# SIMPLICITY = EFFICIENCY = READABILITY

## A Simple INFIX to PREFIX algorithm.

Cohen Shimon
Computer Science Department
Hebrew University, Jerusalem, Israel.

February 1981

## ABSTRACT

The following is not the expected breakthrough in the methodology of computer programming. It is just a small contribution to this troubled topic. It proposes an improved (simple) version for infix expression evaluation. The main idea behind this algorithm is it's simplicity which surprisingly leads to a (very) flexible, efficient, readable, small (I dare say pure) program.
The simple purpose of this small paper is to promote the ideal of small and simple programs.

## INTRODUCTION

It is my belief (and many others) that simplicity is (most of the time) a basic and effective programming methodology. It has many consequences: Short code, readability, efficient code, flexibility. Knuth in his well known paper "Structured programming with GOTO statement" explained how easy it is to ruin a program "for the sake of efficiency". And it is true since many programmers tend to justify their "mixed-up" code with the famous claim: "But I want it to work fast" As it turns out (in many cases) because of constant improvment in the program (debug) and in the requirements (analysis) the program becomes messy unreadable and in most cases inefficient. It was well put: " Let us leave the efficiency problem to the compiler".

We choose to demonstrate this approach with a new algorithm to execute (translate) infix numeric expression. It is like the traditional method where we keep everything on a stack but instead of managing the stack ourselves we use the language recursion mechanism (PASCAL or LISP).

The following is a description of a flexible calculator which can be easily extended to deal with many operators. It supports left and right priorities and unlimited nested paranthesis. Unary operators can come in any number before an expression
like: ---1
The algorithm is an improved version in comparision with the conventional methods for arithmetic expression evaluation. The EVAL function which is the basis for all this computation is short and uses a simple method. This program should explain itself both when you read it (structured programming) and when you are using it (see the example). Pay attention to options 1,2,3 in the text itself.

## SOME EXPLANATION (of the algorithm)

We use the following abbreviations:
(expr == expression, var == variable, opr == operator)

## SIMPLICITY = EFFICIENCY = READABILITY

We look at the expression as follows:

<expr> ::= <basic expr> <oper 1> <rest expr>

where:

<basic expr> ::= constant | var | ( <expr> ) | <unary opr> <basic expr>

It is true that in all expression we find some basic staff in the begining:
a, (12 + 13) or -5, -a
And now to <rest expr>

<rest expr> ::= <expr 1> <opr 2> <expr 2>

Why ??? <opr 2> is the first operator with priority less or equal to the pri
ity of <opr 1>. So what we have to do is first to compute the value of <ba
expr> then to compute <expr 1> and in the end to apply <opr 1> to the value
the <basic expr> and the value of <expr 1>. Now we have a new value on the l
side and we can say that we have: <value> <opr 2> <expr 2>. It is very m
like the definition of <expr>. So we braek down <expr 2> the way we did w
<rest expr> and continue the evaluation.

The overall view of <expr> is therefore:

```
 <expr> ::= <basic expr> <opr 1> <expr 1> <opr 2> <expr 2>

            <           value            >
```

The EVAL function knows to evaluate expression with operators whose prio
ties are greater (not equal) to LVL (the parameter of EVAL). We assume that
the operators have positive (non-zero) priorities. The first call to is EVAL
which means to eval all the expression. (since all the operators their prio
ties is bigger than zero) Later we use EVAL to eval <expr 1>. The recurs
mechnism keeps those values and operators which have to wait for higher orde
computations.

Note that in this method we scan the expression one time from left to right.

The simple structure of the EVAL functions make it easy to add new ope
tors by adding the appropriate statements in functions 'initcalc' and 'execu
or 'exec1'.

