软件质量保证与测试实验报告

**院（系）：**  **软件工程学院**

**专业： 软件工程**

**姓名:** 刘永伦

**班级：**  **软件171 学号: 17425037**

**成绩:**

[1、测试中经典问题的实现 4](#_Toc18312)

[1.1三角形问题 4](#_Toc28448)

[1.1.1三角形问题描述 4](#_Toc23264)

1.1.2三角形问题分析与设计…………………………………………………………………………..4

[1.1.3三角形问题实现 5](#_Toc25051)

[1.2次日问题 7](#_Toc31902)

[1.2.1次日问题描述 7](#_Toc3902)

[1.2.2次日问题分析与设计 8](#_Toc26192)

[1.2.3 次日问题实现 9](#_Toc19765)

[2.黑盒测试 12](#_Toc18789)

[2.1三角形问题的黑盒测试用例设计 12](#_Toc8877)

[2.1.1等价类划分测试 12](#_Toc19452)

[2.1.2决策类划分测试 14](#_Toc5254)

[2.1.3边界值测试 15](#_Toc435)

[2.2次日问题的黑盒测试用例设计 16](#_Toc26941)

[2.2.1等价类划分测试 16](#_Toc23323)

[2.2.2决策表划分测试 17](#_Toc11007)

[2.2.3边界值测试 19](#_Toc11782)

[3、白盒测试 20](#_Toc25417)

[3.1三角形问题的白盒测试用例设计 20](#_Toc1323)

[3.1.1流程图 20](#_Toc9335)

[3.1.2条件覆盖 21](#_Toc6634)

[3.1.3路径覆盖 21](#_Toc22779)

[3.2次日问题的白盒测试用例设计 21](#_Toc12603)

[3.2.1流程图 21](#_Toc19726)

[3.2.2条件覆盖 22](#_Toc7439)

[3.2.3路径覆盖 23](#_Toc20242)

[4.Eclipse环境下用Junit进行单元测试 23](#_Toc28067)

[4.1 Eclipse环境下用Junit环境配置 23](#_Toc25616)

[4.2 Junit测试脚本的设计 25](#_Toc31390)

[4.2.1三角形问题 25](#_Toc14563)

[4.2.2次日问题 26](#_Toc17414)

[4.3 测试结果 28](#_Toc8187)

[4.3.1三角形 28](#_Toc6226)

[4.3.2次日问题 30](#_Toc32605)

[5. 测试结果分析 32](#_Toc15356)

5.1测试结果……………………………………………………………………………………………………….32

5.1.1 测试用例执行结果……………………………………………………………………………….32

[5.1.2 发现缺陷的用例及设计方法 32](#_Toc25549)

[5.2 测试总结 32](#_Toc27581)

### 1 测试中经典问题的实现

## 1.1三角形问题

### 1.1.1 三角形问题描述

从一个输入对话框中读取三个整数，这三个整数值代表了三角 形三边的长度。程序显示提示信息，指出该三角形究竟是不是不规 则三角形、等腰三角形、还是等边三角形。

### 1.1.2 三角形问题分析与设计

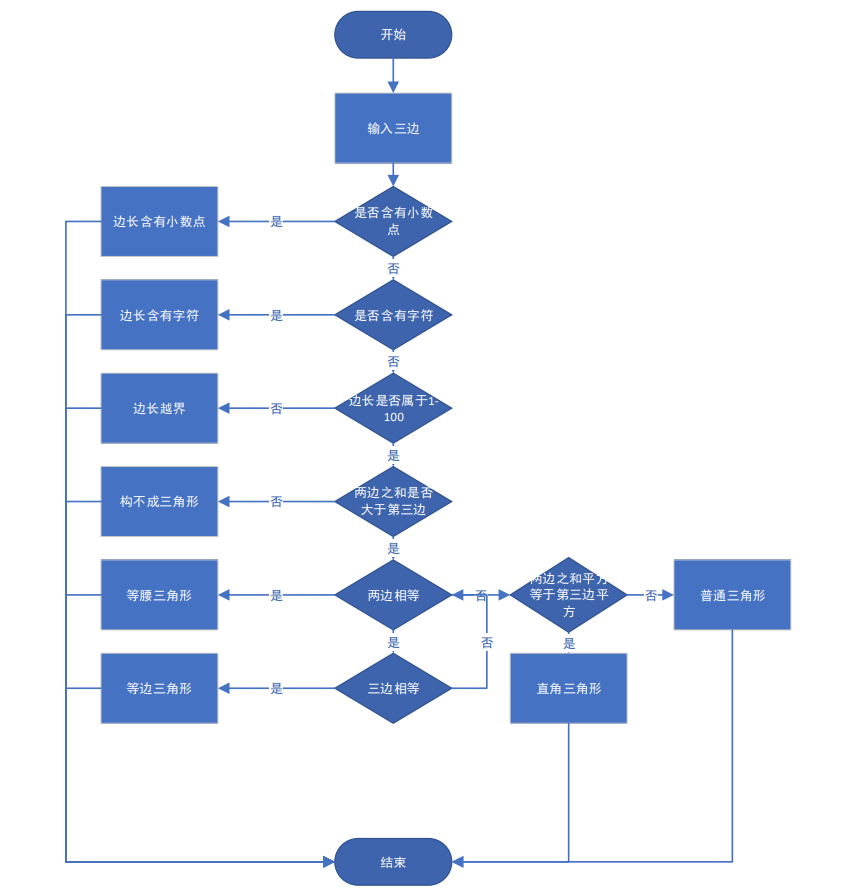


图1.三角形问题的分析流程图

### 1.1.3 三角形问题实现

package test\_exam  
  
import (  
 "fmt"  
 "regexp"  
 "strconv"  
 "strings"  
)  
  
type Triangle struct {  
}  
  
func (Triangle) IsNum (str []byte) bool {  
 reg := regexp.MustCompile("^[-+]?(([0-9]+)([.]([0-9]+))?|([.]([0-9]+))?)$")  
 // 返回是否只有数字  
 return reg.Match(str)  
}  
// 判断字符串是否含有"."分隔符  
func (Triangle) ContainsDot(str string) bool {  
 if strings.ContainsRune(str,'.') {  
 return true  
 }  
 return false  
}  
  
// 判断字符串是否为空  
func (Triangle) IsEmpty (str string) bool {  
 if len(str) == 0 {  
 return true  
 }  
 return false  
}  
  
func (f Triangle) IsTriangle (str1,str2,str3 string) {  
 // 保证三边非空且都是整数，并且都是数字  
 IsNotEmpty,IsNum,NotContainDot := f.AllNotEmpty(str1,str2,str3),f.AllAreNum(str1,str2,str3),f.AllNotContainDot(str1,str2,str3)  
  
 if IsNum && IsNotEmpty && NotContainDot {  
 a,\_ := strconv.Atoi(str1)  
 b,\_ := strconv.Atoi(str2)  
 c,\_ := strconv.Atoi(str3)  
 if !mustFit(a,b,c) {  
 fmt.Println("这三边无法构成三角形")  
 } else {  
 switch typeOfTriangle(a, b, c) {  
 case 1:  
 fmt.Println("等边三角形")  
 case 2:  
 fmt.Println("等腰三角形")  
 case 3:  
 fmt.Println("直角三角形")  
 case 4:  
 fmt.Println("普通三角形")  
 }  
 }  
 } else {  
 if !IsNum {  
 fmt.Println("输入中有错误字符")  
 } else if !IsNotEmpty {  
 fmt.Println("数据不完整")  
 } else {  
 fmt.Println("包含小数点")  
 }  
 }  
}  
// 三角形三边条件  
func mustFit(a,b,c int) bool {  
 if a + b <= c || a + c <= b || b + c <=a{  
 return false  
 }  
 return true  
}  
// 判定三角形类型 1 等边， 2 等腰，3 直角，4 普通  
func typeOfTriangle (a,b,c int) int {  
 if a == b && b == c {  
 return 1  
 } else if a ==b || a ==c || b ==c {  
 return 2  
 } else if a\*a + b\*b == c || a\*a + c\*c == b\*b || b\*b+c\*c == a\*a {  
 return 3  
 } else {  
 return 4  
 }  
}  
  
func (f Triangle) AllAreNum (str1,str2,str3 string) bool {  
 if f.IsNum([]byte(str1)) && f.IsNum([]byte(str2)) && f.IsNum([]byte(str3)) {  
 return true  
 }  
 return false  
}  
  
func (f Triangle) AllNotContainDot(str1,str2,str3 string) bool {  
 return !(f.ContainsDot(str1) || f.ContainsDot(str2) || f.ContainsDot(str3))  
}  
  
func (f Triangle) AllNotEmpty (str1,str2,str3 string) bool {  
 return !(f.IsEmpty(str1) || f.IsEmpty(str2) || f.IsEmpty(str3))  
}

