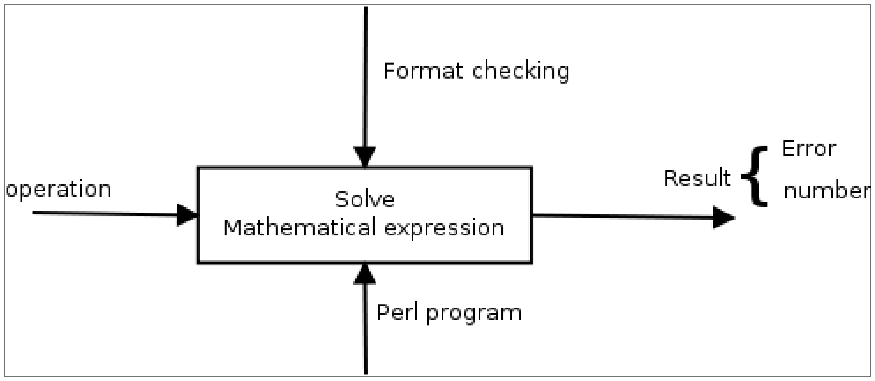
A basic calculator with the algorithm of operator precedence

Introduction
To calculate a basic expression we need a basic algorithm that parse a string and manage operator priority.

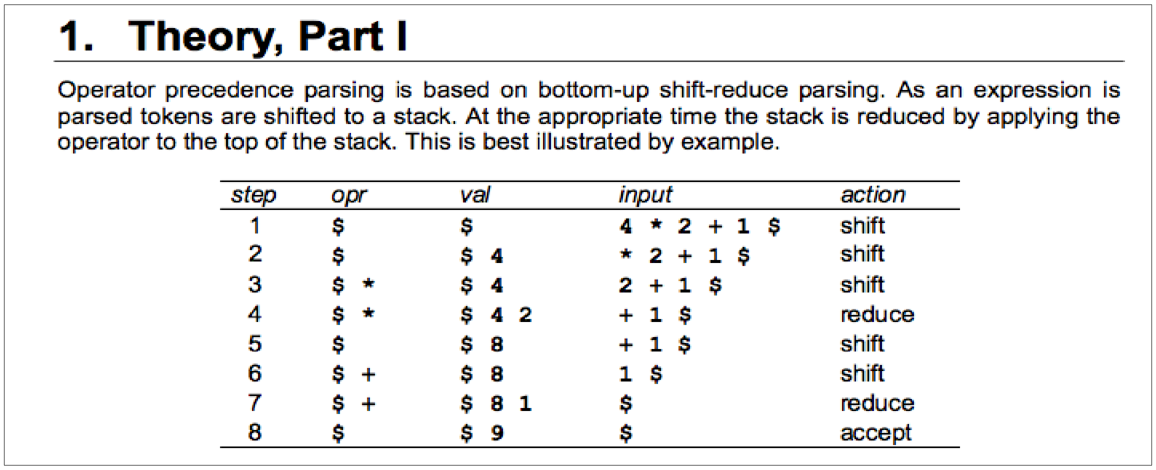
Architecture with SADT[1]
First version of the algorithm does not take care about priority with parenthesis (not managed yet). We put only a basic operation and it calculates the expression. We do not not take care if the expression is well formed or not (syntax, grammar included). Second version of the algorithm it is. Parenthesis not managed yet. The SADT diagram below take care about include the second version. Yet we don't dive into neaty greety of the algorithm. The result for the second algorithm is the result of the operation or, the error message associated to operation.



Regular expression and equation (a.k.a operation)
SADT description

Example
We want to calculate the expression 4*2+1. This example does not take care about how well formed is the mathematical expression. To do that we need two stacks one for the operand and, one for the operator. Watch the picture below[2]:

1Structured Analysis and Design Technique(SADT)
2See THOMAS NIEMANN Portland, Oregon web site: epaperpress.com



1. Theory, Part I

Operator precedence parsing is based on bottom-up shift-reduce parsing. As an expression is parsed tokens are shifted to a stack. At the appropriate time the stack is reduced by applying the operator to the top of the stack. This is best illustrated by example.

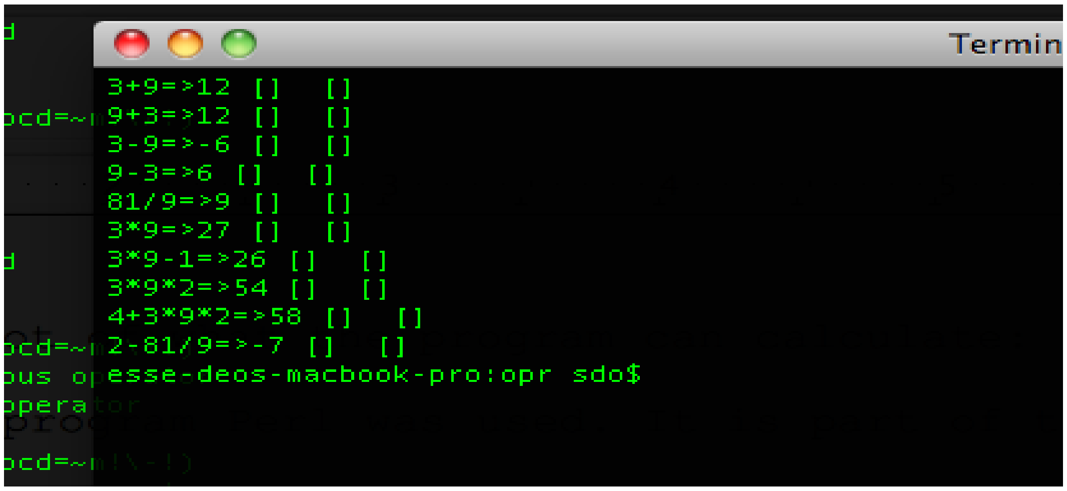
step	opr	val	input	action
1	$	$	4 * 2 + 1 $	shift
2	$	$ 4	* 2 + 1 $	shift
3	$ *	$ 4	2 + 1 $	shift
4	$ *	$ 4 2	+ 1 $	reduce
5	$	$ 8	+ 1 $	shift
6	$ +	$ 8	1 $	shift
7	$ +	$ 8 1	$	reduce
8	$	$ 9	$	accept

Example of a string parsed
with the bottom-up shift reduce parsing

200110503

The program first version

Here a screen shot of what the program can calculate:



```
3+9=>12 []   []
9+3=>12 []   []
3-9=>-6 []   []
9-3=>6 []   []
81/9=>9 []   []
3*9=>27 []   []
3*9-1=>26 []   []
3*9*2=>54 []   []
4+3*9*2=>58 []   []
2-81/9=>-7 []   []
esse-deos-macbook-pro:opr sdo$
```

Example of some operations

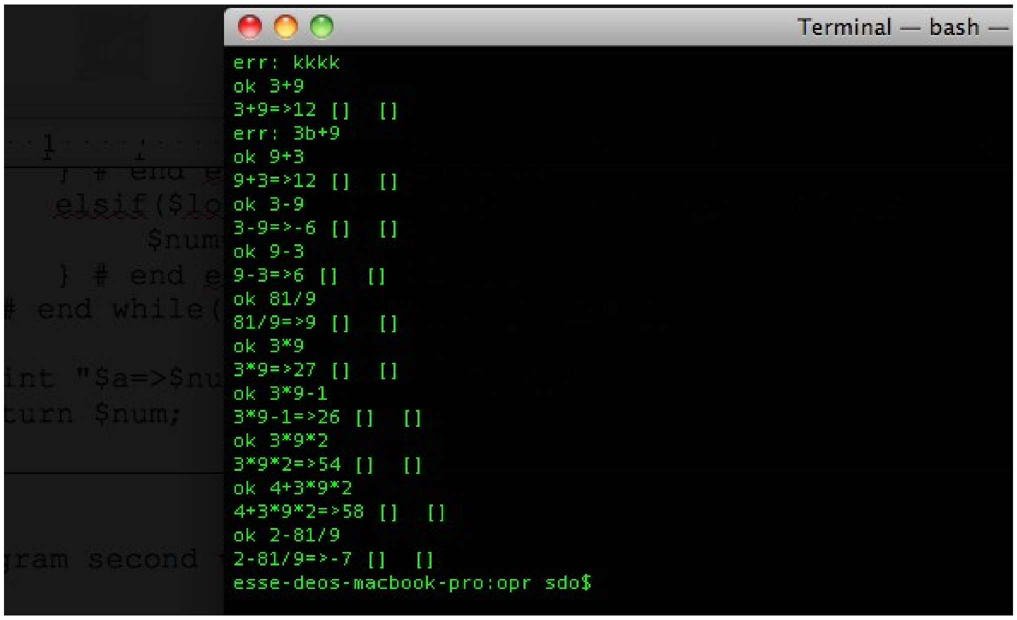
We can see we have two pairs of square brakets. That's the contents of the stacks when operations are over.

To realize this program Perl was used. It is part of the project so it feets well.

```
#!/usr/bin/perl

use strict;

reg("3+9");# string to analyze
reg("9+3");# string to analyze
reg("3-9");# string to analyze
reg("9-3");# string to analyze
reg("81/9");# string to analyze
reg("3*9");# string to analyze
reg("3*9-1");# string to analyze
reg("3*9*2");# string to analyze
reg("4+3*9*2");# string to analyze
reg("2-81/9");# string to analyze
#reg("3+9+2*3+8");# string to analyze
#reg("3*9+2*3+8");# string to analyze
#reg("3*9+2+3+2");# string to analyze
#reg("3*9+2-3+2");# string to analyze
#reg("3*9-2+3+2");# string to analyze
#reg("3*9-2-3+2");# string to analyze

sub reg{
     my ($a)=@_;
     my @opd=();# Stack for operand
     my @opt=();# Stack for operator
     my $pbeg=0;# Position at the begining
     my $pend=0;# Position at the end
     my $size=length($a);# size of string
     my $i=0;
     my $num=();
     my $c=();# current char

     while($i<$size){ # begin while($i<$size)
          $c=substr($a,$i,1);# gets one character
          if($c=~m!\+!){ # begin if($c=~m!\+!)
               push @opd,$num;# shift number
               if(scalar(@opt)==0){ # begin if(scalar(@opt)==0)
                    push @opt,$c;# shift operator
               } # end if(scalar(@opt)==0)
               else{ # begin else
                    my $locd=pop @opt;# unshift operator to check
precedency
                    if($locd=~m!\*!){ # begin if($locd=~m!\*!)
                         my $opt1=pop @opd; # reduce
                         my $opt2=pop @opd; # reduce
                         my $res=$opt1*$opt2;

                         push @opd,$res;# shift operand
                         push @opt,$c;# shift operator
                    } # end if($locd=~m!\*!)
                    elsif($locd=~m!\/!){ # begin elsif($locd=~m!\/!)
                         my $opt1=pop @opd; # reduce
                         my $opt2=pop @opd; # reduce
                         my $res=$opt2/$opt1;

                         push @opd,$res;# shift operand
                         push @opt,$c;# shift operator
                    } # end elsif($locd=~m!\/!)
                    elsif($locd=~m!\+!){ # begin elsif($locd=~m!\+!)
                         my $opt1=pop @opd; # reduce
                         my $opt2=pop @opd; # reduce
                         my $res=$opt1+$opt2;

                         push @opd,$res;# shift operand
                         push @opt,$c;# shift operator
                    } # end elsif($locd=~m!\+!)
                    elsif($locd=~m!\-!){ # begin elsif($locd=~m!\-!)
                         my $opt1=pop @opd; # reduce
                         my $opt2=pop @opd; # reduce
                         my $res=$opt2-$opt1;

                         push @opd,$res;# shift operand
                         push @opt,$c;# shift operator
                    } # end elsif($locd=~m!\-!)
               } # end else
               $num=();
          } # end if($c=~m!\+!)
          elsif($c=~m!\-!){ # begin elsif($c=~m!\-!)
               push @opd,$num;# shift number
               if(scalar(@opt)==0){ # begin if(scalar(@opt)==0)
                    push @opt,$c;# shift operator
               } # end if(scalar(@opt)==0)
               else{ # begin else
                    my $locd=pop @opt;# unshift operator to check
precedency # reduce
                    if($locd=~m!\*!){ # begin if($locd=~m!\*!)
                         my $opt1=pop @opd; # reduce
                         my $opt2=pop @opd; # reduce
                         my $res=$opt1*$opt2;

                         push @opd,$res;# shift operand
                         push @opt,$c;# shift operator
                    } # end if($locd=~m!\*!)
                    elsif($locd=~m!\/!){ # begin elsif($locd=~m!\/!)
                         my $opt1=pop @opd; # reduce
                         my $opt2=pop @opd; # reduce
                         my $res=$opt1/$opt2;

                         push @opd,$res;# shift operand
                         push @opt,$c;# shift operator
                    } # end elsif($locd=~m!\/!)
                    elsif($locd=~m!\+!){ # begin elsif($locd=~m!\+!)
                         my $opt1=pop @opd; # reduce
                         my $opt2=pop @opd; # reduce
                         my $res=$opt1+$opt2;

                         push @opd,$res;# shift operand
                         push @opt,$c;# shift operator
                    } # end elsif($locd=~m!\+!)
                    elsif($locd=~m!\-!){ # begin elsif($locd=~m!\-!)
                         my $opt1=pop @opd; # reduce
                         my $opt2=pop @opd; # reduce
                         my $res=$opt2-$opt1;

                         push @opd,$res;# shift operand
                         push @opt,$c;# shift operator
                    } # end elsif($locd=~m!\-!)
               } # end else
               $num=();
          } # end elsif($c=~m!\-!)
          elsif($c=~m!\*!){ # begin elsif($c=~m!\*!)
               push @opd,$num;# shift number
               if(scalar(@opt)==0){
                    push @opt,$c;# shift operator
               }
               else{ # begin else
                    my $locd=pop @opt;# unshift operator to check
     precedency # reduce
                    if($locd=~m!\*!){ # begin if($locd=~m!\*!)
                         my $opt1=pop @opd; # reduce
                         my $opt2=pop @opd; # reduce
                         my $res=$opt1*$opt2;

                         push @opd,$res;# shift operand
                         push @opt,$c;# shift operator
                    } # end if($locd=~m!\*!)
                    elsif($locd=~m!\/!){ # begin elsif($locd=~m!\/!)
                         my $opt1=pop @opd; # reduce
                         my $opt2=pop @opd; # reduce
                         my $res=$opt2/$opt1;

                         push @opd,$res;# shift operand
                         push @opt,$c;# shift operator
                    } # end elsif($locd=~m!\/!)
                    elsif($locd=~m!\+!){ # begin elsif($locd=~m!\+!)
                         push @opt,$locd;# shift previous operator
                         push @opt,$c;# shift current operator
                    } # end elsif($locd=~m!\+!)
                    elsif($locd=~m!\-!){ # begin elsif($locd=~m!\-!)
                         push @opt,$locd;# shift previous operator
                         push @opt,$c;# shift current operator
                    } # end elsif($locd=~m!\-!)
               } # end else
               $num=();
          } # end elsif($c=~m!\*!)
          elsif($c=~m!\/!){ # begin elsif($c=~m!\/!)
```

```
               push @opd,$num;# shift number
               if(scalar(@opt)==0){ # begin if(scalar(@opt)==0)
                    push @opt,$c;# shift operator
               } # end if(scalar(@opt)==0)
               else{ # begin else
                    my $locd=pop @opt;# unshift operator to check
precedency # reduce
                    if($locd=~m!\*!){ # begin if($locd=~m!\*!)
                         my $opt1=pop @opd; # reduce
                         my $opt2=pop @opd; # reduce
                         my $res=$opt1*$opt2;

                         push @opd,$res;# shift operand
                         push @opt,$c;# shift operator
                    } # end if($locd=~m!\*!)
                    elsif($locd=~m!\/!){ # begin elsif($locd=~m!\/!)
                         my $opt1=pop @opd; # reduce
                         my $opt2=pop @opd; # reduce
                         my $res=$opt1/$opt2;

                         push @opd,$res;# shift operand
                         push @opt,$c;# shift operator
                    } # end elsif($locd=~m!\/!)
                    elsif($locd=~m!\+!){ # begin elsif($locd=~m!\+!)
                         push @opt,$locd;# shift previous operator
                         push @opt,$c;# shift current operator
                    } # end elsif($locd=~m!\+!)
                    elsif($locd=~m!\-!){ # begin elsif($locd=~m!\-!)
                         push @opt,$locd;# shift previous operator
                         push @opt,$c;# shift current operator
                    } # end elsif($locd=~m!\-!)
               } # end else
               $num=();
          } # end elsif($c=~m!\/!)
          else{ # begin else
               $num.="$c";# concatenate string (number)
          } # end else
          $i++;
     } # end while($i<$size)
     # flush stacks
     while(scalar(@opd)||scalar(@opt)){ # begin while(scalar(@opd)||
scalar(@opt))
          my $opt1=pop @opd; # reduce
          my $locd=pop @opt;# unshift operator to check precedency #
reduce

          if($locd=~m!\*!){ # begin if($locd=~m!\*!)
               $num=$num*$opt1;
          } # end if($locd=~m!\*!)
          elsif($locd=~m!\/!){ # begin elsif($locd=~m!\/!)
               $num=$opt1/$num;
          } # end elsif($locd=~m!\/!)
          elsif($locd=~m!\+!){ # begin elsif($locd=~m!\+!)
               $num=$num+$opt1;
          } # end elsif($locd=~m!\+!)
          elsif($locd=~m!\-!){ # begin elsif($locd=~m!\-!)
               $num=$opt1-$num;
          } # end elsif($locd=~m!\-!)
     } # end while(scalar(@opd)||scalar(@opt))

     print "$a=>$num [@opt]  [@opd]\n";
     return $num;
}
```

