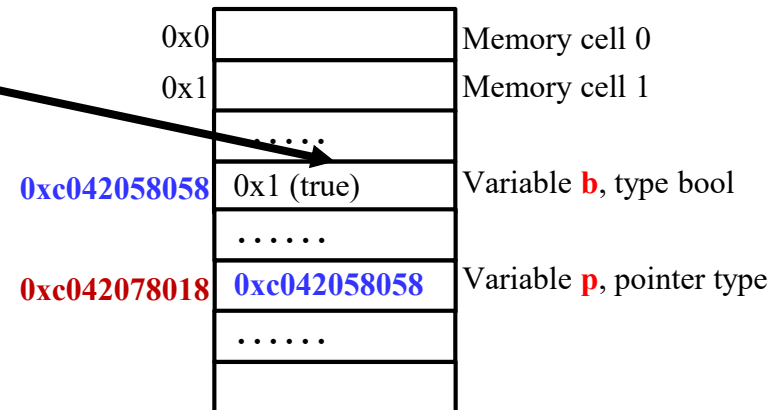


Illustration of pointers

```
func main() {  
    b := true           // Boolean variable b  
    p := &b             // p holds b's address  
    fmt.Println(p)      // Print b's address  
    fmt.Println(*p)     // Print b's value  
    *p = false          // Modify b's value  
    fmt.Println(b)      // Print b's value  
    *p = !(*p)          // Use and modify b's value by negation  
    fmt.Println(b)  
}
```



```
> go run ./pointer.go  
0xc042058058  
true  
false  
true  
>
```



3.4.8 The file abstraction 文件与文件系统

- To organize and **persistently** store chunks of information
- Files are organized as a hierarchy (tree)
 - Leaf nodes are files; internal nodes are **directories** (special files)
- A file is identified by a **file name** (file path, or **path**)

- **Absolute path** 绝对路径: all the way from the root (/)

- Absolute path of **Autumn.bmp**: /cs101/Prj2/Autumn.bmp

- **Relative path** 相对路径 of **Autumn.bmp**

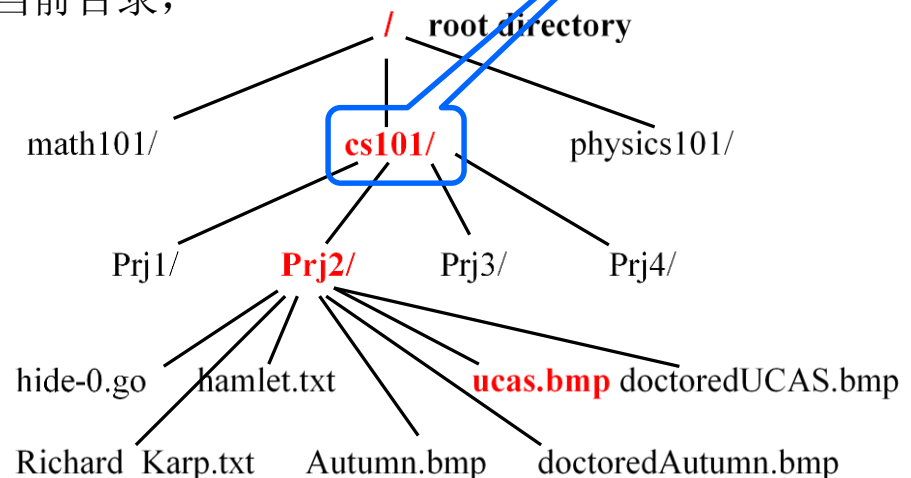
- Related to the **current directory** 当前目录, i.e., **working directory** 工作目录

- **./Autumn.bmp** if the working directory is /cs101/Prj2/

- **../Prj2/Autumn.bmp** if the working directory is /cs101/Prj2/

- **Home directory** 缺省目录

- The default directory when log in. Assume /cs101 is the home directory



```
>pwd
/cs101/
```

You're here

3.4.8 The file abstraction

- To organize and **persistently** store chunks of information
- Files are organized as a hierarchy (tree)
 - Leaf nodes are files; internal nodes are **directories** (special files)
- A file is identified by a **file name** (file path, or **path**)

- **Absolute path**: all the way from the root (/)

