INE5416 - Paradigmas da Programação (2015/2)

Relatório 2: Estrutura das Linguagens
Caique Rodrigues Marques 13204303

Neste roteiro de prática, o aluno deve pesquisar sobre estrutura das linguagens de programação, principalmente como os arquivos de gramática são feitos e gerados.

Parte 1

A linguagem tem como principal função comunicar e expressar, assim, uma linguagem de programação serve para comunicar à máquina quais operações a fazer.

No entanto, máquinas reconhecem apenas o binário, a sua língua materna, por isso os compiladores traduzem as linguagens de programação de alto nível, mais amigável à pessoa, para o binário. Entre as duas camadas, linguagens de alto nível e linguagem binária, tem a linguagem de baixo-nível, o Assembly, que é pouco amigável e pode variar de máquina para máquina.

Para comunicar à máquina é necessário que a linguagem de programação usada esteja bem escrita, seguindo a sintaxe e contenha o conjunto léxico definido. Softwares são utilizados para garantir as qualidades desejadas:

- ***Flex*** (*Fast Lexical Analyzer*): Uma ferramenta para gerar scanners. Um scanner, às vezes chamado como "tokenizador", é um programa que reconhece padrões léxicos no código-fonte, ou seja, verifica se o vocabulário está de acordo com os padrões definidos pela linguagem. O programa lê um dado arquivo de entrada ou seu próprio padrão de entrada, se não houver arquivos direcionados a ele, para criar descrições para gerar um scanner. As descrições são em formatos de pares de expressões regulares e código C, chamados de regras. Flex gera uma saída em código-fonte C, lex.yy.c por padrão, que define uma função yylex(). Este arquivo pode ser compilado e "linkado" com a biblioteca flex runtime para gerar um arquivo executável. Flex foi escrito em 1987 por Vern Paxson, na linguagem C, o software surgiu como uma alternativa open-source ao lex.

- ***Lex***: Gera programas para ser usado como analisador léxico de textos. Os arquivos de entrada possuem expressões regulares para serem encontradas e ações escritas em C mostrando o que executar quando uma expressão for encontrada. Um programa em C é gerado, chamado de lex.yy. Quando executado, o programa copia partes não reconhecidas da entrada para a saída, e executa a ação em código C para cada expresão regular reconhecida. Lex foi escrito por Mike Lesk e Eric Schmidt em 1975.

- ***Yacc*** (*Yet Another Compiler Compiler*): Converte uma gramática livre-de-contexto e uma tradução de código para um conjunto de tabelas para um analisador sintático LR (left-right) e tradutor. A gramática pode ser

ambígua, as regras especificadas previamente são usadas para quebrar as ambiguidades. O arquivo gerado, y.tab.c, pode ser compilado por um compilador C para produzir um programa chamado yyparse. Este programa pode ser carregado com um analisador léxico (lex ou o flex). Yacc foi desenvolvido no começo dos anos 1970 por Stephen C. Johnson da AT&T Corp. e escrito em B, depois reescrito em C.

- ***GNU Bison***: É um gerador de analisador sintático (parser) no estilo do Yacc. Ele é compatível com arquivos designados ao Yacc, portanto, os arquivos possuem a extensão .y. No entanto, o arquivo gerado não possui nomes fixos (como yyparse no Yacc), usa prefixos do arquivo de entrada. Bison foi escrito por Robert Corbett em 1988 e ganhou a compatibilidade com o Yacc graças a Richard Stallman.

Parte 2

y.tab.h

O arquivo é gerado pelo analisador sintático Yacc/Bison, ele contém as definições de cada tipo correspondendo a um valor decimal (o primeiro valor é 257, número após o último elemento da tabela ASCII).

scan.l

Os arquivos de extensão .l são arquivos Lex/Flex, como difinido antes, eles são usados para definir a estrutura léxica do programa, neste caso, da própria linguagem C. Em scan.l, como qualquer arquivo lex, é divido em três seções, separadas por "%%", onde, de cima para baixo, começa na seção de definição, depois, seção de regras e, por fim, seção de código em C.

```
D [0-9]
L [a-zA-Z_]
H [a-fA-F0-9]
E [Ee][+-]?{D}+
FS (f|F|l|L)
IS (u|U|l|L)*

%{
#include <stdio.h>
#include "y.tab.h"

void count();
%}
```

Na seção de definição (acima), são definidos o conjunto de possíveis símbolos para uma expressão regular, ou seja, o alfabeto. Nas últimas cinco linhas há uma seção de código em C, onde há duas inclusões, o padrão standard do C e o

arquivo y.tab.h. Por fim, a função count é chamada, esta foi definida na última seção do arquivo, que será explicada adiante.

Na seção de regras, apenas relacionado os caracteres (o alfabeto) com os tipos definidos no arquivo y.tab.c. Antes de retornar com o inteiro correspondente ao caractere, a função count() é chamada.

Na seção de código C é onde mostra como o programa deve se comportar. Há quatro funções definidas: yywrap(), comment(), count() e check_type().

```
yywrap() {
    return(1);
}

comment() {
    char c, c1;

loop:
    while ((c = input()) != '*' && c != 0)
            putchar(c);

        if ((c1 = input()) != '/' && c != 0) {
                unput(c1);
                goto loop;
        }

        if (c != 0)
                putchar(c1);
    }

int column = 0;

void count() {
        int i;

        for (i = 0; yytext[i] != '\0'; i++)
                if (yytext[i] == '\n')
                        column = 0;
                else if (yytext[i] == '\t')
                        column += 8 - (column % 8);
                else
                        column++;

        ECHO;
}
```

- A função yywrap() é a versão default onde apenas retorna o inteiro 1. Ela é usada após o scanner do lex chegar ao fim do arquivo;

- A função comment(), é definido o sistema de comentários que são marcados como os símbolos que estiverem entre "/*" e "*/", representando o início e o fim do comentário, respectivamente;
- A função count() é o analisador sintático, ele verifica a palavra recebida, num loop, até o final "\0". A varíavel "column" representa onde o cursor está posicionado e, caso o texto lido seja um "\n", o cursor volta ao início, ou seja, foi acrescentada um nova linha. Se o texto lido foi um "\t", significando "tab" é adicionado um espaçamento a mais na posição do cursor;
- A função check_type() apenas retorna o tipo (ou o identificador) do caractere digitado (seja int, enum, char, etc.).

gram.y

O arquivo Bison/Yacc gerado é o analisador sintático dos arquivos gerados pelo lex, na primeira seção é estabelecido os tokens que a linguagem deve conter, ou seja, os tipos presentes. Na segunda parte, após o "%%", é estabelecido a sintaxe das expressões usadas na linguagem, ou seja, a "forma" da expressão. Exemplos:

```
and_expr
: equality_expr
| and_expr '&' equality_expr
;
```

Acima mostra como deve ser uma expressão and ("e"), com duas proposições a serem comparadas entre o sinal de "&".

```
init_declarator
 : declarator
 | declarator '=' initializer
 ;
```

É definido a declaração de uma varíavel: definindo o nome da variável e após o sinal "=" é o inicializador.

Fontes

- http://flex.sourceforge.net/;
- manual page do Bison (http://dinosaur.compilertools.net/bison/manpage.html);
- manual page do Lex (http://plan9.bell-labs.com/magic/man2html/1/lex);
- manual page do Yacc (http://plan9.bell-labs.com/magic/man2html/1/yacc).