20110512
The program second version



```
err: kkkk
ok 3+9
3+9=>12 []  []
err: 3b+9
ok 9+3
9+3=>12 []  []
ok 3-9
3-9=>-6 []  []
ok 9-3
9-3=>6 []  []
ok 81/9
81/9=>9 []  []
ok 3*9
3*9=>27 []  []
ok 3*9-1
3*9-1=>26 []  []
ok 3*9*2
3*9*2=>54 []  []
ok 4+3*9*2
4+3*9*2=>58 []  []
ok 2-81/9
2-81/9=>-7 []  []
esse-deos-macbook-pro:opr sdo$
```

Printing of the tests of
the second version of the algorithm

We can see that when it is not well formed *err: <expr>* is printed.

```
#!/usr/bin/perl

use strict;

reg("kkkk");# string to analyze
reg("3+9");# string to analyze
reg("3b+9");# string to analyze
reg("9+3");# string to analyze
reg("3-9");# string to analyze
reg("9-3");# string to analyze
reg("81/9");# string to analyze
reg("3*9");# string to analyze
reg("3*9-1");# string to analyze
reg("3*9*2");# string to analyze
reg("4+3*9*2");# string to analyze
reg("2-81/9");# string to analyze
#reg("3+9+2*3+8");# string to analyze
#reg("3*9+2*3+8");# string to analyze
#reg("3*9+2+3+2");# string to analyze
#reg("3*9+2-3+2");# string to analyze
#reg("3*9-2+3+2");# string to analyze
#reg("3*9-2-3+2");# string to analyze

sub reg{
     my ($mathExpr)=@_;
     my @opd=();# Stack for operand
     my @opt=();# Stack for operator
     my $pbeg=0;# Position at the begining
     my $pend=0;# Position at the end
     my $size=length($mathExpr);# size of string
     my $i=0;
     my $num=();
     my $c=();# current char

     $mathExpr=~s/[\ ]*//g;
     # Checks if expression is all right
     if($mathExpr!~m/^[0-9]{1,}([\-\+\*\/]{1,1}[0-9]{1,})*$/g){ #
begin if($mathExpr!~m/^[0-9]{1,}([\-\+\*\/]{1,1}[0-9]{1,})*$/g)
          print "err: $mathExpr\n";
          return;
     } # end if($mathExpr!~m/^[0-9]{1,}([\-\+\*\/]{1,1}[0-9]
{1,})*$/g)
     print "ok $mathExpr\n";
     while($i<$size){ # begin while($i<$size)
          $c=substr($mathExpr,$i,1);# gets one character
          if($c=~m!\+!){ # begin if($c=~m!\+!)
               push @opd,$num;# shift number
               if(scalar(@opt)==0){ # begin if(scalar(@opt)==0)
                    push @opt,$c;# shift operator
               } # end if(scalar(@opt)==0)
               else{ # begin else
                    my $locd=pop @opt;# unshift operator to check
precedency
                    if($locd=~m!\*!){ # begin if($locd=~m!\*!)
                         my $opt1=pop @opd; # reduce
                         my $opt2=pop @opd; # reduce
                         my $res=$opt1*$opt2;
                         push @opd,$res;# shift operand
                         push @opt,$c;# shift operator
                    } # end if($locd=~m!\*!)
                    elsif($locd=~m!\/!){ # begin elsif($locd=~m!\/!)
                         my $opt1=pop @opd; # reduce
                         my $opt2=pop @opd; # reduce
                         my $res=$opt2/$opt1;

                         push @opd,$res;# shift operand
                         push @opt,$c;# shift operator
                    } # end elsif($locd=~m!\/!)
                    elsif($locd=~m!\+!){ # begin elsif($locd=~m!\+!)
                         my $opt1=pop @opd; # reduce
                         my $opt2=pop @opd; # reduce
                         my $res=$opt1+$opt2;

                         push @opd,$res;# shift operand
                         push @opt,$c;# shift operator
                    } # end elsif($locd=~m!\+!)
                    elsif($locd=~m!\-!){ # begin elsif($locd=~m!\-!)
                         my $opt1=pop @opd; # reduce
                         my $opt2=pop @opd; # reduce
                         my $res=$opt2-$opt1;

                         push @opd,$res;# shift operand
                         push @opt,$c;# shift operator
                    } # end elsif($locd=~m!\-!)
               } # end else
               $num=();
          } # end if($c=~m!\+!)
          elsif($c=~m!\-!){ # begin elsif($c=~m!\-!)
               push @opd,$num;# shift number
               if(scalar(@opt)==0){ # begin if(scalar(@opt)==0)
                    push @opt,$c;# shift operator
               } # end if(scalar(@opt)==0)
               else{ # begin else
                    my $locd=pop @opt;# unshift operator to check
precedency # reduce
                    if($locd=~m!\*!){ # begin if($locd=~m!\*!)
                         my $opt1=pop @opd; # reduce
                         my $opt2=pop @opd; # reduce
                         my $res=$opt1*$opt2;

                         push @opd,$res;# shift operand
                         push @opt,$c;# shift operator
                    } # end if($locd=~m!\*!)
                    elsif($locd=~m!\/!){ # begin elsif($locd=~m!\/!)
                         my $opt1=pop @opd; # reduce
                         my $opt2=pop @opd; # reduce
                         my $res=$opt1/$opt2;

                         push @opd,$res;# shift operand
                         push @opt,$c;# shift operator
                    } # end elsif($locd=~m!\/!)
                    elsif($locd=~m!\+!){ # begin elsif($locd=~m!\+!)
                         my $opt1=pop @opd; # reduce
                         my $opt2=pop @opd; # reduce
                         my $res=$opt1+$opt2;

                         push @opd,$res;# shift operand
                         push @opt,$c;# shift operator
                    } # end elsif($locd=~m!\+!)
                    elsif($locd=~m!\-!){ # begin elsif($locd=~m!\-!)
                         my $opt1=pop @opd; # reduce
                         my $opt2=pop @opd; # reduce
                         my $res=$opt2-$opt1;

                         push @opd,$res;# shift operand
                         push @opt,$c;# shift operator
                    } # end elsif($locd=~m!\-!)
               } # end else
               $num=();
          } # end elsif($c=~m!\-!)
          elsif($c=~m!\*!){ # begin elsif($c=~m!\*!)
               push @opd,$num;# shift number
               if(scalar(@opt)==0){
                    push @opt,$c;# shift operator
               }
               else{ # begin else
                    my $locd=pop @opt;# unshift operator to check
     precedency # reduce
                    if($locd=~m!\*!){ # begin if($locd=~m!\*!)
                         my $opt1=pop @opd; # reduce
                         my $opt2=pop @opd; # reduce
                         my $res=$opt1*$opt2;

                         push @opd,$res;# shift operand
                         push @opt,$c;# shift operator
                    } # end if($locd=~m!\*!)
                    elsif($locd=~m!\/!){ # begin elsif($locd=~m!\/!)
                         my $opt1=pop @opd; # reduce
                         my $opt2=pop @opd; # reduce
                         my $res=$opt2/$opt1;

                         push @opd,$res;# shift operand
                         push @opt,$c;# shift operator
                    } # end elsif($locd=~m!\/!)
                    elsif($locd=~m!\+!){ # begin elsif($locd=~m!\+!)
```

```
                        push @opt,$locd;# shift previous operator
                        push @opt,$c;# shift current operator
                   } # end elsif($locd=~m!\+!)
                   elsif($locd=~m!\-!){ # begin elsif($locd=~m!\-!)
                        push @opt,$locd;# shift previous operator
                        push @opt,$c;# shift current operator
                   } # end elsif($locd=~m!\-!)
              } # end else
              $num=();
         } # end elsif($c=~m!\*!)
         elsif($c=~m!\/!){ # begin elsif($c=~m!\/!)
              push @opd,$num;# shift number
              if(scalar(@opt)==0){ # begin if(scalar(@opt)==0)
                   push @opt,$c;# shift operator
              } # end if(scalar(@opt)==0)
              else{ # begin else
                   my $locd=pop @opt;# unshift operator to check
precedency # reduce
                   if($locd=~m!\*!){ # begin if($locd=~m!\*!)
                        my $opt1=pop @opd; # reduce
                        my $opt2=pop @opd; # reduce
                        my $res=$opt1*$opt2;

                        push @opd,$res;# shift operand
                        push @opt,$c;# shift operator
                   } # end if($locd=~m!\*!)
                   elsif($locd=~m!\/!){ # begin elsif($locd=~m!\/!)
                        my $opt1=pop @opd; # reduce
                        my $opt2=pop @opd; # reduce
                        my $res=$opt1/$opt2;

                        push @opd,$res;# shift operand
                        push @opt,$c;# shift operator
                   } # end elsif($locd=~m!\/!)
                   elsif($locd=~m!\+!){ # begin elsif($locd=~m!\+!)
                        push @opt,$locd;# shift previous operator
                        push @opt,$c;# shift current operator
                   } # end elsif($locd=~m!\+!)
                   elsif($locd=~m!\-!){ # begin elsif($locd=~m!\-!)
                        push @opt,$locd;# shift previous operator
                        push @opt,$c;# shift current operator
                   } # end elsif($locd=~m!\-!)
              } # end else
              $num=();
         } # end elsif($c=~m!\/!)
         else{ # begin else
              $num.="$c";# concatenate string (number)
         } # end else
         $i++;
```

```
     } # end while($i<$size)
     # flush stacks
     while(scalar(@opd)||scalar(@opt)){ # begin while(scalar(@opd)||
scalar(@opt))
          my $opt1=pop @opd; # reduce
          my $locd=pop @opt;# unshift operator to check precedency #
reduce

          if($locd=~m!\*!){ # begin if($locd=~m!\*!)
               $num=$num*$opt1;
          } # end if($locd=~m!\*!)
          elsif($locd=~m!\/!){ # begin elsif($locd=~m!\/!)
               $num=$opt1/$num;
          } # end elsif($locd=~m!\/!)
          elsif($locd=~m!\+!){ # begin elsif($locd=~m!\+!)
               $num=$num+$opt1;
          } # end elsif($locd=~m!\+!)
          elsif($locd=~m!\-!){ # begin elsif($locd=~m!\-!)
               $num=$opt1-$num;
          } # end elsif($locd=~m!\-!)
     } # end while(scalar(@opd)||scalar(@opt))

     print "$mathExpr=>$num [@opt]  [@opd]\n";
     return $num;
}
```

20110513

The next version of the basic calculator include parenthesis. It has not be fully tested but it seems working.

