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

计算机科学导论实验

编程基础-1

从hello到Name2Number.go

基本数据类型、基本语句、循环

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

提纲

- 目标：理解编程入门知识
  - 编辑、编译、执行、调试的程序开发生命周期
  - 程序结构、声明、赋值语句、字符串、数组、for循环、打印语句
  - 良好编码习惯

- 步骤
  - 在云计算环境和本机环境重现hello.go
  - 理解并重现name_to_number-0.go，用你自己的姓名拼音
  - 理解并重现name_to_number.go，用你自己的姓名拼音
  - 开发Name2Number.go
    - 将name_to_number.go改造为符合四条良好编程习惯的程序

课件中包含教科书未包括的素材引用，特此致谢

1. 执行几个简单程序

重现：云计算环境、本机环境

```
package main   // The main package
func main() {  // a main function that does nothing
}              // 它的主函数体是空的
```

命令提示符
Command
prompt

```
> code null.go
> go build null.go
> ./null
>
```

编辑命令
编译命令
执行命令

光标
cursor

Shell是命令行解释器

- null.go
  - 程序有两个语句（statement）
  - 同学们写的程序属于主包（main package）
  - 每个主包有一个主函数（main function）
- hello.go
  - fmt是Go语言自带的程序包
    称为库（library）
    - 其他人开发了fmt供用户重用，
      fmt 包含函数fmt.Println
    - 注意：点号标记法（dot notation）
  - 程序中的函数很像数学函数
    - 输入数据→输出数据

  但可包含副作用（side effect）
    - 此例hello.go程序的主函数
      只有副作用，即打印hello!

```
package main   // hello.go程序的主包
import "fmt"   // 该程序导入一个库包fmt
func main() {  // 该程序声明一个主函数
  fmt.Println("hello!")  // 输出语句打印出hello!
}
```

```
> go build hello.go
> ./hello
hello!
>
```

Compile and execute
in two commands

```
> go run hello.go
hello!
>
```

Compile and execute in one command
两个命令可以合并成一个命令

本课程调用fmt软件包中的三个函数

```
package fmt                  // The fmt package，它不是主程序包，而是库包
……
func Println(…) …{           // It contains a function Println which other
      …                      // programs can call by using fmt.Println
}                                              点号标记法（dot notation）
……
func Printf(…) …{            // It contains a function Printf which other
  …                          // programs can call by using fmt.Printf
}
……
func Scanf(…) …{             // It contains a function Scanf which other
     …                       // programs can call by using fmt.Scanf
}
```

两个输出语句 The following two statements do the same thing.

```
fmt.Printf("hello! %d\n",63)
fmt.Println("hello!",63)
```

输出结果相同

```
> hello! 63
```

可能出现各种错误

- 将打印语句fmt.Println(“hello!”) 替换成
  - fmt.Printf(“hello!\n”)　　必须显式加换行符（即\n转义符）
  - fmt.Printf(“%c%c%c%c%c%c\n”,XXX)
    - 用对应“hello!”的正确的ASCII字符编码替换XXX

产生同样输出结果，即打印出hello!

- 重现编译错误

```
package main
import "fmt"
func main() {
        fmt.Printf("hello!\n")
        fmt.Printf("%c%c%c%c%c%c\n",'h',“e”,"l",'l','o','!')
}
```

重现运行时错误

```
package main
import "fmt"
func main() {
        fmt.Printf("hello!\n")
        fmt.Printf("%c%c%c%c%c%c\n", "hello!")
}
```

可能出现各种错误

- 将打印语句fmt.Println(“hello!”) 替换成
  - fmt.Printf(“hello!\n”)　　必须显式加换行符（即\n转义符）
  - fmt.Printf(“%c%c%c%c%c%c\n”,XXX)
    - 用对应“hello!”的正确的ASCII字符编码替换XXX

产生同样输出结果，即打印出hello!

