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

CS101

Systems Thinking

Data Abstractions, Control Abstractions

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
    - Representing numbers, characters
    - Review of bit, byte, character, integer, array, and slice, as well as struct
    - Pointers, files
  - Control abstractions
    - 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

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

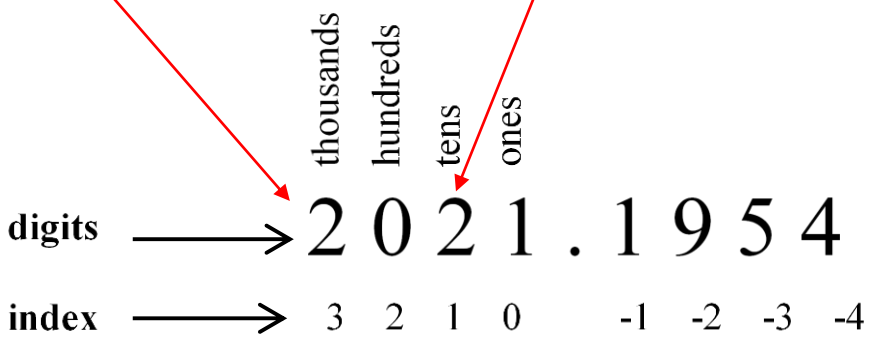
3.4.1 Positional number systems

- Computers (and humans) use positional number systems
- Why?
  - What happens if we use Roman numerals?
    - 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, i is index, {0,1,2,3,4,5,6,7,8,9} is digit set
- The key point: The value of a digit depends on both the digit and the position (index) of the digit
  - 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,…,b-1} is the digit set
- There are three positional number systems often used
- Binary
  - $a = \sum_{i=0}^{n-1}(a_i \times 2^i)$, $b = 2$, digit set = {0, 1}
  - Used in computers. This is what computers can understand
- Decimal
  - $a = \sum_{i=0}^{n-1}(a_i \times 10^i)$, $b = 10$, digit set = {0,1,2,3,4,5,6,7,8,9}
  - Used by humans and high-level language programs
- Hexadecimal
  - $a = \sum_{i=0}^{n-1}(a_i \times 16^i)$, $b = 16$,
    digit set = {0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F}
  - Used by humans 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

Use floating-point numbers to represent reals

- How to represent $\pi \approx 3.1415927$?
  - A simple way: converting whole and fraction into binary
    - 3.1415927 = 11.001001000011111011011
  - Scientific notation: 3.1415927 =
  - Not unique:
    - $3.1415927 = 31415927 \times 10^{-7} = 0.31415927 \times 10^{1} = \ldots$

Exponent

$= 3.1415927 \times 10^{0}$

Significant

- 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 $\sqrt{-5}$)

$\pi \approx 3.1415927 \times 10^{0} \approx 1.5707964 \times 2^{1}$

$\approx \quad +1.10010010000111111011011 \times 2^{00000001};$

$\rightarrow \quad +.10010010000111111011011 \times 2^{00000001};$ omit default left-most 1

Sign Exponent Significant

$=$ **0**10000000**10010010000111111011011**; 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
>
```

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")
        }
}
```

- 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 Abs(X+Y-Z) < 10^{-12}

3.4.3 Review ASCII, Unicode, UTF-8

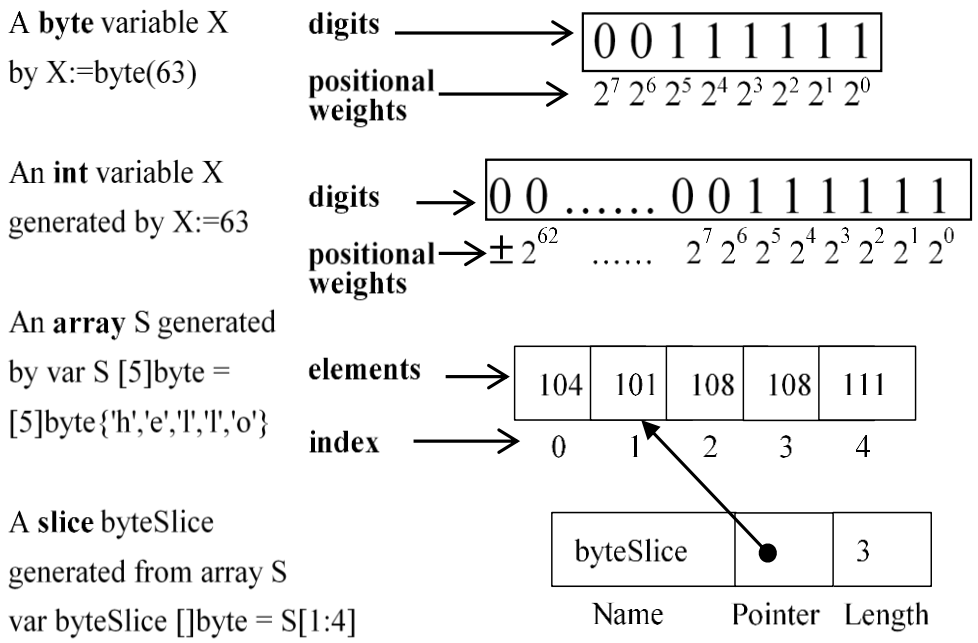
- ASCII encodes English characters, using 7 bits
- Unicode encodes the world's characters
  - using 0000~FFFF$_{16}$, 10000~10FFFF$_{16}$
    - more than 1 million code points, some combinations reserved
    - A character's Unicode encoding needs at least 16 bits
- UTF-8 (Unicode Transformation Format – 8-bit)
  - A variable-width character encoding implementing Unicode
  - Historical standard; most widely used in Internet

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
𐍈	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

```
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

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

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



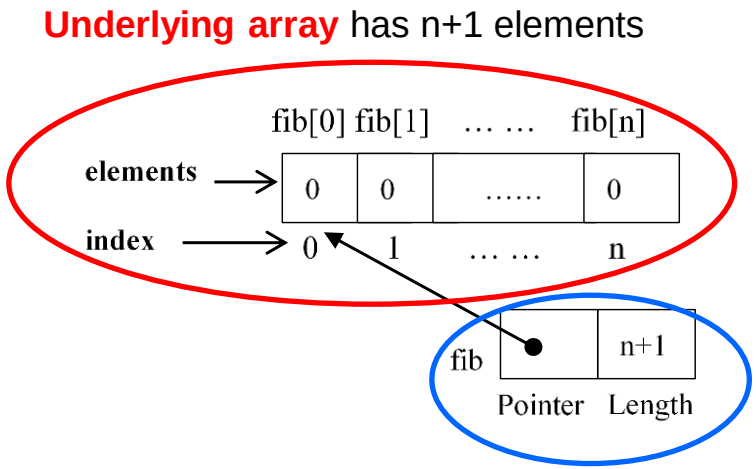
The struct type

- 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
- What if the elements are of different types?
- Use struct
  - 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
    - FanWang.active = false assigns the false value to the active field of student Fan Wang, indicating that he is not actively enrolled

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

Remarks

- byte and int are the types most often used
  - Use int to represent integers
  - Use int to represent characters, assuming leading 0's
    - '?' in previous slide
  - Use byte (which is the same as uint8) to represent ASCII characters and small natural numbers
- 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 ith element of the array, which is initialized to 0. Also, len(fib)=n+1



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: 0011111**1**; Output: 0011111**0**
- 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 = 0011111**1**	Given input
v = $\overline{0}\ \overline{0}\ \overline{1}\ \overline{1}\ \overline{1}\ \overline{1}\ \overline{1}\ \overline{1}$ = 1100000**0**	Bitwise NOT
v = 11000000 & 00000001 = 0000000**0**	Bitwise AND
x = 00111111 & 11111110 = 0011111**0**	Bitwise AND
x = 00111110 | 00000000 = 0011111**0**	Bitwise O

The code can be simplified

- Example 1: Inverting the rightmost bit of a byte
  - Input: 0011111**1**; Output: 0011111**0**

