

Estructuras dinámicas de datos lineales

Lic. Ronaldo Canizales & Mgtr. Guillermo Cortes

Departamento de Electrónica e Informática, UCA

Ciclo Virtual 02/2021

¿Cómo resuelve la computadora expresiones matemáticas como $(A+B)*C$?

Notaciones - Jueves 23 de septiembre

1. Notaciones infija, prefija y postfija

Sencillo para el ser humano:

- Notación infija:
 $A+B$

Sencillo para la computadora:

- Notación prefija (polaca): $+AB$
- Notación postfija (polaca inversa): $AB+$



Jan Łukasiewicz

1.1 Conversión de manera manual (infija a postfija)

Expresión infija:

$$A + (B * C)$$

Convertir la multiplicación:

$$A + (BC*)$$

Convertir la suma:

$$A(BC*)+$$

Expresión postfija:

$$ABC * +$$

Ejemplo 1

1.1 Conversión de manera manual (infija a postfija)

Expresión infija:

$$(A + B) * C$$

Convertir la suma:

$$(AB+) * C$$

Convertir la multiplicación:

$$(AB+)C*$$

Expresión postfija:

$$AB + C *$$

Ejemplo 2

1.1 Conversión de manera manual (infija a postfija)

Infija

$$A + B$$

$$A + B - C$$

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

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

Postfija

$$AB +$$

$$AB + C -$$

$$AB + CD - *$$

$$AB * C + D - EF / GH + / +$$

1.1 Conversión de manera manual (infija a postfija)

$$A + B - C$$

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

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

1.1 Conversión de manera manual (infija a postfija)

$$A + B - C \quad (A + B) * (C - D) \quad A \$ B * C - D + (E / F) / (G + H)$$

$$AB + - C \quad (AB +) * (CD -) \quad A \$ B * C - D + (EF /) / (GH +)$$

$$AB + C - \quad (AB +)(CD -) * \quad AB \$ * C - D + (EF /) / (GH +)$$

$$AB + CD - * \quad AB \$ C * - D + (EF /)(GH +) /$$

$$AB \$ C * D - + (EF /)(GH +) /$$

$$AB \$ C * D - (EF /)(GH +) / +$$

$$AB \$ C * D - EF / GH + / +$$

1.2 Conversión de manera manual (infija a prefija)

Infija

$$A + B$$

$$A + B - C$$

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

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

Prefija

$$+AB$$

$$- + ABC$$

$$* + AB - CD$$

$$+ - * \$ ABCD // EF + GH$$

1.2 Conversión de manera manual (infija a prefija)

$$A + B - C \quad (A + B) * (C - D) \quad A * B * C - D + (E / F) / (G + H)$$

1.2 Conversión de manera manual (infija a prefija)

$$A + B - C$$

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

$$A\$B * C - D + (E/F)/(G + H)$$

$$+AB - C$$

$$(+AB) * (-CD)$$

$$A\$B * C - D + (/EF)/(+GH)$$

$$-+ABC$$

$$*(+AB)(-CD)$$

$$\$AB * C - D + (/EF)/(+GH)$$

$$* + AB - CD$$

$$*\$ABC - D + (/EF)(+GH)$$

$$-*\$ABCD + (/EF)(+GH)$$

$$+- * \$ABCD / (/EF)(+GH)$$

$$+ - * \$ABCD // EF + GH$$

Expresión en notación infija: $(A+B)*C$



Algoritmo que la convierte a prefija o postfija



Expresión en notación prefija o postfija: $AB+C^*$

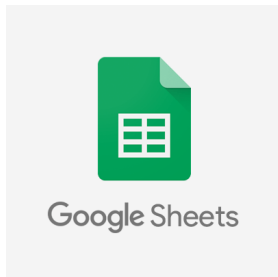


Algoritmo que calcula la solución



Respuesta (valor numérico)

