



Departamento de Informática e Estatística  
Universidade Federal de Santa Catarina  
Brasil

## **Analizador Léxico**

Leandro Hideki Aihara  
Thiago Martendal Salvador  
Pablo Daniel Riveros Strapasson

**Professor**  
Alvaro Junior Pereira Franco

Florianópolis  
Julho de 2021

# Sumário

<b>1 Identificação dos Tokens</b>	<b>3</b>
<b>2 Produção das definições regulares para cada token</b>	<b>4</b>
<b>3 Diagramas de Transição</b>	<b>5</b>
<b>4 Tabela de Símbolos</b>	<b>10</b>
Definições no código:	10
<b>5 Sobre a Ferramenta Flex</b>	<b>11</b>
<b>6 Como usar o Analisador Léxico</b>	<b>12</b>
<b>7 Compilando o Projeto</b>	<b>13</b>
<b>8 Considerações Finais</b>	<b>13</b>

# 1 Identificação dos Tokens

Token	Identificador	Token	Identificador
Ident	ID = 1	Semicolon	PV = 24
Int_constant	ICT = 2	New	NEW = 25
Float_constant	FCT = 3	Def	DF = 26
String_constant	SCT = 4	Read	RD = 27
Plus	ADD = 5	Print	PR = 28
Minus	SUB = 6	Null	NL = 29
Multiply	MUL = 7	Int	INT = 30
Divide	DIV = 8	Float	FLT = 31
Percentage	PRC = 9	String	STR = 32
Assign	ATR = 10	If	IF = 33
Greater	MAR = 11	Else	ELS = 34
Lower	MER = 12	For	FOR = 35
Equal	CMP = 13	Break	BRK = 36
Greater_or_equal	MAI = 14	Return	RET = 37
Lower_or_equal	MEI = 15	Error	ERR = 38
Different	DIF = 16		
Left_parenthesis	P1 = 17		
Right_parenthesis	P2 = 18		
Left_brace	CV1 = 19		
Right_brace	CV2 = 20		
Left_bracket	CL1 = 21		
Right_bracket	CL2 = 22		
Comma	VI = 23		

## 2 Produção das definições regulares para cada token

Token	ER	Token	ER
Ident	[a-zA-Z][a-zA-Z0-9]*	Semicolon	;
Int_constant	[[digit:]]+	New	new
Float_constant	[[digit:]]*\.[[:digit:]]*	Def	def
String_constant	\"([^\"] \\\" \\\\)*[^\"]?\"	Read	read
Plus	+	Print	print
Minus	-	Null	null
Multiply	*	Int	int
Divide	/	Float	float
Percentage	%	String	string
Assign	=	If	if
Greater	>	Else	else
Lower	<	For	for
Equal	==	Break	break
Greater_or_equal	>=	Return	return
Lower_or_equal	<=	Error*	.
Different	!=		
Left_parenthesis	(		
Right_parenthesis	)		
Left_brace	{		
Right_brace	}		
Left_bracket	[		
Right_bracket	]		
Comma	,		

