第03章 中间代码 Pcode (上)

上一章中介绍了 TinyC 源程序的语法，本章及下一章介绍中间代码 Pcode 的语法，同时介绍 Pcode 虚拟机的内部结构、如何用 Pcode 模拟器运行 Pcode 、以及 Pcode 命令与 TinyC 程序之间的对应关系。

3.1 Pcode 、 Pcode 虚拟机及 Pcode 模拟器概述

Pcode 是 TinyC 编译器的中间代码，是本人参考 pascal 编译器的中间代码 pcode 、并结合逆波兰表达式（后缀表达式）的逻辑后，设计出的一种非常简单的、基于栈和符号表的虚拟代码。

Pcode 虚拟机是一个用来运行 Pcode 命令的、假想的机器，它包括：一个代码区（code）、一个指令指针（eip）、一个栈（stack）、一个变量表（var_table)、一个函数表（func_table）以及一个标签表（label_table）。

Pcode 模拟器是本人用 Python 编写的一个解释和运行 Pcode 的程序，它实现了 Pcode 虚拟机的全部要素。

Pcode 的所有命令都是对栈顶及附近的元素进行操作的，如 push/pop 命令分别将元素入栈和出栈，add 命令将栈顶的两个元素取出，相加后再放回栈顶。如：

```
x = 1 + 2 * 3;
```

可以翻译成以下Pcode：

```
push 1
push 2
push 3
mul
add
pop x
```

看起来是不是很眼熟，和所谓的逆波兰表达式（后缀表达式）有点相似吧?

```
1 2 3 * +
```

Pcode 中以分号 ”;” 开始的为注释，以标识符加冒号的为标签（如 “Label:” ）。

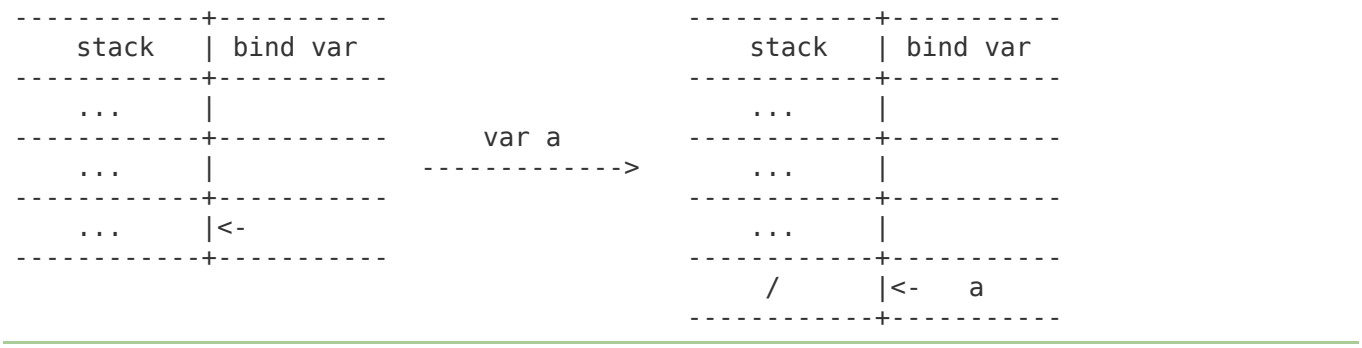
Pcode 命令一共只有7组，都是非常简单的命令，其中也可以分为系统命令和自定义命令两种，自定义命令其实就是函数调用，是对系统命令的扩充。以下详细介绍 Pcode 的系统命令、各命令执行过程中 Pcode 虚拟机的状态变化、如何创建自定义命令（函数）、以及如何用 Pcode 模拟器运行 Pcode 。