2. Algoritmo que calcula la solución (input: postfija)




Algoritmos paso a paso



Postfija \rightarrow solución



Infija \rightarrow postfija

Convertir char a int 

3. Algoritmo que convierte infijo a postfijo

Algoritmo que convierte infijo en postfijo

```
s = the empty stack
while( not end of input ) {
    symb = next input character;
    if( symb is an operand )
        and symb to the postfix string;
    else {
        while( !empty(s) && prcd( top(s), symb ) {
            topsymb = pop( s );
            add topsymb to the postfix string;
        }
        if( empty( s ) || symb != ')' )
            push( s, symb );
        else
            topsymb = pop( s );
    }
}
while( !empty( s ) ) {
    topsymb = pop( s );
    add topsymb to the postfix string;
}
```

Example 1: $A + B * C$

	<i>symb</i>	<i>postfix string</i>	<i>opstk</i>
1	A	A	
2	+	A	+
3	B	AB	+
4	*	AB	+*
5	C	ABC	+*
6		ABC*	+
7		ABC*+	

Precedencia de operadores

```
prcd( '*', '+' ) = TRUE
prcd( '+', '+' ) = TRUE
prcd( '+', '*' ) = FALSE
prcd( '(', op ) = FALSE
prcd( op, '(' ) = FALSE
prcd( op, ')' ) = TRUE
prcd( ')', op ) = ERROR
prcd( '$', '$' ) = FALSE
```

3. Algoritmo que convierte infijo a postfijo

Algoritmo que convierte infijo en postfijo

```
s = the empty stack
while( not end of input ) {
    symb = next input character;
    if( symb is an operand )
        and symb to the postfix string;
    else {
        while( !empty(s) && prcd( top(s), symb ) {
            topsymb = pop( s );
            add topsymb to the postfix string;
        }
        if( empty( s ) || symb != ')' )
            push( s, symb );
        else
            topsymb = pop( s );
    }
}
while( !empty( s ) ) {
    topsymb = pop( s );
    add topsymb to the postfix string;
}
```

Example 2: $(A + B) * C$

sybm	postfix string	opstk
((
A	A	(
+	A	(+
B	AB	(+
)	AB +	
*	AB +	*
C	AB + C	*
	AB + C *	

Precedencia de operadores

```
prcd( '*', '+' ) = TRUE
prcd( '+', '+' ) = TRUE
prcd( '+', '*' ) = FALSE
prcd( '(', op ) = FALSE
prcd( op, '(' ) = FALSE
prcd( op, ')' ) = TRUE
prcd( ')', op ) = ERROR
prcd( '$', '$' ) = FALSE
```

3. Algoritmo que convierte infijo a postfijo

Algoritmo que convierte infijo en postfijo

```
s = the empty stack

while( not end of input ) {
    symb = next input character;
    if( symb is an operand )
        and symb to the postfix string;
    else {
        while( !empty(s) && prcd( top(s), symb ) ) {
            topsymb = pop( s );
            add topsymb to the postfix string;
        }
        if( empty( s ) || symb != ')' )
            push( s, symb );
        else
            topsymb = pop( s );
    }
}

while( !empty( s ) ) {
    topsymb = pop( s );
    add topsymb to the postfix string;
}
```

Example 3: $((A - (B + C)) * D) \$ (E + F)$

<i>symb</i>	<i>postfix string</i>	<i>opstk</i>
((
(((
A	A	((
-	A	((-
(A	((- (
B	AB	((- (
+	AB	((- (+
C	ABC	((- (+
)	ABC +	((-
)	ABC + -	(
*	ABC + -	(*
D	ABC + - D	(*
)	ABC + - D *	
\$	ABC + - D *	\$
(ABC + - D *	\$(
E	ABC + - D * E	\$(
+	ABC + - D * E	\$(+
F	ABC + - D * EF	\$(+
)	ABC + - D * EF +	\$
	ABC + - D * EF + \$	

4. Ventajas y limitaciones

Ventajas

- Sencillo de comprender y programar.
- Eficiente en cuanto a espacio de memoria.

Limitaciones

- Las expresiones no tienen que contener errores.