## SIMPLICITY = EFFICIENCY = READABILITY

PARTIAL LISTING

```
procedure initcalc ;
   (* In the beginning it make sense *)
begin
     for i:= 0 to 127 do
       begin
            vars[chr(i)] := 0 ;
            priorleft[chr(i)] := 0 ;
            priorright[chr(i)] := 0 ;
       end ;
     priorleft ['+']:= 3 ; priorleft ['-'] := 4 ;
     priorright['+']:= 3 ; priorright['-'] := 4 ;
     priorleft ['*']:= 7 ; priorleft ['/'] := 6 ;
     priorright['*']:= 7 ; priorright['/'] := 6 ;
     binoprs := [')','.','+','-','*','/'] ;
end (* init *) ;

function execute (op : char ; x,y : integer ) :integer ;
var execresult : integer ;
begin
                  (* Here you can add new operators *)
      case op of
             '+' : execresult := x+y ;
             '-' : execresult := x-y ;
             '*' : execresult := x*y ;
             '/' : execresult := x div y ;
      end ;
         (* start debug *)
         if vars['3'] <> 0 then begin
           write (' in execute     ---> ');
           write (op:3,x:5,y:5);
           writeln('  --> result: ',execresult:1);
         end;
         (* end debug *)
      execute := execresult ;
end;

function getnum : integer ;
(* A function for every kid in town *)
var  num : integer ;
begin
   num := 0 ;
   while inputline[i] in ['0'..'9'] do
      begin
         num := num * 10 + ord (inputline[i] ) - ord ('0') ;
         i := i + 1 ;
      end ;
   getnum := num ;
end ;

function eval (lvl : integer ) : integer ;
var opnow : char ;
     loprand , roprand : integer ;
function exec1 (op1 : char ; oprand : integer ) : integer ;
(* For unary operators *)
begin
   case op1 of
        '+' : exec1 := oprand ;
        '-' : exec1 := - oprand ;
   end ;
end ;

function evalbasic : integer ;
var   c : char ;
begin
   c := inputline[i] ;
   (* start debug *)
```

SIMPLICITY = EFFICIENCY = READABILITY

```
   if vars['1'] <> 0 then begin
      write (' in eval basic ---> ');
      writell ;
      write (' level is:',lvl:4,' ---> ');
      writeln('1':i);
   end;
   (* end debug *)
        if c in ['a' .. 'z' ] then
            begin
                 evalbasic := vars[c]  ;
                 i := i + 1 ;
            end
   else if c in ['+' ,  '-' ] then
            begin
                 i := i + 1 ; (* recursive call *)
                 evalbasic := exec1 (c,evalbasic) ;
            end
   else if c in ['0' .. '9' ] then
            begin
                 evalbasic := getnum ;
            end
   else if c =  '('           then
            begin
                i := i + 1 ;
                evalbasic := eval (0) ;
                i := i + 1
            end
   else error(1) ;
end ;

begin (*   of function EVAL *)
     loprand := evalbasic ;
     opnow := inputline [i] ;
     if not ( opnow in binoprs ) then error(2) ;
     while priorleft[opnow] > lvl do
       begin
        (* start debug *)
        if vars['2'] <> 0 then begin
           write (' in eval binary---> ');
           writell ;
           write (' level is:',lvl:4,' ---> ');
           writeln('.':i);
        end;
        (* end debug *)
         i := i + 1 ;
         roprand := eval (priorright[opnow] ) ;
         loprand := execute(opnow,loprand,roprand) ;
         opnow := inputline[i] ;
         if not ( opnow in binoprs ) then error(2) ;
       end ;
      eval := loprand ;
end ;
```