- Absolute path of Autumn.bmp: /cs101/Prj2/Autumn.bmp

- **Relative path** of Autumn.bmp

- Related to the **current directory**, i.e., **working directory**

- **./Autumn.bmp** if the working directory is /cs101/Prj2/

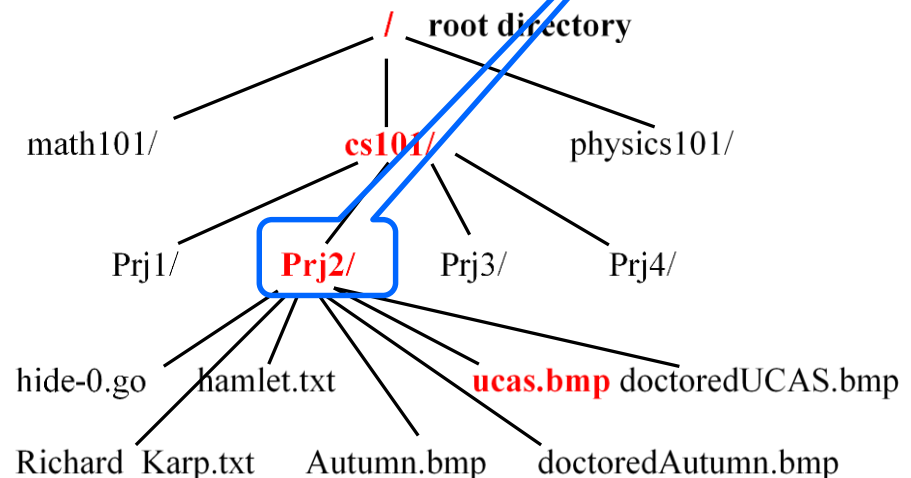
- **../Prj2/Autumn.bmp** if the working directory is /cs101/Prj2/

- **Home directory**

- The default directory when login in. Assume /cs101 is the home directory

```
>cd Prj2
/cs101/Prj2
>pwd
/cs101/Prj2
```

You're here



Hide text in a picture

- Write a program
 - to hide the text of Shakespeare's *Hamlet*
 - in a picture file Autumn.bmp
 - such that the doctored file shows no visible difference from the original picture
- and another program to recover the text

HAMLET

DRAMATIS PERSONAE

CLAUDIUS king of Denmark. (KING CLAUDIUS:)

HAMLET son to the late, and nephew to the present king.

.....

ACT ISCENE I Elsinore. A platform before the castle.

.....

PRINCE FORTINBRAS Let four captains
Bear Hamlet, like a soldier, to the stage;
For he was likely, had he been put on,
To have proved most royally: and, for his passage,
The soldiers' music and the rites of war
Speak loudly for him.
Take up the bodies: such a sight as this
Becomes the field, but here shows much amiss.
Go, bid the soldiers shoot.
[A dead march. Exeunt, bearing off the dead
bodies; after which a peal of ordnance is shot off]



Original picture



After careless hiding



After careful hiding

Look inside a file: data and metadata

- A file is a group of bits **and** may be structured
 - 文件是一组比特，可能是有结构的
 - 无结构例子: $F(500) = 1394\ 2322\ 4561\ 6978\ 8013\ 9724\ 3828\ 7040\ 7283\ 9500\ 7025\ 6587\ 6973\ 0726\ 4108\ 9629\ 4832\ 5571\ 6228\ 6329\ 0691\ 5576\ 5887\ 6222\ 5212\ 94125$
- 有结构例子: Data and metadata of file Autumn.bmp
 - Data: bits of the actual picture (Pixel Array)
 - Metadata 元数据 : data about the picture data
 - BMP format in the file: File Header, Info Header
 - Other data associated with the file

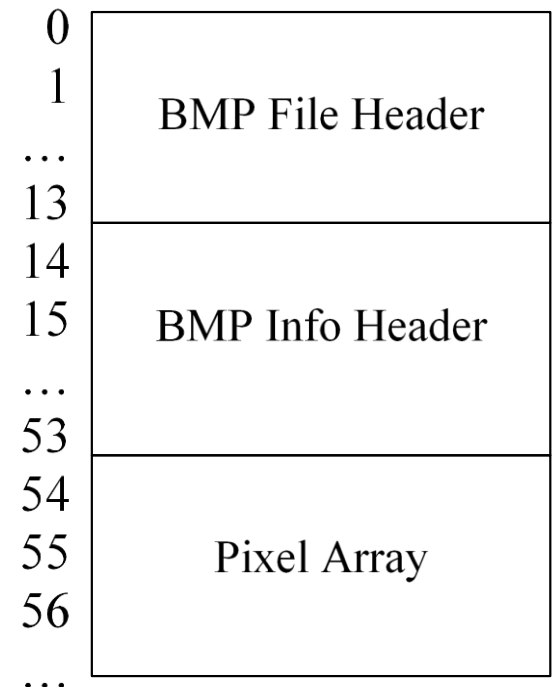


Autumn.bmp

The extension **bmp** says it's a bit map image file

Addresses 0~53 hold metadata

The pixel array for actual image data starts at address 54



Other types of metadata

- Can be seen by running “ls -l Autumn.bmp”
 - ls -l Autumn.bmp中，l是小写的L，不是大写的i
- The file name, the file size, the time of creation (last modification)
- Access permissions 三类三种权限
 - 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

