UFRR

UNIVERSIDADE FEDERAL DE RORAIMA
CENTRO DE CIÊNCIA E TECNOLOGIA - CCT
DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO – DCC
DCC405 – ESTRUTURA DE DADOS II
PROF. ACAUAN C. RIBEIRO

DCC

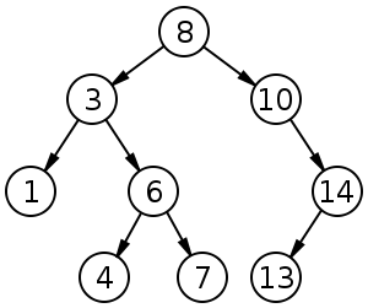
MATRÍCULA:	NOME:	NOTA:

Exercício Avaliativo 1

DATA: 11/04/2023

Percurso em pré-ordem (RED): visite o nó raiz, então visite recursivamente a subárvore esquerda, então a subárvore direita
Percurso em ordem (ERD): visita recursivamente a subárvore esquerda, visita o nó raiz e visita recursivamente a subárvore direita
Percurso em pós-ordem (EDR): visita recursivamente a subárvore esquerda, depois a subárvore direita e depois o nó raiz

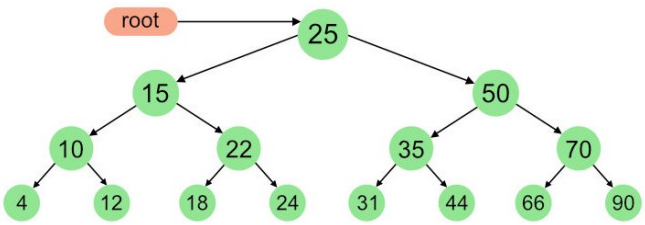
Questão 1) Analise as árvores seguintes e execute os percursos (travessias) solicitadas



In-order:

Pre-order:

Pos-order:



In-order:

Pre-order:

Pos-order:

Questão 2) Reconstrua a árvore binária a partir dos percursos apresentados

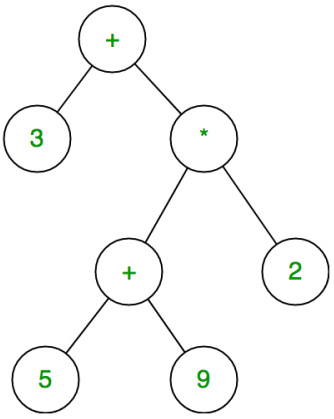
Árvore 1:
In-ordem: 5,6,8,9,10,20,25,26,31
Pré-ordem: 10,5,8,6,9,25,20,31,26

Árvore 2:
In-ordem: 2,5,10,15,20,22,25,50,51,55
Pré-ordem: 25,10,5,2,20,15,22,50,55,51

Árvore 3:
In-ordem: A C K M L J Z B F N D Y W E
Pré-ordem: B M K A C J L Z N F W D Y E

Árvore de Expressão

A **árvore de expressão** é uma árvore binária em que cada nó interno corresponde ao operador e cada nó folha corresponde ao operando, então, por exemplo, a árvore de expressão para 3 + ((5+9)*2) seria:



A travessia em ordem da árvore de expressão produz uma versão infixa de determinada expressão pós-fixada (o mesmo que a travessia em pós-ordem, ela fornece a expressão pós-fixada)

Avaliando a expressão representada por uma árvore de expressão:

```
Let t be the expression tree
If  t is not null then
      If t.value is operand then
                Return  t.value
      A = solve(t.left)
      B = solve(t.right)

      // calculate applies operator 't.value'
      // on A and B, and returns value
      Return calculate(A, B, t.value)
```

Construção da árvore de expressão:

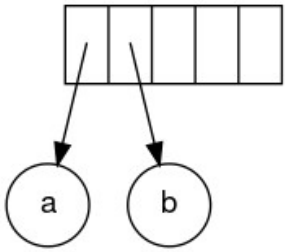
Agora, para construir uma árvore de expressão, usamos uma pilha. Fazemos um loop pela expressão de entrada e fazemos o seguinte para cada caractere.

1. Se um caractere for um operando, coloque-o na pilha
2. Se um caractere for um operador, retire dois valores da pilha, torne-os seus filhos e adicione o nó atual a pilha.
3. No final, o único elemento da pilha será a raiz de uma árvore de expressão.