3.2 变量声明命令

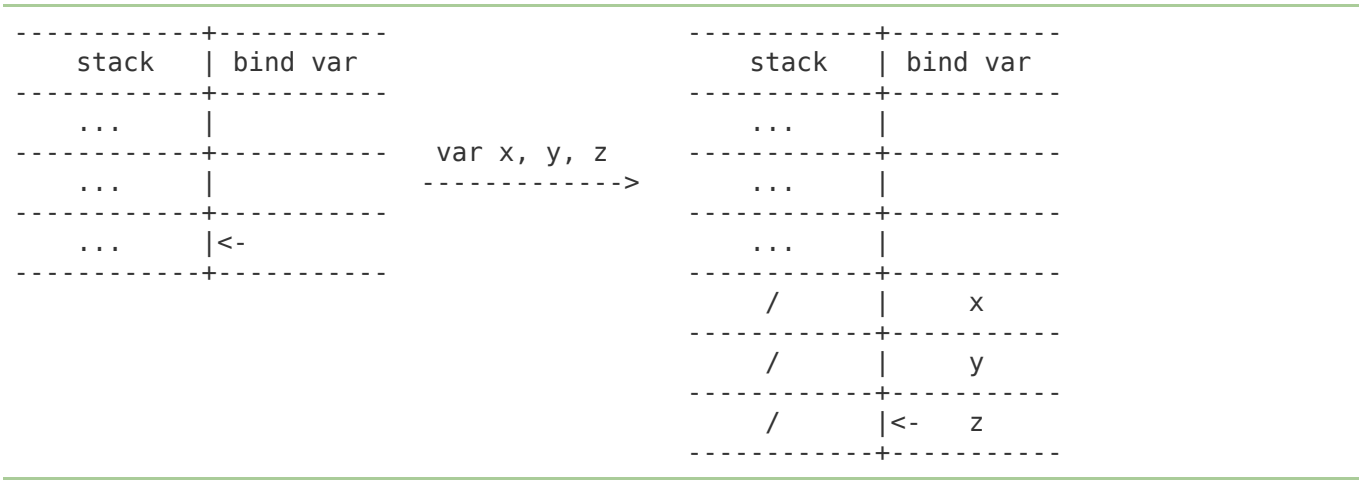
var 命令，声明变量，向下增长栈的空间，将新增的空间分配（绑定）给刚声明的变量，并将变量名及分配给它的地址保存到变量表中。有以下用法：

```
var a           ; 栈顶向下增长 1 个单元，将新的栈顶单元分配（绑定）给 a
var x, y, z     ; 栈顶向下增长 3 个单元，将新的栈顶单元分配（绑定）给 x, y, z
```

“var a” 命令运行后栈及符号表的变化如下所示，其中左边为栈，右边为绑定的符号表，”<-” 指向栈顶，该命令运行后，栈顶向下增长1个单元，并将变量a绑定到新的栈顶单元上。斜杠 “/” 来表示此单元尚未赋初始值，如果此单元在被赋初值之前被使用（读取），则虚拟机将出错终止。



“var x, y, z” 命令运行后栈及符号表的变化所示，该命令运行后，栈顶向下增长 3 个单元，并将变量 x, y, z 绑定到新的栈顶单元上。



var 命令运行后， Pcode 虚拟机会将刚刚声明的变量及分配给它的地址记录在变量表中，在后面的命令中可以根据变量名称来引用其内容。

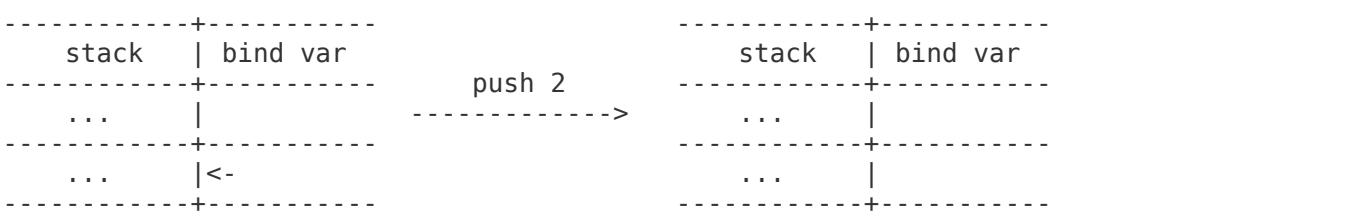
以上图示中，栈的增长方向都是向下，这是为了和大部分计算机系统架构和编译原理教材的惯例保持一致。