- Example of access permissions

- ioutil.WriteFile("./doctoredAutumn.bmp", p, 0666)
- Every user can read and write, but cannot execute
 - -rw-rw-rw-
 - 0666 = 0110110110

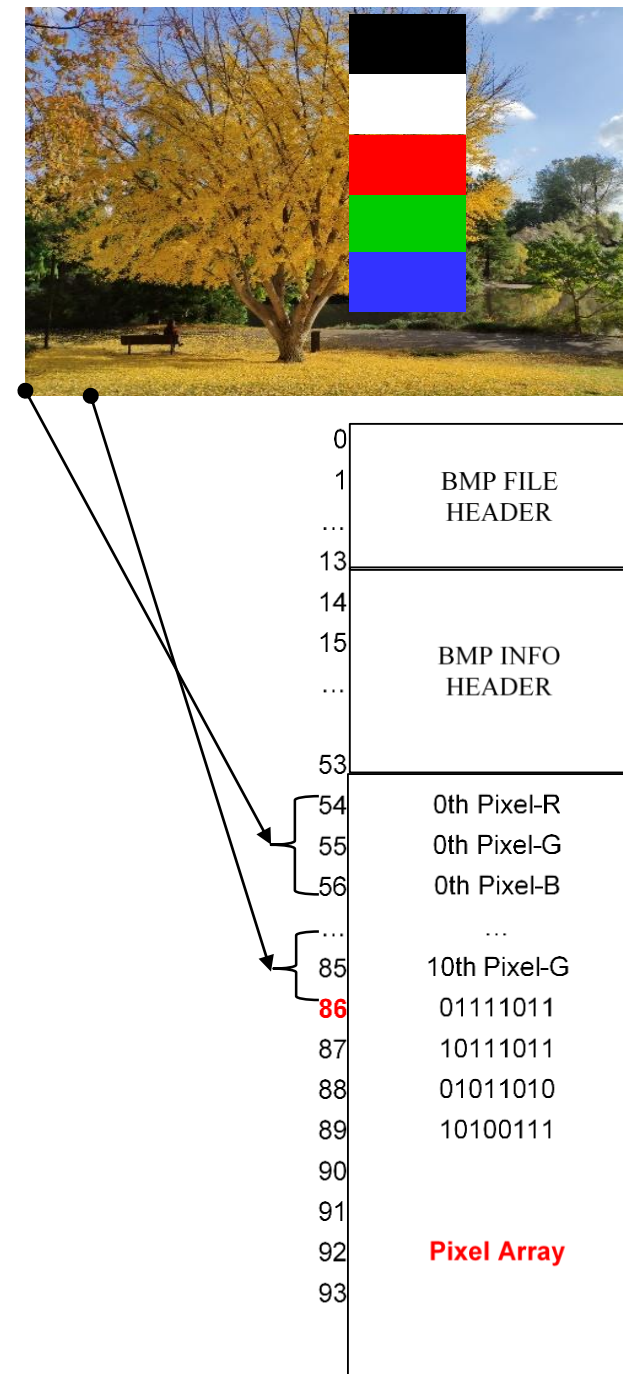
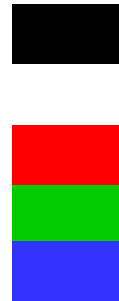
Owner			Group			Others		
r	w	e	r	w	e	r	w	e
1	1	-	1	1	-	1	1	-

0666: 拥有者（用户本人）、组、他人可读、可写，不可执行
这是一个数据文件，不是程序文件

0700表示什么权限？

How is the image of Autumn.bmp stored in Pixel Array?

