第3章 Go顺序结构程序设计

通过前两章的知识可以了解到，Go 程序的基本分发单位是包(Package)，一个包由预处理命令(Import)、常量声明、全局变量声明和若干函数组成。一个函数又由声明部分和执行语句组成。不同类型的执行语句，可以将 Go 程序组织成：顺序结构、分支结构和循环结构。本章将介绍几种简单的 Go 语句，并使用它们设计顺序结构的 Go 程序。

3.1 顺序结构程序设计和基本语句

在进行程序设计时，一般有两部分工作要做：一部分是数据设计，另一部分是操作设计。数据设计的结果是一系列数据描述语句，主要用来定义数据类型，完成数据的初始化等(第 2 章主要就是这方面的知识内容)。操作设计的结果是一系列操作控制语句，其作用是向计算机系统发出操作指令，以完成对数据的加工和流程控制(本教材第 3～5 章主要就是介绍这方面的知识内容)。

3.1.1 顺序程序结构

计算机程序设计结构一般可分为三种类型：顺序结构、选择结构和循环结构。顺序结构(Chronological Structure)程序设计是最简单的，只要按照解决问题的顺序写出相应的语句就行，它的执行顺序是自上而下，依次执行。不过大多数情况下顺序结构都是作为程序的一部分，与其他结构一起构成一个复杂的程序，例如分支结构中的复合语句，循环结构中的循环体等。

3.1.2 简单语句

在 Go 语言中，无论是运算操作还是流程控制，都是由相应的语句完成的。组成顺序结构程序的简单语句主要包括表达式语句、输入输出语句、空语句和函数调用语句。输入输出语句将在 3.3 节专门介绍，这里仅介绍表达式语句、空语句和函数调用语句。

1. 表达式语句

由表达式组成的语句叫表达式语句。表达式语句很简单，表达式加上分号“；”就是一个表达式语句。在编辑 Go 源程序代码时，如果一行只有一条表达式语句，分号可以省略；如果一行包含多条语句，则必须使用分号将不同语句隔开。例如：

```
//一行只有一条语句分号可以省略
name := "李明"
fmt.Println("My name is ",name)
//一行有多条语句必须使用分号隔开
name1 := "李明" ;fmt.Println("My name is ",name1)
```

表达式语句可以分为赋值语句和运算符表达式语句,其作用是计算表达式或改变变量的值。

(1) 赋值语句。在表述一个算法时,经常要引入变量,并赋给该变量一个值。用来表明赋给某一个变量一个具体的确定值的语句叫做赋值语句(Assignment Statement)。在算法语句中,赋值语句是最基本的语句。

Go 语言中的赋值语句由赋值表达式后跟一个分号组成,在程序设计过程中,赋值语句应用十分广泛。在 2.3.1 节中了解到,赋值表达式由赋值运算符“=”实现,变量应在“=”的左边,值应在“=”的右边。值可以是一个常量或某种类型的数值,还可以由其他表达式产生或是某个函数的返回值。例如:

```
//将数值直接赋给变量
name := "李明"
//将一个常量赋给变量
pi := math.Pi
//将表达式结果赋给变量
sum := 100 + 200
//将函数的返回值赋给变量
strlen := len("Hello World!")
```

(2) 运算符表达式语句。在 Go 语言中,运算符表达式语句通常由运算符表达式后跟一个分号组成,例如:

```
i++;
i--
```

在该例中,“i++;”语句的功能是变量 i 的值增 1;“i--;”语句的功能是变量 i 的值减 1。这两条语句通常用于循环控制语句中,对循环控制变量进行增、减操作。

2. 函数调用语句

函数调用语句由函数调用表达式后跟一个“;”组成,其主要作用是完成特定的任务,和表达式语句一样,如果一行只有一条表达式语句,分号也可以省略。

Go 语言函数分为:内部函数、标准库函数和自定义函数,要了解函数的更多知识可先参阅第 7 章内容。内部函数是 Go 编译器自身提供的一系列函数,共有十几个(见附录 A);自定义函数是用户自己声明的函数,它只能在当前包中调用。对于内部函数和自定义函数都可以直接调用,格式如下:

```
functionName(参数列表)
```

例如：

```
//调用内置函数 len()
strlen := len("Hello World!")
//调用自定义函数 f1()
data := f1()
```

除了标准函数和自定义函数，Go 语言还提供了丰富的标准库函数，这些标准库函数都以包的形式分类组织起来(要了解 Go 语言提供的主要包和库函数可参加附录 C)。对于标准库函数，要先导入包然后才能调用，格式如下：

```
import "packageName"
packageName.functionName(参数列表)
```

例如：

```
import "fmt"
fmt.Scanf("%f",&x)
fmt.Printf("%f",x)
```

该例中标准输入、输出函数由 fmt 包提供，所以要调用这两个函数首先要导入 fmt 包。第一条语句使用 import 语句导入 fmt 包；第二条语句调用标准输入函数 Scanf()，从键盘输入 x 的值；第三条语句调用标准输出函数 Printf()，将 x 的值输出到显示器上。

3. 空语句

空语句用一个分号“；”表示，一般形式为：

```
;
```

空语句在语法上占有一条简单语句的位置，而执行该语句不调用任何操作。空语句有时用来作流程的转向点，流程从程序其他地方转到此语句处；空语句也可以用来作为循环语句中的循环体，循环体是空语句，表示循环体什么也不做，即无限循环；空语句还经常用作程序功能扩展，比如给程序预留一些功能扩展的位置，而暂时不编写程序语句代码，留作以后再扩展。所以，空语句看似简单，但也可以灵活运用。

3.1.3 复合语句

复合语句(Compound Statement)是由一对花括号“{}”将多条语句组合在一起而构成的，在语法上相当于一条语句，又称为分程序。复合语句的一般形式为：

```
{
    [内部数据描述语句]
    操作语句 1
    操作语句 2
      ⋮
```

```
    操作语句 n
}
```

使用复合语句时要注意：

(1) 在复合语句的“内部数据描述语句”中定义的变量是局部变量，仅在复合语句中有效。

(2) 复合语句结束的右大括号“}”之后，不需要再加分号“；”。

例如：

```
//复合语句
package main
import (
    "fmt"
)
func main() {
    var x,y int = 1,2
    {
        var x int = 2
        {
            var x int = 3
            fmt.Println(x,y)
        }
        fmt.Println(x,y)
    }
    fmt.Println(x,y)
}
```

测试结果为：

```
3  2
2  2
1  2
```

在该例中，main 函数只有一条语句——复合语句，在这条复合语句中又嵌套了另一条复合语句。如果给它们分层，main 函数是第一层，复合语句是第二层，嵌套的复合语句是第三层。在各层中都声明了各自的局部变量 x。所以，各层引用的 x 都是各自的局部变量 x，对应不同的内存空间。因此，在各层复合语句中给 x 赋值时，不会影响其他层 x 变量的值。复合语句常用于流程控制语句中执行多条语句。