3.3 入栈及出栈命令

push / pop 命令，将元素放入栈顶，或取出栈顶元素。有以下用法：

```
push 2   ; 将常数 2 入栈
push a   ; 将变量 a 的值入栈， a 必须已被声明、且已被赋值过
pop      ; 将栈顶向上减少一个单位
pop a    ; 取出栈顶元素，并赋给变量 a ， a 必须已被声明
```

“push 2” 命令运行后，常数 2 被放入栈顶，如下：



```
                                                          2       |<-
                                                    ------------+-----------
```

“push a” 命令运行后，变量 a 的值 <5> 被放入栈顶，如下图。

```
------------+-----------                            ------------+-----------
    stack   | bind var                                  stack   | bind var
------------+-----------                            ------------+-----------
     5      |    a                                       5      |    a
------------+-----------         push a             ------------+-----------
    ...     |                 -------------->           ...     |
------------+-----------                            ------------+-----------
    ...     |<-                                         ...     |
------------+-----------                            ------------+-----------
                                                         5      |<-
                                                    ------------+-----------
```

当虚拟机执行 push 命令时，若后面是一个变量名，则虚拟机会在其变量表中查找此变量名，如果查到了，且该变量的值不是空值 “/” ，则将此变量名对应的值放入栈顶，但若此时该变量尚未被赋初值（为空值 “/” ），则虚拟机将出错而终止，如果没有查找到，则虚拟机也会出错终止。

“pop” 命令运行后，栈顶向上减少一个单位，栈顶元素被丢弃，如下：

```
------------+-----------                            ------------+-----------
    stack   | bind var                                  stack   | bind var
------------+-----------          pop               ------------+-----------
    ...     |                 -------------->           ...     |
------------+-----------                            ------------+-----------
    ...     |                                           ...     |<-
------------+-----------                            ------------+-----------
    ...     |<-
------------+-----------
```

“pop a” 命令运行后，栈顶的元素被取出，并将其值赋给了变量 a ，相当于 a = stack.pop() ，此命令是唯一一个能给 直接 给变量赋值的命令。栈的变化如下：

```
------------+-----------                            ------------+-----------
    stack   | bind var                                  stack   | bind var
------------+-----------                            ------------+-----------
    ...     |    a                                       5      |    a
------------+-----------         pop a              ------------+-----------
    ...     |                 -------------->           ...     |
------------+-----------                            ------------+-----------
    ...     |                                           ...     |<-
------------+-----------                            ------------+-----------
     5      |<-
------------+-----------
```

当虚拟机执行 pop 命令后，若后面是一个变量名，虚拟机会在其变量表中查找此变量名，若查到了，则虚拟机会将栈顶元素取出，赋给该变量名对应的栈单元，若没查到，虚拟机会出错终止。

此处同样需要注意的是，若此时 栈顶单元 尚未被赋初值（为空值 “/” ），则虚拟机将出错而终止。总而言之，栈上未被赋初值的单元是不能被使用（读取）的，此约束对后面将要介绍的所有命令都有效，因此后面就不再重复申明此约束了。

3.4 数据运算命令

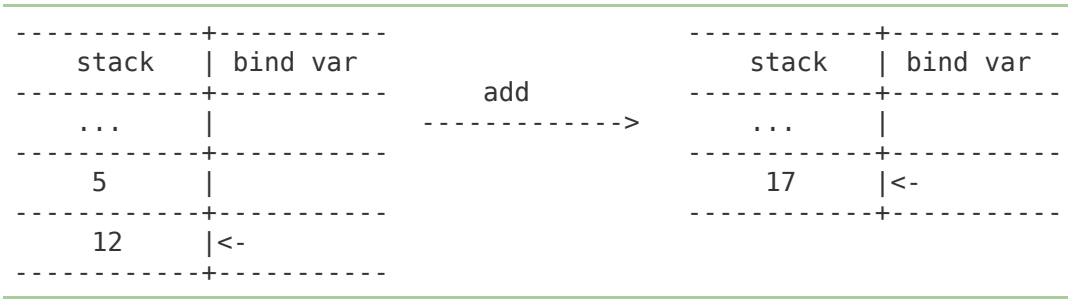
add / sub / mul / div / mod / cmpeq / cmpne / cmpgt / cmplt / cmpge / cmple / and / or / not / neg 命令，包括算术、比较和逻辑运算命令。对应于 TinyC 中的以下运算符：

```
+, -, *, /, %, ==, !=, >, <, >=, <=, &&, ||, !, -
```

