Trabalho 1 da disciplina de Tradutores*

Ana Paula Martins Tarchetti[17/0056082]

Departamento de Ciência da Computação, Universidade de Brasília (UnB) - Brasília, DF, 70910-900
aptarchetti@gmail.com

1 Introdução e motivação

Um compilador consiste em traduzir um programa em uma linguagem de programação fonte para uma linguagem de programação alvo. Além disso, um compilador também relata possíveis erros que possam estar presentes no código-fonte. A compreensão do processo de compilação/tradução permite a utilização das técnicas aprendidas em aplicações futuras. Exemplos desse uso são: editores de texto, sistemas de recuperação de informações, reconhecimento de padrões, construção de novas linguagem de programação, tipografia, desenho digital e verificadores [Aho03].

Este trabalho tem como objetivo fixar de forma prática o entendimento desse processo de construção de compiladores computacionais. Dessa forma, este trabalho traz um relatório da implementação feita com base nos processos envolvidos na compilação/tradução de programas. A linguagem de programação fonte que será utilizada se baseia em um subconjunto da linguagem C com algumas alterações, e novas primitivas que foram feitas para facilitar a manipulação de listas. Em adição a isso, a linguagem de programação alvo da compilação é a *Three Address Code*, que foi criada pelos autores do livro [Aho03].

1.1 Analisador léxico

A primeira etapa que foi implementada é o Analisador léxico também chamado de Analisador Linear. A função dessa etapa é identificar em um código as sequências de caracteres que formam *tokens*, sendo esse a unidade mínima que compõe um programa e é composto da seguinte forma: <nome do token, atributos do token>.

Para realizar a implementação dessa etapa foi utilizada a ferramenta *Flex*, [PEM07], cujo intuito é gerar programas que realizam correspondência de padrões em um dado texto. Então, por meio dessa ferramenta, foram declaradas expressões regulares (que foram detalhadas no Apêndice B) para indicar as seguintes definições:

- LETRA: qualquer caractere alfabético;
- DIGITO: qualquer algarismo;

* Professora: Cláudia Nalon

- ID: identificadores de funções e variáveis;
- INTEGER: reconhece números inteiros;
- FLOAT: reconhece números de ponto flutuante;
- STRING_LITERAL: cadeias de texto delimitadas por aspas duplas;
- ESPAÇO: qualquer espaço em branco;
- Outras expressões regulares para reconhecimento de erros específicos.

Após isso, foram definidas regras de identificação de *tokens*. As regras declaradas são divididas em:

- Palavras-chave: **int**, **float**, **list**, **read**, **write**, **writeln**, **main**, **return**, **if**, **else**, **for**, **NIL**;
- Novas primitivas: "**?**", "**%**", "**»**", "**«**", ":", "**!**" (ambíguo, podendo também denotar negação);
- Operadores: **e**/**ou**, >, <, >=, <=, ==, **!**=, +, **-**, *****, /, =;
- Delimitadores e outros: **vírgula**, **ponto e vírgula**, **parênteses** e **chaves**;
- Regras auxiliares: Ignorar espaços, indicar comentários e indicar erros léxicos e suas linhas e colunas correspondentes.

2 Compilação, Execução e Testes

Os passos para compilação e execução do analisador podem ser encontrados no arquivo **README.txt** do diretório principal do projeto, ou, a partir do diretório principal, siga os seguintes comandos:

```
$ flex -o src/lex.yy.c src/lexico.l
$ gcc -g -Wall src/lex.yy.c -o tradutor
$ ./tradutor < tests/<nome_do_teste>
```

Tendo isso, foram criados os seguintes **arquivos de teste** para validar o funcionamento esperado nesta etapa da implementação: (i) **correto_teste_1.c** e (ii) **correto_teste_2.c** que não apresentam erros; (iii) **errado_teste_1.c**: apresenta dois erros, um de símbolo não reconhecido na linha dois e um de comentário não finalizado na linha 5; por fim (iv) **errado_teste_2.c**: apresenta dois erros, um de sufixo inválido na linha 2 e um de demasiados pontos decimais na linha 3.

Referências

[Aho03] Alfred V Aho. *Compilers: principles, techniques and tools (for Anna University), 2/e*. Pearson Education India, 2003.

[PEM07] Vern Paxson, Will Estes, and John Millaway. Lexical analysis with flex. *University of California*, page 28, 2007.

