

Compiladores  
2016/01

Trabalho Prático 01:  
Analisador Léxico e Tabela de Símbolos

Felipe Duarte, Gustavo Borba, Juan Lopes  
Belo Horizonte, 23 de abril de 2016

# Sumário

<b>1</b>	<b>Introdução</b>	<b>1</b>
1.1	A Linguagem Pas-c . . . . .	1
<b>2</b>	<b>Uso do compilador</b>	<b>2</b>
2.1	Compilando o Compilador Pas-c . . . . .	2
2.2	Compilando um programa em Pas-c . . . . .	2
<b>3</b>	<b>A Implementação do Compilador</b>	<b>3</b>
3.1	A abordagem utilizada na implementação . . . . .	3
3.2	A Estrutura do Projeto . . . . .	3
3.3	Principais Classes da Aplicação . . . . .	3
<b>4</b>	<b>Resultados dos testes especificados</b>	<b>4</b>
4.1	Teste 1 . . . . .	4
4.2	Teste 2 . . . . .	4
4.3	Teste 3 . . . . .	4
4.4	Teste 4 . . . . .	5
4.5	Teste 5 . . . . .	5
4.6	Teste 6 . . . . .	5
4.7	Teste 7 . . . . .	6
4.8	Teste 8 . . . . .	6

## 1 Introdução

O programa pas-c é um projeto desenvolvido na aula de Compiladores do Curso de Engenharia da Computação do CEFET-MG. Pas-c é a linguagem homônima, definida para esse compilador, cuja semântica dos comandos e expressões é a tradicional de linguagens como Pascal e C (daí seu nome).

### 1.1 A Linguagem Pas-c

Definimos a gramática da linguagem:

```

1 | program ::= [ var decl-list ] begin stmt-list end
2 | decl-list ::= decl ";" { decl ";" }
3 | decl ::= ident-list is type
4 | ident-list ::= identifier { "," identifier }
5 | type ::= int | string
6 | stmt-list ::= stmt ";" { stmt ";" }
7 | stmt ::= assign-stmt | if-stmt | do-stmt | read-stmt | write-stmt
8 | assign-stmt ::= identifier ":"=" simple_expr
9 | if-stmt ::= if condition then stmt-list end
10 |           | if condition then stmt-list else stmt-list end
11 | condition ::= expression
12 | do-stmt ::= do stmt-list stmt-suffix
13 | stmt-suffix ::= while condition
14 | read-stmt ::= in "(" identifier ")"
15 | write-stmt ::= out "(" writable ")"
16 | writable ::= simple_expr
17 | expression ::= simple_expr | simple_expr relop simple_expr
18 | simple_expr ::= term | simple_expr addop term
19 | term ::= factor-a | term mulop factor-a
20 | fator-a ::= factor | not factor | "-" factor
21 | factor ::= identifier | constant | "(" expression ")"
22 | relop ::= "=" | ">" | ">=" | "<" | "<=" | "<>"
23 | addop ::= "+" | "-" | or
24 | mulop ::= "*" | "/" | and
25 | constant ::= integer_const | literal
26 | integer_const ::= nozero { digit } | "0"
27 | literal ::= " { caractere } "
28 | identifier ::= ( letter ) { letter | digit }
29 |           | "-" ( letter | digit ) { letter | digit }
30 | letter ::= [A-Za-z]
31 | digit ::= [0-9]
32 | nozero ::= [1-9]
33 | caractere ::= um dos 256 caracteres do conjunto ASCII,
34 |               exceto "{", "}" e quebra de linha
35
```

## 2 Uso do compilador

### 2.1 Compilando o Compilador Pas-c

Para compilar o seu código (neste momento, apenas como analisador léxico), basta ter um compilador G++ versão 4.8 ou superior com cmake instalado em sua máquina.

Na pasta do projeto execute o comando:

```
1 | make
```

Esse comando deverá gerar a seguinte seqüência de comandos:

```
1 | g++ -include -c -Wall -g -DRUN_TESTS src/Scanner.cpp -o src/Scanner.o
2 | g++ -include -c -Wall -g -DRUN_TESTS src/TestCase.cpp -o src/TestCase.o
3 | g++ -include -c -Wall -g -DRUN_TESTS src/Token.cpp -o src/Token.o
4 | g++ -include -c -Wall -g -DRUN_TESTS src/main.cpp -o src/main.o
5 | g++ src/Scanner.o src/TestCase.o src/Token.o src/main.o -o pasc
```

Pronto! Agora o executável **pasc** está pronto para compilar seus programas em Pas-c!

### 2.2 Compilando um programa em Pas-c

Para executar o compilador, basta executar o comando:

```
pasc caminho_para_seu_codigo.pasc
```

## 3 A Implementação do Compilador

### 3.1 A abordagem utilizada na implementação

O compilador **pas-c** utiliza um analisador léxico recursivo, e quem vai saber explicar essas parada toda é o japonês, eu nem estudei pra segunda prova ainda.

