

stack /stak/	Pilhas Stack
Prof. Bruno Travençolo (com adaptações feitas por Paulo H. R. Gabriel)	

1

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
- ▶ 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
- ▶ É um tipo especial de lista, em que a inserção e a remoção são realizadas sempre na mesma extremidade
 - ▶ LIFO – Last In First Out (último a entrar, primeiro a sair)

▶

2

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
-



3

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
-



4

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 ();
}
```

► Fonte: <https://www.ime.usp.br/~pf/algoritmos/aulas/pilha.html>

5

Exemplo: Expressão matemática

► Motivação: avaliação de uma expressão matemática

► $5 * ((9 + 8) * (4 * 6)) + 7$

► A pilha é a ED ideal para isso, pois 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());
```

► Fonte: Algorithms in C Sedgewick

6

Exemplo: Expressão matemática

- ▶ Essa ordem de cálculo exige que os operandos (os números) estejam na pilha antes dos operadores (+ ou * neste exemplo)
- ▶ 9 e 8 estão na pilha para só depois ser feita 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());
```

▶ Fonte: Algorithms in C Sedgewick

7

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());
```

▶ Fonte: Algorithms in C Sedgewick

8

Exemplo: Expressão matemática

- ▶ Converter infix para posfixa

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

Note que os operandos (A, B, C, etc.) aparecem na mesma ordem na expressão infix 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.

▶ Fonte: <https://www.ime.usp.br/~pf/algoritmos/aulas/pilha.html>

9

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. Os operadores são transferidos quando desempilhados nos passos descritos acima

infix $((A * (B * C + D)))$

▶ Fonte: <https://www.ime.usp.br/~pf/algoritmos/aulas/pilha.html>

10

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

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

▶ Fonte: <https://www.ime.usp.br/~pf/algoritmos/aulas/pilha.html>

11

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

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

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

stack

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

▶ Fonte: <https://www.ime.usp.br/~pf/algoritmos/aulas/pilha.html>

12

14

14

inf

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

i=2

posf

A										
---	--	--	--	--	--	--	--	--	--	--

j=1

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=2

posf

A										
---	--	--	--	--	--	--	--	--	--	--

j=1

stack

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

inf

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

i=2

posf

A										
---	--	--	--	--	--	--	--	--	--	--

j=1

stack

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

```

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

```


Diagram illustrating the execution of a recursive function `empilha` for the expression `(A*(B*C+D))`. The diagram shows the state of the `inf` (input) and `posf` (output) strings and the `stack` at each step.

The initial state is:

- `inf`: `(A * (B * C + D))` (index `i=2`)
- `posf`: `A` (index `j=1`)
- `stack`: `(`

The function `empilha` processes the input string `inf` character by character. It pushes characters onto the `stack` and pops them to build the postfix string `posf`. The diagram shows the state after several recursive calls, with the `posf` string being built as `A`.

A yellow arrow points to the condition `posf[j++] = x;` with the label **FALSE**, indicating a failed attempt to pop from an empty stack.

17

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

i=2

posf A [] [] [] [] [] [] [] []

j=1

stack ([] [] [] [] [] [] [] []

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

i=2

posf A [] [] [] [] [] [] [] []

j=1

stack ([] [] [] [] [] [] [] []

$X \leftarrow ($

```

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

```

18

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

posf A [] [] [] [] [] [] [] []
 j=1

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];
    }
}

```

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

19

inf ((A * ((B * C + D))))
 i=3

posf A [] [] [] [] [] [] [] []
 j=1

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];
    }
}

```

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

20

inf ((A * (B * C + D)))
i=3

posf A [] [] [] [] [] [] [] []
j=1

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];
    }
}

```

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

21

inf ((A * (B * C + D)))
i=4

posf A [] [] [] [] [] [] [] []
j=1

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]; ←
    }
}

