

stack  
/stak/

Pilhas  
Stack

Prof. Bruno Travençolo

# Pilhas

---

- ▶ Estrutura de dados linear usada para armazenar e organizar dados
- ▶ Sequência de elementos do mesmo tipo
- ▶ Nessa estrutura somente temos conhecimento do último elemento inserido
  - ▶ Diferente de listas, que podemos percorrer todos os elementos.
  - ▶ Diferente de listas duplamente encadeadas, que podemos acessar o primeiro/último elemento via funções especializadas nessas tarefas (`front()/back()`).
- ▶ A remoção de um elemento sempre será do elemento que estiver a menos tempo na pilha
  - ▶ Diferente de listas, que podemos remover elementos das extremidades por meio de funções especializadas (`pop_front()`; `pop_back()`) ou qualquer elemento `erase(int pos)`
- ▶ É um tipo especial de lista, em que a inserção e a remoção são realizadas sempre na mesma extremidade
  - ▶ Em uma pilha feita com uma lista estática sequencial, os elementos são inseridos/removidos sempre do final da lista
  - ▶ Em uma pilha feita com uma lista dinâmica encadeada com ponteiro para a cabeça da lista, os elementos sempre são inseridos e removidos da cabeça da lista
- ▶ LIFO – Last In First Out (último a entrar, primeiro a sair)



# Pilhas

---

- ▶ “Mas como faço então pra consultar um elemento ‘no meio’ da pilha?”
  - ▶ Não faz. Se for necessário consultar algum elemento diferente do último inserido, isso significa que não é necessário usar uma pilha. Talvez uma lista serviria
- ▶ Vantagens: estrutura mais simples e mais flexível de implementar. Não há preocupação com vários ponteiros e também o número de operações é menor



# Pilhas

---

- ▶ Operações de manipulação da pilha
  - ▶ Inserir um elemento na pilha\*
  - ▶ Remover um elemento da pilha\*
  - ▶ Acesso ao elemento da pilha\*
  - ▶ Verificar se a pilha está cheia ou vazia
- ▶ Operações relacionadas a estrutura da pilha
  - ▶ Criar a pilha
  - ▶ Destruir a pilha

\* Inserção, remoção e acesso é sempre feito no topo

---



# Aplicação: parênteses e colchetes

## ► Bem formada?

( ( ) [ ( ) ] )

```
#define N 100
char pilha[N];
int t;
```

```
// Esta função devolve 1 se a string ASCII s
// contém uma sequência bem-formada de
// parênteses e colchetes e devolve 0 se
// a sequência é malformada.
```

```
int bemFormada (char s[])
{
    criapilha ();
    for (int i = 0; s[i] != '\0'; ++i) {
        char c;
        switch (s[i]) {
            case ')': if (pilhavazia ()) return 0;
                     c = desempilha ();
                     if (c != '(') return 0;
                     break;
            case ']': if (pilhavazia ()) return 0;
                     c = desempilha ();
                     if (c != '[') return 0;
                     break;
            default: empilha (s[i]);
        }
    }
    return pilhavazia ();
}
```

# Exemplo: Expressão matemática

---

- ▶ Motivação: avaliação de uma expressão matemática
  - ▶  $5 * ((9 + 8) * (4 * 6)) + 7$
- ▶ A pilha é a ED ideal para isso, por permite guardar os valores intermediários das operações

```
push(5);  
push(9);  
push(8);  
push(pop() + pop());  
push(4);  
push(6);  
push(pop()*pop());  
push(pop()*pop());  
push(7);  
push(pop()+pop());  
push(pop()*pop());  
printf("%d\n", pop());
```

# Exemplo: Expressão matemática

---

- ▶ Essa ordem de cálculo exige que os operandos (os números) estejam na pilha antes do operadores (+ ou \* neste exemplo)
- ▶ 9 e 8 estão na pilha para só depois ser feito a operação de soma

```
push(5);  
push(9);  
push(8);  
push(pop() + pop());  
push(4);  
push(6);  
push(pop()*pop());  
push(pop()*pop());  
push(7);  
push(pop()+pop());  
push(pop()*pop());  
printf("%d\n", pop());
```

# Exemplo: Expressão matemática

---

- ▶ Qualquer operação aritmética pode ser reescrita

- ▶  $(5 * ((9 + 8) * (4 * 6)) + 7)$

- ▶ Passa a ser

- ▶  $5\ 9\ 8\ +\ 4\ 6\ *\ 7\ +\ *$

- ▶ Essa notação é chamada de *Reverse Polish* ou **postfix (posfixa)**. A primeira notação, que estamos acostumados, é chamada de **infix (infixa)**

- ▶ Na notação **postfix** não utilizamos parênteses

```
push(5);
push(9);
push(8);
push(pop() + pop());
push(4);
push(6);
push(pop()*pop());
push(pop()*pop());
push(7);
push(pop()+pop());
push(pop()*pop());
printf("%d\n", pop);
```



# Exemplo: Expressão matemática

## ► Converter infix a para posfixa

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

Note que os operandos (A, B, C, etc.) aparecem na mesma ordem na expressão infix a e na correspondente expressão posfixa. Note também que a notação posfixa dispensa parênteses e regras de precedência entre operadores (como a precedência de \* sobre + por exemplo), que são indispensáveis na notação infix a.

