Analisador léxico[1]

Alexandre Yuji Kajihara[1]

Universidade Tecnológica Federal do Paraná – UTFPR

DACOM – Departamento Acâdemico de Computação do Curso de Bacharelado em Ciência da Computação
Campo Mourão, Paraná, Brasil

[1]alexandre.ykz@gmail.com

Resumo

Para desenvolver um compilador é necessário várias etapas, e uma delas é o analisador léxico, que tem como objetivo identificar as pecularidades de uma determinada linguagem de programação. O método utilizado foi definir uma linguagem de programação que no nosso caso foi o T++, para que possamos montar o nosso analisador léxico. Depois de definido a linguagem verificamos como se representa um identificador, um comentário, um espaço na linguagem escolhida. Também averiguamos quais são palavras reservadas, símbolos, tipos que a linguagem possui. Após isso, definimos a linguagem de programação para desenvolver o analisador que foi a linguagem C, a ferramenta que irá nos auxiliar que foi o Flex e por fim, o JFLAP que foi utilizado para montarmos o nosso autômato. O resultado que tivemos foi que conseguimos identificar os identificadores, palavras reservadas, tipos, comentários, símbolos, espaços e valores do código-fonte. A conclusão é que essa primeira etapa para se construir um compilador é bem simples, já que apenas temos que identificar o que é cada caractere, palavra, símbolo ou valor do código-fonte. O relatório está dividido em oito partes que são: introdução, objetivo, fundamentação, materiais, procedimentos e resultados, discussão dos resultados, conclusões e referências.

1. Introdução

Para montar o analisador léxico foi necessário saber como é composta a linguagem T++, como quais são as palavras reservadas, símbolos, tipos, valores, como descrever os identificadores, comentários, espaço. Além disso, precisamos relembrar alguns conceitos como de autômatos finitos, expressões regulares e algoritmos. Com esses conhecimentos representamos as expressões regulares com base nas características que a linguagem foi definida, representamos o autômato com todas as pecularidades da linguagem, e com o algoritmo conseguimos identificar a composição da linguagem.

2. Objetivo

O objetivo de desenvolver o analisador léxico é conseguir identificar o que representa certas palavras ou símbolos dos código-fonte de um programa feito em T++. Por exemplo, quando encontramos a palavra repita no código-fonte devemos imprimir na tela que repita é uma palavra reservada da linguagem. Identificando o que cada palavra ou símbolo representa será útil para próxima etapa para se desenvolver um compilador que é o analisador sintático.

3. Fundamentação

Os fundamentos necessários para implementar esse o analisador léxico foi expressões regulares, autômato finito e algoritmos. A expressão regular auxiliou na representação as pecularidades da linguagem, como identificadores na linguagem T++ tem que obrigatoriamente começar com uma letra, e podem ter n letras ou n dígitos. Com os autômatos conseguimos representar todas as características da linguagem em um único autômato. Em relação, com o algoritmo precisamos ter o conhecimento de linguagem de programção que foi escolhida, que no nosso caso foi a linguagem C, em que só usamos seus recursos básico, como um laço de repetição e a função de imprimir na tela. Além da linguagem C, tivemos que aprender a utilizar o LEX que é uma ferramenta que auxilia a gerar o analisador léxico, porém a estrutura de um arquivo LEX é algo bem simples.

4. Materiais

Os materiais utilizados foram um *laptop* com o Sistema Operacional Ubuntu 16.10 64 bits com a configuração Intel Core i7-6500U, 16 GB de memória RAM (*Random Access Memory*), um compilador para linguagem C que foi o GCC na versão 6.2.0, o LEX na

1 Trabalho desenvolvido para a disciplina de BCC36B – Compiladores

versão 2.6.1, gedit na versão 3.22.0, alguns códigos em T++ e o JFLAP versão 7.0 para montar o autômato. Com o gedit que é um editor de texto escrevemos alguns códigos na linguagem T++ e o arquivo scanner.l que é o que é aceito pelo LEX. Com o GCC e LEX, criamos nosso analisador léxico que aceita linguagem T++.