3.2 Go 程序语法注意事项

程序语句是计算机程序中一个完整的操作单位，程序主要是通过执行语句完成各种工作。程序语句会向计算机发出操作指令，来实现程序的各种功能。

3.2.1 Go 程序语句和分号的使用

Go 语言程序语句一般分为变量声明语句、表达式语句、控制语句、函数调用语句、输入输出语句、返回语句等。

用户在编写程序时要注意，Go 程序语句末尾没有分号“；”。其实和 C 语言一样，Go 语言的正式语法也使用分号来终止语句。但不同的是，这些分号由词法分析器在扫描源代码过程中使用简单的规则自动插入，因此用户输入源代码多数时候就不需要分号了。

通常 Go 程序仅在 for 循环语句中使用分号，以此来分开计数器初始化语句、条件表达式语句和步进表达式语句。

Go 语言不允许将两条语句写到一行，如果必须要写在一行，要使用分号将这两条语句分隔开。

3.2.2 Go 程序语句块和左大括号约定

语句块就是用“{”和“}”括起来的若干条程序语句。例如，一个函数中的语句都包含在用“{”和“}”括起来的函数体中，同样还有分支结构或循环结构的语句块也被括起来。这些语句块在逻辑上都属于一个整体。

Go 语言规定，函数、控制结构(if、for、switch 或 select)的左大括号“{”，“必须和函数声明或控制结构放在同一行”。如果将左大括号“放在函数声明和控制语句的下一行”，编译器会在左大括号的前方自动插入一个分号，这可能导致异常的结果。

用“{”和“}”把相关代码括起来，有助于更清晰、更准确地定义程序逻辑边界，有助于更容易地阅读和分析程序。

3.2.3 注释语句

注释语句(Comment Statement)，是在程序的开始或中间，不具有任何功能实现的作用，仅仅是对程序进行说明的语句。Go 程序的注释方式有两种：单行注释和多行(块)注释。形如：

```
//单行注释
/*
   多行(块)注释
*/
```

3.3 数据输入输出

一个计算机程序任务通常由数据输入、数据处理、数据输出三部分组成，所以数据输入、输出语句是程序中最普通的语句，几乎每一个程序都要使用它。Go 语言提供了多种用于实现数据输入、输出的函数，这些函数在 fmt 包中实现。在输入、输出数据时，首先要导入 fmt 包，导入语句为：

```
import "fmt"
```

3.3.1 标准输出函数

数据输出(Data Output),是计算机对各类输入数据进行加工处理后,将结果以用户所要求的形式输出到标准输出设备上(比如显示器)。在 Go 标准库的 fmt 包中,有三种标准输出函数:Print()、Printf()和 Println()。

1. Print()函数

Print()函数的功能是按照系统默认格式字符串和参数表,生成一个打印字符串,然后再将其输出到标准输出设备上,输出时会在操作数之间自动加上空格。Print()函数执行后会返回输出的字节数或错误类型,Print()函数原型定义如下:

```
func Print(a ... interface{}) (n int,err error)
```

例 3-1 使用 Print()函数默认格式打印输出数据。

```
 1 //Print()函数默认格式打印
 2 package main
 3
 4 import(
 5     "fmt"
 6 )
 7 func main() {
 8     var id int = 100
 9     var name string = "李明"
10     var grade float32 = 91.5
11     fmt.Print("默认格式打印: \n")
12     fmt.Print(" 学号: ",id)
13     fmt.Print("\n")
14     fmt.Print(" 姓名: ",name)
15     fmt.Print("\n")
16     fmt.Print(" 成绩: ",grade)
17 }
```

编译并运行该程序,输出结果为:

```
默认格式打印:
学号: 100
姓名: 李明
成绩: 91.5
```

在例 3-1 中可以看到,由于 Print()函数不支持格式化输出,所以像回车、换行这些操作要用户自己来控制。比如该例中使用 fmt.Print("\n")语句,来实现一个换行操作。

2. Println()函数

Println()函数和Print()函数的功能基本一致，不同的是Println()函数会在输出结束后再自动输出一个换行，Println()函数的原型定义如下：

```
func Println(a ...interface{}) (n int,err error)
```

例3-1的输出效果如果使用Println()函数实现，则代码如例3-2所示。

例3-2 使用Println()函数默认格式打印输出数据。

```
1 //Println()函数默认格式打印
2 package main
3
4 import(
5     "fmt"
6 )
7 func main() {
8     var id int = 100
9     var name string = "李明"
10    var grade float32 = 91.5
11    fmt.Println("默认格式打印：")
12    fmt.Println(" 学号：",id)
13    fmt.Println(" 姓名：",name)
14    fmt.Println(" 成绩：",grade)
15 }
```

编译并运行该程序，输出结果为：

```
默认格式打印：
学号：100
姓名：李明
成绩：91.5
```

通过例3-2的输出结果可以看出和例3-1的输出效果一模一样，所以在遇到要求按照默认格式输出，还需输出一个空格的场合，通常使用Println()函数来完成，而且代码比使用Print()函数更简洁清晰。

3. Printf()函数

前面介绍的Print()和Println()函数都是采用默认格式输出数据，如果要按照更灵活的格式输出数据就要使用Printf()函数，Printf()函数的原型定义如下：

```
func Printf(format string,a ...interface{}) (n int,err error)
```

ftm包中的Printf()函数类似于C语言的printf()函数，会按照格式化(format)字符串将操作数格式化输出到标准输出设备上，Printf()函数也会返回输出的字节数和错误类型。