```
x := byte(63)      // assign 63=00111111 to variable x
x = x ^ 0x1        // bitwise XOR of x and 00000001
                   // i.e., 0011111**1** ^ 00000001 = 0011111**0**
```

- However, the Masking mechanism is more general-purpose

Replacing the least significant 2 bits of a byte

(used in Project Text Hider)

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

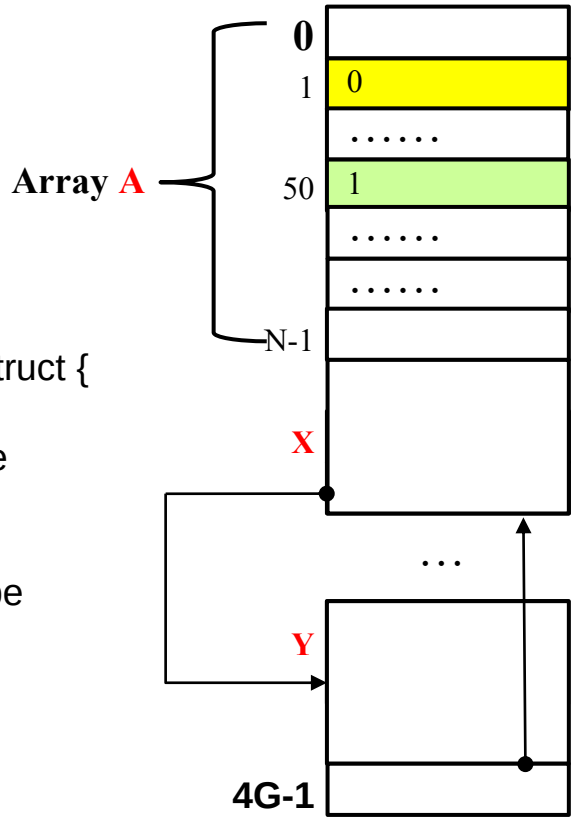
Note: 0x3 = 00000011; 0xFC = 11111100

3.4.7 Pointers and addressing modes

- Array is simple
  - Linear, consecutive arrangement of N elements of the same type
  - 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?

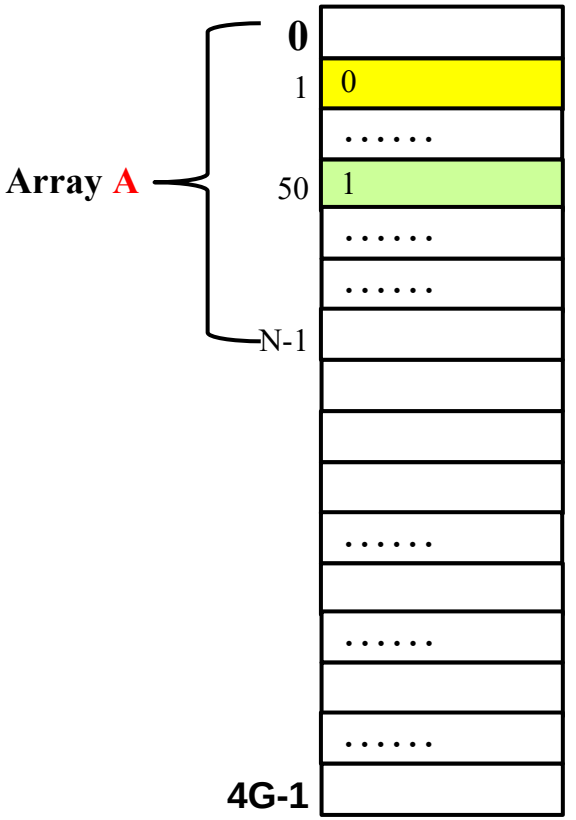
- Pointer brings flexibility
  - Nonlinear arrangements, where the elements can jump around
  - Example: two variables X and Y connected by pointers
    - Indicated by the two arrows

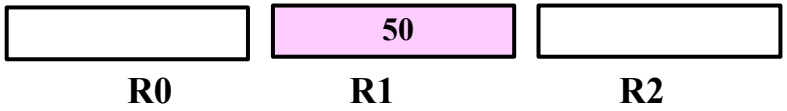
```
type myType struct {
   element int
   next *myType
}

