Trabalho Prático de Compiladores

Relatório Parte 2

Jonathan Henrique Silva
Arthur de Freitas Abeilice
30 de outubro de 2019

INTRODUÇÃO

Esta é a parte 2 do projeto de compiladores, em que devemos criar um compilador para poder interpretar a linguagem que foi definida para a turma. Essa etapa foi feita em Java, assim como as próximas partes serão, e a primeira parte foi feita. A Segunda parte do trabalho prático trata-se de fazer o analisador sintático, que compõe a representação e análise das regras de disposição das palavras, ou, melhor dizendo, os tokens, nas linhas dos códigos que serão testados de acordo com as regras da gramática que foram atualizadas.

EXECUTANDO O COMPILADOR

Abrir o netbeans, adicionar um novo projeto que estará nos arquivos do trabalho enviado, e escolher rodar o compilador no 'run'. Alterar os argumentos para "teste1.txt", ou para qualquer outro teste, e depois rodar com o argumento escolhido.

METODOLOGIA

A técnica aprendida do parser descendente foi escolhida, ,e, portanto, a linguagem foi gramática foi adaptada para ser LL(0). Foi analisada e reconstruída a gramática dada, e construído as tabelas de first e follow, como também uma tabela do parser.

As mudanças que fizemos na linguagem foram ter modificado decl-list , ident-list, stmt-list, if-stmt , expression, simple-exp e term, criando decl-rec, ident-rec, stmt-rec, exp-end, simple-end e term-end na gramática.

	FIRST	**FOLLOW**
p-start	start	$
program	int, float, string, identifier, if, do, scan, print	$
decl-list	int, float, string	identifier, if, do, scan, print
decl-rec	int, float, string	identifier, if, do, scan, print
decl	int, float, string	int, float, string, identifier, if, do, scan, print
ident-list	identifier	";"
ident-rec	",", λ	";"
type	int, float, string	identifier
stmt-list	identifier, if, do, scan, print	exit

stmt-rec	identifier, if, do, scan, print	exit
stmt	identifier, if, do, scan, print	identifier, if, do, scan, print, exit
assign-stmt	identifier	";"
if-stmt	if	identifier, if, do, scan, print, exit
if-end	else, end	identifier, if, do, scan, print, exit
condition	"(", identifier, integer_const, float_const, literal, not, "-"	then, end
while-stmt	do	identifier, if, do, scan, print, exit
stmt-sufix	while	identifier, if, do, scan, print, exit
read-stmt	scan	";"
write-stmt	print	";"
writable	"(", identifier, integer_const, float_const, literal, not, "-", literal	")"
expression	"(", identifier, integer_const, float_const, literal, not, "-"	then, ")"
exp-end	"==", ">", ">=", "<", "<=", "<>", λ	then, ")"
simple-exp	"(", identifier, integer_const, float_const, literal, not, "-"	"==", ">", ">=", "<", "<=", "<>", then, ")"
simple-end	"+", "-", or, λ	"==", ">", ">=", "<", "<=", "<>", then, ")"
term	"(", identifier, integer_const, float_const, literal, not, "-"	"+", "-", or, λ
term-end	"*", "/", and, λ	"+", "-", or, λ
factor-a	"(", identifier, integer_const, float_const, literal, not, "-"	"*", "/", and, λ
factor	"(", identifier, integer_const, float_const, literal	"*", "/", and, λ
relop	"==", ">", ">=", "<", "<=", "<>"	"(", identifier, integer_const, float_const, literal, not, "-"
addop	"+", "-", or	"(", identifier, integer_const, float_const, literal, not, "-"
mulop	"*", "/", and	"(", identifier, integer_const, float_const, literal, not, "-"
constant	integer_const, float_const, literal	"*", "/", and, λ

