Implementação de Analisador Sintático com o Bison

Gabryella Mika Tanigawa[1] e Júlia Mendes[1]

[1]Ciência da Computação - Instituto de Biociências, Letras e Ciências Exatas

Outubro de 2024

Relatório apresentado à disciplina de Compiladores do curso de Ciência da Computação da Universidade Estadual Paulista.

INTRODUÇÃO

A análise sintática do compilador é crucial para mapear um programa fonte para uma programa objeto. Ela sucede a análise léxica e utiliza os tokens gerados para criar uma representação intermediária chamada árvore de derivação, que identifica a estrutura gramatical da linguagem. À vista disso, o objetivo do presente trabalho é desenvolver um analisador sintático com o uso da ferramenta Bison capaz de identificar os comandos na linguagem de programação C.

ALFABETO

O alfabeto da linguagem analisada pelo analisador léxico e sintático é composto pelos seguintes elementos:

1. Letras Maiúsculas e Minúsculas:
   - A, B, C, ..., Z.
   - a, b, c, ..., z.
2. Dígitos:
   - 0, 1, 2, 3, 4, 5, 6, 7, 8, 9.
3. Delimitadores:
   - Parênteses: (,)
   - Chaves: **{** , **}**
   - Pontos e vírgulas: **;**
   - Vírgulas: **,**
4. Símbolos:
   - + , - , * , / , % , | , & , = , . , ! , < , >
5. Espaços em branco e quebras de linha:
   - ' ' , $\backslash n$

REGRAS DE PRODUÇÃO

A fim de diferenciá-los dos não terminais, os terminais da gramática estão escritos em **negrito**.

- `funcao principal :  int main parenteses bloco`

A regra produz a sintaxe da função principal, definida pelo tipo int seguido do identificador main, depois pela aplicação da regra parenteses e, por fim, da regra bloco.

Exemplos
Aceitos:

- `int main(){...}`

Erro Sintático:

- `int maain(){...}`
- `float main(){..}`
- `main(){...}`

Parenteses da Função Principal

- `parenteses:  ( )`

Regra de produção definida pelo separador (seguido do).

Exemplos
Aceitos:

- `( )`

Erro Sintático:

- `(()`
- `(`

Bloco

- `bloco:  { cs }`

A regra produz a sintaxe dos blocos de comando, definidos pelo separador {, seguido da regra cs e do outro separador }.

Exemplos
Aceitos:

- `{int nota1 = 7.5;}`
- `{int i = 0; while(i<10){i++;}}`

Erro Sintático:

- `{{float nota1 = 7.5;}`
- `{int i, j, k;`

Lista de Comandos

- `cs:  c | c cs`

A regra produz uma sintaxe definida por uma regra c ou por c seguido de, novamente, cs, definindo uma recurssão.

Exemplos
Aceitos:

- `double media = 0.0;`
- `int i; for(i=0;i<10;i++){...}`

Comandos

- `c:  decl; | atri; | una; | if (cond) bloco else | while (cond) bloco | printf (write); | scanf (read); | for (atri;cond;una) bloco`

A regra produz sintaxe para a regra dcl, atri, una, para o condicional if, while, for, printf e scanf.

Exemplos
Aceitos:

- `if(media<5){printf(Reprovado);}`
- `scanf(%f, &media);`

Erro Sintático:

- `print(Aprovado);`
- `int i = 0`

Else

- `Else:` **`else`** `bloco | ε`

A regra de produção define o identificador else seguido da regra bloco ou cadeia vazia.

Exemplos
Aceitos:

- `else{i--;}`
- `else{printf(Reprovado);}`

Erro Sintático:

- `else{i++;`
- `ele{soma = soma + 1;}`

Declaração

- `decl: tipo lista_var`

A regra produz a sintaxe da declaração de variáveis, definida pela regra tipo seguida de lista_var.

Exemplos
Aceitos:

- `double soma = 0.0`

Erro Sintático:

- `flot = 10`

Tipos

- `tipo:` **`int`** `|` **`float`** `|` **`double`**

A regra de produção define os identificadores int, float ou double para declaração de variáveis

Exemplos
Aceitos:

- `int`
- `float`
- `double`

Erro Sintático:

- `long int`
- `doube`

Lista de Variáveis

- `lista_var:` **`id,`** `lista_var |` **`id`** `| atri, lista_var | atri`

A regra de produção é definida por um identificador ou por um identificador, seguido pelo separador , e, novamente, pela regra lista_var, definindo uma recurssão, ou, também, pela regra atri ou pela regra atri, seguido do separador , e da regra lista_var, definindo, novamente, uma recurssão.

Exemplos
Aceitos:

- `nota, soma = 0`
- `nota1, nota2, nota3`
- `iniciou = 1, terminou = 0`

Erro Sintático:

- `soma = 0 media`

Atribuição

- `atri:  id = exp | id = una`

A regra produz a sintaxe da atribuição, definida por um identificador, seguido de = e da regra exp, ou definida por um identificador e =, seguidos pela regra una.