\* Caso de erro para qualquer lexema não identificado

### 3 Diagramas de Transição

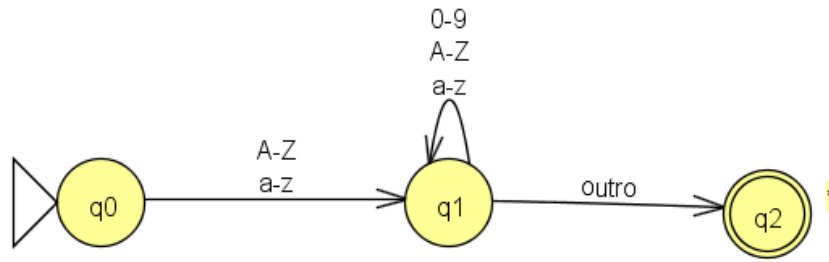


Figura 1. Token Ident

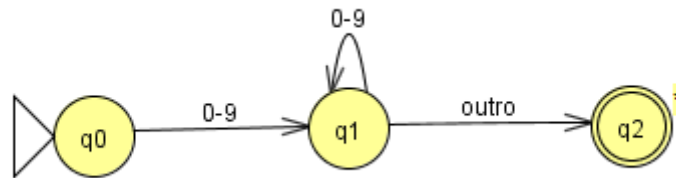


Figura 2. Token Int\_constant

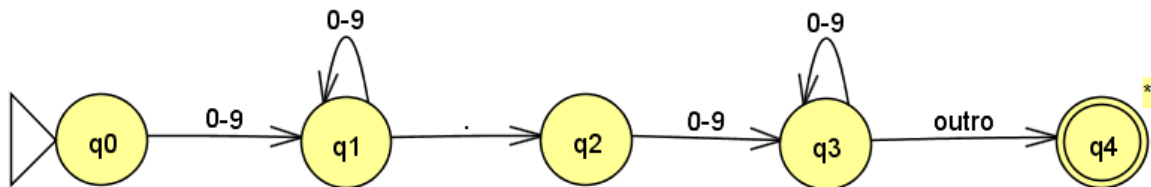


Figura 3. Token Float\_constant

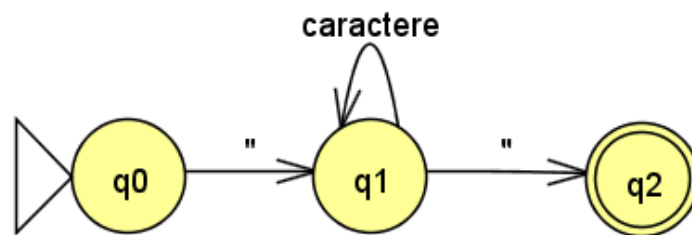


Figura 4. Token String\_constant

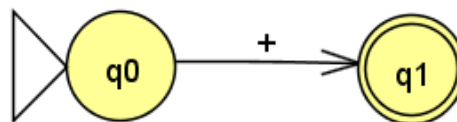


Figura 5. Token Plus

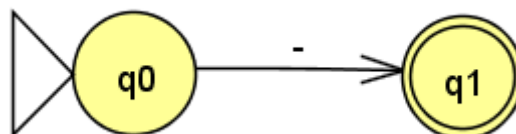


Figura 6. Token Minus



Figura 7. Token Multiply

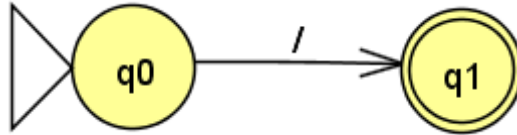


Figura 8. Token Divide

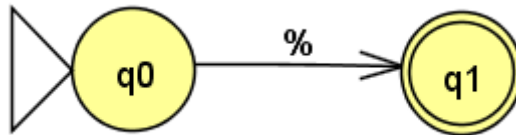


Figura 9. Token Percentage

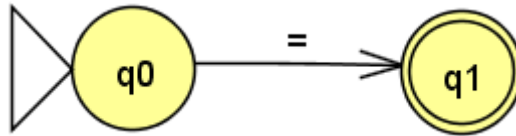


Figura 10. Token Assign

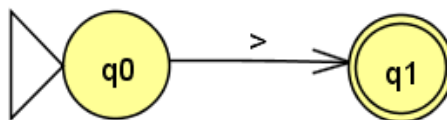


Figura 11. Token Greater



Figura 12. Token Lower

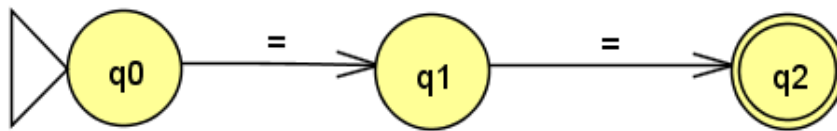


Figura 13. Token Equal