- 重现编译错误

```
package main
import "fmt"
func main() {
        fmt.Printf("hello!\n")
        fmt.Printf("%c%c%c%c%c%c\n",'h',“e”,"l",'l','o','!')
}
```

重现运行时错误

```
package main
import "fmt"
func main() {
        fmt.Printf("hello!\n")
        fmt.Printf("%c%c%c%c%c%c\n", "hello!")
}
```

正确代码是

```
package main
import "fmt"
func main() {
        fmt.Printf("hello!\n")
        fmt.Printf("%c%c%c%c%c%c\n", 104, 101, 108, 108, 111, 33)
}
```

Go程序的基本结构

主包声明语句	package main	// 定义主包
导入语句	import …	// 导入其他人写的包，如不调用则不出现
常量声明语句	const …	// 声明常量，可不出现
变量声明语句	var …	// 声明变量，可不出现
函数声明语句	func X(…) …{…}	// 声明程序员自定义的函数，可不出现
主函数声明语句	func main() {	// 声明主包中必须有的主函数
	…	// 主函数的函数体
	}	
	main()	**// 缺省的主函数调用，不出现**

```
func X(…) …{                    // 如函数体等括起来的{…}称为一个代码块（code block）
   常量声明语句                  // 局部常量
   变量声明语句                  // 局部变量
   赋值语句
   循环语句
   条件判断语句
   函数调用语句，如打印语句
}
```

2.1 从字符串到数的变换

- 两个程序例子："Alan Turing"变换到1045
  - 使用%d的简版程序　　name_to_number-0.go
  - 使用%c的程序　　name_to_number.go
  - 哈希进阶实验，实时查找10亿人（如微信登录）
    - O(N) → O(logN) → O(1)　10亿步 → 30步 → 1步

- 新出现的变量类型与语句
  - 变量类型
    - 整数：int
    - 符号串：string
      - 字符数组
  - 语句类型
    - 赋值语句
    - 循环语句（for loop）

```
package main     //name_to_number-0.go
import "fmt"
func main() {
        var name string = "Alan Turing"
        sum := 0
        for i := 0; i < len(name); i++ {
                sum = sum + int(name[i])
        }
        fmt.Printf("%d\n", sum)
}
```

ASCII encodings for "Alan Turing"

= [65, 108, 97, 110, 32, 84, 117, 114, 105, 110, 103]

$D_3D_2D_1D_0$ \ $D_6D_5D_4$	000	001	010	011	100	101	110	111
0000	NUL	DLE	SP	0	@	P	`	p
0001	SOH	DC1	!	1	A	Q	a	q
0010	STX	DC2	”	2	B	R	b	r
0011	ETX	DC3	#	3	C	S	c	s
0100	EOT	DC4	$	4	D	T	d	t
0101	ENQ	NAK	%	5	E	U	e	u
0110	ACK	SYN	&	6	F	V	f	v
0111	BEL	ETB	’	7	G	W	g	w
1000	BS	CAN	(	8	H	X	h	x
1001	HT	EM	)	9	I	Y	i	y
1010	LF	SUB	*	:	J	Z	j	z
1011	VT	ESC	+	;	K	[	k	{
1100	FF	FS	,	<	L	\	l	|
1101	CR	GS	-	=	M	]	m	}
1110	SO	RS	.	>	N	^	n	~
1111	SI	US	/	?	O	_	o	DEL

name_to_number-0.go

- 符号串string是字符数组
- 声明变量的两种方式

```
var name string
name = "Alan Turing"
```

等价于

```
var name string = "Alan Turing"
```

基本等价于

```
name := "Alan Turing"
```

```
package main    // name_to_number-0.go
import "fmt"
func main() {
        var name string = "Alan Turing"
        sum := 0
        for i := 0; i < len(name); i++ {
                sum = sum + int(name[i])
        }
        fmt.Printf("%d\n", sum)
}
```