# Exemplo: Expressão matemática

---

- ▶ Todo parêntese esquerdo é colocado na pilha.
- ▶ Ao encontrar um parêntese direito, o algoritmo desempilha tudo até encontrar um parêntese esquerdo, que também é desempilhado.
- ▶ Ao encontrar um  $+$  ou um  $-$ , o algoritmo desempilha tudo até encontrar um parêntese esquerdo, que não é desempilhado.
- ▶ Ao encontrar um  $*$  ou um  $/$ , o algoritmo desempilha tudo até encontrar um parêntese esquerdo ou um  $+$  ou um  $-$ .  
Constantes e variáveis são transferidos diretamente de inf para a expressão posfixa.

infix    

(	A	*	(	B	*	C	+	D	)	)
---	---	---	---	---	---	---	---	---	---	---

# Exemplo: Expressão matemática

---

- ▶ Traduzir para notação posfixa a expressão infixa armazenada em uma string
- ▶  $(A*(B*C+D))$
- ▶ Supor que a expressão esteja em uma string em C

infix 

(	A	*	(	B	*	C	+	D	)	)
---	---	---	---	---	---	---	---	---	---	---

- ▶ Devemos criar um vetor para armazenar o resultados posfixo

posfix 

--	--	--	--	--	--	--	--	--	--	--	--

- ▶ E uma pilha pra avaliar expressão

stack 

--	--	--	--	--	--	--	--	--	--	--	--

# Exemplo: Expressão matemática

---

inf    ( A \* ( B \* C + D ) )

posf    

--	--	--	--	--	--	--	--	--	--	--	--

stack    

--	--	--	--	--	--	--	--	--	--	--	--

```
char *infixaParaPosfixa (char *inf) {  
    int n = strlen (inf);  
    char *posf;  
    posf = malloc ((n+1) * sizeof (char));  
    criapilha ();
```

inf    ( A \* ( B \* C + D ) )  
          \*

posf    

--	--	--	--	--	--	--	--	--	--	--	--

stack    ( 

--	--	--	--	--	--	--	--	--	--	--

```
empilha (inf[0]); // empilha '('
```

inf

(	A	*	(	B	*	C	+	D	)	)
i=1										

posf

j=0										

stack

(										
---	--	--	--	--	--	--	--	--	--	--

```
int j = 0;
for (int i = 1; inf[i] != '\0'; ++i) {
    switch (inf[i]) {
        char x;
        case '(': empilha (inf[i]);
                    break;
        case ')': x = desempilha ();
                    while (x != '(') {
                        posf[j++] = x;
                        x = desempilha ();
                    }
                    break;
        case '+':
        case '-': x = desempilha ();
                    while (x != '(') {
                        posf[j++] = x;
                        x = desempilha ();
                    }
                    empilha (x);
                    empilha (inf[i]);
                    break;
        case '*':
        case '/': x = desempilha ();
                    while (x != '(' && x != '+' && x != '-') {
                        posf[j++] = x;
                        x = desempilha ();
                    }
                    empilha (x);
                    empilha (inf[i]);
                    break;
        default: posf[j++] = inf[i];
    }
}
```

```

int j = 0;
for (int i = 1; inf[i] != '\0'; ++i) {
    switch (inf[i]) {
        char x;
        case '(': empilha (inf[i]);
                    break;
        case ')': x = desempilha ();
                    while (x != '(') {
                        posf[j++] = x;
                        x = desempilha ();
                    }
                    break;
        case '+':
        case '-': x = desempilha ();
                    while (x != '(') {
                        posf[j++] = x;
                        x = desempilha ();
                    }
                    empilha (x);
                    empilha (inf[i])
                    break;
        case '*':
        case '/': x = desempilha ();
                    while (x != '(' && x != '+' && x != '-') {
                        posf[j++] = x;
                        x = desempilha ();
                    }
                    empilha (x);
                    empilha (inf[i]);
                    break;
        default: posf[j++] = inf[i];
    }
}

