Descrição do Trabalho Prático 01 de Compiladores

Eduardo Miranda

Outubro 2024

1 Introdução

O trabalho prático (TP) consiste na implementação de um **Analisador Léxico (Parser)** para uma versão simplificada da linguagem de programação Java. Tal analisador pode ser implementado em qualquer linguagem de programação.

2 Proposta de TP

Conforme mencionado, o TP proposto consiste em implementar analisador léxico para uma simplificação da linguagem Java, chamaremos essa nova linguagem de Java−−.

2.1 Sobre o Parser

O programa resultante deve possibilitar que o usuário informe um arquivo escrito na linguagem Java−− e retorne uma lista com os seus respectivos (token, lexama, linha e coluna) para cada token encontrado no arquivo.

2.1.1 Possíveis tipos de Tokens

O parser poderá possuir os seguintes tipos de tokens:
Operadores aritméticos:

- Adição: +;
- Subtração: -;
- Multiplicação: *;
- Divisão: /;
- Módulo: %

Operadores lógicos:

- OU lógico: ||;
- E lógico: &&;
- NÃO lógico: !;

Operadores relacionais:

- Igualdade: ==;
- Diferença: !=;
- Maior que: >;
- Maior igual que: ≥;
- Menor que: <;
- Menor igual que: ≤;

Operadores de atribuição:

- Atribuição: =
- Atribuição com soma: +=
- Atribuição com subtração: -=
- Atribuição com multiplicação: *=
- Atribuição com divisão: /=
- Atribuição com módulo: % =

Palavras reservadas:

- **int:** identifica declaração de variáveis inteiras;
- **float:** identifica declaração de variáveis ponto flutuante;
- **string:** identifica declaração de variáveis do tipo string;
- **for:** identifica início de bloco de repetição do tipo for;
- **while:** identifica início de bloco de repetição do tipo while;
- **break:** chamada para sair de um laço de repetição;
- **continue:** chamada para pular uma iteração de um laço de repetição;
- **if:** identifica início de bloco de condicional;
- **else:** identifica segunda parte do bloco condicional caso não verdadeiro;
- **return:** chamada para retornar um valor e no nosso caso encerrar o programa

- **system:** palavra prévia para comandos de entrada e saida;
- **out:** indicador para comando de saida;
- **print:** chamada para mostrar alguma informação na tela;
- **in:** indicador para comando de entrada;
- **scan:** chamada para ler alguma informação do teclado;

Símbolos aceitos (Separadores ou delimitadores):

- **;** - ponto e virgula;
- **,** - virgula;
- **{** - abre chaves;
- **}** - fecha chaves;
- **(** - abre parênteses;
- **)** - fecha parênteses
- **.** - ponto final

Strings:

- Toda string deve começar e terminar com o caractere **"**.
- Strings podem contem o caractere '\', dependendo da linguagem será necessário algum tipo de tratamento especial.

Números:

- Números inteiros DECIMAIS (Base 10) são escritos seguindo a seguinte expressão regular: $[1..9][0..9]^+$
- Números inteiros OCTAIS (Base 8) são escritos seguindo a seguinte expressão regular: $0[0..9]^+$
- Números inteiros HEXADECIMAIS (Base 16) são escritos seguindo a seguinte expressão regular: $0x([0..9] \cup [A..F])^+$
- Números flutuantes são escritos seguindo a seguinte expressão regular: $[0..9]^+.[0..9]^*$

  Observação: Como a expressão regular dos números flutuantes pode não ter nada na segunda parte e isso pode tornar mais complexa as próximas etapas. Transformaremos a expressão em $[0..9]^+.[0..9]^*0$ visto que adicionar um '0' a direita depois da virgula não vai modificar nosso número original. Esse zero será adicionado somente a nível de lexema, o fonte original não tem a obrigação de colocar esse zero no fim de um número flutuante.

Variáveis:

- As variáveis (identificadores) são escritas seguindo a seguinte expressão regular:

$$([a-z] \cup [A-Z])([a-z] \cup [A-Z] \cup [0-9])^*$$

Observação: Esta expressão regular que gera os lexemas das variáveis dos programas também gera as palavras reservadas, logo será necessário criar tratamento especificos para identificação dos tokens corretos.

3 Algumas Dicas

Durante o desenvolvimento, é importante não se perder nos detalhes. Portanto, é recomendado que a implementação inicie somente pelas funcionalidades básicas. Só depois de garantir que as funcionalidades básicas estão funcionando conforme planejado, deve-se considerar a implementação de melhoramentos e funcionalidades adicionais. Também recomenda-se que trechos mais complicados do código sejam acompanhados de comentários que esclareçam o seu funcionamento/objetivo/parâmetros de entrada e resultados.

O TP também será avaliado em termos de:

- Modularização;
- Legibilidade (nomes de variáveis significativos, código bem formatado, uso de comentários);
- Consistência (formatação uniforme);
- Otimização de Código (quão otimizado o código está, se as estruturas de dados utilizadas são as melhores, checagens desnecessárias, etc...);

4 Funcionalidades

As seguintes funcionalidades devem estar funcionando corretamente ao entregar o trabalho prático:

4.1 Funcionalidade 1

Retornar uma lista de (token, lexema, linha, coluna) ao executar o parser para algum arquivo escrito na linguagem Java−− arbitrário.

4.2 Funcionalidade 2

Permitir que o arquivo a ser compilado seja informado via terminal, exemplo:

```
python3 <analisadorLexico.py> <arqTeste.java>
```

5 Submissão

Além de serem submetidos via link no Portal Didático da disciplina, o TP também deve ser enviado para o endereço eletrônico `eduardomiranda@cefetmg.br`, tendo como assunto TP_Compiladores_2024_2. No corpo do email deve aparecer o nome completo de cada integrante do grupo e um arquivo .zip com o programa e o manual de utilização.