Lab 1: Evaluation Using Reverse Polish Notation

DD2325 Applied Programming and Computer Science

October 31, 2007

1 Introduction

In this lab, you will program a mini-calculator using *reverse polish notation* (RPN) as an internal representation. RPN is a way of writing an *expression*, which is particularly handy when evaluating the expression. The process of evaluating an expression in RPN illustrates the usefulness of the *stack* as a data structure.

An expression is a sequence of symbols. In this lab, we will be dealing with arithmetic expressions. Then, the symbols used in expressions are digits, the operators +, -, *, / and the left and right-paranthesis.

In *infix* form, the operators are placed between their operands and the paranthesis are used to clarify in which order the operators will be applied. In this notation, a precedence relation is specified where * and / have higher priority then + and -; in the case of equal precedence, the operators applied from left to right.

Examples: 2 + 3 , 4 * (2 + 3), ((3 + 5 * 1) / 8) * 14

In the reverse polish form (also called *postfix*), operators are placed after their operands. In this notation, paranthesis are not needed:

Examples: 2 3 + , 4 2 3 + * , 3 5 1 * + 8 / 14 *

2 Algorithm1: Conversion from Infix to RPN (Postfix)

We use the function p for specifying the priorities of the operators:

op	$+$	$-$	$*$	$/$
$p(op)$	1	1	2	2

Given an infix expression $U_1 \ldots U_m$, where U_i is either an operand(in this case an integer value), an operator or a paranthes, the algorithm below outputs the expression in RPN. The algorithm makes use of one stack.

```
for i=1 to m
     if U_i is an operand:        Transfer U_i to output
     if U_i is a left paranthes:  Push U_i to stack
     if U_i is a right paranthes: Pop symbols from stack and transfer
                                  them to output until a left-paranthes
                                  is met. Pop left-paranthes.
     if U_i is an operator:       Let the top stack element be t. Pop and
                                  transfer symbols from stack to output
                                  until:
                                     p(t) < p(U_i) or
                                     t is a left-paranthes or
                                     the stack is empty.
                                  Push U_i to the stack

Transfer the remaining symbols in stack to output
```

2.1 Example

Infix	Stack (Top to the right)	RPN
$((3+5*1)/8)*14$	ϵ	ϵ
$(3+5*1)/8)*14$	(	ϵ
$+5*1)/8)*14$	((	3
$5*1)/8)*14$	((+	3
$*1)/8)*14$	((+	3 5
$1)/8)*14$	$((+*$	3 5
$)/8)*14$	$((+*$	$3\,5\,1\ *\ +$
$/8)*14$	(	$3\,5\,1\ *\ +$
$8)*14$	(/	$3\,5\,1\ *\ +$
$)*14$	(/	$3\,5\,1\ *\ +8$
$*14$	ϵ	$3\,5\,1\ *\ +8\,/$
14	$*$	$3\,5\,1\ *\ +8\,/$
ϵ	$*$	$3\,5\,1\ *\ +8\,/\,14$
ϵ	ϵ	$3\,5\,1\ *\ +8\,/\,14\,*$

3 Algorithm2: Evaluating an Expression in RPN

Given a postfix expression $V_1 \ldots V_n$ where V_i is either an operand, an operator or a paranthes, the following algorithm evaluates the expression.

```
i = 1
while i<= n
     if V_i is an operand:        Push V_i to stack
     if V_i is an operator:       Apply V_i to the top two elements of the
                                  stack. Replace these by the result in the
                                  stack.

     i = i + 1

Output result from stack
```

3.1 Example

Evaluation of 3 5 1 * + 8 / 14 *

Stack
ϵ
3
3 5
3 5 1
3 5
8
8 8
1
1 14
14

4 The Task

You will program a calculator in MATLAB, which takes an (infix) arithmetic expression of the type above and first converts this to RPN and then using the RPN expression evaluates it. Algorithms 1 and 2 will be implemented for this purpose. The input, output, as well as the stack will be strings in MATLAB. You can assume that the input is a well-formed expression.

4.1 An Example Run

```
> rpn_calc('((3+5*1)/8)*14')
        In RPN notation: 3 5 1 * + 8 / 14 *

        Result of evaluation: 14
```

4.2 Groups

You can work in groups of at most 2.

4.3 Demos

Demonstrations will be done during lab hours. Please make sure you can explain every part of your program.