Centro Federal de Educação Tecnológica

Compiladores

Implementação de um Compilador

Analisador Sintático (Parser)

Alunos:
Lucas Loscheider Reis Muniz
Pedro Brandão Belisário

18 de junho, 2022

1 Introdução

Este trabalho foi realizado com o intuito de desenvolver um compilador em linguagem Java para gramática proposta segundo a estrutura presente no livro "Compiladores - princípios, técnicas e ferramentas". Dessa forma, este relatório tem como objetivo descrever as decisões tomadas durante o projeto para a construção do analisador sintático.

2 Analisador Léxico

Inicialmente, criamos uma classe chamada Parser. Nesta classe instanciamos a classe Lexer do projeto anterior a qual armazena palavras reservadas e também faz a leitura dos caracteres e monta os tokens. Também da classe léxico, instanciamos um token chamado de look. ele armazenará o token em si o qual é enviado ao Parser.

Começando pelo construtor da classe **Parser**, adicionamos várias tags de operação em vetores do tipo tag divididos por tipos. o Vetor relop contém comparadores, o vetor addop contém contas de adição e o operador or e por último o vetor mulop o qual contém operações de multiplicação e o comparador and. No construtor instanciamos o léxico.

Na função **move** logo abaixo, nosso token look recebe o scan o qual identifica o token lido.

A função **error** armezena as mensagens de erros sintáticos que serão retornadas ao termino da análise léxica. Esses erros, na maioria dos casos, não impedem a análise sintática de prosseguir e eles também informam a linha em que aconteceu o erro.

Na função **match**, recebemos a tag do token esperado de acordo com a gramática proposta, de forma que caso o token recebido pelo sintático seja diferente do esperado uma mensagem de erro é gerada sinalizando o tipo do esperado do token.

3 Recuperação de Erros

Para a recuperação de erros do parser, a heurística por nós proposta é que caso seja encontrado algum erro sintático, o parser, após armazenar a mensagem de erro, continuará funcionando assumindo que o token que estava

incorreto era, na verdade, o token esperado. Esta abordagem para a recuperação de erros por ser ingênua ao partir do princípio que o programador gerou o erro sintático por ter digitado errado o próximo token esperado, se mostrou muito pouco eficiente, como será visto durante os testes.

Ressalta-se também que também ocorre a recuperação de erros sintáticos ocorre mesmo quando há erros léxicos no programa proposto, de forma que caso seja encontrado algum erro léxico, ele será armazenado num vetor de erros léxicos e o parser pedirá o próximo pelo próximo token até que seja um token válido.

4 Manipulação da Gramática

Como o parser desenvolvido era um LL(1), a gramática originalmente proposta (Gramática 1), teve que passar por pequenas alterações em suas linhas 10, 11 e 20 e 21 devido ao prefixo comum para a construção do **if-stmt** e da **expression**, respectivamente. E nas linha 23 e 25 devido à recursão à esquerda que ocorreram na construção do **simple-expr** e **term**, respectivamente.

Desta forma após a manilação destas construções de forma que a gramática se torne livre de contexto e possa ser analisada por um parser LL1, chegamos na gramática 2, em que todas as contruções que impediriam a análise sintática se tornaram em duas construções, como pode ser visto nas linhas 10 e 11, 20 e 21, 22 e 23, 24 e 25 da gramática 2, para as construções do **if-stmt**, **expression**, **simple-expr** e **term**, respectivamente.