Here are some basic tests functionality:

```
err: kkkk
++++err: )(3+9)
(3+9)=>12 []  []
(3+9)+1=>13 []  []
err: (3*()+9)
1+(3+9)=>13 []  []
1+(3+9)+2=>15 []  []
1+(3+(9+3+9))+2=>27 []  []
1+(3+9+2)+2=>17 []  []
2*(3+9)=>24 []  []
1+2*(3+9)=>25 []  []
(1+2)*(3+9)=>36 []  []
(1+2*(2+(2+6)*2)*2)*(3+9)=>876 []  []
(1+2*(2+(2+6)*2)*2)*(3+9)-10*3+9=>837 []  []
3+9=>12 []  []
err: 3b+9
9+3=>12 []  []
3-9=>-6 []  []
```

```
9-3=>6 []  []
81/9=>9 []  []
3*9=>27 []  []
3*9-1=>26 []  []
3*9*2=>54 []  []
4+3*9*2=>58 []  []
2-81/9=>-7 []  []
3+9+2*3+8=>26 []  []
3*9+2*3+8=>41 []  []
3*9+2+3+2=>34 []  []
3*9+2-3+2=>28 []  []
3*9-2+3+2=>30 []  []
3*9-2-3+2=>24 []  []
```

Basic tests of functionality

Checks different syntax errors s.a () or at the first char). Verify that to each open parenthesis match a closed parenthesis. Calculate order precedency. The higher priority are the parenthesis and lowest - + signs. Multiplication and division operands are not as important in priority as parenthesis but more than + and -.

Here is the source code:

```
#!/usr/bin/perl

use strict;

calc("kkkk"); # string to analyze
calc(")(3+9)"); # string to analyze
calc("(3+9)"); # string to analyze
calc("(3+9)+1"); # string to analyze
calc("(3*()+9)"); # string to analyze
calc("1+(3+9)"); # string to analyze
calc("1+(3+9)+2"); # string to analyze
calc("1+(3+(9+3+9))+2"); # string to analyze
calc("1+(3+9+2)+2"); # string to analyze
calc("2*(3+9)"); # string to analyze
calc("1+2*(3+9)"); # string to analyze
calc("(1+2)*(3+9)"); # string to analyze
calc("(1+2*(2+(2+6)*2)*2)*(3+9)"); # string to analyze
calc("(1+2*(2+(2+6)*2)*2)*(3+9)-10*3+9"); # string to analyze
calc("3+9"); # string to analyze
calc("3b+9"); # string to analyze
calc("9+3"); # string to analyze
calc("3-9"); # string to analyze
calc("9-3"); # string to analyze
calc("81/9"); # string to analyze
calc("3*9"); # string to analyze
calc("3*9-1"); # string to analyze
calc("3*9*2"); # string to analyze
calc("4+3*9*2"); # string to analyze
calc("2-81/9"); # string to analyze
calc("3+9+2*3+8"); # string to analyze
calc("3*9+2*3+8"); # string to analyze
calc("3*9+2+3+2"); # string to analyze
calc("3*9+2-3+2"); # string to analyze
calc("3*9-2+3+2"); # string to analyze
calc("3*9-2-3+2"); # string to analyze

sub calc{ # begin sub calc
     my ($mathExpr)=@_;
     my @opd=(); # Stack for operand
     my @opt=(); # Stack for operator
     my $pbeg=0; # Position at the begining
     my $pend=0; # Position at the end
     my $size=length($mathExpr); # size of string
     my $i=0;
     my $num=();
     my $c=(); # current char

     $mathExpr=~s/[\ ]*//g;
     # sanitary tests
     # Checks if expression is all right
     if($mathExpr!~m/^[()0-9]{1,}([\-\+\*\/]{1,1}[()0-9]{1,})*$/g)
{ # begin if($mathExpr!~m/^[0-9]{1,}([\-\+\*\/]{1,1}[0-9]{1,})*$/g)
          print "err: $mathExpr\n";
          return;
     } # end if($mathExpr!~m/^[0-9]{1,}([\-\+\*\/]{1,1}[0-9]
{1,})*$/g)
     if($mathExpr=~m/^[\)\*\/]*/g){ # begin
if($mathExpr=~m/^[\)\*\/]*/g)
          print "err: $mathExpr\n";
          return;
     } # end if($mathExpr=~m/^[\)\*\/]*/g)
     if($mathExpr=~m/\(\)/g){ # begin if($mathExpr=~m/\(\)/g)
          print "err: $mathExpr\n";
          return;
     } # end if($mathExpr=~m/\(\)/g)
     $num=0;
     while($i<$size){ # begin while($i<$size)
          $c=substr($mathExpr,$i,1); # gets one character
          if($c=~m/\(/){ # begin if($c=~m/\(/)
               $num++;
          } # end if($c=~m/\(/)
          elsif($c=~m/\)/){ # begin elsif($c=~m/\)/)
               $num--;
          } # end elsif($c=~m/\)/)
          $i++;
     } # end while($i<$size)
     if($num>0){
          print "err($num): $mathExpr\n";
          return;
     } # end if($mathExpr!~m/^[\)\*\/]*/g)
     # end sanitary tests
     $i=0;
     $num=();
     while($i<$size){ # begin while($i<$size)
          $c=substr($mathExpr,$i,1); # gets one character
          if($c=~m!\+!){ # begin if($c=~m!\+!)
               if(length($num)>0){ # begin if(length($num)==0)
#                   $num=0;
                    push @opd,$num; # shift number
               } # end if(length($num)==0)
               if(scalar(@opt)==0){ # begin if(scalar(@opt)==0)
                    push @opt,$c; # shift operator
               } # end if(scalar(@opt)==0)
               else{ # begin else
                    my $locd=pop @opt; # unshift operator to check
precedency
                    if($locd=~m!\*!){ # begin if($locd=~m!\*!)
                         my $opt1=pop @opd; # reduce
                         my $opt2=pop @opd; # reduce
                         my $res=$opt1*$opt2; # operation done

                         push @opd,$res; # shift operand
                         push @opt,$c; # shift operator
                    } # end if($locd=~m!\*!)
                    elsif($locd=~m!\/!){ # begin elsif($locd=~m!\/!)
                         my $opt1=pop @opd; # reduce
                         my $opt2=pop @opd; # reduce
                         my $res=$opt2/$opt1; # operation done

                         push @opd,$res; # shift operand
                         push @opt,$c; # shift operator
                    } # end elsif($locd=~m!\/!)
                    elsif($locd=~m!\+!){ # begin elsif($locd=~m!\+!)
                         my $opt1=pop @opd; # reduce
                         my $opt2=pop @opd; # reduce
                         my $res=$opt1+$opt2; # operation done

                         push @opd,$res; # shift operand
                         push @opt,$c; # shift operator
                    } # end elsif($locd=~m!\+!)
                    elsif($locd=~m!\-!){ # begin elsif($locd=~m!\-!)
                         my $opt1=pop @opd; # reduce
                         my $opt2=pop @opd; # reduce
                         my $res=$opt2-$opt1; # operation done
                         push @opd,$res; # shift operand
                         push @opt,$c; # shift operator
                    } # end elsif($locd=~m!\-!)
                    elsif($locd=~m!\(!){ # begin if($locd=~m!\(!)
          #              push @opd,$num; # shift operand
                         push @opt,$locd; # shift operator
                         push @opt,$c; # shift operator
                    } # end if($locd=~m!\(!)
               } # end else
               $num=();
          } # end if($c=~m!\+!)
          elsif($c=~m!\-!){ # begin elsif($c=~m!\-!)
               if(length($num)>0){ # begin if(length($num)>0)
                    push @opd,$num; # shift number
               } # end if(length($num)==0)
               if(scalar(@opt)==0){ # begin if(scalar(@opt)==0)
                    push @opt,$c; # shift operator
               } # end if(scalar(@opt)==0)
               else{ # begin else
                    my $locd=pop @opt; # unshift operator to check
precedency # reduce
                    if($locd=~m!\*!){ # begin if($locd=~m!\*!)
                         my $opt1=pop @opd; # reduce
                         my $opt2=pop @opd; # reduce
                         my $res=$opt1*$opt2; # operation done

                         push @opd,$res; # shift operand
                         push @opt,$c; # shift operator
                    } # end if($locd=~m!\*!)
                    elsif($locd=~m!\/!){ # begin elsif($locd=~m!\/!)
                         my $opt1=pop @opd; # reduce
                         my $opt2=pop @opd; # reduce
                         my $res=$opt1/$opt2;

                         push @opd,$res; # shift operand
                         push @opt,$c; # shift operator
                    } # end elsif($locd=~m!\/!)
                    elsif($locd=~m!\+!){ # begin elsif($locd=~m!\+!)
                         my $opt1=pop @opd; # reduce
                         my $opt2=pop @opd; # reduce
                         my $res=$opt1+$opt2; # operation done

                         push @opd,$res; # shift operand
                         push @opt,$c; # shift operator
                    } # end elsif($locd=~m!\+!)
                    elsif($locd=~m!\-!){ # begin elsif($locd=~m!\-!)
                         my $opt1=pop @opd; # reduce
                         my $opt2=pop @opd; # reduce
                         my $res=$opt2-$opt1; # operation done
                         push @opd,$res; # shift operand
                         push @opt,$c; # shift operator
                    } # end elsif($locd=~m!\-!)
               } # end else
               $num=();
          } # end elsif($c=~m!\-!)
          elsif($c=~m!\*!){ # begin elsif($c=~m!\*!)
               if(length($num)>0){ # begin if(length($num)>0)
                    push @opd,$num; # shift number
               } # end if(length($num)==0)
               if(scalar(@opt)==0){
                    push @opt,$c; # shift operator
               }
               else{ # begin else
                    my $locd=pop @opt; # unshift operator to check
     precedency # reduce
                    if($locd=~m!\*!){ # begin if($locd=~m!\*!)
                         my $opt1=pop @opd; # reduce
                         my $opt2=pop @opd; # reduce
                         my $res=$opt1*$opt2; # operation done

                         push @opd,$res; # shift operand
                         push @opt,$c; # shift operator
                    } # end if($locd=~m!\*!)
                    elsif($locd=~m!\/!){ # begin elsif($locd=~m!\/!)
                         my $opt1=pop @opd; # reduce
                         my $opt2=pop @opd; # reduce
                         my $res=$opt2/$opt1; # operation done

                         push @opd,$res; # shift operand
                         push @opt,$c; # shift operator
                    } # end elsif($locd=~m!\/!)
                    elsif($locd=~m!\+!){ # begin elsif($locd=~m!\+!)
                         push @opt,$locd; # shift previous operator
                         push @opt,$c; # shift current operator
                    } # end elsif($locd=~m!\+!)
                    elsif($locd=~m!\-!){ # begin elsif($locd=~m!\-!)
                         push @opt,$locd; # shift previous operator
                         push @opt,$c; # shift current operator
                    } # end elsif($locd=~m!\-!)
               } # end else
               $num=();
          } # end elsif($c=~m!\*!)
          elsif($c=~m!\/!){ # begin elsif($c=~m!\/!)
               if(length($num)>0){ # begin if(length($num)>0)
                    push @opd,$num; # shift number
               } # end if(length($num)==0)
               if(scalar(@opt)==0){ # begin if(scalar(@opt)==0)
```

```
                    push @opt,$c; # shift operator
               } # end if(scalar(@opt)==0)
               else{ # begin else
                    my $locd=pop @opt; # unshift operator to check
precedency # reduce
                    if($locd=~m!\*!){ # begin if($locd=~m!\*!)
                         my $opt1=pop @opd; # reduce
                         my $opt2=pop @opd; # reduce
                         my $res=$opt1*$opt2; # operation done

                         push @opd,$res; # shift operand
                         push @opt,$c; # shift operator
                    } # end if($locd=~m!\*!)
                    elsif($locd=~m!\/!){ # begin elsif($locd=~m!\/!)
                         my $opt1=pop @opd; # reduce
                         my $opt2=pop @opd; # reduce
                         my $res=$opt1/$opt2;

                         push @opd,$res; # shift operand
                         push @opt,$c; # shift operator
                    } # end elsif($locd=~m!\/!)
                    elsif($locd=~m!\+!){ # begin elsif($locd=~m!\+!)
                         push @opt,$locd; # shift previous operator
                         push @opt,$c; # shift current operator
                    } # end elsif($locd=~m!\+!)
                    elsif($locd=~m!\-!){ # begin elsif($locd=~m!\-!)
                         push @opt,$locd; # shift previous operator
                         push @opt,$c; # shift current operator
                    } # end elsif($locd=~m!\-!)
               } # end else
               $num=();
          } # end elsif($c=~m!\/!)
          elsif($c=~m!\(!){ # begin elsif($c=~m!\(!)
               if(length($num)>0){ # begin if(length($num)>0)
                    push @opd,$num; # shift number
               } # end if(length($num)>0)
               if(scalar(@opt)==0){ # begin if(scalar(@opt)==0)
                    push @opt,$c; # shift operator
               } # end if(scalar(@opt)==0)
               else{ # begin else
                    push @opt,$c; # shift operator
               } # end else
               $num=();
          } # end elsif($c=~m!\(!)
          elsif($c=~m!\)!){ # begin elsif($c=~m!\)!)
               if(length($num)>0){ # begin if(length($num)>0)
                    push @opd,$num; # shift number
               } # end if(length($num)>0)
               if(scalar(@opt)==0){ # begin if(scalar(@opt)==0)
```

```
                    #push @opt,$c; # shift operator
                    print "++++err: $mathExpr\n";
                    return;
               } # end if(scalar(@opt)==0)
               else{ # begin else
                    my $locd=pop @opt; # unshift operator to check
precedency # reduce
                    # we calculculate till ( is met
                    while($locd!~m/\(/){ # begin while($locd!~m/\(/)
                         if($locd=~m!\*!){ # begin if($locd=~m!\*!)
                              my $opt1=pop @opd; # reduce
                              my $opt2=pop @opd; # reduce
                              my $res=$opt1*$opt2; # operation done

                              push @opd,$res; # shift operand
                              #$opt1=pop @opt; # reduce
                         } # end if($locd=~m!\*!)
                         elsif($locd=~m!\/!){ # begin
elsif($locd=~m!\/!)
                              my $opt1=pop @opd; # reduce
                              my $opt2=pop @opd; # reduce
                              my $res=$opt1/$opt2;

                              push @opd,$res; # shift operand
                              #$opt1=pop @opt; # reduce
                         } # end elsif($locd=~m!\/!)
                         elsif($locd=~m!\+!){ # begin
elsif($locd=~m!\+!)
                              my $opt1=pop @opd; # reduce
                              my $opt2=pop @opd; # reduce
                              my $res=$opt1+$opt2; # operation done

                              push @opd,$res; # shift operand
                              #$opt1=pop @opt; # reduce
                         } # end elsif($locd=~m!\+!)
                         elsif($locd=~m!\-!){ # begin
elsif($locd=~m!\-!)
                              my $opt1=pop @opd; # reduce
                              my $opt2=pop @opd; # reduce
                              my $res=$opt2-$opt1; # operation done

                              push @opd,$res; # shift operand
                              #$opt1=pop @opt; # reduce
                         } # end elsif($locd=~m!\-!)
                         $locd=pop @opt; # unshift operator to check
precedency # reduce
                    } # end while($locd!~m/\(/)
               } # end else
               $num=();
```

```
          } # end elsif($c=~m!\)!)
          else{ # begin else
               $num.="$c"; # concatenate string (number)
          } # end else
          $i++;
     } # end while($i<$size)
     if(length($num)==0){
          $num=pop @opd; # reduce
     }
     # flush stacks
     while(scalar(@opd)||scalar(@opt)){ # begin while(scalar(@opd)||
scalar(@opt))
          my $opt1=pop @opd; # reduce
          my $locd=pop @opt; # unshift operator to check precedency
# reduce

          if($locd=~m!\*!){ # begin if($locd=~m!\*!)
               $num=$num*$opt1;
          } # end if($locd=~m!\*!)
          elsif($locd=~m!\/!){ # begin elsif($locd=~m!\/!)
               $num=$opt1/$num;
          } # end elsif($locd=~m!\/!)
          elsif($locd=~m!\+!){ # begin elsif($locd=~m!\+!)
               $num=$num+$opt1;
          } # end elsif($locd=~m!\+!)
          elsif($locd=~m!\-!){ # begin elsif($locd=~m!\-!)
               $num=$opt1-$num;
          } # end elsif($locd=~m!\-!)
          else{ # begin else
               $num=$opt1;
          } # end else
     } # end while(scalar(@opd)||scalar(@opt))

     print "$mathExpr=>$num [@opt]  [@opd]\n";
     return $num;
} # end sub calc
```