## EXAMPLE RUN

The following is an example run with the above pascal program. User input follows the ENTER...... lines.

```
 ) >- priorities ->  left: 0  right: 0
 * >- priorities ->  left: 7  right: 7
 + >- priorities ->  left: 3  right: 3
 - >- priorities ->  left: 4  right: 4
 . >- priorities ->  left: 0  right: 0
 / >- priorities ->  left: 6  right: 6

 A=>   0  B=>   0  C=>   0  D=>   0  E=>   0  F=>   0  G=>   0
 H=>   0  I=>   0  J=>   0  K=>   0  L=>   0  M=>   0  N=>   0
 O=>   0  P=>   0  Q=>   0  R=>   0  S=>   0  T=>   0  U=>   0
 V=>   0  W=>   0  X=>   0  Y=>   0  Z=>   0
 a=>   0  b=>   0  c=>   0  d=>   0  e=>   0  f=>   0  g=>   0
 h=>   0  i=>   0  j=>   0  k=>   0  l=>   0  m=>   0  n=>   0
 o=>   0  p=>   0  q=>   0  r=>   0  s=>   0  t=>   0  u=>   0
 v=>   0  w=>   0  x=>   0  y=>   0  z=>   0
(* In the following lines we turn ON the debuging flags *)
ENTER......1=1.
value of 1 is :1
ENTER......2=2.
in eval basic ---> 2=2.
level is:   0 --->   1
value of 2 is :2
ENTER......3=3.
in eval basic ---> 3=3.
level is:   0 --->   1
value of 3 is :3
ENTER......a=1+2+3.
in eval basic ---> a=1+2+3.
level is:   0 --->   1
in eval binary---> a=1+2+3.
level is:   0 --->    .
in eval basic ---> a=1+2+3.
level is:   3 --->     1
in execute    --->   +    1    2  --> result: 3
in eval binary---> a=1+2+3.
level is:   0 --->      .
in eval basic ---> a=1+2+3.
level is:   3 --->       1
in execute    --->   +    3    3  --> result: 6
value of a is :6
ENTER......b=---3.
in eval basic ---> b=---3.
level is:   0 --->   1
in eval basic ---> b=---3.
level is:   0 --->    1
in eval basic ---> b=---3.
level is:   0 --->     1
in eval basic ---> b=---3.
level is:   0 --->      1
value of b is :-3
ENTER......c=-+-+-121.
in eval basic ---> c=-+-+-121.
level is:   0 --->   1
in eval basic ---> c=-+-+-121.
level is:   0 --->    1
in eval basic ---> c=-+-+-121.
level is:   0 --->     1
in eval basic ---> c=-+-+-121.
level is:   0 --->      1
in eval basic ---> c=-+-+-121.
level is:   0 --->       1
in eval basic ---> c=-+-+-121.
level is:   0 --->        1
value of c is :-121
```

SIMPLICITY = EFFICIENCY = READABILITY

```
ENTER......d=2+3*4.
in eval basic ---> d=2+3*4.
level is:   0 --->   1
in eval binary---> d=2+3*4.
level is:   0 --->    .
in eval basic ---> d=2+3*4.
level is:   3 --->     1
in eval binary---> d=2+3*4.
level is:   3 --->      .
in eval basic ---> d=2+3*4.
level is:   7 --->       1
in execute    --->   *    3    4  --> result: 12
in execute    --->   +    2   12  --> result: 14
value of d is :14
(* Now we change the priority of +    watch the difference *)
ENTER......+8,8.
priority of + changed to 8 <-> 8
ENTER......e=2+3*4.
in eval basic ---> e=2+3*4.
level is:   0 --->   1
in eval binary---> e=2+3*4.
level is:   0 --->    .
in eval basic ---> e=2+3*4.
level is:   8 --->     1
in execute    --->   +    2    3  --> result: 5
in eval binary---> e=2+3*4.
level is:   0 --->      .
in eval basic ---> e=2+3*4.
level is:   7 --->       1
in execute    --->   *    5    4  --> result: 20
value of e is :20
ENTER......?
 ) >- priorities ->  left: 0  right: 0
 * >- priorities ->  left: 7  right: 7
 + >- priorities ->  left: 8  right: 8
 - >- priorities ->  left: 4  right: 4
 . >- priorities ->  left: 0  right: 0
 / >- priorities ->  left: 6  right: 6

 A=>   0  B=>   0  C=>   0  D=>   0  E=>   0  F=>   0  G=>   0
 H=>   0  I=>   0  J=>   0  K=>   0  L=>   0  M=>   0  N=>   0
 O=>   0  P=>   0  Q=>   0  R=>   0  S=>   0  T=>   0  U=>   0
 V=>   0  W=>   0  X=>   0  Y=>   0  Z=>   0
 a=>   6  b=>  -3  c=>-121  d=>  14  e=>  20  f=>   0  g=>   0
 h=>   0  i=>   0  j=>   0  k=>   0  l=>   0  m=>   0  n=>   0
 o=>   0  p=>   0  q=>   0  r=>   0  s=>   0  t=>   0  u=>   0
 v=>   0  w=>   0  x=>   0  y=>   0  z=>   0
ENTER......x=(((((9*7)-8)---1))).
in eval basic ---> x=(((((9*7)-8)---1))).
level is:   0 --->   1
in eval basic ---> x=(((((9*7)-8)---1))).
level is:   0 --->    1
in eval basic ---> x=(((((9*7)-8)---1))).
level is:   0 --->     1
in eval basic ---> x=(((((9*7)-8)---1))).
level is:   0 --->      1
in eval basic ---> x=(((((9*7)-8)---1))).
level is:   0 --->       1
in eval basic ---> x=(((((9*7)-8)---1))).
level is:   0 --->        1
in eval binary---> x=(((((9*7)-8)---1))).
level is:   0 --->         .
in eval basic ---> x=(((((9*7)-8)---1))).
level is:   7 --->          1
in execute    --->   *    9    7  --> result: 63
in eval binary---> x=(((((9*7)-8)---1))).
level is:   0 --->            .
in eval basic ---> x=(((((9*7)-8)---1))).
level is:   4 --->             1
in execute    --->   -   63    8  --> result: 55
```

