# 第3章

# 程序流程控制结构

程序是通过计算机解决问题和实现功能的重要途径。现实问题往往复杂多变，程序是对现实世界的模拟，为了能够有效地模拟现实世界，需要借助流程控制语句来修改语句执行顺序。Swift 提供了顺序结构、选择结构、循环结构三种流程控制结构。

## 3.1 顺序结构

视频讲解

Swift 顺序结构是最简单的程序结构，也是最常用的程序结构，对应的流程控制是按照书写顺序进行执行。Swift 程序是从上至下进行编写，所以 Swift 顺序结构的程序控制流方向就是从上至下。

Swift 程序由若干条语句组成。在 Swift 程序中每条语句之后不需要添加分号表示结束，将多条语句写到同一行中时，需要使用分号“;”进行分隔。Swift 顺序结构流程图如图 3.1 所示。

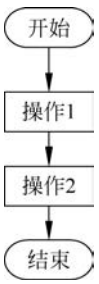

图 3.1 Swift 顺序结构流程图

**例 3.1** A * B 问题。从控制台输入两个整数，分别用变量 A 和 B 进行保存，在屏幕中输出 A * B 的结果。

程序代码：

```
/**
 * 功能:A * B 问题
 * 作者:罗良夫
  */
//A * B
var input1:String? = readLine()
var input2:String? = readLine()
var num1 = Int(input1!)
var num2 = Int(input2!)
print("\(num1!) * \(num2!) = \(num1! * num2!)")
```

执行结果：

```
3 * 4 = 12
```

**例 3.2** 反向输出一个三位数。将一个三位数反向输出，例如输入 358，则输出 853。

程序代码：

```
/**
 * 功能:反向输出三位数
```

```
 * 作者:罗良夫
  */
var num:Int
let s = readLine()
num = Int(s!)!
var h = num / 100
var t = num/10 % 10
var u = num % 10
print("\(num)反向输出结果为:\(u),\(t),\(h)")
```

执行结果：

```
123 反向输出结果为:3,2,1
```

## 3.2 选择结构

现实生活中经常会出现一个事物有多种可能的情况，在程序中通过选择结构来表达这种现象。Swift 提供了 if 和 switch 两种结构来实现选择操作。

视频讲解

### 3.2.1 if 结构

if 结构的功能是根据条件选择执行语句，if 结构的语义类似于汉语中的“如果……那么……”。

if 结构的语法格式如下：

```
if 条件表达式{
    语句块
}
```

**注**：if 关键字后的小括号可以省略；条件表达式的结果为布尔型，即只能是 true 或 false，不能和整数之间进行转换；条件表达式后的大括号不能省略。

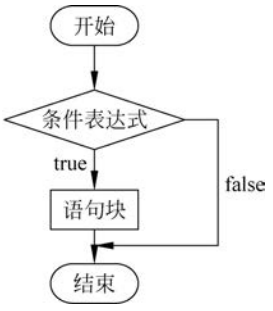

图 3.2 if 结构的流程图

if 结构的流程图如图 3.2 所示。

**例 3.3** 判断年龄是否已达到成年。从键盘输入年龄，如果大于或等于 18 周岁则输出已成年。

程序代码：

```
/**
 * 功能:判断年龄是否已达到成年年龄
 * 作者:罗良夫
  */
var age:Int?
let s = readLine()
age = Int(s!)
if age! >= 18 {
    print("已成年!")
}
```

执行结果：

```
已成年!
```

if 可以与 let 组合进行可选项绑定，防止可选型数据为 nil 导致程序退出，增强代码的健壮性。

可选项绑定的语法格式：

```
if let 常量名 = 可选项{
    使用可选项
}else{
    对可选项为 nil 进行处理
}
```

**注**：可选项绑定时不需要使用“!”进行拆包。

**例 3.4**　可选项绑定示例。

程序代码：

```
/**
 * 功能:可选项绑定
 * 作者:罗良夫
  */
var name:String?  = "罗良夫"
if let tmp = name{
    print("欢迎",name,"登录系统!",separator:"")
}else{
    print("name 为空!")
}
```

