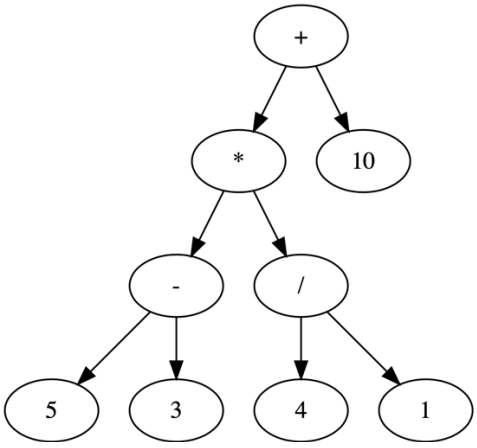
Universidade Federal do Espírito Santo – Departamento de Informática
Estruturas de Dados I (INF09292)
2º Trabalho Prático
Período: 2020/2 EARTE
Professora Patrícia Dockhorn Costa

Neste trabalho você implementará um programa para avaliar expressões aritméticas por meio de uma **Árvore Binária de Expressões Aritméticas**.

O que é uma árvore binária de expressões aritméticas?

Uma árvore binária de expressões aritméticas é uma estrutura de dados que permite avaliar (calcular) o valor da expressão por meio de sua organização hierárquica. Nesta árvore, nós folha são operandos (números) e nós internos representam os operadores (*, /, +, -). Neste trabalho, suportaremos apenas operadores binários, ou seja, que utilizam 2 números (ou duas sub-expressões, também binárias). Considere a expressão aritmética ((5-3)*(4/1))+10. A árvore de expressões equivalente a esta expressão é a seguinte:



Como montar a árvore a partir de uma expressão em formato de string?

O processo de interpretar uma *string* contendo uma expressão e a transformar em uma árvore binária é denominado de "*parsing*". Para simplificar a lógica do algoritmo do *parsing* dessas expressões, faremos uso extensivo de parênteses (por vezes, desnecessários):

- Todo número é cercado por parênteses;
- Todo operador e seus dois números serão cercados por parênteses;
- Todo operador e suas duas sub-expressões serão cercados por parênteses.

Portanto, a expressão anterior estaria no formato: ((((5)-(3))*((4)/(1)))+(10)).

O seu programa deve:

- ler as expressões de um arquivo texto de entrada (em formato de string);
- fazer o parsing das strings e montar as árvores binárias de expressões na memória;
- calcular os valores das expressões por meio das árvores; e,
- escrever os valores das expressões em um arquivo de saída.

Entrada do Programa:

Um arquivo texto contendo as expressões, uma por linha, no formato indicado acima. Para simplificar, assuma que não há erros de sintaxe nas expressões. Para facilitar, não trabalharemos com números negativos. Exemplo de arquivo de entrada (entrada.txt):

```
((((5)-(3))*((4)/(1)))+(10))
((((10)*(3))+(5))/((10)-(5)))
((((47)-(7))*(2))-(3))
((((120)*(2))-(100))*(10))
```

Saídas do Programa:

Em um arquivo "saida.txt": o valor calculado das expressões, uma por linha, seguindo a ordem do arquivo de entrada. Exemplo de arquivo de saída:

```
18
7
77
1400
```

Em um arquivo "graphviz.txt": além disto, é esperado um arquivo para visualizarmos as árvores formadas no graphviz (https://dreampuf.github.io/GraphvizOnline/), uma para cada expressão, como a seguir (graphviz.txt):

```
 strict graph {

no1[label="+"];
no1--no2;
no2[label="*"];
no2--no3;
no3[label="-"];
no3--no4;
no4[label=5];
no3--no5;
no5[label=3];
no2--no6;
no6[label="/"];
```

```
no6--no7;
no7[label=4];
no6--no8;
no8[label=1];
no1--no9;
no9[label=10];
}

strict graph {

    no1[label="/"];
    no1--no2;
    no2[label="+"];
    no2--no3;
    no3[label="*"];
    no3--no4;
    no4[label=10];
    no3--no5;
    no5[label=3];
    no2--no6;
    no6[label=5];
    no1--no7;
    no7[label="-"];
    no7--no8;
    no8[label=10];
    no7--no9;
    no9[label=5];
}

strict graph {

    no1[label="-"];
    no1--no2;
    no2[label="*"];
    no2--no3;
    no3[label="-"];
    no3--no4;
    no4[label=47];
    no3--no5;
    no5[label=7];
    no2--no6;
    no6[label=2];
    no1--no7;
    no7[label=3];
}

strict graph {

    no1[label="*"];
    no1--no2;
    no2[label="-"];
```

```
    no2--no3;
    no3[label="*"];
    no3--no4;
    no4[label=120];
    no3--no5;
    no5[label=2];
    no2--no6;
    no6[label=100];
    no1--no7;
    no7[label=10];
}
```

Nota Importante: acima ofereço uma forma de *serializar* a árvore (*in-orden*). Caso você queira usar outra, tudo bem. Desde que a árvore final esteja correta e possa ser visualizada no Graphviz.

Arquivos Esperados pelo *Classroom*:

1) TAD Árvore de Expressões (arv.h e arv.c)
2) Arquivo do cliente <nome_dupla.c>

Bom Trabalho!!!!