Relatório do Analisador Sintático

Compiladores

Ana Beatriz Gomes Takehara
Heloísa Silveira Bula

Docente: Renata Spolon

São José do Rio Preto 2024

1 Alfabeto

O alfabeto é um conjunto finito e não vazio de símbolos que serão empregados na formação de cadeias. Neste projeto, os símbolos que estão incluídos nesse grupo são:

- Letras maiúsculas e minúsculas com ou sem acento:
  - {a | b | c | ... | y | z }
  - {A | B | C | ... | Y | Z }
  - {á | é | í | ó | ú | ã | õ | â | ê | î | ô | û | à | è | ì | ò | ù | ç }
  - {Á | É | Í | Ó | Ú | Ã | Õ | Â | Ê | Î | Ô | Û | À | È | Ì | Ò | Ù | Ç }
- Números:
  - {0 | 1 | 2 | ... | 8 | 9}
- Operadores Aritméticos:
  - {+ | - | * | / | % | = }
- Operadores Lógicos:
  - { > | < | '|' | & | ! }
- Separadores:
  - { (|) | { | } }
- Pontuações:
  - { . | " | # | , | ; | : | \ }

O alfabeto sofreu apenas 2 alterações em relação ao projeto do Analisador Léxico, não são mais aceitos os colchetes e nem as aspas simples. Essa mudança foi optada pois nenhuma das expressões que geram nossa linguagem utilizam esses símbolos.

2 Expressões Regulares

As expressões regulares formam as regras descritas e aceitas pelo anaflex. Para permitir um funcionamento adequado do Analisador Sintático, precisamos adicionar algumas novas expressões regulares.

Incluímos uma regra para aceitar e retornar um tipo de dado, definindo para nossa linguagem apenas os tipos int, float e string. Além disso, foram adicionados comandos específicos na forma de palavras reservadas, incluindo if, else e elseif para regras condicionais, assim como for, while e do para regras de repetição. Também introduzimos a palavra reservada main para definir o programa principal, além dos comandos printf e scanf, que permitem a exibição de saída e a entrada de dados, respectivamente.

Para o funcionamento correto do 'printf' e do 'scanf', adicionamos uma expressão, a qual chamamos 'CODIGO_AUX', para possibilitar a utilização de: "%d", "%s" e "%f".

```
{PORCENTAGEM}{CODIGO}                { return CODIGO_AUX;}
```

Além disso, colocamos mais 6 expressões que permitem a realização do incremento e decremento das variáveis.

```
{CARACTERE}+{MAIS}{MAIS}          {return INCREMENTO; }
{CARACTERE}+{NUMERO}+{MAIS}{MAIS}    {return INCREMENTO; }
{CARACTERE}+{NUMERO}+{CARACTERE}+{MAIS}{MAIS}  {return INCREMENTO; }

{CARACTERE}+{MENOS}{MENOS}          {return INCREMENTO; }
{CARACTERE}+{NUMERO}+{MENOS}{MENOS}    {return INCREMENTO; }
{CARACTERE}+{NUMERO}+{CARACTERE}+{MENOS}{MENOS}  {return INCREMENTO; }
```

Por fim, incluimos as expressões regulares para utilização dos operadores lógicos && e ||.

3 Gramática

3.1 Regras auxiliares

3.1.1 Tipos

É a regra que verifica se estão sendo usados somente os tipos de variáveis definidos previamente:

int
float
string

```
//TIPO QUE UMA VARIÁVEL PODE TER

Tipo: TIPO_INT | TIPO_FLOAT | STRING ;
```

3.1.2 Condicao_aux e Condicao

As regras 'Condicao' e 'Condicao_aux' são utilizada nas estruturas condicionais e de repetição para impor condições específicas, permitindo a construção de expressões lógicas e aritméticas:

```
//CONDIÇÕES QUE VÃO PODER SER UTILZADAS NOS COMANDO CONDICIONAIS E NOS LAÇOS DE REPETIÇÃO

Condicao: Condicao_aux OP_LOG Condicao_aux ;

Condicao_aux: Condicao_aux OP_LOG Condicao_aux
            | INT
            | IDENTIFICADOR
            | FLOAT
            | FLOAT OP_AR FLOAT
            | FLOAT OP_AR INT
            | FLOAT OP_AR IDENTIFICADOR
            | INT OP_AR INT
            | INT OP_AR FLOAT
            | INT OP_AR IDENTIFICADOR
            | IDENTIFICADOR OP_AR IDENTIFICADOR
            | IDENTIFICADOR OP_AR INT
            | IDENTIFICADOR OP_AR FLOAT ;
```