Go 语言提供的格式化字符串类型非常丰富，其含义和作用如表 3-1 所示。

表 3-1 format 字符串的含义和作用

作用对象	格式字符	说 明
所有类型	%v	以数据对象的基本格式输出
	%#v	输出数值的同时输出 Go 语法表示
	%T	输出数据类型，比如 int、float 等
	%%	输出“%”
布尔型	%t	输出布尔值“true”或“false”
整型	%b	以二进制格式输出
	%c	以 Unicode 字符格式输出
	%d	以十进制格式输出
	%o	以八进制格式输出，0～7
	%q	输出的每个字符自动加单引号
	%x	以十六进制格式输出，使用小写字母 a～f
	%X	以十六进制格式输出，使用大写字母 A～F
	%U	Unicode 格式：U+1234，等价于“U+%04X”
浮点型、复数	%b	无小数部分、两位指数的科学记数法，和 strconv. FormatFloat 的‘b’转换格式一致
	%e	科学记数法，如：－1234.456e+78
	%E	科学记数法，如：－1234.456E+78
	%f	有小数部分，但无指数部分，如：123.456
	%g	根据实际情况采用%e 或%f 格式(以获得更简洁的输出)
	%G	根据实际情况采用%E 或%f 格式(以获得更简洁的输出)
字符串、切片	%s	直接输出字符串或切片
	%q	输出字符串的同时自动加双引号
	%x	每个字节用两字符十六进制数表示(使用小写 a～f)
	%X	每个字节用两字符十六进制数表示(使用大写 A～F)
指针	%p	使用以 0x 开头的十六进制数表示
其他格式符	+	输出数值的正负号，对%q(%+q)按 ASCII 码输出
	－	使用空格填充右侧空缺，而不是默认的左侧
	#	切换格式：在八进制前加 0(%#o)，在十六进制前加 0x(%#x)或 0X(%#X)，去掉指针的 0x(%#p)。对于%q(%#q)输出无修饰符的字符串，对于%U(%#U)输出可打印的 Unicode 字符
	' '	对于数字(% d)会在数字前留一个空格，对于切片或字符串(% x)或(% X)会以十六进制输出
	0	用前置 0 代替空格填补空缺

(1) 通用格式化操作符。通过表 3-1 可以看出，%v、%#v、%T、%%这几种操作符可以作用于任何数据类型。尤其是使用操作符“%v”时，用户不需考虑数据的具体类型就可以正确输出，这种功能对用户输出数据来说非常方便。使用操作符“%#v”能输出 Go 语法表示，可以方便程序检错。使用操作符“%T”能输出数据类型，可以方便程序分析。使用操作符“%%”可以输出百分号“%”，可以用于一些统计结果输出。

例 3-3 通用格式化输出。

```
//通用格式化输出
package main

import(
    "fmt"
)
func main() {
    var id int = 100
    var name string = "李明"
    var grade float32 = 91.5
    fmt.Println("通用格式化输出: ")
    fmt.Printf("%v %v %v\n",id,name,grade)
    fmt.Printf("%#v %#v %#v\n",id,name,grade)
    fmt.Printf("%T %T %T\n",id,name,grade)
    fmt.Printf("60%% %%\n")
}
```

编译并运行该程序,输出结果为:

```
通用格式化输出:
100 李明 91.5
100 "李明" 91.5
int string float32
60%
```

(2) 布尔型数据格式输出。通过 2.2.1 节的知识内容知道,在 Go 语言中布尔型数据的取值是“true”和“false”,这和其他高级程序设计语言不同,比如 C 语言中可以使用“0”代表布尔型的“false”,使用“1”代表布尔型的“true”。在使用 Printf()函数时,格式化操作符“t”用于输出布尔型数据。

例 3-4 布尔型数据输出。

```
//布尔型数据输出
package main

import(
    "fmt"
)
func main() {
    var yes,no = true,false
    fmt.Println("布尔型数据输出: ")
    fmt.Printf("%t %t\n",yes,no)
}
```

编译并运行该程序,输出结果为:

```
布尔型数据输出:
true false
```

(3) 整型数据格式输出。通过 2.2.2 节的知识内容可以了解到,Go 语言中的整型数据类型非常丰富,比如 int 型、int8 型、int32 型等。另外,整型数据在计算机中的表示形式也多种多样,比如二进制、十进制、八进制、十六进制等。最后,由于计算机字符编码不同,比如 ASCII 编码、Unicode 编码、UTF-8 编码等,整型数据也可以按不同编码字符输出。所以,Go 语言中整型数据的格式化输出方式多种多样,非常灵活,如表 3-1 所示。

例 3-5 整型数据格式化输出。

```
//整型数据格式化输出
package main

import(
    "fmt"
)
func main() {
    var year int = 20013
    fmt.Println("整型数据格式化输出: ")
    fmt.Printf("二进制格式: %b\n",year)
    fmt.Printf("十进制格式: %d\n",year)
    fmt.Printf("八进制格式: %o\n",year)
    fmt.Printf("十六进制格式(a~f): %x\n",year)
    fmt.Printf("十六进制格式(A~F): %X\n",year)
    fmt.Printf("数值对应的 Unicode 编码: %c\n",year)
    fmt.Printf("Unicode 格式: %U\n",year)
    fmt.Printf("Unicode 编码使用单引号括起来: %q\n",year)
}
```

编译并运行该程序,输出结果为:

```
整型数据格式化输出:
二进制格式: 100111000101101
十进制格式: 20013
八进制格式: 47055
十六进制格式(a~f): 4e2d
十六进制格式(A~F): 4E2D
数值对应的 Unicode 编码: 中
Unicode 格式: U+4E2D
Unicode 编码使用单引号括起来: '中'
```

(4) 浮点型和复数格式输出。通过 2.2.3 节的知识内容可以了解到,用户通常在进行浮点型数据运算时采用十进小数制形式,而计算机内部则采用指数形式来存储浮点型数据。所以在进行浮点型数据输出时,可以采用十进制小数形式,也可以采用指数形式(科学记数法)。

对于复数,它是由“实部”和“虚部”两部分组成的,“实部”、“虚部”其实质就是两个浮点型数据,可以使用 real()和 imag()函数获取,然后再采用合适的浮点型数输出格式就可以了。

例 3-6 浮点型数据和复数的格式化输出。

```
//浮点型数据和复数的格式化输出
package main

import(
    "fmt"
)
func main() {
    var f float32 = 123.4567
    var cp = complex(1.2,3.4)
    fmt.Println("浮点型数、复数格式化输出：")
    fmt.Printf("无小数两位指数科学记数法：%b\n",f)
    fmt.Printf("科学记数法(小写)：%e\n",f)
    fmt.Printf("科学记数法(大写)：%E\n",f)
    fmt.Printf("根据实际情况采用%%e或%%f(小写)：%g\n",f)
    fmt.Printf("根据实际情况采用%%e或%%f(大写)：%G\n",f)
    fmt.Printf("只有小数部分无指数部分：%f\n",f)
    fmt.Printf("保留 2 位小数：%6.2f\n",f)
    fmt.Printf("复数%v的实部 = %g 虚部 = %g\n",cp,real(cp),imag(cp))
}
```

编译并运行该程序，输出结果为：

```
浮点型数、复数格式化输出：
无小数两位指数科学记数法：16181717p-17
科学记数法(小写)：1.234567e+02
科学记数法(大写)：1.234567E+02
根据实际情况采用%e或%f(小写)：123.4567
根据实际情况采用%e或%f(大写)：123.4567
只有小数部分无指数部分：123.456703
保留 2 位小数：123.46
复数(1.2+3.4i)的实部 = 1.2 虚部 = 3.4
```