var X, Y myType

X.next = *Y
Y.next = *X
```



Contrasting three addressing modes

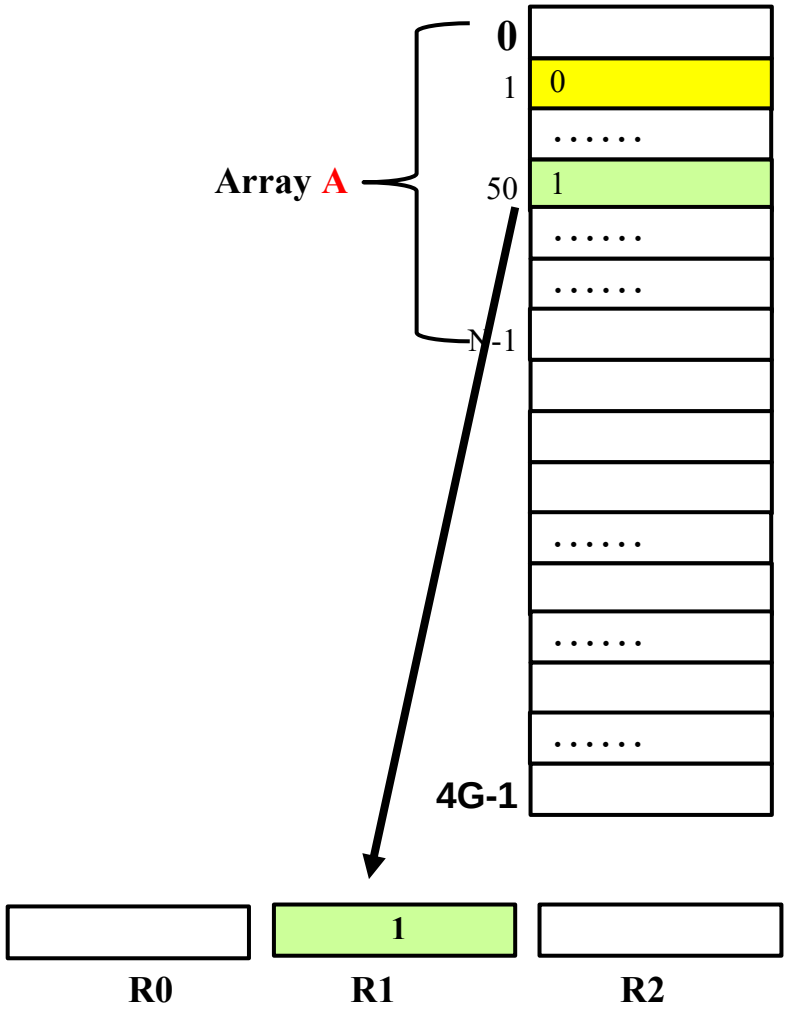
- Array is simple
  - Linear, consecutive arrangement
- Pointer brings flexibility
  - Nonlinear arrangements
- Pointers are supported by the *indirect addressing* mode
  - Contrasting three addressing modes
    - **Immediate mode**: MOV 50, R1;
      - 50 → R1, i.e., R1=50
    - No memory access





Contrasting three addressing modes

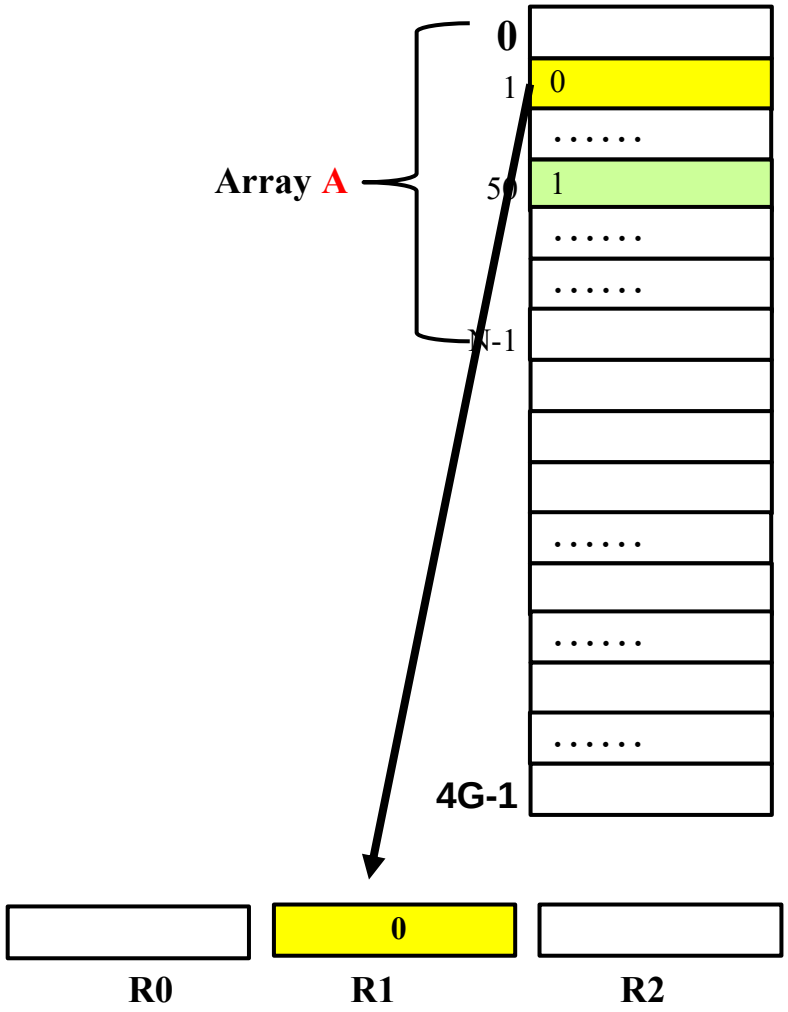
- Array is simple
  - Linear, consecutive arrangement
- Pointer brings flexibility
  - Nonlinear arrangements
- Pointers are supported by the *indirect addressing* mode
  - Contrasting three addressing modes
    - Immediate mode: MOV 50, R1;
      - 50 → R1, i.e., R1=50
    - **Direct mode**: MOV M[50],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



Contrasting three addressing modes

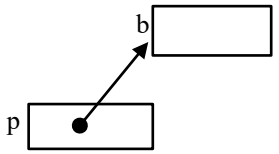
- Array is simple
  - Linear, consecutive arrangement
- Pointer brings flexibility
  - Nonlinear arrangements

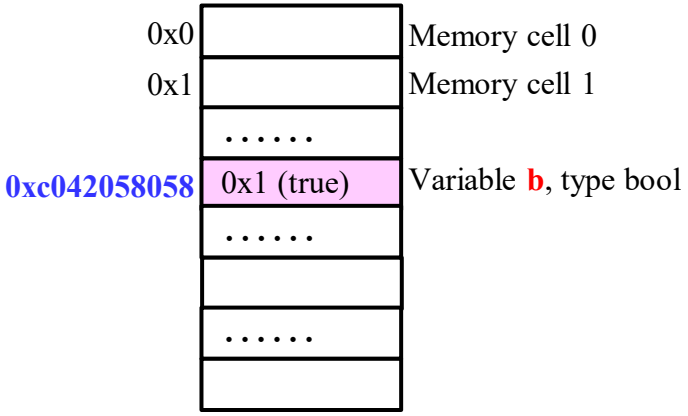
- Pointers are supported by the *indirect addressing* mode
  - Contrasting three addressing modes
    - Immediate mode: MOV 50, R1;
      - 50 → R1, i.e., R1=50
    - Direct mode: MOV M[50], R1;
      - M[50] → R1, i.e., R1=1
    - **Indirect mode**: MOV M[M[50]], R1;
      - M[M[50]] → R1, i.e., M[M[50]] is M[1], R1=M[1]=0
      - First get the address
      - Then access for the value



Step-by-step Illustration of pointers

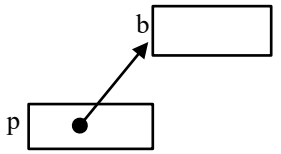
```
func main() {
   b := true            // Boolean variable b
   p := & b             // p holds b's address
   fmt.Println(p)       // Print b's address
   fmt.Println(*p)      // Print b's value; dereference p
   *p =  false          // Modify b's value
   fmt.Println(b)       // Print b's value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





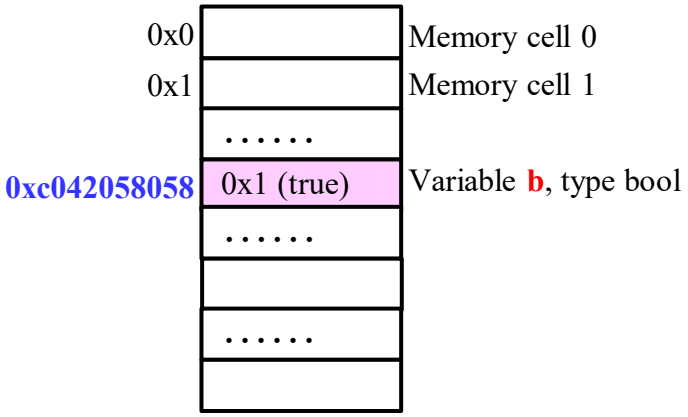
Step-by-step Illustration of pointers

```
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)
}
```



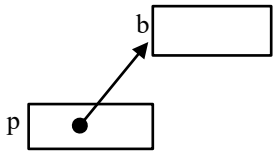
1-minute Quiz:

What is the output of the final statement?