## 1.2 次日问题

### 1.2.1 次日问题描述

输入年月日，其中年份的有效取值范围为【1900，2100】，请输 出输入日期的下一天，例如输入 2013 年 9 月 29 日，输出为 2013 年 9 月 30 日。若输入日期非法，例如输入 2013 年 2 月 30 日，则输出 “输入日期不存在”，若输入日期超出取值范围，例如输入 2013 年 9 月 32 日，则输出“输入日期超出范围”。

Nextday 函数： 分析不同情况下的输入，可以得到不同输出

（1） 当输入数值只要有一个不符合范围，提示“不合理范围”

（2） 当输入非月头月尾数值时，输出日期将天数加 1

（3） 当输入非 2 月月尾时，输出日期将月份加 1

（4） 当输入的为年尾时，输出日期将年份加 1

（5） 当输入闰年 28 日时，输出日期将天数加 1

（6） 当输入闰年 29 日时，输出日期将月份加 1

（7） 当输入非闰年 27 日时，输出日期将天数加 1

（8） 当输入非闰年 28 日时，输出日期将月份加 1

（9） 当输入非闰年 29 日时，提示“输入错误”

### 1.2.2 次日问题分析与设计

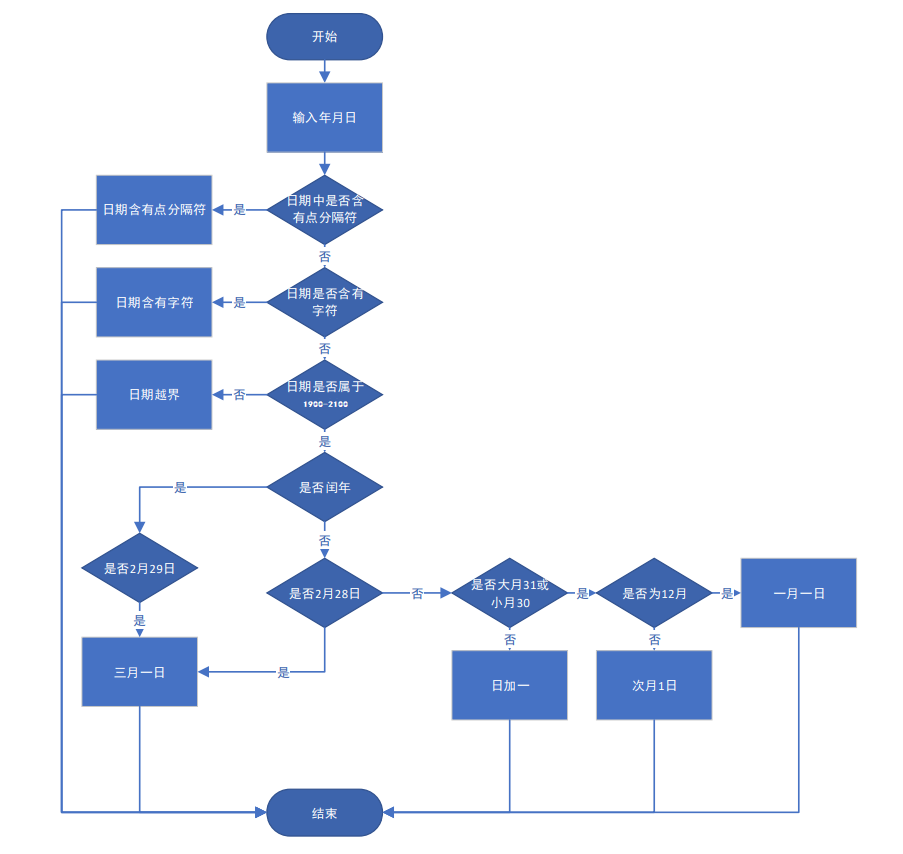


图2.次日问题分析

### 1.2.3 次日问题实现

package test\_exam  
  
import (  
 "fmt"  
 "regexp"  
 "strconv"  
 "strings"  
)  
  
type Unit struct {}  
  
func (Unit) IsNum (str []byte) bool {  
 reg := regexp.MustCompile("^[-+]?(([0-9]+)([.]([0-9]+))?|([.]([0-9]+))?)$")  
 // 返回是否只有数字  
 return reg.Match(str)  
}  
  
// 判断字符串是否为空  
func (Unit) IsEmpty (str string) bool {  
 if len(str) == 0 {  
 return true  
 }  
 return false  
}  
  
// 判断字符串是否含有"."分隔符  
func (Unit) ContainsDot(str string) bool {  
 if strings.ContainsRune(str,'.') {  
 return true  
 }  
 return false  
}  
// 判断是否为闰年  
func(Unit) IsLeapYear (year int) bool {  
 if (year % 4 == 0 && year % 100 != 0) || year % 400 == 0 {  
 return true  
 }  
 return false  
}  
  
func(f Unit) NextDay(str1,str2,str3 string) {  
 IsNotEmpty,IsNum,NotContainDot := f.AllNotEmpty(str1,str2,str3),f.AllAreNum(str1,str2,str3),f.AllNotContainDot(str1,str2,str3)  
 if NotContainDot && IsNum && IsNotEmpty {  
 year,\_ := strconv.Atoi(str1)  
 month,\_ := strconv.Atoi(str2)  
 day,\_ := strconv.Atoi(str3)  
 if year <= 2100 && year >= 1900 {  
 if month <= 12 && month >= 1 {  
 if day <=31 && day >= 1 {  
 if f.IsLeapYear(year) && month == 2 && day == 29 {  
 fmt.Println("下一日期是 ",year," 年 3 月 1 日")  
 } else if !f.IsLeapYear(year) && month ==2 && day == 28 {  
 fmt.Println("下一日期是 ",year," 年 3 月 1 日")  
 } else if month == 12 && day == 31 {  
 fmt.Println("下一日期是 ",year+1," 年 1 月 1 日")  
 } else if BigMonth(month) && day == 31 {  
 fmt.Println("下一日是",year," 年 ",month+1," 月 1 日")  
 } else if month == 4 || month == 6 || month == 9 ||month==11 && day == 30 {  
 fmt.Println("下一日是",year," 年 ",month+1," 月 1 日")  
 } else if !f.IsLeapYear(year) && month == 2 && day > 28 {  
 fmt.Println("本年二月没有这么多天")  
 } else if month == 4 || month == 6 || month == 9 ||month==11 && day == 31 {  
 fmt.Println(month, " 月没有31天")  
 } else {  
 day = day + 1  
 fmt.Println("下一日是：",year, " 年 ", month," 月 ",day," 日")  
 }  
 }else {  
 fmt.Println("日越界")  
 }  
 }else{  
 fmt.Println("月越界")  
 }  
 }else {  
 fmt.Println("年越界")  
 }  
 } else {  
 if !IsNum {  
 fmt.Println("含不合法的字符")  
 } else if !NotContainDot {  
 fmt.Println("含有小数点")  
 } else {  
 fmt.Println("数据不全，出现空字符串")  
 }  
 }  
}  
  
func (f Unit) AllAreNum (str1,str2,str3 string) bool {  
 if f.IsNum([]byte(str1)) && f.IsNum([]byte(str2)) && f.IsNum([]byte(str3)) {  
 return true  
 }  
 return false  
}  
  
func (f Unit) AllNotContainDot(str1,str2,str3 string) bool {  
 return !(f.ContainsDot(str1) || f.ContainsDot(str2) || f.ContainsDot(str3))  
}  
  
func (f Unit) AllNotEmpty (str1,str2,str3 string) bool {  
 return !(f.IsEmpty(str1) || f.IsEmpty(str2) || f.IsEmpty(str3))  
}  
  
func BigMonth(a int) bool {  
 if a == 1||a==3||a==5||a==7||a==8||a==10||a==12 {  
 return true  
 }  
 return false  
}

# **2. 黑盒测试**

## **1 三角形问题的黑盒测试用例设计**

### **1.1 边界值分析**

人们从长期的测试工作经验得知，大量的错误是发生在输入或输出范围的边界上的，而不是在输入范围的内部。因此针对各种边界情况设计测试用例，可以查出更多的错误。例如，在做三角形计算时，要输入三角形的 3 个边长A、B 和 C。这 3 个数值应当满足 A>0、B>0、C>0、A+B>C、A+C>B、B+C>A，才能构成三角形。但如果把 6 个不等式中的任何一个大于号“>”错写成大于等于号 “≥”，那就不能构成三角形。问题恰恰出现在容易被疏忽的边界附近。这里所说的边界是指相当于输入等价类和输出等价类而言，稍高于其边界值及稍低于其边界值的一些特定情况。

