Expressões
(IED-001)

Prof. Dr. Silvio do Lago Pereira

Departamento de Tecnologia da Informação

Faculdade de Tecnologia de São Paulo

Expressões

Expressões

são compostas por:

- **operandos**: constantes numéricas.
- **operadores**: operações aritméticas binárias (**+**, **–**, ***** e **/**).
- **delimitadores**: parênteses de abertura e de fechamento.

Notações:

- **Infixa** .....: **A + B** (fácil de escrever e difícil de avaliar)
- **Prefixa** (polonesa) .....: **+ A B**
- **Posfixa** (polonesa reversa).....: **A B +** (difícil de escrever e fácil de avaliar)

Objetivos:

- Converter uma expressão infixa em posfixa.
- Avaliar a forma posfixa de uma expressão.

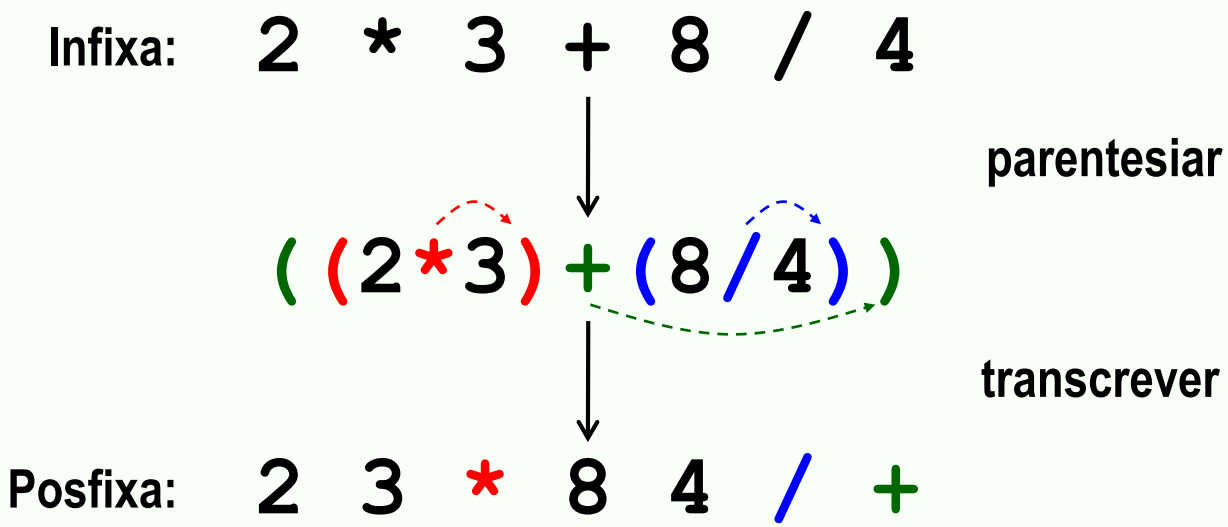
Para simplificar, vamos supor que os operandos são constantes inteiras de um único dígito!

Expressões

Estratégia para converter infixa em posfixa:

- **Parentesiar** completamente a expressão, de acordo com as prioridades dos operadores.
- **Transcreve**r a expressão, descartando parênteses e movendo os operadores para as posições ocupadas por seus respectivos parênteses de fechamento.

Exemplo 1. Conversão de infixa para posfixa



Como automatizar esse procedimento de conversão?

Expressões

Exemplo 2. Conversão de infixa parentesiada para posfixa: `((2*3)+(8/4))`

Elemento	Ação	Pilha	Posfixa
(	descartar	[]	""
(	descartar	[]	""
2	anexar à posfixa	[]	"2"
*	empilhar	[*]	"2"
3	anexar à posfixa	[*]	"23"
)	desempilhar e anexar	[]	"23*"
+	empilhar	[+]	"23*"
(	descartar	[+]	"23*"
8	anexar à posfixa	[+]	"23*8"
/	empilhar	[+,/]	"23*8"
4	anexar à posfixa	[+,/]	"23*84"
)	desempilhar e anexar	[+]	"23*84/"
)	desempilhar e anexar	[]	"23*84/+"

Expressões

Vantagem da forma posfixa:

- Não há parênteses na forma posfixa.
- As operações aparecem na ordem em que elas devem ser efetuadas.
- O valor da forma posfixa pode ser facilmente obtido com o auxílio de uma pilha.

Exemplo 3. Avaliação da forma posfixa: `23*84/+`

Elemento	Ação	Pilha
2	empilhar	[2]
3	empilhar	[2,3]
*	desempilhar dois valores e empilhar seu **produto**	[6]
8	empilhar	[6,8]
4	empilhar	[6,8,4]
/	desempilhar dois valores e empilhar seu **quociente**	[6,2]
+	desempilhar dois valores e empilhar sua **soma**	[8]

O resultado final da avaliação da expressão é o número que sobra no topo da pilha!