Source code of the calculator

20110522
Due to extra tests it seems that some basic operation could not be performed correctly. A test function was done. Extra tests upon calculation too.

Here are the tests results:

```
kkkk:BAD 1 waited but ERR returned ; err: kkkk
kkkk:OK ERR waited and ERR returned
)(3+9):BAD 0 waited but ERR returned ; err: )(3+9)
)(3+9):OK ERR waited and ERR returned
)(3+9):OK ERR waited and ERR returned
```

```
(3+9):OK 12 waited and 12 returned
(3+9):BAD ERR waited but 12 returned ;
(3+9):BAD 11 waited but 12 returned ;
(3+9)+1:OK 13 waited and 13 returned
(3*()+9):BAD   waited but ERR returned ; err: (3*()+9)
1+(3+9):OK 13 waited and 13 returned
1-(3+9):BAD 12 waited but -11 returned ;
1-(3+9):BAD -12 waited but -11 returned ;
1+(3+9)+2:BAD 14 waited but 15 returned ;
1+(3+(9+3+9))+2:OK 27 waited and 27 returned
1+(3+9+2)+2:OK 17 waited and 17 returned
2*(3+9):OK 24 waited and 24 returned
1+2*(3+9):OK 25 waited and 25 returned
(1+2)*(3+9):OK 36 waited and 36 returned
(1+2*(2+(2+6)*2)*2)*(3+9):OK 876 waited and 876 returned
(1+2*(2+(2+6)*2)*2)*(3+9)-10*3+9:OK 837 waited and 837 returned
3+9:OK 12 waited and 12 returned
3b+9:BAD 0 waited but ERR returned ; err: 3b+9
9+3:OK 12 waited and 12 returned
3-9:OK -6 waited and -6 returned
9-3:OK 6 waited and 6 returned
81/9:OK 9 waited and 9 returned
3*9:OK 27 waited and 27 returned
3*9-1:OK 26 waited and 26 returned
3*9*2:OK 54 waited and 54 returned
4+3*9*2:OK 58 waited and 58 returned
2-81/9:OK -7 waited and -7 returned
3+9+2*3+8:OK 26 waited and 26 returned
3*9+2*3+8:OK 41 waited and 41 returned
3*9+2+3+2:OK 34 waited and 34 returned
3*9+2-3+2:OK 28 waited and 28 returned
3*9-2+3+2:BAD 24 waited but 30 returned ;
3*9-2-3+2:OK 24 waited and 24 returned
3*(9-2)-3+2:OK 20 waited and 20 returned
3*(9-2)-3+2:OK 20 waited and 20 returned
3*((9-2)-3)+2:OK 14 waited and 14 returned
(2+(4-2)*3+(2-3)*2)*(3+9):OK 72 waited and 72 returned
```
Bold lines are the new tests.

Here the new listing :

```
#!/usr/bin/perl

use strict;

check(1,calc("kkkk")); # string to analyze
check("ERR",calc("kkkk")); # string to analyze
check(0,calc(")(3+9)")); # string to analyze
check("ERR",calc(")(3+9)")); # string to analyze
check("ERR",calc(")(3+9)")); # string to analyze
check(12,calc("(3+9)")); # string to analyze
check("ERR",calc("(3+9)")); # string to analyze
check(11,calc("(3+9)")); # string to analyze
check(13,calc("(3+9)+1")); # string to analyze
check(" ",calc("(3*()+9)")); # string to analyze
check(13,calc("1+(3+9)")); # string to analyze
check(12,calc("1-(3+9)")); # string to analyze
check(-12,calc("1-(3+9)")); # string to analyze
check(14,calc("1+(3+9)+2")); # string to analyze
check(27,calc("1+(3+(9+3+9))+2")); # string to analyze
check(17,calc("1+(3+9+2)+2")); # string to analyze
check(24,calc("2*(3+9)")); # string to analyze
check(25,calc("1+2*(3+9)")); # string to analyze
check(36,calc("(1+2)*(3+9)")); # string to analyze
check(876,calc("(1+2*(2+(2+6)*2)*2)*(3+9)")); # string to analyze
check(837,calc("(1+2*(2+(2+6)*2)*2)*(3+9)-10*3+9")); # string to
analyze
check(12,calc("3+9")); # string to analyze
check(0,calc("3b+9")); # string to analyze
check(12,calc("9+3")); # string to analyze
check(-6,calc("3-9")); # string to analyze
check(6,calc("9-3")); # string to analyze
check(9,calc("81/9")); # string to analyze
check(27,calc("3*9")); # string to analyze
check(26,calc("3*9-1")); # string to analyze
check(54,calc("3*9*2")); # string to analyze
check(58,calc("4+3*9*2")); # string to analyze
check(-7,calc("2-81/9")); # string to analyze
check(26,calc("3+9+2*3+8")); # string to analyze
check(41,calc("3*9+2*3+8")); # string to analyze
check(34,calc("3*9+2+3+2")); # string to analyze
check(28,calc("3*9+2-3+2")); # string to analyze
check(24,calc("3*9-2+3+2")); # string to analyze
check(24,calc("3*9-2-3+2")); # string to analyze
check(20,calc("3*(9-2)-3+2")); # string to analyze
check(20,calc("3*(9-2)-3+2")); # string to analyze
check(14,calc("3*((9-2)-3)+2")); # string to analyze
check(72,calc("(2+(4-2)*3+(2-3)*2)*(3+9)")); # string to analyze

sub calc{ # begin sub calc
     my ($mathExpr)=@_;
     my @opd=(); # Stack for operand
     my @opt=(); # Stack for operator
     my $pbeg=0; # Position at the begining
     my $pend=0; # Position at the end
     my $size=length($mathExpr); # size of string
     my $i=0;
     my $num=();
     my $c=(); # current char

     $mathExpr=~s/[\ ]*//g;
     # sanitary tests
     # Checks if expression is all right
     if($mathExpr!~m/^[()0-9]{1,}([\-\+\*\/]{1,1}[()0-9]{1,})*$/g)
{ # begin if($mathExpr!~m/^[0-9]{1,}([\-\+\*\/]{1,1}[0-9]{1,})*$/g)
          return ("ERR", "err: $mathExpr",$mathExpr);
     } # end if($mathExpr!~m/^[0-9]{1,}([\-\+\*\/]{1,1}[0-9]
{1,})*$/g)
     if($mathExpr=~m/^[\)\*\/]*/g){ # begin
if($mathExpr=~m/^[\)\*\/]*/g)
          return ("ERR","err: $mathExpr",$mathExpr);
     } # end if($mathExpr=~m/^[\)\*\/]*/g)
     if($mathExpr=~m/\(\)/g){ # begin if($mathExpr=~m/\(\)/g)
          return ("ERR","err: $mathExpr",$mathExpr);
     } # end if($mathExpr=~m/\(\)/g)
     $num=0;
     while($i<$size){ # begin while($i<$size)
          $c=substr($mathExpr,$i,1); # gets one character
          if($c=~m/\(/){ # begin if($c=~m/\(/)
               $num++;
          } # end if($c=~m/\(/)
          elsif($c=~m/\)/){ # begin elsif($c=~m/\)/)
               $num--;
          } # end elsif($c=~m/\)/)
          $i++;
     } # end while($i<$size)
     if($num>0){
          return ("ERR","err($num): $mathExpr",$mathExpr);
     } # end if($mathExpr!~m/^[\)\*\/]*/g)
     # end sanitary tests
     $i=0;
     $num=();
     while($i<$size){ # begin while($i<$size)
          $c=substr($mathExpr,$i,1); # gets one character
          if($c=~m!\+!){ # begin if($c=~m!\+!)
               if(length($num)>0){ # begin if(length($num)>0)
#                   $num=0;
                    push @opd,$num; # shift number
               } # end if(length($num)>0)
               if(scalar(@opt)==0){ # begin if(scalar(@opt)==0)
                    push @opt,$c; # shift operator
               } # end if(scalar(@opt)==0)
               else{ # begin else
                    my $locd=pop @opt; # unshift operator to check
precedency
                    if($locd=~m!\*!){ # begin if($locd=~m!\*!)
                         my $opt1=pop @opd; # reduce
                         my $opt2=pop @opd; # reduce
                         my $res=$opt1*$opt2; # operation done

                         push @opd,$res; # shift operand
                         push @opt,$c; # shift operator
                    } # end if($locd=~m!\*!)
                    elsif($locd=~m!\/!){ # begin elsif($locd=~m!\/!)
                         my $opt1=pop @opd; # reduce
                         my $opt2=pop @opd; # reduce
                         my $res=$opt2/$opt1; # operation done

                         push @opd,$res; # shift operand
                         push @opt,$c; # shift operator
                    } # end elsif($locd=~m!\/!)
                    elsif($locd=~m!\+!){ # begin elsif($locd=~m!\+!)
                         my $opt1=pop @opd; # reduce
                         my $opt2=pop @opd; # reduce
                         my $res=$opt1+$opt2; # operation done

                         push @opd,$res; # shift operand
                         push @opt,$c; # shift operator
                    } # end elsif($locd=~m!\+!)
                    elsif($locd=~m!\-!){ # begin elsif($locd=~m!\-!)
                         my $opt1=pop @opd; # reduce
                         my $opt2=pop @opd; # reduce
                         my $res=$opt2-$opt1; # operation done

                         push @opd,$res; # shift operand
                         push @opt,$c; # shift operator
                    } # end elsif($locd=~m!\-!)
                    elsif($locd=~m!\(!){ # begin if($locd=~m!\(!)
          #              push @opd,$num; # shift operand
                         push @opt,$locd; # shift operator
                         push @opt,$c; # shift operator
                    } # end if($locd=~m!\(!)
               } # end else
               $num=();
          } # end if($c=~m!\+!)
          elsif($c=~m!\-!){ # begin elsif($c=~m!\-!)
               if(length($num)>0){ # begin if(length($num)>0)
                    push @opd,$num; # shift number
               } # end if(length($num)==0)
               if(scalar(@opt)==0){ # begin if(scalar(@opt)==0)
                    push @opt,$c; # shift operator
               } # end if(scalar(@opt)==0)
               else{ # begin else
                    my $locd=pop @opt; # unshift operator to check
precedency # reduce
                    if($locd=~m!\*!){ # begin if($locd=~m!\*!)
```

```
                         my $opt1=pop @opd; # reduce
                         my $opt2=pop @opd; # reduce
                         my $res=$opt1*$opt2; # operation done

                         push @opd,$res; # shift operand
                         push @opt,$c; # shift operator
                    } # end if($locd=~m!\*!)
                    elsif($locd=~m!\/!){ # begin elsif($locd=~m!\/!)
                         my $opt1=pop @opd; # reduce
                         my $opt2=pop @opd; # reduce
                         my $res=$opt1/$opt2;

                         push @opd,$res; # shift operand
                         push @opt,$c; # shift operator
                    } # end elsif($locd=~m!\/!)
                    elsif($locd=~m!\+!){ # begin elsif($locd=~m!\+!)
                         my $opt1=pop @opd; # reduce
                         my $opt2=pop @opd; # reduce
                         my $res=$opt1+$opt2; # operation done

                         push @opd,$res; # shift operand
                         push @opt,$c; # shift operator
                    } # end elsif($locd=~m!\+!)
                    elsif($locd=~m!\-!){ # begin elsif($locd=~m!\-!)
                         my $opt1=pop @opd; # reduce
                         my $opt2=pop @opd; # reduce
                         my $res=$opt2-$opt1; # operation done

                         push @opd,$res; # shift operand
                         push @opt,$c; # shift operator
                    } # end elsif($locd=~m!\-!)
                    elsif($locd=~m!\(!){ # begin elsif($locd=~m!\(!)
                         push @opt,$locd; # shift operator
                         push @opt,$c; # shift operator
                    } # end elsif($locd=~m!\(!)
               } # end else
               $num=();
          } # end elsif($c=~m!\-!)
          elsif($c=~m!\*!){ # begin elsif($c=~m!\*!)
               if(length($num)>0){ # begin if(length($num)>0)
                    push @opd,$num; # shift number
               } # end if(length($num)==0)
               if(scalar(@opt)==0){ # begin if(scalar(@opt)==0)
                    push @opt,$c; # shift operator
               } # end if(scalar(@opt)==0)
               else{ # begin else
                    my $locd=pop @opt; # unshift operator to check
precedency # reduce
                    if($locd=~m!\*!){ # begin if($locd=~m!\*!)
                         my $opt1=pop @opd; # reduce
                         my $opt2=pop @opd; # reduce
                         my $res=$opt1*$opt2; # operation done

                         push @opd,$res; # shift operand
                         push @opt,$c; # shift operator
                    } # end if($locd=~m!\*!)
                    elsif($locd=~m!\/!){ # begin elsif($locd=~m!\/!)
                         my $opt1=pop @opd; # reduce
                         my $opt2=pop @opd; # reduce
                         my $res=$opt2/$opt1; # operation done

                         push @opd,$res; # shift operand
                         push @opt,$c; # shift operator
                    } # end elsif($locd=~m!\/!)
                    elsif($locd=~m!\+!){ # begin elsif($locd=~m!\+!)
                         push @opt,$locd; # shift previous operator
                         push @opt,$c; # shift current operator
                    } # end elsif($locd=~m!\+!)
                    elsif($locd=~m!\-!){ # begin elsif($locd=~m!\-!)
                         push @opt,$locd; # shift previous operator
                         push @opt,$c; # shift current operator
                    } # end elsif($locd=~m!\-!)
                    elsif($locd=~m!\(!){ # begin elsif($locd=~m!\(!)
                         push @opt,$locd; # shift operator
                         push @opt,$c; # shift operator
                    } # end elsif($locd=~m!\(!)
               } # end else
               $num=();
          } # end elsif($c=~m!\*!)
          elsif($c=~m!\/!){ # begin elsif($c=~m!\/!)
               if(length($num)>0){ # begin if(length($num)>0)
                    push @opd,$num; # shift number
               } # end if(length($num)==0)
               if(scalar(@opt)==0){ # begin if(scalar(@opt)==0)
                    push @opt,$c; # shift operator
               } # end if(scalar(@opt)==0)
               else{ # begin else
                    my $locd=pop @opt; # unshift operator to check
precedency # reduce
                    if($locd=~m!\*!){ # begin if($locd=~m!\*!)
                         my $opt1=pop @opd; # reduce
                         my $opt2=pop @opd; # reduce
                         my $res=$opt1*$opt2; # operation done

                         push @opd,$res; # shift operand
                         push @opt,$c; # shift operator
                    } # end if($locd=~m!\*!)
                    elsif($locd=~m!\/!){ # begin elsif($locd=~m!\/!)
```

```
                    my $opt1=pop @opd; # reduce
                    my $opt2=pop @opd; # reduce
                    my $res=$opt1/$opt2;

                    push @opd,$res; # shift operand
                    push @opt,$c; # shift operator
               } # end elsif($locd=~m!\/!)
               elsif($locd=~m!\+!){ # begin elsif($locd=~m!\+!)
                    push @opt,$locd; # shift previous operator
                    push @opt,$c; # shift current operator
               } # end elsif($locd=~m!\+!)
               elsif($locd=~m!\-!){ # begin elsif($locd=~m!\-!)
                    push @opt,$locd; # shift previous operator
                    push @opt,$c; # shift current operator
               } # end elsif($locd=~m!\-!)
               elsif($locd=~m!\(!){ # begin elsif($locd=~m!\(!)
                    push @opt,$locd; # shift operator
                    push @opt,$c; # shift operator
               } # end elsif($locd=~m!\(!)
          } # end else
          $num=();
     } # end elsif($c=~m!\/!)
     elsif($c=~m!\(!){ # begin elsif($c=~m!\(!)
          if(length($num)>0){ # begin if(length($num)>0)
               push @opd,$num; # shift number
          } # end if(length($num)>0)
          if(scalar(@opt)==0){ # begin if(scalar(@opt)==0)
               push @opt,$c; # shift operator
          } # end if(scalar(@opt)==0)
          else{ # begin else
               push @opt,$c; # shift operator
          } # end else
          $num=();
     } # end elsif($c=~m!\(!)
     elsif($c=~m!\)!){ # begin elsif($c=~m!\)!)
          if(length($num)>0){ # begin if(length($num)>0)
               push @opd,$num; # shift number
          } # end if(length($num)>0)
          if(scalar(@opt)==0){ # begin if(scalar(@opt)==0)
               return ("ERR","err: $mathExpr",$mathExpr);
          } # end if(scalar(@opt)==0)
          else{ # begin else
               my $locd=pop @opt; # unshift operator to check
precedency # reduce
               # we calculculate till ( is met
               while($locd!~m/\(/){ # begin while($locd!~m/\(/)
                    if($locd=~m!\*!){ # begin if($locd=~m!\*!)
                         my $opt1=pop @opd; # reduce
                         my $opt2=pop @opd; # reduce
```

```
                              my $res=$opt1*$opt2; # operation done

                              push @opd,$res; # shift operand
                              #$opt1=pop @opt; # reduce
                         } # end if($locd=~m!\*!)
                         elsif($locd=~m!\/!){ # begin
elsif($locd=~m!\/!)
                              my $opt1=pop @opd; # reduce
                              my $opt2=pop @opd; # reduce
                              my $res=$opt1/$opt2;

                              push @opd,$res; # shift operand
                         } # end elsif($locd=~m!\/!)
                         elsif($locd=~m!\+!){ # begin
elsif($locd=~m!\+!)
                              my $opt1=pop @opd; # reduce
                              my $opt2=pop @opd; # reduce
                              my $res=$opt1+$opt2; # operation done

                              push @opd,$res; # shift operand
                         } # end elsif($locd=~m!\+!)
                         elsif($locd=~m!\-!){ # begin
elsif($locd=~m!\-!)
                              my $opt1=pop @opd; # reduce
                              my $opt2=pop @opd; # reduce
                              my $res=$opt2-$opt1; # operation done

                              push @opd,$res; # shift operand
                         } # end elsif($locd=~m!\-!)
                         $locd=pop @opt; # unshift operator to check
precedency # reduce
                    } # end while($locd!~m/\(/)
               } # end else
               $num=();
          } # end elsif($c=~m!\)!)
          else{ # begin else
               $num.="$c"; # concatenate string (number)
          } # end else
          $i++;
     } # end while($i<$size)
     if(length($num)==0){
          $num=pop @opd; # reduce
     }
     # flush stacks
     while(scalar(@opd)||scalar(@opt)){ # begin while(scalar(@opd)||
scalar(@opt))
          my $opt1=pop @opd; # reduce
          my $locd=pop @opt; # unshift operator to check precedency
# reduce
```

```
          if($locd=~m!\*!){ # begin if($locd=~m!\*!)
               $num=$num*$opt1;
          } # end if($locd=~m!\*!)
          elsif($locd=~m!\/!){ # begin elsif($locd=~m!\/!)
               $num=$opt1/$num;
          } # end elsif($locd=~m!\/!)
          elsif($locd=~m!\+!){ # begin elsif($locd=~m!\+!)
               $num=$num+$opt1;
          } # end elsif($locd=~m!\+!)
          elsif($locd=~m!\-!){ # begin elsif($locd=~m!\-!)
               $num=$opt1-$num;
          } # end elsif($locd=~m!\-!)
          else{ # begin else
               $num=$opt1;
          } # end else
     } # end while(scalar(@opd)||scalar(@opt))

     #print "$mathExpr=>$num [@opt]  [@opd]\n";
     return ($num,"",$mathExpr);
} # end sub calc

sub check{
     my ($res,$rres,$mess,$expr)=@_;#result wanted;result
returned;message error if one

     printf("$expr:");
     if("$res" eq "$rres"){ # begin if("$res" eq "$rres")
          print "OK $res waited and $rres returned\n";
     } # end if("$res" eq "$rres")
     else{
          print "BAD $res waited but $rres returned ; $mess\n";
     }
}
```