Step-by-step Illustration of pointers

```
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
>
```



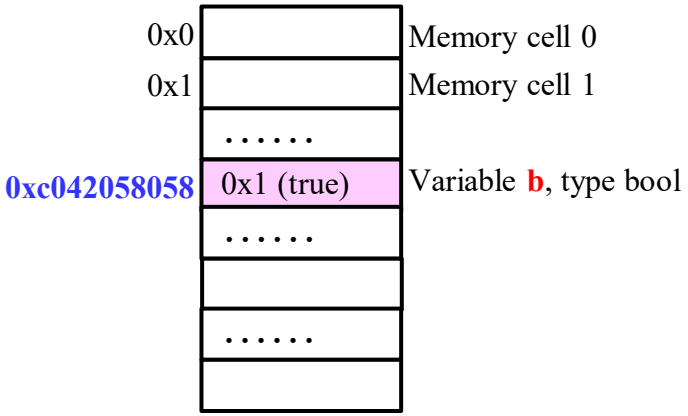
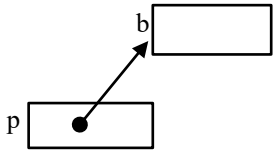


Illustration of pointers

```
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
>
```

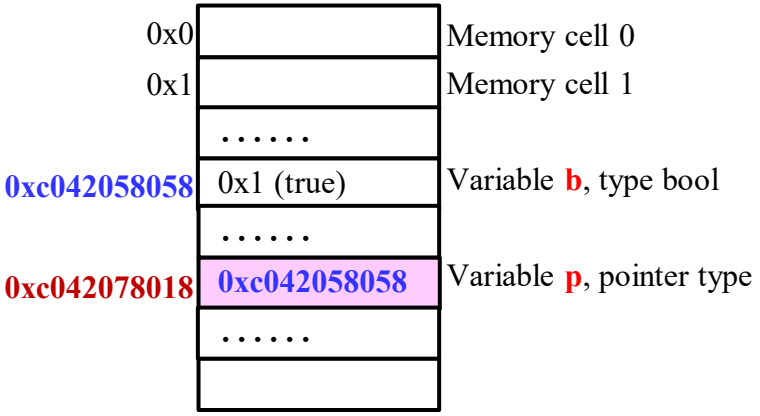
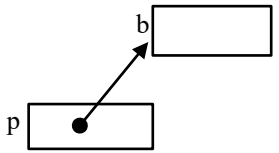


Illustration of pointers

```
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
>
```



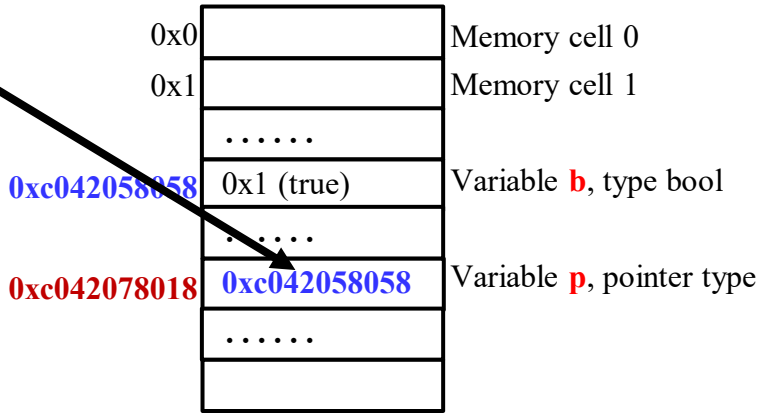
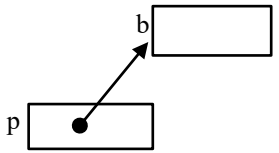


Illustration of pointers

```
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
   *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
>
```



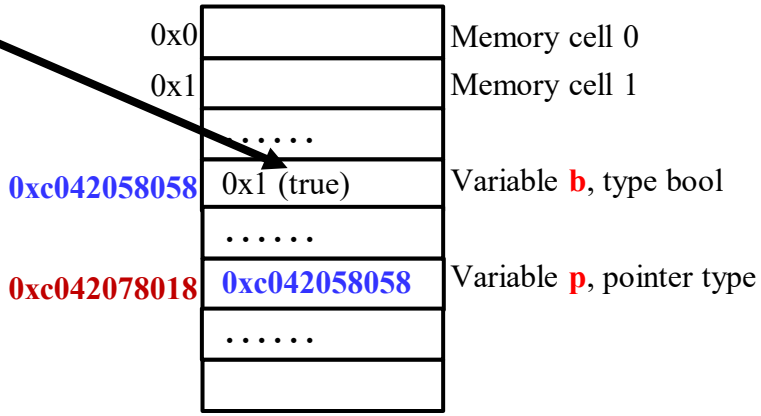
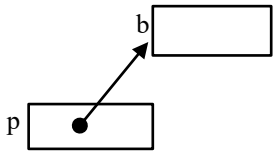


Illustration of pointers

```
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
>
```

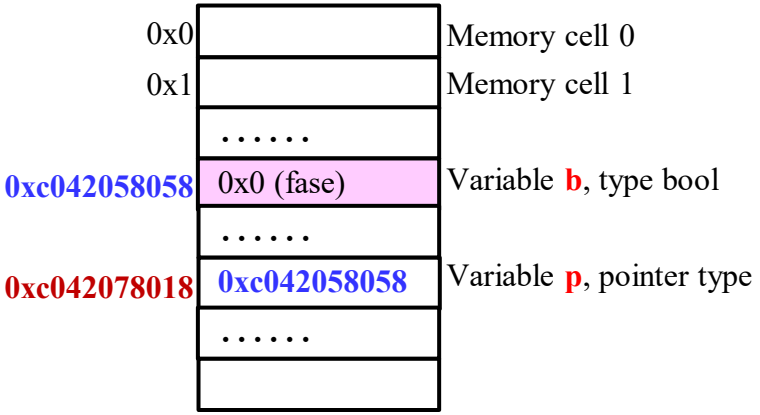
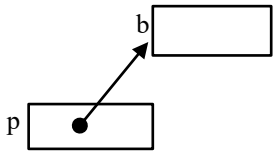


Illustration of pointers

```
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
>
```