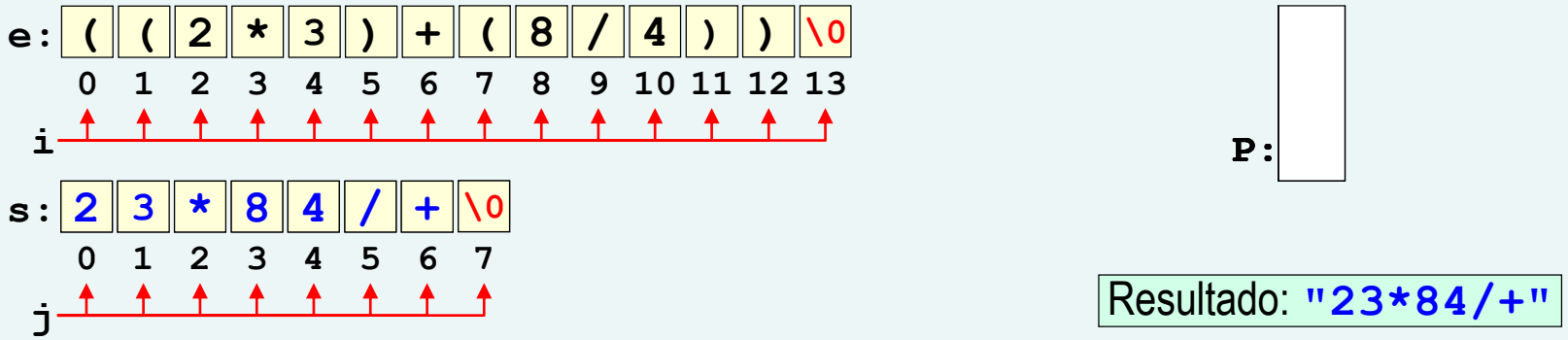
Conversão para posfixa

Expressões

Algoritmo de conversão (1ª versão)**:**

A conversão de expressão **infixa parentesiada** em posfixa é feita do seguinte modo:

- Inicie com uma pilha **`P`** e uma expressão posfixa **`s`** vazias.
- Para cada elemento da expressão infixa, da esquerda para a direita, faça :
  - Se for um parêntese de **abertura**, descarte-o.
  - Se for um **operando**, anexe-o à expressão posfixa **`s`**.
  - Se for um **operador**, insira-o na pilha **`P`**.
  - Se for um parêntese de **fechamento**, remova um item de **`P`** e anexe-o a **`s`**.
- No final, devolva **`s`** como resultado.



Expressões

Exemplo 4. Programa de conversão de infixa parentesiada [1ª versão]

```
#include <stdio.h>
#include <ctype.h>
#include <string.h>
#include "pilha.h"

char *posfixa(char *e) {
   static char s[256];
   int j = 0;
   Pilha P = pilha(256);
   for(int i=0; e[i]; i++)
      if( isdigit(e[i]) ) s[j++] = e[i];
      else if( strchr("+*-/",e[i]) ) empilha(e[i],P);
      else if( e[i]==')' ) s[j++] = desempilha(P);
   s[j] = '\0';
   destroip(&P);
   return s;
}
```

Expressões

Exemplo 4. Programa de conversão [continuação]

```
int main(void) {
    char e[513];
    printf("Infixa? ");
    gets(e);
    printf("Posfixa: %s\n", posfixa(e));
    return 0;
}
```

Exercício 1. Teste do programa

Digite e teste o programa do Exemplo 4, com as seguintes expressões infixas (coloque todos os parênteses necessários para que o programa funcione corretamente):

- `2 * 3 + 8 / 4`
- `9 - 5 - 1`
- `2 + 3 * 4 - 5`
- `(3 + 4) * (8 - 6) / 2`

Como proceder quando a expressão infixa não está **completamente** parentesiada?