Listing of the prgram that can do basic calculation

20110607
This is the listing of tests done and related results:

```
(3+9):OK 12 waited and 12 returned
1-3*9-2:OK -28 waited and -28 returned
1+3*9-2:OK 26 waited and 26 returned
1-3*9+2:BAD 26 waited but -24 returned ;
(1-3)*(9+2):OK -22 waited and -22 returned
(1-3)*(9+2)-1:OK -23 waited and -23 returned
2*((1-3)*(9+2)-1):OK -46 waited and -46 returned
2*((1-3)*(9+2)-1)+3:OK -43 waited and -43 returned
(2+3)-2*((1-3)*(9+2)-1)+3:BAD -38 waited but 54 returned ;
(2+3)-2*((1-3)*(9+2)-1)+3:OK 54 waited and 54 returned
(2+3)*(2*((1-3)*(9+2)-1)+3):BAD 54 waited but -215 returned ;
```

```
1-3:BAD -28 waited but -2 returned ;
kkkk:BAD 1 waited but ERR returned ; err: kkkk
kkkk:OK ERR waited and ERR returned
)(3+9):BAD 0 waited but ERR returned ; err: )(3+9)
)(3+9):OK ERR waited and ERR returned
)(3+9):OK ERR waited and ERR returned
(3+9):BAD ERR waited but 12 returned ;
(3+9):BAD 11 waited but 12 returned ;
(3+9)+1:OK 13 waited and 13 returned
(3*()+9):BAD   waited but ERR returned ; err: (3*()+9)
1+(3+9):OK 13 waited and 13 returned
1-(3+9):BAD 12 waited but -11 returned ;
1-(3+9):BAD -12 waited but -11 returned ;
1+(3+9)+2:BAD 14 waited but 15 returned ;
1+(3+(9+3+9))+2:OK 27 waited and 27 returned
1+(3+9+2)+2:OK 17 waited and 17 returned
2*(3+9):OK 24 waited and 24 returned
1+2*(3+9):BAD 25 waited but ERR returned ; err:
(1+2)*(3+9):OK 36 waited and 36 returned
(1+2*(2+(2+6)*2)*2)*(3+9):OK 876 waited and 876 returned
(1+2*(2+(2+6)*2)*2)*(3+9)-10*3+9:BAD 837 waited but 855 returned ;
3+9:OK 12 waited and 12 returned
3b+9:BAD 0 waited but ERR returned ; err: 3b+9
9+3:OK 12 waited and 12 returned
3-9:OK -6 waited and -6 returned
9-3:OK 6 waited and 6 returned
81/9:OK 9 waited and 9 returned
3*9:OK 27 waited and 27 returned
3*9-1:OK 26 waited and 26 returned
3*9*2:OK 54 waited and 54 returned
4+3*9*2:BAD 58 waited but ERR returned ; err:
2-81/9:BAD -7 waited but ERR returned ; err:
3+9+2*3+8:OK 26 waited and 26 returned
3*9+2*3+8:OK 41 waited and 41 returned
3*9+2+3+2:OK 34 waited and 34 returned
3*9+2-3+2:OK 28 waited and 28 returned
3*9-2+3+2:BAD 24 waited but 30 returned ;
3*9-2-3+2:OK 24 waited and 24 returned
3*(9-2)-3+2:OK 20 waited and 20 returned
3*(9-2)-3+2:OK 20 waited and 20 returned
3*((9-2)-3)+2:OK 14 waited and 14 returned
(2+(4-2)*3+(2-3)*2)*(3+9):OK 72 waited and 72 returned
```