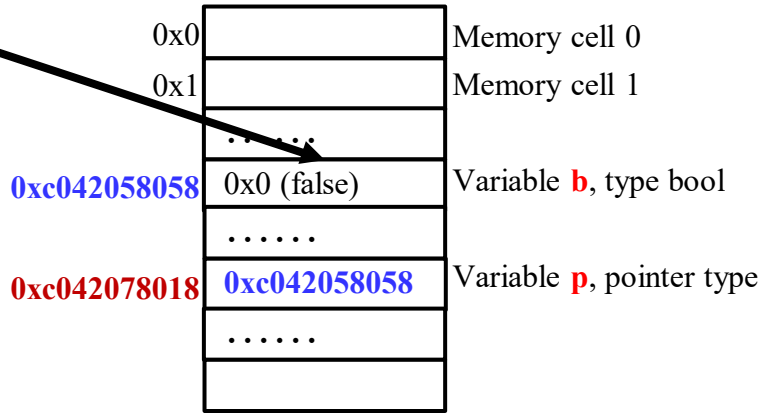
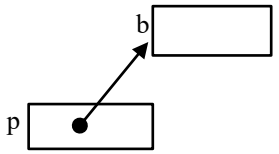


Illustration of pointers

```
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 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
>
```

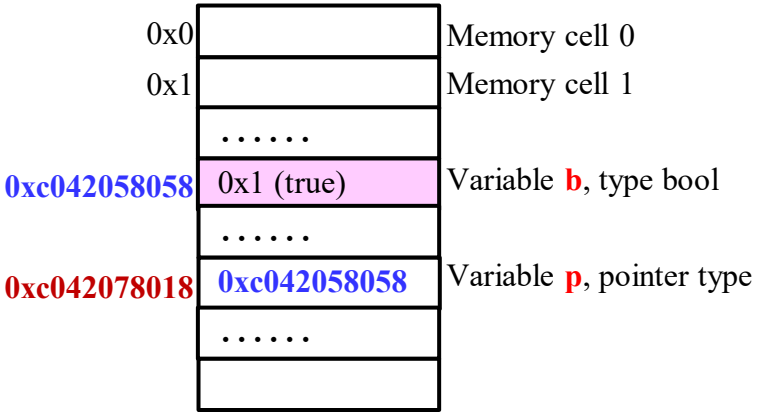
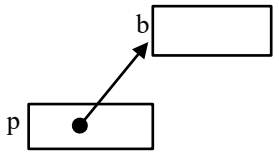
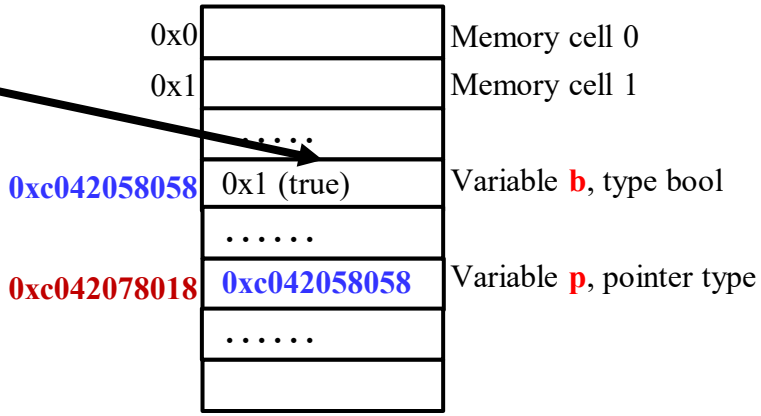


