CENTRO FEDERAL DE EDUCAÇÃO TECNOLÓGICA DE MINAS GERAIS - CEFET MG
DEPARTAMENTO DE COMPUTAÇÃO - COMPILADORES

RELATÓRIO DO TRABALHO PRÁTICO PARTE 2
ANALISADOR SINTÁTICO

Guilherme Moreira de Carvalho

04 de Junho de 2023

INTRODUÇÃO

O analisador sintático implementado recebe como parâmetro um arquivo texto para verificar a concordância com a gramática especificada.

```
> comp [arquivo texto]
```

A implementação, realizada em C++, pode ser compilada com o comando *make*.

O analisador recebe os tokens do arquivo, por meio do analisador léxico implementado na primeira parte do trabalho, e verifica se a sequência está correta. Para isso, a gramática foi definida como LL(1), o que permite a criação de um parser recursivo conhecendo apenas um token.

Entretanto, algumas mudanças foram necessárias:

1. Casos em que o prefixo de duas derivações são iguais

if_stmt := **if** condition **then** stmt_list **end** | **if** condition **then** stmt_list **else** stmt_list **end**	if_stmt := **if** condition **then** stmt_list A A := **end** | **else** stmt_list **end**

expression := simple-expr | simple-expr relop simple-expr	expression := simple-expr B B := λ | relop simple-expr

2. Casos em que há recursão à esquerda

simple-expr := simple-expr addop term | term	simple-expr := term C C := λ | addop term C

term := term mulop factor_a | factor_a	term := factor_a D D := λ | mulop factor_a D

Como saída, são exibidos qual era o token *next* esperado e qual foi de fato o token lido com seus lexema e tipo. Caso haja algum token fora da ordem esperada é exibido uma mensagem de erro. Caso contrário, o processo é repetido até que alcance *EOF*.

CLASSES

- **SINTAX**

Possui como atributos um analisador léxico, que coleciona os tokens a serem analisados, e um token atual a ser comparado com o esperado.

O método *eat()* faz essa comparação e o atualiza se não há divergências ou exibe erro se houver.

Os demais métodos recebem o nome das variáveis não terminais da gramática e implementa o comportamento da derivação dessas: onde há uma variável terminal aplica-se o método eat() sobre ela e, onde há uma outra variável não-terminal, o método relacionado a esta é chamado.

```
// Program ::= program Identifier [DeclList] begin StmtList end “.”
void Sintax::Program() {
    eat(T_PROGRAM);
    eat(T_ID);
    DeclList();
    eat(T_BEGIN);
    StmtList();
    eat(T_END);
    eat(T_PONTO);
```

RESULTADOS

programa teste1
  a, b is int;
  result is int;
  a, x is float;
begin
  a = 12a;
  x = 12.;
  read (a);
  read (b);
  read (c)
  result = (a*b + 1) / (c+2);
  write {Resultado: };
  write (result);
end.
	Next: PROGRAM; Token: “programa”
	Token Inesperado: “programa”, ID
	Linha 1

IMAGEM 1 - Primeiro teste e sua saída

O programa inicia com um erro - a label *program* escrita como *programa* a qualifica como uma variável ao invés de palavra reservada; a atribuição *a = 12a;* gera erro, pois esperava uma constante numérica apenas; o token *12.* é registrado como erro, pois faltam os decimais do número de ponto flutuante; a falta de *‘;’* ao fim do último comando *read* gera erro, pois de acordo com a gramática, statements são delimitados por tal caractere (com exceção do último); falta de “()” delimitando a expressão do método *write* também qualifica erro sintático; a presença de um *‘;’* após o último comando *write* produz erro como explicado anteriormente.

```
program teste1
   a, b is int;
   result is int;
   a, x is float;
begin
   a = 12;
   x = 12.0;
   read (a);
   read (b);
   read (c);
   result = (a*b + 1) / (c+2);
   write ({Resultado: });
   write (result)
end.
```

IMAGEM 2 - Exemplo 1 escrito corretamente e sua saída

program teste2
 	
a, b, c:int;
d, _var: float;
 	
teste2 = 1;
Read (a);
b = a * a;
c = b + a/2 * (35/b);
write c;
val := 34.2
c = val + 2.2 + a;
write (val)
end.
	program teste2
	
	a, b, c is int;
	d, var is float;
	begin
	teste2 = 1;
	read (a);
	b = a * a;
	c = b + a/2 * (35/b);
	write (c);
	val = 34.2;
	c = val + 2.2 + a;
	write (val)
	end.