执行结果：

```
欢迎罗良夫登录系统!
```

## 3.2.2　if-else 结构

视频讲解

if-else 结构的功能是计算 if 后面的条件表达式，结果为 true 时执行 if 部分的语句块，结果为 false 时执行 else 部分的语句块，适合运用于“是/否”“真/假”等只有两种结果的情况。

if-else 结构的语法格式：

```
if 条件表达式{
    语句块 1
}else{
    语句块 2
}
```

**注**：if-else 在执行过程中，语句块 1 和语句块 2 只有一个会被执行。

if-else 结构的流程图如图 3.3 所示。

**例 3.5**　比较两个数的大小。从键盘输入两个数，比较大小后，输出其中的较大者。

程序代码：

```
/**
 * 功能:比较两数大小
 * 作者:罗良夫
  */
var n1:Int? , n2:Int?
let s1 = readLine()                    //输入 128
```

```
let s2 = readLine()                          //输入 1024
n1 = Int(s1!)
n2 = Int(s2!)
if n1! >= n2!{
    print(n1!)
}else{
    print(n2!)
}
```

执行结果：

```
1024
```

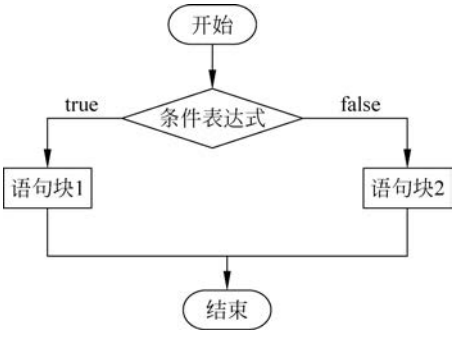


图 3.3 if-else 结构流程图

**例 3.6** 判断闰年。从键盘输入年份，如果年份能被 4 整数但不能被 100 整除，或者年份能被 400 整除时，输出年份是闰年。

程序代码：

```
/**
 * 功能:判断闰年
 * 作者:罗良夫
  */
var year:Int?
let s = readLine()                           //输入 2022
year = Int(s!)
if year! % 4 == 0 && year! % 100 != 0 || year! % 400 == 0{
    print("\(year!)是闰年!")
}else{
    print("\(year!)不是闰年!")
}
```

执行结果：

```
2022 不是闰年!
```

视频讲解

## 3.2.3 if-else if-else 结构

if-else 只能表示有两种可能的情况，对于有多种可能性的情况需要使用 if-else if-else 来表示，if-else if-else 的功能是从多种情况中选择一种情况执行。

if-else if-else 结构语法格式：

```
if 条件表达式 1{
    语句块 1
}else if 条件表达式 2{
    语句块 2
}else if 条件表达式 3{
    语句块 3
}else{
    语句块 4
}
```