Illustration of pointers

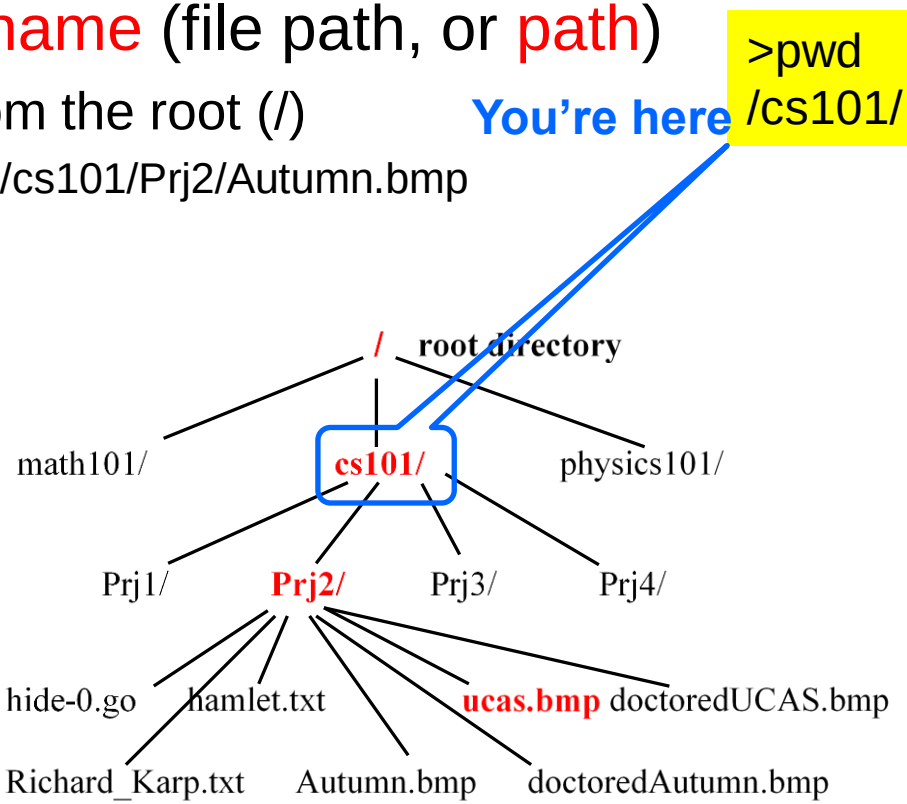
```
func main() {
   b := true            // Boolean variable b
   p := & b             // p holds b's address
   fmt.Println(p)       // Print b's address
   fmt.Println(*p)      // Print b's value
   *p =  false          // Modify b's value
   fmt.Println(b)       // Print b's value
   *p = !(*p) 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

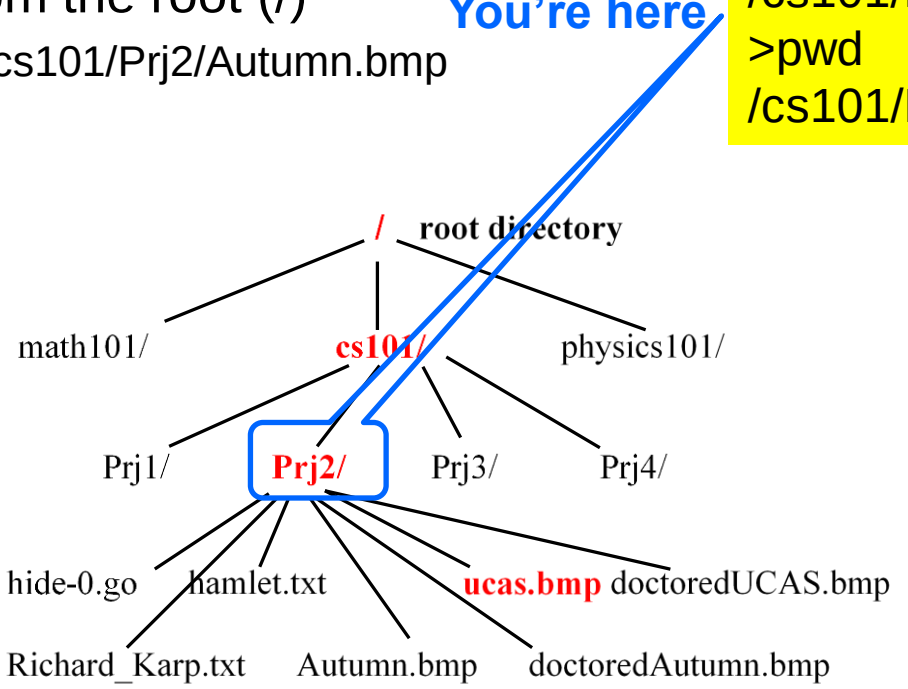
- To organize and **persistently** store chunks of information
- Files are organized as a hierarchy (tree)
  - Leaf nodes are files; internal nodes are directories (special files)
- A file is identified by a file name (file path, or path)
  - Absolute path: all the way from the root (/)
    - Absolute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



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



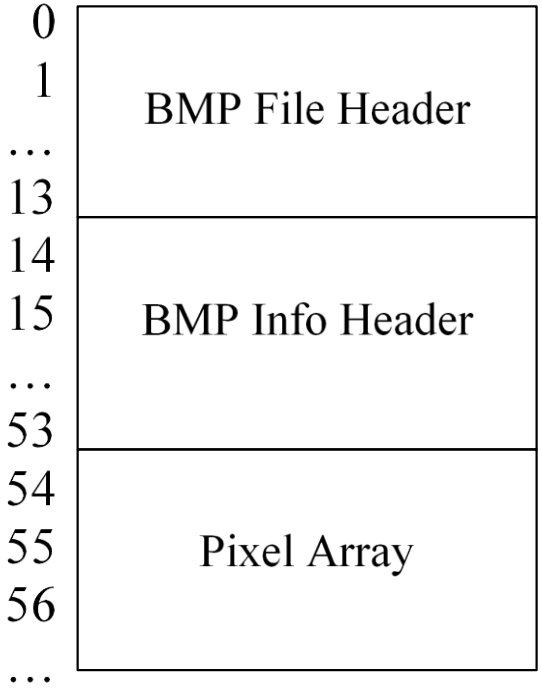
Look inside a file: data and metadata

- Data and metadata of file Autumn.bmp
- Data: bits of the actual picture (Pixel Array)
- Metadata: data about the picture data
  - BMP format in the file: File Header, Info Header
  - Other data associated with the file

Addresses 0~53 hold metadata

The pixel array for actual image data starts at address 54

Autumn.bmp



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

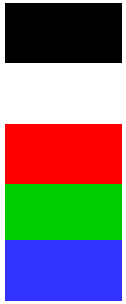
Other types of metadata

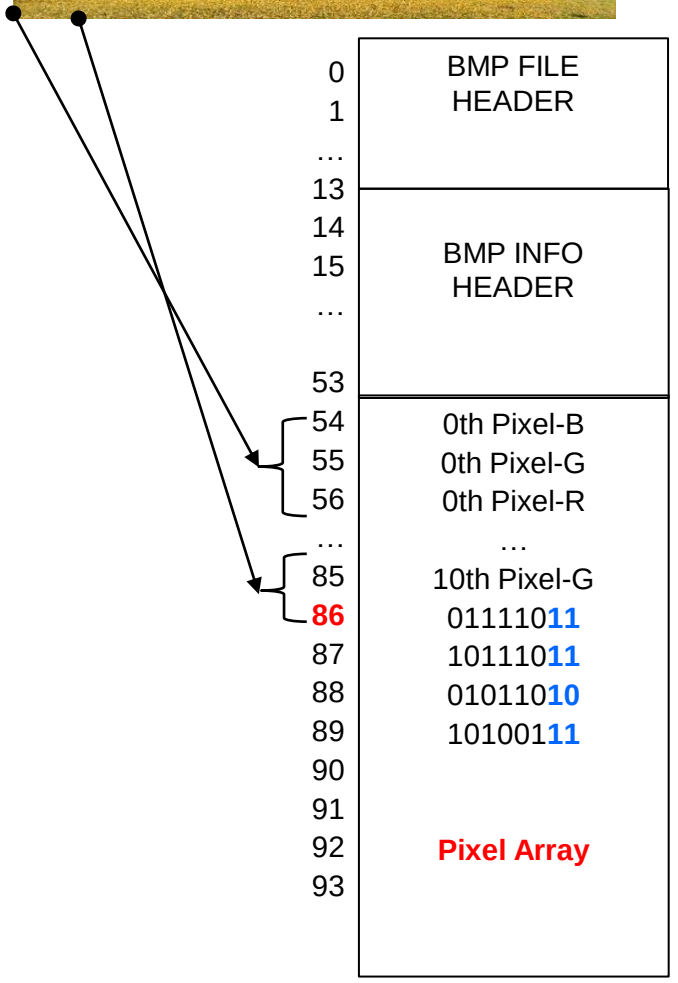
- Can be seen by running “ls -l Autumn.bmp”
  - The file name, the file size, the time of creation (last modification)
  - Access permissions
    - Rights to read, write, and execute a file by the owner of the file, by the group the owner belonging to, and by other users
- Example of access permissions
  - ioutil.WriteFile("./doctoredAutumn.bmp", p, 0666)
  - Every user can to 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	-

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

- Pixel = Picture Element
- Pixel Array holds the pixels of the image, starting from the low-left corner going right, and then up row by row
- Each pixel has three bytes for
  - Color depth values of RGB, i.e., the primary colors of red, green, blue