Gramática Final

```
p-start ->      start program
program ->      decl-list stmt-list exit |
                stmt-list exit
decl-list ->    decl decl-rec
decl-rec ->     decl decl-rec |
                λ
decl ->         type ident-list ";"
ident-list ->   identifier ident-rec
ident-rec ->    "," identifier ident-rec |
                 λ
type ->         int |
                float |
                string
stmt-list ->    stmt stmt_rec
stmt-rec ->     stmt stmt-rec |
                λ
stmt ->         assign-stmt ";" |
                if-stmt |
                while-stmt |
                read-stmt ";" |
                write-stmt ";"
assign-stmt ->  identifier "=" simple_expr
if-stmt ->      if condition then stmt-list if-end
If-end ->       end |
                else stmt-list end
condition ->    expression
while-stmt ->   do stmt-list stmt-sufix
stmt-sufix ->   while condition end
read-stmt ->    scan "(" identifier ")"
write-stmt ->   print "(" writable ")"
writable ->     simple-expr
expression ->   simple-expr exp-end

exp-end ->      relop simple-expr |
                λ
simple-expr->   term simple-end
simple-end->    addop term simple-end|
                λ
term ->         factor-a term-end
term-end ->     mulop factor-a term-end |
                λ
fator-a ->      factor |
                not factor |
                "-" factor
factor ->       identifier |
                constant |
                "(" expression ")"
relop ->        "==" | ">" | ">=" | "<" | "<=" | "<>"
addop ->        "+" | "-" | or
mulop ->        "*" | "/" | and
constant ->     integer_const | float_const | literal
```

Para a análise sintática, foi criada somente uma classe nova, SyntaxAnalyser, com todos os métodos responsáveis por essa análise. Além disso, foi adicionada uma chamada à essa classe na classe principal, logo após a análise léxica, passando a lista de tokens identificados para serem analisados.

Essa classe possui os métodos *eat*, que consome uma token e chama o método de erro se a token não era a esperada, além de métodos para cada produção da gramática, usando de switches e do método *eat* para validar se o código está correto.

Ela também possui o método *error*, que é chamado quando um token não esperado é encontrado. Esse método é responsável pela recuperação de erro, que, no nosso caso, simplesmente ignora a token e continua a leitura. Embora isso funcione em muitos casos, em alguns o analisador fica incerto em relação ao que ele deveria esperar para prosseguir, resultando num mesmo erro sendo repetido para cada token até o final da “subprodução”, como será visível nos testes a seguir.

RESULTADOS

- teste1.txt

```
start
    int a, b;
    int result;
    float a,x,total;
    a = 2;
    x = 1.1;
    scan (b);
    scan (y)
    result = (a*b ++ 1) / 2;
    print “Resultado: ”;
    print (result);
    print (“Total: ”);
    total = y / x;
    print (“Total: ”;
    print (total);
exit
```

- teste 1 resultado

Unexpected token "result" at line 9
Unexpected token "+" at line 9
Unexpected token "Resultado: " at line 10
Unexpected token ";" at line 10
Unexpected token ";" at line 14

- **teste2.txt**

```
start
    int a, c;
    float d, e;
    a = 0; d = 3.5
    c = d / 1.2;
    Scan (a);
    Scan (c);
    b = a * a;
    c = b + a * (1 + a*c);
    print (“Resultado: ”);
    print c;
    d = 34.2
    e = val + 2.2;
    print (“E: ”);
    print (e);
    a = b + c + d)/2;
```

- **teste 2 resultado**

Unexpected token "c" at line 5
Unexpected token "(" at line 6
Unexpected token "(" at line 7
Unexpected token "c" at line 11
Unexpected token ";" at line 11
Unexpected token "e" at line 13
Unexpected token ")" at line 16
Unexpected token ")" at line 16
Unexpected token ")" at line 16

- **teste3.txt**

```
int pontuacao, pontuacaoMaxina, disponibilidade;
string pontuacaoMinima;
disponibilidade = “Sim”;
pontuacaoMinima = 50;
pontuacaoMaxima = 100;
/* Entrada de dados
Verifica aprovação de candidatos */
do
print(“Pontuacao Candidato: ”);
scan(pontuacao);
print(“Disponibilidade Candidato: ”);
scan(disponibilidade);
if ((pontuação > pontuacaoMinima) and (disponibilidade==“Sim”) then
out(“Candidato aprovado”);
else
out(“Candidato reprovado”)
end
while (pontuação >= 0)end
exit
```

- **teste 3 resultado**

Unexpected token "int" at line 0
Unexpected token "then" at line 13
Unexpected token "(" at line 14
Unexpected token "(" at line 16
Unexpected token "end" at line 17

- **teste4.txt**

```
start
    Int a, aux, b;
    string nome, sobrenome, msg;
    print(“Nome: ”);
    scan (nome);
    print(“Sobrenome: ”);
    scan (sobrenome);
    msg = “Ola, ” + nome + “ ” +
    sobrenome + “!”;
    msg = msg + 1;
    print (msg);
    scan (a);
    scan(b);
    if (a>b) then
    aux = b;
    b = a;
    a = aux;
    end;
    print (“Apos a troca: ”);
    out(a);
    out(b)
exit
```

