INSTITUTO FEDERAL
Paraná
Campus Pinhais

Bacharelado em ciência da computação
Linguagens formais e autômatos
Trabalho, parte 1: análise léxica 24 de abril de 2025

1 Análise léxica

Um compilador é um programa que traduz código escrito em uma linguagem de programação para outra linguagem. Geralmente, a linguagem para a qual o código é traduzido é a linguagem de máquina (*Assembly*), que pode ser mais diretamente traduzida para um arquivo no formato executável pelo conjunto ⟨sistema operacional + processador⟩. Entretanto, vale ressaltar que nada impede que a tradução seja para outra linguagem de programação de alto nível.

O processo de compilação consiste de diversas etapas que, de certa forma, interpretam o código fonte em diferentes aspectos. Essa interpretação pode então ser usada para gerar o código na linguagem "de destino" da tradução.

A primeira etapa é chamada de *análise léxica*, que transforma o código em uma sequência de *tokens* ou *lexemas*. Os *tokens* são a menor parte significativa de uma linguagem de programação, como as palavras-chave, identificadores, constantes numéricas, operadores, ...

2 Analisador léxico da linguagem C−−

Sua tarefa neste momento é implementar o analisador léxico para uma linguagem fictícia chamada *C−−*[1]. Para isso, você deverá utilizar seus conhecimentos de expressões regulares e uma ferramenta chamada *Flex*.

Seu analisador deverá ler um código em C−− da entrada padrão e, para cada *token* encontrado, imprimir qual o lexema identificado. Você também deve imprimir mensagens de erro quando *tokens* desconhecidos forem encontrados. As mensagens de erro devem conter a linha do código-fonte em que o problema ocorreu.

3 A linguagem C−−

Nossa linguagem será um subconjunto da linguagem C. Seu analisador deve identificar os seguintes *tokens*:

- Identificador: Nomes de variáveis ou funções, são sequências de letras, números e *underscores* (_). Um identificador não pode iniciar com números;
- Palavras-chave: A linguagem possui algumas palavras-chave usadas para construir os comandos e expressões da linguagem. São elas: `break, continue, else, for, if, return, struct, while`;
- Tipos de dados: A linguagem possui os seguintes tipos de dados, usados para variáveis, argumentos de função e retorno de função: `char, int, long, short, void`;
- Constantes numéricas: A linguagem aceita números inteiros apenas;

[1] Nossa linguagem, obviamente, foi baseada em C. Você pode conferir as definições dos elementos léxicos da linguagem C em `https://www.gnu.org/software/gnu-c-manual/gnu-c-manual.html#Lexical-Elements`. A leitura da seção é recomendada para que o estudante perceba que, ao mesmo tempo que não tiramos muita coisa da linguagem, existem algumas situações que seriam mais complexas de tratar

- Constantes alfanuméricas: A linguagem deve aceitar constantes de caractere (entre aspas simples) e constantes de string (entre aspas duplas). Nessas constantes, são aceitos letras, números e os símbolos `!, @, #, $, %, ¨, &, *, (, ), -, =, +, \`. A linguagem não aceita strings que possuem quebra de linha;
- Operadores: São divididos em operadores aritméticos, lógicos e de comparação. São eles: `+, -, *, /, %, ||, &&, ==, !=, <, <=, >, >=, !`;
- Operador de atribuição: `=`;
- Delimitadores ou separadores: `(, ), [, ], {, }, ;, ,`

A linguagem deve ignorar espaços em branco e tabulações. Lembre-se de contar as linhas quando aparecerem quebras de linha.

4 O *Flex*

A ferramenta *GNU Flex*[2] é um gerador de analisadores léxicos. Você deve criar um arquivo texto que conterá suas regras de expressões regulares e código C auxiliar.

Um exemplo de arquivo *Flex* é o que segue abaixo:

```
%{
int num_lines = 0, num_chars = 0;
%}

%option noyywrap

%%

\n      { ++num_lines; ++num_chars; }
.       { ++num_chars; }

%%

int main()
{
 yylex();
 printf( "# of lines = %d, # of chars = %d\n", num_lines, num_chars );
}
```

Note que a primeira seção, antes de %%, contém definições de código em C (entre %{ e %}) e outras definições na sintaxe do próprio *Flex*. Aqui você pode declarar variáveis e funções que você quer usar ao longo do seu analisador. Na segunda seção, você deve declarar as regras de expressões regulares e as ações a serem tomadas ao reconhecer cada expressão. As ações também são expressões em C.

Na última seção, você deve declarar a função `main`, que deve chamar a função `yylex()`, que faz a leitura e correspondência das expressões regulares.

[2] `https://ftp.gnu.org/old-gnu/Manuals/flex-2.5.4/html_mono/flex.html`

4.1 Compilando arquivos *Flex*

Se chamarmos o arquivo do exemplo anterior de `example.lex`, podemos executar `flex example.lex` para converter o arquivo *Flex* em um arquivo C, chamado `lex.yy.c`. Esse arquivo pode ser compilado normalmente e, ao executar, fará a correspondência das expressões regulares e executará as ações programadas por você.

Recomendo a "leitura" do arquivo `lex.yy.c` gerado a partir do exemplo acima. Note como um exemplo terrivelmente simples gerou um programa enorme. Note como o programa não é, de forma alguma, dos mais convenientes, bonitos, ou seguidor das boas práticas de programação. Por fim (e o mais importante), tente encontrar trechos do programa que parecem descrever um autômato finito.

5 O trabalho

Essa é a parte 1 do trabalho da disciplina. O resultado desta primeira parte será incrementada, em momento oportuno.

O trabalho pode ser realizado no máximo em duplas.

6 O que entregar

Você deverá entregar, até 29/abril (conforme calendário), um arquivo *.zip* contendo uma pasta, que por sua vez contém:

- O arquivo `.lex` que tiver até o momento. O arquivo pode estar incompleto, mas não pode estar vazio. Lembre-se que você deveria usar o horário de aula e da APS para realizar esse trabalho;
- Um arquivo de texto chamado `README.md` contendo o nome completo dos integrantes do grupo e quaisquer outras informações sobre o progresso do trabalho que julgarem relevantes;
- Os arquivos que o grupo usou para testar o analisador léxico (arquivos *.cmm*).

O arquivo compactado descrito acima deve ser nomeado com as iniciais do nome de cada integrante separados por um hífen. Por exemplo: se o Fulano da Silva fez o trabalho com o João de Souza, o arquivo deve ser nomeado `fs-js.zip`. A pasta, dentro do arquivo compactado, deve ter o mesmo nome, a não ser pela extensão.

Esse trabalho não será avaliado individualmente, e serve apenas como APS e como início ao trabalho da disciplina. Este sim, será avaliado e produzirá conceito.