**注**：else if 部分可以包含多个；条件表达式允许包含多个；在任意一次执行过程中，只有一个语句块会被执行。

if-else if-else 结构的流程图如图 3.4 所示。

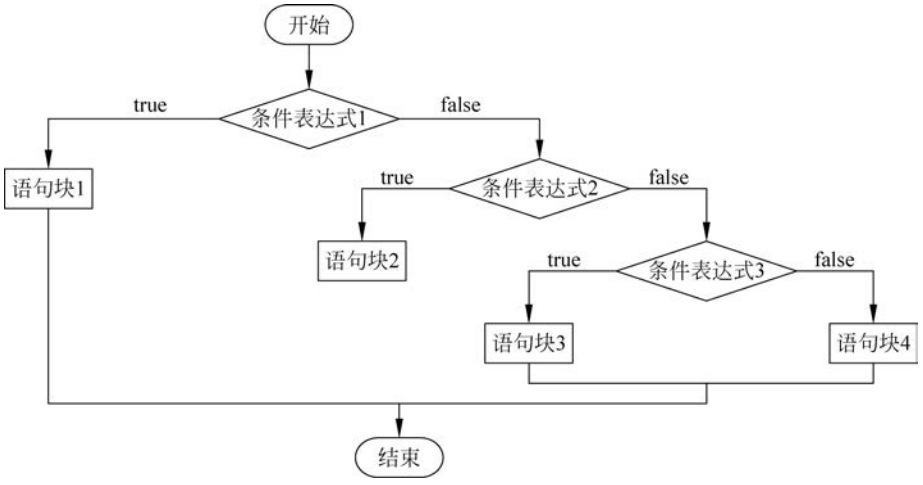


图 3.4　if-else if-else 结构流程图

**例 3.7**　输入分段函数的值。从键盘输入 x 的值，如果 x<=0，y=x+15；如果 x>0 且 x<=1，y=1−x；如果 x>1，y=x−21。

程序代码：

```
/**
 * 功能:分段函数
 * 作者:罗良夫
  */
var x:Double?
var y:Double
let s = readLine()                          //输入 126
x = Double(s!)
if x! <= 0{
    y = x! + 15
}else if x! > 0 && x! <= 1{
    y = 1 - x!
}else{
    y = x! - 21
}
print("当 x = \(x!)时,y = \(y)。")
```

执行结果：

当 x = 126.0 时,y = 105.0

视频讲解

## 3.2.4 switch 结构

switch 结构的功能是从多个选项中选择一种情况执行。switch 可以包含多个条件判断,与 if 结构的区别是,if 是实现二选一的情况,switch 是实现多选一的情况。

switch 结构语法格式:

```
switch 表达式 1{
    case 值 1: 语句块 1
    case 值 2:语句块 2
    ⋮
    default:语句块 n
}
```

switch 结构的执行过程是先计算表达式 1,然后将结果与 case 后的值进行比对,如果值相等则执行对应 case 后的语句块,执行完 case 语句块后退出 switch 结构;switch 后的表达式结果可以为整数类型、浮点数类型、布尔类型、字符类型、字符串类型、元组类型;case 后的值需要与表达式结果的数据类型一致;default 语句块不能省略;case 中可以添加多个值,多个值之间用逗号分隔;switch 结构可以在 case 中添加 fallthrough 关键字,直接执行下一个 case 后的语句块。

switch 结构流程图如图 3.5 所示。

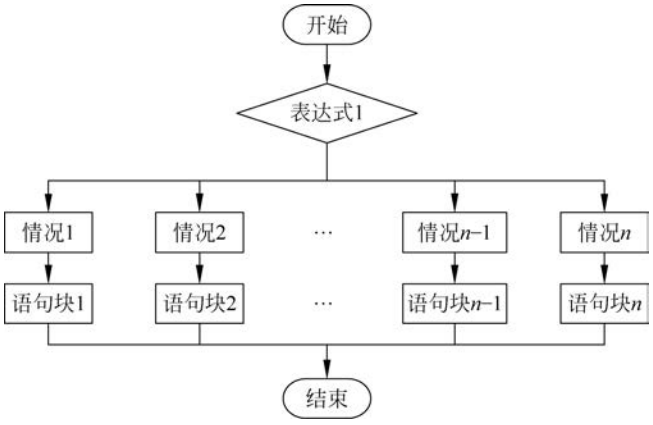


图 3.5 switch 结构流程图

**例 3.8** 成绩段输出。从键盘输入成绩,输出成绩对应的等级。

程序代码:

```
/**
 * 功能:成绩分段输出
 * 作者:罗良夫
   */
var score:Double?
let s = readLine()                          //输入 90
score = Double(s!)
switch score!{
```

```
    case 0..<60:
        print("成绩等级:D")
    case 60..<70:
        print("成绩等级:C")
    case 70..<90:
        print("成绩等级:B")
    case 90...100:
        print("成绩等级:A!")
    default:
        print("成绩错误!")
}
```