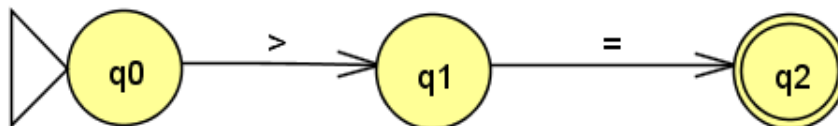


Figura 14. Token Greater\_or\_equal

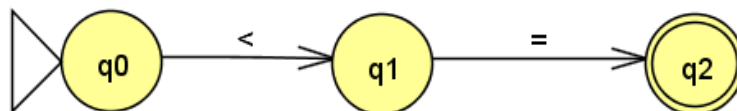


Figura 15. Token Lower\_or\_equal

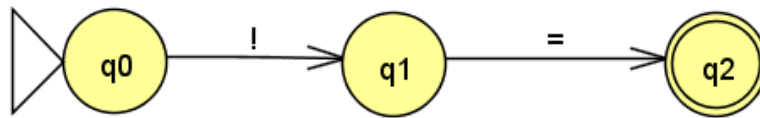


Figura 16. Token Different

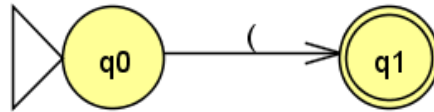


Figura 17. Token Left\_parenthesis

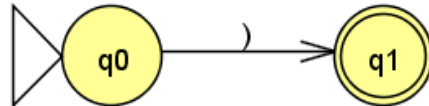


Figura 18. Token Right\_parenthesis

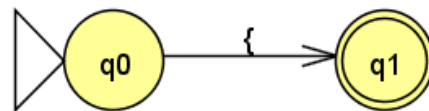


Figura 19. Token Left\_brace

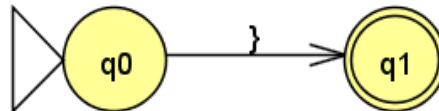


Figura 20. Token Right\_brace

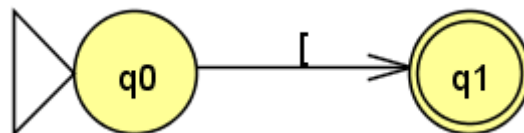


Figura 21. Token Left\_bracket

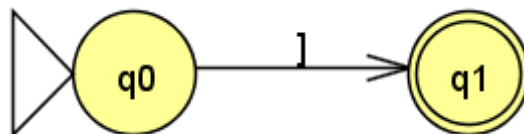


Figura 22. Token Right\_bracket

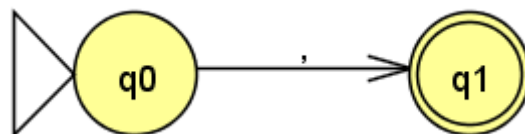


Figura 23. Token Comma

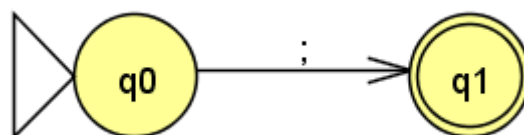


Figura 24. Token Semicolon

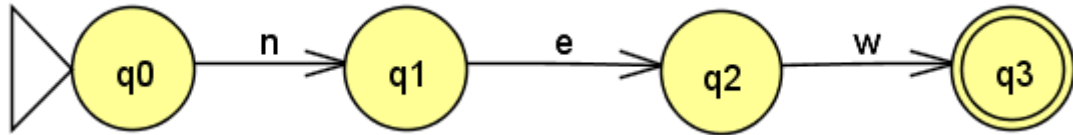


Figura 25. Token New