Algorithm 1 Gramática proposta

program ::= **routine** body
body ::= [decl-list] **begin** stmt-list **end**
decl-list ::= **declare** decl{“;”decl “;”}
decl ::= type ident-list
ident-list ::= identifier {“,” identifier}
type ::= **int** | **float** | **char**
stmt-list ::= stmt{“;”stmt}
stmt ::= assign-stmt | if-stmt | while-stmt | repeat-stmt | read-stmt | write-stmt
assign-stmt ::= identifier “:=” simple-expr
if-stmt ::= **if** condition **then** stmt-list **end**
| **if** condition **then** stmt-list **else** stmt-list **end**
condition ::= expression
repeat-stmt ::= **repeat** stmt-list stmt-suffix
stmt-suffix ::= **until** condition
while-stmt ::= stmt-prefix stmt-list **end**
stmt-prefix ::= **while** condition **do**
read-stmt ::= **read** “(” identifier “)”
write-stmt ::= **write** “(” writable “)”
writable ::= simple-expr | literal
expression ::= simple-expr
| simple-expr relop simple-expr
simple-expr ::= term
| simple-expr addop term
term ::= factor-a
| term mulop factor-a
factor-a ::= factor | **not** factor | “-” factor
factor ::= identifier | constant | “(” expression “)”
relop ::= “=” | “>” | “>=” | “<” | “<=” | “<>”
addop ::= “+” | “-” | **or**
mulop ::= “*” | “/” | **and**
constant ::= integer_const | float_const | char_const
integer_const ::= digit^+
float_const ::= digit^+ “.” digit^+
char_const ::= “ ‘ ” carac “ ’ ”
literal ::= “ “ ” caractere* “ ” ”
identifier ::= letter (letter | digit)*
letter ::= [A-Za-z]
digit ::= [0-9]
carac ::= um dos caracters ASCII
caractere ::= um dos caracters ASCII, exceto as aspas e quebra de linha

Algorithm 2 Gramática corrigida

1: program ::= **routine** body
2: body ::= [decl-list] **begin** stmt-list **end**
3: decl-list ::= **declare** decl{“;”decl “;”}
4: decl ::= type ident-list
5: ident-list ::= identifier {“,” identifier}
6: type ::= **int** | **float** | **char**
7: stmt-list ::= stmt{“;”stmt}
8: stmt ::= assign-stmt | if-stmt | while-stmt | repeat-stmt | read-stmt | write-stmt
9: assign-stmt ::= identifier “:=” simple-expr
10: if-stmt ::= **if** condition **then** stmt-list if-stmt-linha
11: if-stmt-linha ::= **end** | **else** stmt-list **end**
12: condition ::= expression
13: repeat-stmt ::= **repeat** stmt-list stmt-suffix
14: stmt-suffix ::= **until** condition
15: while-stmt ::= stmt-prefix stmt-list **end**
16: stmt-prefix ::= **while** condition **do**
17: read-stmt ::= **read** “(” identifier “)”
18: write-stmt ::= **write** “(” writable “)”
19: writable ::= simple-expr | literal
20: expression ::= simple-expr expression-linha
21: expression-linha ::= { relop simple-expr }
22: simple-expr ::= term simple-expr-linha
23: simple-expr-linha ::= { addop term simple-expr-linha }
24: term ::= factor-a term-linha
25: term-linha ::= { mulop factor-a term-linha }
26: factor-a ::= factor | **not** factor | “-” factor
27: factor ::= identifier | constant | “(” expression “)”
28: relop ::= “=” | “>” | “>=” | “<” | “<=” | “<>”
29: addop ::= “+” | “-” | **or**
30: mulop ::= “*” | “/” | **and**
31: constant ::= integer_const | float_const | char_const
32: integer_const ::= digit^+
33: float_const ::= digit^+ “.” digit^+
34: char_const ::= “ ‘ ” carac “ ’ ”
35: literal ::= “ “ ” caractere* “ ” ”
36: identifier ::= letter (letter | digit)*
37: letter ::= [A-Za-z]
38: digit ::= [0-9]
39: carac ::= um dos caracters ASCII
40: caractere ::= um dos caracters ASCII, exceto as aspas e quebra de linha

5 Testes

5.1 Teste 1

Algorithm 3 Teste 1

```
1: routine
2: declare
3: int a, b;
4: int resul;
5: float a, x;
6: begin
7: a := 12a;
8: x := 12;
9: read (a);
10: read (b);
11: read (c)
12: result := (a*b + 1) / (c+2);
13: write "Resultado: ";
14: write (result);
15: end
```