#### **1.1.1一般边界值测试**

在三角形问题描述中，除了要求边长是整数外，没有给出其它的限制条件。在此，我们将三角形每边边长的取范围值设值为[1, 100]。

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | b | c | 预期输出 |
| 1 | 60 | 1 | 等腰三角形 |
| 2 | 60 | 2 | 等腰三角形 |
| 3 | 60 | 60 | 等边三角形 |
| 4 | 50 | 99 | 等腰三角形 |
| 5 | 50 | 100 | 构不成三角形 |
| 7 | 2 | 60 | 等腰三角形 |
| 8 | 60 | 60 | 等边三角形 |
| 9 | 99 | 50 | 等腰三角形 |
| 10 | 100 | 50 | 构不成三角形 |
| 11 | 60 | 60 | 等腰三角形 |
| 12 | 60 | 60 | 等腰三角形 |
| 13 | 60 | 60 | 等边三角形 |
| 14 | 50 | 50 | 等腰三角形 |
| 15 | 50 | 50 | 构不成三角形 |

**1.1.2 健壮边界值测试**

| **三角形问题的健壮性边界值测试** | | | | |
| --- | --- | --- | --- | --- |
| 测试用例编号 | 数据列表 | | | 预期输出 |
|  | a | b | c |  |
| 1 | 0 | 50 | 50 | 抛出异常 |
| 2 | 1 | 50 | 50 | 等腰三角形 |
| 3 | 2 | 50 | 50 | 等腰三角形 |
| 4 | 50 | 50 | 50 | 等边三角形 |
| 5 | 99 | 50 | 50 | 等腰三角形 |
| 6 | 100 | 50 | 50 | 构不成三角形 |
| 7 | 101 | 50 | 50 | 抛出异常 |
| 8 | 50 | 0 | 50 | 抛出异常 |
| 9 | 50 | 1 | 50 | 等腰三角形 |
| 10 | 50 | 2 | 50 | 等腰三角形 |
| 11 | 50 | 99 | 50 | 等腰三角形 |
| 12 | 50 | 100 | 50 | 构不成三角形 |
| 13 | 50 | 101 | 50 | 抛出异常 |
| 14 | 50 | 50 | 0 | 抛出异常 |
| 15 | 50 | 50 | 1 | 等腰三角形 |
| 16 | 50 | 50 | 2 | 等腰三角形 |
| 17 | 50 | 50 | 99 | 等腰三角形 |
| 18 | 50 | 50 | 100 | 构不成三角形 |
| 19 | 50 | 50 | 101 | 抛出异常 |

### **1.2 等价类划分**

等价类划分是一种典型的黑盒测试方法，用这一方法设计测试用例完全不考虑程序的内部结构，只根据对程序的要求和说明，即需求规格说明书。我们必须仔细分析和推敲说明书的各项需求，特别是功能需求。把说明中对输入的要求和输出的要求区别开来并加以分解。

等价类划分的办法是把程序的输入域划分成若干部分，然后从每个部分中选取少数代表性数据作为测试用例。每一类的代表性数据在测试中的作用等价于这一类中的其他值，也就是说，如果某一类中的一个例子发现了错误，这一等价类中的其他例子也能发现同样的错误；反之，如果某一类中的一个例子没有发现错误，则这一类中的其他例子也不会查出错误（除非等价类中的某些例子属于另一等价类，因为几个等价类是可能相交的）。使用这一方法设计测试用例，首先必须在分析需求规格说明的基础上划分等价类，列出等价类表。

|  |  |  |  |
| --- | --- | --- | --- |
| **三角形等价类划分描述** | | | |
| 输入条件 | 输入三个整数 | 有效等价类 | 无效等价类 |
| 1. 整数 2. 三个整数 3. 非零整数 4. 正数 | 13.一边为非整  数  14.两边为非整  数  15.三边为非整  数  16.边长个数小  于 3  17.边长个数大  于 3  18.一边为 0  19.两边为 0  20.三边为 0  21.一边小于 0  22.两边小于 0  23.三边小于 0  24.一边大于 100  25.两边大于 100  26.三边大于 100 |
| 输出条件 | 构成普通三角形 | 5.a+b>c  6.a+c>b  7.b+c>a | 27.a+b<=c  28.b+c<=a  29.c+a<=b |
| 构成等腰三角形 | 8.a=b 且满足 5  9.a=c 且满足 6  10.b=c 且满足 7 |  |
| 构成等边三角形 | 1. a=b=c |  |
| 构不成三角形 | 12.至 少 不 满 足  5,6,7 中的一  个 |

### **1.3 决策表**

在一些数据处理问题中，某些操作是否实施依赖于多个逻辑条件的取值在这些逻辑条件取值的组合所构成的多种情况下，分别执行不同的操作 处理这类问题的一个非常有力的分析和表达工具是判定表，或称决策表 （Decision Table）在所有功能性测试方法中，基于判定表的测试方法是最严格的。

**表 3 三角形问题等价类划分测试**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 测试用例编号 | 等价类条件 | 测试用例 | | | | 预期结果 |
| a | b | c | d |  |
| 1 | 1,2,3,4,5,6,7 | 2 | 3 | 4 |  | 普通三角形 |
| 2 | 1,2,3,4,5,6,7,8 | 2 | 2 | 3 |  | 等腰三角形 |
| 3 | 1,2,3,4,5,6,7,9 | 2 | 3 | 2 |  | 等腰三角形 |
| 4 | 1,2,3,4,5,6,7,10 | 3 | 2 | 2 |  | 等腰三角形 |
| 5 | 1,2,3,4,5,6,7,11 | 2 | 2 | 2 |  | 等边三角形 |
| 6 | 1,2,3,4,12 | 1 | 2 | 3 |  | 构不成三角形 |
| 7 | 1,2,3,4,5,6,7,13 | 2.1 | 3 | 4 |  | 抛出异常 |
| 8 | 1,2,3,4,5,6,8,14 | 2.1 | 2.1 | 3 |  | 抛出异常 |
| 9 | 1,2,3,4,5,6,8,15 | 2.1 | 2.1 | 2.1 |  | 抛出异常 |
| 10 | 1,3,4,16 | 2 | 3 | 空 |  | 未测试 |
| 11 | 1,2,3,17 | 2 | 3 | 4 | 3 | 未测试 |
| 12 | 1,2,18 | 0 | 2 | 3 |  | 构不成三角形，边长越界 |
| 13 | 1,2,19 | 0 | 0 | 2 |  | 构不成三角形，边长越界 |
| 14 | 1,2,20 | 0 | 0 | 0 |  | 构不成三角形，边长越界 |
| 15 | 1,2,3,21 | -1 | 2 | 2 |  | 构不成三角形，边长越界 |
| 16 | 1,2,3,22 | -1 | -2 | 2 |  | 构不成三角形，边长越界 |
| 17 | 1,2,3,23 | -2 | -2 | -2 |  | 构不成三角形，边长越界 |
| 18 | 1,2,3,4,24 | 111 | 99 | 99 |  | 构不成三角形，边长越界 |
| 19 | 1,2,3,4,25 | 111 | 111 | 99 |  | 构不成三角形，边长越界 |
| 20 | 1,2,3,4,26 | 111 | 111 | 111 |  | 构不成三角形，边长越界 |
| 21 | 1,2,3,4,27 | 5 | 3 | 2 |  | 构不成三角形 |
| 22 | 1,2,3,4,28 | 5 | 2 | 3 |  | 构不成三角形 |
| 23 | 1,2,3,4,29 | 2 | 5 | 3 |  | 构不成三角形 |

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| C1:a<b+c? | F | T | T | T | T | T | T | T | T | T | T |
| C2:b<a+c? | - | F | T | T | T | T | T | T | T | T | T |
| C3:c<a+b? | - | - | F | T | T | T | T | T | T | T | T |
| C4:a=b? | - | - | - | T | T | T | T | F | F | F | F |
| C5:a=c? | - | - | - | T | T | F | F | T | T | F | F |
| C6:b=c? | - | - | - | T | F | T | F | T | F | T | F |
| A1: | X | X | X |  |  |  |  |  |  |  |  |
| A2: |  |  |  |  |  |  |  |  |  |  | X |
| A3: |  |  |  |  |  |  | X |  | X | X |  |
| A4: |  |  |  | X |  |  |  |  |  |  |  |
| A5: |  |  |  |  | X | X |  | X |  |  |  |