```

inf    ( A \* ( B \* C + D ) )  
           i=1

posf    

--	--	--	--	--	--	--	--	--	--	--	--

  
           j=0

stack    ( 

--	--	--	--	--	--	--	--	--	--	--

inf    ( A \* ( B \* C + D ) )  
           i=1

posf    A 

--	--	--	--	--	--	--	--	--	--	--

  
           j=1

stack    ( 

--	--	--	--	--	--	--	--	--	--	--





stack	(									
-------	---	--	--	--	--	--	--	--	--	--







```
char x;
case '(': empilha (inf[i]);
        break;
case ')': x = desempilha ();
        while (x != '(') {
            posf[j++] = x;
            x = desempilha ();
        }
        break;
```

```
case '+':  
case '-': x = desempilha ();  
         while (x != '(') {  
             posf[j++] = x;  
             x = desempilha ();  
         }  
         empilha (x);  
         empilha (inf[i])  
         break;
```

```
case '*':  
case '/': x = desempilha ();  
        while (x != '(' && x != '+' && x != '-') {  
            posf[j++] = x;  
            x = desempilha ();  
        }
```

```
empilha (x);
empilha (inf[i]);
break;
```

```
default: posf[j++] = inf[i];
```

} }

stack	(								
-------	---	--	--	--	--	--	--	--	--

 $x \leftarrow ($

```

int j = 0;
for (int i = 1; inf[i] != '\0'; ++i) {
    switch (inf[i]) {
        char x;
        case '(': empilha (inf[i]);
                    break;
        case ')': x = desempilha ();
                    while (x != '(') {
                        posf[j++] = x;
                        x = desempilha ();
                    }
                    break;
        case '+':
        case '-': x = desempilha ();
                    while (x != '(') {
                        posf[j++] = x;
                        x = desempilha ();
                    }
                    empilha (x);
                    empilha (inf[i])
                    break;
        case '*':
        case '/': x = desempilha ();
                    while (x != '(' && x != '+' && x != '-') {
                        posf[j++] = x;
                        x = desempilha ();
                    }
                    empilha (x);
                    empilha (inf[i]);
                    break;
        default: posf[j++] = inf[i];
    }
}

```

inf    ( | A | \* | ( | B | \* | C | + | D | ) | )  
           i=2

posf    A |   |   |   |   |   |   |   |   |   |  
           j=1

stack    ( |   |   |   |   |   |   |   |   |   |

inf    ( | A | \* | ( | B | \* | C | + | D | ) | )  
           i=2

posf    A |   |   |   |   |   |   |   |   |   |  
           j=1

stack    ( | \* |   |   |   |   |   |   |   |



```
char x;  
case '(': empilha (inf[i]);  
        break;  
case ')': x = desempilha ();  
        while (x != '(') {  
            posf[j++] = x;  
            x = desempilha ();  
        }  
        break;
```

```

case '+':
case '-': x = desempilha ();
        while (x != '(') {
            posf[j++] = x;
            x = desempilha ();
        }
empilha (x);
empilha (inf[i])
break;

```

```
case '*':
case '/': x = desempilha ();
        while (x != '(' && x != '+' && x != '-') {
            posf[j++] = x;
            x = desempilha ();
        }
        empilha (x);
        empilha (inf[i]);
        break;
default: posf[j++] = inf[i];
```

} }

stack	(	*	(							
-------	---	---	---	--	--	--	--	--	--	--



} }



stack	(	*	(						
-------	---	---	---	--	--	--	--	--	--







```
char x;
case '(': empilha (inf[i]);
        break;
case ')': x = desempilha ();
        while (x != '(') {
            posf[j++] = x;
            x = desempilha ();
        }
        break;
```

```
case '+':  
case '-': x = desempilha ();  
        while (x != '(') {  
            posf[j++] = x;  
            x = desempilha ();  
        }  
empilha (x);  
empilha (inf[i])  
break;
```

```
case '*':
case '/': x = desempilha ();
        while (x != '(' && x != '+' && x != '-') {
            posf[j++] = x;
            x = desempilha ();
        }
```

```
empilha (x);  
empilha (inf[i]);  
break;
```

```
default: posf[j++] = inf[i];
```

$$\left. \begin{array}{l} \{ \\ \} \end{array} \right\}$$

stack	(	*	(							
-------	---	---	---	--	--	--	--	--	--	--

 $x \leftarrow ($



```

int j = 0;
for (int i = 1; inf[i] != '\0'; ++i) {
    switch (inf[i]) {
        char x;
        case '(': empilha (inf[i]);
                    break;
        case ')': x = desempilha ();
                    while (x != '(') {
                        posf[j++] = x;
                        x = desempilha ();
                    }
                    break;
        case '+':
        case '-': x = desempilha ();
                    while (x != '(') {
                        posf[j++] = x;
                        x = desempilha ();
                    }
                    empilha (x);
                    empilha (inf[i]);
                    break;
        case '*':
        case '/': x = desempilha ();
                    while (x != '(' && x != '+' && x != '-') {
                        posf[j++] = x;
                        x = desempilha ();
                    }
                    empilha (x);
                    empilha (inf[i]);
                    break;
        default: posf[j++] = inf[i];
    }
}