Na primeira execução do Teste 1 (Agoritmo 3), gerou somente um erro na linha 7, pois como foi encontrado o token "a´´, ao invez de um ponto e virgula, que significaria que haveria mais construções após esta linha, o compilador seguindo a heuristica proposta no capítulo 3, considerou que seria esperado o token "end´´ ao final do programa e terminou a execução. Após a correção deste erro, foi testado o Algoritimo 4, que gerou o mesmo tipo de erro, porém desta vez na linha 11;

Algorithm 4 Teste 1 - Primeira correção

```
1: routine
2: declare
3: int a, b;
4: int resul;
5: float a, x;
6: begin
7: a := 12;
8: x := 12;
9: read (a);
10: read (b);
11: read (c)
12: result := (a*b + 1) / (c+2);
13: write "Resultado: ";
14: write (result);
15: end
```

Após a correção do Algoritmo 4, foi feito um novo teste (Algoritmo 5 que gerou as seguintes mensagens de erro:

1. Próximo à linha 14: Erro sintático - Era esperado um ABRE_PARENTESES
2. Próximo à linha 14: Esperava um IDENTIFICADOR, CONSTANTE OU ABRE PARENTESES
3. Próximo à linha 14: Erro sintático - Era esperado um FECHA_PARENTESES
4. Próximo à linha 15: Erro sintático - Era esperado um END

Que após suas correções, como pode ser visto no algoritmo 6 não gerou mais nenhum erro sintático

Algorithm 5 Teste 1 - Segunda correção

```
1: routine
2: declare
3: int a, b;
4: int resul;
5: float a, x;
6: begin
7: a := 12;
8: x := 12;
9: read (a);
10: read (b);
11: read (c);
12: result := (a*b + 1) / (c+2);
13: write "Resultado: ";
14: write (result);
15: end
```

Algorithm 6 Teste 1 - Correção Final

```
1: routine
2: declare
3: int a, b;
4: int resul;
5: float a, x;
6: begin
7: a := 12;
8: x := 12;
9: read (a);
10: read (b);
11: read (c);
12: result := (a*b + 1) / (c+2);
13: write ("Resultado: ");
14: write (result)
15: end
```

5.2 Teste 2

Algorithm 7 Teste 2

```
1: routine
2: int a, b, c;
3: float d, var
4: begin
5: read (a);
6: b := a * a;
7: c := b + a/2 * (35/b); %aplica formula%
8: write c;
9: val := 34.2
10: c = val + 2.0 + a;
11: write (val)
12: end
```

Na primeira execução do Teste 2 (Agoritmo 3) como estava faltando a palavra **declare** após o **routine**, o compilador, por causa da heuristica gerou proposta, gerou inumeras mensagens de erros porque não ocorreu sincronia entre a execução do parser e os tokens enviados pelo léxico. Desta forma, por praticidade optamos por primeiro fazer esta correção a avaliar todo as mensagens buscando por qual poderia corresponder à uma correta. De forma semelhante ao primeiro erro, ao executarmos o código com sua correção (Algoritmo 8, ocorreu uma falha na sincronia entre o léxico e sintático após reportar a ausência de ponto e virgula na linha 3, pois ao considerar que o **begin** que o léxico havia passado como o ponto e vírgula que era esperado, o parser ao não encontrar um **begin** reportou que todas as linhas até o final do arquivo estavam incorretas.

Algorithm 8 Teste 2 - Primeira correção

```
1: routine
2: declare
3: int a, b, c;
4: float d, var
5: begin
6: read (a);
7: b := a * a;
8: c := b + a/2 * (35/b); %aplica formula%
9: write c;
10: val := 34.2
11: c = val + 2.0 + a;
12: write (val)
13: end
```

Após a correção do Algoritmo 8, foi feito um novo teste (Algoritmo 9 que gerou as seguintes mensagens de erro:

1. Próximo à linha 9: Erro sintático - Era esperado um ABRE_PARENTESES
2. Próximo à linha 9: Esperava um IDENTIFICADOR, CONSTANTE OU ABRE PARENTESES
3. Próximo à linha 9: Erro sintático - Era esperado um FECHA_PARENTESES
4. Próximo à linha 10: Erro sintático - Era esperado um END

Que após suas correções, como pode ser visto no algoritmo 10, gerou a mensagem de erro que era esperado uma atribuição na linha 11, e após esta correção (Algoritmo 11 não gerou mais nenhum erro sintático.