**1.3.2 基于决策表的三角形测试用例**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 用例编号 | a | b | c | 预期输出 |
| 1 | 4 | 1 | 2 | 构 不 成 三角形 |
| 2 | 1 | 4 | 2 | 构 不 成 三角形 |
| 3 | 1 | 2 | 4 | 构 不 成 三角形 |
| 4 | 5 | 5 | 5 | 等 边 三 角形 |
| 5 | ? | ? | ? | 不可能 |
| 6 | ? | ? | ? | 不可能 |
| 7 | 2 | 2 | 3 | 等 腰 三 角 |
| 8 | ? | ? | ? | 不可能 |
| 9 | 2 | 3 | 2 | 等 腰 三 角形 |
| 10 | 3 | 2 | 2 | 等 腰 三 角形 |
| 11 | 3 | 4 | 5 | 普 通 三 角形 |

1. **次日问题的黑盒测试用例设计**

## **2 次日问题的黑盒测试用例设计**

### **2.1 边界值测试**

Month 的取值范围为：[1,12]

Day 的取值范围为：[1,31]

Year 的取值范围为：[1900,2100]

**表 5 次日问题的边界值测试**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 用例编号 | Year | Month | Day | 期望输出 |
| 1 | 1811 | 6 | 15 | 年越界 |
| 2 | 1912 | 6 | 15 | 1912年6月 16 日 |
| 3 | 1913 | 6 | 15 | 1913年 6月 16 日 |
| 4 | 1917 | 6 | 15 | 1917年 6月 16 日 |
| 5 | 2011 | 6 | 15 | 2011年 6月 16 日 |
| 6 | 2012 | 6 | 15 | 2012年6月 16 日 |
| 7 | 2113 | 6 | 15 | 年越界 |
| 8 | 1912 | 6 | -1 | 日越界 |
| 11 | 1912 | 6 | 29 | 1912年 6月 30 日 |

### **2.2等价类划分测试**

NextDate 函数包含三个变量：month 、 day 和 year ，函数的输出为输入日期后一天的日期。 例如，输入为 2006 年 3 月 7 日，则函数的输出为 2006 年 3 月 8 日 。要求输入变量 month 、 day 和 year 均为整数值，并且满足下列条件：

①1≤month≤12

②1≤day≤31

③1920≤year≤2050

1)有效等价类为：

M1＝{月份：1≤月份≤12}

D1＝{日期：1≤日期≤31}

Y1＝{年：1900≤年≤2100}

　　2)若条件 ① ~ ③中任何一个条件失效，则 NextDate 函数都会产生一个输出，指明相应的变量超出取值范围，比如 "month 的值不在 1-12 范围当中 " 。显然还存在着大量的 year 、 month 、 day 的无效组合， NextDate 函数将这些组合作统一的输出： " 无效输入日期 " 。其无效等价类为：

M2＝{月份：月份<1}

M3＝{月份：月份>12}

D2＝{日期：日期<1}

D3＝{日期：日期>31}

Y2＝{年：年<1900}

Y3＝{年：年>2100}

健壮性等价类

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 用例编号 | Year | Month | Day | 预期输出 |
| 1 | 1912 | 6 | 30 | 下 一 日 是 :  1912 年 7 月 1 日 |
| 3 | 2000 | 2 | 23 | 下一日是：2000 年 3 月 1日 |
| 4 | 1912 | 7 | 31 | 下一日是:1912 年 8 月 1日 |
| 5 | 1909 | 2 | 28 | 下一日是：1909 年 3 月 1日 |
| 6 | 2005 | 10 | 31 | 下 一 日 是 :  2005 年 11 月 1  日 |
| 7 | 1997 | 11 | 31 | 11 月可没有 31  天 |
| 8 | 1999 | 12 | 31 | 下一日是：2000 年 1 月 1日 |
| 9 | 1900 | 2 | 29 | 本年 2 月没有  那么多天 |
| 10 | Abcd | 6 | 6 | 日期不合法，含  异常字符。 |
| 11 | 6.6 | 6 | 6 | 日期不是整数 |
| 12 | 1787 | 6 | 5 | 年越界 |
| 13 | 2118 | 7 | 2 | 年越界 |
| 14 | 2000 | 0 | 6 | 月越界 |
| 15 | 2000 | 13 | 6 | 月越界 |

### **2.3 决策表划分测试**

NextDate 函数求解给定某个日期的下一个日期的可能操作（动作桩）如下：

变量 day 加 1 操作；

变量 day 复位操作；

变量 month 加 1 操作；

变量 month 复位操作；

变量 year 加 1 操作。

根据上述动作桩发现 NextDate 函数的求解关键是日和月的问题，通常可以在下面等价类（条件桩）的基础上建立决策表：

M1＝{month：month 有 30 天}

M2＝{month：month 有 31 天，12 月除外}

M3＝{month：month 是 12 月}

M4＝{month：month 是 2 月}

D1＝{day：1≤day≤27}

D2＝{day：day＝28}

D3＝{day：day＝29}

D4＝{day：day＝30}

D5＝{day：day＝31}

Y1＝{year：year 是闰年}

Y2＝{year：year 不是闰年}

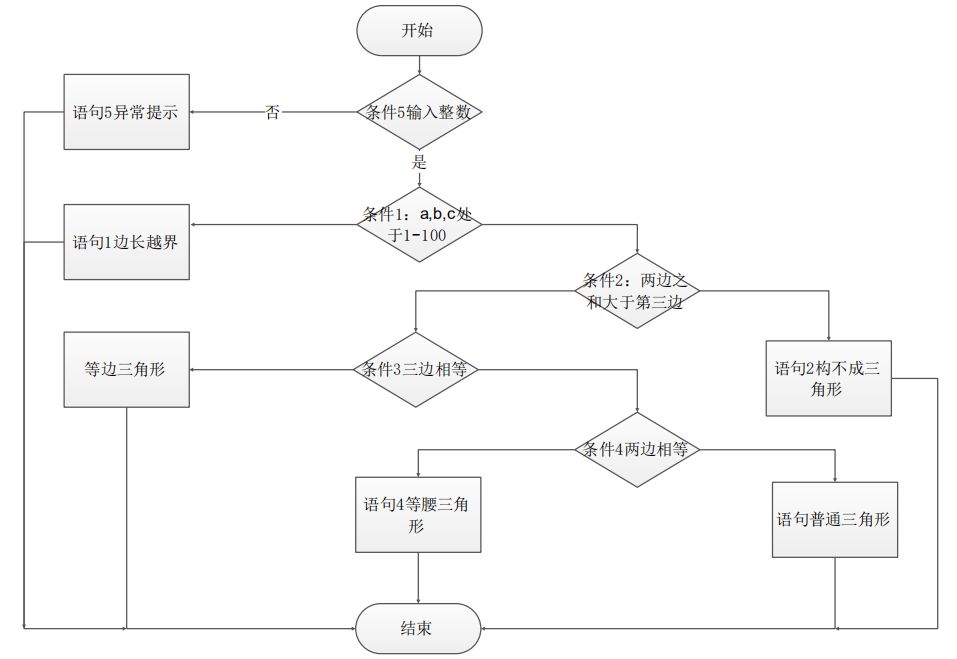
|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 规则选项 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
| 条件 |  |  |  |  |  |  |  |  |  |  |  |
| C1: month在 | M1 | M1 | M1 | M1 | M1 | M2 | M2 | M2 | M2 | M2 | M3 |
| C2: day在 | D1 | D2 | D3 | D4 | D5 | D1 | D2 | D3 | D4 | D5 | D1 |
| C3: year在 | - | - | - | - | - | - | - | - | - | - | - |
| 动作： |  |  |  |  |  |  |  |  |  |  |  |
| A1:不可能 |  |  |  |  | √ |  |  |  |  |  |  |
| A2: day加1 | √ | √ | √ | √ |  | √ | √ | √ | √ |  | √ |
| A3: day 复位 |  |  |  |  | √ |  |  |  |  | √ |  |
| A4：month加一 |  |  |  |  | √ |  |  |  |  | √ |  |
| A5:month复位 |  |  |  |  |  |  |  |  |  |  |  |
| A6：yea加1 |  |  |  |  |  |  |  |  |  |  |  |

