Calculadora de Expressões Aritméticas
scc0201 — Trabalho 04

Professor: Moacir A. Ponti
PAE: Edesio Alcobaça e Leonardo Ribeiro
Monitores: Giuliano Lourençon e João Vitor Tristão

1 Resumo

Implemente um programa que, dado uma sequência de prioridades de operadores e uma lista de expressões aritméticas, verifique se estas são expressões válidas e imprima todos os resultados na tela caso positivo, montando uma árvore de resolução das operações.

2 Descrição

O objetivo deste projeto é desenvolver uma calculadora de expressões aritméticas. As expressões aritméticas a serem avaliadas podem conter operações de soma, subtração, multiplicação, divisão, potenciação, radiciação, exponenciação e logaritmo representadas pelos símbolos +, -, *, /, **, sqrt(), exp() e log() respectivamente.

O seu programa deve aceitar o uso de parênteses, chaves e colchetes nas expressões e resolver as operações binárias de acordo com a precedência dos operadores e a ocorrência de parênteses. Considere as seguintes informações no desenvolvimento do projeto:

- Será listada uma sequência com os 5 operadores binários no início do programa, essa sequência definirá a prioridade dessas operações (primeiro operador da lista terá maior prioridade que os demais);
- Os operandos são valores reais e não negativos;
- Espaços em branco e tabulações presentes nas expressões devem ser desconsiderados;
- O programa realiza operações binárias (dois operandos) (+, -, *, /, **) e unárias (um operando) (sqrt(), exp(), log());
- Considere como ‘2’ o índice da radiciação (ou seja, raiz quadrada);
- Considere a base ‘2’ para as operações de logaritmo;
- Considere a base ‘e’ para as operações de exponenciação;
- A calculadora lê uma expressão por linha, terminada pelo caractere ‘,’ (vírgula);
- A cada expressão lida, a calculadora devolve a resposta em seguida;
- O final da execução é indicado pelo caractere ‘;’ (ponto e vírgula) no final de uma linha contendo uma expressão;
- As respostas devem exibir duas casas de precisão decimal com aproximação por truncamento;
- Considere que não haverá erros de overflow numérico;

- Considere que cada expressão terá, no máximo, 1.000 caracteres, incluindo dígitos, espaços, operadores e etc.
- A calculadora deve ler da entrada padrão e escrever na saída padrão. Caso uma expressão mal formada seja informada ou aconteça alguma operação cuja saída não pertença ao conjunto dos números reais, tais como divisão por zero, raiz quadrada negativa ou logarítmo não-positivo, sua calculadora deve exibir a seguinte mensagem de erro no lugar da resposta: 'Expressao incorreta.'

3 Árvores Binárias

Para a execução deste trabalho, você deverá utilizar a estrutura de dados conhecida como Árvore Binária (AB).

Você poderá ver mais sobre Árvores Binárias por esses links:

- `https://www.ime.usp.br/~pf/algoritmos/aulas/bint.html`
- `https://pt.wikibooks.org/wiki/Algoritmos_e_Estruturas_de_Dados/%C3%81rvores_Bin%C3%A1rias`

4 Fluxo do Projeto

Dividido em 4 partes principais:

4.1 Entrada de Dados

4.1.1 Sequência de Operadores

Primeiramente seu programa deverá ler uma sequência ordenada das 5 operadores binários (+,-,*,/,**). Essa será a prioridade das operações.

Por exemplo, em uma entrada onde a sequência for (*, -, /, **, +), as operações de multiplicação acontecerão antes das de subtração, que acontecerão antes das de divisão, e assim por diante.

Nessa mesma sequência, a expressão "2**2+3*5-9/3" dará resultado 6 (faça o teste).

CUIDADO COM OS PARÊNTESES, CHAVES E COLCHETES: toda expressão que estiver dentro de um desses delimitadores têm prioridade maior sobre as demais. Assim, a expressão "(2**[2+3])*5-{9/3}" dará resultado 157 (faça o teste).

Operadores unários não necessitam de prioridade, pois serão sempre tratados posteriormente à finalização de um delimitador, ou seja, quando não houver mais operações a serem realizadas dentro de seu delimitador, ele pode ser calculado.

A expressão "log{2*(5+21/7)}+sqrt[81]" dará resultado 13 (faça o teste).

4.1.2 Expressões

Após a listagem da sequência dos operadores, será listada uma sequência de expressões aritméticas com os operadores (unários e binários), números naturais e os delimitadores (parênteses, chaves e colchetes).