```

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

22

inf ([A * ([B * C + D)]))
i=4

posf A [] [] [] [] [] [] [] []
j=1

stack ([* ([] [] [] [] [] [] [] [])

inf ([A * ([B * C + D)]))
i=4

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];
    }
}

```

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

23

inf ([A * ([B * C + D)]))
i=5

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];
    }
}

```

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

24

inf ([A * (B * C + D)])
i=5

posf [A B]
j=2

stack ([* (]))

inf ([A * (B * C + D)])
i=5

posf [A B]
j=2

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];
    }
}

```

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

25

inf ([A * (B * C + D)])
i=5

posf [A B]
j=2

stack ([* (]))

inf ([A * (B * C + D)])
i=5

posf [A B]
j=2

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];
    }
}

```

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

26

[illegible]

27

[illegible]

28

30

30

32

32

34

34

```

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

```

Diagram illustrating the execution of the algorithm for the expression `(A*(B*C+D))` at step `i=7`:

- inf**: `(A * (B * C + D))` (Index `i=7` points to the closing parenthesis `)`)
- posf**: `A B C` (Index `j=3` points to the end of the stack)
- stack**: `(* (*` (The stack contains the opening parenthesis `(`, the multiplication operator `*`, the opening parenthesis `(`, and another multiplication operator `*`)

Diagram illustrating the execution of the algorithm for the expression `(A*(B*C+D))` at step `i=7` (continued):

- inf**: `(A * (B * C + D))` (Index `i=7` points to the closing parenthesis `)`)
- posf**: `A B C *` (Index `j=4` points to the end of the stack)
- stack**: `(*` (The stack contains the opening parenthesis `(` and the multiplication operator `*`)

The diagram shows the state of the algorithm after processing the closing parenthesis `)` at index `i=7`. The `posf` array now contains the entire expression `A B C *` , and the `stack` contains the opening parenthesis `(` and the multiplication operator `*`. The `inf` array remains the same.

The diagram illustrates the execution of a stack-based expression evaluator. It shows three states:

- Initial State:**
 - `inf`: `(A * (B * C + D))`
 - `posf`: An empty array.
 - `stack`: An empty array.
- After reading 'A':**
 - `posf`: Contains `A`.
 - `stack`: Contains `(`.
- After reading '*':**
 - `posf`: Contains `A B C`.
 - `stack`: Contains `(*`.

The diagram uses boxes to represent arrays and arrows to show pointer movement.

inf ([A * (B * C + D)])
i=7

posf [A B C]
j=3

stack ([* ([*]])

inf ([A * (B * C + D)])
i=7

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];
    }
}

```

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

37

inf ([A * (B * C + D)])
i=7

posf [A B C]
j=3

stack ([* ([*]])

inf ([A * (B * C + D)])
i=7

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];
    }
}

```

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

38

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];
    }
}

```

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

39

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 (* (+

```

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];
    }
}

```

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

40

inf

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

i=9

posf

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

j=5

stack

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

inf

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

i=9

posf

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

j=5

stack

(*	(
---	---	---	--	--	--	--	--	--	--	--

```

int j = 0;
for (int i = 1; inf[i] != '\0'; ++i) {
    switch (inf[i]) {
        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=9

posf A B C * D
j=5

stack (* (+
j=5

inf ((A * (B * C + D)))
i=9

posf A B C * D +
j=6

stack (* (+
j=5

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];
    }
}

```

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

43

inf ((A * (B * C + D)))
i=9

posf A B C * D
j=5

stack (* (+
j=5

inf ((A * (B * C + D)))
i=9

posf A B C * D +
j=6

stack (* (+
j=5

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];
    }
}

```

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

44

inf (A * (B * C + D)) i=9

posf A B C * D j=5

stack (* (+ j=5

inf (A * (B * C + D)) i=9

posf A B C * D + j=6

stack (* j=6

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];
    }
}

```

← FALSE

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

45

inf (A * (B * C + D)) i=10

posf A B C * D + j=6

stack (* j=6

```

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];
    }
}

```

←

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

46

inf ((A * (B * C + D)))
i=10

posf A B C * D +
j=6

stack (*
j=6

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];
    }
}