IMAGEM 3 - Códigos original e correto do exemplo 2

O símbolo *:* não faz parte da linguagem e por isso é considerado inválido; variáveis não podem começar com *'_'*; a atribuição *teste2 = 1* se trata de um erro semântico, não sintático; a escrita errônea do método *read* com 'R' maiúsculo é um erro léxico, pois a linguagem é case sensitive; a falta de *'()'* no método write é reportado como erro bem como falta de *';'* depois do penúltimo statement (atribuição de c).

program
a, aux is int;
b is float
begin
b = 0;
in (a);
in(b);
if (a>b) then //troca variaveis
aux = b;
b = a;
a = aux
end;
write(a;
write(b)
	program teste3
	a, aux is int;
	b is float;
	begin
	b = 0;
	
	if (a>b) then /*troca variaveis*/
	aux = b;
	b = a;
	a = aux
	end;
	write(a);
	write(b)
	end.

IMAGEM 4 - Códigos original e correto do exemplo 3

O tratamento de métodos criados pelo programador é feito pelo analisador semântico.

```
programa teste4
/* Teste4 do meu compilador
      pontuacao, pontuacaoMaxina, disponibilidade is inteiro;
      pontuacaoMinima is char;
begin
   pontuacaoMinima = 50;
   pontuacaoMaxima = 100;
   write({Pontuacao do candidato: });
   read(pontuacao);
   write({Disponibilidade do candidato: });
   read(disponibilidade);

   while (pontuacao>0 & (pontuacao<=pontuacaoMaxima)
do
      if ((pontuação > pontuacaoMinima) &&
(disponibilidade==1)) then
         write({Candidato aprovado.})
      else
         write({Candidato reprovado.})
      end
      write({Pontuacao do candidato: });
      read(pontuacao);
      write({Disponibilidade do candidato: });
      read(disponibilidade);
   end
end
```

```
program teste4
/* Teste4 do meu compilador*/
      pontuacao, pontuacaoMaxina, disponibilidade is int;
      pontuacaoMinima is char;
begin
   pontuacaoMinima = 50;
   pontuacaoMaxima = 100;
   write({Pontuacao do candidato: });
   read(pontuacao);
   write({Disponibilidade do candidato: });
   read(disponibilidade);

   while (pontuacao>0 &&
(pontuacao<=pontuacaoMaxima)) do
      if ((pontuacao > pontuacaoMinima) &&
(disponibilidade==1)) then
         write({Candidato aprovado.})
      else
         write({Candidato reprovado.})
      end;
      write({Pontuacao do candidato: });
      read(pontuacao);
      write({Disponibilidade do candidato: });
      read(disponibilidade)
   end
end.
```

IMAGEM 5 - IMAGEM 4 - Códigos original e correto do exemplo 4

O comentário não foi fechado, assim todo o corpo do programa não é computado.

```
/* Teste do meu compilador */
program teste5
   a, b, c, maior is int;
   outro is char;
begin
   repeat
      write({A});
      read(a);
      write({B});
      read(b);
      write({C});
      read(c);
      if ( (a>b) && (a>c) ) end
         maior = a

      else
         if (b>c) then
            maior = b;

         else
            maior = c
         end
      end;
      write({Maior valor:}});
      write (maior);
      write ({Outro? (S/N)});
      read(outro);
   until (outro == 'N' || outro == 'n)
end
```

```
/* Teste do meu compilador */
program teste5
   a, b, c, maior is int;
   outro is char;
begin
   repeat
      write({A});
      read(a);
      write({B});
      read(b);
      write({C});
      read(c);
      if ( (a>b) && (a>c) ) then
         maior = a

      else
         if (b>c) then
            maior = b

         else
            maior = c
         end
      end;
      write({Maior valor:});
      write (maior);
      write ({Outro? (S/N)});
      read(outro)
   until (outro == 'N' || outro == 'n')
end.
```

IMAGEM 6 - IMAGEM 4 - Códigos original e correto do exemplo 5