Apêndice A Gramática

⟨*programa*⟩ ::= ⟨*lista de declarações*⟩

⟨*lista de declarações*⟩ ::= ⟨*declaração*⟩⟨*lista de declarações*⟩
| ⟨*declaração*⟩
| ⟨ε⟩

⟨*declaração*⟩ ::= ⟨*declaração de variável*⟩
| ⟨*declaração de função*⟩

⟨*declaração de variável*⟩ ::= ⟨*tipo de variável*⟩⟨*ID*⟩‘`;`’

⟨*declaração de função*⟩ ::= ⟨*tipo de variável*⟩⟨*ID*⟩‘`(`’⟨*parâmetros*⟩‘`)`’⟨*definição de função*⟩

⟨*definição de função*⟩ ::= ‘`{`’⟨*blocos de comandos*⟩‘`return`’⟨*valor*⟩‘`;`’‘`}`’

⟨*parâmetros*⟩ = ⟨*lista de parâmetros*⟩
| ⟨ε⟩

⟨*lista de parâmetros*⟩ ::= ⟨*parâmetro*⟩‘`,`’⟨*lista de parâmetros*⟩
| ⟨*parâmetro*⟩

⟨*parâmetro*⟩ ::= ⟨*tipo de variável*⟩⟨*ID*⟩

⟨*blocos de comandos*⟩ :: ‘`{`’⟨*blocos de comandos*⟩‘`}`’
| ⟨*declaração de variável*⟩⟨*blocos de comandos*⟩
| ⟨*comando*⟩⟨*blocos de comandos*⟩
| ⟨ε⟩

⟨*comando*⟩ ::= ⟨*chamada de função*⟩
| ⟨*comando de atribuição*⟩
| ⟨*comando de condicional*⟩
| ⟨*comando de iteração*⟩
| ⟨*comando de retorno*⟩

⟨*chamada de função*⟩ ::= ⟨*ID*⟩‘`(`’⟨*passagem de parâmetros*⟩‘`)`’‘`;`’
| ⟨*nome de função primitiva*⟩‘`(`’⟨*passagem de parâmetros*⟩‘`)`’‘`;`’

⟨*comando de atribuição*⟩ ::= ⟨*ID*⟩‘`=`’⟨*valor*⟩‘`;`’
| ⟨*ID*⟩‘`=`’⟨*chamada de função*⟩‘`;`’

⟨*comando de condicional*⟩ ::= ‘`if`’‘`(`’⟨*condição*⟩‘`)`’⟨*blocos de comandos*⟩
| ‘`if`’‘`(`’⟨*condição*⟩‘`)`’⟨*blocos de comandos*⟩‘`else`’⟨*blocos de comandos*⟩

⟨*comando de iteração*⟩ ::= ‘`for`’⟨*expressão de iteração*⟩⟨*blocos de comandos*⟩

⟨*expressão de iteração*⟩ ::= ‘`(`’⟨*comando de atribuição*⟩‘`;`’⟨*condição*⟩‘`;`’⟨*comando de atribuição*⟩‘`)`’

⟨*comando de retorno*⟩ ::= ‘`return`’⟨*valor*⟩‘`;`’

⟨*tipo de variável*⟩ ::= ‘`int`’
| ‘`float`’
| ‘`list`’

⟨*passagem de parâmetros*⟩ = ⟨*lista de valores*⟩
| ⟨ε⟩

⟨*lista de valores*⟩ ::= ⟨*valor*⟩‘`,`’⟨*lista de valores*⟩
| ⟨*valor*⟩

⟨*valor*⟩ ::= ⟨*expressão*⟩
| ⟨*ID*⟩
| string
| numero
| ‘’

⟨*ID*⟩ :: = **ID**

⟨*nome de função primitiva*⟩ ::= ‘`!`’
| ‘`?`’
| ‘`«`’
| ‘`»`’
| ‘`:`’
| ‘`%`’
| ‘`read`’
| ‘`write`’
| ‘`writeln`’

Apêndice B Tabela de Tokens

Token	Expressões Regulares (Regex)	Padrão
LETRA	[a-zA-Z]	letras (maiúsculas e minúsculas)
DIGITO	[0-9]	dígitos de 0 a 9
ID	[_a-zA-Z][_a-zA-Z0-9]*	identificadores
INTEGER	{DIGITO}+	número inteiro
FLOAT	({DIGITO}*\.?{DIGITO}+ | $\{DIGITO\} + \.$)	número com ponto flutuante
STRING_LITERAL	\"(\\.[^"\\])*\"	expressões entre aspas duplas
ESPACO	[\t\v\f\r]	todos os tipos de espaço em branco