Expressões

Exemplo 5. Conversão de infixa para posfixa: `2*(7+3*5)-4` [2ª versão]

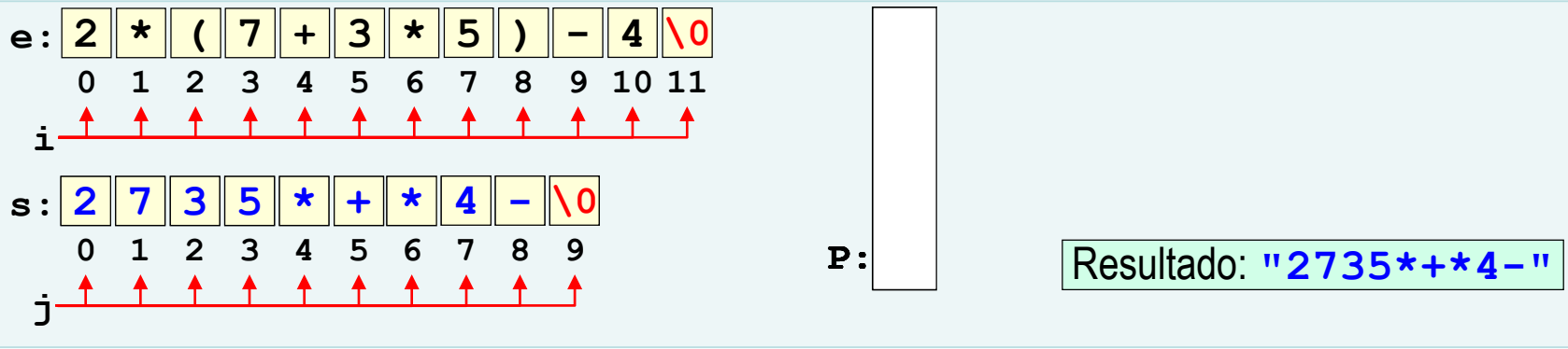
Elemento	Ação	Pilha	Posfixa
2	anexar à posfixa	[]	"2"
*	empilhar	[*]	"2"
(	empilhar	[*,(]	"2"
7	anexar à posfixa	[*,(]	"27"
+	empilhar	[*,(,+]	"27"
3	anexar à posfixa	[*,(,+]	"273"
*	empilhar	[*,(,+,*]	"273"
5	anexar à posfixa	[*,(,+,*]	"2735"
)	desempilhar e anexar até encontrar	[*,(]	"2735*+"
	'(' e, depois, descartá-lo	[*]	"2735*+"
-	desempilhar e anexar	[]	"2735*+*"
	empilhar	[-]	"2735*+*"
4	anexar à posfixa	[-]	"2735*+*4"
\0	desempilhar e anexar, até pilha vazia	[]	"2735*+*4-"

Expressões

Algoritmo de conversão (2ª versão)**:**

A conversão de expressão **infixa** em posfixa é feita do seguinte modo:

- Inicie com uma pilha **`P`** e uma expressão posfixa **`s`** vazias.
- Para cada elemento da expressão infixa, da esquerda para a direita, faça :
  - Se for um parêntese de **abertura**, empilhe-o em **`P`**.
  - Se for um **operando**, anexe-o à expressão posfixa **`s`**.
  - Se for um **operador**, enquanto o operador no topo tiver maior ou igual prioridade, desempilhe-o e anexe-o a **`s`**; depois, empilhe o operador encontrado na expressão.
  - Se for um parêntese de **fechamento**, enquanto o item no topo não for parêntese de abertura, desempilhe-o e anexe-o **`s`**; depois, desempilhe e descarte o parêntese.
- Enquanto a pilha **`P`** não estiver vazia, desempilhe um operador e anexe-o a **`s`**.
- Devolva **`s`** como resultado.



Expressões

Segunda estratégia para converter de infixa para posfixa:

- **Definir** as prioridades dos operadores (considerando o parêntese de abertura como operador).
- **Transcrever** a expressão, anexando os operandos na saída e empilhando operadores de alta prioridade sobre aqueles de baixa prioridade (se necessário, desempilhar e anexar operadores).

Exemplo 6. Prioridade dos operadores

```
int prio(char o) {
    switch( o ) {
        case '(': return 0;
        case '+':
        case '-': return 1;
        case '*':
        case '/': return 2;
    }
    return -1; // operador inválido!
}
```

O operador `'('` tem prioridade **máxima** na cadeia de entrada e prioridade **mínima** na pilha!

Expressões

Exemplo 7. Conversão de infixa em posfixa [2ª versão]

```
char *posfixa(char *e) {
   static char s[256];
   int j = 0;
   Pilha P = pilha(256);
   for(int i=0; e[i]; i++)
      if( e[i]=='(' ) empilha('(',P);
      else if( isdigit(e[i]) ) s[j++] = e[i];
      else if( strchr("+-/*",e[i]) ) {
         while( !vaziap(P) && prio(topo(P))>=prio(e[i]) )
            s[j++] = desempilha(P);
         empilha(e[i],P);
      }
      else if( e[i] == ')' ) {
         while( topo(P)!='(' )
           s[j++] = desempilha(P);
         desempilha(P);
      }                                          [continua...]
```

