**prog4翻译**

Page1

本周，你将完成一个简单的后缀式计算器，一个Clac编程语言的翻译器（后缀表达式翻译器）。

..

Testing

我们将通过运行一些来自test\_prog(clac\_program,initial\_stack,fianl\_stack,result)中的测试用例来测试你的Clac实现。

我们将检查四方面：

1.你的代码必须遵守lib文件夹中的库接口。我们将编译你的代码看看你对队和栈的不同实现，所以你必须且只能用我们给的函数和类型。

2.当输入内容有效时，你的解释器必须运行正常且输出正确的栈。

3.当输入内容无效时，你的解释器必须停止且提示用户输入了无效的内容。

4.当我们不加-d编译的时候，所有的运算要有好的线性运行时间。（这意味着除了skip和pick所有的运算要有常量运行时间。skip　n个元素或复制第n个元素到栈时间复杂度应该是o(n)。

clac-test.c0文件包含了如何去写和运行测试的例子，并且README.txt文件解释的怎么去编译怎么去运行测试。

Page2

...

# 1、‘计算器’简要介绍（就是逆波兰表达式）

Clac是一个新的基于栈的编程语言（由RPS研究发展）。任何关于Clac和Forth或PostScript的相似之处纯属巧合。在这次作业的第一部分，我们将要实现‘计算器’的核心特征，在Task2中，我们将要在其中加入一些更好玩的特性。

Clac就像是一个交互式计算器。当它运行的时候，它将保持一个”运算数栈”。输入一些数字将会简单的将这些数压入到”运算数栈”内。当一个运算符（比如+或者是×等）被输入时（原文是遇到），这个运算符会被作用于”运算数栈”顶的两个数，并且会把运算结果重新压入到”运算数栈”中，当输入一个换行符的时候会输出栈顶的一个元素，比如：我们输入 3 4 + 并且紧接着输入一个换行符

% ./claculator

Clac top level

clac>> 3 4 +

7

Clac输出了一个 7 ，这个7现在在栈顶（要不然栈为空）。

我们现在输入-9 2 / 外加一个换行符，Clac输出一个 -4。

clac>> -9 2 /

-4

这时，栈中包含了一个7和一个-4（-4在栈顶），当我们再输入一个 - 和一个换行符时，

clac>> -

11

我们得到了11，因为7-（-4）= 11。我们可以通过输入 quit 来退出。

clac>> quit

11

Bye!