Algorithm 9 Teste 1 - Segunda correção

```
1: routine
2: declare
3: int a, b, c;
4: float d, var;
5: begin
6: read (a);
7: b := a * a;
8: c := b + a/2 * (35/b); %aplica formula%
9: write c;
10: val := 34.2
11: c = val + 2.0 + a;
12: write (val)
13: end
```

Algorithm 10 Teste 2 - Terceira correção

```
1: routine
2: declare
3: int a, b, c;
4: float d, var
5: begin
6: read (a);
7: b := a * a;
8: c := b + a/2 * (35/b); %aplica formula%
9: write (c);
10: val := 34.2;
11: c = val + 2.0 + a;
12: write (val)
13: end
```

Algorithm 11 Teste 2 - Correção Final

```
1: routine
2: declare
3: int a, b, c;
4: float d, var
5: begin
6: read (a);
7: b := a * a;
8: c := b + a/2 * (35/b); %aplica formula%
9: write (c);
10: val := 34.2;
11: c := val + 2.0 + a;
12: write (val)
13: end
```

5.3 Teste 3

Algorithm 12 Teste 3

```
1: routine
2: declare
3: int a, aux;
4: float b;
5: begin
6: B := 0;
7: read (a);
8: read(b);
9: if (a¡¿) b then
10: begin
11:
12: if (a¿b) then
13: aux := b;
14: b := a;
15: a := aux;
16: end;
17: write(a;
18: write(b)
19: end
20: else
21: write("Numeros iguais.");
22: end
```

Na primeira execução do Teste 3 (Agoritmo 12), foram gerados os erros:

1. Próximo à linha 9: Esperava um IDENTIFICADOR, CONSTANTE OU ABRE PARENTESES
2. Próximo à linha 9: Erro sintático - Era esperado um THEN
3. Próximo à linha 9: Esperava um IF, WHILE, REPEAT, READ, WRITE
4. Próximo à linha 9: Erro sintático - Era esperado um END
5. Próximo à linha 10: Erro sintático - Era esperado um END

Que ao corrigí-los, e ao testar o algoritmo 13, foi gerado o conjunto de erros abaixo:

1. Próximo à linha 15: Esperava um IF, WHILE, REPEAT, READ, WRITE
2. Próximo à linha 16: Erro sintático - Era esperado um FECHA_PARENTESES
3. Próximo à linha 17: Erro sintático - Era esperado um END
4. Próximo à linha 17: Erro sintático - Era esperado um END

Algorithm 13 Teste 3 - Primeira correção

```
1: routine
2: declare
3: int a, aux;
4: float b;
5: begin
6: B := 0;
7: read (a);
8: read(b);
9: if (a¡¿ b) then
10: begin
11:
12: if (a¿b) then
13: aux := b;
14: b := a;
15: a := aux;
16: end;
17: write(a);
18: write(b)
19: end
20: else
21: write("Numeros iguais.");
22: end
```

E ao corrigí-los, o algoritmo 14, foi gerado somente um erro na linha 20, pois a última construção do programa não deve estar seguida por um ponto

e virgula, como pode ser visto nas linha 2 e 7 da gramática 2. Por fim, não houve mais erros sintáticos no algoritmo 15. Vale ressaltar que optamos por retirar o comando **else** na linha 20 ao invés do comando **end** da linha 19 do algoritmo 13, pois após conversarmos decidimos que desta forma torna-se claro o parser é responsável por avaliar se o programa está correto de acordo com a gramática da linguagem e não avaliar a corretude do programa, ou seja, se o resultado apresentado é aquele esperado pelo programador.