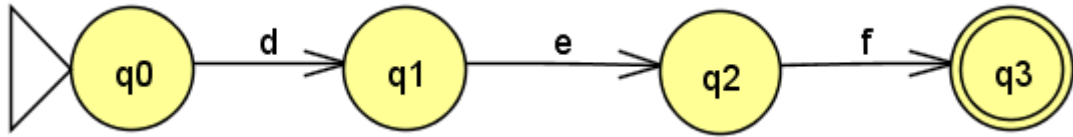


Figura 26. Token Def

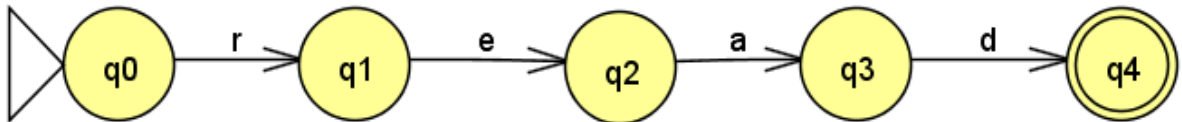


Figura 27. Token Read

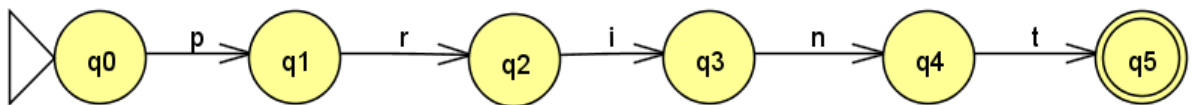


Figura 28. Token Print

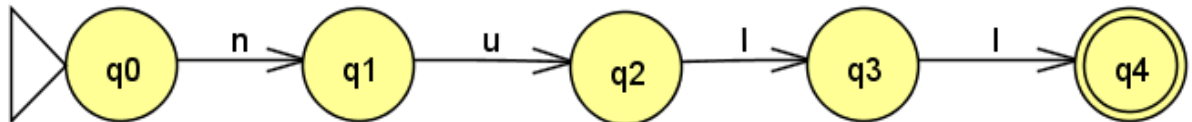


Figura 29. Token Null

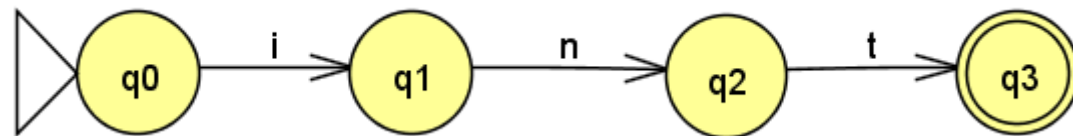


Figura 30. Token Int

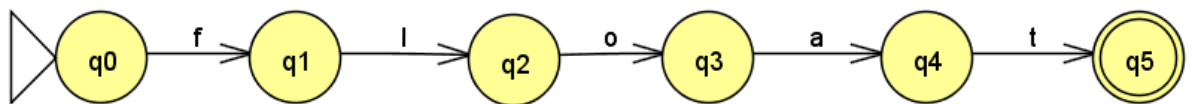


Figura 31. Token Float



Figura 32. Token String

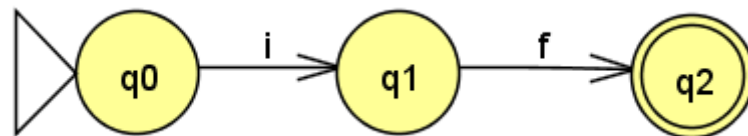


Figura 33. Token If





Figura 34. Token Else

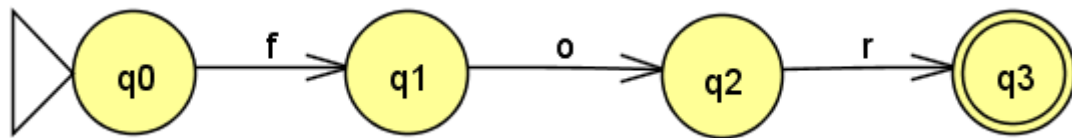


Figura 35. Token For

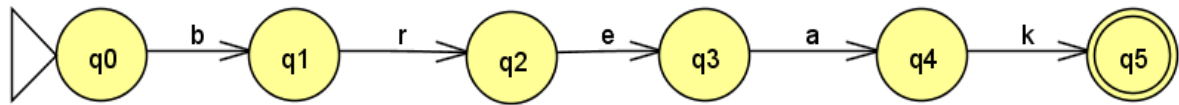


Figura 36. Token Break

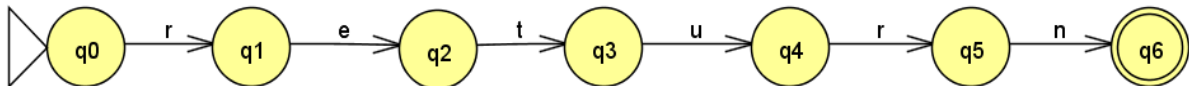


Figura 37. Token Return

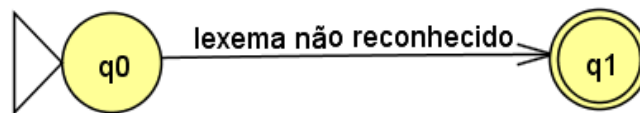


Figura 38. Token Error

## 4 Tabela de Símbolos

A tabela de símbolos é representada por uma tabela hash realizada pela biblioteca map do conjunto STL do C++.