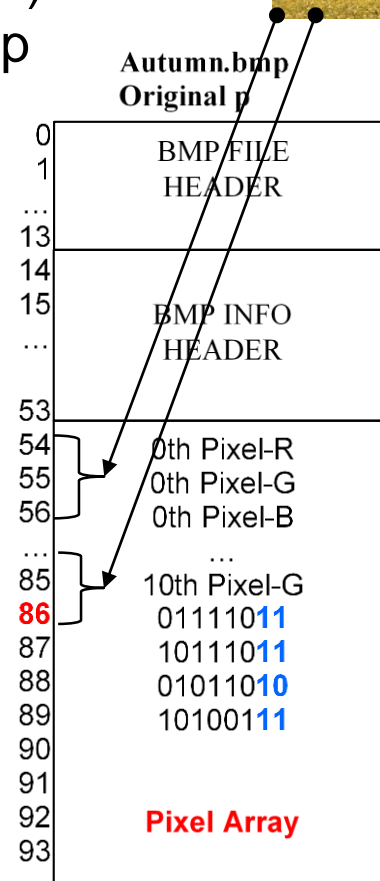
- Pixel = Picture Element 像素
- Pixel Array holds the pixels of the image, starting from the low-left corner going right, and then up row by row
 - 第一个像素是在图像左下角
- Each pixel has three bytes for
 - Color depth values of RGB, i.e., the primary colors of red, green, blue
 - 每个像素有红绿蓝三种原色，需要三个字节
 - 原色深度值=0~255，越大越深
 - RGB values = (0, 0, 0) → Black
 - RGB values = (255, 255, 255) → White
 - RGB values = (255, 0, 0) → Red
 - RGB values = (0, 255, 0) → ?
 - RGB values = (0, 0, 255) → ?
- The first pixel is the 0th element of Pixel Array
 - Uses addresses 54, 55, and 56 to store its three RGB color depth values 红绿蓝原色深度值



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

We need the length when recovering in show.go

- `p, _ := ioutil.ReadFile("./Autumn.bmp")` to read the image file into byte array `p`
- Recall that a function can return multiple values
 - This function returns two values, the second of which is not needed by this code
 - Use a placeholder symbol `'_'`
 - Also called **the blank identifier**



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

- `p, _ := ioutil.ReadFile("./Autumn.bmp")`
to read the image file into byte array `p`
- `t, _ := ioutil.ReadFile("./hamlet.txt")`
to read the text file into byte array `t`
- Length `len(t)` is a 64-bit integer
 - Hide every 2 bits in a byte of `p`
 - Need 32 bytes
 - `S = 54, T = 32`
- `modify(len(t), p[S:S+T], T)`
to hide `len(t)` in `p[54:86]`



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-R	54	Hide 2 bits of len(t)
55	0th Pixel-G	55	Hide 2 bits of len(t)
56	0th Pixel-B	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	

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

- `p, _ := ioutil.ReadFile("./Autumn.bmp")`
to read the image file into byte array `p`
- `t, _ := ioutil.ReadFile("./hamlet.txt")`
to read the text file into byte array `t`
- Length `len(t)` is a 64-bit integer
 - Hide every 2 bits in a byte of `p`
 - Need 32 bytes
 - `S = 54, T = 32`
- `modify(len(t), p[S:S+T], T)`
to hide `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 3.4.6 in this lecture



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-R	54	Hide 2 bits of len(t)
55	0th Pixel-G	55	Hide 2 bits of len(t)
56	0th Pixel-B	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	

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

- `p, _ := ioutil.ReadFile("./Autumn.bmp")`
to read the image file into byte array `p`
- `t, _ := ioutil.ReadFile("./hamlet.txt")`
to read the text file into byte array `t`
- `t[0]` holds the **1st character** 'H' = 72
- `modify(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	01111011
87	10111011
88	01011010
89	10100111



Modified `p[86:90]`

	01111000
	10111010
	01011000
	10100101

Autumn.bmp
Original p

0	BMP FILE HEADER
1	
...	
13	
14	
15	
...	
53	
54	
55	0th Pixel-R
56	0th Pixel-G
57	0th Pixel-B
...	...
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	
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	

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

- `p, _ := ioutil.ReadFile("./Autumn.bmp")`
to read the image file into byte array `p`
- `t, _ := ioutil.ReadFile("./hamlet.txt")`
to read the text file into byte array `t`
- To hide all `t[i]`, $i = 0$ to `len(t)`

```
for i:=0; i<len(t); i++{
    offset := S+T+(i*4)
    modify(int(t[i]), p[offset:offset+C], C)
}
```

- Each iteration hides `t[i]` in `p[S+T+(i*4):S+T+(i*4)+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-R	54	Hide 2 bits of len(t)
55	0th Pixel-G	55	Hide 2 bits of len(t)
56	0th Pixel-B	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	