```

inf    (   A   \*   (   B   \*   C   +   D   )   )

i=6

posf   A   B  

j=2

stack   (   \*   (   \*  





```

int j = 0;
for (int i = 1; inf[i] != '\0'; ++i) {
    switch (inf[i]) {
        char x;
        case '(': empilha (inf[i]);
                    break;
        case ')': x = desempilha ();
                    while (x != '(') {
                        posf[j++] = x;
                        x = desempilha ();
                    }
                    break;
        case '+':
        case '-': x = desempilha ();
                    while (x != '(') {
                        posf[j++] = x;
                        x = desempilha ();
                    }
                    empilha (x);
                    empilha (inf[i]);
                    break;
        case '*':
        case '/': x = desempilha ();
                    while (x != '(' && x != '+' && x != '-') {
                        posf[j++] = x;
                        x = desempilha ();
                    }
                    empilha (x);
                    empilha (inf[i]);
                    break;
        default: posf[j++] = inf[i];
    }
}

```

inf    (   A   \*   (   B   \*   C   +   D   )   )

i=7

posf   A   B   C  

j=3

stack   (   \*   (   \*  

$x \leftarrow *$







```
char x;
case '(': empilha (inf[i]);
        break;
case ')': x = desempilha ();
        while (x != '(') {
            posf[j++] = x;
            x = desempilha ();
        }
        break;
```

```
case '+':  
case '-': x = desempilha ();  
         while (x != '(') {  
             post[j++] = x;  
             x = desempilha ();  
         }  
empilha (x);  
empilha (inf[i])  
break;
```

```
case '*':
case '/': x = desempilha ();
        while (x != '(' && x != '+' && x != '-') {
            posf[j++] = x;
            x = desempilha ();
        }
        empilha (x);
        empilha (inf[i]);
        break;
default: posf[j++] = inf[i];
```

```
default: posf[j++] = inf[i];
```

---

[algoritmos/aulas/pilha.html](http://algoritmos/aulas/pilha.html)

[gortinos/aulas/pilha.html](http://gortinos/aulas/pilha.html)

posf

A	B	C							
---	---	---	--	--	--	--	--	--	--

j=3

inf      ( A \* ( B \* C + D ) )

i=7

posf    A B C    j=3

stack	(	*	(							
-------	---	---	---	--	--	--	--	--	--	--

$$X \leftarrow *$$

} }

TRUE

```
char x;
case '(': empilha (inf[i]);
        break;
case ')': x = desempilha ();
        while (x != '(') {
            posf[j++] = x;
            x = desempilha ();
        }
        break;
```

```
case '+':
case '-': x = desempilha ();
        while (x != '(') {
            posf[j++] = x;
            x = desempilha ();
        }
empilha (x);
empilha (inf[i])
break;
```

```
case '*':
case '/': x = desempilha ();
        while (x != '(' && x != '+' && x != '-') {
            posf[j++] = x;
            x = desempilha ();
        }
        empilha (x);
        empilha (inf[i]);
        break;
default: posf[j++] = inf[i];
```

stack	(	*	(							
-------	---	---	---	--	--	--	--	--	--	--

$$X \leftarrow *$$

} }





} }

} }

```

int j = 0;
for (int i = 1; inf[i] != '\0'; ++i) {
    switch (inf[i]) {
        char x;
        case '(': empilha (inf[i]);
                    break;
        case ')': x = desempilha ();
                    while (x != '(') {
                        posf[j++] = x;
                        x = desempilha ();
                    }
                    break;
        case '+':
        case '-': x = desempilha ();
                    while (x != '(') {
                        posf[j++] = x;
                        x = desempilha ();
                    }
                    empilha (x);
                    empilha (inf[i]);
                    break;
        case '*':
        case '/': x = desempilha ();
                    while (x != '(' && x != '+' && x != '-') {
                        posf[j++] = x;
                        x = desempilha ();
                    }
                    empilha (x);
                    empilha (inf[i]);
                    break;
        default: posf[j++] = inf[i];
    }
}