注意：name[4] 是空格' ' =32

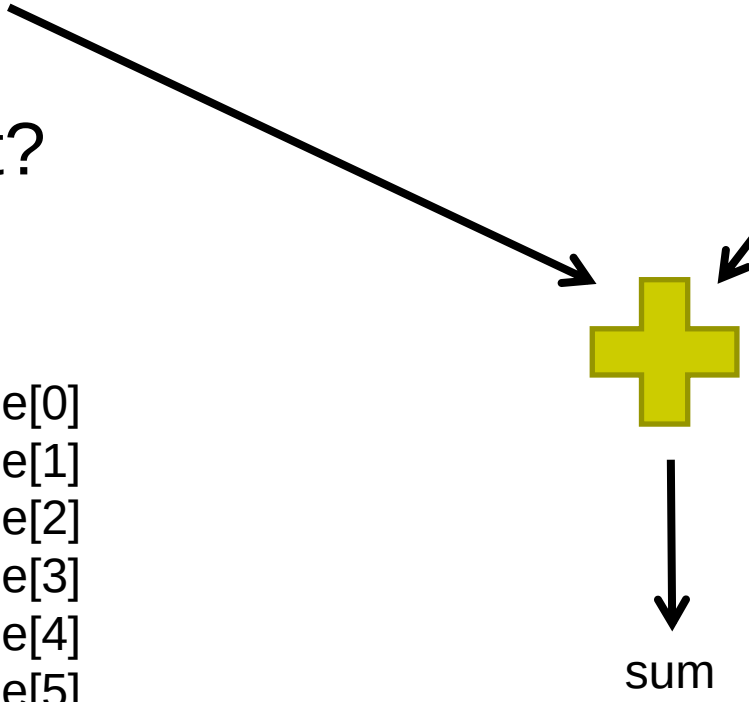
	A	l	a	n		T	u	r	i	n	g
name = [65	108	97	110	32	84	117	114	105	110	103]	
index →	0	1	2	3	4	5	6	7	8	9	10

len(name) is 11

name[0]='A'=65, name[1]='l'=108, name[2]='a'=97,
name[3]='n'=110, name[4]=' '=32, name[5]='T'=84,
name[6]='u'=117, name[7]='r'=114, name[8]='i'=105,
name[9]='n'=110, name[10]='g'=103.

How to add up elements of an array?

- Problem: add up the 11 elements of name[i]
  - name = [65 108 97 110 32 84 117 114 105 110 103]



- How to do it?
  - Solution 1

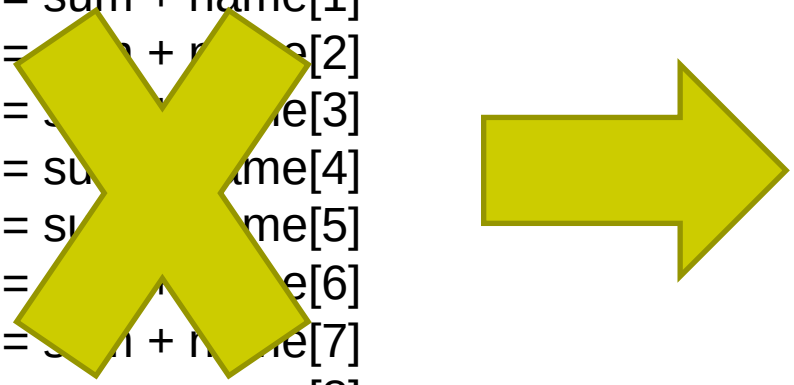
```
sum := 0
sum = sum + name[0]
sum = sum + name[1]
sum = sum + name[2]
sum = sum + name[3]
sum = sum + name[4]
sum = sum + name[5]
sum = sum + name[6]
sum = sum + name[7]
sum = sum + name[8]
sum = sum + name[9]
sum = sum + name[10]
```

How to add up elements of an array?