Check results

- Write the complete `hide-0.go`
 - Execute and display result
 - > `go run hide-0.go`
 - > `display doctoredAutumn.bmp`
 - The Text Hider project
 - Produce `hide.go` with good coding practices
 - Also need to write `show.go` to recover
 - Change `hide-0.go` to `hide-1.go`
 - by modifying the most significant 2 bits (the rightmost 2 bits) of each byte of Pixel Array
- > `go run hide-1.go`
- > `display doctoredAutumn.bmp`



Original Autumn.bmp



doctoredAutumn.bmp
Modifying rightmost 2 bits



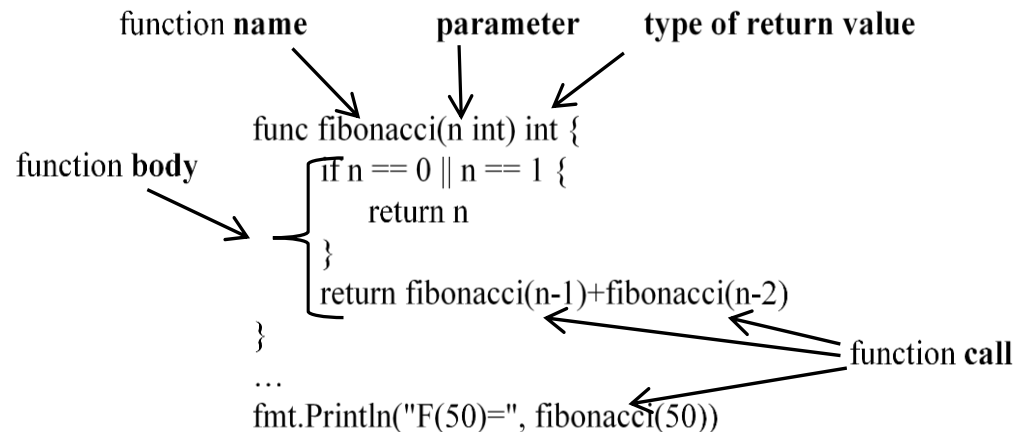
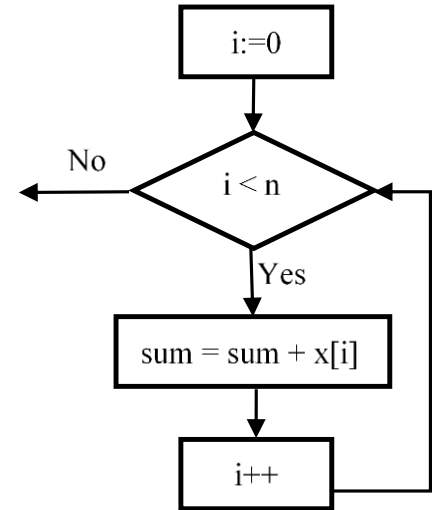
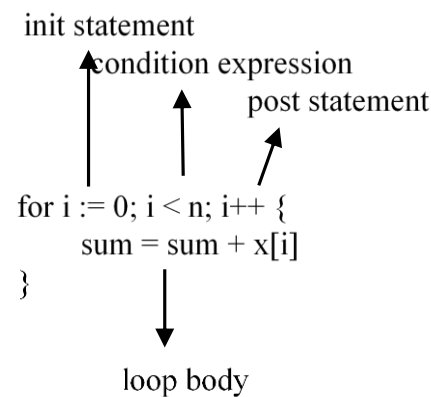
doctoredAutumn.bmp
Modifying leftmost 2 bits

3.5 Review of control abstractions

- **Precedence** in an expression
 - For $x*b+c \parallel i < 7$, the precedence is $((x*b)+c) \parallel (i < 7)$
 - When in doubt, use parentheses
- **Sequence** of statements: follow the syntactic order
- **Selection** (conditional): if-then-else statement

```
if i<7 {  
    fmt.Println(i)  
}
```