Cada expressão virá em uma linha e terminada com uma vírgula (,), porém a última expressão terminará com um ponto-e-vírgula (;).

4.2 Verificação de Erro

Algumas expressões podem vir mal-formatadas, e isso deve ser verificado antes de começar sua resolução.

Caso isso aconteça, uma mensagem de erro deverá ser enviada no lugar da resposta: 'Expressao incorreta.'

4.2.1 Delimitadores Mal-Formatados

Um dos grandes problemas que podem acontecer nas expressões é a má-formatação dos delimitadores por pelo menos um dos 3 seguintes motivos:

1. Delimitador fecha sem antes ser aberto: "2+3)*5"
2. Delimitador abre e não é fechado: "exp[4/2"
3. Delimitadores fora de ordem: "1+(2/{8/4)}"

Dica: Esse problema pode ser resolvido utilizando uma pilha.

4.2.2 Operadores Duplicados

Operadores binários necessitam de dois operandos. No caso de duplicação de operadores, o programa deve gerar um erro. Por exemplo, caso apareça "++" na expressão.

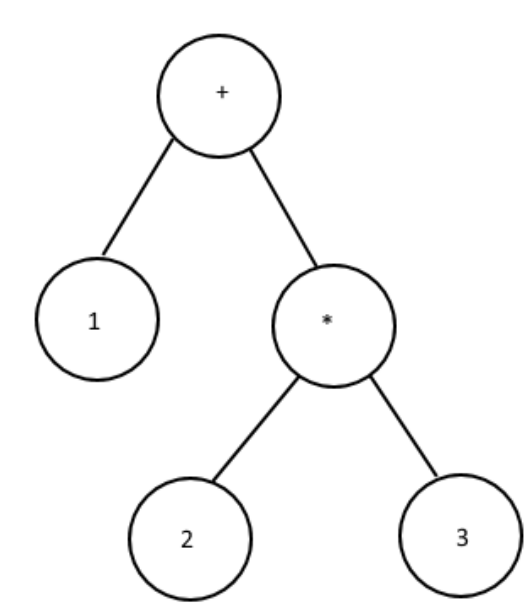
4.3 Montagem da Árvore Binária

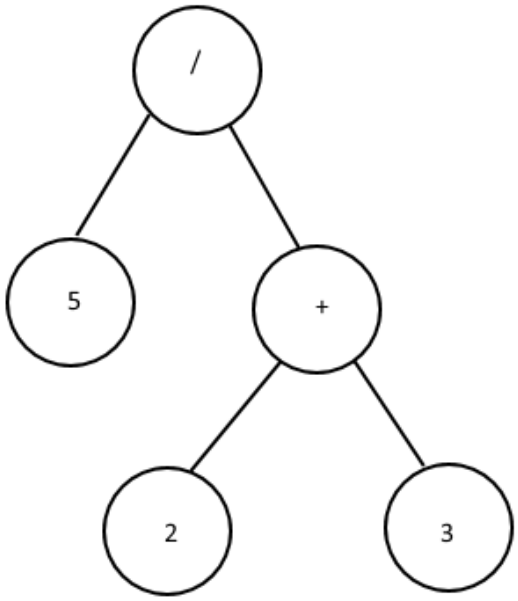
Se seu programa chegou até aqui, significa que sua expressão está válida e pronta para ser resolvida, mas para isso é preciso estruturá-la para que isso aconteça de uma maneira mais fácil.

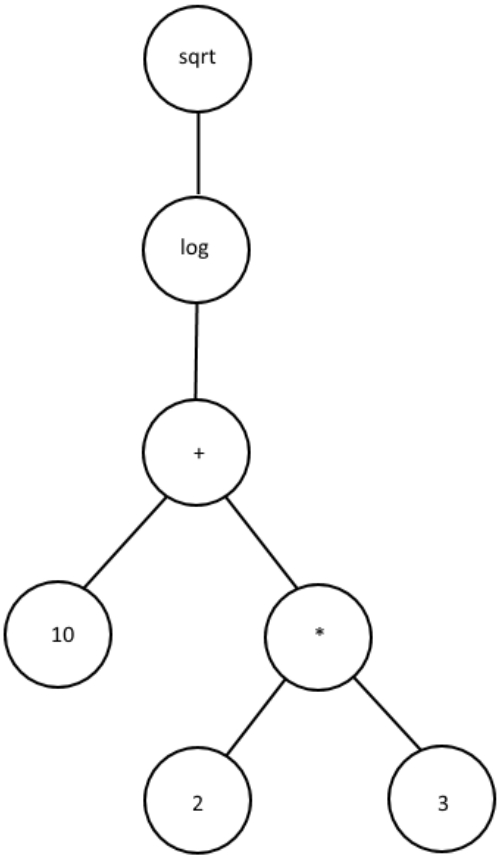
Uma das maneiras de se resolver este problema é o uso de Árvores Binárias, as quais podem ser utilizadas para transformar equações (com a prioridade padrão dos operadores) representadas na forma:

