## 四、函数与自定义函数的使用

### 1、函数是什么？

函数（Function）在数学中指的是自变量与值之间的映射，但在程序中中函数的概念要复杂的多。

程序中的函数可以理解为子程序，调用者会传入参数，函数处理后将处理结果返回（Return）。

### 2. CorScript 普通函数

语法：function <函数名>（<参数列表（可选）>）

函数体

end.

参数列表中只需指定参数名，各个参数以逗号分隔，参数名不能重复。

若将参数列表留空，则表明此函数不需要参数。

函数被调用时参数会从左到右依次传入，如：

```
var a = func(1, 3, "Hello")
        ①  ②  ③   实际参数
        ↓  ↓  ↓
function func(a, b, c)   形式参数
```

参数与普通的变量并不相同，参数都是对实际参数的引用。也就是说对参数的修改会直接影响到调用者。

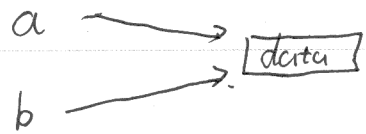
#例 4.2.1.

```
function test(a).
    a=a+1.
end.
var b=10.
test(b).
system.out.println(b).
```

程序将会输出11，因为a是b的一个引用。

提示：引用是什么？

引用相当于一个变量的"别名"。从底层来看，则是两个或多个变量名指向同一块数据，如例4.2.1：




正常情况下是一个变量名对应一个数据：


我们以后还要讲到共享数据的指针，指针与引用完全不同。

函数拥有其独立的作用域。

3. 在函数中返回值.

函数处理完数据后一般会返回一个结果。若不返回，CorScript 都会默认返回 0。要返回数据，应使用返回语句：

return 表达式

返回语句会立刻终止函数的运行并返回表达式的值。

若想仅终止函数运行而不返回值，可以忽略return语句的表达式。

返回的值将作为函数调用表达式的值。

作业：尝试使用 CorScript 表示分段函数 $f(x)=\begin{cases}\sin x, x\le 0\\ \ln(x+1), x>0\end{cases}$

提示：正弦函数为 math.sin(x)，自然对数函数为 math.ln(x).

4. 递归与返回值优化

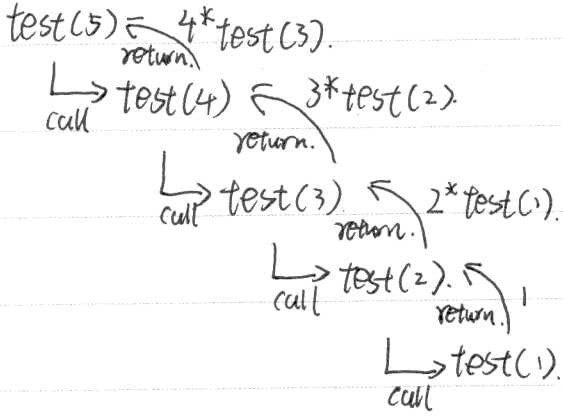
递归指的是函数调用自身的行为。递归算法是计算机科学中常用的算法。

#例 4.4.1.

```
function test(n)
    if n<=1
        return n
    else
```

```
    return n*test(n-1).
  end.
end.
```

例4.4.1实现了计算阶乘。让我们以5!(即test(5))为例分析一下该程序。




也就是说，test(5)等价于1*2*3*4*5。在这里我们多设计了函数在n==1处终止了递归，所以这是一个有限递归。test(1)被称为递归终点。

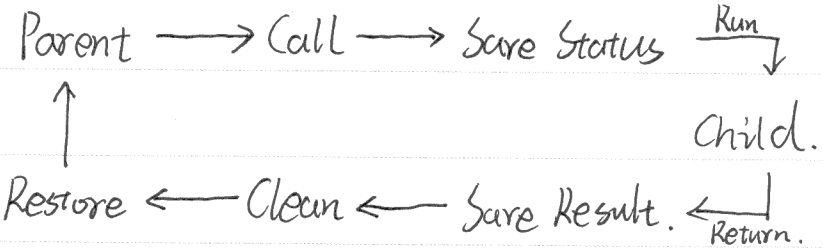
若不设计递归终点，递归将无限下陷，这种递归被称为无限递归。由于系统限制递归往往有一个上限，在CmScript中是1024层。

Tips：在函数式编程语言中，由于不存在if、while等结构，故允许无限递归。

作业：利用递归实现计算 1+2+3+4+…+n 的值

递归固然强大，但我们也必须要知道递归的代价。

前面我们介绍过函数实际上是一种子程序，在调用子程序之前，系统会将父程序的状态保存到内存中，然后将机器的控制权交给子程序。子程序退出时，会将返回值保存到内存中，然后清清子程序使用过的资源，然后从内存中读取保存的状态并恢复。




函数调用流程图

所以，CorScript 以及 C/C++/Java 等主流编程语言不支持无限递归的原因之一是内存有限，无法保存太多的状态和返回值。

CorScript 编程语言为提高函数执行效率加入了返回值优化的功能。例如例4.4.1中的程序，如去掉每一次的乘以 n，若调用 test(100)，传统编程语言会把 1 这个值复制 100 次，但 CorScript 将只会复制 1 次。