Algorithm 14 Teste 3 - Segunda correção

```
1: routine
2: declare
3: int a, aux;
4: float b;
5: begin
6: B := 0;
7: read (a);
8: read(b);
9: if (a¡¿ b) then
10: begin
11:
12: if (a¿b) then
13: aux := b;
14: b := a;
15: a := aux;
16: end;
17: write(a);
18: write(b)
19: end;
20: write("Numeros iguais.");
21: end
```

Algorithm 15 Teste 1 - Correção Final

```
routine
declare
int a, aux;
float b;
begin
B := 0;
read (a);
read(b);
if (a¡¿ b) then
begin

if (a¿b) then
aux := b;
b := a;
a := aux;
end;
write(a);
write(b)
end;
write("Numeros iguais.")
end
```

5.4 Teste 4

Algorithm 16 Teste 4

```
1: routine
2: declare
3: int pontuacao, pontuacaoMaxina, disponibilidade;
4: char pontuacaoMinima;
5: begin
6: pontuacaoMinima = 50;
7: pontuacaoMaxima = 100;
8: write(”Pontuacao do candidato: ”);
9: read(pontuacao);
10: write(”Disponibilidade do candidato: ”);
11: read(disponibilidade);
12: %
13: Processamento
14: %
15: while (pontuacao¿0 and (pontuacao¡=pontuacaoMaxima) do
16: if ((pontuação ¿ pontuacaoMinima) and (disponibilidade=1)) then
17: write(”Candidato aprovado.”)
18: else
19: write(”Candidato reprovado.”)
20: end
21: write(”Pontuacao do candidato: ”);
22: read(pontuacao);
23: write(”Disponibilidade do candidato: ”);
24: read(disponibilidade);
25: end;
26: end
```

Na primeira execução do Teste 4 (Agoritmo 16), foram gerados os erros:

1. Próximo à linha 6: Erro sintático - Era esperado um ATRIBUICAO
2. Próximo à linha 7: Erro sintático - Era esperado um ATRIBUICAO
3. Próximo à linha 15: Erro sintático - Era esperado um FECHA_PARENTESES

4. Próximo à linha 16: Erro sintático - Era esperado um DO
5. Próximo à linha 16: Esperava um IF, WHILE, REPEAT, READ, WRITE
6. Próximo à linha 16: Erro sintático - Era esperado um END
7. Próximo à linha 16: Erro sintático - Era esperado um END

Que ao corrigí-los, e ao testar o algoritmo 17, foi gerado o somente o erro abaixo:

1. Próximo à linha 25: Esperava um IF, WHILE, REPEAT, READ, WRITE

Que após corrigí-lo e testar o algoritmo 18 não foi gerado mais nenhum erro sintático.

Algorithm 17 Teste 4 - Primeira correção

```
1: routine
2: declare
3: int pontuacao, pontuacaoMaxina, disponibilidade;
4: char pontuacaoMinima;
5: begin
6: pontuacaoMinima := 50;
7: pontuacaoMaxima := 100;
8: write("Pontuacao do candidato: ");
9: read(pontuacao);
10: write("Disponibilidade do candidato: ");
11: read(disponibilidade);
12: %
13: Processamento
14: %
15: while (pontuacao¿0 and (pontuacao¡=pontuacaoMaxima)) do
16: if ((pontuação ¿ pontuacaoMinima) and (disponibilidade=1)) then
17: write("Candidato aprovado.")
18: else
19: write("Candidato reprovado.")
20: end;
21: write("Pontuacao do candidato: ");
22: read(pontuacao);
23: write("Disponibilidade do candidato: ");
24: read(disponibilidade);
25: end
26: end
```

Algorithm 18 Teste 4 - Correção Final

```
1: routine
2: declare
3: int pontuacao, pontuacaoMaxina, disponibilidade;
4: char pontuacaoMinima;
5: begin
6: pontuacaoMinima := 50;
7: pontuacaoMaxima := 100;
8: write("Pontuacao do candidato: ");
9: read(pontuacao);
10: write("Disponibilidade do candidato: ");
11: read(disponibilidade);
12: %
13: Processamento
14: %
15: while (pontuacao¿0 and (pontuacao¡=pontuacaoMaxima)) do
16: if ((pontuação ¿ pontuacaoMinima) and (disponibilidade=1)) then
17: write("Candidato aprovado.")
18: else
19: write("Candidato reprovado.")
20: end;
21: write("Pontuacao do candidato: ");
22: read(pontuacao);
23: write("Disponibilidade do candidato: ");
24: read(disponibilidade)
25: end
26: end
```