```

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

47

inf ((A * (B * C + D)))
i=10

posf A B C * D +
j=6

stack (*
j=6

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];
    }
}

```

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

48

inf ((A * (B * C + D)))
i=10

posf A B C * D +
j=6

stack (*
j=6

inf ((A * (B * C + D)))
i=10

posf A B C * D +
j=6

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];
    }
}

```

TRUE

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

49

inf ((A * (B * C + D)))
i=10

posf A B C * D +
j=6

stack (*
j=6

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];
    }
}

```

TRUE

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

50

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];
    }
}

```

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

51

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];
    }
}

```

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

52

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]) {
        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];
    }
}

```

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

53

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]) {
        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];
    }
}

```

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

54

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];
    }
}

```

←

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

55

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];
    }
}

```

←

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

56

inf (A + B * C) i=6

posf A B C

stack (+ *

inf (A + B * C)

posf A B C

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];
    }
}

```

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

57

inf (A + B * C) i=7

posf A B C

stack (+ *

inf (A + B * C)

posf A B C

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];
    }
}

```

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

58

inf (A + B * C) i=7

posf A B C

stack (+ *

inf (A + B * C)

posf A B C *

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];
    }
}

```

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

59

inf (A + B * C) i=7

posf A B C

stack (+ *

inf (A + B * C)

posf A B C * +

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];
    }
}

```

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

60

inf ([A + B * C]) i=7

posf A B C

stack ([+ *

inf ([A + B * C])

posf A B C * +

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];
    }
}

```

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

61

Outro exemplo: chamada de sub-rotinas

Rotina A	Rotina B	Rotina C	Rotina D
1 print "A"	1 call C	1 print "C"	1 print "D"
2 call C	2 print "B"	2 call D	2 return
3 call B	3 call D	3 return	
4 call D	4 call C		
5 return	5 return		

Qual o resultado da execução da rotina A?

Fonte: Adaptado das notas de aula do prof. Thiago A. S. Pardo (ICMC/USP)

62

Outro exemplo: chamada de sub-rotinas

1. Um computador está executando a rotina X e, durante a execução de X, encontra uma chamada à rotina Y
2. Interrompe a execução de X e se inicia a execução de Y
3. Quando termina a execução de Y, o computador deve saber o que fazer, isto é, onde voltar na rotina X

► **Fonte:** Adaptado das notas de aula do prof. Thiago A. S. Pardo (ICMC/USP)

63

Outro exemplo: chamada de sub-rotinas

- **Dificuldade**
 - O que estava sendo executado quando uma sub-rotina foi interrompida?
 - Para onde voltar agora que se chegou ao fim de uma sub-rotina?
- **Solução**
 - A cada chamada de sub-rotina, armazenar o endereço de retorno (rotina e número da linha, por exemplo)
 - Como armazenar o endereço de retorno de chamadas sucessivas: pilha

► **Fonte:** Adaptado das notas de aula do prof. Thiago A. S. Pardo (ICMC/USP)

64

Outro exemplo: chamada de sub-rotinas

► Dificuldade

- O que estava sendo executado quando uma sub-rotina foi interrompida?
- Para onde voltar agora que se chegou ao fim de uma sub-rotina?

► Solução

- A cada chamada de sub-rotina, armazenar o endereço de retorno (rotina e número da linha, por exemplo)
- Como armazenar o endereço de retorno de chamadas sucessivas: pilha

► **Fonte:** Adaptado das notas de aula do prof. Thiago A. S. Pardo (ICMC/USP)

65

Outro exemplo: chamada de sub-rotinas

► A cada comando **call**

- Empilha (*push*) o endereço para retornar depois
- Passa a executar a nova sub-rotina

► A cada comando **return**

- Desempilha (*pop*) o último endereço armazenado
- Passa a executar a partir do endereço desempilhado

► **Fonte:** Adaptado das notas de aula do prof. Thiago A. S. Pardo (ICMC/USP)

66