- Problem: add up the 11 elements of name[i]
  - name = [65 108 97 110 32 84 117 114 105 110 103]

- How to do it?
  - Solution 1

```
sum := 0
sum = sum + name[0]
sum = sum + name[1]
sum = sum + name[2]
sum = sum + name[3]
sum = sum + name[4]
sum = sum + name[5]
sum = sum + name[6]
sum = sum + name[7]
sum = sum + name[8]
sum = sum + name[9]
sum = sum + name[10]
```



Solution 2 Why?

```
sum := 0
sum = sum + int(name[0])
sum = sum + int(name[1])
sum = sum + int(name[2])
sum = sum + int(name[3])
sum = sum + int(name[4])
sum = sum + int(name[5])
sum = sum + int(name[6])
sum = sum + int(name[7])
sum = sum + int(name[8])
sum = sum + int(name[9])
sum = sum + int(name[10])
```

Add up [65 108 97 110 32 84 117 114 105 110 103]

name[0] = 65 = 01000001

Value Name	Binary representation
sum (right-side)	0000000000000000000000000000000000000000000000000000000000000000
name[0]	01000001
int(name[0])	0000000000000000000000000000000000000000000000000000000001000001
sum (left-side)	0000000000000000000000000000000000000000000000000000000001000001

var name string = "Alan Turing"
var sum int = 0

Type of name: an array of byte, i.e., uint8
Type of sum: 64-bit integer

```
sum := 0
sum = sum + name[0]
sum = sum + name[1]
…
sum = sum + name[10]
```

Solution 1

```
sum := 0
sum = sum + int(name[0])
sum = sum + int(name[1])
…
sum = sum + int(name[10])
```

Solution 2

Add up [65 108 97 110 32 84 117 114 105 110 103]

name[0] = 65 = 01000001

Value Name	Binary representation
sum (right-side)	00000000000000000000000000000000000000000000000000000000
name[0]	01000001
int(name[0])	00000000000000000000000001000001
sum (left-side)	00000000000000000000000001000001

类型转换操作
Type casting operation int(name[0])
converts byte type to int type
By padding 56 0's

```
sum := 0
sum = sum + name[0]
sum = sum + name[1]
…
sum = sum + name[10]
```

Solution 1

```
sum := 0
sum = sum + int(name[0])
sum = sum + int(name[1])
…
sum = sum + int(name[10])
```

Solution 2

Add up [65 108 97 110 32 84 117 114 105 110 103]

name[1] = 108 = 01101100

Value Name	Binary representation
sum (right-side)	00000000000000000000000000000000000000000000000000000001000001
name[1]	01101100
int(name[1])	00000000000000000000000000000000000000000000000000000001101100
sum (left-side)	00000000000000000000000000000000000000000000000000000010101101

sum := 0
sum = sum + int(name[0])
= 0 + 65 = 65
= 00000000000000000000000000000000000000000000000000000001000001
sum = sum + int(name[1])
= 65 + 108 = 173
= 00000000000000000000000000000000000000000000000000000010101101

Code for solution 2 still has problems

1. Tied to particular students 绑定了学生姓名
   - Does not work for students with len(name) != 11
2. Tedious, repetitive code 重复代码
3. The length of code is proportional to the problem size! 直线程序
   - **A sign of possible bad design**
     - What if len(name) == 1 million?
   - 违反了程序有限性原则

- Project 1: Turing Adder
  - We don't want the size of transition table of Turing machine to be proportional to input length N

```
sum := 0
sum = sum + int(name[0])
sum = sum + int(name[1])
sum = sum + int(name[2])
sum = sum + int(name[3])
sum = sum + int(name[4])
sum = sum + int(name[5])
sum = sum + int(name[6])
sum = sum + int(name[7])
sum = sum + int(name[8])
sum = sum + int(name[9])
sum = sum + int(name[10])
```

幸好Go语言提供了for loop循环抽象

- 初始化: 数组索引从零开始 i = 0
- 重复执行循环体，每次迭代索引加1，直到i≥11

```
package main
import "fmt"
func main() {
   var name string = "Alan Turing"
   sum := 0
   sum = sum + int(name[0])
   sum = sum + int(name[1])
   sum = sum + int(name[2])
   sum = sum + int(name[3])
   sum = sum + int(name[4])
   sum = sum + int(name[5])
   sum = sum + int(name[6])
   sum = sum + int(name[7])
   sum = sum + int(name[8])
   sum = sum + int(name[9])
   sum = sum + int(name[10])
   fmt.Printf("%d\n", sum)
}
```