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 规则选项 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 |
| 条件 |  |  |  |  |  |  |  |  |  |  |  |
| C1: month在 | M3 | M3 | M3 | M3 | M4 | M4 | M4 | M4 | M4 | M4 | M4 |
| C2: day在 | D2 | D3 | D4 | D5 | D1 | D2 | D2 | D3 | D3 | D4 | D5 |
| C3: year在 | - | - | - | - | - | Y1 | Y2 | Y1 | Y2 | - | - |
| 动作： |  |  |  |  |  |  |  |  |  |  |  |
| A1:不可能 |  |  |  |  |  |  |  |  | √ | √ | √ |
| A2: day加1 | √ | √ | √ |  | √ | √ |  |  |  |  |  |
| A3: day 复位 |  |  |  | √ |  |  | √ | √ |  |  |  |
| A4：month加一 |  |  |  |  |  |  | √ | √ |  |  |  |
| A5:month复位 |  |  |  | √ |  |  |  |  |  |  |  |
| A6：yea加1 |  |  |  | √ |  |  |  |  |  |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 测试用例 | month | day | year | 预期输出 |
| 1-3 | 6 | 15 | 2007 | 下一日是:2007 年6月 16 日 |
| 4 | 6 | 30 | 2007 | 下一日是: 2007 年7月 1 日 |
| 5 | 6 | 31 | 2007 | 6月可没有 31 天 |
| 6-9 | 1 | 15 | 2007 | 下一日是:2007年1月 16 日 |
| 10 | 1 | 31 | 2007 | 下一日是: 2007 年2月 1 日 |
| 11-14 | 12 | 15 | 2007 | 下一日是:2007年12 月 16 日 |
| 15 | 12 | 31 | 2007 | 下一日是：2008 年1月 1 日 |
| 16 | 2 | 15 | 2007 | 下一日是:2007年2月 16 日 |
| 17 | 2 | 28 | 2000 | 下一日是：2000 年  3月 1 日 |
| 18 | 2 | 28 | 2007 | 下一日是：2007 年  3月 1 日 |
| 19 | 2 | 29 | 2000 | 本年 2 月没有那么多天 |
| 20 | 2 | 29 | 2007 | 本年 2 月没有那么 多天 |
| 21-22 | 2 | 30 | 2007 | 本年 2 月没有那么 多天 |

# **3白盒测试**

## **1 三角形问题的白盒测试用例设计**



### **1.1语句覆盖**

|  |  |  |  |
| --- | --- | --- | --- |
| 用例编号 | 用例输入 | 期望输出 | 测试路径描述 |
| 1 | -2,3,4 | 边长越界 | 条件5，条件1，语句1 |
| 2 | A,3,4 | 构不成三角形，边长含字符 | 条件5，路径5 |
| 3 | 2.1,3,4 | 边长不是整数 | 条件5，语句5 |
| 4 | 2,3,4 | 一般三角形 | 条件5，条件1，条件2，条件3，条件4 |
| 5 | 2,2,3 | 等腰三角形 | 条件5，条件1，条件2，条件3，条件4 |
| 6 | 3,3,3 | 等边三角形 | 条件5，条件1，条件2，条件3，条件4 |
| 7 | Null,null,null | 数据不完整 | 条件1 |

### **1.2判定覆盖**

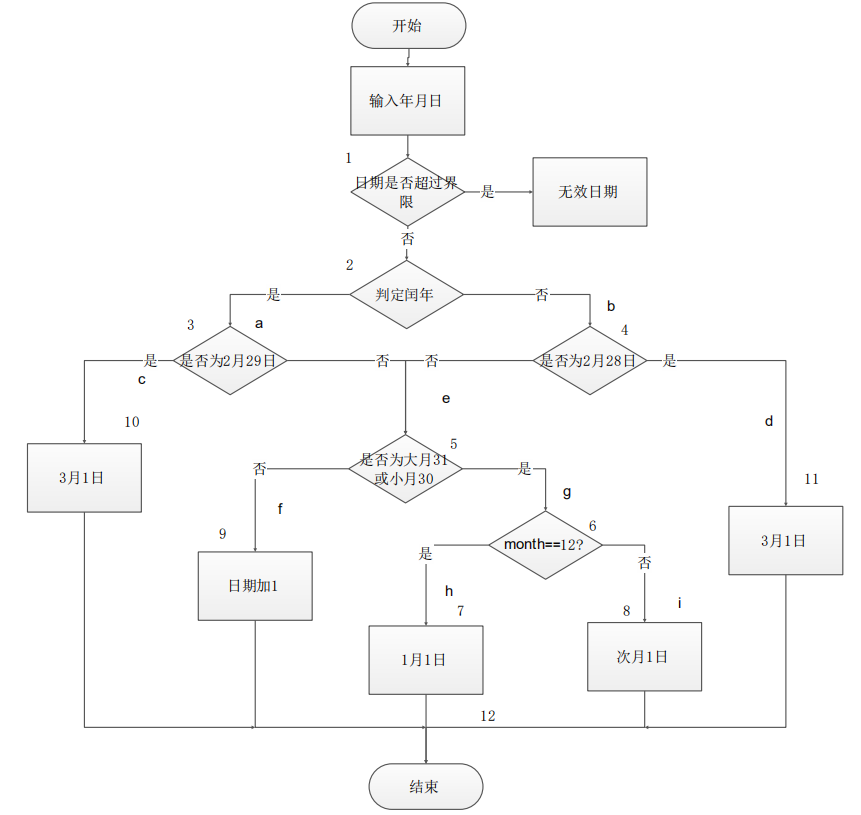
|  |  |  |  |
| --- | --- | --- | --- |
| 用例编号 | 用例输入 | 期望输出 | 测试路径描述 |
| 1 | 50，50,50 | 等边三角形 | 条件1,2,3,5真 |
| 2 | 1,2,3 | 构不成三角形 | 条件1,5真，2假 |
| 3 | 3.14,5,6 | 边长不是整数 | 条件5假 |
| 4 | A,5,6 | 构不成三角形，边长含字符 | 条件5假 |
| 5 | 2,2,3 | 等腰三角形 | 条件1,2,4,5真，3假 |
| 6 | 4,5,6 | 一般三角形 | 条件1,2,5真，3,4假 |

### **1.3路径覆盖**

以下测试路径描述中，TX 表示 X 为 true，FX 表示 X 为 false

|  |  |  |  |
| --- | --- | --- | --- |
| 用例编号 | 用例输入 | 预期输出 | 测试路径描述 |
| 1 | -2，2,4 | 边长越界 | F1,T2,T3 |
| 2 | 3,-4,120 | 边长越界 | T1,F2,F3 |
| 3 | 3,5,9 | 构不成三角形 | T4,T5,F6 |
| 4 | 8,1,3 | 构不成三角形 | F4,T5,T6 |
| 5 | 2,8,4 | 构不成三角形 | T4,F5,T6 |
| 6 | 3,3,3 | 等边三角形 | T7,T8 |
| 7 | 2,3,4 | 普通三角形 | F7,F8,F9,F10,F11 |
| 8 | 2,2,3 | 等腰三角形 | T9 |
| 9 | 3,4,3 | 等腰三角形 | T11 |
| 10 | 4,5,5 | 等腰三角形 | T10 |

## **2 次日问题的白盒测试用例设计**



### **2.1语句覆盖**

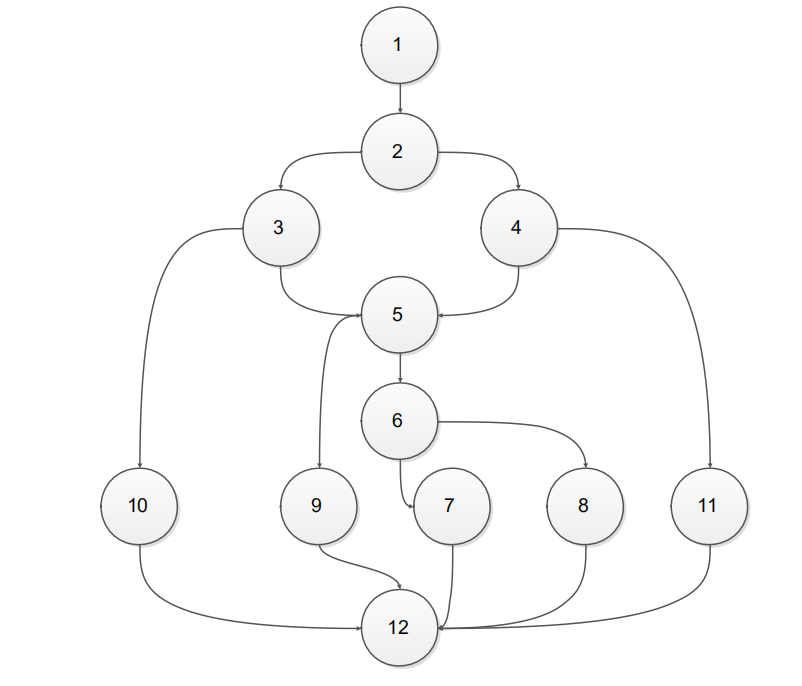
|  |  |  |  |
| --- | --- | --- | --- |
| 用例编号 | 用例输入 | 预期输出 | 测试路径描述 |
| 1 | 2000 2 29 | 2000年3月1日 | T1,T2 |
| 2 | 2001 2 28 | 2001年3月1日 | F1,T3 |
| 3 | 2000 3 28 | 2000年3月29日 | T1,F2,F4 |
| 4 | 2001 11 31 | 2002年1月1日 | F1,F3,T4,T5 |
| 5 | 2001 11 30 | 2001年12月1日 | F1,F3,T4,F5 |

