

Compiladores: Analizador Léxico e Sintático

Luiz Otávio Resende Vasconcelos

Setembro 2017

1 Introdução

Compiladores é um ramo importante da Computação que tem como objetivo estudar e encontrar as melhores técnicas e maneiras de transformar um código alto nível de uma linguagem específica e gerar um novo código (chamado de código-objeto) semanticamente equivalente em outra linguagem (comumente código de máquina).

Neste trabalho, foi proposto desenvolver uma parte de um compilador (análise léxica e sintática) que reconhecesse uma linguagem com a seguinte gramática:

```
program  → block
block    → { decls stmts }
decls    → decls decl |  $\epsilon$ 
decl     → type id ;
type     → int | char | bool | float
stmts    → stmts stmt |  $\epsilon$ 
stmt     → id = expr ;
           | if ( rel ) stmt
           | if ( rel ) stmt else stmt
           | while ( rel ) stmt
           | block
rel      → expr < expr | expr <= expr | expr >= expr |
           | expr > expr | expr
expr     → expr + term | expr - term | term
term     → term * unary | term / unary | unary
unary    → - unary | factor
factor   → num | real
```

Figure 1: Gramática

2 Implementação

A implementação foi feita utilizando as ferramentas de código aberto Flex, para análise léxica, e Bison, para a sintática.

A cada iteração, o Bison solicita ao Flex um novo token que é capturado do código fonte através de uma expressão regular. O Bison então, em posse da gramática da linguagem, verifica se aquela expressão está sintaticamente correta.

2.1 Flex

Flex (The Fast Lexical Analyzer) é uma ferramenta de análise Léxica disponível no repositório <https://github.com/westes/flex>.

Para utilizá-lo, foi criado um arquivo com a extensão .l (flex.l).

O arquivo deve ser dividido em três secções: **cabeçalho**, **regras** e **funções auxiliares**.

2.1.1 Cabeçalho

No cabeçalho fazemos algumas declarações e incluímos o bison.h

2.1.2 Regras

Nas regras, é onde inserimos nossas expressões regulares para capturarmos os tokens do programa e retornamos ao Bison. As expressões regulares utilizadas foram:

```
\{|\\}|\\(|\\)|\\;
```

Para capturar os tokens. São retornados em sua forma literal.

```
\\*|\\-|\\+|\\=|\\|\\<|\\>
```

Para capturar os operadores. São retornados em sua forma literal.

```
if
```

Para capturar a palavra reservada 'if'. É retornado um identificador IF.

```
else
```

Para capturar a palavra reservada 'else'. É retornado um identificador ELSE.

```
while
```

Para capturar a palavra reservada 'while'. É retornado um identificador WHILE.

`int|char|bool|float`

Para capturar as palavra reservada que denotam o tipo das variáveis. É retornado um identificador TYPE.

`[0-9]+\.[0-9]+`

Para capturar lexemas do tipo float. É retornado um identificador FLOAT.

`[0-9]+`

Para capturar lexemas do tipo int. É retornado um identificador INT.

`true|false`

Para capturar lexemas do tipo boolean. É retornado um identificador BOOL.

`[a-zA-Z0-9]+`

Para capturar lexemas do tipo string. É retornado um identificador STRING.

.

Para todo o resto, não faça nada.

2.1.3 Funções Auxiliares

É onde definimos funções auxiliares em C para nosso programa.

Para esse trabalho, na etapa do Lex não será necessário definir nenhuma função auxiliar.

2.2 Bison

GNU Bison é uma ferramenta de análise sintática disponível no endereço <https://www.gnu.org/software/bison/> mantido pela Free Software Foundation.

Em suma, Bison é um gerador de parser que converte uma gramática livre de contexto em uma LR determinística ou LR generalizada utilizando tabelas de parser LALR.

Para utilizá-lo, foi criado um arquivo com a extensão .y (bison.y).

Assim como o Flex, o arquivo Bison deve ser dividido em três secções: **cabeçalho**, **regras** e **funções auxiliares**.

2.2.1 Cabeçalho

No cabeçalho fazemos apenas algumas declarações pertinentes à linguagem C++

2.2.2 Regras

Nas regras, é onde inserimos nossa gramática que será utilizada para analisar os tokens que vem do Lex.

Caso nenhuma regra seja encontrada para um determinado token, há uma recusa e um erro é então gerado: "Parser error!".

Gramática utilizada:

```
program:
    %empty
    | program block
    ;

block:
    '{' decls stmts '}'
    ;

decls:
    decls decl
    | %empty
    ;

decl:
    TYPE STRING ';'
    ;

stmts:
    stmts stmt
    | %empty
    ;

stmt:
    STRING '=' expr ';'
    | IF '(' rel ')' stmt
    | IF '(' rel ')' stmt ELSE stmt
    | WHILE '(' rel ')' stmt
    | block
    ;

rel:
    expr '<' expr
    | expr '<' '=' expr
    | expr '>' '=' expr
    | expr '>' expr
    ;
```

```

expr:
    expr '+' term
    | expr '-' term
    | term
    ;

term:
    term '*' unary
    | term '/' unary
    | unary
    ;

unary:
    '-' unary
    | factor
    ;

factor:
    INT
    | FLOAT
    ;

```

2.2.3 Funções Auxiliares

Para o Bison, utilizaremos duas funções:

Abrir o código fonte na main:

```

int main(int, char**) {
    FILE *myfile = fopen("./source", "r");
    if (!myfile) {
        cout << "I can't open ./tests/test1!" << endl;
        return -1;
    }
    yyin = myfile;
    do {
        yyparse();
    } while (!feof(yyin));
}

```

O Bison recebe o input na variável reservada *yyin*.

E a função de *handle errors*:

```

void yyerror(const char *s) {
    cout << "Parse error! Message: " << s << endl;
    exit(-1);
}

```

```
}
```

Que acusará um erro de parser quando um token for rejeitado.

3 Utilização

Para testar um código fonte, basta inseri-lo na pasta do projeto e renomeá-lo para *source* e então executar:

```
bison -d bison.y &&  
flex flex.l &&  
g++ bison.tab.c lex.yy.c -lfl -o compiler &&  
./compiler
```

Caso seu código não seja aceito, será apresentado um erro no console.

4 Conclusão

Análise Léxica e Sintática são etapas fundamentais de todo compilador. São etapas que requerem atenção, pois, como somente a análise léxica tem acesso ao código fonte, um mínimo erro compromete todo o compilador.

Para tais etapas, as ferramentas descritas apresentaram grande facilidade no desenvolvimento e boa performance. Os diversos códigos testados apresentaram o resultado esperado.

Referências

- [1] Free Software Foundation. Gnu bison, 2014.
- [2] Westes. Flex, 2017.
- [3] Wikipedia. Compiladores, 2017.