  - 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



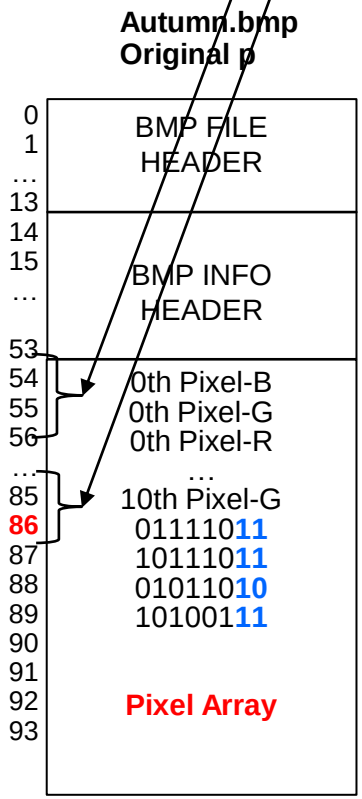


How to hide the length of a text file in a picture?

We need the length when recover in show.go

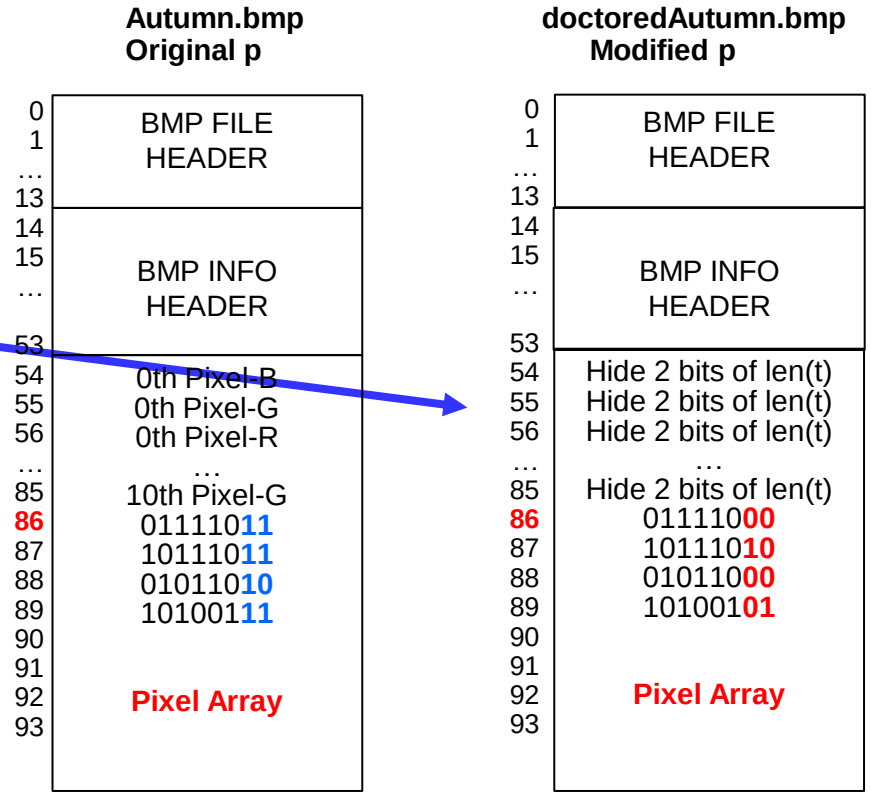
- p, _ := ioutil.ReadFile("./Autumn.bmp") to read the image file into byte array p

- 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]



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

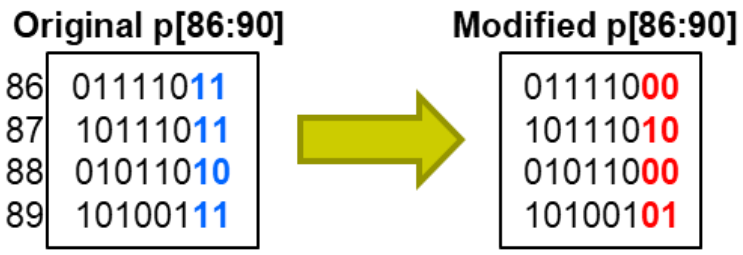
0	BMP FILE HEADER
1	
…	
13	
14	BMP INFO HEADER
15	
…	
53	
54	0th Pixel-B
55	0th Pixel-G
56	0th Pixel-R
…	…
85	10th Pixel-G
86	01111011
87	10111011
88	01011010
89	10100111
90	
91	
92	Pixel Array
93	

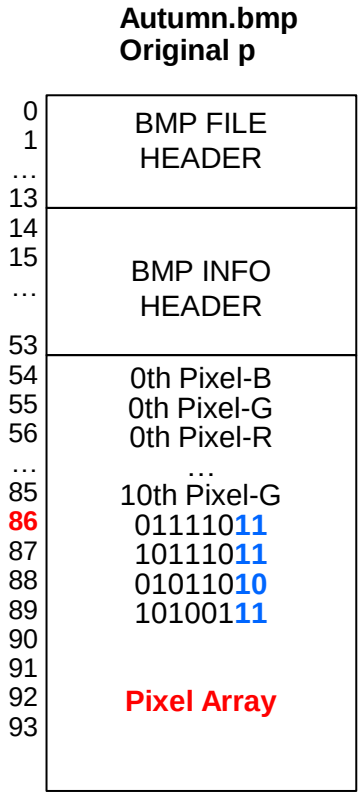
doctoredAutumn.bmp
Modified p

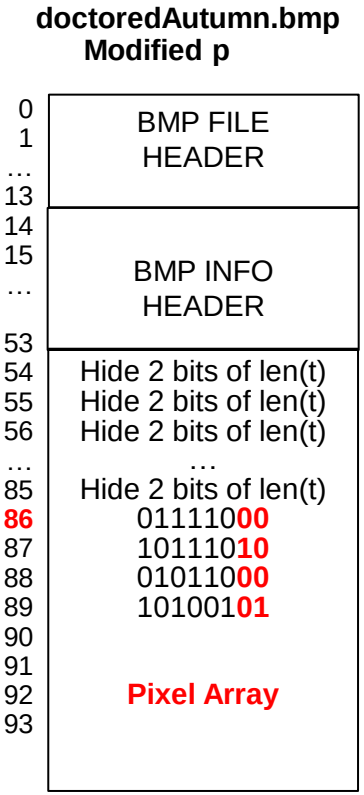
0	BMP FILE HEADER
1	
…	
13	
14	BMP INFO HEADER
15	
…	
53	
54	Hide 2 bits of len(t)
55	Hide 2 bits of len(t)
56	Hide 2 bits of len(t)
…	…
85	Hide 2 bits of len(t)
86	01111000
87	10111010
88	01011000
89	10100101
90	
91	
92	Pixel Array
93	

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
- 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]







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

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 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	

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

- 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

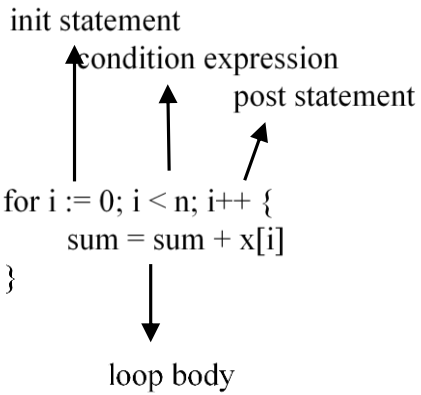
doctoredAutumn.bmp
Modifying leftmost 2 bits

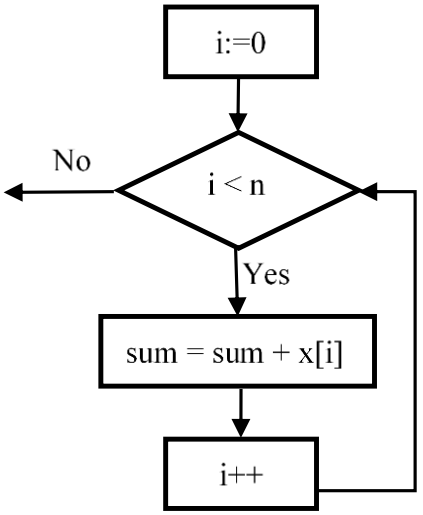
3.5 Review of control abstractions

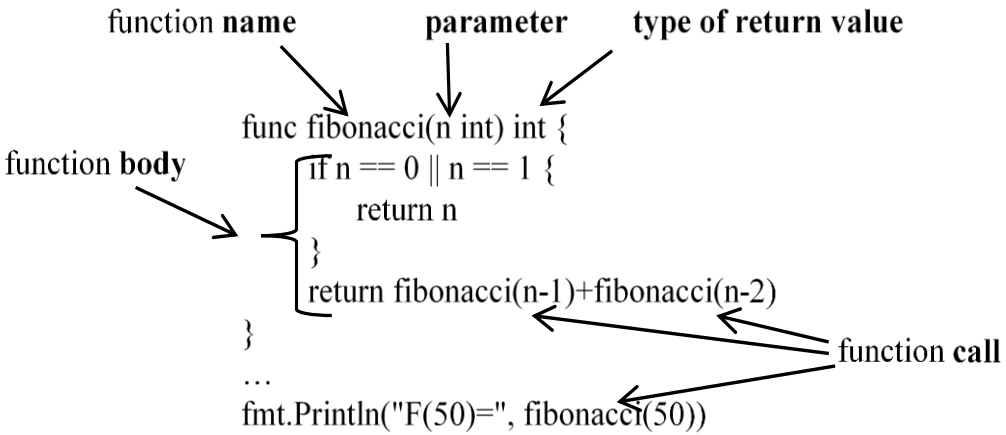
- Precedence in an expression
  - For x*b+c || i < 7, the precedence is ((x*b)+c) || (i < 7)
  - When in doubt, use parentheses
- Sequence of statements: follow the syntactic order
- Selection (conditional): if-then-else statement

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

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



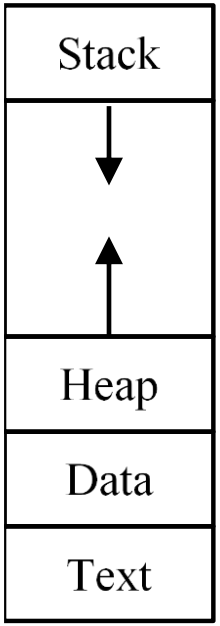




Segments of Text, Data, Heap and Stack

- 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)
}
```



