2º Trabalho de Compiladores - Analisador Sintático

Jader Gomes Nascimento e Fernando Guimarães Pinheiro

January 28, 2014

Abstract

Documentação referente à segunda parte do trabalho de compiladores, que consiste em um analisador sintático.

1 Introdução

O objetivo deste trabalho é estabelecer um analisador sintático para a linguagem GPortugol. O analisador sintático é a segunda parte de um compilador, e sua função é analisar a sintaxe dos elementos contidos em um arquivo de entrada (código fonte). Assim, nesta segunda etapa, busca-se verificar se o arquivo de entrada está dentro dos padrões da linguagem GPortugol, que será definida mais abaixo.

Utilizando a ferramenta Bison para gerar um analisador sintático, a tarefa foi facilitada. Nesta documentação, são descritas as regras da linguagem e as dificuldades encontradas pela dupla.

2 Linguagem GPortugol

Nesta seção iremos definir o que é a linguagem e suas restrições.

2.1 Declaração do algoritmo

Iniciamente deve ser declarado o nome do algoritmo. A sintaxe é definida a seguir:

```
algoritmo idade;
```

O nome do algoritmo é obrigatório, devendo sempre estar no ínicio do algortimo.

```
algoritmo
: declaracao_algoritmo bloco_variaveis bloco_inicio declacarao_funcoes
| declaracao_algoritmo bloco_variaveis bloco_inicio
;
```

2.2 Bloco de variavéis

Na linguagem GPortugol, existe um bloco de variavéis onde todas as variavéis podem ser declaradas, como mostra abaixo:

```
variáveis
    idade: inteiro;
    nome: literal;
fim-variáveis
```

Esse bloco de variavéis, como mostrado abaixo é um bloco opcional, mas sempre que for utilizado deve começar com a palavra variavéis e termina com a palavra fim-variavéis. Esse bloco deve sempre estar depois da declaração das variavéis e antes do bloco principal, podendo ser vazio, ou seja, apenas as palavras variavéis e fim-variavéis. Abaixo segue a sintaxe:

```
bloco_variaveis
: token_pr_variaveis declaracao_variaveis token_pr_fim_variaveis
| token_pr_variaveis token_pr_fim_variaveis
|
;

declaracao_variaveis
: lista_variaveis token_dois_pontos tipo_variavel token_ponto_virgula
| declaracao_variaveis lista_variaveis token_dois_pontos tipo_variavel
      token_ponto_virgula
;
```

2.3 Bloco principal

Bloco principal é a regra onde está contido tudo o que pode ser feito dentro de uma função ou comando: atribuições, declarações, chamadas de funções etc. Nesse trabalho o bloco principal foi chamado de bloco ínicio.

```
início
    imprima("Digite um número:");
    fat := leia();
    imprima("Fatorial de ", fat, " é igual a ", fatorial(fat));
fim

bloco_inicio
: token_pr_inicio lista_comandos token_pr_fim
| token_pr_inicio token_pr_fim
;
```

O bloco ínicio deve sempre começar e terminar com as palavras ínicio e fim respectivamente, sendo que todo programa deve existir o ínicio e fim, mesmo que seja vazio. Entre o ínicio e fim tem a lista de comandos que são aceitas pela linguagem.

```
lista_comandos
: lista_comandos comando
```

```
| comando
;

comando
: atribuicao
| chamada_funcao token_ponto_virgula
| chamada_funcao_interna
| comando_retorne
| comando_se
| comando_enquanto
| comando_para
| comando_escolha
;
```

A lista de comandos são todos comandos aceitaveis da linguagem, que pode ser uma atribuição, chamada de função, comando de retorno e comandos condicionais.

2.4 Atribuição

Definimos como atribuição apenas o comando abaixo:

```
x : inteiro;
```

Não é permitida atribuição múltipla, por exemplo, a := b := 1. A seguir, segue como está no .y:

```
valor_esquerda
: token_identificador
| token_identificador matriz_colchetes
;

atribuicao
: valor_esquerda token_atribuicao expressao token_ponto_virgula
;
```

2.5 Expressão

```
expressao
: expressao token_pr_ou termo_1
| expressao token_ou termo_1
| termo_1
;

termo_1
: termo_1 token_pr_e termo_2
| termo_1 token_e termo_2
| termo_2
;
```

```
termo_2
: termo_2 token_ou_bit termo_3
| termo_3
;

termo_3
: termo_3 token_xor_bit termo_4
| termo_4
;

termo_4
: termo_4 token_e_bit termo_5
| termo_5
;

termo_5
: termo_5 token_igual termo_6
| termo_5 token_diferente termo_6
| termo_6
;

termo_6
: termo_6 token_menor termo_7
| termo_6 token_menor_igual termo_7
| termo_6 token_maior termo_7
| termo_6 token_maior_igual termo_7
| termo_7
;

termo_7
: termo_7 token_soma termo_8
| termo_7 token_subtracao termo_8
| termo_8
;

termo_8
: termo_8 token_multiplicacao termo_9
| termo_8 token_divisao termo_9
| termo_8 token_modulo termo_9
| termo_9
;

termo_9
: token_soma termo_9
| token_subtracao termo_9
| token_nao termo_9
| token_abre_parenteses expressao token_fecha_parenteses
| token_identificador
| valor_primitivo
```

```
| chamada_funcao
| chamada_funcao_interna
;
```

Foi divido a expressão em varios termos para que fosse respeitada a ordem de procedência dos operados, assim a multiplicação tem maior procedência que a soma por exemplo. Uma expressão também pode ser uma chamada de função, que sera definida abaixo.