SIMPLICITY = EFFICIENCY = READABILITY

```
in eval binary---> x=(((((9*7)-8)---1))).
level is:   0 --->               .
in eval basic ---> x=(((((9*7)-8)---1))).
level is:   4 --->                1
in eval basic ---> x=(((((9*7)-8)---1))).
level is:   4 --->                 1
in eval basic ---> x=(((((9*7)-8)---1))).
level is:   4 --->                  1
in execute    --->   -   55    1  --> result: 54
value of x is :54
ENTER......f=120/12/4.
in eval basic ---> f=120/12/4.
level is:   0 --->   1
in eval binary---> f=120/12/4.
level is:   0 --->      .
in eval basic ---> f=120/12/4.
level is:   6 --->       1
in execute    --->   /  120   12  --> result: 10
in eval binary---> f=120/12/4.
level is:   0 --->         .
in eval basic ---> f=120/12/4.
level is:   6 --->          1
in execute    --->   /   10    4  --> result: 2
value of f is :2
(* Now we will have different priorities for left and right *)
ENTER....../8,7.
priority of / changed to 8 <-> 7
ENTER......g=120/12/4.
in eval basic ---> g=120/12/4.
level is:   0 --->   1
in eval binary---> g=120/12/4.
level is:   0 --->      .
in eval basic ---> g=120/12/4.
level is:   7 --->       1
in eval binary---> g=120/12/4.
level is:   7 --->         .
in eval basic ---> g=120/12/4.
level is:   7 --->          1
in execute    --->   /   12    4  --> result: 3
in execute    --->   /  120    3  --> result: 40
value of g is :40
ENTER......?
 ) >- priorities ->  left: 0  right: 0
 * >- priorities ->  left: 7  right: 7
 + >- priorities ->  left: 8  right: 8
 - >- priorities ->  left: 4  right: 4
 . >- priorities ->  left: 0  right: 0
 / >- priorities ->  left: 8  right: 7

 A=>   0  B=>   0  C=>   0  D=>   0  E=>   0  F=>   0  G=>   0
 H=>   0  I=>   0  J=>   0  K=>   0  L=>   0  M=>   0  N=>   0
 O=>   0  P=>   0  Q=>   0  R=>   0  S=>   0  T=>   0  U=>   0
 V=>   0  W=>   0  X=>   0  Y=>   0  Z=>   0
 a=>   6  b=>  -3  c=>-121  d=>  14  e=>  20  f=>   2  g=>  40
 h=>   0  i=>   0  j=>   0  k=>   0  l=>   0  m=>   0  n=>   0
 o=>   0  p=>   0  q=>   0  r=>   0  s=>   0  t=>   0  u=>   0
 v=>   0  w=>   0  x=>  54  y=>   0  z=>   0
ENTER......!
(* And in the end the love you get is equal to the love you gave *)
                                               The BEATELS       *)
(*
```