5. Procedimentos e resultados

Para começar o analisador léxico descobrimos quais seriam as pecularidades da linguagem T++. Em que a mesma só aceita valores naturais, inteiros, flutuantes, exponenciais, arranjos unidimensionais e bidimensionais. Além disso, em uma função quando não é declarado nenhum tipo, podemos assumir que é uma função do tipo void, ou seja, não retorna nada. Ainda sobre as características da linguagem temos que as variáveis são locais ou globais, temos que os comentários começam com chaves e terminam com chaves, que os identificadores começam obrigatoriamente com uma letra e podem ter n letras ou n dígitos, e que temos quatro tipos de espaço, que é o espaço em branco, \n, \r e o \t. As duas últimas características da linguagem, seria as palavras reservada que seriam: se, então, senão, fim, repita, flutuante, retorna, até, leia, escreve, inteiro e os símbolos presentes na linguagem T++ que são: adição, subtração, multiplicação, divisão, igual, vírgula, atribuição, maior, maior igual, menor, menor igual, diferente, abre parênteses, fecha parênteses,
dois pontos, abre colchetes, fecha colchetes, abre chaves e fecha chaves.

Após sabermos tudo isso à respeito da linguagem T++ escrevemos três programa, que são teste1.tpp que é um programa que resolve fatorial, teste2.tpp que também é um programa que resolve fatorial, mas de uma maneira diferente do teste1.tpp e o teste3.tpp que é um selection sort, todos seguem as características da linguagem.

Após todas essas etapas, criamos o arquivo scanner.l que é o que é compatível com LEX. Esse arquivo ele tem algumas particularidades que devem ser respeitadas, mas resumindo ele combina algumas características deles para reconhecer as expressões regulares e as características da linguagem C. Então, definimos as expressões regulares que irão identificar cada umas dessas características da linguagem T++ e imprimimos na tela o caractere ou a palavra e classificamos ele como podendo ser um símbolo, identificador, palavra reservada, inteiro, flutuante, comentário, etc através da função print da linguagem C. Abaixo podemos ver na Tabela 1 as expressões regulares que criamos.

Tabela 1: expressões regulares da linguagem T++.

	Expressões regulares
Dígito	[0-9]
Letras	[a-zA-ZãáàÃÁÀõÕóÓéÉíÍúÚ_]
Natural	{Dígito}+
Inteiro	{Natural}
Flutuante	{Inteiro}("."{Natural})
Exponencial	({Inteiro}|{Flutuante})("e"|"E") {Inteiro}
Reservada	("se"|"então"|"senão"|"fim"|"repita"|"flutuante"|"retorna"|"até"|"leia"|"escreve"|"inteiro")
Símbolo	("+"|"-"|"*"|"/"|"="|","|":="|"<"|">"|"<="|">="|"<>"|"("|")"|":"|"["|"]"|"{"|"}")
Identificador	{Letras}({Letras}|{Dígito})*
Comentário	("{")({Letras}*|{Dígito}*|(" ")*|{Símbolo}*|("."|"!"|"@"|"#"|"$"|"%")*)*("}")
Espaço	[\n\r\t]+

Algumas informações que vale a pena ressaltar que escrevemos “Dígito” na expressão regular para evitar a repetição de escrever [0-9]. Além disso, o caractere ‘|’ representa uma opção, por exemplo em símbolo pode ser + ou – ou os outros caracteres. Outros dois caracteres que vale a pena ressaltar é o ‘+’ e ‘*’ em que o ‘+’ significa que deve se repetir pelo menos uma vez ou mais, já o ‘*’ ou fecho de Kleen siginfica que podemos repetir zero vezes ou mais. Um detalhe que irá nos auxiliar na próxima etapa que é o analisador sintático é colocar valores para cada símbolo, palavra reservada, comentário, etc, que está representado em um enum (que seria um conjunto de valores inteiros representados por identificadores), logo no começo do arquivo scanner.l. O último detalhe presente no arquivo scanner.l é que a função yylex é ela que faz a classificação do conteúdo do código-fonte.