3.1.3 ExpressãoAritmética

A Expressão Aritmética aceita um conjunto de expressões realizando as 4 operações aritméticas básicas: adição(+), subtração(-), divisão(/), multiplicação(*); junto com a operação que salva o resto da divisão(%), sendo ela exata ou não. O símbolo igual(=) foi usado para simbolizar atribuição ou resultado. Configurando-se os seguintes casos aceitos:

ExpressãoAritmetica OP_AR ExpressaoAritmetica
variável OP_AR variável
Num_Inteiro OP_AR Num_Inteiro
Num_Inteiro OP_AR Num_Float
Num_Inteiro OP_AR variavel
Num_Float OP_AR Num_Float
Num_Float OP_AR variavel
Num_Float OP_AR Num_Inteiro
variável OP_AR Num_Int
variável OP_AR Num_Float
variável = variável

```
//EXPRESSÃO ARITMÉTICA

ExpressaoAritmetica: ExpressaoAritmetica OP_AR ExpressaoAritmetica
                   | IDENTIFICADOR
                   | INT
                   | FLOAT
                   | IDENTIFICADOR OP_AR INT
                   | IDENTIFICADOR OP_AR FLOAT
                   | IDENTIFICADOR OP_AR IDENTIFICADOR
                   | IDENTIFICADOR IGUAL IDENTIFICADOR ;
```

3.1.4 Identificador_aux

Essa expressão é usada como um auxiliar para o comando 'printf' e atribuição de 'string', com ela pode-se escrever qualquer frase ou palavra independente de seguir as restrições de um identificador normal.

```
//FUNÇÃO AUXILIAR UTILIZADA PARA ESCREVER QUALQUER COISA NO PRINT E NA ATRIBUIÇÃO DE UMA STRING

Identificador_aux: IDENTIFICADOR | IDENTIFICADOR Identificador_aux
                 | PRINTF | PRINTF Identificador_aux
                 | IF | IF Identificador_aux
                 | ELSE | ELSE Identificador_aux
                 | ELSEIF | ELSEIF Identificador_aux
                 | STRING  | STRING Identificador_aux
                 | WHILE | WHILE Identificador_aux
                 | FOR | FOR Identificador_aux
                 | PRINTF | PRINTF Identificador_aux
                 | SCANF | SCANF Identificador_aux
                 | MAIN | MAIN Identificador_aux
                 | INT | INT Identificador_aux
                 | FLOAT | FLOAT Identificador_aux
                 | OP_AR | OP_AR Identificador_aux
                 | OP_LOG | OP_LOG Identificador_aux
                 | OR | OR Identificador_aux
                 | AND | AND Identificador_aux
                 | NOT | NOT Identificador_aux
                 | IGUAL | IGUAL Identificador_aux
                 | VIRGULA | VIRGULA Identificador_aux
                 | BARRA | BARRA Identificador_aux
                 | PVIRGULA | PVIRGULA Identificador_aux
                 | PONTO | PONTO Identificador_aux
                 | DOIS_PONTOS | DOIS_PONTOS Identificador_aux
                 | HASHTAG | HASHTAG Identificador_aux
                 | ABRE_PARENTESIS | ABRE_PARENTESIS Identificador_aux
                 | FECHA_PARENTESIS | FECHA_PARENTESIS Identificador_aux ;
```

3.1.5 Atribuição Aux

Comando que vai ser chamado dentro dos comandos de repetição. O formato aceito é:

variável = número inteiro;
variável = variável;

```
Atribuicao_aux: IDENTIFICADOR IGUAL INT PVIRGULA
              | IDENTIFICADOR IGUAL IDENTIFICADOR PVIRGULA;
```

3.2 Principais

São as expressões que realizam os comandos principais da linguagem.