O objetivo da tabela de símbolos é armazenar os identificadores encontrados no código, com as posições em linha e coluna onde ocorre.

A tabela é composta por um identificador representado por uma string que compõe o lexema do identificador, e uma estrutura de atributos que armazena um conjunto com as posições onde o lexema ocorre no código.

**A tabela de símbolos armazena apenas os identificadores que nomeiam variáveis e funções.**

Definições no código:

```
std::map<std::string, Atributo> tabelaSimbolos; // A Tabela de  
símbolos
```

A estrutura Atributo:

```
typedef struct Atributo {  
    std::set<Posicao> pos; // Número das linhas onde ocorre o  
    identificador  
} Atributo;
```

A estrutura Posicao:

```
typedef struct Posicao {  
    int linha; // Linha atual  
    int coluna; // Coluna atual  
} Posicao;
```

## 5 Sobre a Ferramenta Flex

Para a realização deste trabalho, foi utilizada uma ferramenta para a construção do analisador léxico, que se trata da ferramenta Flex.

O Flex se trata de um programa que constrói o analisador a partir de definições léxicas descritas em um arquivo de extensão ‘.l’, onde estas definições são convertidas para um programa de alto nível em C ou C++ que associa lexemas a tokens definidos pelo programador no arquivo de definições.

O Flex também disponibiliza opções de configuração que podem ser chamadas no arquivo de definições para guiar o compilador na geração do analisador. Estas opções podem ser declaradas com `%option nome_da_opção`.

As opções são declaradas fora dos escopos de definição.

A documentação deve ser consultada para analisar as opções disponíveis. O uso das opções não é obrigatório.

### Sintaxe do Flex

O arquivo de definições segue o formato:

```
% {  
    // Escopo para inclusão de arquivos e definição de funções auxiliares  
% }  
  
%option // Normalmente as opções são declaradas neste espaço  
  
%%  
    // Escopo para a declaração de lexemas com seus respectivos tokens  
%%  
  
// Ao restante do arquivo podem ser implementadas funções auxiliares
```