Ciente de todas essas informações, podemos compilar os nossos arquivos, que envolve algumas etapas. Primeiramente, compilamos o arquivo LEX com o seguinte comando “lex scanner.l” em que ele resultará em um arquivo chamado lex.yy.c que será compilado pelo GCC, com o seguinte comando “gcc lex.yy.c -o scanner”, após isso gera se um executável chamado scanner. Agora para executarmos é só dar o seguinte comando “./scanner nome_do_arquivo_teste”. Para evitar de fazer todo esse processo quando mexemos no scanner.l fizemos um Makefile, a fim de poupar tempo. Na Figura 1, podemos ver uma parte do resultado em que conseguimos identificar um identificador, símbolo, palavra reservada.

```
<RESERVADA, leia>
<SÍMBOLO, (>
<IDENTIFICADOR, n>
<SÍMBOLO, )>
<IDENTIFICADOR, escreva>
<SÍMBOLO, (>
<IDENTIFICADOR, fatorial>
<SÍMBOLO, (>
<IDENTIFICADOR, fatorial>
<SÍMBOLO, (>
<IDENTIFICADOR, n>
<SÍMBOLO, <>
<INTEIRO, 1>
<SÍMBOLO, ,>
<IDENTIFICADOR, a>
<SÍMBOLO, )>
<SÍMBOLO, ,>
<IDENTIFICADOR, n>
<SÍMBOLO, +>
<INTEIRO, 2>
<SÍMBOLO, )>
<SÍMBOLO, )>
<RESERVADA, fim>
```

Figura 1: resultado parcial do analisador léxico usando o teste1.tpp.

Com essa etapa pronta montamos o nosso autômato no JFLAP usando as expressões regulares, com algumas modificações. Por exemplo, para identificar um identificador que deve ter começar obrigatoriamente com uma letra, fizemos que só vai para o estado de aceitação se tiver essa letra. Isso foi feito para substituir o símbolo ‘+’, que garante que tenha pelo menos um ou mais letras. Abaixo podemos ver o autômato com as pecularidades da linguagem T++, como podemos ver na Figura 2.

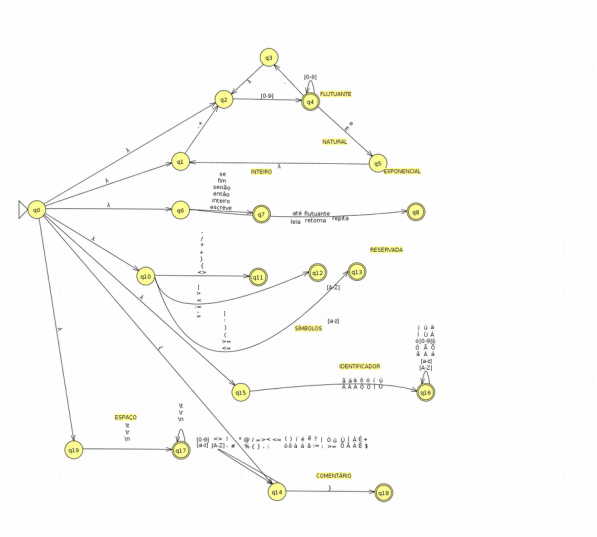

Figura 2: autômato da linguagem T++.

O resultado é que em todos os nossos três testes da linguagem T++, conseguimos fazer com que o nosso analisador léxico conseguisse identificar e classificar tudo que estava presente no código-fonte.

6. Discussão dos resultados

Após os resultados do nosso programa vimos que conseguimos realizar o analisador léxico com sucesso, e que em nossos três testes conseguimos identificar e classificar todo o código-fonte. Podemos afirmar, que essa primeira etapa de identificar cada caractere, símbolo e classificar eles é uma tarefa bem simples. Além disso, tratamos de alguns casos especiais, como o de comentário no qual podemos ter comentário dentro de outro comentário, o de exponencial em que pode ter um “e” ou “E”, para representar a exponenciação. Caso esteja presente um comentário dentro de outro comentário ou uma exponenciação com um desses dois caracteres, o nosso analisador léxico conseguirá classificar cada um deles.

7. Conclusão

Concluimos, que essa primeira etapa para se construir um compilador é muito importante e fundamental. Isso porque, é o analisador léxico que vai colocar os *tokens* de tipos, símbolos, identificadores, valores, comentários, espaços em branco, etc, em cada palavra ou caractere do código-fonte. Além disso, conseguimos perceber que esse passo para desenvolver um compilador é importante pelo fato que classificar de maneira errada nessa fase pode acarretar em danos nas próximas etapas. Isso porque se passarmos valores errados para o analisador sintático, com certeza ele gerará uma árvore errada.

8. Referências

[1] LOUDEN, K.C. Compiladores – Princípios e Práticas. Ed. Thomson Pioneira, 2004.z