Now this is the program:

```
#!/usr/bin/perl

use strict;

check(12,calc("(3+9)")); # string to analyze
check(-28,calc("1-3*9-2")); # string to analyze
check(26,calc("1+3*9-2")); # string to analyze
check(26,calc("1-3*9+2")); # string to analyze
check(-22,calc("(1-3)*(9+2)")); # string to analyze
check(-23,calc("(1-3)*(9+2)-1")); # string to analyze
check(-46,calc("2*((1-3)*(9+2)-1)")); # string to analyze
check(-43,calc("2*((1-3)*(9+2)-1)+3")); # string to analyze
check(-38,calc("(2+3)-2*((1-3)*(9+2)-1)+3")); # string to analyze
check(54,calc("(2+3)-2*((1-3)*(9+2)-1)+3")); # string to analyze
check(54,calc("(2+3)*(2*((1-3)*(9+2)-1)+3)")); # string to analyze
check(-28,calc("1-3")); # string to analyze
check(1,calc("kkkk")); # string to analyze
check("ERR",calc("kkkk")); # string to analyze
check(0,calc(")(3+9)")); # string to analyze
check("ERR",calc(")(3+9)")); # string to analyze
check("ERR",calc(")(3+9)")); # string to analyze
check("ERR",calc("(3+9)")); # string to analyze
check(11,calc("(3+9)")); # string to analyze
check(13,calc("(3+9)+1")); # string to analyze
check(" ",calc("(3*()+9)")); # string to analyze
check(13,calc("1+(3+9)")); # string to analyze
check(12,calc("1-(3+9)")); # string to analyze
check(-12,calc("1-(3+9)")); # string to analyze
check(14,calc("1+(3+9)+2")); # string to analyze
check(27,calc("1+(3+(9+3+9))+2")); # string to analyze
check(17,calc("1+(3+9+2)+2")); # string to analyze
check(24,calc("2*(3+9)")); # string to analyze
check(25,calc("1+2*(3+9)")); # string to analyze
check(36,calc("(1+2)*(3+9)")); # string to analyze
check(876,calc("(1+2*(2+(2+6)*2)*2)*(3+9)")); # string to analyze
check(837,calc("(1+2*(2+(2+6)*2)*2)*(3+9)-10*3+9")); # string to
analyze
check(12,calc("3+9")); # string to analyze
check(0,calc("3b+9")); # string to analyze
check(12,calc("9+3")); # string to analyze
check(-6,calc("3-9")); # string to analyze
check(6,calc("9-3")); # string to analyze
check(9,calc("81/9")); # string to analyze
check(27,calc("3*9")); # string to analyze
check(26,calc("3*9-1")); # string to analyze
check(54,calc("3*9*2")); # string to analyze
check(58,calc("4+3*9*2")); # string to analyze
check(-7,calc("2-81/9")); # string to analyze
check(26,calc("3+9+2*3+8")); # string to analyze
check(41,calc("3*9+2*3+8")); # string to analyze
check(34,calc("3*9+2+3+2")); # string to analyze
check(28,calc("3*9+2-3+2")); # string to analyze
check(24,calc("3*9-2+3+2")); # string to analyze
check(24,calc("3*9-2-3+2")); # string to analyze
check(20,calc("3*(9-2)-3+2")); # string to analyze
check(20,calc("3*(9-2)-3+2")); # string to analyze
check(14,calc("3*((9-2)-3)+2")); # string to analyze
check(72,calc("(2+(4-2)*3+(2-3)*2)*(3+9)")); # string to analyze



sub calc{ # begin sub calc
     my ($mathExpr)=@_;
     my @opd=(); # Stack for operand
     my @opt=(); # Stack for operator
     my $pbeg=0; # Position at the begining
     my $pend=0; # Position at the end
     my $size=length($mathExpr); # size of string
     my $i=0;
     my $num=();
     my $c=(); # current char
     my $opt1=(); # operand
     my $opt2=(); # operand
     my $opt3=(); # operand
     my $res=(); # reduce
     my $locd1=(); # unshift operator to check precedency # reduce
     my $locd2=(); # unshift operator to check precedency # reduce

     # --------------------------------------------------------------
     # begin sanitary tests

     $mathExpr=~s/[\ ]*//g;# prune out all spaces in expression

     # Checks if expression is all right with characters used
     if($mathExpr!~m/^[()0-9]{1,}([\-\+\*\/]{1,1}[()0-9]{1,})*$/g)
{ # begin if($mathExpr!~m/^[0-9]{1,}([\-\+\*\/]{1,1}[0-9]{1,})*$/g)
          return ("ERR", "err: $mathExpr",$mathExpr);
     } # end if($mathExpr!~m/^[0-9]{1,}([\-\+\*\/]{1,1}[0-9]
{1,})*$/g)

     # checks if some operands are sarting the expression
     if($mathExpr=~m/^[\)\*\/]*/g){ # begin
if($mathExpr=~m/^[\)\*\/]*/g)
          return ("ERR","err: $mathExpr",$mathExpr);
     } # end if($mathExpr=~m/^[\)\*\/]*/g)
     if($mathExpr=~m/\(\)/g){ # begin if($mathExpr=~m/\(\)/g)
          return ("ERR","err: $mathExpr",$mathExpr);
     } # end if($mathExpr=~m/\(\)/g)
     # Counting open and close parenthesis
     $num=0;
     while($i<$size){ # begin while($i<$size)
          $c=substr($mathExpr,$i,1); # gets one character
          if($c=~m/\(/){ # begin if($c=~m/\(/)
               $num++;
          } # end if($c=~m/\(/)
          elsif($c=~m/\)/){ # begin elsif($c=~m/\)/)
               $num--;
          } # end elsif($c=~m/\)/)
          $i++;
     } # end while($i<$size)
     if($num>0){
          return ("ERR","err($num): $mathExpr",$mathExpr);
     } # end if($mathExpr!~m/^[\)\*\/]*/g)
     # end sanitary tests
     # -------------------------------------------------------------

     # -------------------------------------------------------------
     $i=0;# initialise counter
     $num=();
     while($i<$size){ # begin while($i<$size)
          $c=substr($mathExpr,$i,1); # gets one character

          # -----------------------------------------------------
          if($c=~m!\+!){ # begin if($c=~m!\+!)
               if(length($num)>0){ # begin if(length($num)>0)
                    push @opd,$num; # shift number
               } # end if(length($num)>0)
               if(scalar(@opt)==0){ # begin if(scalar(@opt)==0)
                    push @opt,$c; # shift operator
               } # end if(scalar(@opt)==0)
               else{ # begin else
                    my $locd=pop @opt; # unshift operator to check
precedency
                    if($locd=~m!\*!){ # begin if($locd=~m!\*!)
                         $opt1=pop @opd; # reduce
                         $opt2=pop @opd; # reduce
                         $res=$opt1*$opt2; # operation done

                         push @opd,$res; # shift operand
                         push @opt,$c; # shift operator
                    } # end if($locd=~m!\*!)
                    elsif($locd=~m!\/!){ # begin elsif($locd=~m!\/!)
                         $opt1=pop @opd; # reduce
                         $opt2=pop @opd; # reduce
                         $res=$opt2/$opt1; # operation done

                         push @opd,$res; # shift operand
                         push @opt,$c; # shift operator
                    } # end elsif($locd=~m!\/!)
                    elsif($locd=~m!\+!){ # begin elsif($locd=~m!\+!)
                         $opt1=pop @opd; # reduce
                         $opt2=pop @opd; # reduce
                         $res=$opt1+$opt2; # operation done

                         push @opd,$res; # shift operand
                         push @opt,$c; # shift operator
                    } # end elsif($locd=~m!\+!)
                    elsif($locd=~m!\-!){ # begin elsif($locd=~m!\-!)
                         $opt1=pop @opd; # reduce
                         $opt2=pop @opd; # reduce
                         $res=$opt2-$opt1; # operation done

                         push @opd,$res; # shift operand
                         push @opt,$c; # shift operator
                    } # end elsif($locd=~m!\-!)
                    elsif($locd=~m!\(!){ # begin if($locd=~m!\(!)
          #              push @opd,$num; # shift operand
                         push @opt,$locd; # shift operator
                         push @opt,$c; # shift operator
                    } # end if($locd=~m!\(!)
               } # end else
               $num=();
          } # end if($c=~m!\+!)
          # ------------------------------------------------------

          # ------------------------------------------------------
          elsif($c=~m!\-!){ # begin elsif($c=~m!\-!)
               if(length($num)>0){ # begin if(length($num)>0)
                    push @opd,$num; # shift number
               } # end if(length($num)==0)
               if(scalar(@opt)==0){ # begin if(scalar(@opt)==0)
                    push @opt,$c; # shift operator
               } # end if(scalar(@opt)==0)
               else{ # begin else
                    my $locd=pop @opt; # unshift operator to check
     precedency # reduce
                    if($locd=~m!\*!){ # begin if($locd=~m!\*!)
                         $opt1=pop @opd; # reduce
                         $opt2=pop @opd; # reduce
                         $res=$opt1*$opt2; # operation done

                         push @opd,$res; # shift operand
                         push @opt,$c; # shift operator
                    } # end if($locd=~m!\*!)
                    elsif($locd=~m!\/!){ # begin elsif($locd=~m!\/!)
                         $opt1=pop @opd; # reduce
                         $opt2=pop @opd; # reduce
                         $res=$opt1/$opt2;

                         push @opd,$res; # shift operand
                         push @opt,$c; # shift operator
                    } # end elsif($locd=~m!\/!)
                    elsif($locd=~m!\+!){ # begin elsif($locd=~m!\+!)
                         $opt1=pop @opd; # reduce
                         $opt2=pop @opd; # reduce
                         $res=$opt1+$opt2; # operation done

                         push @opd,$res; # shift operand
                         push @opt,$c; # shift operator
                    } # end elsif($locd=~m!\+!)
                    elsif($locd=~m!\-!){ # begin elsif($locd=~m!\-!)
                         $opt1=pop @opd; # reduce
                         $opt2=pop @opd; # reduce
                         $res=$opt2-$opt1; # operation done

                         push @opd,$res; # shift operand
                         push @opt,$c; # shift operator
                    } # end elsif($locd=~m!\-!)
                    elsif($locd=~m!\(!){ # begin elsif($locd=~m!\(!)
                         push @opt,$locd; # shift operator
                         push @opt,$c; # shift operator
                    } # end elsif($locd=~m!\(!)
               } # end else
               $num=();
          } # end elsif($c=~m!\-!)
          # -----------------------------------------------------

          # -----------------------------------------------------
          elsif($c=~m!\*!){ # begin elsif($c=~m!\*!)
               if(length($num)>0){ # begin if(length($num)>0)
                    push @opd,$num; # shift number
               } # end if(length($num)==0)
               if(scalar(@opt)==0){ # begin if(scalar(@opt)==0)
                    push @opt,$c; # shift operator
               } # end if(scalar(@opt)==0)
               else{ # begin else
                    my $locd=pop @opt; # unshift operator to check
precedency # reduce
                    if($locd=~m!\*!){ # begin if($locd=~m!\*!)
                         $opt1=pop @opd; # reduce
                         $opt2=pop @opd; # reduce
                         $res=$opt1*$opt2; # operation done

                         push @opd,$res; # shift operand
                         push @opt,$c; # shift operator
                    } # end if($locd=~m!\*!)
                    elsif($locd=~m!\/!){ # begin elsif($locd=~m!\/!)
                         $opt1=pop @opd; # reduce
                         $opt2=pop @opd; # reduce
                         $res=$opt2/$opt1; # operation done
```

```
                              push @opd,$res; # shift operand
                              push @opt,$c; # shift operator
                         } # end elsif($locd=~m!\/!)
                         elsif($locd=~m!\+!){ # begin elsif($locd=~m!\+!)
                              push @opt,$locd; # shift previous operator
                              push @opt,$c; # shift current operator
                         } # end elsif($locd=~m!\+!)
                         elsif($locd=~m!\-!){ # begin elsif($locd=~m!\-!)
                              push @opt,$locd; # shift previous operator
                              push @opt,$c; # shift current operator
                         } # end elsif($locd=~m!\-!)
                         elsif($locd=~m!\(!){ # begin elsif($locd=~m!\(!)
                              push @opt,$locd; # shift operator
                              push @opt,$c; # shift operator
                         } # end elsif($locd=~m!\(!)
                    } # end else
                    $num=();
               } # end elsif($c=~m!\*!)
               # -----------------------------------------------------

               # -----------------------------------------------------
               elsif($c=~m!\/!){ # begin elsif($c=~m!\/!)
                    if(length($num)>0){ # begin if(length($num)>0)
                         push @opd,$num; # shift number
                    } # end if(length($num)==0)
                    if(scalar(@opt)==0){ # begin if(scalar(@opt)==0)
                         push @opt,$c; # shift operator
                    } # end if(scalar(@opt)==0)
                    else{ # begin else
                         my $locd=pop @opt; # unshift operator to check
precedency # reduce
                         if($locd=~m!\*!){ # begin if($locd=~m!\*!)
                              $opt1=pop @opd; # reduce
                              $opt2=pop @opd; # reduce
                              $res=$opt1*$opt2; # operation done

                              push @opd,$res; # shift operand
                              push @opt,$c; # shift operator
                         } # end if($locd=~m!\*!)
                         elsif($locd=~m!\/!){ # begin elsif($locd=~m!\/!)
                              $opt1=pop @opd; # reduce
                              $opt2=pop @opd; # reduce
                              $res=$opt1/$opt2;

                              push @opd,$res; # shift operand
                              push @opt,$c; # shift operator
                         } # end elsif($locd=~m!\/!)
                         elsif($locd=~m!\+!){ # begin elsif($locd=~m!\+!)
                    push @opt,$locd; # shift previous operator
                    push @opt,$c; # shift current operator
                } # end elsif($locd=~m!\+!)
                elsif($locd=~m!\-!){ # begin elsif($locd=~m!\-!)
                    push @opt,$locd; # shift previous operator
                    push @opt,$c; # shift current operator
                } # end elsif($locd=~m!\-!)
                elsif($locd=~m!\(!){ # begin elsif($locd=~m!\(!)
                    push @opt,$locd; # shift operator
                    push @opt,$c; # shift operator
                } # end elsif($locd=~m!\(!)
            } # end else
            $num=();
        } # end elsif($c=~m!\/!)
        # -----------------------------------------------------

        # -----------------------------------------------------
        elsif($c=~m!\(!){ # begin elsif($c=~m!\(!)
            if(length($num)>0){ # begin if(length($num)>0)
                push @opd,$num; # shift number
            } # end if(length($num)>0)
            if(scalar(@opt)==0){ # begin if(scalar(@opt)==0)
                push @opt,$c; # shift operator
            } # end if(scalar(@opt)==0)
            else{ # begin else
                push @opt,$c; # shift operator
            } # end else
            $num=();
        } # end elsif($c=~m!\(!)
        # -----------------------------------------------------

        # -----------------------------------------------------
        elsif($c=~m!\)!){ # begin elsif($c=~m!\)!)
            if(length($num)>0){ # begin if(length($num)>0)
                push @opd,$num; # shift number
            } # end if(length($num)>0)
            if(scalar(@opt)==0){ # begin if(scalar(@opt)==0)
                return ("ERR","err: $mathExpr",$mathExpr);
            } # end if(scalar(@opt)==0)
            else{ # begin else
                my $locd=pop @opt; # unshift operator to check
precedency # reduce
                # we calculculate till ( is met
                while($locd!~m/\(/){ # begin while($locd!~m/\(/)
                    if($locd=~m!\*!){ # begin if($locd=~m!\*!)
                        $opt1=pop @opd; # reduce
                        $opt2=pop @opd; # reduce
                        $res=$opt1*$opt2; # operation done
```

```
                              push @opd,$res; # shift operand
                              #$opt1=pop @opt; # reduce
                         } # end if($locd=~m!\*!)
                         elsif($locd=~m!\/!){ # begin
elsif($locd=~m!\/!)
                              $opt1=pop @opd; # reduce
                              $opt2=pop @opd; # reduce
                              $res=$opt1/$opt2;

                              push @opd,$res; # shift operand
                         } # end elsif($locd=~m!\/!)
                         elsif($locd=~m!\+!){ # begin
elsif($locd=~m!\+!)
                              $opt1=pop @opd; # reduce
                              $opt2=pop @opd; # reduce
                              $res=$opt1+$opt2; # operation done

                              push @opd,$res; # shift operand
                         } # end elsif($locd=~m!\+!)
                         elsif($locd=~m!\-!){ # begin
elsif($locd=~m!\-!)
                              $opt1=pop @opd; # reduce
                              $opt2=pop @opd; # reduce
                              $res=$opt2-$opt1; # operation done

                              push @opd,$res; # shift operand
                         } # end elsif($locd=~m!\-!)
                         $locd=pop @opt; # unshift operator to check
precedency # reduce
                    } # end while($locd!~m/\(/)
               } # end else
               $num=();
          } # end elsif($c=~m!\)!)
          # -----------------------------------------------------

          # -----------------------------------------------------
          else{ # begin else
               $num.="$c"; # concatenate string (number)
          } # end else
          # -----------------------------------------------------

          $i++; # go to next character
     } # end while($i<$size)

     if(length($num)!=0){
          push @opd,$num; # shift operand
     }

     my $s=scalar(@opt);
     # ----------------------------------------------------
     # flush stacks
     while(scalar(@opd)||scalar(@opt)){ # begin while(scalar(@opd)||
scalar(@opt))
          $opt1=pop @opd; # reduce
          $opt2=pop @opd; # reduce
          $locd1=pop @opt; # unshift operator to check precedency #
reduce
          if($s==2){ # begin if($s==2)
               if($locd1=~m/[\+\-]/){
               # begin if($locd1=~m/[\+\-]/)
                    $locd2=pop @opt; # unshift operator to check
precedency # reduce
                    #
----------------------------------------------------
                    if($locd2=~m!\+!){ # begin if($locd2=~m!\+!)
                         $opt3=pop @opd; # reduce
                         $res=$opt3+$opt2;

                         push @opd,$res; # shift result
                         push @opd,$opt1; # shift result
                         push @opt,$locd1; # shift operand
                    } # end if($locd2=~m!\+!)
                    #
----------------------------------------------------

                    #
----------------------------------------------------
                    elsif($locd2=~m!\-!){ # begin
elsif($locd2=~m!\-!)
                         $opt3=pop @opd; # reduce
                         $res=$opt3-$opt2;

                         push @opd,$res; # shift result
                         push @opd,$opt1; # shift result
                         push @opt,$locd1; # shift operand
                    } # end elsif($locd2=~m!\-!)
                    #
----------------------------------------------------
                    # restablish context
                    $opt1=pop @opd; # reduce
                    $opt2=pop @opd; # reduce
                    $locd1=pop @opt; # unshift operator to check
precedency # reduce
               }
               # end if($locd1=~m/[\+\-]/)
          } # end if($s==2)

          # ----------------------------------------------------
          if($locd1=~m!\*!){ # begin if($locd1=~m!\*!)
               $res=$opt1*$opt2;
          } # end if($locd1=~m!\*!)
          # ------------------------------------------------------

          # ------------------------------------------------------
          elsif($locd1=~m!\/!){ # begin elsif($locd1=~m!\/!)
               $res=$opt2/$opt1;
          } # end elsif($locd1=~m!\/!)
          # ------------------------------------------------------

          # ------------------------------------------------------
          elsif($locd1=~m!\+!){ # begin elsif($locd1=~m!\+!)
               $res=$opt1+$opt2;
          } # end elsif($locd1=~m!\+!)
          # ------------------------------------------------------

          # ------------------------------------------------------
          elsif($locd1=~m!\-!){ # begin elsif($locd1=~m!\-!)
               $res=$opt2-$opt1;
          } # end elsif($locd1=~m!\-!)
          # ------------------------------------------------------

          # ------------------------------------------------------
          else{ # begin else
               if(length($opt2)==0){
                    $res=$opt1;
               }else{
                    $res=$opt2;
               }
          } # end else
          # ------------------------------------------------------
     } # end while(scalar(@opd)||scalar(@opt))
     # ------------------------------------------------------

     if(length($res)==0){
          if(scalar(@opd)>0){
               $res=pop @opd;
          }else{
               return ("ERR","err:",$mathExpr);
          }
     }
     return ($res,"",$mathExpr);
} # end sub calc

sub check{
     my ($res,$rres,$mess,$expr)=@_;#result wanted;result
returned;message error if one
```

```
     printf("$expr:");
     if("$res" eq "$rres"){ # begin if("$res" eq "$rres")
          print "OK $res waited and $rres returned\n";
     } # end if("$res" eq "$rres")
     else{
          print "BAD $res waited but $rres returned ; $mess\n";
     }
}
```

