Atividade Avaliativa – RA4

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 o analisador semântico e a geração de código Assembly para a linguagem explicitada neste documento. Lembre-se a linguagem que será descrita a seguir **está incompleta e é baseada na linguagem** que usamos na avaliação da RA-1. **Nesta fase, você irá precisar gerar código em Assembly**.

O programa que você irá desenvolver usará a mesma linguagem que usamos até o momento, que por segurança, repito a seguir, acrescida dos artefatos de decisão e repetição que vocês criaram nas fases anteriores. Cada grupo criou um artefato diferente e precisa manter este mesmo artefato para esta fase.

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 sintaxes 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, nas fases anteriores você criou a sintaxe para os artefatos *de* tomada de decisão e repetição, comuns na maior parte das linguagens de programação. E, certamente, deve ter entendido que a recursão é uma opção aos laços de repetição.

Os textos de teste deverão conter valores literais inteiros e reais. Lembre-se, cada texto é um código e como tal deve ser considerado.

Para teste do seu trabalho, você deverá fornecer, no mínimo 3 arquivos de texto contendo, no mínimo 3 linhas com expressões aritméticas e artefatos que você criou. Observe que agora, você

pode criar programas. Para calcular o fatorial de um número, ou um determinado item da sequência de Fibonacci etc. Estes arquivos de teste deverão estar disponíveis no mesmo ambiente online, e 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. **Você precisará postar um link para este repositório no canvas para formalizar a entrega. Contudo, toda a avaliação será realizada apenas com a prova de autoria que será realizada na data da entrega.**

A DATA DA ENTREGA NÃO SERÁ POSTERGADA.

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

Seu programa, o analisador semântico com geração de código Assembly, não pode ter um menu, ou qualquer forma de seleção entre os seus arquivos de teste. O programa, o analisador semântico com geração de código Assembly, deverá ser executado recebendo como argumento, na linha de comando, o nome do arquivo de teste. E apenas o nome do arquivo de testes.

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 semântico que irá validar os tipos numéricos, inteiro e ponto flutuante, usando cálculo de sequentes e em seguida, gerar o código Assembly contendo o mesmo algoritmo dos arquivos de testes. Este código Assembly será executado na plataforma Arduino. **Lembre-se, sua linguagem inclui as operações que definimos no começo do trabalho e os artefatos que você criou para tomada de decisão e repetição.**

Você já gerou o string de tokens, analisador léxico, já gerou a árvore sintática, analisador semântico. Agora você terá que percorrer a árvore sintática gerada, uma árvore para cada arquivo de testes, validar os tipos, com typecasting automático (indicado na linha de comando) ou não e, a partir desta árvore, gerar o Assembly necessário a execução.

PREFERENCIALMENTE A SAÍDA DO ARDUINO DEVE SER FEITA VIA RS232 PARA LEITURA NO TERMINAL.

ENTREGAS, AVALIAÇÃO e PROVA DE AUTORIA

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.

Prova de Autoria: usando algum aplicativo gratuito, disponível online, será sorteado um dos alunos do grupo para representar o grupo na prova de autoria. Este aluno sorteado escolherá um número entre 1 e 10. O número escolhido corresponde a pergunta que o aluno deverá responder. Após a apresentação satisfatória do trabalho, durante a qual o aluno sorteado explicará como o programa criado funciona. Caso o aluno não responda, ou não consiga explicar o trabalho, o grupo perderá 35% da nota atribuída ao projeto. **AVALIAÇÃO, SERÁ REALIZADA APENAS DURANTE A PROVA DE AUTORIA. A POSTAGEM NO CANVAS É APENAS DOCUMENTAL.**

O trabalho será avaliado quanto à:

1. Entrega do analisador semântico vale até **70% da nota total. Com as seguintes ressalvas**:
    a) Não usou apenas um comando para fazer a análise semântica e gerar o código **perde 7%;**
    b) Não percorreu a árvore integral do texto de teste para o julgamento de tipos e para a geração de código **perde 35% da nota;**
    c) Não usou os artefatos de decisão e repetição, perde de **20%;**
    d) Não fez o julgamento de tipos perde **35%** da nota**;**
    e) Usou apenas números inteiros resulta na **perde 35%** da nota.
    f) Código Assembly não rodou no Arduino, ou o código gerado não representa o mesmo algoritmo (série de operações, valores, linhas e artefatos) do texto de teste **perde 70%** da nota.
2. Organização e legibilidade do código, o grupo **recebe 15% da nota**; A Organização será julgada por meio de análise da documentação do código. Feita no próprio código e nos textos extras que o grupo achar conveniente.
3. Robustez diante de casos complexos ou com erros **recebe 15%**. O professor irá modificar dois dos seus arquivos, um para testar a correção e outro para testar casos de erro.
4. Trabalhos iguais provocam o zeramento de todos os trabalhos identificados como iguais. Cabe ao professor, e apenas ao professor, definir o quanto um trabalho é igual ao outro. E este julgamento será feito durante a prova de autoria.