Atividade Avaliativa – RA1

Este trabalho pode ser realizado em grupos de até 4 alunos. **Grupos com mais de 4 alunos irão provocar a anulação do trabalho.** Você deve ler todo documento antes de começar e considerar o seguinte código de ética: *você poderá discutir todas as questões com seus colegas de classe, professores e amigos. Poderá também consultar os livros de referência da disciplina, livros na biblioteca virtual ou não, e a internet de forma geral e abrangente nos idiomas que desejar. Contudo, o trabalho é seu e deverá ser realizado por você. Cópias ensejarão a anulação do trabalho.*

OBJETIVO

Pesquisar e praticar. Pesquisar os conteúdos que irão complementar o material apresentado em sala ou nos livros sugeridos na ementa e praticar estes mesmos conceitos. Esta é uma oportunidade para aprimorar sua formação e se destacar. Uma avaliação com oportunidade de crescimento acadêmico e profissional.

DESCRIÇÃO DO TRABALHO

Seu objetivo será desenvolver um programa, usando Python, C, ou C++, capaz de abrir um arquivo de texto, contendo expressões e declarações simples, com uma expressão por linha e criar os analisadores léxico e sintático para a linguagem explicitada neste documento.

O programa que você irá desenvolver deverá ler o código em txt e gerar um *string* de *tokens* e a árvore sintática abstrata referente a cada um dos textos de código.

CARACTERÍSTICAS ESPECIAIS DA LINGUAGEM

As expressões serão escritas em notação RPN, segundo o formato a seguir:

a) Adição:+, no formato (A B +);
b) Subtração: - no formato (A B -) ;
c) Multiplicação: * no formato (A B *);
d) Divisão Real: | no formato (A B |);
e) Divisão de inteiros: / no formato (A B /);
f) Resto da Divisão de Inteiros: % no formato (A B %);
g) Potenciação: ^ no formato (A B ^);

Neste caso, A e B representam números reais. **Use o ponto para indicar a vírgula decimal.** As sitaxes definidas nas alíneas a até g definem as operações binárias desta linguagem.

Todas as operações serão executadas sobre números reais de meia precisão (16 bits/ IEEE754) CASO VOCÊ USE UM MICROPROCESSADOR DE 16 BITS, DEVE USAR PRECISÃO SIMPLES (32 bits/ IEEE754). CASO VOCÊ USE UM MICROPROCESSADOR DE 32 BITS DEVE USAR PRECISÃO DUPLA (64 bits/ IEEE754). CASO VOCÊ USE UM MICROPROCESSADOR DE 64 BITS DEVE USAR PRECISÃO QUADRUDUPLA (128 bits/ IEEE754). As únicas exceções são a divisão de inteiros e a operação resto da divisão de inteiros que devem ser realizadas em números inteiros. Além disso, **O expoente da operação de potenciação será sempre um inteiro positivo**. Todas as expressões desta linguagem podem ser aninhadas para a criação de expressões compostas, sem nenhum limite definido de aninhamento. Isto implica na possibilidade de que o texto lido contenha linhas com expressões como:

a) (A (C D *) +)
b) ((A B %) (D E *) /)

Onde A, B, C, D e E representam números reais. Além das expressões já definidas existem três comandos especiais (N RES), (V MEM) e (MEM), de tal forma que:

a) (N RES): devolve o resultado da expressão que está N linhas antes, onde N é um número inteiro não negativo;
b) (V MEM): armazena um valor, V, em um espaço de memória chamado de MEM, capaz de armazenar um valor real;
c) (MEM): devolve o valor armazenado na memória. Se a memória não tiver sido usada anteriormente devolve o número real zero. Cada arquivo de textos é um escopo de aplicação.

Além disso, mantendo o uso de parênteses você deve criar a sintaxe para os artefatos *if...then...else* e *for* comuns em muitas linguagens de programação. **Cabe a você criar a sintaxe que estes artefatos terão na sua linguagem desde que eles estejam entre parênteses. LEMBRE-SE A RECURÇÃO É UMA OPÇÃO PARA LAÇOS DE REPETIÇÃO.**

Para teste, você deverá fornecer, no mínimo 3 arquivos de texto contendo, no mínimo 10 linhas com expressões aritméticas. Estes arquivos deverão estar disponíveis no ambiente online Repl.it, no mesmo diretório do código fonte do seu programa. Ou no seu computador pessoal, desde que todo o código esteja postado em um repositório do GitHub, documentando e comentado em inglês.

Os testes devem incluir situações de erro, relatórios de correção, ou a explicitação do erro para que o usuário possa corrigir o código.

Cada arquivo de testes deve conter todas as expressões e declarações possíveis na linguagem.

Seu programa não pode ter um menu, ou qualquer forma de seleção entre os seus arquivos de teste. O programa deverá ser executado recebendo como argumento, na linha de comando, o nome do arquivo de teste.

Atenção: as primeiras linhas do seu código devem conter os nomes dos integrantes do grupo, em ordem alfabética, além do nome do grupo no ambiente virtual de aprendizagem (Canvas). A nota será lançada para o grupo.

TECNOLOGIAS ESSENCIAIS

Para fazer jus aos pontos referentes a este trabalho você deverá criar um analisador léxico baseado em uma máquina de estados finitos determinística e um analisador sintático cujo parser seja LL(1).

A documentação do seu trabalho deverá incluir o diagrama de transição da máquina de estados finitos, um link para um ambiente de simulação desta máquina e uma breve explicação do seu funcionamento com exemplos de aceitação e de não-aceitação.

A documentação do seu trabalho deverá incluir as regras de produção que você irá criar, os conjuntos FRIST e FOLLOW e a Tabela de Derivação utilizada na criação do parser LL(1).

ENTREGAS

Você deverá postar no Ambiente Virtual de Aprendizado da Instituição (Canvas) o link para o repositório do GitHub que contenha todo o seu código e toda a sua documentação. Durante a avaliação, o professor irá verificar a árvore sintática gerada, o *string* de *tokens* e a documentação.

AVALIAÇÃO

As operações aritméticas com ponto-flutuante, segundo a norma IEEE754 e as especificações do seu trabalho valem até 30% da nota.

O *string* de *tokens* gerado corretamente, desde que as mensagens de erro funcionem, vale até 30% da nota.

A árvore sintática criada com o parser LL(1) corretamente documentada e com as respectivas mensagens de erro vale até 40% da nota.