```

inf    (   A   \*   (   B   \*   C   +   D   )   )

i=8

posf   A   B   C   \*  

j=4

stack   (   \*   (   +  



```

int j = 0;
for (int i = 1; inf[i] != '\0'; ++i) {
    switch (inf[i]) {
        char x;
        case '(': empilha (inf[i]);
                    break;
        case ')': x = desempilha ();
                    while (x != '(') {
                        posf[j++] = x;
                        x = desempilha ();
                    }
                    break;
        case '+':
        case '-': x = desempilha ();
                    while (x != '(') {
                        posf[j++] = x;
                        x = desempilha ();
                    }
                    empilha (x);
                    empilha (inf[i])
                    break;
        case '*':
        case '/': x = desempilha ();
                    while (x != '(' && x != '+' && x != '-') {
                        posf[j++] = x;
                        x = desempilha ();
                    }
                    empilha (x);
                    empilha (inf[i]);
                    break;
        default: posf[j++] = inf[i];
    }
}

```

inf    ( | A | \* | ( | B | \* | C | + | D | ) | )  
i=8

posf    A | B | C | \* |   |   |   |   |   |   |  
j=4

stack   ( | \* | ( | + |   |   |   |   |   |   |

inf    ( | A | \* | ( | B | \* | C | + | D | ) | )  
i=8

posf    A | B | C | \* | D |   |   |   |   |   |   |  
j=5

stack   ( | \* | ( | + |   |   |   |   |   |   |







stack	(	*	(	+					
-------	---	---	---	---	--	--	--	--	--

```
char x;  
case '(': empilha (inf[i]);  
        break;  
case ')': x = desempilha ();  
        while (x != '(') {  
            posf[j++] = x;  
            x = desempilha ();  
        }  
        break;
```

```
case '+':  
case '-': x = desempilha ();  
        while (x != '(') {  
            posf[j++] = x;  
            x = desempilha ();  
        }  
        empilha (x);  
        empilha (inf[i])  
        break;
```

```
case '*':
case '/': x = desempilha ();
        while (x != '(' && x != '+' && x != '-') {
            posf[j++] = x;
            x = desempilha ();
        }
        empilha (x);
        empilha (inf[i]);
        break;
default: posf[j++] = inf[i];
```

stack	(	*	(							
-------	---	---	---	--	--	--	--	--	--	--

$$X \leftarrow +$$

} }



```
char x;
case '(': empilha (inf[i]);
        break;
case ')': x = desempilha ();
        while (x != '(') {
            posf[j++] = x;
            x = desempilha ();
        }
        break;
```

```
case '+':  
case '-': x = desempilha ();  
         while (x != '(') {  
             posf[j++] = x;  
             x = desempilha ();  
         }  
         empilha (x);  
         empilha (inf[i])  
         break;
```

```
case '*':
case '/': x = desempilha ();
        while (x != '(' && x != '+' && x != '-') {
            posf[j++] = x;
            x = desempilha ();
        }
        empilha (x);
        empilha (inf[i]);
        break;
default: posf[j++] = inf[i];
```

```
default: posf[j++] = inf[i];
```

posf    

A	B	C	*	D					
---	---	---	---	---	--	--	--	--	--

j=5

inf      (   A   \*   (   B   \*   C   +   D   )   )

i=9

posf    

A	B	C	*	D	+					
---	---	---	---	---	---	--	--	--	--	--

  
j=6

 $x \leftarrow ($

```
char x;  
case '(': empilha (inf[i]);  
        break;  
case ')': x = desempilha ();  
        while (x != '(') {  
            posf[j++] = x;  
            x = desempilha ();  
        }  
        break;
```

```
case '+':  
case '-': x = desempilha ();  
         while (x != '(') {  
             posf[j++] = x;  
             x = desempilha ();  
         }  
         empilha (x);  
         empilha (inf[i])  
         break;
```

```
case '*':  
case '/': x = desempilha ();  
        while (x != '(' && x != '+' && x != '-') {  
            posf[j++] = x;  
            x = desempilha ();  
        }  
        empilha (x);  
        empilha (inf[i]);  
        break;  
default: posf[j++] = inf[i];
```