3.2.1 Programa_principal

Essa regra é responsável por verificar a estrutura principal do programa. O formato aceito é:

main() { Comandos }

```
//ESTRUTURA DO PROGRAMA PRINCIPAL

Programa_principal: MAIN ABRE_PARENTESIS FECHA_PARENTESIS ABRE_CHAVE Comandos FECHA_CHAVE ;
```

3.2.2 Comandos

É o grupo de expressões usadas para construção de programas, sendo elas: condicionais, loops, atribuição, printf, scanf, declaração.

```
//COMANDOS QUE VÃO PODER SER UTILIZADOS NO PROGRAMA

Comandos: Declaracao
        | Atribuicao
        | Repeticao
        | Condicional
        | PrintfComando
        | ScanfComando
        | Comandos Comandos
        | error PVIRGULA {yyerrok;};
```

3.2.3 Declaração

É a regra que permite que uma variável seja declarada. O formato aceito é:

tipo variavel = ExpressaoAritmetica;
tipo variavel = ExpressaoAritmetica, Declaracao_aux;
tipo variavel, Declaracao_aux;

tipo variavel;
tipo variavel = ”Identificador_aux”;

```
//DECLARAÇÃO DE VARIÁVEL

Declaracao: Tipo Declaracao_aux PVIRGULA ;

Declaracao_aux: IDENTIFICADOR IGUAL ExpressaoAritmetica
              | IDENTIFICADOR IGUAL ExpressaoAritmetica VIRGULA Declaracao_aux
              | IDENTIFICADOR VIRGULA Declaracao_aux
              | IDENTIFICADOR
              | IDENTIFICADOR IGUAL ASPAS_DUPLAS Identificador_aux ASPAS_DUPLAS ;
```

3.2.4 Atribuição

É a regra que permite a atribuição de um valor à uma variável. O formato aceito é:

variavel = ExpressaoAritmetica;
variavel = ”Identificador_aux”;
variavel++;
variavel–;

```
//ATRIBUIÇÃO DE VARIÁVEL

Atribuicao: IDENTIFICADOR IGUAL ExpressaoAritmetica PVIRGULA
          | IDENTIFICADOR IGUAL ASPAS_DUPLAS Identificador_aux ASPAS_DUPLAS PVIRGULA
          | INCREMENTO PVIRGULA ;
```

3.2.5 Condicional

É a regra que permite que os programas utilizem as condições para executar blocos. O formato aceito é:

if (Condicao) { Comandos}
elseif (Condicao) { Comandos }
else { Comandos }

```
//CONDICIONAL

Condicional: IF ABRE_PARENTESIS Condicao FECHA_PARENTESIS ABRE_CHAVE Comandos FECHA_CHAVE
           | ELSEIF ABRE_PARENTESIS Condicao FECHA_PARENTESIS ABRE_CHAVE Comandos FECHA_CHAVE
           | ELSE ABRE_CHAVE Comandos FECHA_CHAVE ;
```

3.2.6 Repetição

É o comando que permite que o programa utilize as estruturas de repetição. O formato aceito é:

while (Condicao) { Comandos }
for (variavel = Atribuicao_aux; Condicao; incremento) { Comandos }
do { Comandos } while (Condicao);

```
//REPETIÇÃO

Repeticao: WHILE ABRE_PARENTESIS Condicao FECHA_PARENTESIS ABRE_CHAVE Comandos FECHA_CHAVE
         | FOR ABRE_PARENTESIS Atribuicao_aux Condicao PVIRGULA INCREMENTO FECHA_PARENTESIS ABRE_CHAVE Comandos FECHA_CHAVE
         | DO ABRE_CHAVE Comandos FECHA_CHAVE WHILE ABRE_PARENTESIS Condicao FECHA_PARENTESIS PVIRGULA ;
```