- **teste 4 resultado**

Unexpected token "a" at line 2
Unexpected token "," at line 2
Unexpected token "," at line 2
Unexpected token "," at line 2

- **teste5.txt**

```
start
    int a, b, c, maior, outro;
    do
       print(“A”);
       scan(a);

       print(“B”);
       scan(b);
       print(“C”);
       scan(c);
```

```
            //Realizacao do teste
            if ( (a>b) and (a>c)
            maior = a
            )
            else
            if (b>c) then
            maior = b;
            else
            maior = c;
            end
            end
            print(“Maior valor:”);
            print (maior);
            print (“Outro? ”);
            scan(outro);
        while (outro >= 0);
    exit
```

- **teste 5 resultado**

Unexpected token ")" at line 11
Unexpected token ")" at line 11
Unexpected token "maior" at line 12
Unexpected token "maior" at line 12
Unexpected token ")" at line 13
Unexpected token ")" at line 13
Unexpected token ")" at line 13
Unexpected token ")" at line 13
Unexpected token ")" at line 13
Unexpected token ")" at line 13
Unexpected token ")" at line 13

- **teste6.txt**

```
start
    int a;
    a = 1;
    do
       if ( a > 0) then
            print(“Tem que fazer um teste aqui”);
       else
            print(“testado”);
       end
       a = a-1;
    while ( a > 0 ) end
exit
```

- **teste 6 resultado**

Compilado sem erros.

Testes Corrigidos:

Teste 1

```
start
      int a, b;
      int result;
      float a,x,total;
      a = 2;
      x = 1.1;
      scan (b);
      scan (y);
      result = (a*b + 1) / 2;
      print (“Resultado: ”);
      print (result);
      print (“Total: ”);
      total = y / x;
      print (“Total: ”);
      print (total);
exit
```

Teste 2

```
start
      int a, c;
      float d, e;
      a = 0; d = 3.5;
      c = d / 1.2;
      scan (a);
      scan (c);
      b = a * a;
      c = b + a * (1 + a*c);
      print (“Resultado: ”);
      print (c);
      d = 34.2;
      e = val + 2.2;
      print (“E: ”);
      print (e);
      a = b + c + d/2;
exit
```

Teste 3

```
start
int pontuacao, pontuacaoMaxina,
disponibilidade;
string pontuacaoMinima;
disponibilidade = “Sim”;
pontuacaoMinima = 50;
pontuacaoMaxima = 100;
/* Entrada de dados
Verifica aprovação de candidatos */
do
print(“Pontuacao Candidato: ”);
scan(pontuacao);
print(“Disponibilidade Candidato: ”);
scan(disponibilidade);
if ((pontuação > pontuacaoMinima) and
(disponibilidade==“Sim”)) then
print(“Candidato aprovado”);
else
print(“Candidato reprovado”);
end
while (pontuação >= 0)end
exit
```

Teste 4

```
start
      int a, aux, b;
      string nome, sobrenome, msg;
      print(“Nome: ”);
      scan (nome);
      print(“Sobrenome: ”);
      scan (sobrenome);
      msg = “Ola, ” + nome + “ ” +
      sobrenome + “!”;
      msg = msg + 1;
      print (msg);
      scan (a);
      scan(b);
      if (a>b) then
      aux = b;
      b = a;
      a = aux;
      end
      print (“Apos a troca: ”);
      print(a);
      print(b);
exit
```

Teste 5

```
start
      int a, b, c, maior, outro;
      do
            print(“A”);
            scan(a);
            print(“B”);
            scan(b);
            print(“C”);
            scan(c);
            //Realizacao do teste
            if ( (a>b) and (a>c)) then
            maior = a;
            else
            if (b>c) then
            maior = b;
            else
            maior = c;
            end
            end
            print(“Maior valor:”);
            print (maior);
            print (“Outro? ”);
            scan(outro);
      while (outro >= 0) end
exit
```

Após as modificações, todos os testes executaram sem erros.

CONCLUSÃO

No final, a analisador sintático funcionou perfeitamente como o esperado, tendo em conta as escolhas feitas para a sua implementação. A sua implementação contribuiu para o entendimento do que foi visto em aula, além de uma forma de colocar tais conhecimentos em prática.