其实我们可以在同一行中输入多个输入。如：

% ./claculator

Clac top level

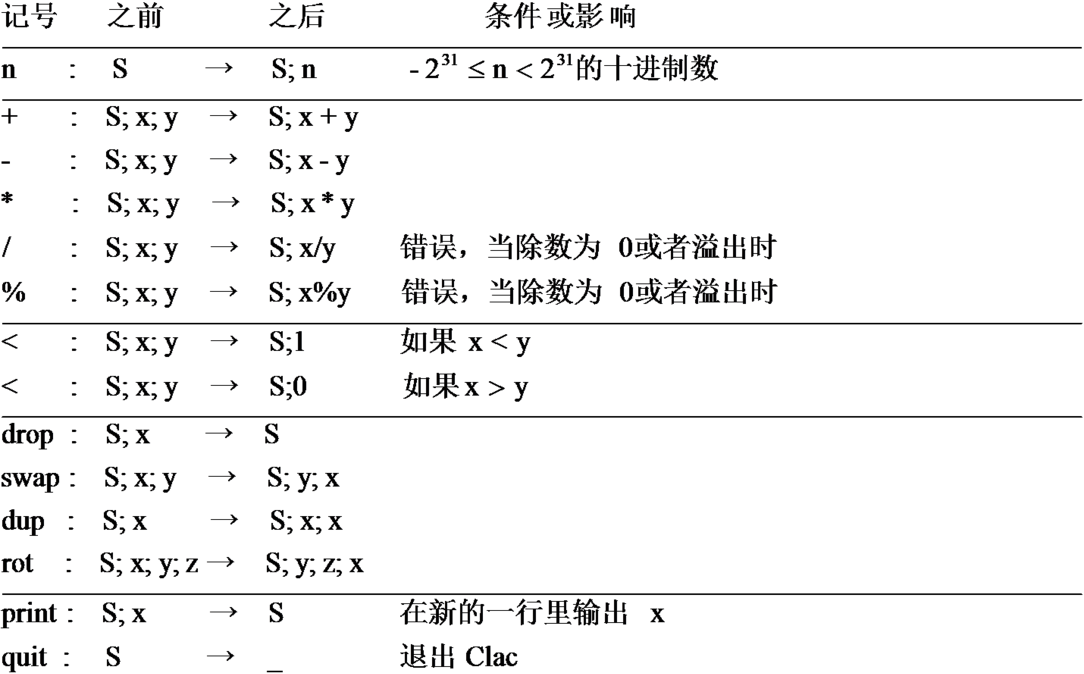
clac>> 11 10 2 9 - + \*

33

请确保你明白了为什么会得到33。（（（２－９）＋１０）×11）

page3

除了用这种算法计算，这里还有几种特殊的计算方式你可以去执行。下面是张你可以在任务1中执行的完整的计算的表。为了详细说明这个计算，我们用S→S'来表示堆S变成了堆S'。在正确结果时，堆和堆的最上面的元素一起写出来！例如，减法可以被陈述为-：S，x，y→S，x-y，这个意味着：“弹出最上面的元素（y）和下一个来自堆的元素（x），从x里面减去y，然后把结果x-y返回到堆里面。”我们写出S用上面的规则意味着在堆里可以有许多其他整数不被计算影响。



你的实现过程应该明确觉察以下情形用一个恰当的错误消息到然后调用错误函数，发出一个错误信号。

1. 记号是不合法的，就是不在任何前面出现的一个表内。记号是大小写敏感的。例如，DUP和Dup是未定义的，而dup可以复制堆最上面的元素。
2. 堆里面没有足够的元素去执行一个运算。例如，当堆里面少于三个元素时调用rot，将出错。
3. 求商或者求模运算时第二个操作数是0，或者求int\_min（）被-1所除的商时，会产生一个溢出，根据c0的定义（见c0参考书的第三面）。这是一个32位二进制的语言，所以，加法，减法，还有乘法就会像在c0里一样不会产生任何溢出错误。

page4

用户的错误(代码错误)应该总是使函数error被调用，而不是使断言assertation失败。文件clac.c0的开始代码中就有一个例子。

## 任务1(10分) 根据上方的描述，扩写clac.c0中的函数。不要改变任何在这个文件中的#use标记,并确保你的函数满足以下声明:

bool eval(queue Q, stack S)

//@ensures \result == false || queue\_empty(Q);

你可以自由地改变这个文件中其他的一切。接下来说明eval的参数和返回值：

文件clac-main.c0中的主函数main和clac-test.c0中的test\_prog函数均会将一行输入转换为由*标记组成的队列*。每个标记只是一个字符串。clac中的这部分功能已经为你写好，欢迎你检查它，但你不需要改变此代码。在clac中，标记只能用空格隔开。例如，3 4+会被解读为两个标记（“3”然后是“4+”），因此这时将产生一个错误，因为标记“4+”没有被定义。

整数堆栈S的初始状态将为空。但是由于输入是一行一行处理的 ，eval函数还应该可以在堆栈非空时被调用，非空栈中存有之前计算得到的值。

eval函数应该从队列Q中列出标记，并根据clac的定义处理它们。当队列Q为空时，eval应该返回true，而此时堆栈S可以处于任何状态。遇到指令“quit”时，eval应该返回false，提示这时主函数main应退出。

你可以在文件lib / queues\_string.c0和lib / stacks\_int.c0中找到堆、栈相关的函数的接口（和讲义9和10中的代码是一样的）。

**参考**：如果你想看一下样例的结果，你可以运行Clac on Andrew的一个参考程序。为了避免混淆，我们调用可执行的clac-ref

% clac-ref

Clac top level

clac>> 4 7 - 2 \*

-6

clac>> quit

-6

Bye!