3.2.7 Printf

Comando Printf é usado para impressão de dados na tela. O formato aceito é:

printf("Identificador_aux");
printf("Identificador_aux: %d", variavel);
printf("%d", variavel);
printf("%d: ", variavel);

```
//PRINT

PrintfComando: PRINTF ABRE_PARENTESIS ASPAS_DUPLAS Identificador_aux CODIGO_AUX ASPAS_DUPLAS VIRGULA ExpressaoAritmetica FECHA_PARENTESIS PVIRGULA
             | PRINTF ABRE_PARENTESIS ASPAS_DUPLAS Identificador_aux ASPAS_DUPLAS FECHA_PARENTESIS PVIRGULA
             | PRINTF ABRE_PARENTESIS ASPAS_DUPLAS CODIGO_AUX ASPAS_DUPLAS VIRGULA ExpressaoAritmetica FECHA_PARENTESIS PVIRGULA
             | PRINTF ABRE_PARENTESIS ASPAS_DUPLAS Identificador_aux CODIGO_AUX Identificador_aux ASPAS_DUPLAS VIRGULA ExpressaoAritmetica FECHA_PARENTESIS PVIRGULA ;
```

3.2.8 Scanf

Comando Scanf faz a leitura de dados e armazena em uma variável. O formato aceito é:

scanf("%s", &variavel);
scanf("%f", &variavel);
scanf("%d", &variavel);

```
//SCANF

ScanfComando: SCANF ABRE_PARENTESIS ASPAS_DUPLAS CODIGO_AUX ASPAS_DUPLAS VIRGULA AND IDENTIFICADOR FECHA_PARENTESIS PVIRGULA ;
```

4 Elementos não permitidos na gramática

1. Nossa gramática não aceita como condição $\leq$ ou $\geq$. Caso seja desejado utilizar essa condição ela tem que ser separada em duas.
2. Nossa gramática não aceita !=.
3. Nossa gramática não aceita o inverter o valor de uma condição utilizando o !. Ex: !(a¿0). *

5 Ferramentas adicionais

Para identificarmos os erros sintáticos de forma mais informativa ao usuário utilizamos a diretiva **%define parse.error detailed**. Durante o processo de análise, se algo fora das regras aparecer ela é ativada e instrui o analisador a gerar mensagens de erro que incluem detalhes, como a linha em que o erro ocorreu e qual foi o tipo do problema.

6 Manual de Usuário

6.1 Instalação

Para fazer a instalação da ferramenta Bison, que é um gerador de analisadores sintáticos, basta abrir o terminal no Linux e usar o seguinte comando:

```
sudo apt-get install bison
```

6.2 Execução

Uma vez que a instalação do Bison foi concluída, podemos agora executar o código fonte. Para facilitar a explicação, suponha que o arquivo fonte do Flex se chame anaflex.l e o arquivo fonte do Bison se chame fonte.y.

1. O primeiro passo é processar o arquivo de especificação do Flex e gerar o código C para um analisador léxico. Este comando gera um arquivo chamado lex.yy.c, que contém a implementação do analisador léxico baseado na especificação fornecida no arquivo anaflex.l. O comando para fazer isso é:

   ```
   flex anaflex.l
   ```

2. Em seguida, devemos processar o arquivo de especificação do Bison e gerar o código C para um analisador sintático. O parâmetro -d gera um arquivo de cabeçalho (fonte.tab.h) que contém as definições dos tokens e outras informações necessárias para a construção do parser.

   ```
   bison -d fonte.y
   ```

3. Após isso, compilamos os arquivos gerados pelo Flex e pelo Bison, criando um executável a partir desse código. Aqui, fonte.tab.c é o arquivo gerado pelo Bison, lex.yy.c é o arquivo gerado pelo Flex, e -lfl vincula a biblioteca Flex. O parâmetro -o fonte especifica o nome do executável a ser gerado.

```
gcc fonte.tab.c lex.yy.c -lfl -o fonte
```

4. Por fim, basta executar o programa fonte utilizando um arquivo ".txt" como entrada de dados. Neste caso, entrada.txt contém os dados que deseja-se processar com o analisador sintático e léxico criados. O programa lerá a entrada, analisará os tokens e executará as ações definidas em pela gramática e no analisador léxico.

```
./fonte entrada.txt
```