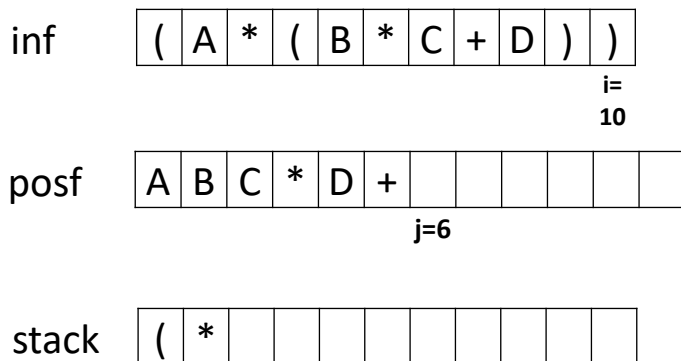
```
default: posf[j++] = inf[i];
```

stack	(	*							
-------	---	---	--	--	--	--	--	--	--

 $x \leftarrow ($ 

} }


**FALSE**




```

int j = 0;
for (int i = 1; inf[i] != '\0'; ++i) {
    switch (inf[i]) {
        char x;
        case '(': empilha (inf[i]);
                    break;
        case ')': x = desempilha ();
                    while (x != '(') {
                        posf[j++] = x;
                        x = desempilha ();
                    }
                    break;
        case '+':
        case '-': x = desempilha ();
                    while (x != '(') {
                        posf[j++] = x;
                        x = desempilha ();
                    }
                    empilha (x);
                    empilha (inf[i]);
                    break;
        case '*':
        case '/': x = desempilha ();
                    while (x != '(' && x != '+' && x != '-') {
                        posf[j++] = x;
                        x = desempilha ();
                    }
                    empilha (x);
                    empilha (inf[i]);
                    break;
        default: posf[j++] = inf[i];
    }
}

```



```
int j = 0;
for (int i = 1; inf[i] != '\0'; ++i) {
    switch (inf[i]) {
```

```
        char x;
        case '(': empilha (inf[i]);
                    break;
        case ')': x = desempilha (); 
                    while (x != '(') {
                        posf[j++] = x;
                        x = desempilha ();
                    }
                    break;
```

```
        case '+':
        case '-': x = desempilha ();
                    while (x != '(') {
                        posf[j++] = x;
                        x = desempilha ();
                    }
                    empilha (x);
                    empilha (inf[i]);
                    break;
```

```
        case '*':
        case '/': x = desempilha ();
                    while (x != '(' && x != '+' && x != '-') {
                        posf[j++] = x;
                        x = desempilha ();
                    }
                    empilha (x);
                    empilha (inf[i]);
                    break;
        default: posf[j++] = inf[i];
```

```
    }
}
```

inf ( A \* ( B \* C + D ) )

i=10

posf A B C \* D +

j=6

stack ( \*

inf ( A \* ( B \* C + D ) )

i=10


posf A B C \* D +

j=6

stack (

$x \leftarrow *$

```
int j = 0;
for (int i = 1; inf[i] != '\0'; ++i) {
    switch (inf[i]) {
```

```
        char x;
        case '(': empilha (inf[i]);
                    break;
        case ')': x = desempilha (); 
                    while (x != '(') {
                        posf[j++] = x;
                        x = desempilha ();
                    }
                    break;
```

```
        case '+':
        case '-': x = desempilha ();
                    while (x != '(') {
                        posf[j++] = x;
                        x = desempilha ();
                    }
                    empilha (x);
                    empilha (inf[i]);
                    break;
```

```
        case '*':
        case '/': x = desempilha ();
                    while (x != '(' && x != '+' && x != '-') {
                        posf[j++] = x;
                        x = desempilha ();
                    }
                    empilha (x);
                    empilha (inf[i]);
                    break;
        default: posf[j++] = inf[i];
```

```
    }
}
```

inf ( A \* ( B \* C + D ) )

i=10

posf A B C \* D +

j=6

stack ( \*

inf ( A \* ( B \* C + D ) )

i=10

posf A B C \* D +

j=6

stack (

$x \leftarrow *$



```
int j = 0;
for (int i = 1; inf[i] != '\0'; ++i) {
    switch (inf[i]) {
```

```
        char x;
        case '(': empilha (inf[i]);
                    break;
        case ')': x = desempilha ();
                    while (x != '(') {
                        posf[j++] = x;
                        x = desempilha ();
                    }
                    break;
```

```
        case '+':
        case '-': x = desempilha ();
                    while (x != '(') {
                        posf[j++] = x;
                        x = desempilha ();
                    }
                    empilha (x);
                    empilha (inf[i]);
                    break;
```

```
        case '*':
        case '/': x = desempilha ();
                    while (x != '(' && x != '+' && x != '-') {
                        posf[j++] = x;
                        x = desempilha ();
                    }
                    empilha (x);
                    empilha (inf[i]);
                    break;
        default: posf[j++] = inf[i];
    }
```

inf ( A \* ( B \* C + D ) )

i=10

posf A B C \* D +

j=6

stack ( \*

inf ( A \* ( B \* C + D ) )