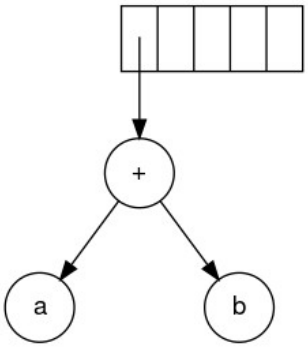
Exemplo:

```
Input:  a b + c d e + * *
Output: a + b * c * d + e
```

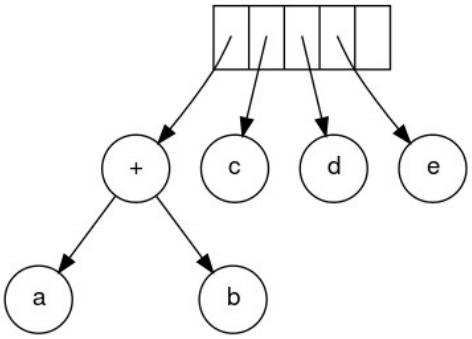
A entrada na notação pós-fixada é: **a b + c d e + * *** Como os dois primeiros símbolos são operandos, árvores de um nó são criadas e os ponteiros para elas são colocados em uma pilha. Por conveniência, a pilha crescerá da esquerda para a direita.



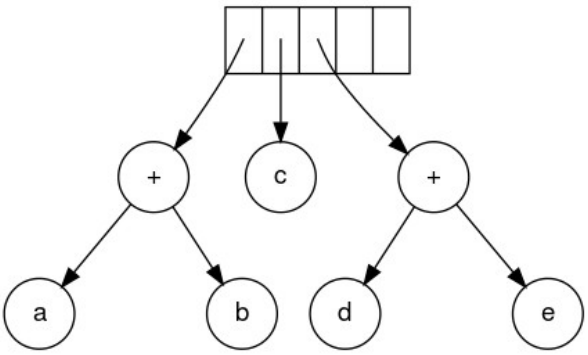
O próximo símbolo é um '+'. Ele abre os dois ponteiros para as árvores, uma nova árvore é formada e um ponteiro para ela é colocado na pilha.



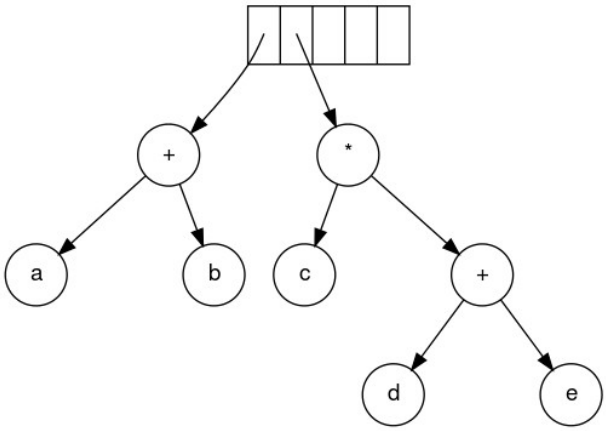
Em seguida, c, d e e são lidos. Uma árvore de um nó é criada para cada um e um ponteiro para a árvore correspondente é colocado na pilha.



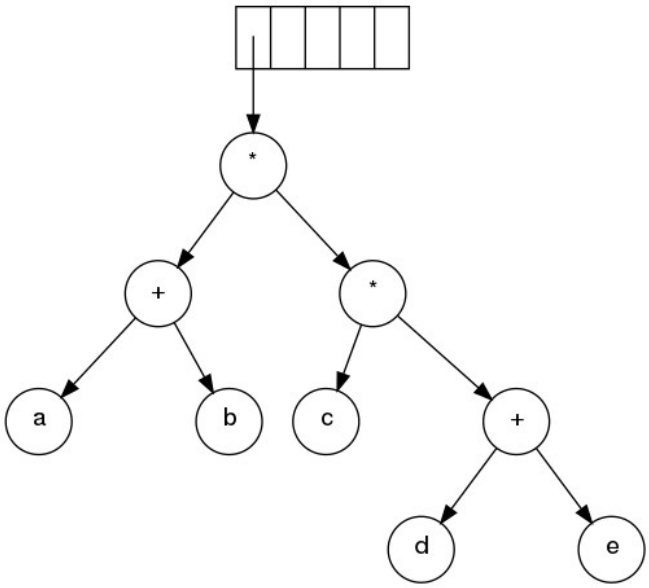
Continuando, um '+' é lido e mescla as duas últimas árvores.



Agora, um '*' é lido. Os dois últimos ponteiros de árvore são exibidos e uma nova árvore é formada com um '*' como raiz.



Finalmente, o último símbolo é lido. As duas árvores são mescladas e um ponteiro para a árvore final permanece na pilha.



Questão 3) Implemente um programa em python que tenha como entrada o percurso em pós-order de uma árvore de expressão e imprima o percurso in-order

Exemplo:
```
input(pós-order): 5 6 3 / + 6 3 / 7 - *
output(in-order): 5 + 6 / 3 * 6 / 3 – 7
```

Questão 4) Tendo como base a árvore criado na Questão 3) que estrutura uma expressão aritmética, Faça uma função chamada eval() que deve retornar a solução da expressão.

Exemplo:
```
input(pós-order): 5 6 3 / + 6 3 / 7 - *
output: -35
```