执行结果：

```
成绩等级:A!
```

**例 3.9**　中奖程序。从键盘输入一个 1～10 的数，当数值为 1 到 4 时提示没有中奖，数值为 5、6 或 7 时提示中了三等奖，数值为 8 或 9 时提示中了二等奖，数值为 10 时提示中了一等奖。

程序代码：

```
/**
 * 功能:中奖程序
 * 作者:罗良夫
 */
var num:Int?
let s = readLine()//输入 10
num = Int(s!)
switch num{
    case 1,2,3,4:
        print("很遗憾,没有中奖!")
    case 5,6,7:
        print("恭喜,中了三等奖!")
    case 8,9:
        print("恭喜,中了二等奖!")
    case 10:
        print("恭喜,中了一等奖!")
    default:
        print("数值异常!")
}
```

执行结果：

```
恭喜,中了一等奖!
```

**例 3.10**　游戏奖励程序。从键盘输入中奖等级以及是否会员，输出奖励信息。

程序代码：

```
/**
 * 功能:游戏奖励
 * 作者:罗良夫
 */
var grade:Int?
var isMem:Bool?
```

```
let gs = readLine()
let gm = readLine()
grade = Int(gs!)
isMem = Bool(gm!)
switch grade!{
    case 1:
        print("恭喜中了一等奖。【礼品:计算机一台】")
        if isMem! == true{
            fallthrough
        }
    case 11:
        print("【会员赠送手机一台】")
    case 2:
        print("恭喜中了二等奖.【礼品:电饭煲一台】")
        if isMem! == true{
            fallthrough
        }
    case 21:
        print("【会员赠送水壶一个】")
    case 3:
        print("恭喜中了三等奖.【礼品:U盘一个】")
        if isMem! == true{
            fallthrough
        }
    case 31:
        print("【会员赠送手机支架一个】")
    default:
        print("数据错误!")
}
```

执行结果:

```
恭喜中了一等奖。【礼品:计算机一台】
【会员赠送手机一台】
```

**例 3.11** 电话号码识别程序。从键盘输入座机的区号与尾号,根据区号输出对应地区名。

程序代码:

```
/**
 *功能:识别电话号码
 *作者:罗良夫
 */
var phoneNumber:(Int,Int)?
let s1 = readLine()
let s2 = readLine()
var areaCode = Int(s1!)
var number = Int(s2!)
phoneNumber = (areaCode!,number!)
switch phoneNumber!{
    case (010,let tel):
        print("来自北京的电话:\(tel)")
    case (022,let tel):
        print("来自天津的电话:\(tel)")
    case (021,let tel):
```

```
        print("来自上海的电话:\(tel)")
    case (023,let tel):
        print("来自重庆的电话:\(tel)")
    default:
        print("未知电话:\(phoneNumber)")
}
```

执行结果：

```
来自北京的电话:68392513
```

## 3.3　循环结构

### 3.3.1　for-in 结构

视频讲解

for-in 结构的功能是进行指定次数的重复操作，适用于遍历一个集合中所有元素的场景，如对区间求和、处理数组中元素、处理字符串中的字符等操作。

for-in 结构的语法格式：

```
for 循环变量 in 集合{
        语句块
}
```

for 关键字之后不能添加小括号；循环变量前不需要使用 let 或 var 关键字，作用域仅限于循环语句内；关键字 in 后面可以是闭区间、半开区间、数组、字典等集合。

for-in 结构流程图如图 3.6 所示。

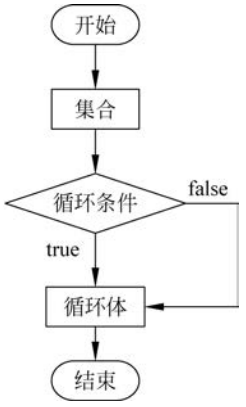

图 3.6　for-in 结构流程图

**例 3.12**　输出九九乘法表。

程序代码：

```
/**
 * 功能:九九乘法表
 * 作者:罗良夫
 */
print("\t\t\t********** 九九乘法表 **********")
for i in 1...9{
    for j in 1...i{
        if(j != i){
         print("\(i) * \(j) = \(i * j) ",terminator:",")
        }else{
             print("\(i) * \(j) = \(i * j)")
        }
    }
}
```

