三. 基础程序结构

1. 顺序结构

顺序结构程序不会分叉，也不会拐弯，从上到下顺序执行。

2. 分支结构

在上一章中我们提到了分支结构。

① if 语句

语法：if 条件式 then

语句块.

end.

或

if 条件式 then

语句块1.

else.

语句块2.

end.

对于第一种if结构，如条件为真，则执行语句块中的内容

对于第二种if结构，如条件为真，执行语句块1，如条件为假，执行语句块2

if语句拥有其独立的作用域。

② switch 语句

不同于if语句，switch语句可以拥有多个分支：

switch 表达式

case 标签(常量).

语句块.

end.

default.

语句块

end.

end.

switch语句会比较每一个case提供的标签并执行与表达式的值相等的标签对应的case中的语句。若未找到匹配的，则执行default中的语句。default可以不写，如不写则将直接跳出。

case可以有无限多个，但其标签必须为支持生成唯一值的常量。每个case和default拥有其独立的作用域。

#例 3.2.1

```
System.out.println("Please enter the a、operator and b:").
var a=system.in.input().
var op=system.in.input().
var b=system.in.input().
switch op.
    case "+"
        system.out.println(a+b).
    end.
```

```
        case "-"
            system.out.println(a-b).
        end.
        case "*"
            system.out.println(a*b).
        end.
        case "/"
            system.out.println(a/b).
        end.
    end.
```

思考：

- 这个程序的作用是什么？
- 如何使用if改写这个程序？
- 编写程序时如何选择switch和if？

首先，很明显，这个程序是一个简单的四则运算计算器。程序分别处理了当op=="+"、当op=="-"、当op=="*"和当op=="/"四种情况，并分别输出四种运算结果。

若用if改写，将会是这个样子：

if ....

....

end.

if ....

....

end.

比用switch要麻烦的多。

但对于一些更复杂的程序，比如：

if a>b&&c<=d&&a!=d.

....

end.

就不能改写为switch。

所以，我们将switch定位为处理单一变量的"所有可能性"，对于多变量或者是无法表示为常量的可能性时，还是无法简化为switch。

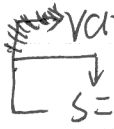
如果不能确定用哪一个，用if绝对不会错就是了。

作业：编写一个简单的应答机器人

示例输入："Hello"　　示例输出："I'm fine!"

3. 循环结构.

我们之前的程序在执行完毕后就直接退出了，要使一部分程序重复执行，我们可以选择复制粘贴很多次。若是程序重复执行次数不固定，或是重复次数很多(比如上百上千次)，复制粘贴就不现用了。我们都知道程序是顺序向下执行的，如果能让程序回到特定的位置，不就能实现重复执行了？如：



这样我们就实现了一个简单的计数器。

在CorScript中，最简单的循环结构loop可以实现此功能：

```
var s=0
loop.
  s=s+1
end.
```

所以我们可以看出，loop循环的语法为：

```
loop.
  语句块
end
```

loop循环的作用就是无脑无限重复执行的快，但往往我们只是需要执行有限次数，完成功能后就继续向下执行。思考：

```
loop
  ....
end
System.out.println("Hello?");
```

最后一行代码会不会执行？

以我们现在掌握的知识来说，最后一行代码并不会执行，因为我们没有办法跳出循环。

那怎样才能实现有限次数的循环呢？再回到我们刚开始的计数器程序，我们加一些东西：

```
var s=0
if s>=10
  并做些什么
end.
s=s+1
....
```

这样，S的值会逐次加1，当S>=10的时候，就跳到循环外部。

在CorScript中，“当”型循环结构 while 可以实现此功能：

```
var s=0
while s<10
    s=s+1
end.
```

while循环的语法为：

```
while 条件.
    语句块.
end.
```

这个循环结构相当于：

```
loop
  if 条件
     语句块.
  else
  #跳出循环
  end.
end.
```

也就是说，当条件为真时，循环执行语句块。

#例 3.3.1

```
var n=system.in.input().
var s=0
var i=1
while i<=n
    s=s+i
end.i=i+1
end.
system.out.println(s).
```

例3.3.1实现了计算 1+2+3+……+n 的值.

思考：

○ n、s、i 的作用分别是什么？

○ 如何实现 1+3+5+……+(2n+1)？

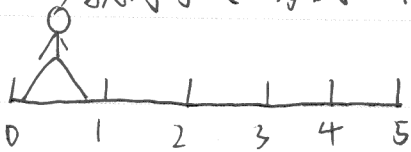
○ 如何实现 1-2+3-4+5+…±n 前面？

在这里n的作用很明显是一个上限，而s的作用是存累加结果，i的作用是遍历范围内的所有数。

在本程序中，i遍历的范围(range)是[1，n]，步长为1。

提示：步长是什么？

原题意，我们可以想象成一个人走一段路：



若步长为1，意思就是每走一步的长度为1。于是每一个脚印与原点的距离就是1的倍数。

同理若步长为2，每一个脚印与原点的距离就是2的倍数。

观察1，3，5，7…… n，我们就发现每一个数之间的差值都是2，所以我们可以认为步长为2。所以只需将

```
j=j+1
```

改成

```
j=j+2
```

即可。

但要实现1-2+3-4……±n，就有些挑战性了。观察发现每两个数之间的差值并不固定，但其绝对值之差却是1。但怎样处理符号呢？我们通过观察发现奇数前都是加号，偶数前都是减号，由此我们就发现了问题的切入点……但怎样判断一个数的奇偶性呢？

由数学知识我们知道偶数的通性是能整除2，而奇数除以2总是余1。CorScript提供了取余运算符——%。但这里需要注意下取余运算符在运算时会忽略小数。所以，只要i%2==0成立，i就是偶数；反之i就是奇数。所以例3.3.1可以写成这样：

```
if i%2==0
  s=s-i
else
  s=s+i
end
i=i+1.
```

通过这种方式我们就实现了计算 1-2+3-4 ---±n

题外话：另一种实现方法是 ~~s=s+i*(-1)^(i+1).~~ s=s+i*(-1)^(i+1).

为什么？

作业：已知 $\frac{\pi}{4} \approx 1-\frac{1}{3}+\frac{1}{5}-\frac{1}{7}+\cdots\cdots+n$，其中 $n<10^{-6}$

计算π的近似值。

4. 直到型循环与循环控制语句

"当"型循环——也就是while——的特点可以总结为"先检查条件，再执行语句"。但在上节最后的作业中，我们发现我们需要提前计算出n的值用于循环的条件。我们想实现"先执行语句，再检查条件"，while就不能用了。CorScript提供了"直到"型循环——loop-until：

语法：loop.

语句块.

until 条件.

end.

先执行语句块，直到条件成立时，跳出循环。

注意在until与end之间不能有任何语句。

while和loop-until都属于固定格式循环，若要实现更复杂的循环，我们要用到循环控制语句：

break	跳出最上层的循环
continue	进入最上层循环的下一轮

这两个语句适用于任何循环。

作业：使用loop循环和循环控制语句分别表示while和loop-until两种循环