i=10

posf A B C \* D +

j=6

stack (

x ← \*

TRUE

```
int j = 0;
for (int i = 1; inf[i] != '\0'; ++i) {
    switch (inf[i]) {
```

```
        char x;
        case '(': empilha (inf[i]);
                    break;
        case ')': x = desempilha ();
                    while (x != '(') {
                        posf[j++] = x;
                        x = desempilha ();
                    }
                    break;
```

```
        case '+':
        case '-': x = desempilha ();
                    while (x != '(') {
                        posf[j++] = x;
                        x = desempilha ();
                    }
                    empilha (x);
                    empilha (inf[i]);
                    break;
```

```
        case '*':
        case '/': x = desempilha ();
                    while (x != '(' && x != '+' && x != '-') {
                        posf[j++] = x;
                        x = desempilha ();
                    }
                    empilha (x);
                    empilha (inf[i]);
                    break;
        default: posf[j++] = inf[i];
    }
```

inf ( A \* ( B \* C + D ) )

i=10

posf A B C \* D +

j=6

stack ( \*

inf ( A \* ( B \* C + D ) )

i=10

posf A B C \* D + \*

j=7

stack

x ← \*

}



```

int j = 0;
for (int i = 1; inf[i] != '\0'; ++i) {
    switch (inf[i]) {

```

```

        char x;
        case '(': empilha (inf[i]);
                    break;
        case ')': x = desempilha ();
                    while (x != '(') {
                        posf[j++] = x;
                        x = desempilha ();
                    }
                    break;

```

```

        case '+':
        case '-': x = desempilha ();
                    while (x != '(') {
                        posf[j++] = x;
                        x = desempilha ();
                    }
                    empilha (x);
                    empilha (inf[i]);
                    break;

```

```

        case '*':
        case '/': x = desempilha ();
                    while (x != '(' && x != '+' && x != '-') {
                        posf[j++] = x;
                        x = desempilha ();
                    }
                    empilha (x);
                    empilha (inf[i]);
                    break;
        default: posf[j++] = inf[i];
    }
}

```

inf ( A \* ( B \* C + D ) )

i=10

posf A B C \* D +

j=6

stack ( \*

inf ( A \* ( B \* C + D ) )

i=10

posf A B C \* D + \*

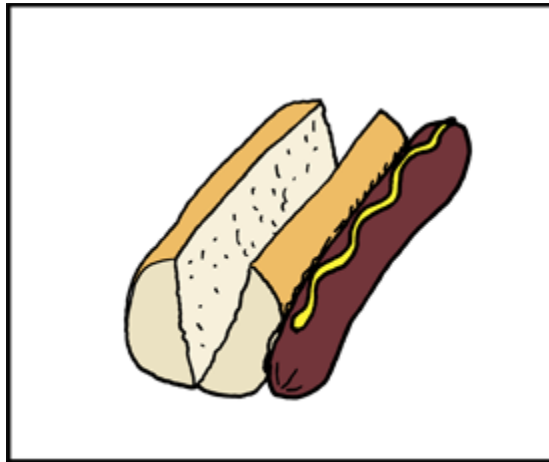
j=7

stack

x←(

}





REVERSE POLISH SAUSAGE



```

int j = 0;
for (int i = 1; inf[i] != '\0'; ++i) {
    switch (inf[i]) {
        char x;
        case '(': empilha (inf[i]);
                break;
        case ')': x = desempilha ();
                while (x != '(') {
                    posf[j++] = x;
                    x = desempilha ();
                }
                break;
        case '+':
        case '-': x = desempilha ();
                while (x != '(') {
                    posf[j++] = x;
                    x = desempilha ();
                }
                empilha (x);
                empilha (inf[i])
                break;
        case '*':
        case '/': x = desempilha ();
                while (x != '(' && x != '+' && x != '-') {
                    posf[j++] = x;
                    x = desempilha ();
                }
                empilha (x);
                empilha (inf[i]);
                break;
        default: posf[j++] = inf[i];
    }
}

```

inf ( A + B \* C )

i=  
2

posf

stack (

inf ( A + B \* C )

posf A

stack (

```
int j = 0;
for (int i = 1; inf[i] != '\0'; ++i) {
    switch (inf[i]) {
```

```
        char x;
        case '(': empilha (inf[i]);
                    break;
        case ')': x = desempilha ();
                    while (x != '(') {
                        posf[j++] = x;
                        x = desempilha ();
                    }
                    break;
```

```
        case '+':
        case '-': x = desempilha ();
                    while (x != '(') {
                        posf[j++] = x;
                        x = desempilha ();
                    }
                    empilha (x);
                    empilha (inf[i])
                    break;
```

```
        case '*':
        case '/': x = desempilha ();
                    while (x != '(' && x != '+' && x != '-') {
                        posf[j++] = x;
                        x = desempilha ();
                    }
                    empilha (x);
                    empilha (inf[i]);
                    break;
        default: posf[j++] = inf[i];
    }
```

inf ( A + B \* C )