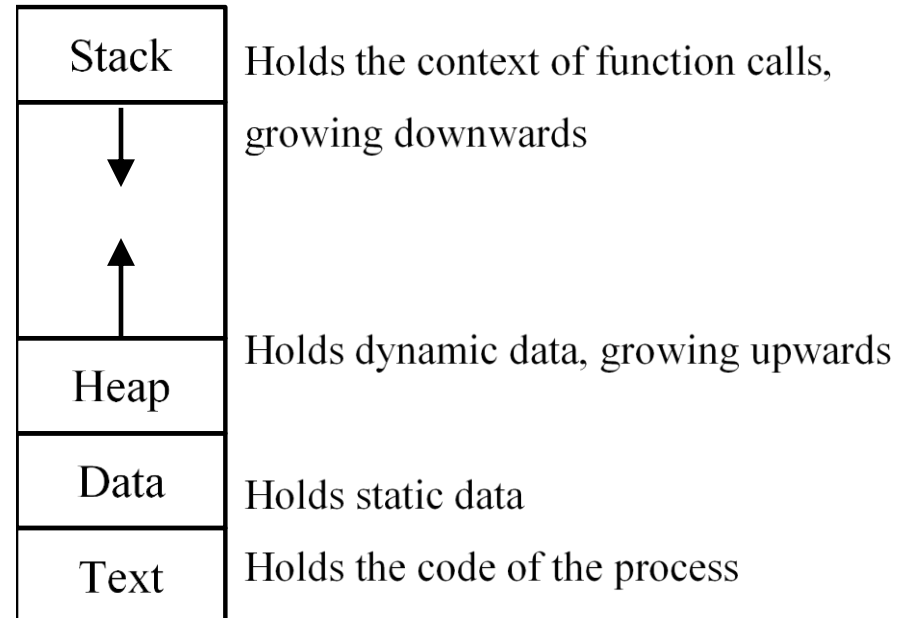
- **Loop**: repetitively execute a body of code
- **Function**: defined once and can be called many times



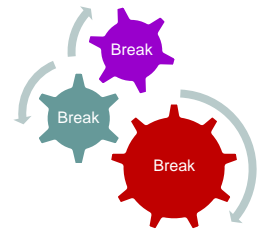
Segments of Text, Data, Heap and Stack

- Segments: areas of memory
 - Text: code of fib-50.go
 - Data: static data
 - Heap:
 - Stack: context of function calls
- How much memory is needed for computing $F(5000000)$?
 - A small constant? Multiple of 5 million? $2^{5000000}$?

```
package main
import "fmt"
func main() {
    fmt.Println("F(50)=", fibonacci(50))
}
func fibonacci(n int) int {
    if n == 0 || n == 1 {
        return n
    }
    return fibonacci(n-1)+fibonacci(n-2)
}
```



Take-Home Messages



- Positional number systems are widely used
 - Computers mostly use binary, decimal and hexadecimal systems
 - Can you extend Table 5.2 with an octal number system (base = 8)?
- IEEE floating number standard is used to represent real numbers
 - Why should we never use the double equal == to test the equality of two floating-point numbers?
- ASCII, Unicode, UTF-8 are used to encode the world's characters
 - Why do we need Unicode and UTF-8, when we already have ASCII?
- Contrast data types of byte, integer, array, slice, struct
 - When to use which?
 - How to represent and operate on bit and character? How to replace 3 bits of a byte?
- Understand the basics of pointers
- Review the concepts of
 - Precedence, sequence, selection, loop, function
 - Memory segments of text, data, heap, stack
- Write `hide-0.go` to understand the file abstraction
 - What is the difference between data and metadata? Give examples

*** Computing Fibonacci numbers of arbitrary word length, with fib.Uint.go

```
package main // fib.Uint.go
import (
    "fmt"
    "math"
)
func main() {
    fmt.Printf("F(100) = %s\n", String(*(fibonacci(100))))
}
type Uint []uint64
func fibonacci(n int) *Uint {
    a := &Uint{0} // a = 0
    b := &Uint{1} // b = 1
    for i := 1; i < n+1; i++ {
        Acc(a, b) // a = a + b
        a, b = b, a
    }
    return a
}
```

// Code for func Acc() and func String() functions

```
> go run fib.Uint.go
F(100) = 354224848179261915075
>
```

Program fib.Uint.go uses a slice of uint64 numbers to represent an unsigned integer of arbitrary length, by defining a type Uint

Function A(a, b) does an accumulation $a = a + b$ where a and b are of type *Uint

```
func Acc(a, b *Uint) {
    x, y := *a, *b
    d := make(Uint, len(y)-len(x))
    x = append(x, d...)
    c := make(Uint, len(x)+1)
    for i := 0; i < len(x); i++ {
        var v uint64
        v = x[i] + y[i] + c[i]
        if v < x[i] || v < y[i] || v < c[i] {
            c[i+1] = 1
        }
        c[i] = v
    }
    if c[len(c)-1] == 0 {
        c = c[:len(c)-1]
    }
    *a = c
}
```