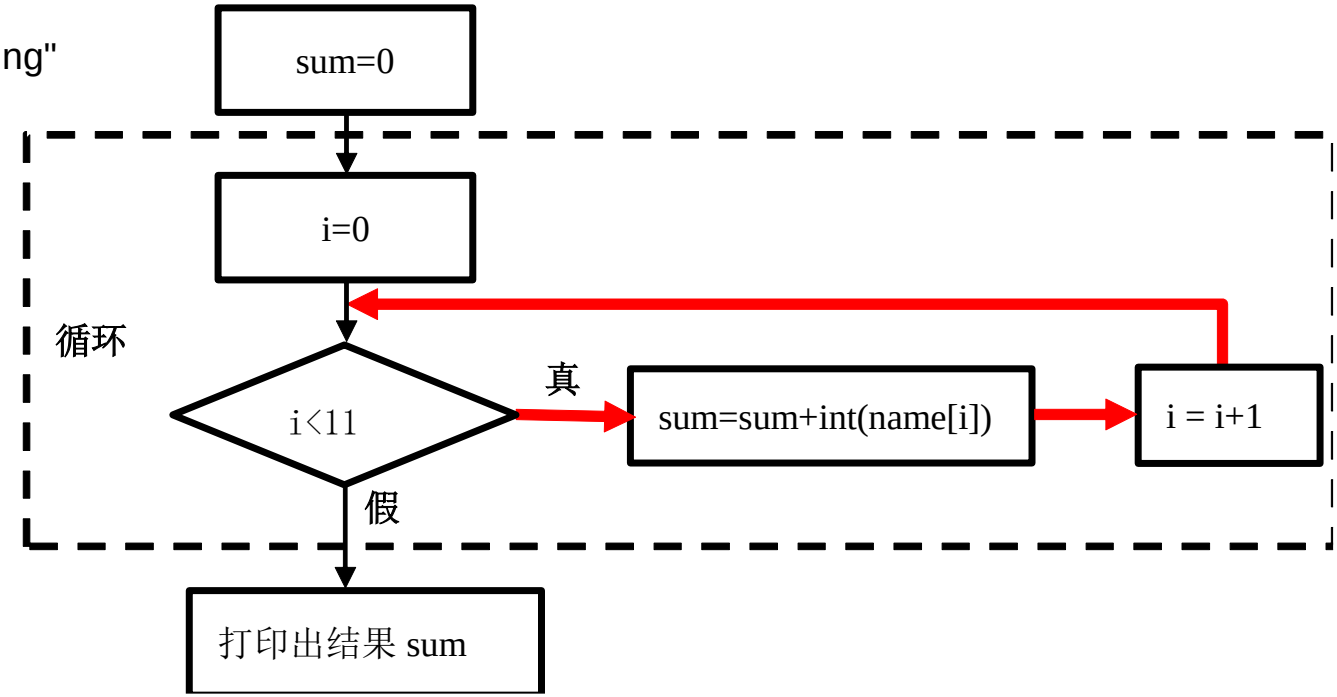
```
package main
import "fmt"
func main() {
   var name string = "Alan Turing"
   sum := 0
   for i := 0; i < 11; i++ {
      sum = sum + int(name[i])
   }
   fmt.Printf("%d\n", sum)
}
```

循环抽象的诀窍：综合变与不变
不变：for循环结构不变
变：索引值变化，累加值sum变化

for loop循环抽象

- 初始化: 数组索引从零开始 i = 0
- 重复执行循环体，每次迭代索引加1，直到i≥11

```
package main
import "fmt"
func main() {
   var name string = "Alan Turing"
   sum := 0
   for i := 0; i < 11; i++ {
      sum = sum + int(name[i])
   }
   fmt.Printf("%d\n", sum)
}
```