### **2.2判定覆盖**

|  |  |  |  |
| --- | --- | --- | --- |
| 用例编号 | 用例输入 | 预期输出 | 测试路径描述 |
| 1 | 2000 2 29 | 2000年3月1日 | ac |
| 2 | 2001 2 28 | 2001年3月1日 | bd |
| 3 | 2000 3 28 | 2000年3月29日 | aef |
| 4 | 2001 12 31 | 2002年1月1日 | begh |
| 5 | 2001 11 30 | 2001年12月1日 | beji |

### **2.3路径覆盖**

（1）控制流图

（2）环形复杂度

方法一：V(G)=5,(封闭区域)+1=6

方法二：V(G)=16(边数)-12(节点)+2=6

方法三:V(G)=5(判定节点)+1 = 6

（3）导出基本路径

根据上面的计算结果，可导出基本路径集，列出程序的独立路径，可得出程

序段的基本路径集中有 6 条独立路径，每条独立路径为一个独立的测试用例，路

径如下：

路径 1：1-2-3-10-12

路径 2：1-2-3-5-9-12

路径 3：1-2-3-5-6-7-12

路径 4：1-2-3-5-6-8-12

路径 5：1-2-4-5-6-8-12

路径 6：1-2-4-11-12

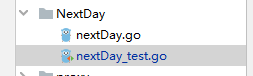
（4）设计测试用例

|  |  |  |  |
| --- | --- | --- | --- |
| 用例编号 | 用例输入 | 预期输出 | 测试路径描述 |
| 1 | 2008 2 29 | 2008年3月1日 | 路径1 |
| 2 | 2008 7 7 | 2008年7月7日 | 路径2 |
| 3 | 2008 12 31 | 2009年1月1日 | 路径3 |
| 4 | 2008 11 30 | 2008年12月1日 | 路径4 |
| 5 | 2007 10 31 | 2008年11月1日 | 路径5 |
| 6 | 2007 2 28 | 2007年3月1日 | 路径6 |

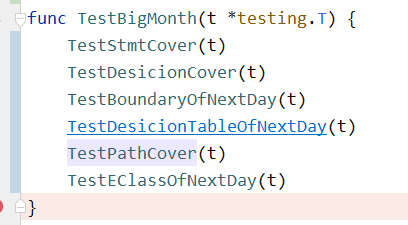
## **3 使用Gotest进行单元测试**

## Go语言测试文件的创建

Golang 的单元测试文件创建非常简单，只需要和需要测试的go文件放在同一文件夹，创建同名文件用\_test作结尾即可创建测试脚本。如图所示：



测试方法只需要函数开头的Test大写首字母，传入参数 t \*testing.T即可，如图所示：

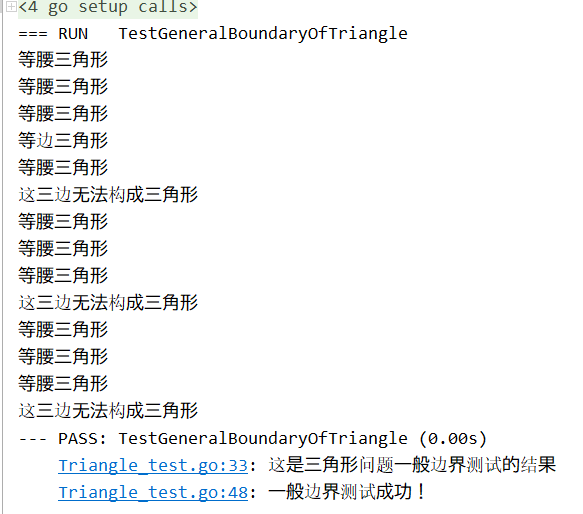


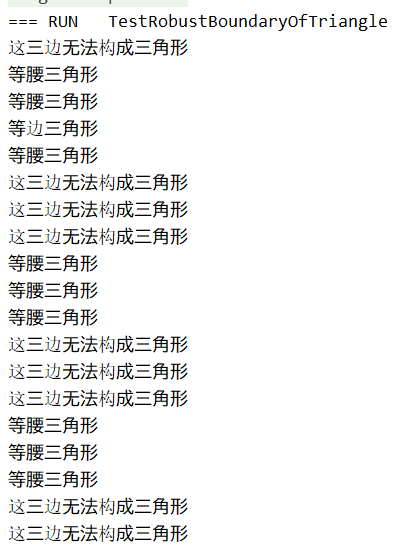
**4 Gotest 测试脚本的设计**

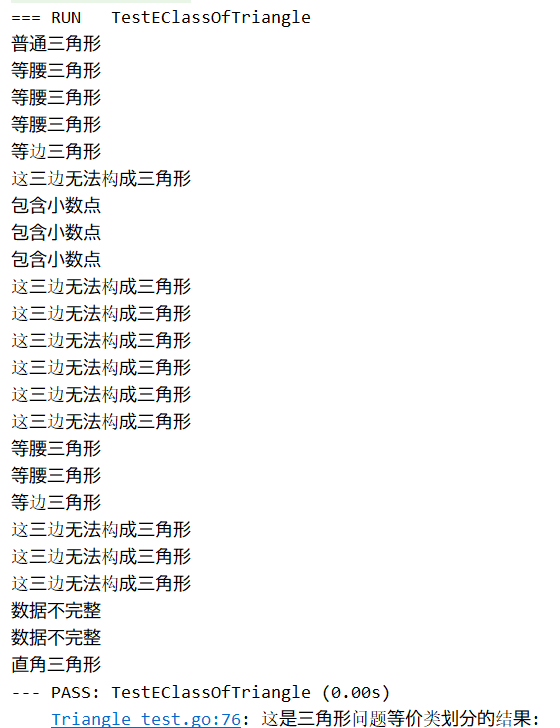
**4.1 三角形的测试脚本**

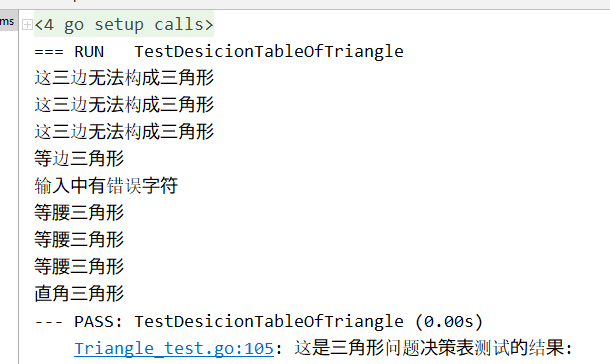
package test\_exam  
  
import (  
 "testing"  
)  
  
var util Triangle  
  
func TestTriangle(t \*testing.T) {  
 TestTriangle\_IsTriangle(t)  
 TestGeneralBoundaryOfTriangle(t)  
 TestConditionCover(t)  
 TestDesicionCover(t)  
 TestEClassOfTriangle(t)  
 TestDesicionTableOfTriangle(t)  
 TestRobustBoundaryOfTriangle(t)  
 TestStmtCover(t)  
}  
  
func TestTriangle\_IsTriangle(t \*testing.T) {  
 util.IsTriangle("a", "1", "2")  
 util.IsTriangle("", "50", "50")  
 util.IsTriangle("101", "50", "50")  
 util.IsTriangle("2.1", "3", "4")  
 util.IsTriangle("100", "50", "50")  
 util.IsTriangle("1", "50", "50")  
 util.IsTriangle("50", "50", "50")  
 util.IsTriangle("4", "5", "6")  
 util.IsTriangle("3", "4", "5")  
}  
  