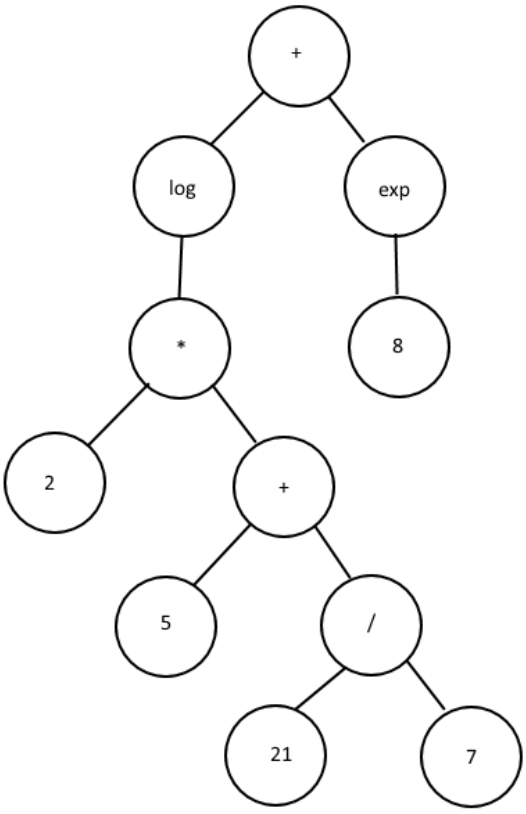
- 1+2*3
- 5/(2+3)
- sqrt[log{10+2*3}]
- log{2*(5+21/7)}+exp[8]

Por uma representação hierárquica, em árvore, na forma::









Vale notar que todos os nós folhas possuem números (operandos), enquanto os outros nós possuem operações com quantidade de nós filhos relacionada a seu tipo (unária ou binária). A resolução é sempre na direção das folhas para a raiz.

4.4 Resolução da Expressão

Com a árvore montada, o próximo e último passo é, finalmente, resolver a expressão e mostrar o resultado na tela no formato double com duas casas decimais.

Lembrando que algumas operações podem ser invalidadas nessa parte do fluxo (divisão por zero, raiz quadrada negativa, logarítmo não-positivo). Caso isso aconteça, uma mensagem de erro deverá ser enviada no lugar da resposta: 'Expressao incorreta.'

5 Exemplos de Entrada e Saída

5.1 Entrada

5.1.1 Exemplo 1

```
**
/
*
-
+

1+2*3,
5/(2-2),
sqrt[log{10+2*3],
log{2**(5+21/7)}+exp[3];
```

5.1.2 Exemplo 2

```
*
-
**
+
/

4**2-3,
5/2-7,
sqrt[log{10+2*3}],
log2**(5+21/7)}+exp[2-4];
```

5.1.3 Exemplo 3

```
+
-
*
/
**

1+3**3,
5*(2-1,
sqrt[log{3-2*1}],
log{2*(5--3/4)}+sqrt[0];
```

5.2 Saída

5.2.1 Exemplo 1

```
7.00
Expressao incorreta.
Expressao incorreta.
28.08
```

5.2.2 Exemplo 2

```
0.25
-1.00
2.00
Expressao incorreta.
```

5.2.3 Exemplo 3

```
64.00
Expressao incorreta.
Expressao incorreta.
Expressao incorreta.
```

6 Entrega e Avaliação

O trabalho será avaliado levando em consideração:

1. Realização dos objetivos / lógica utilizada
2. Uso de comentários e estrutura no código (e.g. identação, legibilidade, modularização)
3. Resultado da plataforma run.codes
4. Eficiência da implementação

ATENÇÃO:

- O projeto deverá ser entregue apenas pelo (`http://run.codes`) no formato de **ZIP Makefile**, ou seja um arquivo compactado com todos os códigos .c, .h e Makefile.
- O prazo está no sistema run.codes
- Em caso de projetos **copiados** de colegas ou da Internet, todos os envolvidos recebem nota zero. Inclui no plágio a cópia com pequenas modificações, cópia de apenas uma parte ou função. Portanto programe seu próprio trabalho.