Page5

你也可以运行cal-ref -trace,这将会使参考程序输出计算的所有中间步骤，例如：

% clac-ref -trace

Clac top level

clac>> 4 7 - 2 \*

stack || queue

|| 4 7 - 2 \*

4 || 7 - 2 \*

4 7 || - 2 \*

-3 || 2 \*

-3 2 || \*

-6 ||

-6

clac>>

这里左边一列显示当前的堆栈，右边一列显示当前的队列，紧接着的是返回的堆栈（没有在上面的例子中显示因为它是空的）

# 2、操作堆栈和队列

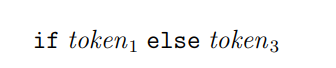
在作业的这部分中，我们将会在Clac中进一步实现一些操作堆栈和队列的功能。第一个功能**skip**选取堆栈顶部的数字n(n必须非负），然后从队列中移走n个标记——如果n是负数，或者在当前队列中没有足够的标记，则产生错误（error）。

你也可以实现两个表现的很像skip的特殊标记:

**if**：跳过2或者0个标记，取决于堆栈顶部是不是0；

**else**：和“1 skip”的功能完全相同；

这两个名字看起来可能有点奇怪，但是这两个标记可以一起让我们实现这个习语：



这里0代表false,任意的非零值代表true,下面这个例子阐明了这一点：

clac>> 6 1 if 2 else -2 +

8

clac>> 6 0 if 2 else -2 +

4

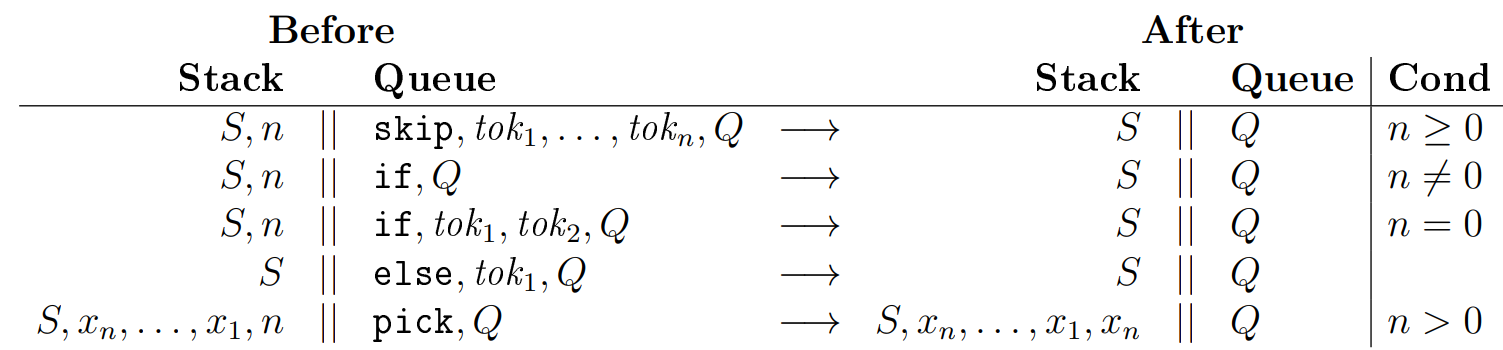
第一行计算得8：当遇到if的时候，6和1都在堆栈中，然后我们push进去2，在这个过程中消耗掉了1，然后我们看到else并跳过下一个记号-2，最后我们把6和2相加，得到了8，它被push进堆栈中，并被打印出来。

在下一行，当遇到if的时候，0在堆栈的顶部，所以我们跳过下两个标记，也就是2和else。然后-2被推进栈中，6和（-2）相加得4，它被push进堆栈中，并被打印出来。

Page6

说明一点：if 和 else 这两个标记不必同时出现。他们应该作为相互独立的clac功能被实现。

我们对Clac的最后一个添加是 **pick**这个标记。如果正数n出现在栈顶，”pick“应该复制在栈里的第n个元素并放在栈顶。如果n不是一个正数或者栈中没有n个元素将导致错误。下面的表概括了我们在这一部分任务中将实现的功能。



## 任务2 （5分）：根据上述说明，在clac.c0中扩展eval，以处理skip，if，else和pick。

# 3　附加题：Clacworks

我们已经解释了10个标记的功能：drop,swap,dup,rot,print,quit,skip,if,else,pick.只要你不管用数字和这十个标记编程的意义，你就能扩展Clac更多的特性，并且利用这些新的标记来写程序。

## TASK 3(奖励分)　通过定义新的标记来扩展你的Clac（增加新的功能）。在bonus.clac中写一个牛逼的程序，并在bonus.txt中描述你的新功能的作用和你的Clac程序。

**感谢：尚骏远,杨凯航，涂超，王敬博，闫航宇，聂之贺对prog4的翻译。**

**声明：本文翻译只做参考作用，产生的后果由读者自负，欢迎对本文翻译不准确或错误之处的修改。**