(5) 字符串和切片格式输出。

例 3-7 字符串和切片的格式化输出。

```
//字符串和切片的格式化输出
package main

import(
    "fmt"
)
func main() {
    var str string = "Go language"
    fmt.Println("字符串和切片格式化输出：")
    fmt.Printf("直接输出字符串或切片%s\n",str)
    fmt.Printf("自动加双引号：%q\n",str)
    fmt.Printf("每个字节用两个字符十六进制表示(a～f)：%x\n",str)
```

```
13      fmt.Printf("每个字节用两个字符十六进制表示(A～F): %X\n",str)
14 }
```

编译并运行该程序,输出结果为:

```
字符串和切片格式化输出:
直接输出字符串或切片:Go language
自动加双引号:"Go language"
每个字节用两个字符十六进制表示(a～f) :476f206c616e6775616765
每个字节用两个字符十六进制表示(A～F) :476F206C616E6775616765
```

(6) 指针类型格式输出。Go 语言也支持指针类型,通过 2.2.2 节的知识内容可以了解到,uintptr 可以用来保存 32 位或 64 位的指针。同样在调用 Printf()函数时,可以使用格式化操作符"p"输出指针地址。

例 3-8 格式化输出指针地址。

```
 1 //格式化输出指针地址
 2 package main
 3
 4 import(
 5     "fmt"
 6 )
 7 func main() {
 8     var i int = 100
 9     var i_pointer * int
10     i_pointer = &i
11     fmt.Println("格式化输出指针地址: ")
12     fmt.Printf("输出以 0x 开头的指针地址: %p\n",i_pointer)
13 }
```

编译并运行该程序,输出结果为:

```
格式化输出指针地址:
输出以 0x 开头的指针地址: 0x10df00e8
```

(7) 其他类型格式输出。除了前面介绍的一些格式化操作符,Go 语言还提供了一些其他格式化操作符,例如"+"、"-"、"#"、"' '"、"0"等,它们的作用可以参见表 3-1 中的说明。

例 3-9 其他类型格式输出。

```
 1 //其他类型格式输出
 2 package main
 3
 4 import(
 5     "fmt"
 6 )
 7
```

```
 8 func main() {
 9     var a int = 97
10     var f float32 = -1.32
11     var p *int
12     var str string = "Golang"
13     p = &a
14     fmt.Println("其他类型格式输出: ")
15     fmt.Printf("输出数值的正负号: %+d, %+g\n",a,f)
16     fmt.Printf("ASCII 码格式输出: %+q\n",a)
17     fmt.Printf("切换格式(#o,#x,#X): %d %#o %#x %#X\n",a,a,a,a)
18     fmt.Printf("消除指针地址前的 0X: %p %#p\n",p,p)
19     fmt.Printf("以十六进制输出字符串: %s,% x\n",str,str)
20 }
```

编译并运行该程序,输出结果为:

```
其他类型格式输出:
输出数值的正负号: +97, -1.32
ASCII 码格式输出: 'a'
切换格式(#o,#x,#X): 97 0141 0x61 0X61
消除指针地址前的 0X: 0x10df00e8 10df00e8
以十六进制输出字符串: Golang,47 6f 6c 61 6e 67
```

3.3.2 标准输入函数

数据输入(Data Input)是当程序在运行过程中,将系统外部原始数据通过标准输入设备传输给系统内部,并将这些数据以外部格式转换为系统便于处理的内部格式的过程,其方式与使用的输入设备密切相关(比如键盘)。在 Go 标准库 fmt 包中,有三种标准输入函数 Scan()、Scanf()和 Scanln()。

1. Scan()函数

Scan()函数的功能是从标准输入设备读取数据,并将使用空格分割的连续数据顺序存入到参数里,换行也将被视为空格。Scan()函数调用成功,返回正确读取的参数的数量 n;如果少于要求提供的参数数量,函数返回 err 并报告错误原因。Scan()函数的原型定义如下:

```
func Scan(a ...interface{}) (n int,err error)
```

例 3-10 使用 Scan()函数从标准输入设备录入数据。

```
1 //使用 Scan()函数从标准输入设备录入数据
2 package main
3
4 import(
5     "fmt"
```

```
 6 )
 7 func main() {
 8     var a int
 9     var f float32
10     var str string
11     fmt.Println("准备录入数据: ")
12     fmt.Scan(&a,&f,&str)
13     fmt.Println("输出录入结果: ")
14     fmt.Println(a,f,str)
15 }
```

编译并运行该程序,测试过程如下:

```
准备录入数据:
100 3.14 Golang ↙
输出录入结果:
100 3.14 Golang
```

2. Scanln()函数

Scanln()函数的功能是从标准输入设备读取数据,并将使用空格分割的连续数据顺序存入到参数里。Scanln()函数调用成功,返回正确读取的参数的数量 n;如果少于要求提供的参数数量,函数返回 err 并报告错误原因。

从上面的说明可以看出,Scanln()函数的功能类似于 Scan()函数,但 Scanln()函数会在读取到换行符时终止录入数据,并且在存入最后一条数据时必须有换行或结束符。Scanln()函数的原型定义如下:

```
func Scanln(a ...interface{}) (n int,err error)
```

例 3-11 使用 Scanln()函数从标准输入设备录入数据。

```
 1 //使用 Scanln()函数从标准输入设备录入数据
 2 package main
 3
 4 import(
 5     "fmt"
 6 )
 7 func main() {
 8     var a int
 9     var f float32
10     var str string
11     fmt.Println("准备录入数据: ")
12     fmt.Scanln(&a,&f,&str)
13     fmt.Println("输出录入结果: ")
14     fmt.Println(a,f,str)
15 }
```

对该程序分为以下三种方式进行测试：

(1) 输入一条数据，然后换行回车。

```
准备录入数据:
100 ↙
输出录入结果:
100 0
```

(2) 输入两条数据，然后换行回车。

```
准备录入数据:
100 3.14 ↙
输出录入结果:
100 3.14
```

(3) 输入三条数据，然后换行回车。

```
准备录入数据:
100 3,14 Golang ↙
输出录入结果:
100 3,14 Golang
```

3. Scanf()函数

Scanf()函数的功能是按照格式化字符从标准输入设备读取数据，并将所读取的数据顺序存入到参数里。Scanf()函数调用成功，返回正确读取的参数的数量 n；如果少于要求提供的参数数量，函数返回 err 并报告错误原因。Scanf()函数的原型定义如下：

```
func Scanf(format string,a ...interface{}) (n int,err error)
```

例 3-12 使用 Scanf()函数格式化录入数据。

```
 1 //使用 Scanf()函数格式化录入数据
 2 package main
 3
 4 import(
 5     "fmt"
 6 )
 7 func main() {
 8     var a int
 9     var f float32
10     var str string
11     fmt.Println("准备录入数据: ")
12     fmt.Scanf("%d,%f,%s",&a,&f,&str)
13     fmt.Println("输出录入结果: ")
14     fmt.Println(a,f,str)
15 }
```