5.5 Teste 5

Algorithm 19 Teste 5

```
1: declare
2: integer a, b, c, maior;
3: char outro;
4: begin
5: repeat
6: write("A: ");
7: read(a);
8: write("B: ");
9: read(b);
10: write("C: ");
11: read(c);
12: if ( (a¿b) and (a¿c) ) end
13: maior := a
14:
15: else
16: if (b¿c) then
17: maior := b;
18:
19: else
20: maior := c
21: end
22: end
23: write("Maior valor:");
24: write (maior);
25: write ("Outro? (S/N)");
26: read(outro);
27: until (outro = 'N' or outro = 'n')
28: end
```

Na primeira execução do Teste 5 (Agoritmo 19), foram gerados os erros:

1. Próximo à linha 1: Erro sintático - Era esperado um ROUTINE
2. Próximo à linha 2: Erro sintático - Era esperado um DECLARE

3. Próximo à linha 2: Erro sintático - Era esperado um BASIC
4. Próximo à linha 2: Erro sintático - Era esperado um IDENTIFICADOR
5. Próximo à linha 2: Erro sintático - Era esperado um PONTO_VIRGULA
6. Próximo à linha 2: Erro sintático - Era esperado um BASIC
7. Próximo à linha 12: Erro sintático - Era esperado um THEN
8. Próximo à linha 19: Esperava um IF, WHILE, REPEAT, READ, WRITE
9. Próximo à linha 23: Erro sintático - Era esperado um UNTIL
10. Próximo à linha 23: Esperava um IDENTIFICADOR, CONSTANTE OU ABRE PARENTESES
11. Próximo à linha 23: Erro sintático - Era esperado um FECHA_PARENTESES
12. Próximo à linha 23: Erro sintático - Era esperado um END

Como podê ser visto, devido à quantidade de argumentos que foram passados dentro do escopo do **declare**, mesmo estando ausente a palavra **routine**, o parser e o analisador léxico conseguiram se sincronizar novamente, ao contrário do que ocorreu no segundo código de testes.

Desta forma, ao corrigir estes erros, ao executar o algoritmo 20, foi gerado o conjunto de erros abaixo.

1. Próximo à linha 3: Erro sintático - Era esperado um BASIC
2. Próximo à linha 13: Erro sintático - Era esperado um THEN
3. Próximo à linha 20: Esperava um IF, WHILE, REPEAT, READ, WRITE
4. Próximo à linha 24: Erro sintático - Era esperado um UNTIL
5. Próximo à linha 24: Esperava um IDENTIFICADOR, CONSTANTE OU ABRE PARENTESES
6. Próximo à linha 24: Erro sintático - Era esperado um FECHA_PARENTESES

7. Próximo à linha 24: Erro sintático - Era esperado um END

Que após corrigí-lo e testar o algoritmo 21 não foi gerado mais nenhum erro sintático.

Algorithm 20 Teste 1 - Primeira correção

```
1: routine
2: declare
3: integer a, b, c, maior;
4: char outro;
5: begin
6: repeat
7: write("A: ");
8: read(a);
9: write("B: ");
10: read(b);
11: write("C: ");
12: read(c);
13: if ( (a¿b) and (a¿c) ) end
14: maior := a
15:
16: else
17: if (b¿c) then
18: maior := b;
19:
20: else
21: maior := c
22: end
23: end
24: write("Maior valor:");
25: write (maior);
26: write ("Outro? (S/N)");
27: read(outro);
28: until (outro = 'N' or outro = 'n')
29: end
```