理解并重现name_to_number-0.go

- 用你自己的姓名拼音，手算姓名编码
  - 如“Xu Zhi Wei”的姓名编码是861
    - 88+117+32+90+104+105+32+87+101+105 = 861
- 用你自己的姓名拼音修改后程序，并向自己说明程序含义
  - 例如，当i=1时，sum = sum + int(name[i])语句中下列值是什么？
    - 左边sum，右边sum，int(name[i])，name[i]
    - 为什么是sum = sum + int(name[i])，而不是sum = sum + name[i]
- 运行修改后程序，验证程序结果正确

```
package main     //name_to_number-0.go
import "fmt"
func main() {
        var name string = "Alan Turing"       // 用你自己的姓名拼音
        sum := 0
        for i := 0; i < len(name); i++ {
                sum = sum + int(name[i])
        }
        fmt.Printf("%d\n", sum)
}
```

理解并重现name_to_number-0.go

- 程序中的关键字、作用域、全局变量、局部变量
  - 列出所有关键字
  - 列出变量的作用域：name，sum，i
  - 列出全局变量
  - 列出局部变量

```
package main     //name_to_number-0.go
import "fmt"
func main() {
        var name string = "Alan Turing"       // 用你自己的姓名拼音
        sum := 0
        for i := 0; i < len(name); i++ {
                sum = sum + int(name[i])
        }
        fmt.Printf("%d\n", sum)
}
```

2.2 采用占位符%c实现格式动词%d

name_to_number-0.go
对比
name_to_number.go

```
package main //name_to_number-0.go
import "fmt"
func main() {
        var name string = "Alan Turing"
        sum := 0
        for i := 0; i < 11; i++ {
                sum = sum + int(name[i])
        }
        fmt.Printf("%d\n", sum)
}
```

```
> ./name_to_number-0
> 1045
>
```

```
package main  // name_to_number.go
import "fmt"
func main() {
   var name string = "Alan Turing"
   sum := 0
   for i := 0; i < 11; i++ {
      sum = sum + int(name[i])
   }
   var sum_bytes [4]byte
   var j int
   for j = 3; sum != 0; j-- {
      sum_bytes[j] = byte(sum%10) + '0'
      sum = sum / 10
   }
   fmt.Printf("%c", sum_bytes[0])
   fmt.Printf("%c", sum_bytes[1])
   fmt.Printf("%c", sum_bytes[2])
   fmt.Printf("%c", sum_bytes[3])
   fmt.Printf("\n")
}
```

```
> ./name_to_number
> 1045
>
```

如何用%c实现%d?

```
package main
import "fmt"
func main() {
   var name string = "Alan Turing"
   sum := 0
   for i := 0; i < 11; i++ {
      sum = sum + int(name[i])
   }
   var sum_bytes [4]byte
   var j int
   for j = 3; sum!= 0; j-- {
      sum_bytes[j] = byte(sum%10) + '0'
      sum = sum / 10
   }
   fmt.Printf("%c", sum_bytes[0])
   fmt.Printf("%c", sum_bytes[1])
   fmt.Printf("%c", sum_bytes[2])
   fmt.Printf("%c", sum_bytes[3])
   fmt.Printf("\n")
}
```

使用整数除法 / 和求余 % 操作，
从数值1045依次摘取出数字5, 4, 0, 1

```
sum % 10
sum = sum / 10
```

sum_bytes[3]	1045 % 10 = 5
sum	1045 / 10 = 104
sum_bytes[2]	104 % 10 = 4
sum	104 / 10 = 10
sum_bytes[1]	10 % 10 = 0
sum	10 / 10 = 1
sum_byte[0]	1 % 10 = 1
sum	1 / 10 = 0

sum_bytes = [‘1’, ‘0’, ‘4’, ‘5’]
数组索引 0 1 2 3

用字符占位符%c实现十进制占位符 %d
fmt.Printf("%d\n", sum)