编译并运行该程序，测试过程如下：

```
准备录入数据:
100,3.14,Golang ↙
输出录入结果:
100 3.14 Golang
```

在调用 Scanf()函数时还需注意的是，上面的 Scanf("%d,%f,%str",&a,&f,&str)语句中的“%d,%f,%str”，表示不同参数之间要使用逗号“,”隔开。在从键盘输入数据时，数据间也要使用逗号“,”隔开。输入参数之间除了使用逗号分隔，还可以使用空格、Tab 键或回车键分隔。

3.4 Strings 包

Go 标准库中的 Strings 包，提供了对字符串进行常用操作的基本函数，比如像字符串查找、判断字符串是否相等、字符串匹配等。

3.4.1 字符串查找函数

如果要判断一个字符串中是否包含某个子串，可以使用 Strings 包中的 Contains()、ContainsAny()、ContainsRune()和 Count()函数。

1. Contains()函数

Contains()函数用于查找子串是否在指定的字符串中，如果在返回 true，否则返回 false。Contains()函数的原型定义如下：

```
func Contains(s,substr string) bool
```

在函数 Contains()中，参数 s 表示需要查找的主串；参数 substr 表示子串。

2. ContainsAny()函数

ContainsAny()函数用于查找字符串中是否包含子串中某个 Unicode 编码格式的字符。如果包含返回 true，否则返回 false。ContainsAny()函数的原型定义如下：

```
func ContainsAny(s,chars string) bool
```

在函数 ContainsAny()中，参数 s 表示需要查找的主串；参数 chars 表示保存的 Unicode 字符集。

3. ContainsRune()函数

ContainsRune()函数用于查找字符串中是否包某个 rune 类型的字符。如果包含返回 true，否则返回 false。ContainsRune()函数的原型定义如下：

```
func ContainsRune(s string,r rune) bool
```

在函数 ContainsRune()中,参数 s 表示需要查找的主串;参数 r 表示 rune 字符。

4. Count()函数

Count()函数用于统计子串在指定字符串中出现的次数,该函数调用成功后返回统计结果,函数原型定义如下:

```
func Count(s,sep string) int
```

在函数 ContainsRune()中,参数 s 表示需要查找的主串;参数 sep 表示需要统计的子串。

例 3-13 字符串查找操作。

```
//字符串查找操作
package main

import(
    "fmt"
    "strings"
)
func main() {
    var str string = "Hello,World!"
    var s1,s2,s3,s4 string = "llo","go","ll","l"
    var ch1,ch2 rune = 'c','d'
    fmt.Println("查找子串是否在指定的字符串中:")
    fmt.Printf("%q在%q中? %t\n",s1,str,strings.Contains(str,s1))
    fmt.Printf("%q在%q中? %t\n",s2,str,strings.Contains(str,s2))
    fmt.Println("查找字符串中是否含有子串中的字符:")
fmt.Printf("%q中含有%q的字符? %t\n",str,s1,strings.ContainsAny(str,s1))
fmt.Printf("%q中含有%q的字符? %t\n",str,s2,strings.ContainsAny(str,s2))
    fmt.Println("查找字符串中是否含有某个字符:")
fmt.Printf("%q中含有字符%q? %t\n",str,ch1,strings.ContainsRune(str,ch1))
fmt.Printf("%q中含有字符%q? %t\n",str,ch2,strings.ContainsRune(str,ch2))
    fmt.Println("统计指定字符串包含子串的个数: ")
    fmt.Printf("%q中含有%d个子串%q.\n",str,strings.Count(str,s3),s3)
    fmt.Printf("%q中含有%d个子串%q.\n",str,strings.Count(str,s4),s4)
}
```

编译并运行该程序,输出结果为:

```
查找子串是否在指定的字符串中:
    "llo"在"Hello,World!"中? true
    "go"在"Hello,World! "中? false
查找字符串中是否含有子串中的字符:
    "Hello,World!"中含有"llo"的字符? True
    "Hello,World!"中含有"go"的字符? True
查找字符串中是否含有某个字符:
```

```
    "Hello,World!"中含有字符'c'? false
    "Hello,World!"中含有字符'd'? true
统计指定字符串包含子串的个数:
    "Hello,World!"中含有 1 个子串"ll".
    "Hello,World!"中含有 3 个子串"l".
```

3.4.2 字符串比较函数

如果想要判断两个字符串是否相等,可以使用 EqualFold()函数,如果相等该函数返回 true,否则返回 false,该函数原型定义如下:

```
func EqualFold(s,t string) bool
```

在函数 EqualFold()中,参数 s 表示需要查找的主串;参数 t 表示需要比较的普串。

例 3-14 判断两个字符串是否相等。

```
 1 //判断两个字符串是否相等
 2 package main
 3
 4 import(
 5     "fmt"
 6     "strings"
 7 )
 8 func main() {
 9     var str1,str2,str3 string = "Go","go","lang"
10     fmt.Println("判断两个字符串是否相等: ")
11     fmt.Printf("%q 等于 %q? %t\n",str1,str2,strings.EqualFold(str1,str2))
12     fmt.Printf("%q 等于 %q? %t\n",str1,str3,strings.EqualFold(str1,str3))
13 }
```

编译并运行该程序,输出结果为:

```
判断两个字符串是否相等:
"Go"等于"go"? true
"Go"等于"lang"? false
```

通过分析例 3-14 的运行结果可以发现,EqualFold()函数在对字符串进行比较时会忽略大小写。

3.4.3 字符串位置索引函数

如果要判断一个子串在主串中出现的位置索引,可以使用 Strings 包中的 Index()、IndexAny()、IndexFunc()、IndexRune()、LastIndex()、LastIndexAny()和 LastIndexFunc()函数。

1. Index()函数

Index()函数用于判断子串在主串中第一次出现的位置，如果存在，返回 int，对应 sep 出现在 s 中的索引位置；如果不存在，则返回－1。该函数原型定义如下：

```
func Index(s,sep string) int
```

在函数 Index()中，参数 s 表示需要查找的主串；参数 sep 表示需要判断的第一次出现位置的字符串。

2. IndexAny()函数

IndexAny()函数用于判断 chars 集中任意一个 Unicode 字符在主串中第一次出现的位置，如果存在，返回 int，对应相关字符出现在 s 中的索引位置；如果不存在，则返回－1。该函数原型定义如下：

```
func IndexAny(s,chars string) int
```

