Relatório descritivo analisador sintático

Julia Rodrigues Gubolin
Docente: Renata Spolon Lobato
Disciplina: Compiladores
Bacharelado em Ciência da Computação UNESP - IBILCE

05 de janeiro

Introdução

Neste relatório, será descrita a linguagem aceita pelo analisador sintático gerado, as expressões aceitas e seus formatos.

Analisador léxico

Algumas alterações no analisar anterior foram feitas, de forma que agora temos alguns novos tokens que são aceitos, além de que são apenas emitidas mensagens de erro, e não mensagens de cada token.

Agora, temos a estrutura de repetição **do while**, 2 operadores lógicos, que são o **OU** e o **E**, e o ID, que são valores que unem letras e números. Além disso, são aceitos também: **(,), {, }**. Os comentários com # foram retirados.

Expressões aceitas

São aceitos pelo analisador sintático: comandos de declaração, atribuição, prints, estruturas condicionais e de repetição. Além disso, temos ainda a distinção entre números reais e inteiros, as expressões aritiméticas e expressões lógicas.

Tokens

Na imagem 1, estão os tokens que estão declarados no arquivo flex/lex "fonte.l" e que serão usados pelo Bison:

Estrutura geral

Na imagem 2, temos a estrutura geral do programa com os comandos possíveis que foram listados acima. O programa principal possui o início, estes comandos e o final.

Figura 1: Tokens.

Declaração de variáveis e tipos numéricos

Na imagem 3, temos os tipos de números (reais e inteiros), além da estrutura de declaração, que pode ser feita de duas formas: a primeira com algum tipo de variável (integer, double, char e boolean), seguida do ID, que representa o nome das variáveis e do ; e a segunda também possui o tipo de variável e o ID, porém após este podemos fazer uma atribuição com o = e uma expressão.

Atribuição e parâmetros

Na imagem 4, temos a maneira como a atribuição mencionada acima é feita. Neste caso, ela pode ser de 3 tipos: após o = podemos encontrar um número e novos valores separados por vígula com ou sem mais atribuições.

Expressões

Na imagem 5, temos as expressões aritméticas e expresões lógicas. As expressões aritméicas podem ser compostas por outras expressões entre parêntesis, apenas por números e relacionados com expressões através de operadores aritméticos

```
/* ESTRUTURA GERAL*/
Programa_principal: INICIO_PROGRAMA comandos FIM_PROGRAMA;

/* Possíveis comandos que podem estar no arquivo teste de entrada */
comandos: decl_var comandos
    | ID atribuicao PONTO_E_VIRGULA comandos
    | condicional comandos
    | loop comandos
    | printar comandos
    | %empty;
```

Figura 2: Estrutura geral.

```
/*Possíveis números*/
var_num: ID
    | NUMERO_REAL
    | NUMERO_INTEIRO;

/* Estrutura de declaração das variáveis */
decl_var: TIPOS_VARIAVEIS ID PONTO_E_VIRGULA
    | TIPOS_VARIAVEIS ID atribuicao PONTO_E_VIRGULA;
```

Figura 3: Declaração e tipos numéricos.

(+, -, *, /). Já as expressões lógicas possuem tanto operadores relacionais quanto lógicos.

Estruturas condicionais e de repetição

Na imagem 6, temos as estruturas condicionais (if, elseif, else) e de repetição (for, while, do while).

Printar

Na imagem 7, temos o printf sem as aspas.

```
/* Estrutura de atribuição */
atribuicao: ATRIBUICAO expressao_arit
    | ATRIBUICAO ID cont_parametros
    | var_num;

cont_parametros: VIRGULA TIPOS_VARIAVEIS ID cont_parametros
    | VIRGULA ID cont_parametros
    | ID cont_parametros
    | %empty;
```

Figura 4: Atribuição e parâmetros.

```
/* Regras das expressões aritméticas */
expressao_arit: arit1 arit2 ;
arit1: ABRE_PARENTESES expressao_arit FECHA_PARENTESES arit2
     | var_num arit2;
arit2: OPERADORES_ARITMETICOS arit1
     | %empty;

/* Regras das expressões lógicas */
expressao_logica: expressao_arit log1 ;
log1: OPERADORES_RELACIONAIS expressao_arit log2
    | %empty;
log2: OPERADORES_LOGICOS expressao_logica
    | %empty;
```

Figura 5: Expressões.

```
bloco_comando: ABRE_CHAVES comandos FECHA_CHAVES;

/* Estruturas condicionais: IF, ELSE */
condicional: IF ABRE_PARENTESES expressao_logica FECHA_PARENTESES bloco_comando cond2 cond1;
cond1: ELSE bloco_comando
    | %empty;
cond2: ELSEIF bloco_comando
    | %empty;

/* Estruturas de repetição: FOR, WHILE E DO WHILE */
loop: FOR ABRE_PARENTESES decl_var expressao_logica PONTO_E_VIRGULA ID atribuicao FECHA_PARENTESES bloco_comando
    | WHILE ABRE_PARENTESES expressao_logica FECHA_PARENTESES bloco_comando;
    | DO ABRE_CHAVES expressao_logica FECHA_CHAVES WHILE ABRE_PARENTESES expressao_logica FECHA_PARENTESES bloco_comando;
```

Figura 6: Estruturas.

```
printar: PRINT ABRE_PARENTESES comandos FECHA_PARENTESES PONTO_E_VIRGULA;

%%
```

Figura 7: Estruturas.