Holds the context of function calls, growing downwards

Holds dynamic data, growing upwards

Holds static data

Holds the code of the process

*** 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 donot 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:])
}
```

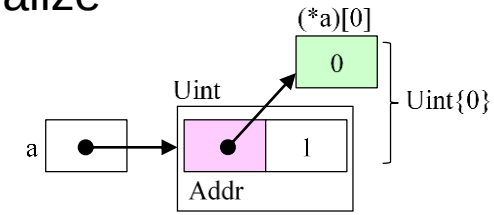
Students donot need to Know the internal of String

*** Some execution details

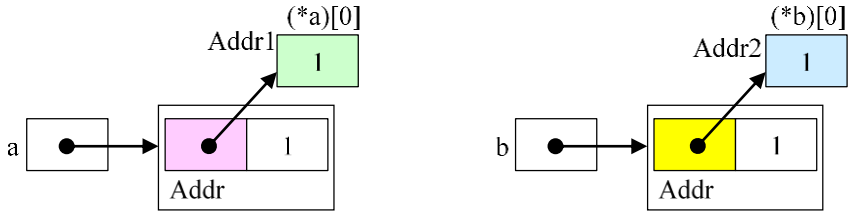
```
package main  // fib.Uint.go
import (
   "fmt"
   "math"
)
func main() {
   fmt.Printf("F(100) = %s\n", String(*(fibonacci(100))))
}
type Uint []uint64
func fibonacci(n int) *Uint {
   a := &Uint{0}                 // a = 0
   b := &Uint{1}                 // b = 1
   for i := 1; i < n+1; i++ {
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

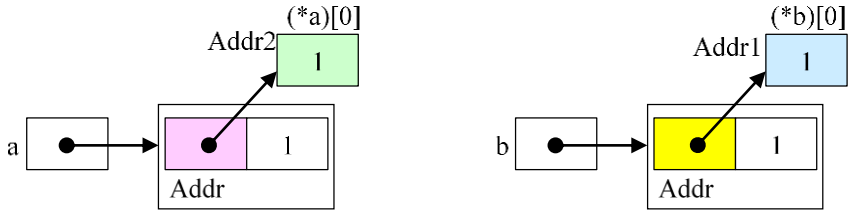
```
> go run fib.Uint.go
F(100) = 354224848179261915075
>
```

Initialize



When i = 1

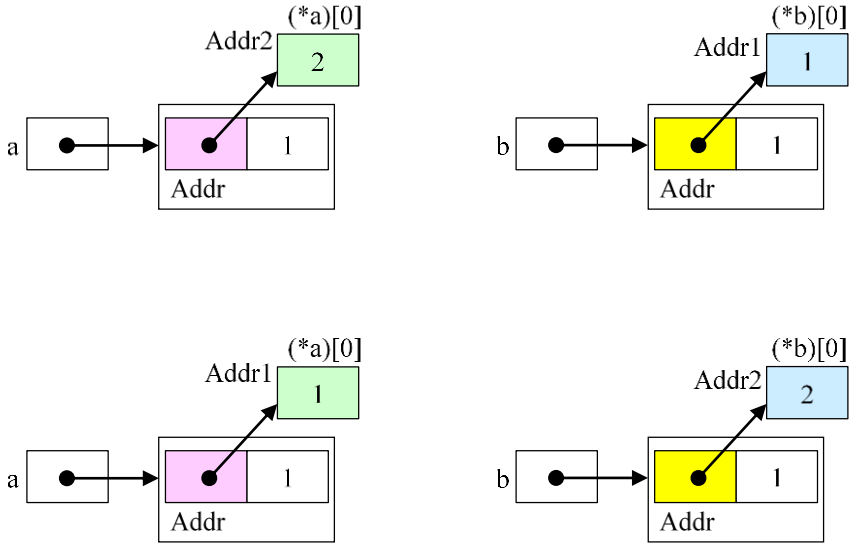




*** Some execution details

```
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

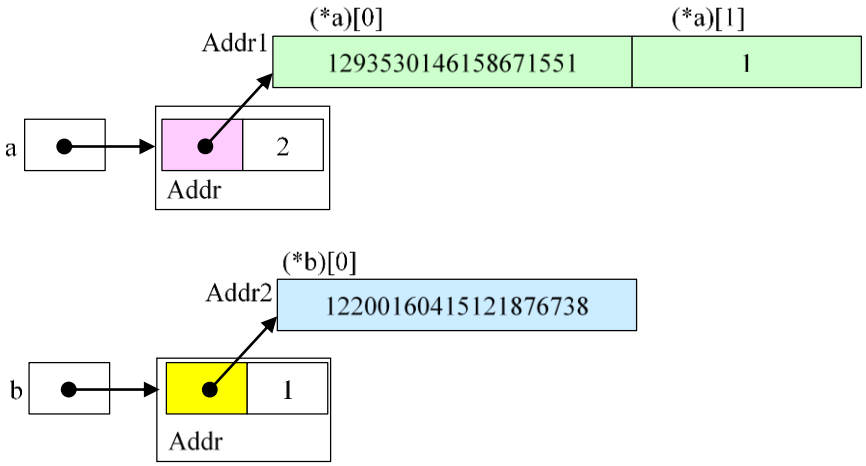



```
> go run fib.Uint.go
F(100) = 354224848179261915075
>
```

*** 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
>
```