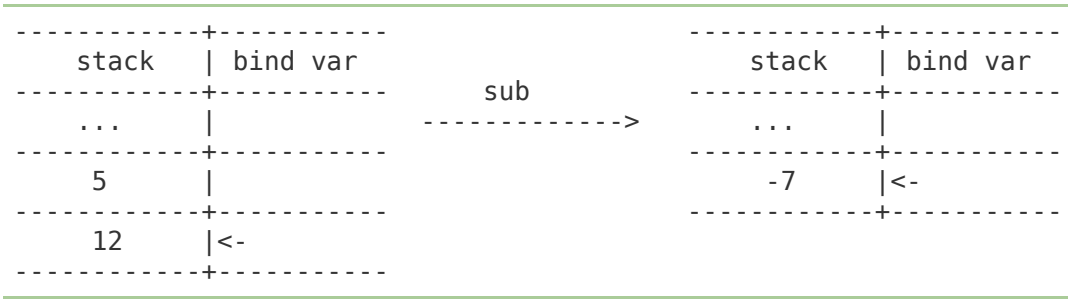
注意最后一个 “ - ” 是反号的意思，应和减号区别开来。

以上命令中，除 not 和 neg 命令外，其余命令均为二元操作命令，先取出栈顶两个元素，进行运算后，再将结果放回栈顶， not 和 neg 命令则为一元操作命令，只对栈顶一个元素进行操作。所有二元操作中， 原栈顶元素是第二个操作符 。

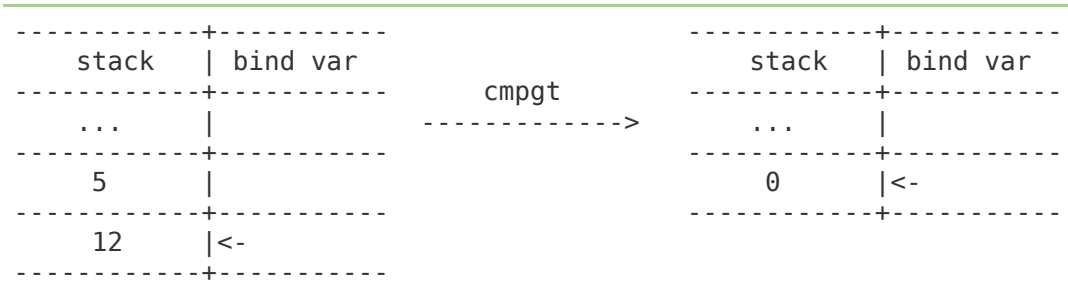
“ add ” 命令运行后栈的变化如下：



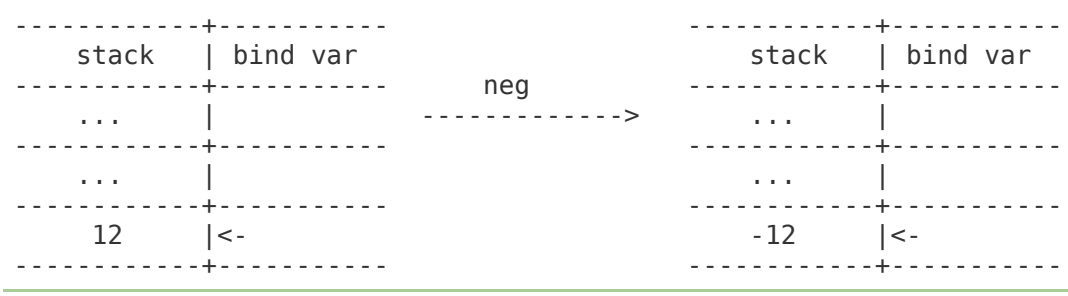
“ sub ” 命令运行后栈的变化如下，注意， 原栈顶元素是第二个操作符 ，最后的结果是 **5 - 12** 。