### 3.2 A Estrutura do Projeto

O projeto segue a estrutura básica de todo projeto em C++, contendo uma pasta **include** e, dentro desta, separados por módulos, os headers (.h) com as declarações das classes. A pasta **src** contém a implementação das classes, bem como a função principal **main**:

```
include/ -- Pasta para os headers
include/frontend -- pasta contendo as definições para os analisadores.
include/backend -- ainda por fazer.
include/test -- Pasta com os headers referentes aos testes unitários
src/ -- Implementação das classes
tests/ -- Pasta com a implementação dos testes unitários
```

### 3.3 Principais Classes da Aplicação

- **Token:** A classe Token descreve a unidade lógica mais básica do compilador. Ela apresenta apenas um valor, em formato de cadeia de caracteres, e seu **Tipo**, a ser elucidado no próximo item.
- **TokenType:** TokenType trata-se apenas de um enum, que define, através de um bitmap, a lista de tokens reconhecidos pelo compilador, bem como um tipo UNKNOWN, para todo e qualquer token que não corresponder a nenhuma definição da linguagem.
- **Scanner:** O analisador léxico está concentrado basicamente na classe Scanner. Essa classe contém os métodos **getNumerical**, **getLiteral** e **getOperator**, que são responsáveis por captar os tokens, bem como o método **getString**. Este último é necessário por que uma vez que se abre um caracter delimitador de string, espaços em brancos passam a ser caracteres significativos. Também estão definidos nessa classe os contadores de linhas e colunas do código-fonte.

## 4 Resultados dos testes especificados

### 4.1 Teste 1

Código fonte testado:

```
1  var
2    a, b, c is int;
3    result is int
4  begin
5    in (a);
6    in (c);
7    b := 10;
8    result := (a * c)/(b + 5 - 345);
9    out(result);
10 end
```

Saída encontrada:

### 4.2 Teste 2

Código fonte testado:

```
1    a, _ is int;
2    b is int;
3    nome is string;
4  begin
5    in (a);
6    in (nome);
7    b := a * a;
8    b := b + a/2 * (3 + 5);
9    out (nome);
10   out(b);
11 end.
```

Saída encontrada:

### 4.3 Teste 3

Código fonte testado:

```
1  var
2    _cont is int;
3    media, altura, soma_ is int;
4  begin
5    _cont := 5;
6    soma = 0;
7
8    do
9      write({Altura: });
10     in (altura);
11     soma := soma altura;
12     _cont := _cont - 1;
13   while(_cont);
```

```
14 |  
15 |   out({Media: });  
16 |   out (soma / qtd);  
17 | end
```

Saída encontrada:

#### 4.4 Teste 4

Código fonte testado:

```
1 | var  
2 |   i, j, k, @total, lsoma is int;  
3 |  
4 | begin  
5 |   i := 4 * (5-3 * 50 / 10;  
6 |   j := i * 10;  
7 |   k := i * j / k;  
8 |   k := 4 + a $;  
9 |   out(i);  
10 |  out(j);  
11 |  out(k);  
12 | end
```

Saída encontrada:

#### 4.5 Teste 5

Código fonte testado:

```
1 | var  
2 |   j, k is int;  
3 |   a, j real;  
4 |  
5 | begin  
6 |   read(j);  
7 |   read(K);  
8 |  
9 |   if (k <> 0)  
10 |     result := j/k  
11 |   else  
12 |     result := 0  
13 |   end;  
14 |  
15 |   out(result);  
16 | end
```

Saída encontrada:

## 4.6 Teste 6

Código fonte testado:

```
1  var
2    a, b, c, maior is int;
3    nomecompletodoalunodeposgraduacao is string;
4
5  start
6    read(a);
7    read(b);
8    read(c);
9
10   maior := 0;
11   if ( a>b and a>c ) then
12     maior := a;
13   else
14     if (b>c) then
15       maior := b;
16     else
17       maior := c;
18     end
19   end
20
21   Out({Maior idade: });
22   out(maior);
23 end
```

Saída encontrada:

## 4.7 Teste 7

Código fonte testado:

```
1  var
2    a, b, c is int;
3    result is int
4  begin
5    in (a);
6    in (c);
7    b := 10;
8    result := (a * c)/(b + 5 - 345);
9    out(result);
10 end
```

Saída encontrada:

## 4.8 Teste 8

Código fonte testado:

```
1  var
2    a, b, c is int;
3    result is int
```



```
4 | begin
5 |   in (a);
6 |   in (c);
7 |   b := 10;
8 |   result := (a * c)/(b + 5 - 345);
9 |   out(result);
10 | end
```

Saída encontrada: