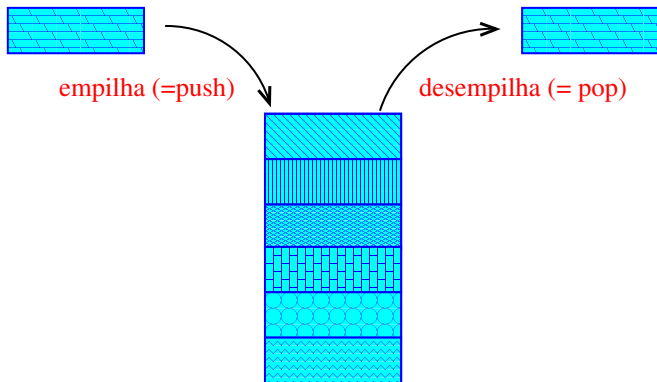


Melhores momentos

AULA 10

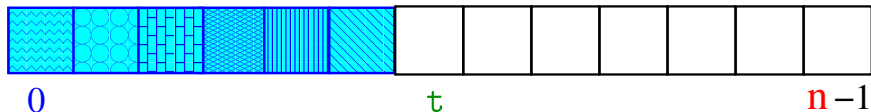
Pilhas

Uma **pilha** (= *stack*) é uma lista (=sequência) dinâmica em que todas as operações (*inserções*, *remoções* e *consultas*) são feitas em uma mesma extremidade chamada de **topo**.



Implementação em um vetor

A pilha será armazenada em um vetor $s[0 \dots n-1]$.



O índice t indica o **topo** ($=top$) da pilha.

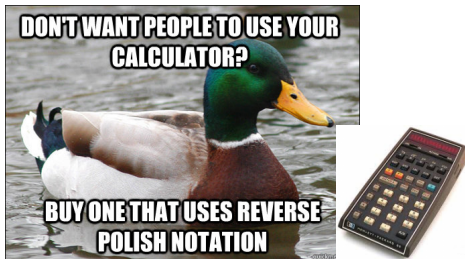
Esta é a **primeira posição vaga** da pilha.

A pilha está **vazia** se “ $t == 0$ ”.

A pilha está **cheia** se “ $t == n$ ”.

AULA 11

Notação polonesa (reversa)



Fonte: <http://www.quickmeme.com/> e
<http://danicollinmotion.com/>

PF 6.3

<http://www.ime.usp.br/~pf/algoritmos/aulas/pilha.html>

http://en.wikipedia.org/wiki/RPN_calculator

http://en.wikipedia.org/wiki/Shunting-yard_algorithm

Notação polonesa

Usualmente os operadores são escritos **entre** os operandos como em

$$(A + B) * D + E / (F + A * D) + C$$

Essa é a chamada **notação infixa**.

Na **notação polonesa** ou **posfixa** os operadores são escritos **depois** dos operandos

$$A B + D * E F A D * + / + C +$$

Notação polonesa

Problema: Traduzir para **notação posfixa** a expressão infixa armazenada em uma cadeia de caracteres **inf**. Suponha que na expressão só ocorrem os **operadores binários** '+', '-', '*', '/', além de '(', e ')'

infixa	posfixa
$A+B*C$	$ABC*+$
$A*(B+C)/D-E$	$ABC+*D/E-$
$A+B*(C-D*(E-F)-G*H)-I*3$	$ABCDEF-*-*GH*-*+I3*-$
$A+B*C/D*E-F$	$ABC*D/E*+F-$
$A+(B-(C+(D-(E+F))))$	$ABCDEF+-+--+$
$A*(B+(C*(D+(E*(F+G))))$	$ABCDEF*G+*+*+*$

Simulação

inf = expressão **infixa**

s = pilha

posf = expressão **posfixa**

Simulação

inf [0 . . i-1]	s [0 . . t-1]	posf [0 . . j-1]
((
(A	(A
(A*	(*	A
(A*((*	A
(A*(B	(*	AB
(A*(B*	(*	AB
(A*(B*C	(*	ABC
(A*(B*C+	(*	ABC*
(A*(B*C+D	(*	ABC*D
(A*(B*C+D)	(*	ABC*D+
(A*(B*C+D))		ABC*D+*

Infixa para posfixa

Recebe uma expressão infixada `inf` e devolve a correspondente expressão `posfixa`.

```
char *infixaParaPosfixa(char *inf) {
    char *posf; /* expressao polonesa */
    int n = strlen(inf);
    int i; /* percorre infixada */
    int j; /* percorre posfixa */
    char *s; /* pilha */
    int t; /* topo da pilha */

    /*aloca area para expressao polonesa*/
    posf = mallocSafe((n+1)*sizeof(char));
    /* 0 '+1'eh para o '\0'*/
```

case '('

```
/* stackInit(n):  inicializa a pilha */
s = mallocSafe(n * sizeof(char));
t = 0;
/* examina cada item da infixada */
for (i = j = 0; i < n; i++) {
    switch (inf[i]) {
        char x; /* item do topo da pilha */
        case '(':
            /* stackPush(inf[i]) */
            s[t++] = inf[i];
            break;
```

case ')':

```
case ')':  
    /* x = stackPop() */  
    while ((x = s[--t]) != '(')  
        posf[j++] = x;  
    break;
```

case '+', case '-'

```
case '+':
```

```
case '-':
```

```
    /* !stackEmpty()
```

```
        && (stackTop()) != '('
```

```
    */
```

```
    while (t != 0
```

```
        && (x = s[t-1]) != '(')
```

```
        posf[j++] = s[--t];
```

```
    /* stackPush(inf[i]) */
```

```
    s[t++] = inf[i];
```

```
    break;
```

case '*', case '/'

```
case '*':
case '/':
/* !stackEmpty() &&
   prec(stackTop()) >= prec(inf[i])
*/
while (t != 0
      && (x = s[t-1]) != '('
      && x != '+' && x != '-')
    posf[j++] = s[--t];
/* stackPush(inf[i]) */
s[t++] = inf[i];
break;
```

default

```
default:
    if (inf[i] != ' ')
        posf[j++] = inf[i];
} /* fim switch */
} /* fim for (i=j=0...) */
```

Finalizações

```
/* desempilha todos os operandos que
   restaram */
/* !stackEmpty() */
while (t != 0)
    posf[j++] = s[--t]; /* stackPop() */
posf[j] = '\0'; /* fim expr polonesa */
/* stackFree() */
free(s);
return posf;
} /* fim funcao */
```


Consumo de tempo e espaço

O consumo de tempo é da função `infixaParaPosfixa(inf)` é proporcional a n , onde n é o número de caracteres na string `inf`.

O espaço extra utilizado pela função `infixaParaPosfixa(inf)` é proporcional a n , onde n é o número de caracteres na string `inf`.