func TestGeneralBoundaryOfTriangle(t \*testing.T){  
 t.Log("这是三角形问题一般边界测试的结果")  
 util.IsTriangle("60", "60", "1")  
 util.IsTriangle("60", "60", "2")  
 util.IsTriangle("60", "60", "2")  
 util.IsTriangle("60", "60", "60")  
 util.IsTriangle("50", "50", "99")  
 util.IsTriangle("50", "50", "100")  
 util.IsTriangle("60", "1", "60")  
 util.IsTriangle("60", "2", "60")  
 util.IsTriangle("50", "99", "50")  
 util.IsTriangle("50", "100", "50")  
 util.IsTriangle("1", "60", "60")  
 util.IsTriangle("2", "60", "60")  
 util.IsTriangle("99", "50", "50")  
 util.IsTriangle("100", "50", "50")  
 t.Log("一般边界测试成功！")  
}  
  
func TestRobustBoundaryOfTriangle(t \*testing.T) {  
 t.Log("这是三角形问题健壮性边界值测试的结果:")  
 util.IsTriangle("0", "50", "50")  
 util.IsTriangle("1", "50", "50")  
 util.IsTriangle("2", "50", "50")  
 util.IsTriangle("50", "50", "50")  
 util.IsTriangle("99", "50", "50")  
 util.IsTriangle("100", "50", "50")  
 util.IsTriangle("101", "50", "50")  
 util.IsTriangle("50", "0", "50")  
 util.IsTriangle("50", "1", "50")  
 util.IsTriangle("50", "2", "50")  
 util.IsTriangle("50", "99", "50")  
 util.IsTriangle("50", "100", "50")  
 util.IsTriangle("50", "101", "50")  
 util.IsTriangle("50", "50", "0")  
 util.IsTriangle("50", "50", "1")  
 util.IsTriangle("50", "50", "2")  
 util.IsTriangle("50", "50", "99")  
 util.IsTriangle("50", "50", "100")  
 util.IsTriangle("50", "50", "101")  
 t.Log("健壮性边界值测试成功")  
}  
  
func TestEClassOfTriangle(t \*testing.T){  
 t.Log("这是三角形问题等价类划分的结果:")  
 util.IsTriangle("2", "3", "4")  
 util.IsTriangle("2", "2", "3")  
 util.IsTriangle("2", "3", "2")  
 util.IsTriangle("3", "2", "2")  
 util.IsTriangle("2", "2", "2")  
 util.IsTriangle("1", "2", "3")  
 util.IsTriangle("2.1", "3", "4")  
 util.IsTriangle("2.1", "2.1", "3")  
 util.IsTriangle("2.1", "2.1", "2.1")  
 util.IsTriangle("0", "2", "3")  
 util.IsTriangle("0", "0", "2")  
 util.IsTriangle("0", "0", "0")  
 util.IsTriangle("-1", "2", "2")  
 util.IsTriangle("-1", "-2", "2")  
 util.IsTriangle("-2", "-2", "-2")  
 util.IsTriangle("111", "99", "99")  
 util.IsTriangle("111", "111", "99")  
 util.IsTriangle("111", "111", "111")  
 util.IsTriangle("5", "3", "2")  
 util.IsTriangle("5", "2", "3")  
 util.IsTriangle("2", "5", "3")  
 util.IsTriangle("", "", "")  
 util.IsTriangle("2", "3", "")  
 util.IsTriangle("3", "4", "5")  
 t.Log("等价类划分测试成功！")  
}  
  
func TestDesicionTableOfTriangle(t \*testing.T){  
 t.Log("这是三角形问题决策表测试的结果:")  
 util.IsTriangle("4", "1", "2")  
 util.IsTriangle("1", "4", "2")  
 util.IsTriangle("1", "2", "4")  
 util.IsTriangle("5", "5", "5")  
 util.IsTriangle("?", "?", "?")  
 util.IsTriangle("2", "2", "3")  
 util.IsTriangle("2", "3", "2")  
 util.IsTriangle("3", "2", "2")  
 util.IsTriangle("3", "4", "5")  
 t.Log("决策表测试成功")  
}  
  
func TestStmtCover(t \*testing.T){  
 t.Log("这是三角形问题语句覆盖的结果:")  
 util.IsTriangle("-2", "3", "4")  
 util.IsTriangle("A", "3", "4")  
 util.IsTriangle("3.1", "3", "4")  
 util.IsTriangle("2", "3", "4")  
 util.IsTriangle("2", "2", "3")  
 util.IsTriangle("2", "2", "2")  
 util.IsTriangle("", "", "")  
 t.Log("语句覆盖测试通过")  
}  
  
func TestDesicionCover(t \*testing.T) {  
 t.Log("这是三角形问题判定覆盖的结果:")  
 util.IsTriangle("50", "50", "50")  
 util.IsTriangle("1", "2", "3")  
 util.IsTriangle("3.14", "5", "6")  
 util.IsTriangle("A", "5", "6")  
 util.IsTriangle("2", "2", "3")  
 util.IsTriangle("4", "5", "6")  
 t.Log("问题判定覆盖测试通过")  
}  
  
func TestConditionCover(t \*testing.T) {  
 t.Log("这是三角形问题条件覆盖的结果:")  
 util.IsTriangle("-2", "2", "4")  
 util.IsTriangle("3", "-4", "120")  
 util.IsTriangle("3", "5", "9")  
 util.IsTriangle("8", "1", "3")  
 util.IsTriangle("2", "8", "4")  
 util.IsTriangle("3", "3", "3")  
 util.IsTriangle("2", "3", "4")  
 util.IsTriangle("2", "2", "3")  
 util.IsTriangle("3", "4", "3")  
 util.IsTriangle("4", "5", "5")  
 t.Log("条件覆盖测试通过")  
}

**4.2 测试结果**



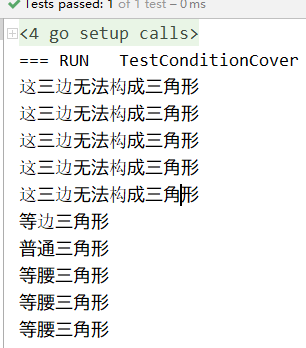












测试指针运行结果：所有测试函数成功pass，测试指针未抛出异常，代码覆盖率为100%

**--- PASS: TestTriangle (0.01s)**

**Triangle\_test.go:33: 这是三角形问题一般边界测试的结果**

**Triangle\_test.go:48: 一般边界测试成功！**

**Triangle\_test.go:142: 这是三角形问题条件覆盖的结果:**

**Triangle\_test.go:153: 条件覆盖测试通过**

**Triangle\_test.go:131: 这是三角形问题判定覆盖的结果:**

**Triangle\_test.go:138: 问题判定覆盖测试通过**

**Triangle\_test.go:76: 这是三角形问题等价类划分的结果:**

**Triangle\_test.go:101: 等价类划分测试成功！**

**Triangle\_test.go:105: 这是三角形问题决策表测试的结果:**

**Triangle\_test.go:115: 决策表测试成功**

**Triangle\_test.go:52: 这是三角形问题健壮性边界值测试的结果:**

**Triangle\_test.go:72: 健壮性边界值测试成功**

**Triangle\_test.go:119: 这是三角形问题语句覆盖的结果:**

**Triangle\_test.go:127: 语句覆盖测试通过**

**PASS**

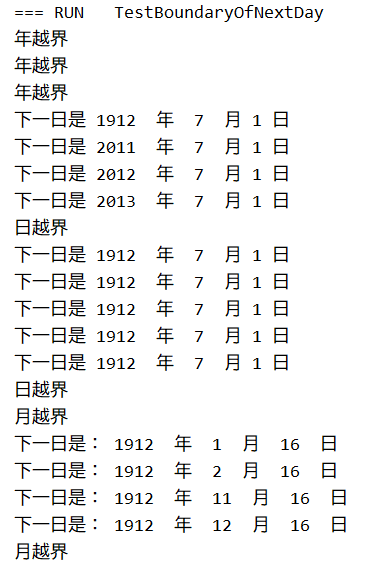
**coverage: 100.0% of statements in ../../../gateway\_demo/...**