四个打印语句fmt.Printf(“%c”, …)
依次打印出1, 0, 4, 5四个字符

为什么要做类型转换

sum_bytes[j] = byte(sum%10) + ‘0’
而不是sum_bytes[j] = sum % 10

```
package main
import "fmt"
func main() {
   var name string = "Alan Turing"
   sum := 0
   for i := 0; i < 11; i++ {
      sum = sum + int(name[i])
   }
                  // 此时，sum == 1045
   var sum_bytes [4]byte
   var j int
   for j = 3; sum != 0; j-- {
      sum_bytes[j] = byte(sum%10) + '0'
      sum = sum / 10
   }
   fmt.Printf("%c", sum_bytes[0])
   fmt.Printf("%c", sum_bytes[1])
   fmt.Printf("%c", sum_bytes[2])
   fmt.Printf("%c", sum_bytes[3])
   fmt.Printf("\n")
}
```

sum_bytes is a uint8 array
sum_byte[j] has type byte
However, sum is type int

Suppose j=3 (initially)
sum is 1045, and
sum%10 is
1045%10 = 5, a 64-bit int value
00000000……00000101

byte(sum%10), i.e., byte(5)
converts this 64-bit value
to a number of type uint8 and value
00000101

byte(5)+ ‘0’ evaluates to
= 5 + 48
= 00000101+00110000
= 00110101 = 53 = ‘5’

Thus, sum_bytes[3] holds ‘5’

2.3良好编码习惯

- The code of name_to_number.go shows bad programming practices
  右面的程序体现了下列四点糟糕的编码习惯
  - 糟糕的变量名 Non-descriptive names
    - name, sum_bytes
  - 魔数 Magic numbers
    - 11, 4, 3
  - 重复代码 Repetitive code
    - of four print statements
  - 无说明 No documentation

```
package main
import "fmt"
func main() {
var name string = "Alan Turing"
    sum := 0
    for i := 0; i < 11; i++ {
       sum = sum + int(name[i])
    }
var sum_bytes [4]byte
var j int
for j = 3; sum != 0; j-- {
   sum_bytes[j] = byte(sum%10) + '0'
   sum = sum / 10
}
fmt.Printf("%c", sum_bytes[0])
fmt.Printf("%c", sum_bytes[1])
fmt.Printf("%c", sum_bytes[2])
fmt.Printf("%c", sum_bytes[3])
fmt.Printf("\n")
}
```

请每位同学提交Name2Number.go

姓名拼音请与国科大邮箱一致。例如邮箱是lizhenying21@mails.ucas.ac.cn，则姓名拼音是"Li Zhen Ying"（每个字拼音首字母大写，之间用1个空格隔开）

- 纠正了前页列出的四个缺点
  - Descriptive names
    - studentName
    - maxLength
    - sumBytes
  - No magic numbers
    - 11 → len(studentName)
    - 4 → maxLength
    - 3 → len(sumBytes) - 1
  - No repetitive code
    - for k loop
  - Documentation
    - 5 lines of comments added

```
package main                // Name2Number.go
import "fmt"
const studentName           = "Alan Turing" // 改成你自己的姓名拼音，如"Li Zhen Ying"
const maxLength             = 4 // 姓名编码最多4个十进制数字
func main() {
    sum := 0
    for i := 0; i < len(studentName); i++ { // add up studentName to sum
        sum = sum + int(studentName[i])
    }
    var sumBytes [maxLength]byte  // array to hold characters of sum
    var j int
    for j = len(sumBytes) - 1; sum != 0; j-- { // extract each digit from sum
        sumBytes[j] = byte(sum%10) + '0'
        sum = sum / 10
    }
    var k int
    for k = j + 1; k < len(sumBytes); k++ {    // print each digit of sum
        fmt.Printf("%c", sumBytes[k])
    }
    fmt.Printf("\n")
}
```

代码文件提交链接：https://www.solid.things.ac.cn:7245/basic_programming