UNIVERSIDADE FEDERAL DE VIÇOSA

***Campus* Florestal**

Curso superior de Ciência da Computação

CCF 441 - Compiladores

Prof. Daniel Mendes Barbosa

DOCUMENTAÇÃO TRABALHO PRÁTICO

Bernardo Veloso Resende - 1279

Gustavo Graf de Sousa - 1283

Wesley Cardoso - 1307

Florestal

2017

SUMÁRIO

INTRODUÇÃO

O presente trabalho tem como objetivo exibir os detalhes da construção de um compilador para a linguagem de programação *Orion*. A documentação está dividida em três partes, em que cada uma irá detalhar as decisões tomadas pelo grupo. A primeira parte explica os pormenores da análise léxica, a etapa seguinte diz respeito a análise sintática e a última sobre a análise semântica.

Um compilador consiste em um programa que recebe um programa fonte, em determinada linguagem de programação, e traduz o mesmo para um programa objeto, na figura 1 pode ser visualizado o processo citado acima. Ele possui diversas fases, cada uma com objetivo diferente, que podem ser vistas na figura 2.

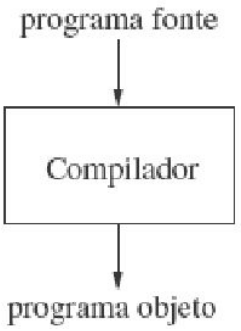

Figura 1: Compilador

A linguagem de alto nível *Orion* é considerada como um mini-pascal, e foi baseada nas linguagens *Pascal* e *Algol 60*. *Orion* possui três tipos básicos que são o *integer, boolean* e o *char,* que através deles são possíveis criar tipos compostos como os *arrays*. Outras características como estruturas de bloco, escopo e visibilidade, etc tem como alicerce as linguagens de programação citadas acima.

Figura 2: Fases de um compilador

DESENVOLVIMENTO

1. Análise Léxica

A primeira fase de um compilador é a análise léxica. Ela recebe como entrada caracteres do programa fonte, e reúne em lexemas. Como saída é produzido uma sequência de *tokens* para cada lexema que será utilizado na fase seguinte, a análise sintática. Este processo pode ser visto na figura 3.

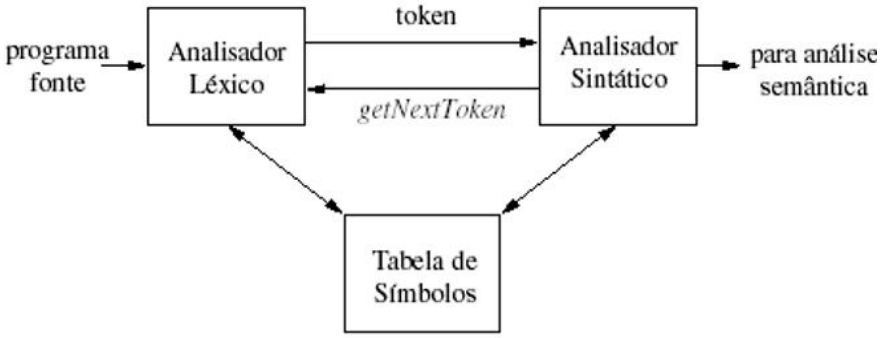

Figura 3: Interações entre o analisador léxico e o analisador sintático

Um lexema é uma sequência de caracteres que casa com um padrão de token. Um token possui um nome e um valor de atributo opcional, formando assim, um par (nome, atributo). E um padrão é uma descrição da forma de como um lexema de um token pode assumir.

Nesta etapa foi utilizado o gerador de analisador léxico LEX que permite escrever padrões para tokens utilizando linguagens regulares. A linguagem utilizada pela ferramenta é a linguagem LEX. A entrada do analisador léxico é um programa cuja a extensão é *.l* e a saída é um arquivo que possui a extensão yy.c. Um exemplo do funcionamento do LEX pode ser visto na figura 4.

Um programa LEX, escrito na linguagem LEX, possui o formato ilustrado na figura 5. Na primeira parte são declaradas as constantes manifestas , variáveis e definições regulares. Na seção seguinte são definidas as regras de tradução cujo possuem o seguinte formato: Padrão { Ação }. E por último são definidas funções

auxiliares que podem ser usadas nas ações. Normalmente estas funções são escritas na linguagem de alto nível C.

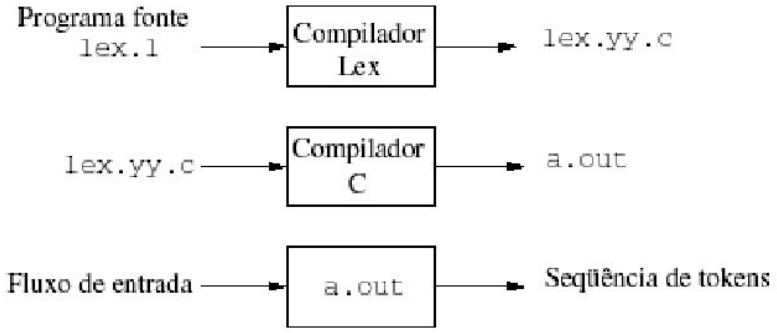

Figura 4: Funcionamento do LEX

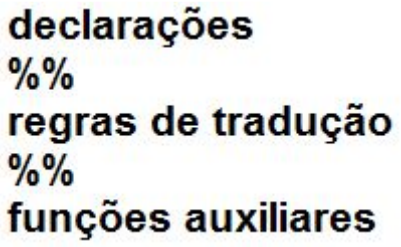

Figura 5: Estrutura de um programa LEX

Para a linguagem Orion, na seção de declarações, os tokens definidos e os lexemas que são reconhecidos por eles podem ser vistos na tabela 1. Eles foram escolhidos através da análise do documento com informações sobre a linguagem disponibilizado pelo professor. Os tokens típicos de uma linguagem normalmente são: palavra-chave, operadores, identificadores, constantes e para cada símbolo de pontuação.