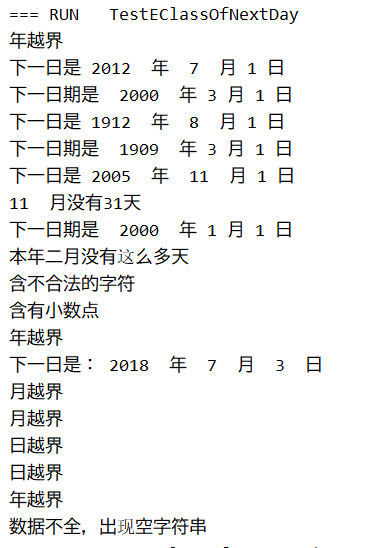
**测试全部通过，代码覆盖率为100%**

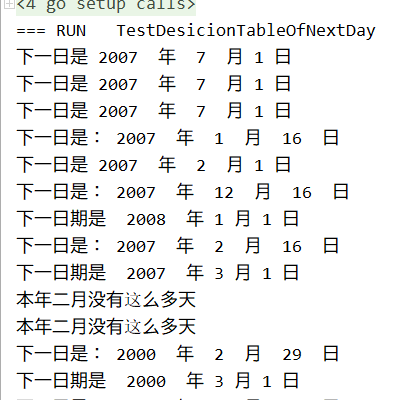
**4.3 次日问题的测试脚本**

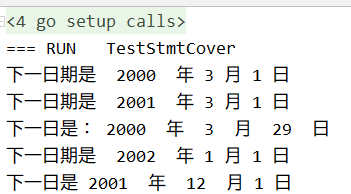
package NextDay  
  
import (  
 "testing"  
)  
  
var util Unit  
  
func TestBigMonth(t \*testing.T) {  
 TestStmtCover(t)  
 TestDesicionCover(t)  
 TestBoundaryOfNextDay(t)  
 TestDesicionTableOfNextDay(t)  
 TestPathCover(t)  
 TestEClassOfNextDay(t)  
}  
  
func TestBoundaryOfNextDay(t \*testing.T) {  
 t.Log("以下是次日问题的边界值用例测试结果:")  
 util.NextDay("1811", "6", "15")  
 util.NextDay("1812", "6", "15")  
 util.NextDay("1813", "6", "15")  
 util.NextDay("1912", "6", "15")  
 util.NextDay("2011", "6", "15")  
 util.NextDay("2012", "6", "15")  
 util.NextDay("2013", "6", "15")  
 util.NextDay("1912", "6", "-1")  
 util.NextDay("1912", "6", "1")  
 util.NextDay("1912", "6", "2")  
 util.NextDay("1912", "6", "29")  
 util.NextDay("1912", "6", "30")  
 util.NextDay("1912", "6", "31")  
 util.NextDay("1912", "6", "32")  
 util.NextDay("1912", "-1", "15")  
 util.NextDay("1912", "1", "15")  
 util.NextDay("1912", "2", "15")  
 util.NextDay("1912", "11", "15")  
 util.NextDay("1912", "12", "15")  
 util.NextDay("1912", "13", "15")  
 t.Log("边界值用例测试通过")  
}  
func TestEClassOfNextDay(t \*testing.T) {  
 t.Log("开始次日问题等价类测试结果:")  
 util.NextDay("1812", "6", "30")  
 util.NextDay("2012", "6", "30")  
 util.NextDay("2000", "2", "29")  
 util.NextDay("1912", "7", "31")  
 util.NextDay("1909", "2", "28")  
 util.NextDay("2005", "10", "31")  
 util.NextDay("1997", "11", "31")  
 util.NextDay("1999", "12", "31")  
 util.NextDay("1900", "2", "29")  
 util.NextDay("abcd", "6", "6")  
 util.NextDay("6.6", "6", "6")  
 util.NextDay("1787", "6", "5")  
 util.NextDay("2018", "7", "2")  
 util.NextDay("2000", "0", "6")  
 util.NextDay("2000", "13", "6")  
 util.NextDay("2000", "6", "0")  
 util.NextDay("2000", "6", "32")  
 util.NextDay("1887", "1", "15")  
 util.NextDay("", "", "")  
 t.Log("等价类测试通过")  
}  
func TestDesicionTableOfNextDay(t \*testing.T) {  
 t.Log("开始次日问题的决策表测试结果:")  
 util.NextDay("2007", "6", "15")  
 util.NextDay("2007", "6", "30")  
 util.NextDay("2007", "6", "31")  
 util.NextDay("2007", "1", "15")  
 util.NextDay("2007", "1", "31")  
 util.NextDay("2007", "12", "15")  
 util.NextDay("2007", "12", "31")  
 util.NextDay("2007", "2", "15")  
 util.NextDay("2007", "2", "28")  
 util.NextDay("2007", "2", "29")  
 util.NextDay("2007", "2", "30")  
 util.NextDay("2000", "2", "28")  
 util.NextDay("2000", "2", "29")  
 util.NextDay("2000", "2", "30")  
 t.Log("决策表测试通过")  
}  
func TestStmtCover(t \*testing.T) {  
 t.Log("开始次日问题语句覆盖的测试:")  
 util.NextDay("2000", "2", "29")  
 util.NextDay("2001", "2", "28")  
 util.NextDay("2000", "3", "28")  
 util.NextDay("2001", "12", "31")  
 util.NextDay("2001", "11", "30")  
 t.Log("测试通过")  
}  
func TestDesicionCover(t \*testing.T) {  
 t.Log("开始次日问题判定覆盖的测试:")  
 util.NextDay("2000", "2", "29")  
 util.NextDay("2001", "2", "28")  
 util.NextDay("2000", "3", "28")  
 util.NextDay("2001", "12", "31")  
 util.NextDay("2001", "11", "30")  
 t.Log("语句覆盖测试通过")  
}  
func TestPathCover(t \*testing.T) {  
 t.Log("开始次日问题路径覆盖的测试:")  
 util.NextDay("2008", "2", "29")  
 util.NextDay("2008", "7", "7")  
 util.NextDay("2008", "12", "31")  
 util.NextDay("2008", "11", "30")  
 util.NextDay("2007", "10", "31")  
 util.NextDay("2007", "2", "28")  
 t.Log("路径覆盖测试通过")  
}

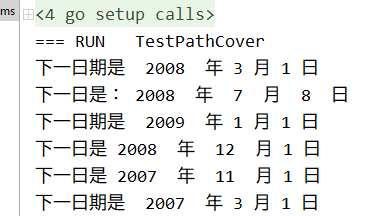
**2.4 次日问题的测试结果**

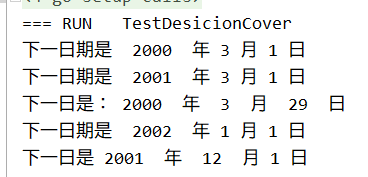












测试指针运行结果：所有测试函数成功pass，测试指针未抛出异常，代码覆盖率为100%

--- PASS: TestBigMonth (0.00s)

nextDay\_test.go:84: 开始次日问题语句覆盖的测试:

nextDay\_test.go:90: 测试通过

nextDay\_test.go:93: 开始次日问题判定覆盖的测试:

nextDay\_test.go:99: 语句覆盖测试通过

nextDay\_test.go:19: 以下是次日问题的边界值用例测试结果:

nextDay\_test.go:40: 边界值用例测试通过

nextDay\_test.go:66: 开始次日问题的决策表测试结果:

nextDay\_test.go:81: 决策表测试通过

nextDay\_test.go:102: 开始次日问题路径覆盖的测试:

nextDay\_test.go:109: 路径覆盖测试通过

nextDay\_test.go:43: 开始次日问题等价类测试结果:

nextDay\_test.go:63: 等价类测试通过

PASS

coverage: 100.0% of statements in ../../../gateway\_demo/...

经过测试，代码均与上述表格中的预期输出一致。

# 5、测试总结：

通过软件三角形问题与次日问题的分析与测试，深刻地理解到以前思考问题是多么的片面，是多么的不周全，实际的软件测试虽然麻烦，但是却非常地能够锻炼人的思维。在测试的过程中仔细地考虑了输入数据的可能情况以及如何提高程序的健壮性，就像是软件测试中的“聪明的猴子”到“愚蠢的猴子”一样。输入的数据完全是不可预知的，各式各样的。

在三角形问题的设计中，从一开始只知道输入正整数，然后考虑到输入数据可能为多边或少边，可能为负数，可能为小数，可能为字符，可能为字符串，可能包含有特殊字符，可能包含“点”操作符，可能是普通三角形，可能是等边三角形，可能是等腰三角形，还有可能是直角三角形，还可能是等腰直角三角形等等细节问题。在实现的过程中考虑到这些问题后然后再一一加以解决，同时提高了思维能力和编程能力和处理细节的能力。

次日问题的设计与测试中与三角形问题类似，但是需要考虑的情况则是更多。 同时也系统的学习并且实践了软件测试的具体方法，比如黑盒测试中的边界值分析，等价类划分，决策表以及白盒测试的语句覆盖，分支覆盖，判定覆盖，路径覆盖等等。并学会了如何使用go test这一golang自带的测试工具进行自动化的测试。