Universidade Federal de Viçosa
Campus Florestal
Ciência da Computação

Trabalho Prático

CCF 441 - Compiladores

Alunos: Wesley Cardoso Silva
Gustavo Graf de Sousa
Bernardo Veloso Resende
Professor: Daniel Mendes

Junho
2017

Universidade Federal de Viçosa
Instituto de Ciências Exatas e Tecnológicas

Relatório

Aluno: Bernardo Veloso Resende
Matrícula:1279
Aluno: Gustavo Graf de Sousa
Matrícula:1283
Aluno: Wesley Cardoso Silva
Matrícula:1307
Professor: Daniel Mendes

Junho
2017

Conteúdo

1 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 corresponde a detalhes da implementação 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.

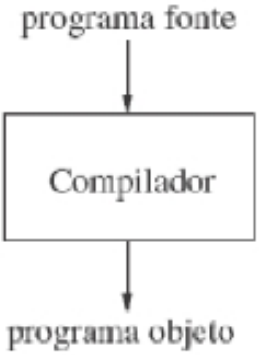

Figura 1: Compilador

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 tem como alicerce as linguagens de programação citadas acima.

Figura 2: Fases de um compilador

2 Desenvolvimento

2.1 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.

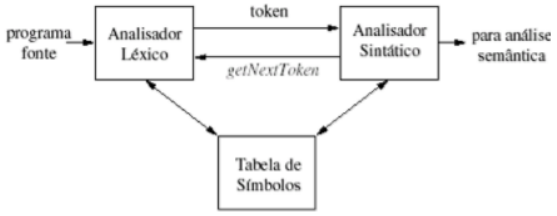

Figura 3: Interações entre o analisador léxico e o analisador sintático

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 que 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.

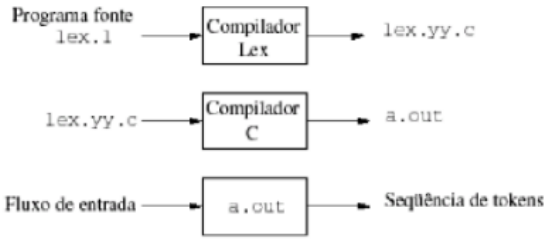

Figura 4: Funcionamento do LEX

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

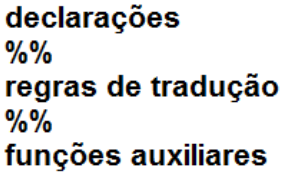

Figura 5: Estrutura de um programa LEX

linguagem disponibilizado pelo professor. Os tokens típicos de uma linguagem normalmente são: palavra-chave, operadores, identificadores, constantes e para cada símbolo de pontuação.

2.2 Análise Sintática

A fase subsquente a análise léxica é a análise sintática. Como entrada ela irá receber os tokens retornados pelo analisador léxico e vai verificar se os mesmos são válidos pela gramática da linguagem. Como saída é produzido a árvore de sintaxe que será enviada para fase seguinte, a qual é chamada análise semântica.

Um analisador sintático tem como função verifcar se um o código fornecido como entrada obedece às regras de programação da linguagem a qual o programador está codificando. Além disso é interessante que a análise léxica auxilie o usuário identificar erros de sintaxe.

Um gerador de analisador sintático é o *yacc*. Ele recebe de entrada um arquivo com a extensão .y, normalmente com o nome translate.y. Após feito a compilação é gerado um outro arquivo cujo nome é y.tab.c. O processo citado anteriormente pode ser visto na figura 6.

Figura 6: Entrada e saída do YACC

Um programa fonte em *yacc* possui o formato da figura 7. Na primeira seção pode-se fazer declarações em C como os tokens. Na subdivisão seguinte são explicitadas as regras de tradução, em que normalmente são usadas gramáticas regulares para tal. E por último são evidenciadas em funções de suporte em C.

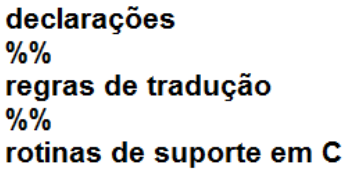

Figura 7: Estrutura de um programa em YACC

2.3 Análise Semântica

2.4 Decisões de projeto

Para a criação do código referente à análise léxica foram definidos quais tokens,lexemas, variáveis e definições regulares seriam reconhecidos de acordo com o material fornecido pelo professor. As definições regulares escolhidas podem ser observadas na tabela 1. O motivo pelo qual essas definições regulares foram implementadas foi devido à necessidade de se ter identificadores, dígito, comentário, etc.Obedecendo as regras fornecidas pela gramática.