Exemplos

Aceitos:

- `nota1 = 0`
- `soma = i++`

Erro Sintático:

- `media 5`

Expressão

- `exp:  exp + termo | exp − termo | termo`

A regra da expressão é definida, recursivamente, por exp, seguido do operador aritmético + e por termo, ou por exp, seguido do − e termo, ou, somente, de termo.

Exemplos

Aceitos:

- `soma + 1`
- `12 + 32`

Erro Sintático:

- `nota1 ++ nota2`

Termo

- `termo:  termo * fator | termo / fator | termo % fator | fator`

A regra é definida, recursivamente, por termo, novamente, seguido do operador aritmético * e por fator, ou termo, seguido do / e fator, ou, ainda, por fator.

Exemplos

Aceitos:

- `5 * 12`

Erro Sintático:

- `nota1 /* 2`

Fator

- `fator:  (exp) | num | real | id`

A regra de produção pode ser definida pelo separador (, seguido da regra exp e do), ou por um número, ou por um real, ou por um identificador.

Exemplos

Aceitos:

- `(nota1 + nota2)`
- `(base + altura) / 2`

Erro Sintático:

- `(num *) + 32`

Condição

- `cond: rel && rel | rel || rel | rel`

A regra produz a sintaxe de uma condição, definida pela regra rel, seguida do operador lógico && e, novamente, rel, ou pela regra rel, seguida do operador lógico —— e rel, ou, somente, pela rel.

Exemplos

Aceitos:

- `10 > (soma) && media == 5`

Erro Sintático:

- `10 == soma |& media != 5`

Relacionais

- `rel: exp > exp | exp < exp | exp == exp | exp ≥ exp | exp ≤ exp | exp != exp`

A regra produz a sintaxe das relações entre as expressões, definida pelas regras exp, rel pode ser definida pelos operadores relacionais ¿, ¡, ==, ≥, ≤ ou !=.

Exemplos

Aceitos:

- `(i + 1) > (media / 2)`

Erro Sintático:

- `12 >== nota1`
- `media = 5`

Unário

- `una: op id | id op`

A regra é definida pela regra op seguido de um identificador, ou de um identificador seguido do op.

Exemplos

Aceitos:

- `++i`
- `decr−−`

Erro Sintático:

- `i+++`

Operador

- `op: ++ | --`

A regra é definida pelooperador unário ++, ou pelo operador unário −−;

Exemplos

Aceitos:

- `++`

Erro Léxico:

- `−+`
- `− − −`

Escrever

- `write:  id | id write`

A regra produz a sintaxe do que pode ser escrito em um printf, é definida, recursivamente, por um identificador e seguido de write ou, somente, por um identificador.

Exemplos
Aceitos:

- `Aprovado`
- `Digite a nota da prova`

Erro Sintático:

- `A sequencia de simbolos % & * #`

Ler

- `read:  %id, &id`

A regra produz a sintaxe de um scanf, definido por um operador %, um identificadot, um separador ,, um & e um identificador.

Exemplos
Aceitos:

- `%f, &media`

Erro Sintático:

- `%d, nota`

MUDANÇAS NA GRAMÁTICA

Abaixo estão as mudanças que fizemos na gramática regular que definiu a linguagem C analisada pelo analisador léxico:

- Criamos expressões regulares para reconhecer as palavras reservadas int, float, double, main, if, else, for, while, printf e scanf. Anteriormente, esses tokens eram reconhecidos, apenas, como identificadores. Entretanto, o analisador sintático exigiu um detalhamento maior.
- Optamos por revomer as aspas(**"**) da linguagem.
- Removemos o caractere **!** como operador lógico, dado que não é possível utilizá-lo no mesmo formato do E lógico e do OU lógico, e, também, porque seu uso estava causando muitas ambiguidades.

COMO COMPILAR

Para compilar e executar o analisador sintático descrito no arquivo '**analisador.y**', a ferramenta BISON e o GCC devem estar instalados no sistema. No terminal Ubuntu:

1. Execute o Bison a partir do arquivo de definição do analisador sintático para gerar o arquivo de cabeçalho, *analisador.tab.h* e o arquivo que contém o código em C, *analisador.tab.c*.

   ```
   bison -d analisador.y
   ```

2. Execute o arquivo de definição do analisador léxico pelo Flex para gerar o arquivo *lex.yy.c*.

   ```
   flex analisadorlexico.l
   ```

3. Compile os arquivos *.c* individualmente. Isso criará os arquivos objetos, *lex.flex.o* e *analisador.y.o*, respectivamente.

   ```
   gcc -c lex.yy.c -o lex.flex.o
   gcc -c analisador.tab.c -o analisador.o
   ```

4. Combine os arquivos objetos em um único executável *analisador*.

```
gcc -o analisador lex.flex.o analisador.y.o -lfl -lm
```

5. Execute o analisador sintático junto ao arquivo de entrada, *codigo_entrada.txt* que será analisado por ambos os analisadores, léxico e sintático.

```
./analisador codigo_entrada.txt
```