“ cmpgt ” 命令运行后栈的变化如下，注意原栈顶元素是第二个操作符，最后的结果是 5 > 12 ，因此是 0 （非真）。



“ neg ” 命令运行后栈的变化如下，栈顶位置不变，栈顶元素被反号。



数据运算命令和入栈、出栈命令组合，即可实现简单的表达式求值。如：

```
a = 1 + 2 * 3;
b = 8 - 5;
```

可以翻译成以下 Pcode：

```
push 1          ; a = 1 + 2 * 3;
push 2
push 3
mul
add
pop a

push 8          ; b = 8 - 5;
push 5
sub
pop b
```

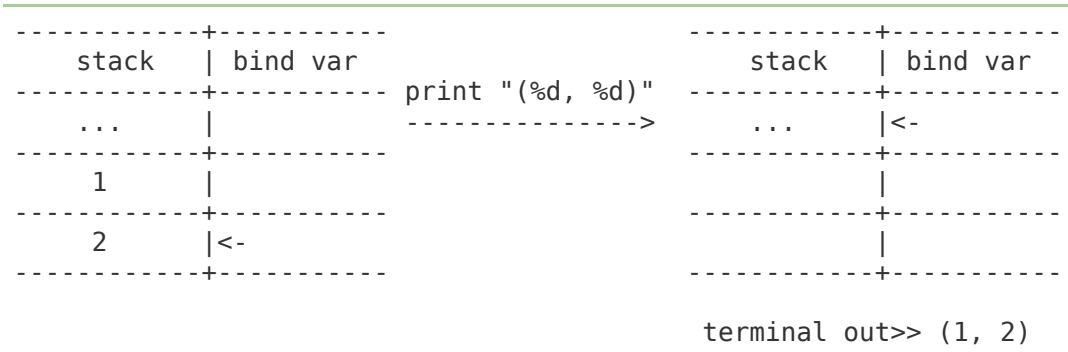
注意表达式中的元素的入栈顺序为 从左向右入栈 ，这样的顺序和人的阅读顺序是一致的。

3.5 输入及输出命令

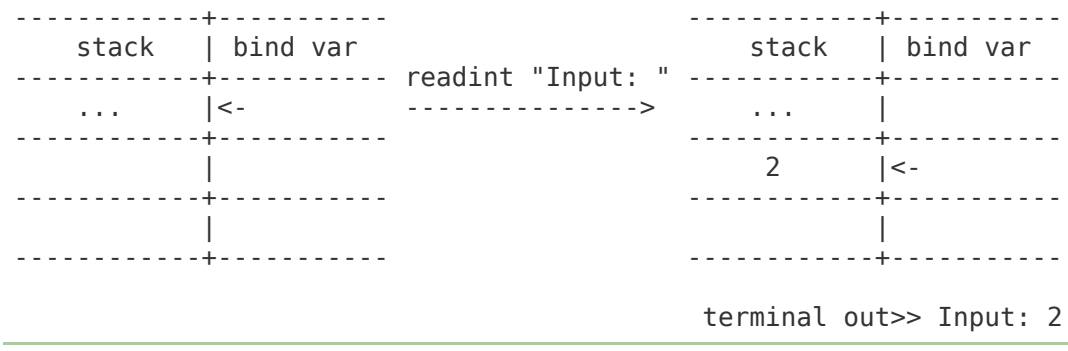
print / readint 命令，用法如下：

```
print "Hello world"         ; 输出: Hello world

push 1
push 2                      ; 相当于 print("(%d, %d)", 1, 2);
print "(%d, %d)"            ; 输出: (1, 2)

readint "Input: "
pop x                       ; 相当于 x = readint("Input: ");
```

print 命令会根据字符串的 “ %d ” 依次将栈顶元素取出，并打印出来，也就是说，上面第二个例子中 print 命令之前入栈的两个参数 1 和 2 ，在 print 后都将出栈。另外注意：参数的入栈的顺序需要从左向右入栈。



readint 命令先打印提示信息，再从标准输入中读取一个整数，返回后将其放入栈顶。



3.6 退出命令

exit 命令，退出虚拟机的运行，并设置退出码，有以下用法：

```
exit 0      ; 退出码为 0
exit a      ; 退出码为 a 的值
exit ~      ; 退出码为栈顶元素的值
```

上面的代码中用 “ ~ ” 来代表栈顶，这将是 Pcode 中的一个约定。

第 **3** 章完