Trabalhando com e sem parênteses

Sim, mas o problema é que as pessoas não fazem assim.

Quer dizer, as pessoas gostam de escrever as coisas assim

$$3 \;+\; 2 \;\times\; \big(5 \;+\; 8 \;\times\; 2\big)$$

E daí, elas sabem que

1. as coisas dentro dos parênteses são feitas primeiro
2. e quando não há parênteses, as multiplicações são feitas primeiro

E agora?

Como é que um computador pode fazer isso?

Bom, aqui é a hora para mais uma esperteza.

Quer dizer, a gente começa observando que dentro dos parênteses não há parênteses

$$3 \;+\; 2 \;\times\; \big(\underbrace{5 \;+\; 8 \;\times\; 2}\big)$$

E nós já temos um algoritmo que calcula expressões sem parênteses.

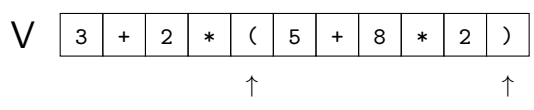
Então, a gente pode pensar que esse algoritmo é uma função.

E que a gente pode chamar essa função para calcular o valor da expressão entre parênteses.

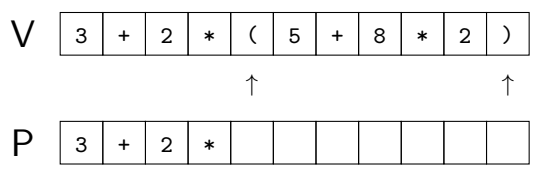
Para a coisa funcionar, é preciso fazer algumas adaptações no algoritmo.

Porque agora, ele vai começar a leitura no `abre-parênteses`.

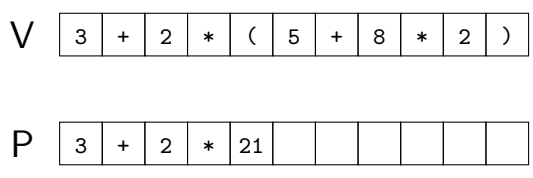
E vai terminar a leitura no `fecha-parênteses`



Nesse ponto, a lista auxiliar P tem a parte anterior da expressão



E a ideia é que após a execução da função, o resultado da expressão entre parênteses já está na lista P



Abaixo, nós temos o código da função

```
func CESP ( V, i, P, j )

    i++                                         // pula o abre-parênteses
    zero = j                                    // imagina que P está vazia

    // 1a ETAPA: calcular as multiplicações

    Enquanto ( V[i] != fecha-parênteses )

        j++;  P[j] = V[i]                       // copia termo para P

        Se ( V[i] == multiplicação )

           i++;    N2 = V[i]
                   op = P[j]
           j--;    N1 = P[j]

           res = Calcula ( N1, op, N2 )

           P[j] = res

        i++

    // 2a ETAPA: calcular as somas

    Enquanto ( j > zero + 1 )

                N2 = P[j]
        j--;    op = P[j]
        j--;    N1 = P[j]

           res = Calcula ( N1, op, N2 )

           P[j] = res

    i++                                         // pula o fecha-parênteses
```

Mas, já dá para advinhar que isso não resolve todos os nossos problemas.

Porque dentro dos parênteses pode haver outro par de parênteses

$$3 \;+\; 2 \;\times\; \big(5 \;+\; 8 \;\times\; (2 \;+\; 3)\big)$$

E agora?

Bom, agora é só fazer a função chamar ela mesma quando ela encontra um par de parênteses.

A coisa fica assim

```
func CECSP ( V, i, P, j )

    i++                                         // pula o abre-parênteses
    zero = j                                    // imagina que P está vazia

    // 1a ETAPA: calcular as multiplicações

    Enquanto ( V[i] != fecha-parênteses )

        Se ( V[i] == abre-parênteses )    CECSP ( V, i, P, j )

        Senão                             j++;  P[j] = V[i]

    Se ( V[i] == multiplicação )

       Se ( V[i] == abre-parênteses )     CECSP ( V, i, P, j )

       Senão                              j++;  P[j] = V[i]
                                          i++

               N2 = V[i]
       j==;    op = P[j]
       j--;    N1 = P[j]

       res = Calcula ( N1, op, N2 )

       P[j] = res

    i++



// 2a ETAPA: calcular as somas

Enquanto ( j > zero + 1 )

            N2 = P[j]
    j--;    op = P[j]
    j--;    N1 = P[j]

    res = Calcula ( N1, op, N2 )

    P[j] = res

i++                                             // pula o fecha-parênteses
```