Algorithm 21 Teste 5 - Correção Final

```
1: routine
2: declare
3: integer a, b, c, maior;
4: char outro;
5: begin
6: repeat
7: write(”A: ”);
8: read(a);
9: write(”B: ”);
10: read(b);
11: write(”C: ”);
12: read(c);
13: if ( (a¿b) and (a¿c) ) then
14: maior := a
15:
16: else
17: if (b¿c) then
18: maior := b
19:
20: else
21: maior := c
22: end
23: end;
24: write(”Maior valor:”);
25: write (maior);
26: write (”Outro? (S/N)”);
27: read(outro)
28: until (outro = ’N’ or outro = ’n’)
29: end
```

5.6 Teste 6 e 7

Algorithm 22 Teste 6

```
1: routine
2: declare
3: float raio, volume;
4: begin
5: raio := 0;
6: volume := 0;
7:
8: write("ESTE PROGRAMA MOSTRA O VOLUME DE UMA ESFERA
   DE ACORDO COM O RAIO.");
9: write("Digite o valor do RAIO: ");
10: read(raio); %Armazenado o valor na variavel raio%
11:
12: volume := 4 * 3.14 * raio * (raio/3); %Calculando o volume da esfera%
13:
14: write("O volume da esfera e: ");
15: write(volume);
16:
17: end
```

Algorithm 23 Teste 7

```
1: routine
2: declare
3: int dias, dia, meses, mes, anos, ano, calculo, diaatual, mesatual, anoatual;
4: begin
5:
6: diaatual := 3;
7: mesatual := 9;
8: anoatual := 2022;
9:
10: write("Insira os dados pessoais. ");
11: write("Dia de nascimento da pessoa: ");
12: read(dia);
13: write("Mês de nascimento da pessoa: ");
14: read(mes);
15: write("Ano de nascimento da pessoa: ");
16: read(ano);
17:
18: calculo := 365*anoatual + 30*mesatual + diaatual - 365*ano - 30*mes -
    dia;
19:
20: anos := calculo/365;
21:
22: meses := calculo/30;
23:
24: dias := calculo - (calculo*365*30);
25:
26: write("Você tem: ");
27: write(anos);
28: write(", ");
29: write(meses);
30: write(" e ");
31: write(dias);
32: write(" de idade.");
33:
34: end
```

O tanto sexto teste, Algoritmo 22, quanto o sétimo teste, Algoritmo 23, geraram o mesmo erro na ultima linha de código antecedendo o último **end**, que é ao colocar um ponto e virgula ao final da linha seria esperado um novo comando e não o **end**. Desta forma ao corrigí-los, os algoritmos 24 e 25 não geraram mais erros.

Algorithm 24 Teste 6 - Correção Final

```
1: routine
2: declare
3: float raio, volume;
4: begin
5: raio := 0;
6: volume := 0;
7:
8: write("ESTE PROGRAMA MOSTRA O VOLUME DE UMA ESFERA
   DE ACORDO COM O RAIO.");
9: write("Digite o valor do RAIO: ");
10: read(raio); %Armazenado o valor na variavel raio%
11:
12: volume := 4 * 3.14 * raio * (raio/3); %Calculando o volume da esfera%
13:
14: write("O volume da esfera e: ");
15: write(volume)
16:
17: end
```

Algorithm 25 Teste 7 - Correção Final

```
1: routine
2: declare
3: int dias, dia, meses, mes, anos, ano, calculo, diaatual, mesatual, anoatual;
4: begin
5:
6: diaatual := 3;
7: mesatual := 9;
8: anoatual := 2022;
9:
10: write("Insira os dados pessoais. ");
11: write("Dia de nascimento da pessoa: ");
12: read(dia);
13: write("Mês de nascimento da pessoa: ");
14: read(mes);
15: write("Ano de nascimento da pessoa: ");
16: read(ano);
17:
18: calculo := 365*anoatual + 30*mesatual + diaatual - 365*ano - 30*mes -
    dia;
19:
20: anos := calculo/365;
21:
22: meses := calculo/30;
23:
24: dias := calculo - (calculo*365*30);
25:
26: write("Você tem: ");
27: write(anos);
28: write(", ");
29: write(meses);
30: write(" e ");
31: write(dias);
32: write(" de idade.")
33:
34: end
```