执行结果：

```
                    ********** 九九乘法表 **********
1 * 1 = 1
2 * 1 = 2,2 * 2 = 4
3 * 1 = 3,3 * 2 = 6,3 * 3 = 9
4 * 1 = 4,4 * 2 = 8,4 * 3 = 12,4 * 4 = 16
```

```
5 * 1 = 5,5 * 2 = 10,5 * 3 = 15,5 * 4 = 20,5 * 5 = 25
6 * 1 = 6,6 * 2 = 12,6 * 3 = 18,6 * 4 = 24,6 * 5 = 30,6 * 6 = 36
7 * 1 = 7,7 * 2 = 14,7 * 3 = 21,7 * 4 = 28,7 * 5 = 35,7 * 6 = 42,7 * 7 = 49
8 * 1 = 8,8 * 2 = 16,8 * 3 = 24,8 * 4 = 32,8 * 5 = 40,8 * 6 = 48,8 * 7 = 56,8 * 8 = 64
9 * 1 = 9,9 * 2 = 18,9 * 3 = 27,9 * 4 = 36,9 * 5 = 45,9 * 6 = 54,9 * 7 = 63,9 * 8 = 72,9 * 9 = 81
```

**例 3.13** 查找并输出数组中的最大和最小值。从键盘输入 10 个年龄,输出最大年龄和最小年龄。

程序代码:

```
/**
 * 功能:查找并输出数组中的最大和最小值
 * 作者:罗良夫
 */
var max:Int = 0 , min:Int = 100 , age:Int?
var ageArr:Array<Int> = []
var s:String?
//输入 12、35、6、37、8、92、74、65、43、80
for i in 0..<10{
    s = readLine()
    age = Int(s!)
    ageArr += [age!]
    if i == 0{
      max = age!
      min = age!
    }
    if age! > max{
        max = age!
    }
    if age! < min{
        min = age!
    }
}
print("\(ageArr)中的最大年龄是:\(max)岁,最小年龄是\(min)岁。")
```

执行结果:

```
[12, 35, 6, 37, 8, 92, 74, 65, 43, 80]中的最大年龄是:92 岁,最小年龄是 6 岁。
```

**例 3.14** 英文加密。从键盘输入明文字符串,对明文加密后进行输出。

程序代码:

```
/**
 * 功能:英文加密
 * 作者:罗良夫
 */
var tc:UInt8 = 0
var ciphertext:String = ""
var plaintext:String?
plaintext = readLine()
for c in plaintext!{
    if c == "z"{
        ciphertext += "a"
    }else if c == "Z"{
```

```
            ciphertext += "A"
        }else{
            tc = c.asciiValue!                    //返回字符 c 对应的 ASCII 码
            //将 ASCII 码转换成字符串
            ciphertext += String(Character(UnicodeScalar(tc + 1)))
        }
    }
    print("明文:\(plaintext!),密文:\(ciphertext)")
```