在函数 IndexAny()中，参数 s 表示需要查找的主串；参数 chars 表示保存的 Unicode 字符集。

3. IndexFunc()函数

IndexFunc()函数用于判断字符串 s 中的每一个传入函数 f()的字符，返回符合函数 f()的第一个字符的位置。如果符合，返回该字符在 s 中的位置；如果都不符合，则返回－1。该函数原型定义如下：

```
func IndexFunc(s string,f func(rune) bool) int
```

在函数 IndexFunc()中，参数 s 表示需要判断的主串；参数 f 表示一个函数，该函数参数是 rune 类型，返回值是 bool，如果 rune 符合 f 函数的逻辑那么返回 true，否则返回 false。

4. IndexRune()函数

IndexRune()函数用于判断 rune 类型的字符 r 在字符串 s 中第一次出现的位置，如果存在，返回 int，对应 r 出现在 s 中的索引位置；如果不存在，则返回－1。该函数原型定义如下：

```
func IndexRune(s string,r rune) int
```

在函数 IndexRune()中，参数 s 表示需要查找的主串；参数 r 表示 rune 类型的字符。

5. LastIndex()函数

LastIndex()函数用于判断子串在主串中最后一次出现的位置，如果存在，返回 int，对应 sep 出现在 s 中的索引位置；如果不存在，则返回－1。该函数原型定义如下：

```
func LastIndex(s,sep string) int
```

在函数 LastIndex()中,参数 s 表示需要查找的主串;参数 sep 表示需要判断的最后一次出现位置的字符串。

6. LastIndexAny()函数

LastIndexAny()函数用于判断 chars 集中任意一个 Unicode 字符在主串中最后一次出现的位置,如果存在,返回 int,对应相关字符出现在 s 中的索引位置;如果不存在,则返回-1。该函数原型定义如下:

```
func LastIndexAny(s,chars string) int
```

在函数 LastIndexAny()中,参数 s 表示需要查找的主串;参数 chars 表示需要判断的最后一次出现位置的 Unicode 字符集。

7. LastIndexFunc()函数

LastIndexFunc()函数用于判断传入函数 f()的字符串中,是否存在符合 f()限定条件的字符。如果 s 中存在此种字符,则返回最后一个字符在 s 中的位置;如果 s 中不存在此种字符,则返回-1。该函数原型定义如下:

```
func LastIndexFunc(s string,f func(rune) bool) int
```

在函数 LastIndexFunc()中,参数 s 表示需要判断的主串;参数 f 表示一个函数,该函数参数是 rune 类型,返回值是 bool,如果 rune 符合 f 函数的逻辑那么返回 true,否者返回 false。

例 3-15 判断子串在主串中出现的位置。

```
//判断子串在主串中出现的位置
package main

import(
    "fmt"
    "strings"
)
func main() {
    var s,sep,chars string = "Golang","an","lang"
    var r rune = 'a'
    fmt.Printf("%q第一次在%q中出现的索引是:%d\n",sep,s,strings.Index(s,sep))
    fmt.Printf("%q中的字符第一次在%q中出现的索引是:%d\n",chars,s,strings.
IndexAny(s,chars))
    fmt.Printf("%q第一个符合 f()的字符索引是:%d\n",s,strings.IndexFunc(s,f))
    fmt.Printf("%q第一次在%q中出现的索引是:%d\n",r,s,strings.IndexRune(s,r))
    fmt.Printf("%q最后一次在%q中出现的索引是:%d\n",sep,s,strings.LastIndex(s,sep))
```

```
16      fmt.Printf("%q中的字符最后一次在%q中出现的索引是: %d\n",chars,s,strings.
LastIndexAny(s,chars))
17      fmt.Printf("%q最后一个符合f()的字符索引是: %d\n",s,strings.LastIndexFunc(s,f))
18 }
19 func f(a rune) bool {
20      if a > 'k' {
21          return true
22      } else {
23          return false
24       }
25 }
```

编译并运行该程序,输出结果为:

```
"an"第一次在"Golang"中出现的索引是: 3
"lang"中的字符第一次在"Golang"中出现的索引是: 2
"Golang"第一个符合f()的字符索引是: 1
'a'第一次在"Golang"中出现的索引是: 3
"an"最后一次在"Golang"中出现的索引是: 3
"lang"中的字符最后一次在"Golang"中出现的索引是: 5
"Golang"最后一个符合f()的字符索引是: 4
```

3.4.4 字符串追加和替换函数

如果要向一个字符串后面追加内容或替换字符串的内容,可以使用 Strings 包中的 Repeat()函数或 Replace()函数。

1. Repeat()函数

Repeat()函数用于向字符串中追加内容,该函数返回一个新的字符串,这个新字符串由 s 重复 count 次构成。该函数原型定义如下:

```
func Repeat(s string,count int) string
```

在函数 Repeat()中,参数 s 表示需要重复的字符串; 参数 count 表示需要重复的次数。

2. Replace()函数

Replace()函数用于把主串 s 中的 old 子串替换为 new 子串,替换次数为 n 次。函数调用成功后返回处理后的新串,该函数原型定义如下:

```
func Replace(s,old,new string,n int) string
```

在函数 Replace()中,参数 s 表示需要替换的主串; 参数 old 表示需要替换的子串; 参数 new 表示用于替换的新子串; 参数 n 表示需要替换的次数,如果 n<0,则全部替换。

例 3-16 字符串追加和替换。

```
//字符串追加和替换
package main

import(
    "fmt"
    "strings"
)
func main() {
    var s = "na"
    var count int = 2
    //字符串追加
    fmt.Println("ba" + strings.Repeat(s,count))
    //字符串替换
    fmt.Println(strings.Replace("google","o","oo",1))
    fmt.Println(strings.Replace("google","o","oo", -1))
}
```

编译并运行该程序,输出结果为:

```
banana
goooogle
gooooogle
```

3.5 Strconv 包

Go 标准库中的 Strconv 包,提供了字符串与基本数据类型相互转换的一些基本函数。Format 系列函数用于将基本数据类型转换为字符串; Parse 系列函数用于将字符串转换为基本数据类型; Atoi()和 Itoa()则是相关函数的简便封装版本。

3.5.1 数值转换为字符串函数

如果要将基本数据类型转换为字符串格式,可以使用 Strconv 包中的 FormatBool()、FormatFloat()、FormatInt()和 FormatUint()函数。

1. FormatBool()函数

FormatBool()函数用于将布尔型数据转换为字符串形式,该函数原型定义如下:

```
func FormatBool(b bool) string
```

函数 FormatBool()的参数 b 表示需要被转换的布尔数,可以是“true”或“false”。

2. FormatFloat()函数

FormatFloat()函数用于将浮点型数转换为字符串形式,该函数能按照指定格式将浮点数转换成字符串,函数原型定义如下:

```
func FormatFloat(f float64,fmt byte,prec,bitSize int) string
```