Na seção de regras de tradução foram implementados o reconhecimento de *tokens* que são retornados para para análise sintática. Além disso, os lexemas encontrados são convertidos para *strings* e são armazenados na variável global *yylval*.

Tabela 1: Seção Definição

Nome do Token	Definição	Descrição Informal
delim	[\ t]	Reconhece espaço em branco, tabulação
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
exp	[*][*]	Reconhece potência
comentario	[\/][*].*[\][\/]	Reconhece comentário
ID	[a-zA-Z][a-zA-Z0-9]{0,31}	ID de tamanho restrito até 32 caracteres.
linha	[\ n]	Reconhece quebra de linha

Para análise sintática foi criado um arquivo chamado *translate.y*. Na primeira seção foram incluidas as bibliotecas da linguagem C, os cabeçalhos

das funções utilizadas e os *tokens* reconhecidos. Na seção seguinte foram definidas as regras da gramática e por último são declaradas funções *yyerror*, *imprimeProg* e *main*.

Os tokens foram definidos pela análise da gramática do documento disponibilizado pelo professor, em que eles estavam em negrito e ao lado direito de cada produção da gramática. Após declarados os tokens, foram definidos as precedências e a associativades dos operadores usando os comandos *%right* e *%left*.

As regras da grámatica sofreu uma única alteração na qual foi removida a regra *tipo_definido*. Ela foi apagada pois ela permitia o reconhecimento de identificadores que fugiam da regra, sendo assim, não identificando certos erros sintáticos nas declarações de variáveis.

Quando um erro sintático é encontrado, a função *yyerror* é chamada e irá exibir a linha, uma mensagem dizendo que ocorreu, um erro sintático e informa o local aproximado do erro. A função *imprimeProg* lê o arquivo de entrada e exibe o programa juntamente com linhas numeradas.

2.5 Exemplos

Para executar os códigos criados para o *LEX* e para o *YACC* foi feito um *shell script* chamado *compila*. No terminal, digite o comando *./compila* ou escreva *sh compila*. Após isso, redija o seguinte comando: *./a.out < nome_entrada.extensão*

A seguir são executados diversos programas em linguagem *Orion* e são exibidos suas respectivas saídas.

3 Metodologia

Para a conclusão deste trabalho foi necessário a ferramenta Flex e uma versão do YACC chamada BYACC funcionando no sistema operacional linux. Para instalação do Flex foi executado o comando: sudo apt-get install Flex, para instalação do Byacc foi executado o comando:apt-get install Byacc. Para executar a análise sintática é preciso ter em uma mesma pasta o arquivo lex.l e o arquivo translate.y, executar os comandos fornecidos no documento "compila"anexado à documentação, serão gerados novos arquivos, dentre eles um arquivo a.out que deverá ser executado junto com o arquivo de entrada que contém o código fonte a ser testado.