2.6 Chamada de função

As chamadas de função sempre começa com o identificador seguido dos parâmetros da chamada de função, que pode ou não existir, sendo os parâmetros entre parenteses

```
chamada_funcao
: token_identificador token_abre_parenteses paramentros_chamada_funcao
      token_fecha_parenteses
| token_identificador token_abre_parenteses token_fecha_parenteses
;

chamada_funcao_interna
: token_pr_imprima token_abre_parenteses paramentros_chamada_funcao
      token_fecha_parenteses token_ponto_virgula
| token_pr_leia token_abre_parenteses token_fecha_parenteses
;

paramentros_chamada_funcao
: paramentros_chamada_funcao token_virgula expressao
| expressao
;
```

Foi definido também a chamada de função interna, que chama as funções imprima e leia, que sao internas de portugol. Os parametros da chamada de função são expressões seguidas do ponto e virgula podendo ou não ser seguida de outras expressões.

2.7 Comandos de seleção

Definiremos agora os comandos de seleção da linguagem GPortugol.

2.7.1 Se-Senão

Este comando deve sempre começar com a palavra se seguido de uma expressão, depois a palavra então seguida de uma lista de comandos, podendo ou não ser seguido pela palavra senão. E para finalizar o comando deve ao final terminar com a palavra fim-se.

```
comando_se
```

```
: token_pr_se expressao token_pr_entao lista_comandos token_pr_fim_se
| token_pr_se expressao token_pr_entao lista_comandos token_pr_senao
      lista_comandos token_pr_fim_se
;
```

Abaixo um exemplo da utilização do comando:

```
se idade >= 18 então
    se idade < 60 então
retorne "adulto";
    senão
retorne "idoso";
    fim-se
senão
    retorne "jovem";
fim-se
```

2.7.2 Escolha

O parametro da função escolha deve ser uma variavél. A função escolha foi definida como abaixo:

```
comando_escolha
: token_pr_escolha token_abre_parenteses token_identificador
      token_fecha_parenteses casos token_pr_fim_escolha
;

casos
: casos caso
| caso
;

caso
: token_pr_caso valor_primitivo token_dois_pontos lista_comandos
      token_pr_para token_ponto_virgula
| token_pr_caso valor_primitivo token_dois_pontos token_pr_para
      token_ponto_virgula
| token_pr_default token_dois_pontos lista_comandos token_pr_para
      token_ponto_virgula
;
```

Toda função escolha deve ser seguida de pelo menos 1 caso. Os casos são definidos como a palavra caso seguida de um valor primitivo da linguagem, depois dois pontos seguido de uma lista de comandos, todo caso deve sempre terminar com a palavra para seguida de ponto e virgula. A função escolha também pode ter um caso default onde esse caso não recebe paramêtro. A função deve terminar com a palavra fim-escolha. Abaixo um exemplo da função escolha:

```
escolha (x)
    caso "soma" :
```

```
imprima("soma de ", a, " e ", b, " é igual a ", soma(a, b));
para;
    caso "sub" :
imprima("subtração de ", a, " e ", b, " é igual a ", sub(a, b));
para;
    caso "mult" :
imprima("multiplicação de ", a, " e ", b, " é igual a ", mult(a, b));
para;
    caso "div" :
imprima("divisão de ", a, " e ", b, " é igual a ", div(a, b));
para;
    default:
imprima("Operação inválida");
para;
fim-escolha
```

2.8 Comandos de repetição

A seguir definiremos os comandos de repetição: enquanto e para.

2.8.1 Enquanto

O comando enquanto é formado da seguinte forma: enquanto seguido de expressão e da palavra faça, depois tem a lista de comandos seguido da palavra fim-enquanto. Abaixo esta o .y.

```
enquanto x < 1 faça
    x := x + 1;
fim-enquanto
```

```
comando_enquanto
: token_pr_enquanto expressao token_pr_faca lista_comandos token_pr_fim_enquanto
;
```

2.8.2 Para

O comando para vai executar uma lista de comandos enquanto o valor de uma variavel estiver dentro de um intervalo.

```
para i de separador até max passo 1 faça
    imprima(i);
fim-para
```

```
comando_para
: token_pr_para valor_esquerda token_pr_de expressao token_pr_ate expressao
      token_pr_faca lista_comandos token_pr_fim_para
| token_pr_para valor_esquerda token_pr_de expressao token_pr_ate expressao
      passo token_pr_faca lista_comandos token_pr_fim_para
;

passo
```

```
: token_pr_passo token_inteiro
| token_pr_passo token_soma token_inteiro
| token_pr_passo token_subtracao token_inteiro
;
```

No codigo acima mostra que o comando para começa sempre com a palavra para seguida de uma variavél, e temos a palavra de seguida de uma expressão e depois a palavra até seguida de outra expressão, depois temos a palavra faça seguida por uma lista de comandos. O comando sempre deve terminar com a palavra fim-para. O comando para também pode ter a palavra passo, que seria quanto a variavél será incrementada ou decrementada.

3 Exemplos

Nos exemplos, foi explorado diversas situações para os diversos tipos de usuários. Foram criadas 10 situações, 5 situações corretas e 5 situações erradas, tentando utilizar todas as funções que foram definadas.