执行结果：

```
明文:Hello world!,密文:Ifmmp!xpsme"
```

## 3.3.2　while 结构

视频讲解

while 结构的功能是根据条件表达式的值来决定是否进行重复操作。while 结构的循环次数在执行前不确定，适用于条件值不确定的场景。

while 结构的语法格式：

```
while 条件表达式{
    语句块
}
```

**注**：while 之后的条件表达式可以添加小括号，也可以省略小括号；条件表达式结果为布尔型；为了避免陷入死循环，需要在语句块中添加使条件表达式为 false 的操作。

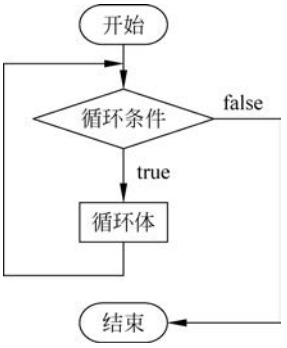

图 3.7　while 结构流程图

while 结构流程图如图 3.7 所示。

**例 3.15**　查找 1000 以内能被 5 和 7 整除的前 10 个整数。

程序代码：

```
/**
 * 功能:查找 1000 以内能被 5 和 7 整除的整数
 * 作者:罗良夫
 */
var i:Int = 1 , count:Int = 0
print("1000 以内能被 5 和 7 整数的前 10 个数:",terminator:"")
while(count < 10){
    if i % 5 == 0 && i % 7 == 0{
        if(count < 9){
            print(i,terminator:",")
        }else{
            print(i,terminator:".")
        }
        count += 1
    }
    i += 1
}
```

执行结果：

```
1000 以内能被 5 和 7 整数的前 10 个数: 35,70,105,140,175,210,245,280,315,350.
```

视频讲解

### 3.3.3 repeat-while 结构

repeat-while 结构的功能是根据条件表达式的值决定是否进行重复，与 while 结构之间的区别是执行次数，repeat-while 至少执行一次循环，while 可能一次循环都不执行。

repeat-while 结构语法格式：

```
repeat{
    语句块
}while 条件表达式
```

循环体的语句块中需要添加终止循环的语句，关键字 while 后面可以省略小括号。

repeat-while 结构流程图如图 3.8 所示。

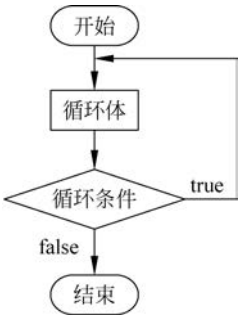


图 3.8 repeat-while 结构流程图

**例 3.16** 查找 1000 以内的水仙花数。

程序代码：

```
/**
 * 功能:查找 1000 以内的水仙花数
 * 作者:罗良夫
 */
var h:Int , t:Int , u:Int , number:Int = 100
print("水仙花数:",terminator:"")
repeat{
    h = number/100
    t = number/10 % 10
    u = number % 10
    if h * h * h + t * t * t + u * u * u == number{
        print(number,terminator:" ")
    }
    number += 1
}while(number < 1000)
```

执行结果：

```
水仙花数:153 370 371 407
```

## 3.4 控制转移语句

视频讲解

### 3.4.1 break 语句

在循环语句的执行过程中有时候需要提前终止循环，Swift 中可通过 break 语句提前结束 for-in、while、repeat-while 循环的执行。当 break 位于嵌套循环中的内层循环时，break 执行后终止所在的内层循环，外层循环仍然会继续执行。

**例 3.17** 计算 1000 以内能被 3 和 7 整除的数。

程序代码：

```
/**
 * 功能:计算 1000 以内能被 3 和 7 整除的数
 * 作者:罗良夫
 */
var num = 1 , count = 0
```

```
while num < 1000 {
    if num % 3 == 0  && num % 7 == 0{
        count += 1
        print(count,num,separator:":")
    }
    if count == 5{
        break
    }
    num += 1
}
```

执行结果：

```
1:21
2:42
3:63
4:84
5:105
```

## 3.4.2　continue 语句

视频讲解

对于循环过程中需要终止本次循环操作的情况，Swift 提供了 continue 语句来实现。在 for-in 结构中执行 continue 后，从 continue 到循环体结束之间的语句会被跳过，进入下一次迭代；在 while 或 repeat-while 结构中执行 continue 后，同样会跳过 continue 之后的循环体语句，直接进入下一次循环条件的判断。