5、Lambda表达式

Lambda表达式是一种匿名函数。在CorScript中，Lambda表达式进一步进行了简化，仅保留了单一语句。

语法：[]=>(<参数列表(可选)>) -> 表达式

Lambda表达式非常灵活，其本身可以出现在其他表达式中，如：

test(2,3,[](a,b)->math.max(a,b))

Lambda表达式本质上还是函数，但只有一个return语句。

#例 4.5.1

```
function test0(x)
    return x^2-2*x+1
end

var test1=[](x)->x^2-2*x+1
```

在这里test0和test1实际上等价。

我们可以发现Lambda表达式在某些方面灵活性要远大于正常函数，但由于Lambda表达式形式过于简单，所以实现的功能也很有限。读者应根据自己的需求在两者之间作出选择。

# 五、数组

## 1、数组是什么？

数组(Array)是最基础的数据结构之一。

数组并不是“一组数”，其本质上是在内存中连续排列的一组数据。


数组的内存结构

## 2、数组的字面量

#例5.2.1

```
var arr = {0, 2, 4, 6, "Hello"}
```

我们可以看到，数组的字面量是大括号括起的、以逗号为间隔的若干个数据。空的大括号代表空数组，用new运算符构造的也是空数组。

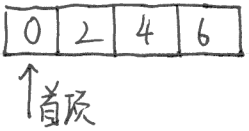
#例5.2.2

```
var a = {}
var b = new array
```

两者等价。

## 3、访问数组中的元素

在第一节中我们讲到数组是在内存中连续排列的一组数据，底层在存储数组时采用存储首元素内存地址的方式：




这样在访问数组元素时，就使用首项地址+偏移量的方式。所以第一项就是首项+0，第二项就是首项+1……依次类推。我们为了能更好理解，将偏移量称为下标。下标的起始值为0，负的下标没有意义。

举例5.3.1.

```
var arr={2,4,8,10}
system.out.println(arr[0]+arr[2]);
```

此程序会输出"10"。

"<数组名>[表达式]"是下标访问运算符。

例5.3.1输出了数组arr的第一项和第三项之和。

作业：计算数组{1,6,8,12,64}的各项之和

4. 变长数组(VLA)

若定义一数组arr：var arr={3,4,5}

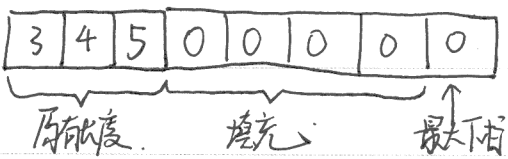
思考：arr[3]有何意义？

很明显，数组arr最后一个元素的下标为2。那么arr[3]很明显超出了arr的范围，这会发生什么呢？答案是：自动扩容。

变长数组(VLA)可以实现用到多少就会自动扩容以满足需求。也就是说永远不会超出范围。

那么问题来了，arr[3]虽然不会超出范围，但它等于多少呢？

变长数组在扩容时会全部填零。也就是说，若访问arr[5]，那么arr[3]、arr[4]和arr[5]都会被填充为0。




变长数组固然方便，但也是一把双刃剑。有时我们不希望数组自动扩容，该怎么办？

思考：若不想让数组扩容，下标最大值是多少？

CorScript为数组提供了获取大小的方法：size

用法：<数组名>.size()　返回数组内容元素的个数

所以，数组下标最大值就是arr.size()-1，我们只需保证下标小于等于最大值即可。

#例5.4.1

```
var arr = {2,4,6,8}
var index = system.in.input()
```

```
if index<=arr.size()-1.
    system.out.println(arr[index]).
else.
    system.out.println("Out of range.").
end.
```

例5.4.1实现了剔除超出范围的下标。

## 5.数组的应用

到目前为止我们接触到的数组元素都很少，实际上数据量小的情况下不使用数组也能解决大多数问题。

数组往往用来解决数据量较大或是数据量大的问题。

在尝试解决问题之前，我们先介绍如何遍历数组。

#例5.5.1

```
var arr=[1,3,5,7,9,11,13,15,17,19]
var i=0
while i<=arr.size()-1
    system.out.println(arr[i]).
    i=i+1.
end.
```

我们通过循环使下标i从0开始，每次循环加1，直到i大于最大下标。也就是说，每一次循环我们都让下标向后移一个单位。

作业：尝试通过遍历数组来数组各元素之和

到目前为止我们数组中的数据都是提前设定的，所以我们首先要介绍如何收集数据。

#例5.5.2.

```
var n=system.in.input().
var arr=new array.
var i=0
while i<n
    arr[i]=system.in.input().
    i=i+1
end.
```

让我们来分析一下这个程序。此程序首先要求用户输入数据的个数，然后将用户输入的n数据依次存入数组。

数组的初始大小为0，我们利用VLA使数组逐渐增长。arr[i]实际上已超出范围，VLA机制会增长并填0，然后我们将用户

的输入赋值给arr[i]，从而实现了存储数据。

我们完成了收集数据，接下来要处理数据。

#例 5.5.3

```
function average(arr)
    var s=0
    var i=0
    while i<=arr.size()-1
        s=s+arr[i]
        i=i+1
    end
    return s/arr.size()
end
```

例5.5.3实现了计算平均值。在上节的作业中我们已经实现了求和，那么平均值的计算只需除以元素个数即可。在这里我们将这个功能包装成了函数，方便多次使用。

作业：① 实现计算一组数据的方差

② 实现找出最大值、最小值

要求：使用函数将功能包装起来