在函数 FormatFloat()中,参数 f 表示需要转换的浮点数;参数 fmt 表示浮点数的格式,可以是 b、e、E、f、g 或 G,具体含义可以参见表 3-1。参数 prec 表示 e、E、f、g、G 格式浮点数的输出精度,如果为 e、E、f 格式,prec 表示小数点后的有效位数。如果为 g、G 格式,prec 表示全部有效位数。如果 prec 为－1,则以最少有效位数表示该数;参数 bitSize 表示浮点型数的位数,比如 32 或 64,分别对应 float32 和 float64。

3. FormatInt()函数

FormatInt()函数用于将整型数转换为字符串形式,该函数能按照指定格式将整型数转换成字符串,函数原型定义如下:

```
func FormatInt(i int64,base int) string
```

在函数 FormatFloat()中,参数 i 表示需要转换的整数;参数 base 表示整型数的进制,比如 2、8、10、16,分别对应二进制、八进制、十进制和十六进制。

4. FormatUint()函数

FormatUint()函数和 FormatInt()函数的功能基本相似,只不过是将无符号整数转换为字符串形式。该函数原型定义如下:

```
func FormatUint(i uint64,base int) string
```

在函数 FormatUint()中,参数 i 表示需要转换的无符号整数;参数 base 表示整型数的进制,比如 2、8、10、16,分别对应二进制、八进制、十进制和十六进制。

例 3-17 将数值转换成字符串。

```
1 //将数值转换成字符串
2 package main
3
4 import(
5     "fmt"
6     "strconv"
7 )
8 func main() {
9     var b bool = false
10    var f float64 = 3.14
11    var i int64 = -1024
12    var ui uint64 = 1024
13    fmt.Printf("将布尔数%t转换成字符串: %q\n",b,strconv.FormatBool(b))
14    fmt.Printf("将浮点数%f转换成字符串: %q\n",f,strconv.FormatFloat(f,'f',2,32))
15    fmt.Printf("将整数%d转换成字符串: %q\n",i,strconv.FormatInt(i,10))
16    fmt.Printf("将无符号整数%d转换成字符串: %q\n",ui,strconv.FormatUint(ui,10))
17 }
```

编译并运行该程序，输出结果为：

```
将布尔数 false 转换成字符串: "false"
将浮点数 3.140000 转换成字符串: "3.14"
将整数 -1024 转换成字符串: "-1024"
将无符号整数 1024 转换成字符串: "1024"
```

3.5.2 字符串转换为数值函数

如果要将字符串转换为基本数据类型，可以使用 Strconv 包中的 ParseBool()、ParseFloat()、ParseInt()和 ParseUint()函数。

1. ParseBool()函数

ParseBool()函数用于将字符串转换成布尔型数据，该函数调用成功，返回对应的布尔数 value；否则，返回一个错误类型 err。该函数的原型定义如下：

```
func ParseBool(str string) (value bool,err error)
```

在函数 ParseBool()中，参数 str 表示需要转换的布尔数的字符串形式。

2. ParseFloat()函数

ParseFloat()函数用于将字符串转换成指定格式的浮点数，该函数调用成功，返回对应的浮点数 f；否则，返回一个错误类型 err。该函数的原型定义如下：

```
func ParseFloat(s string,bitSize int) (f float64,err error)
```

在函数 ParseFloat()中，参数 s 表示需要转换的浮点数的字符串形式；参数 bitSize 表示浮点型数的位数，比如 32 或 64，分别对应 float32 和 float64。

3. ParseInt()函数

ParseInt()函数用于将字符串转换为参数指定的整型数，该函数调用成功，返回对应的整型数 i；否则，返回一个错误类型 err。该函数原型定义如下：

```
func ParseInt(s string,base int,bitSize int) (i int64,err error)
```

在函数 ParseInt()的参数中，s 表示需要转换的整型数的字符串形式；参数 base 表示整型数的进制，比如 2、8、10、16，分别对应二进制、八进制、十进制和十六进制；参数 bitSize 表示返回结果的位数，比如 0、8、16、32 和 64，分别对应 int、int8、int16、int32 和 int64。

4. ParseUint()函数

函数 ParseUint()和函数 ParseInt()的功能基本类似，只不过它是将字符串转换为参数指定的无符号整数，该函数调用成功，返回对应的无符号整数 n；否则，返回一个错误类型

err。该函数原型定义如下：

```
func ParseUint(s string,b int,bitSize int) (n uint64,err error)
```

在函数 ParseUint()中，参数 s 表示需要转换的无符号整数的字符串形式；参数 base 表示整型数的进制，比如 2、8、10、16，分别对应二进制、八进制、十进制和十六进制；参数 bitSize 表示返回结果的位数，比如 0、8、16、32 和 64，分别对应 uint、uint8、uint16、uint32 和 uint64。

例 3-18 将字符串转换成数值。

```
//将字符串转换成数值
package main

import(
    "fmt"
    "strconv"
)
func main() {
    var strb,strf,stri,strui string = "false","3.14","-1024","1024"
    var b bool
    var f float64
    var i int64
    var ui uint64
    b,_ = strconv.ParseBool(strb)
    f,_ = strconv.ParseFloat(strf,32)
    i,_ = strconv.ParseInt(stri,10,32)
    ui,_ = strconv.ParseUint(strui,10,32)
    fmt.Printf("将字符串 %q 转换成布尔数: %t\n",strb,b)
    fmt.Printf("将字符串 %q 转换成浮点数: %f\n",strf,f)
    fmt.Printf("将字符串 %q 转换成整数: %d\n",stri,i)
    fmt.Printf("将字符串 %q 转换成无符号整数: %d\n",strui,ui)
}
```

编译并运行该程序，输出结果为：

```
将字符串"false"转换成布尔数: false
将字符串"3.14"转换成浮点数: 3.140000
将字符串"-1024"转换成整数: -1024
将字符串"1024"转换成无符号整数: 1024
```

3.5.3 Atoi()和 Itoa()函数

除了 FormatInt()和 ParseInt()函数，Strconv 包中还提供了更简单的 Atoi()和 Itoa()函数，用于整型数和字符串之间的相互转换。

1. Atoi()函数

函数 Atoi()的作用和函数 ParseInt()一样，是将字符串转换为整数，只不过更简单。该函数调用成功，返回对应的整型数 i；否则，返回一个错误类型 err。该函数原型定义如下：

```
func Atoi(s string) (i int,err error)
```

在函数 Atoi()中,参数 s 表示需要转换的整数的字符串形式。

2. Itoa()函数

函数 Itoa()和函数 FormatInt()的作用一样,是将整数转化成字符串形式。该函数调用成功,返回整数的字符串形式。

```
func Itoa(i int) string
```

