ICC043/IEC585 – Paradigmas de Linguagens de Programação 23/09/2019
Prof. Rafael Giusti

Trabalho Prático 1

1 Instruções gerais

- Este trabalho pode ser realizado em grupos de 2 a 5 discentes. A correção será mais estrita para grupos maiores. Recomenda-se grupos de 2 ou 3 integrantes.
- O tempo para a realização do trabalho será de um mês. Portanto a entrega deverá ser feita até ~~24 de outubro~~ 31/out/2019 (*atualização*: postergado em uma semana).
- Os trabalhos deverão ser entregues através do ColabWeb. Cada grupo deverá eleger um representante para fazer a submissão.
- A capa do trabalho deverá conter os nomes de todos os integrantes.

2 Descrição do trabalho

O objetivo do trabalho é estudar uma linguagem de programação, criticá-la e identificar aspectos de sua gramática. Para isso, cada grupo deverá escolher uma linguagem de programação até o dia 30/set/2019 e registrar no ColabWeb. Os grupos estão encorajados a escolherem linguagens variadas. Será permitido no máximo três grupos com uma mesma linguagem de programação.

O trabalho será dividido em três partes:

- Parte 1 (peso 4): análise crítica da linguagem escolhida, comparando-a com uma linguagem de referência "*mainstream*" (ver lista na Seção 2.1);
- Parte 1 (peso 1): Identificação da gramática de atributos e dos principais pontos da gramática da linguagem escolhida (ver Seção 2.2);
- Parte 1 (peso 3): Implementação de um analisador léxico (ver Seção 2.3).

Os grupos deverão entregar um relatório. Serão objeto de avaliação a qualidade do texto (clareza e aderência às normas do português) e cobertura dos objetivos. O analisador léxico será avaliado com respeito ao funcionamento e ao bom desenvolvimento—modularização, documentação interna, boas práticas de programação etc.

Os trabalhos serão avaliados com notas de 0 a 10, com pesos supracitados.

2.1 Parte 1 – Análise crítica

Para esta parte, cada grupo deverá escolher uma linguagem de estudo e uma linguagem de referência.

- A *linguagem de estudo* poderá ser qualquer linguagem de programação imperativa (predominantemente procedimental ou orientada a objetos). Deve ser indicada no ColabWeb;
- A *linguagem de referência* deve ser C, C++, Python ou Java.

Para completar esta etapa do trabalho, o grupo deve buscar na *web* a especificação da linguagem e também consultar as principais características da linguagem. O relatório deve apresentar uma discussão tão extensão quanto necessário para caracterizar a linguagem. Por exemplo, tópicos que podem ser abordados:

- Quando a linguagem foi criada e com qual objetivo?
- A linguagem foi criada para atender um nicho específico? Por exemplo, SNOBOL foi criada para manipulação de cadeias de caracteres, enquanto PHP foi criada para facilitar a construção de páginas *web*.
- A linguagem é multiparadigma ou favorece quase totalmente um paradigma específico? É uma linguagem predominantemente estática ou dinâmica?
- Outros...

Importante: o objetivo do trabalho é estudar linguagens de programação, preferencialmente linguagens modernas. Uma vez escolhida a linguagem, o grupo deve indicar um representante para preencher a base de dados "Grupos e Linguagens" no ColabWeb. Há um limite de **três grupos para cada linguagem**.

O grupo deve também escolher ao menos uma linguagem de referência. Deve constar no relatório uma discussão sobre a linguagem, tomando-se como base o conjunto de critérios e caraterísticas vistas em aula (facilidade de leitura, de escrita etc.). O objetivo é comparar elementos da linguagem escolhida contra a(s) linguagem(ns) de referência.

2.2 Parte 2 – Gramática

Nesta parte do trabalho, o grupo deverá identificar a gramática da linguagem escolhida. O relatório deve conter *pelo menos* as partes da gramática que permitem escrever programas contendo:

- Uma declaração/definição de função;
- Tipos de dados inteiros e de ponto flutuante (equivalente a `int` e `double` em C);
- Uma chamada de função (sendo possível retornar um valor);
- Um comando de seleção (`if`);

- Um comando de repetiação (`for`, `while` ou equivalente);
- Literais de inteiros e de pontos flutuantes (pelo menos o essencial, sendo opcional incluir sufixos, prefixos e bases);
- Variáveis e parâmetros dos tipos elencados;
- Expressões algébricas contendo pelo menos os operadores aritméticos e sub-expressões com parênteses.