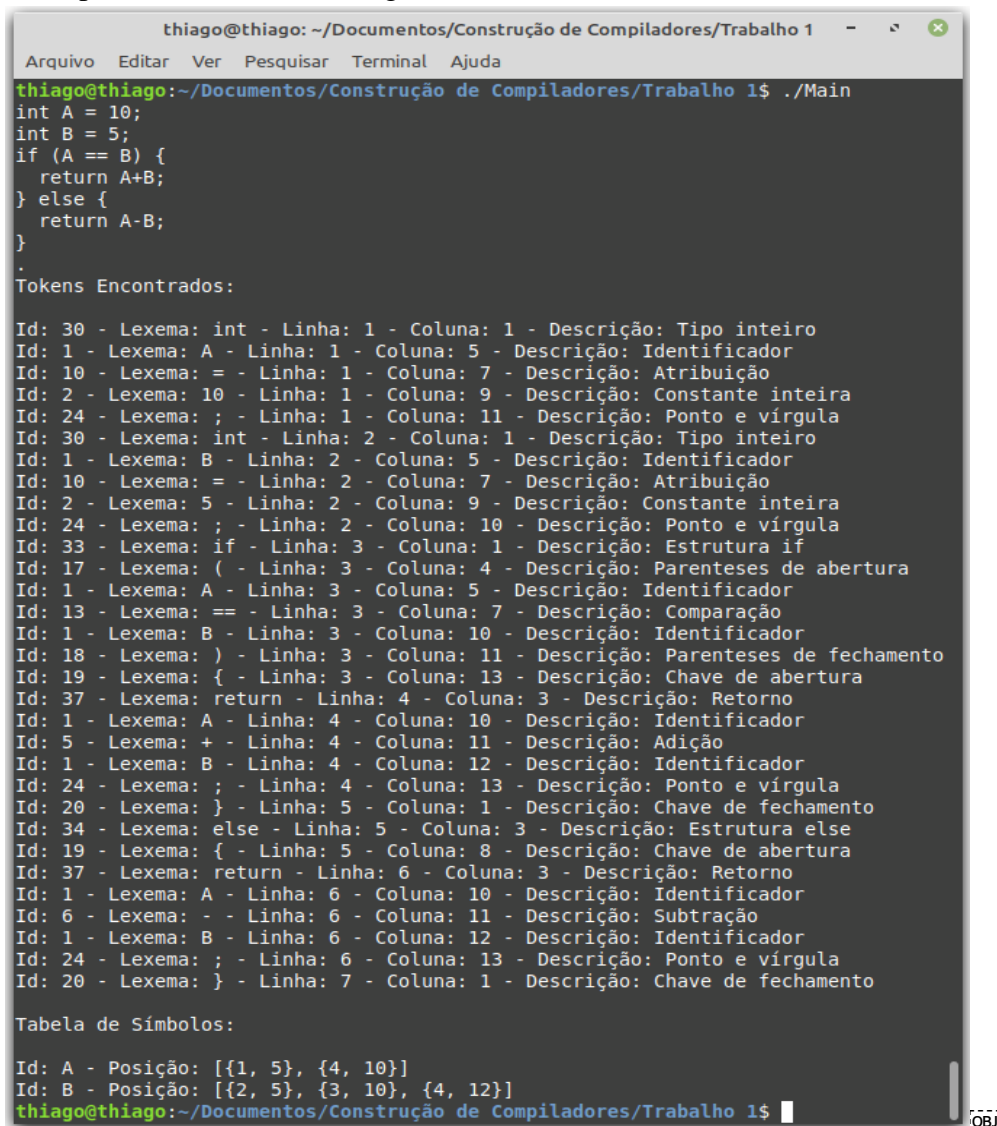
## 6 Como usar o Analisador Léxico

Para executar o programa, basta digitar **./Main** e o nome do programa que se deseja analisar. O nome do programa deve conter a extensão lcc.

Exemplo: **./Main programa.lcc**

Caso seja digitado apenas o comando **./Main**, será aberta uma área para digitação de código no console. Quando esta área é acessada, pode-se digitar quaisquer comandos da linguagem, que serão analisados linha por linha. Quando se deseja encerrar a área de digitação, basta pular uma linha com enter, digitar apenas um ponto '.' e então a área de entrada de código via console é encerrada, e logo em seguida são exibidos os tokens e a tabela de símbolos. O ponto deve ser digitado sem aspas.