Students do not need to
Know the internal of Acc

*** Code of fib.Uint.go, continued

```
package main // fib.Uint.go
import (
    "fmt"
    "math"
)
func main() {
    fmt.Printf("F(100) = %s\n", String(*(fibonacci(100))))
}
type Uint []uint64
func fibonacci(n int) *Uint {
    a := &Uint{0}           // a = 0
    b := &Uint{1}           // b = 1
    for i := 1; i < n+1; i++ {
        Acc(a, b)           //a = a + b
        a, b = b, a
    }
    return a
}
// Code for func Acc() and func String() functions
```

```
> go run fib.Uint.go
F(100) = 354224848179261915075
>
```

String(x) converts a Uint value into a decimal string

```
func String(x Uint) string {
    if len(x) == 1 && (x)[0] == 0 {
        return "0"
    }
    i := int(float64(len(x)*64)/math.Log2(float64(10))) + 1
    s := make([]byte, i)
    var r byte = 0
    for ; len(x) != 0; i-- {
        temp := make(Uint, len(x))
        var dividend uint64 = 0
        for i := len(x) - 1; i >= 0; i-- {
            dividend = dividend<<32 + uint64(x[i]>>32)
            q := dividend / 10
            r = byte(dividend - (q<<3 + q<<1))
            temp[i] = q
            dividend = uint64(r)
            dividend = dividend<<32 + uint64(x[i]<<32>>32)
            q = dividend / 10
            r = byte(dividend - (q<<3 + q<<1))
            temp[i] = temp[i]<<32 + q
            dividend = uint64(r)
        }
        if temp[len(temp)-1] == 0 {
            temp = temp[:len(temp)-1]
        }
        x, s[i-1] = temp, r+'0'
    }
    i = 0
    for s[i] == 0 { i++ }
    return string(s[i:])
}
```

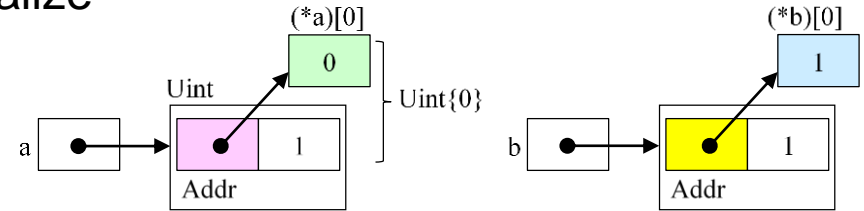
Students do not need to
Know the internal of String

*** Some execution details

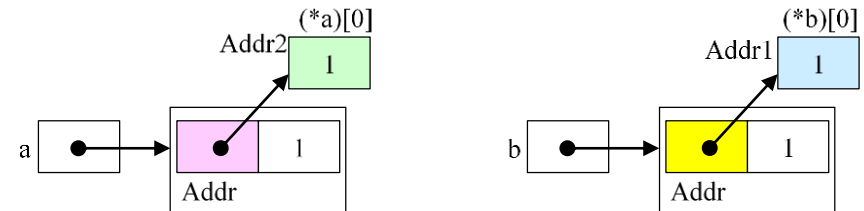
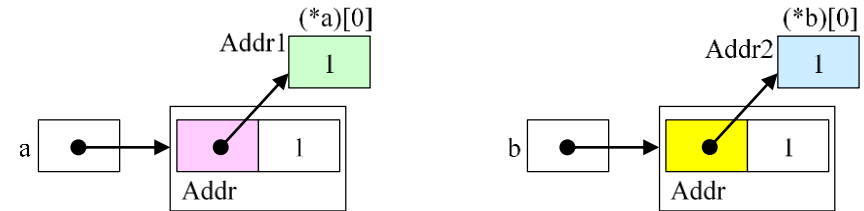
```
package main // fib.Uint.go
import (
    "fmt"
    "math"
)
func main() {
    fmt.Printf("F(100) = %s\n", String(*(fibonacci(100))))
}
type Uint [uint64]
func fibonacci(n int) *Uint {
    a := &Uint{0} // a = 0
    b := &Uint{1} // b = 1
    for i := 1; i < n+1; i++ {
        Acc(a, b) // a = a + b
        a, b = b, a
    }
    return a
}
// Code for func Acc() and func String() functions
```

```
> go run fib.Uint.go
F(100) = 354224848179261915075
>
```