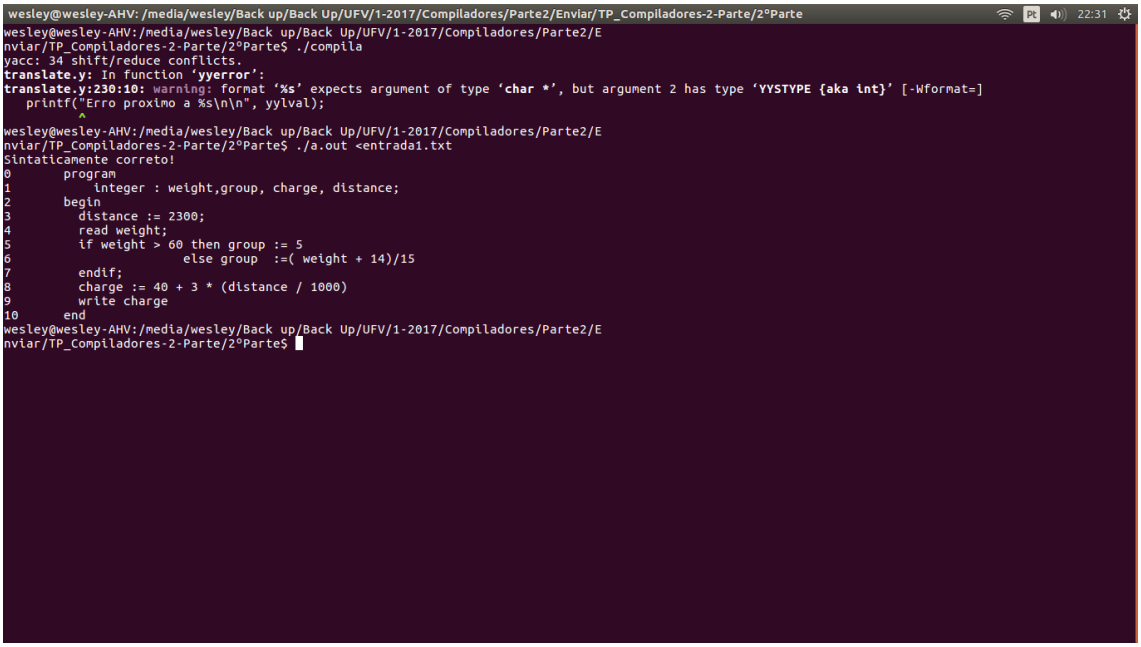

Figura 8: Programa 1 sem erro de sintaxe

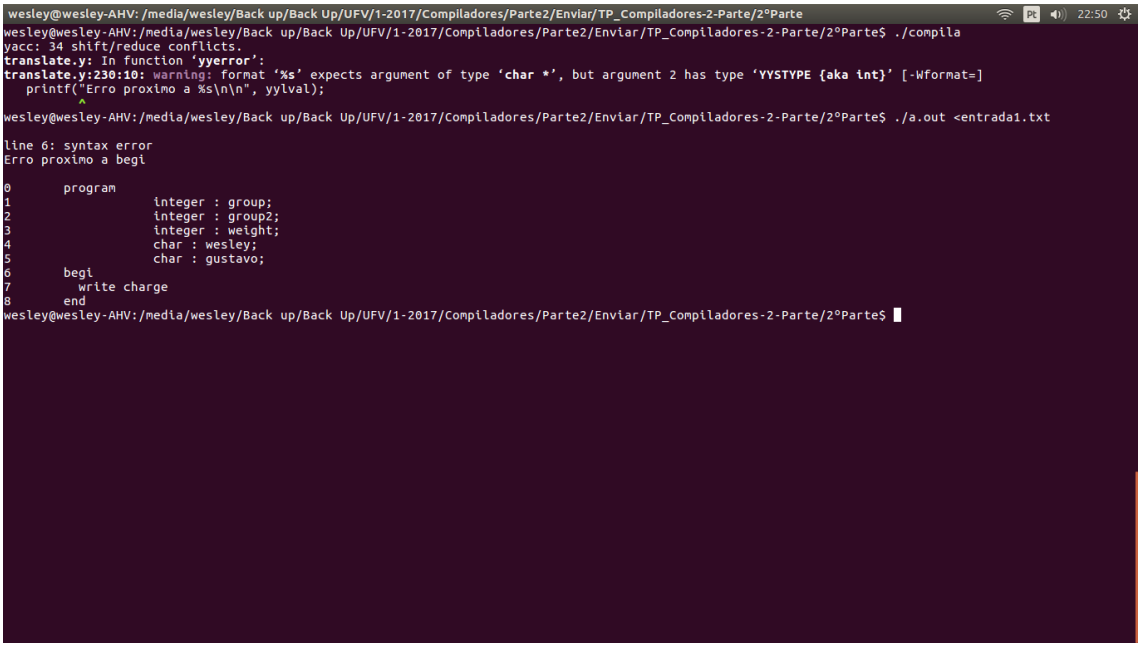

Figura 9: Programa 2 com erro de sintaxe

4 Conclusão

Bibliografia

Análise léxica.Disponível em:¡http://producao.virtual.ufpb.br/books/tautologico/intro-comp/livro/capitulos/lexic
Acesso em: 19, maio de 2017.