i=  
3

posf A

stack (

inf ( A + B \* C )

posf A

stack ( +

Desempilha e empilha ( ) }

```

int j = 0;
for (int i = 1; inf[i] != '\0'; ++i) {
    switch (inf[i]) {
        char x;
        case '(': empilha (inf[i]);
                    break;
        case ')': x = desempilha ();
                    while (x != '(') {
                        posf[j++] = x;
                        x = desempilha ();
                    }
                    break;
        case '+':
        case '-': x = desempilha ();
                    while (x != '(') {
                        posf[j++] = x;
                        x = desempilha ();
                    }
                    empilha (x);
                    empilha (inf[i])
                    break;
        case '*':
        case '/': x = desempilha ();
                    while (x != '(' && x != '+' && x != '-') {
                        posf[j++] = x;
                        x = desempilha ();
                    }
                    empilha (x);
                    empilha (inf[i]);
                    break;
        default: posf[j++] = inf[i];
    }
}

```

inf    ( | A | + | B | \* | C | ) | | | |

i=4

posf   A | | | | | | | | | |

stack   ( | + | | | | | | | | |

inf    ( | A | + | B | \* | C | ) | | | |

posf   A | B | | | | | | | | | |

stack   ( | + | | | | | | | | |



```

int j = 0;
for (int i = 1; inf[i] != '\0'; ++i) {
    switch (inf[i]) {
        char x;
        case '(': empilha (inf[i]);
                break;
        case ')': x = desempilha ();
                while (x != '(') {
                    posf[j++] = x;
                    x = desempilha ();
                }
                break;
        case '+':
        case '-': x = desempilha ();
                while (x != '(') {
                    posf[j++] = x;
                    x = desempilha ();
                }
                empilha (x);
                empilha (inf[i])
                break;
        case '*':
        case '/': x = desempilha ();
                while (x != '(' && x != '+' && x != '-' && x != '/') {
                    posf[j++] = x;
                    x = desempilha ();
                }
                empilha (x);
                empilha (inf[i]);
                break;
        default: posf[j++] = inf[i];
    }
}

```

inf    ( | A | + | B | \* | C | ) | | | |

i=5

posf   A | B | | | | | | | | | |

stack   ( | + | | | | | | | | |

inf    ( | A | + | B | \* | C | ) | | | |

posf   A | B | | | | | | | | | |

stack   ( | + | | | | | | | | |

Retira e coloca + na stack



```

int j = 0;
for (int i = 1; inf[i] != '\0'; ++i) {
    switch (inf[i]) {
        char x;
        case '(': empilha (inf[i]);
                break;
        case ')': x = desempilha ();
                while (x != '(') {
                    posf[j++] = x;
                    x = desempilha ();
                }
                break;
        case '+':
        case '-': x = desempilha ();
                while (x != '(') {
                    posf[j++] = x;
                    x = desempilha ();
                }
                empilha (x);
                empilha (inf[i])
                break;
        case '*':
        case '/': x = desempilha ();
                while (x != '(' && x != '+' && x != '-') {
                    posf[j++] = x;
                    x = desempilha ();
                }
                empilha (x);
                empilha (inf[i]);
                break;
        default: posf[j++] = inf[i];
    }
}

```

inf    ( | A | + | B | \* | C | ) | | | |

i=5

posf   A | B | | | | | | | | | |

stack   ( | | | | | | | | | |

inf    ( | A | + | B | \* | C | ) | | | |

posf   A | B | | | | | | | | | |

stack   ( | + | \* | | | | | | | |

Retira e coloca + na stack



```

int j = 0;
for (int i = 1; inf[i] != '\0'; ++i) {
    switch (inf[i]) {
        char x;
        case '(': empilha (inf[i]);
                    break;
        case ')': x = desempilha ();
                    while (x != '(') {
                        posf[j++] = x;
                        x = desempilha ();
                    }
                    break;
        case '+':
        case '-': x = desempilha ();
                    while (x != '(') {
                        posf[j++] = x;
                        x = desempilha ();
                    }
                    empilha (x);
                    empilha (inf[i]);
                    break;
        case '*':
        case '/': x = desempilha ();
                    while (x != '(' && x != '+' && x != '-') {
                        posf[j++] = x;
                        x = desempilha ();
                    }
                    empilha (x);
                    empilha (inf[i]);
                    break;
        default: posf[j++] = inf[i];
    }
}

```

inf    ( | A | + | B | \* | C | ) | | | |

i=6

posf   A | B | C | | | | | | | |

stack   ( | + | \* | | | | | | |

inf    ( | A | + | B | \* | C | ) | | | |

posf   A | B | C | | | | | | | |

stack   ( | + | \* | | | | | | |