Initialize



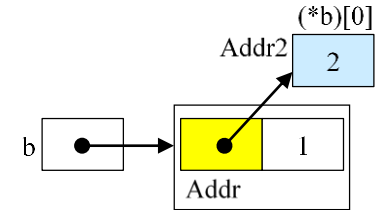
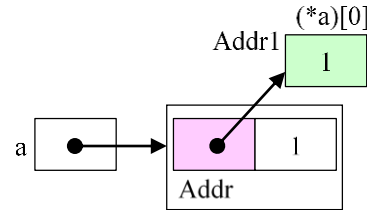
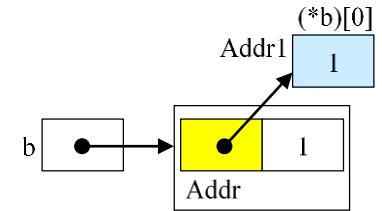
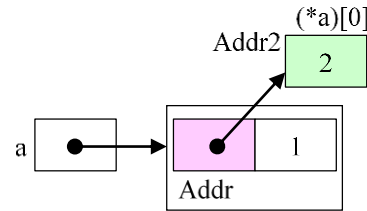
When i = 1



*** Some execution details

```
package main // fib.Uint.go
import (
    "fmt"
    "math"
)
func main() {
    fmt.Printf("F(100) = %s\n", String(*(fibonacci(100))))
}
type Uint []uint64
func fibonacci(n int) *Uint {
    a := &Uint{0} // a = 0
    b := &Uint{1} // b = 1
    for i := 1; i < n+1; i++ {
        Acc(a, b) // a = a + b
        a, b = b, a
    }
    return a
}
// Code for func Acc() and func String() functions
```

When i = 2

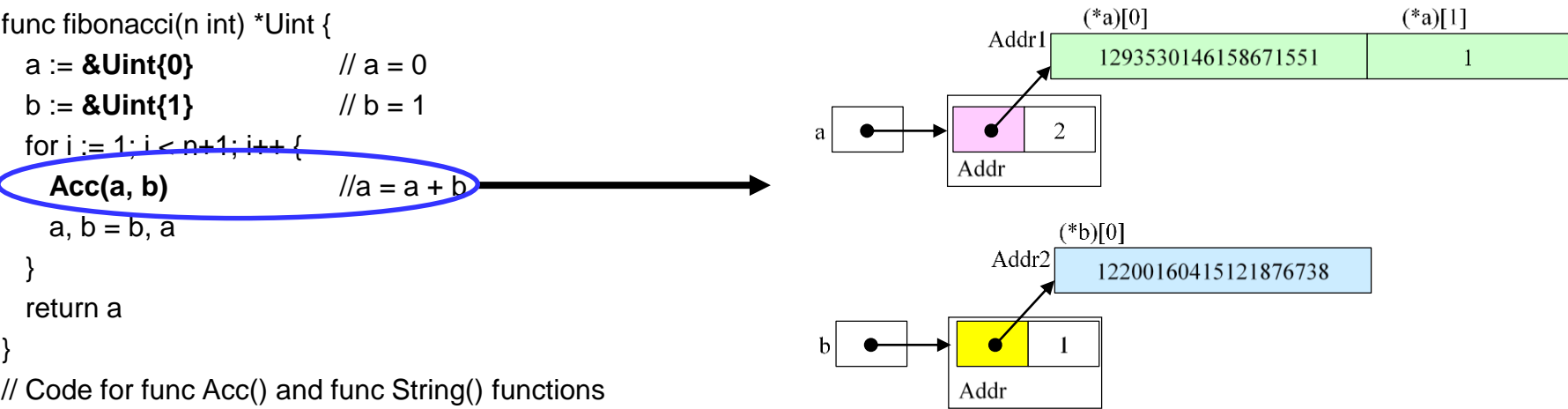


```
> go run fib.Uint.go
F(100) = 354224848179261915075
>
```

*** Some execution details

```
package main // fib.Uint.go
import (
    "fmt"
    "math"
)
func main() {
    fmt.Printf("F(100) = %s\n", String(*(fibonacci(100))))
}
type Uint []uint64
func fibonacci(n int) *Uint {
    a := &Uint{0} // a = 0
    b := &Uint{1} // b = 1
    for i := 1; i < n+1; i++ {
        Acc(a, b) //a = a + b
        a, b = b, a
    }
    return a
}
// Code for func Acc() and func String() functions
```

When i = 93



```
> go run fib.Uint.go
F(100) = 354224848179261915075
>
```