20110608
Due to the add of extra use warning in the proram some warnings shown up. Defined reserved word had to be added in the program. Tests results seemed not changed. The library to make measures on timing was added Time::HiRes.

```
use strict;
use warnings;
use Time::HiRes qw(usleep ualarm gettimeofday tv_interval);
```

Libraries added in bold characters

this is the new code:

```
#!/usr/bin/perl

use strict;
use warnings;
use Time::HiRes qw(usleep ualarm gettimeofday tv_interval);

my ($s,$m)=gettimeofday();
check(12,calc("(3+9)")); # string to analyze
($s,$m)=gettimeofday();
check(-28,calc("1-3*9-2")); # string to analyze
($s,$m)=gettimeofday();
check(26,calc("1+3*9-2")); # string to analyze
($s,$m)=gettimeofday();
check(26,calc("1-3*9+2")); # string to analyze
($s,$m)=gettimeofday();
check(-22,calc("(1-3)*(9+2)")); # string to analyze
($s,$m)=gettimeofday();
check(-23,calc("(1-3)*(9+2)-1")); # string to analyze
($s,$m)=gettimeofday();
check(-46,calc("2*((1-3)*(9+2)-1)")); # string to analyze
($s,$m)=gettimeofday();
check(-43,calc("2*((1-3)*(9+2)-1)+3")); # string to analyze
($s,$m)=gettimeofday();
check(-38,calc("(2+3)-2*((1-3)*(9+2)-1)+3")); # string to analyze
($s,$m)=gettimeofday();
check(54,calc("(2+3)-2*((1-3)*(9+2)-1)+3")); # string to analyze
($s,$m)=gettimeofday();
```

```
check(54,calc("(2+3)*(2*((1-3)*(9+2)-1)+3)")); # string to analyze
($s,$m)=gettimeofday();
check(-28,calc("1-3")); # string to analyze
($s,$m)=gettimeofday();
check(1,calc("kkkk")); # string to analyze
($s,$m)=gettimeofday();
check("ERR",calc("kkkk")); # string to analyze
($s,$m)=gettimeofday();
check(0,calc(")(3+9)")); # string to analyze
($s,$m)=gettimeofday();
check("ERR",calc(")(3+9)")); # string to analyze
($s,$m)=gettimeofday();
check("ERR",calc(")(3+9)")); # string to analyze
($s,$m)=gettimeofday();
check("ERR",calc("(3+9)")); # string to analyze
($s,$m)=gettimeofday();
check(11,calc("(3+9)")); # string to analyze
($s,$m)=gettimeofday();
check(13,calc("(3+9)+1")); # string to analyze
($s,$m)=gettimeofday();
check(" ",calc("(3*()+9)")); # string to analyze
($s,$m)=gettimeofday();
check(13,calc("1+(3+9)")); # string to analyze
($s,$m)=gettimeofday();
check(12,calc("1-(3+9)")); # string to analyze
($s,$m)=gettimeofday();
check(-12,calc("1-(3+9)")); # string to analyze
($s,$m)=gettimeofday();
check(14,calc("1+(3+9)+2")); # string to analyze
($s,$m)=gettimeofday();
check(27,calc("1+(3+(9+3+9))+2")); # string to analyze
($s,$m)=gettimeofday();
check(17,calc("1+(3+9+2)+2")); # string to analyze
($s,$m)=gettimeofday();
check(24,calc("2*(3+9)")); # string to analyze
($s,$m)=gettimeofday();
check(25,calc("1+2*(3+9)")); # string to analyze
($s,$m)=gettimeofday();
check(36,calc("(1+2)*(3+9)")); # string to analyze
($s,$m)=gettimeofday();
check(876,calc("(1+2*(2+(2+6)*2)*2)*(3+9)")); # string to analyze
($s,$m)=gettimeofday();
check(837,calc("(1+2*(2+(2+6)*2)*2)*(3+9)-10*3+9")); # string to
analyze
($s,$m)=gettimeofday();
check(12,calc("3+9")); # string to analyze
($s,$m)=gettimeofday();
check(0,calc("3b+9")); # string to analyze
($s,$m)=gettimeofday();
```

```
check(12,calc("9+3")); # string to analyze
($s,$m)=gettimeofday();
check(-6,calc("3-9")); # string to analyze
($s,$m)=gettimeofday();
check(6,calc("9-3")); # string to analyze
($s,$m)=gettimeofday();
check(9,calc("81/9")); # string to analyze
($s,$m)=gettimeofday();
check(27,calc("3*9")); # string to analyze
($s,$m)=gettimeofday();
check(26,calc("3*9-1")); # string to analyze
($s,$m)=gettimeofday();
check(54,calc("3*9*2")); # string to analyze
($s,$m)=gettimeofday();
check(58,calc("4+3*9*2")); # string to analyze
($s,$m)=gettimeofday();
check(-7,calc("2-81/9")); # string to analyze
($s,$m)=gettimeofday();
check(26,calc("3+9+2*3+8")); # string to analyze
($s,$m)=gettimeofday();
check(41,calc("3*9+2*3+8")); # string to analyze
($s,$m)=gettimeofday();
check(34,calc("3*9+2+3+2")); # string to analyze
($s,$m)=gettimeofday();
check(28,calc("3*9+2-3+2")); # string to analyze
($s,$m)=gettimeofday();
check(24,calc("3*9-2+3+2")); # string to analyze
($s,$m)=gettimeofday();
check(24,calc("3*9-2-3+2")); # string to analyze
($s,$m)=gettimeofday();
check(20,calc("3*(9-2)-3+2")); # string to analyze
($s,$m)=gettimeofday();
check(20,calc("3*(9-2)-3+2")); # string to analyze
($s,$m)=gettimeofday();
check(14,calc("3*((9-2)-3)+2")); # string to analyze
($s,$m)=gettimeofday();
check(72,calc("(2+(4-2)*3+(2-3)*2)*(3+9)")); # string to analyze



sub calc{ # begin sub calc
     my ($mathExpr)=@_;
     my @opd=(); # Stack for operand
     my @opt=(); # Stack for operator
     my $pbeg=0; # Position at the begining
     my $pend=0; # Position at the end
     my $size=length($mathExpr); # size of string
     my $i=0;
     my $num=();
     my $c=(); # current char
     my $opt1=(); # operand
     my $opt2=(); # operand
     my $opt3=(); # operand
     my $res=(); # reduce
     my $locd1=""; # unshift operator to check precedency # reduce
     my $locd2=""; # unshift operator to check precedency # reduce

     # --------------------------------------------------------------
     # begin sanitary tests

     $mathExpr=~s/[\ ]*//g;# prune out all spaces in expression

     # Checks if expression is all right with characters used
     if($mathExpr!~m/^[()0-9]{1,}([\-\+\*\/]{1,1}[()0-9]{1,})*$/g)
{ # begin if($mathExpr!~m/^[0-9]{1,}([\-\+\*\/]{1,1}[0-9]{1,})*$/g)
          return ("ERR", "err: $mathExpr",$mathExpr);
     } # end if($mathExpr!~m/^[0-9]{1,}([\-\+\*\/]{1,1}[0-9]
{1,})*$/g)

     # checks if some operands are sarting the expression
     if($mathExpr=~m/^[\)\*\/]*/g){ # begin
if($mathExpr=~m/^[\)\*\/]*/g)
          return ("ERR","err: $mathExpr",$mathExpr);
     } # end if($mathExpr=~m/^[\)\*\/]*/g)
     if($mathExpr=~m/\(\)/g){ # begin if($mathExpr=~m/\(\)/g)
          return ("ERR","err: $mathExpr",$mathExpr);
     } # end if($mathExpr=~m/\(\)/g)
     # Counting open and close parenthesis
     $num=0;
     while($i<$size){ # begin while($i<$size)
          $c=substr($mathExpr,$i,1); # gets one character
          if($c=~m/\(/){ # begin if($c=~m/\(/)
               $num++;
          } # end if($c=~m/\(/)
          elsif($c=~m/\)/){ # begin elsif($c=~m/\)/)
               $num--;
          } # end elsif($c=~m/\)/)
          $i++;
     } # end while($i<$size)
     if($num>0){
          return ("ERR","err($num): $mathExpr",$mathExpr);
     } # end if($mathExpr!~m/^[\)\*\/]*/g)
     # end sanitary tests
     # --------------------------------------------------------------

     # --------------------------------------------------------------
     $i=0;# initialise counter
     $num="";
     while($i<$size){ # begin while($i<$size)
          $c=substr($mathExpr,$i,1); # gets one character

          # -------------------------------------------------------
          if($c=~m!\+!){ # begin if($c=~m!\+!)
               if(length($num)>0){ # begin if(length($num)>0)
                    push @opd,$num; # shift number
               } # end if(length($num)>0)
               if(scalar(@opt)==0){ # begin if(scalar(@opt)==0)
                    push @opt,$c; # shift operator
               } # end if(scalar(@opt)==0)
               else{ # begin else
                    my $locd=pop @opt; # unshift operator to check
precedency
                    if($locd=~m!\*!){ # begin if($locd=~m!\*!)
                         $opt1=pop @opd; # reduce
                         $opt2=pop @opd; # reduce
                         $res=$opt1*$opt2; # operation done

                         push @opd,$res; # shift operand
                         push @opt,$c; # shift operator
                    } # end if($locd=~m!\*!)
                    elsif($locd=~m!\/!){ # begin elsif($locd=~m!\/!)
                         $opt1=pop @opd; # reduce
                         $opt2=pop @opd; # reduce
                         $res=$opt2/$opt1; # operation done

                         push @opd,$res; # shift operand
                         push @opt,$c; # shift operator
                    } # end elsif($locd=~m!\/!)
                    elsif($locd=~m!\+!){ # begin elsif($locd=~m!\+!)
                         $opt1=pop @opd; # reduce
                         $opt2=pop @opd; # reduce
                         $res=$opt1+$opt2; # operation done

                         push @opd,$res; # shift operand
                         push @opt,$c; # shift operator
                    } # end elsif($locd=~m!\+!)
                    elsif($locd=~m!\-!){ # begin elsif($locd=~m!\-!)
                         $opt1=pop @opd; # reduce
                         $opt2=pop @opd; # reduce
                         $res=$opt2-$opt1; # operation done

                         push @opd,$res; # shift operand
                         push @opt,$c; # shift operator
                    } # end elsif($locd=~m!\-!)
                    elsif($locd=~m!\(!){ # begin if($locd=~m!\(!)
          #              push @opd,$num; # shift operand
                         push @opt,$locd; # shift operator
                         push @opt,$c; # shift operator
                    } # end if($locd=~m!\(!)
               } # end else
               $num="";
          } # end if($c=~m!\+!)
          # -----------------------------------------------------

          # -----------------------------------------------------
          elsif($c=~m!\-!){ # begin elsif($c=~m!\-!)
               if(length($num)>0){ # begin if(length($num)>0)
                    push @opd,$num; # shift number
               } # end if(length($num)==0)
               if(scalar(@opt)==0){ # begin if(scalar(@opt)==0)
                    push @opt,$c; # shift operator
               } # end if(scalar(@opt)==0)
               else{ # begin else
                    my $locd=pop @opt; # unshift operator to check
precedency # reduce
                    if($locd=~m!\*!){ # begin if($locd=~m!\*!)
                         $opt1=pop @opd; # reduce
                         $opt2=pop @opd; # reduce
                         $res=$opt1*$opt2; # operation done

                         push @opd,$res; # shift operand
                         push @opt,$c; # shift operator
                    } # end if($locd=~m!\*!)
                    elsif($locd=~m!\/!){ # begin elsif($locd=~m!\/!)
                         $opt1=pop @opd; # reduce
                         $opt2=pop @opd; # reduce
                         $res=$opt1/$opt2;

                         push @opd,$res; # shift operand
                         push @opt,$c; # shift operator
                    } # end elsif($locd=~m!\/!)
                    elsif($locd=~m!\+!){ # begin elsif($locd=~m!\+!)
                         $opt1=pop @opd; # reduce
                         $opt2=pop @opd; # reduce
                         $res=$opt1+$opt2; # operation done

                         push @opd,$res; # shift operand
                         push @opt,$c; # shift operator
                    } # end elsif($locd=~m!\+!)
                    elsif($locd=~m!\-!){ # begin elsif($locd=~m!\-!)
                         $opt1=pop @opd; # reduce
                         $opt2=pop @opd; # reduce
                         $res=$opt2-$opt1; # operation done

                         push @opd,$res; # shift operand
                         push @opt,$c; # shift operator
                    } # end elsif($locd=~m!\-!)
                    elsif($locd=~m!\(!){ # begin elsif($locd=~m!\(!)
                         push @opt,$locd; # shift operator
                         push @opt,$c; # shift operator
                    } # end elsif($locd=~m!\(!)
               } # end else
               $num="";
          } # end elsif($c=~m!\-!)
          # ---------------------------------------------------

          # ---------------------------------------------------
          elsif($c=~m!\*!){ # begin elsif($c=~m!\*!)
               if(length($num)>0){ # begin if(length($num)>0)
                    push @opd,$num; # shift number
               } # end if(length($num)==0)
               if(scalar(@opt)==0){ # begin if(scalar(@opt)==0)
                    push @opt,$c; # shift operator
               } # end if(scalar(@opt)==0)
               else{ # begin else
                    my $locd=pop @opt; # unshift operator to check
precedency # reduce
                    if($locd=~m!\*!){ # begin if($locd=~m!\*!)
                         $opt1=pop @opd; # reduce
                         $opt2=pop @opd; # reduce
                         $res=$opt1*$opt2; # operation done

                         push @opd,$res; # shift operand
                         push @opt,$c; # shift operator
                    } # end if($locd=~m!\*!)
                    elsif($locd=~m!\/!){ # begin elsif($locd=~m!\/!)
                         $opt1=pop @opd; # reduce
                         $opt2=pop @opd; # reduce
                         $res=$opt2/$opt1; # operation done

                         push @opd,$res; # shift operand
                         push @opt,$c; # shift operator
                    } # end elsif($locd=~m!\/!)
                    elsif($locd=~m!\+!){ # begin elsif($locd=~m!\+!)
                         push @opt,$locd; # shift previous operator
                         push @opt,$c; # shift current operator
                    } # end elsif($locd=~m!\+!)
                    elsif($locd=~m!\-!){ # begin elsif($locd=~m!\-!)
                         push @opt,$locd; # shift previous operator
                         push @opt,$c; # shift current operator
                    } # end elsif($locd=~m!\-!)
                    elsif($locd=~m!\(!){ # begin elsif($locd=~m!\(!)
                         push @opt,$locd; # shift operator
                         push @opt,$c; # shift operator
                    } # end elsif($locd=~m!\(!)
               } # end else
               $num="";
          } # end elsif($c=~m!\*!)
          # -----------------------------------------------------

          # -----------------------------------------------------
          elsif($c=~m!\/!){ # begin elsif($c=~m!\/!)
               if(length($num)>0){ # begin if(length($num)>0)
                    push @opd,$num; # shift number
               } # end if(length($num)==0)
               if(scalar(@opt)==0){ # begin if(scalar(@opt)==0)
                    push @opt,$c; # shift operator
               } # end if(scalar(@opt)==0)
               else{ # begin else
                    my $locd=pop @opt; # unshift operator to check
precedency # reduce
                    if($locd=~m!\*!){ # begin if($locd=~m!\*!)
                         $opt1=pop @opd; # reduce
                         $opt2=pop @opd; # reduce
                         $res=$opt1*$opt2; # operation done

                         push @opd,$res; # shift operand
                         push @opt,$c; # shift operator
                    } # end if($locd=~m!\*!)
                    elsif($locd=~m!\/!){ # begin elsif($locd=~m!\/!)
                         $opt1=pop @opd; # reduce
                         $opt2=pop @opd; # reduce
                         $res=$opt1/$opt2;

                         push @opd,$res; # shift operand
                         push @opt,$c; # shift operator
                    } # end elsif($locd=~m!\/!)
                    elsif($locd=~m!\+!){ # begin elsif($locd=~m!\+!)
                         push @opt,$locd; # shift previous operator
                         push @opt,$c; # shift current operator
                    } # end elsif($locd=~m!\+!)
                    elsif($locd=~m!\-!){ # begin elsif($locd=~m!\-!)
                         push @opt,$locd; # shift previous operator
                         push @opt,$c; # shift current operator
                    } # end elsif($locd=~m!\-!)
                    elsif($locd=~m!\(!){ # begin elsif($locd=~m!\(!)
                         push @opt,$locd; # shift operator
                         push @opt,$c; # shift operator
                    } # end elsif($locd=~m!\(!)
               } # end else
               $num="";
          } # end elsif($c=~m!\/!)
          # -----------------------------------------------------
          # ----------------------------------------------------
          elsif($c=~m!\(!){ # begin elsif($c=~m!\(!)
               if(length($num)>0){ # begin if(length($num)>0)
                    push @opd,$num; # shift number
               } # end if(length($num)>0)
               if(scalar(@opt)==0){ # begin if(scalar(@opt)==0)
                    push @opt,$c; # shift operator
               } # end if(scalar(@opt)==0)
               else{ # begin else
                    push @opt,$c; # shift operator
               } # end else
               $num="";
          } # end elsif($c=~m!\(!)
          # ----------------------------------------------------

          # ----------------------------------------------------
          elsif($c=~m!\)!){ # begin elsif($c=~m!\)!)
               if(length($num)>0){ # begin if(length($num)>0)
                    push @opd,$num; # shift number
               } # end if(length($num)>0)
               if(scalar(@opt)==0){ # begin if(scalar(@opt)==0)
                    return ("ERR","err: $mathExpr",$mathExpr);
               } # end if(scalar(@opt)==0)
               else{ # begin else
                    my $locd=pop @opt; # unshift operator to check
precedency # reduce
                    # we calculculate till ( is met
                    while($locd!~m/\(/){ # begin while($locd!~m/\(/)
                         if($locd=~m!\*!){ # begin if($locd=~m!\*!)
                              $opt1=pop @opd; # reduce
                              $opt2=pop @opd; # reduce
                              $res=$opt1*$opt2; # operation done

                              push @opd,$res; # shift operand
                              #$opt1=pop @opt; # reduce
                         } # end if($locd=~m!\*!)
                         elsif($locd=~m!\/!){ # begin
elsif($locd=~m!\/!)
                              $opt1=pop @opd; # reduce
                              $opt2=pop @opd; # reduce
                              $res=$opt1/$opt2;

                              push @opd,$res; # shift operand
                         } # end elsif($locd=~m!\/!)
                         elsif($locd=~m!\+!){ # begin
elsif($locd=~m!\+!)
                              $opt1=pop @opd; # reduce
                              $opt2=pop @opd; # reduce
                              $res=$opt1+$opt2; # operation done
                              push @opd,$res; # shift operand
                         } # end elsif($locd=~m!\+!)
                         elsif($locd=~m!\-!){ # begin
elsif($locd=~m!\-!)
                              $opt1=pop @opd; # reduce
                              $opt2=pop @opd; # reduce
                              $res=$opt2-$opt1; # operation done

                              push @opd,$res; # shift operand
                         } # end elsif($locd=~m!\-!)
                         $locd=pop @opt; # unshift operator to check
precedency # reduce
                    } # end while($locd!~m/\(/)
               } # end else
               $num="";
          } # end elsif($c=~m!\)!)
          # -----------------------------------------------------

          # -----------------------------------------------------
          else{ # begin else
               $num.="$c"; # concatenate string (number)
          } # end else
          # -----------------------------------------------------

          $i++; # go to next character
     } # end while($i<$size)

     if(length($num)!=0){
          push @opd,$num; # shift operand
     }

     my $s=scalar(@opt);
     $locd1="";
     $locd2="";
     # -----------------------------------------------------
     # flush stacks
     $opt1=$opt2="";
     $locd1=$locd2="";
     while(scalar(@opd)||scalar(@opt)){ # begin while(scalar(@opd)||
scalar(@opt))
          $opt1=$opt2="";
          $opt1=pop @opd; # reduce
          $opt2=pop @opd; # reduce
          $locd1=pop @opt; # unshift operator to check precedency #
reduce
          if($s==2){ # begin if($s==2)
               if($locd1=~m/[\+\-]/){ # begin if($locd1=~m/[\+\-]/)
                    if(scalar(@opt)>0){
                         $locd2=pop @opt; # unshift operator to
check precedency # reduce
                         #
-----------------------------------------------------
                         if($locd2=~m!\+!){ # begin if($locd2=~m!\
+!)
                              $opt3=pop @opd; # reduce
                              $res=$opt3+$opt2;

                              push @opd,$res; # shift result
                              push @opd,$opt1; # shift result
                              push @opt,$locd1; # shift operand
                         } # end if($locd2=~m!\+!)
                         #
-----------------------------------------------------

                         #
-----------------------------------------------------
                         elsif($locd2=~m!\-!){ # begin
elsif($locd2=~m!\-!)
                              $opt3=pop @opd; # reduce
                              $res=$opt3-$opt2;

                              push @opd,$res; # shift result
                              push @opd,$opt1; # shift result
                              push @opt,$locd1; # shift operand
                         } # end elsif($locd2=~m!\-!)
                         #
-----------------------------------------------------
                         # restablish context
                         $opt1=pop @opd; # reduce
                         $opt2=pop @opd; # reduce
                         $locd1=pop @opt; # unshift operator to
check precedency # reduce
          #if(length("$opt2")==0){$opt2=0;}
          #if(length("$opt1")==0){$opt1=0;}
                    }
               } # end if($locd1=~m/[\+\-]/)
          } # end if($s==2)
          if(!defined($opt1)){
               return ("ERR","err:",$mathExpr);
          }
          if(!defined($opt2)){
               return ("ERR","err:",$mathExpr);
          }

          # -----------------------------------------------------
          if($locd1=~m!\*!){ # begin if($locd1=~m!\*!)
               $res=$opt1*$opt2;
          } # end if($locd1=~m!\*!)
          # -----------------------------------------------------

          # -----------------------------------------------------
          elsif($locd1=~m!\/!){ # begin elsif($locd1=~m!\/!)
               $res=$opt2/$opt1;
          } # end elsif($locd1=~m!\/!)
          # -----------------------------------------------------

          # -----------------------------------------------------
          elsif($locd1=~m!\+!){ # begin elsif($locd1=~m!\+!)
               $res=$opt1+$opt2;
          } # end elsif($locd1=~m!\+!)
          # -----------------------------------------------------

          # -----------------------------------------------------
          elsif($locd1=~m!\-!){ # begin elsif($locd1=~m!\-!)
               $res=$opt2-$opt1;
          } # end elsif($locd1=~m!\-!)
          # -----------------------------------------------------

          # -----------------------------------------------------
          else{ # begin else
               if(length($opt2)==0){
                    $res=$opt1;
               }else{
                    $res=$opt2;
               }
          } # end else
          # -----------------------------------------------------
     } # end while(scalar(@opd)||scalar(@opt))
     # -----------------------------------------------------

     if(length($res)==0){
          if(scalar(@opd)>0){
               $res=pop @opd;
          }else{
               return ("ERR","err:",$mathExpr);
          }
     }
     return ($res,"",$mathExpr);
} # end sub calc

sub check{
     my ($res,$rres,$mess,$expr)=@_;#result wanted;result
returned;message error if one
     my $s=length($expr);
     printf("$expr:");
     if("$res" eq "$rres"){ # begin if("$res" eq "$rres")
```

```
          print "OK $res $rres $expr $s\n";
     } # end if("$res" eq "$rres")
     else{
          print "BAD $res $rres $expr $s\n";
          #print "BAD $res waited but $rres returned ; $mess\n";
     }
}
```