**例 3.18**　打印奇数。从键盘输入一个整数，显示从 1 到该整数范围内的奇数。

程序代码：

```
/**
 * 功能:打印奇数
 * 作者:罗良夫
 */
var num:Int?
let s = readLine()
num = Int(s!)
print("1 到\(num!)范围的奇数:",terminator:"")
for i in 1...num!{
    if i % 2 == 0{
        continue
    }
    if i == num! || i == num! - 1{
        print(i,terminator:".")
    }else{
        print(i,terminator:",")
    }
}
```

执行结果：

```
1 到 50 范围的奇数:
1,3,5,7,9,11,13,15,17,19,21,23,25,27,29,31,33,35,37,39,41,43,45,47,49.
```

视频讲解

### 3.4.3 forloop 语句

break 语句在默认情况下只能跳出当前循环，对于多层循环的跳转需要配合使用 forloop 标签实现。forloop 标签的功能是从内层循环跳转到标签处的外层循环。Swift 中的 forloop 标签名可以是符合标识符命名规则的名称。

**例 3.19** forloop 功能示例。计算 1 到 10 范围内奇数和偶数的和。

程序代码：

```
/**
 * 功能:forloop 功能示例
 * 作者:罗良夫
 */
  var sum1 = 0, sum2 = 0, t = 0
  for i in 1...10{
    t = i
    forloop:for n in 1...i{
        if i % 2 != 0{
            for j in 1...t{
                sum1 += j
                print(j , terminator:" ")
                if j == t{
                    print()
                    break forloop
                }
            }
        }else{
            for k in 1...t{
                sum2 += k
                print(k , terminator:" ")
                if k == t{
                    print()
                    break forloop
                }
            }
        }
    }
}
print("奇数部分的总和 = \(sum1)")
print("偶数部分的总和 = \(sum2)")
```

执行结果：

```
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
1 2 3 4 5 6
1 2 3 4 5 6 7
1 2 3 4 5 6 7 8
1 2 3 4 5 6 7 8 9
1 2 3 4 5 6 7 8 9 10
奇数部分的总和 = 95
偶数部分的总和 = 125
```

## 3.5　小结

程序是现实世界的模拟，为了表达现实世界中的复杂情况，有时需要改变程序的执行流程，Swift 提供了顺序结构、选择结构和循环结构来控制程序的执行流程。

顺序结构的执行流程是按程序的书写顺序执行，适用于执行过程比较固定的情况。

选择结构中 if、if-else、if-else if-else 的执行流程是根据条件运算结果确定执行顺序，Switch 结构是根据表达式运算结果与 case 值的匹配情况决定执行顺序。

循环结构中 for-in 结构根据集合范围重复执行操作，while、repeat-while 结构根据循环条件决定是否重复执行操作，循环过程中可以通过 break、continue、forloop 来改变程序的执行顺序。

## 习题

**一、单选题**

1. Swift 程序中顺序结构的执行顺序是(　　)。

A. 从左到右　　B. 从上到下　　C. 从下到上　　D. 从右到左

2. 下列选项中语法正确的是(　　)。

A.
```
var name="罗良夫"
print(name)
```

B.
```
if(age<18)
print age<18
```

C.
```
for(i=0;i<100;i++)
sum+=i
```

D.
```
switch a
case 1:print(one)
case 2:print(two)
```

**二、编程题**

1. 编写程序在控制台输出 1，2，3，…，20。
2. 编写程序输出 1 到 100 之间的所有奇数之和。

## 实训　选择结构与循环结构

**1. 选择结构**

```
var season = "autumn"
if season == "spring" {
print("春季")
} else if season == "summer" {
print("夏季")
} else if season == "autumn" {
print("秋季")
} else {
print("冬季")
}
```

### 2. 循环结构

```
i = 0
res = 0
repeat {
res += i
i += 1
} while i <= 1000
print("Sum of 1 + ... + 1000 = \(res)")
```