Expressões

Exemplo 7. Conversão de infixa em posfixa [continuação]

```
    while( !vaziap(P) )
        s[j++] = desempilha(P);
    s[j] = '\0';
    destroip(&P);
    return s;
}
```

Exercício 2. Teste do programa

Altere o programa do Exemplo 4, substituindo a função **`posfixa()`** pela segunda versão. Em seguida, teste-o com as seguintes expressões infixas:

- `2 * 3 + 8 / 4`
- `9 – 5 – 1`
- `2 + 3 * 4 - 5`
- `(3 + 4) * (8 - 6) / 2`

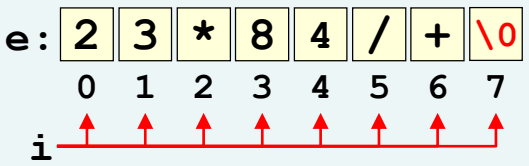
Avaliação da posfixa

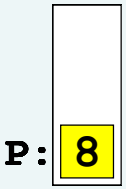
Expressões

Algoritmo de avaliação:

O valor de uma expressão **posfixa** pode ser calculado do seguinte modo:

- Inicie com uma pilha vazia **`P`**.
- Para cada elemento da expressão, da esquerda para a direita, faça:
  - Se for um **operando**, empilhe em **`P`** o seu valor numérico.
  - Se for um **operador**, desempilhe de **`P`** dois valores, aplique o operador a esses valores e empilhe em **`P`** o resultado obtido.
- No final, devolva como resultado o valor existente do topo de **`P`**.





Resultado: 8

Expressões

Exemplo 8. Avaliação da posfixa

```
int valor(char *e) {
   Pilha P = pilha(256);
   for(int i=0; e[i]; i++)
      if( isdigit(e[i]) ) empilha(e[i]-'0',P);
      else {
        int y = desempilha(P);
        int x = desempilha(P);
        switch( e[i] ) {
           case '+': empilha(x+y,P); break;
           case '-': empilha(x-y,P); break;
           case '*': empilha(x*y,P); break;
           case '/': empilha(x/y,P); break;
        }
      }
   int z = desempilha(P);
   destroip(&P);
   return z;
}
```

Expressões

Exercício 3. Programa completo

Crie um programa completo para ler uma expressão aritmética na forma infixa e exibir sua forma posfixa correspondente, bem como seu valor numérico.

Por exemplo, para a expressão infixa **`"2*3+8/4"`**, o programa deve apresentar como saída a forma posfixa **`"23*84/+"`** e o valor numérico **`8`**.

Exercício 4. Expressões booleanas

Com base nos algoritmos descritos, crie um programa para ler uma expressão booleana completamente parentesiada e exibir sua forma posfixa correspondente, bem como seu valor numérico.

Considere que as expressões são compostas por:

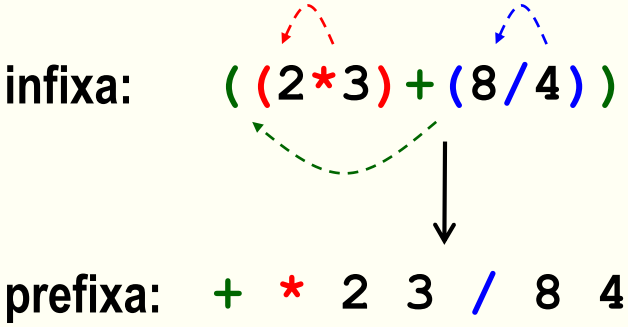
- **Operandos**: letras maiúsculas **`F`** e **`V`**, com valores numéricos **`0`** e **`1`**, respectivamente.
- **Operadores**: **`!`** (não), **`&`** (e) e **`|`** (ou), da maior para a menor prioridade.
- **Delimitadores**: parênteses de abertura e fechamento.

Por exemplo, para a expressão booleana infixa parentesiada **`"((!F)|(F&V))"`**, o programa deve apresentar como saída a forma posfixa **`"F!FV&|"`** e o valor numérico **`1`**.

Expressões

Exercício 5. Infixa para prefixa

Crie a função **`prefixa(e)`**, que devolve a forma prefixa da expressão aritmética completamente parentesiada **`e`**. Em seguida, faça um programa para testar a função.



Dicas:

- Percorra a expressão infixa da direita para a esquerda.
- Use **`strlen()`** para determinar o tamanho da expressão infixa.
- Use **`_strrev()`** para inverter uma cadeia de caracteres.

Exercício 6. Avaliação da forma prefixa

Crie a função **`valpre(e)`**, que devolve o valor da expressão aritmética prefixa **`e`**. Em seguida, faça um programa para testar a função.

Fim