Nome do Token	Definição	Descrição informal
delim	[\t \n]	Reconhece espaço em branco, tabulação, quebra de linha
ws	{delim}+	Uma ou mais instâncias de delim
digit	[0-9]	Numeros de 0 a 9
integer	[integer]	Reconhece a palavra integer
inteiro	{digit}+	Uma ou mais instâncias de digit
false	false	Reconhece a palavra false
true	true	Reconhece a palavra true
boolean	({true}|{false})	Reconhece true ou false
relacional	<|>|=|<=|>=|not=	Reconhece operadores lógicos
aritmetico	[-|+|*|/]	Reconhece operadores aritméticos
exp	[*][*]	Reconhece potência
parentese	[\(|\)]	Reconhece parente ()
comentario	[\/][*].*[*][\/]	Reconhece comentarios
ID	[a-zA-Z][a-zA-Z0-9]*	Zero ou mais instâncias de letras e numeros.

:

Tabela 1: Seção Definição.

Na seção seguinte, regras de tradução, foram definidos os padrões e suas respectivas ações. Eles podem ser vistos na tabela 2.

Ação	Padrão
{ws}	{/*nenhuma ação e nenhum retorno*/}

{inteiro}	printf("Foi encontrado um Inteiro,LEXEMA: %s\n",yytext);
integer	printf("Foi encontrado um Integer,LEXEMA: %s\n",yytext);
"|"	printf("Foi encontrado um OU,LEXEMA: %s\n",yytext);
"&"	printf("Foi encontrado um AND,LEXEMA: %s\n",yytext);
"not"	printf("Foi encontrado um NOT,LEXEMA: %s\n",yytext);
"not="	printf("Foi encontrado um Not Equal,LEXEMA: %s\n",yytext);
";"	printf("Foi encontrado um Ponto e virgula,LEXEMA: %s\n",yytext);
":"	printf("Foi encontrado Dois Pontos,LEXEMA: %s\n",yytext);
","	printf("Foi encontrado uma Vírgula,LEXEMA: %s\n",yytext);
":="	printf("Foi encontrado um Símbolo de atribuição,LEXEMA: %s\n",yytext);
begin	printf("Foi encontrado um Begin,LEXEMA: %s\n",yytext);
{boolean}	printf("Foi encontrado um Boolean,LEXEMA: %s\n",yytext);
char	printf("Foi encontrado um Char,LEXEMA: %s\n",yytext);
do	printf("Foi encontrado um Do,LEXEMA: %s\n",yytext);
doit	printf("Foi encontrado um Do it,LEXEMA: %s\n",yytext);
else	printf("Foi encontrado um Else,LEXEMA: %s\n",yytext);
end	printf("Foi encontrado um End,LEXEMA: %s\n",yytext);

endif	printf("Foi encontrado um Endif,LEXEMA: %s\n",yytext);
endwhile	printf("Foi encontrado um EndWhile,LEXEMA: %s\n",yytext);
exit	printf("Foi encontrado um Exit,LEXEMA: %s\n",yytext);
if	printf("Foi encontrado um If,LEXEMA: %s\n",yytext);
procedure	printf("Foi encontrado um Procedure,LEXEMA: %s\n",yytext);
program	printf("Foi encontrado um Program,LEXEMA: %s\n",yytext);
reference	printf("Foi encontrado um Reference,LEXEMA: %s\n",yytext);
repeat	printf("Foi encontrado um Repeat,LEXEMA: %s\n",yytext);
read	printf("Foi encontrado um Read,LEXEMA: %s\n",yytext);
return	printf("Foi encontrado um Return,LEXEMA: %s\n",yytext);
then	printf("Foi encontrado um Then,LEXEMA: %s\n",yytext);
type	printf("Foi encontrado um Type,LEXEMA: %s\n",yytext);
until	printf("Foi encontrado um Until,LEXEMA: %s\n",yytext);
value	printf("Foi encontrado um Value,LEXEMA: %s\n",yytext);
write	printf("Foi encontrado um Write,LEXEMA: %s\n",yytext);
while	printf("Foi encontrado um While,LEXEMA: %s\n",yytext);
{relacional}	printf("Foi encontrado um Operador Relacional,LEXEMA: %s\n",yytext);

{exp}	printf("Foi encontrado um Operador exponenciação,LEXEMA: %s\n",yytext);
{aritmetico}	printf("Foi encontrado um Operador Aritmetico,LEXEMA: %s\n",yytext);
{parentese}	printf("Foi encontrado um Parentese,LEXEMA: %s\n",yytext);
{comentario}	printf("Foi encontrado um Comentario,LEXEMA: %s\n",yytext);
{ID}	printf("Foi encontrado um Identificador,LEXEMA: %s\n",yytext);

Tabela 2: Regras de Tradução.

2. Análise Sintática
3. Análise Semântica

CONCLUSÃO

REFERÊNCIAS BIBLIOGRÁFICAS

AHO, Alfred V; LAM, Monica S; SETHI, Ravi; ULLMAN, Jeffrey D. **Compiladores:** Princípios, técnicas e ferramentas. Tradução Daniel Vieira. 2ª ed. São Paulo: Pearson Addison Wesley, 2008.