在函数 Itoa()中,参数 i 是需要转换的整数。

例 3-19 整数和字符串的相互转换。

```
//整数和字符串的相互转换
package main

import(
    "fmt"
    "strconv"
)
func main() {
    var i int = -1024
    var s string = "1024"
    value,_ := strconv.Atoi(s)
    str := strconv.Itoa(i)
    fmt.Printf("将字符串 %q 转换成整数: %d\n",s,value)
    fmt.Printf("将整数 %d 转换成字符串: %q\n",i,str)
}
```

编译并运行该程序,输出结果为:

```
将字符串"1024"转换成整数: 1024
将整数 -1024 转换成字符串: "-1024"
```

3.6 顺序结构程序举例

通过第 2 章的学习掌握了程序中常量与变量的声明方法,通过本章的学习掌握了数据的输入、处理和输出等方法,这样就可以利用这些知识设计简单的顺序结构程序了。下面介绍几个顺序程序设计的例子,对前面所学知识加以验证和巩固。

3.6.1 求平均值

例 3-20 从键盘输入三个整型数,然后计算它们的平均值,最后输出计算结果,小数点后保留两位有效位。

从这个程序的任务需求可以看出，完成这个计算任务共需 3 步：

① 数据输入；② 计算平均值；③ 输出计算结果。

由于 Go 语言是强类型语言，所以在设计本例算法时还需注意数据类型一致问题，题目中要求输入的三个数据为整型，而它们的平均数应该为浮点型，所以在计算平均数时要进行类型转换，即将整型转换为浮点型。程序代码如下：

```
//计算平均数
package main

import(
    "fmt"
)

func main() {
    var sum1,sum2,sum3 int
    var average float32
    fmt.Println("请输入 3 个整数：")
    fmt.Scanf("%d,%d,%d",&sum1,&sum2,&sum3)
    fmt.Println("计算平均数...")
    average = float32(sum1 + sum2 + sum3) /3.0
    fmt.Printf("输出计算结果：%6.2f\n",average)
}
```

编译并运行该程序，测试过程如下：

```
请输入 3 个整数：
13,22,-7↙
计算平均数...
输出计算结果：9.33
```

3.6.2 计算三角形面积周长

例 3-21 从键盘输入三角形三个边长，求三角形面积和周长，输出计算结果，小数点后保留两位有效位。

假设三角形的三边分别为 a、b、c，周长为 l，面积为 area，则

$$l=a+b+c$$

$$area=\sqrt{s(s-a)(s-b)(s-c)}\text{（其中 } s=l/2\text{）}$$

所以，解决该问题至少需要以下几步：

(1) 输入三角形三边 a、b、c。

(2) 计算三角形周长 l。

(3) 计算半周长 hl。

(4) 计算三角形面积 area。

(5) 输出计算结果。

该程序需要使用 math 包中的 Sqrt()函数计算平方根，所以在使用前要导入 math 包。

程序代码如下：

```
1 //计算三角形面积和周长
2 package main
3
4 import(
5     "fmt"
6     "math"
7 )
8
9 func main() {
10     var a,b,c,l,hl,area float64
11     fmt.Println("请输入三角形三边：")
12     fmt.Scanf("%f,%f,%f",&a,&b,&c)
13     l = a + b + c
14     fmt.Printf("三角形周长 = %6.2f\n",l)
15     hl = l * 0.5
16     area = math.Sqrt(hl * (hl - a) * (hl - b) * (hl - c))
17     fmt.Printf("三角形面积 = %6.2f\n",area)
18 }
```

编译并运行该程序，测试过程如下：

```
请输入三角形三边：
5.1,4.5,3.2 ↙
三角形周长 = 12.80
三角形面积 =  7.11
```

3.6.3 求解一元二次方程

例 3-22 求解一元二次方程 $ax^2+bx+c=0(b^2-4ac>0)$的实数根，输出计算结果，小数点后保留两位有效位。

由一元二次方程的求根公式：

$$x=(-b\pm\sqrt{b^2-4ac})/2a$$

设两个中间量 p、q，$p=-b/2a$，$q=\sqrt{b^2-4ac}/2a$，

可得 $x_1=p+q$，$x_2=p-q$

所以，解决该问题需要以下几个步骤：

(1) 输入一元二次方程的三个系数。

(2) 计算判别式 disc。

(3) 计算中间量 p、q。

(4) 计算方程的根 x_1、x_2。

(5) 输出方程的根 x_1、x_2。

例 3-22 的代码如下：

```
1 //计算一元二次方程的根
2 package main
3
```

```
import(
    "fmt"
    "math"
)

func main() {
    var a,b,c,disc,x1,x2,p,q float64
    fmt.Println("请输入一元二次方程三个系数: ")
    fmt.Scanf("%f,%f,%f",&a,&b,&c)
    disc = b*b-4*a*c
    p = -b/(2*a)
    q = math.Sqrt(disc)/(2*a)
    x1 = p+q
    x2 = p-q
    fmt.Printf("一元二次方程的根 x1 = %6.2f x2 = %6.2f\n",x1,x2)
}
```

编译并运行该程序,测试过程如下:

```
请输入一元二次方程三个系数:
1.1,3.1,2.1 ↙
一元二次方程的根 x1 = -1.13 x2 = -1.69
```

小结

本章主要介绍了Go语言的顺序结构程序设计方法,并介绍了Go顺序程序结构、简单语句和复合语句。另外,还介绍了在设计Go程序时的注意事项,包括语句单位划分、左大括号约定、注释语句等。最后介绍了fmt包中的输入输出函数,strings包中的字符串处理函数,strconv包中的字符串转换函数。

通过这一章的学习,首先要掌握顺序结构程序设计的方法,然后要掌握程序中的数据输入输出方法,最后要灵活运用Go标准库函数,设计出高效简洁的顺序结构程序。

习题

3.1 输入一个三位整数,求出该数每个位上的数字之和。如123,每个位上的数字和就是1+2+3=6。

3.2 输入三个float64类型浮点数,分别求出它们的和、平均值、平方和以及平方和的开方,并输出所求出的各个值。

3.3 设f表示华氏温度、c表示摄氏温度、k表示绝对温度,将华氏温度转换为摄氏温度和绝对温度的公式分别为:

c=5/9×(f-32)(摄氏温度)

k=273.16+c(华氏温度)

编写程序，要求通过键盘输入 f 的值，计算 c 和 k 的值并输出。

3.4 编写程序，把极坐标(r,θ)转换为直角坐标(x,y)，其中 θ 的单位为度。转换公式是：

$$x = r \times \cos\theta$$

$$y = r \times \sin\theta$$

提示：要调用 cos()函数和 sin()函数需导入 math 包。

3.5 通过键盘输入英文字符串，并统计字符串中的英文字母的个数，同时输出字符串字节长度。