O grupo também deverá identificar a gramática de expressões e discutir como a linguagem especifica precedência e associatividade de operadores. Se a linguagem é descrita com recursão explícita, então basta incluir a gramática de expressões no relatório.

O relatório deve conter a tabela de precedência e associatividade da linguagem. Compare a tabela com a da linguagem de referência. O número de níveis parece adequado? Argumente.

2.3 Parte 3 – Analisador léxico

O grupo deverá desenvolver um analisador léxico para a linguagem de programação escolhida.

O analisador léxico deverá ser utilizado através da linha de comando e deverá ler o programa de um arquivo ou a partir do teclado. Caso a leitura seja feita a partir de um arquivo, o analisador léxico deverá receber o nome do arquivo como argumento.

O analisador léxico deverá ser capaz de identificar programas que contenham os elementos descritos na gramaica da Seção 2.2. Isto é, não é necessário reconhecer o léxico completo da linguagem de programação escolhida, apenas do subconjunto anteriormente mencionado.

A saída do analisador léxico deverá ser uma sequência de pares `lexema -> token`. A saída deverá ser impressa na tela. Para simplificar a leitura, os lexemas deverão ser alinhados à direita e os *emph* à esquerda.

Por exemplo, se a linguagem de programação escolhida contiver as seguintes regras léxicas:

```
<inteiro> ::= 0
            | [-] <positivo> {<digito>}
<op_soma> ::= + | -
<op_mult> ::= * | / | %
<positivo> ::= 1 | 2 | ... | 9
<digito> ::= 0 | <positivo>
```

Então a saída para a sentença `-10 + 5-1` poderá ser

```
-10 -> LIT_INT
  + -> OP_SOMA
  5 -> LIT_INT
  - -> OP_SUB
  1 -> LIT_INT
```

Observe quea linguagem não faz diferenciação explícita dos operadores de soma e subtração, agrupando-os por “modalidade”. O mesmo ocorre com os operadores de “multiplicação”. É pro-

vável que a especificação da linguagem escolhida faça o mesmo. Não obstante, o analisador léxico deve ser capaz de distinguir esses *tokens*, portanto o grupo deverá estudar a especificação da linguagem com atenção.

O analisador léxico deve ser capaz de ignorar espaços em branco e comentários. Observe que, no exemplo anterior, alguns operandos estavam separados por espaços e outros não. O exemplo não inclui comentários, mas a linguagem de programação provavelmente terá suporte a eles. Comentários não são *tokens*!

O grupo deve tomar cuidado para que os comentários não causem junção de *tokens* sem relação. Por exemplo, a cadeia abaixo em C, C++ ou Java *não pode ser* interpretada como se fosse uma atribuição válida:

```
int num;
num = 12/*...*/34;
```

O analisador léxico deve ser capaz de identificar erros de sintaxe primitivos. Erros de sintaxe que podem ser capturados nessa etapa da análise do programa incluem um comentário de várias linhas que não foi fechado ou uma string que não foi fechada corretamente.

Em certas linguagens, uma literal numérica com prefixo incorreto também é um erro de sintaxe. Por exemplo: `0b15` é um erro em Python, pois literais que começam com `0b\` são números binários. O analisador léxico não precisa, entretanto, detectar esse tipo de erro, pois não fazem parte do conjunto essencial mencionado na Seção 2.2.

A analisador léxico pode ser feito em Python, C, C++, Java, Python ou Perl. Caso o grupo escolha implementar o analisador léxico em Python, não será permitido o uso de *notebooks*. O relatório deve conter instruções claras de compilação e/ou execução do analisador léxico. O código-fonte do analisador deverá ser submetido como um arquivo `.zip`, `.rar` ou `.gz` e deverá incluir programas de teste cobrindo todos os *tokens* mencionados na Seção 2.2. O relatório deverá conter uma explicação das decisões de projeto na implementação do analisador e um programa de exemplo, bem como a saída produzida pelo analisador.

A avaliação do analisador levará em conta o bom funcionamento do programa, a cobertura das especificações do trabalho e a adoção de boas práticas de programação (código organizado, bem documentado, corretamente indentado e com nomes de variáveis e funções descritivas).