Exemplo de uma área de código diretamente no console:



```
thiago@thiago: ~/Documentos/Construção de Compiladores/Trabalho 1
Arquivo Editar Ver Pesquisar Terminal Ajuda
thiago@thiago:~/Documentos/Construção de Compiladores/Trabalho 1$ ./Main
int A = 10;
int B = 5;
if (A == B) {
    return A+B;
} else {
    return A-B;
}
.
Tokens Encontrados:
Id: 30 - Lexema: int - Linha: 1 - Coluna: 1 - Descrição: Tipo inteiro
Id: 1 - Lexema: A - Linha: 1 - Coluna: 5 - Descrição: Identificador
Id: 10 - Lexema: = - Linha: 1 - Coluna: 7 - Descrição: Atribuição
Id: 2 - Lexema: 10 - Linha: 1 - Coluna: 9 - Descrição: Constante inteira
Id: 24 - Lexema: ; - Linha: 1 - Coluna: 11 - Descrição: Ponto e vírgula
Id: 30 - Lexema: int - Linha: 2 - Coluna: 1 - Descrição: Tipo inteiro
Id: 1 - Lexema: B - Linha: 2 - Coluna: 5 - Descrição: Identificador
Id: 10 - Lexema: = - Linha: 2 - Coluna: 7 - Descrição: Atribuição
Id: 2 - Lexema: 5 - Linha: 2 - Coluna: 9 - Descrição: Constante inteira
Id: 24 - Lexema: ; - Linha: 2 - Coluna: 10 - Descrição: Ponto e vírgula
Id: 33 - Lexema: if - Linha: 3 - Coluna: 1 - Descrição: Estrutura if
Id: 17 - Lexema: ( - Linha: 3 - Coluna: 4 - Descrição: Parenteses de abertura
Id: 1 - Lexema: A - Linha: 3 - Coluna: 5 - Descrição: Identificador
Id: 13 - Lexema: == - Linha: 3 - Coluna: 7 - Descrição: Comparação
Id: 1 - Lexema: B - Linha: 3 - Coluna: 10 - Descrição: Identificador
Id: 18 - Lexema: ) - Linha: 3 - Coluna: 11 - Descrição: Parenteses de fechamento
Id: 19 - Lexema: { - Linha: 3 - Coluna: 13 - Descrição: Chave de abertura
Id: 37 - Lexema: return - Linha: 4 - Coluna: 3 - Descrição: Retorno
Id: 1 - Lexema: A - Linha: 4 - Coluna: 10 - Descrição: Identificador
Id: 5 - Lexema: + - Linha: 4 - Coluna: 11 - Descrição: Adição
Id: 1 - Lexema: B - Linha: 4 - Coluna: 12 - Descrição: Identificador
Id: 24 - Lexema: ; - Linha: 4 - Coluna: 13 - Descrição: Ponto e vírgula
Id: 20 - Lexema: } - Linha: 5 - Coluna: 1 - Descrição: Chave de fechamento
Id: 34 - Lexema: else - Linha: 5 - Coluna: 3 - Descrição: Estrutura else
Id: 19 - Lexema: { - Linha: 5 - Coluna: 8 - Descrição: Chave de abertura
Id: 37 - Lexema: return - Linha: 6 - Coluna: 3 - Descrição: Retorno
Id: 1 - Lexema: A - Linha: 6 - Coluna: 10 - Descrição: Identificador
Id: 6 - Lexema: - - Linha: 6 - Coluna: 11 - Descrição: Subtração
Id: 1 - Lexema: B - Linha: 6 - Coluna: 12 - Descrição: Identificador
Id: 24 - Lexema: ; - Linha: 6 - Coluna: 13 - Descrição: Ponto e vírgula
Id: 20 - Lexema: } - Linha: 7 - Coluna: 1 - Descrição: Chave de fechamento

Tabela de Símbolos:
Id: A - Posição: [{1, 5}, {4, 10}]
Id: B - Posição: [{2, 5}, {3, 10}, {4, 12}]
thiago@thiago:~/Documentos/Construção de Compiladores/Trabalho 1$
```

Na oitava linha, antes de se exibir os tokens encontrados, está o caractere ‘.’ que demarca o fim da seção. O ponto serve simplesmente para demarcar o fim da execução de entrada direta, sendo que este não faz parte dos tokens da linguagem, e é disponível apenas na versão de leitura do console (um ponto encontrado em um arquivo lcc pode ser tratado como um erro léxico se este não fizer parte de uma string).

## 7 Compilando o Projeto

Para a etapa de compilação, existe um arquivo makefile com os comandos de compilação do Flex e do C++ para gerar o programa.

O comando para acessar o makefile é: **make compilar**

Os comandos de compilação registrados no makefile são:

1. **flex -+ lexico.l**
2. **g++ main.cpp entrada.cpp analise\_lexica.cpp lex.yy.cc -lfl -o Main**

Relembrando, para executar: **./Main nome\_programa.lcc**

## 8 Considerações Finais

Após a análise léxica ser concluída, é exibida a lista de tokens que foram reconhecidos na linguagem. Lexemas que não se associam com algum padrão léxico são atribuídos a um token de erro.

Caso ocorra um erro léxico no código, a tabela de símbolos não é exibida, e o token da linha com erro pode ser encontrado na lista de tokens que foi exibida.

Caso seja necessário instalar o flex, basta usar o comando do makefile: **make instalar-flex**