PUSH(s,top,x)

```
1.[check for stack overflow]
    if top>=n
    Write('stack overflow')
    Return
2. [increment top]
    top=top+1;
3.[insert element]
    s[top]=x
4.[finished]
    return
```

POP(s,top)

```
1.[check for stack underflow]
            If top = 0
            write "stack underflow on pop"
            exit
2. [decrement pointer]
            top = top – 1
3. [return former top element of stack]
            Return (s[top + 1])
```

PEEP(s,top,i)

```
1.[check for stack underflow]
        If (top – i + 1) <= 0
        write "stack underflow on peep"
        exit
2. [return ith element from top of the stack]
        return(s[top – i + 1])
```

CHANGE(s,top,x,i)

```
1.[check for stack underflow]
    If (top – i + 1) <= 0
    write “stack underflow on change”
    exit
2. [return ith element from top of the stack]
    s[top – i + 1] = x
3. [return]
    return
```

RECOGNIZE

```
1.   top = 1
     s[top] = 'c'
2.   next = nextchar[string]
     repeat while next ≠ 'c'
          if next = ' '
               then write "invalid string"
               exit
          else call push(s, top,next)
               next = nextchar[string]
```

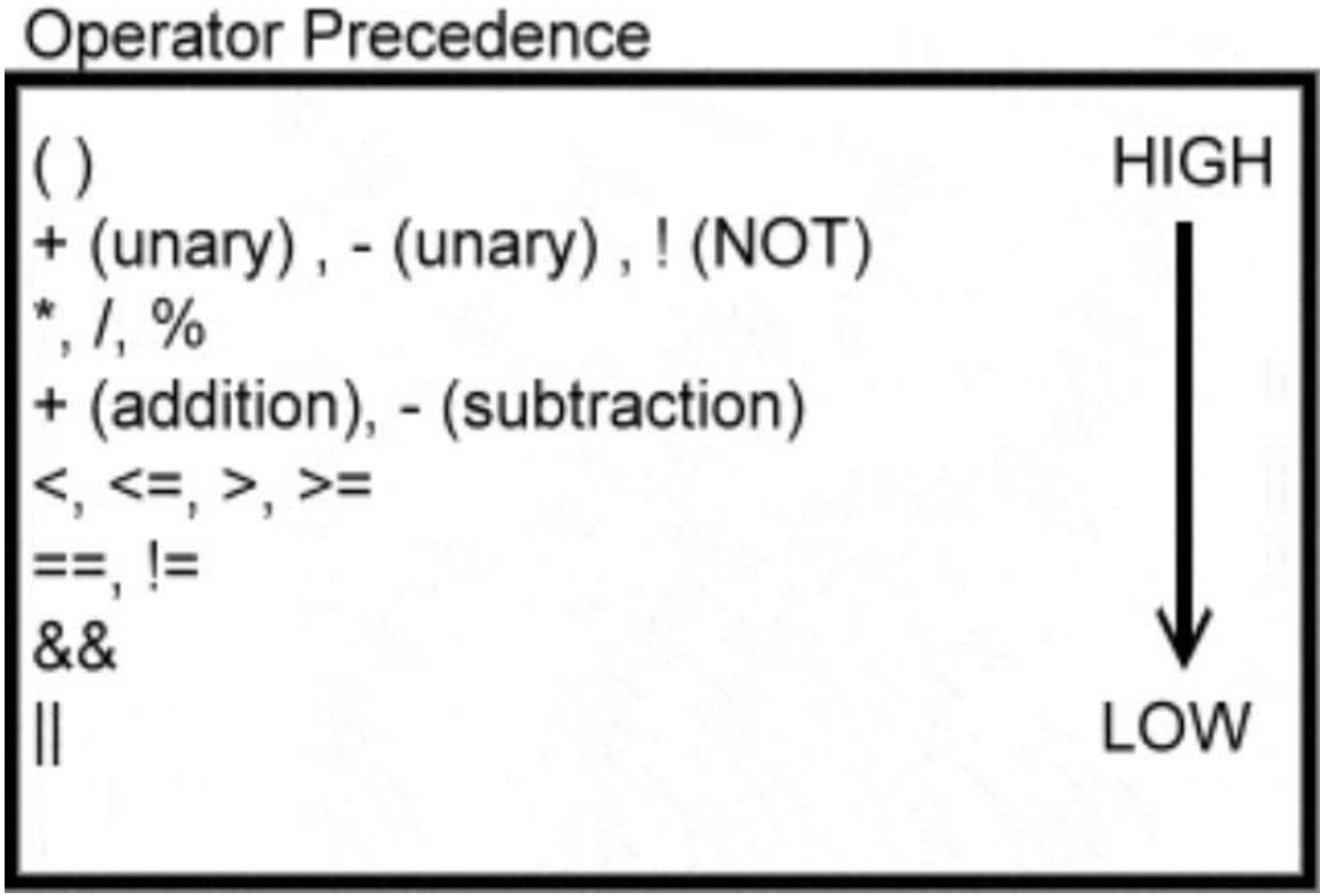
RECOGNIZE(cont…)

```
3.  Repeat while s[top] ≠ ‘c’
        next = nextchar(string)
        x = pop(s,top)
        if next ≠ x
            write “invalid string”
            exit
4.  If next = ‘ ‘
        write “valid string”
    else write “invalid string”
5.  exit
```

PRECEDENCE TABLE(for unparenthesized suffix)

SYMBOL	PRECEDENCE (f)	RANK (r)
+,-	1	-1
*,/	2	-1
a,b,c..	3	1
#	0	-

Operator precedence



Operator Precedence

( )
+ (unary) , - (unary) , ! (NOT)
*, /, %
+ (addition), - (subtraction)
<, <=, >, >=
==, !=
&&
||

UNPARENTHESIZED SUFFIX/POSTFIX

```
1.  top = 1
    s[top] = ‘#’
2.  Polish = ‘ ‘
    rank = 0
3.  next = nextchar[infix]
4.  Repeat thru step 6 while next ≠ ‘#’
```

UNPARENTHESIZED SUFFIX/POSTFIX(cont..)

```
5.  Repeat while f(next) <= f(s[top])
         temp=pop(s,top)
         polish = polish + temp
         rank = rank + r(temp)
         if rank < 1
             write "invalid"
                 Exit
6.   Call push(s,top,next)
     next = nextchar(infix)
```

UNPARENTHESIZED SUFFIX/POSTFIX(cont…)

```
7.  Repeat while s[top] ≠ '#'
          temp = pop(s,top)
          polish = polish + temp
          rank = rank + r(temp)
          if rank < 1
               write "invalid"
               exit
8.  If rank = 1
          write "valid"
    else
          write "invalid"
          exit
```

Stack trace of A+B#

INFIX	CONTENTS OF STACK	Postfix	RANK
	#		
A	#A		
+	#+	A	1
B	#+B	A	1
#	-	AB+	1

PRECEDENCE TABLE(for parenthesized suffix

Symbol	Input precedence function(f)	Stack precedence function (g)	Rank function (r)
+,-	1	2	-1
*,/	3	4	-1
↑	6	5	-1
Variables	7	8	1
(	9	0	-
)	0	-	-

REVPOL

```
1.   top = 1
     s[top] = '('
2.   Polish = ' '
     rank = 0
3.   next = nextchar[infix]
4.   Repeat thru step 7 while next ≠ ''
```

REVPOL(cont…)

```
5.   If top < 1
          write "invalid" exit
     repeat while f(next) < g(s[top])
          temp = pop(s,top)
          polish = polish + temp
          rank = rank + r(temp)
          if rank < 1
               write "invalid"
               exit
```

REVPOL(cont…)

```
6.   If $f(next) \neq g(s[top])$
          call push(s,top,next)
     else
          pop(s,top)
7.   next = nextchar(infix)
8.   If $top \neq 0$ or $rank \neq 1$
          write “invalid”
     else
          write “valid”
          exit
```

Applications of stack

- Recursion
- Polish expressions
- Tower of Hanoi

Tower of Hanoi

- All disks must be on tower at all times.
- Only the disk on top of any tower can be moved.
- Only small disks are allowed to lye on top of larger disks.
- You are allowed as many moves as necessary , but obviously less is better!