Especificação da Linguagem Regular

Visão Geral

Este documento define formalmente a linguagem regular reconhecida pelo analisador léxico. A linguagem combina elementos lexicais inspirados em francês, criando uma sintaxe única para construções de programação.

Alfabeto da Linguagem

Caracteres Básicos

- **Letras**: `a-z`, `A-Z`
- **Dígitos**: `0-9`
- **Caracteres especiais**: `+`, `-`, `X`, `/`, `:`, `Λ`, `V`, `;`, `,`, `(`, `)`, `{`, `}`, `_`, `.`, `*`, `!`
- **Espaços em branco**: espaço, tab (`\t`), carriage return (`\r`), nova linha (`\n`)

Definições de Expressões Regulares

Expressões Básicas

```
DIGITO = [0-9]
PRIMEIRA_LETRA = [a-zA-UW-Z]
LETRA = [a-zA-Z]
```

Identificadores

```
ID = {PRIMEIRA_LETRA}({LETRA}|{DIGITO}|_)*
```

Descrição: Identificadores começam com uma letra (excluindo V e X que são usados como operadores) seguida de zero ou mais letras, dígitos ou underscores.

Exemplos válidos:

- `variavel`
- `contador_1`
- `minhaVar`
- `A123`

Exemplos inválidos:

- `123abc` (não pode começar com dígito)
- `_var` (não pode começar com underscore)

Literais Numéricos

Números Inteiros

```
INT = {DIGITO}+
```

Descrição: Sequência de um ou mais dígitos.

Exemplos: `0`, `42`, `1234`, `9999`

Números de Ponto Flutuante

```
FLOAT = {DIGITO}+"."{DIGITO}+
```

Descrição: Sequência de dígitos, seguida de ponto, seguida de sequência de dígitos.

Exemplos: `3.14`, `0.5`, `123.456`, `999.001`

Tokens da Linguagem

Palavras-chave (Keywords)

Token	Palavra-chave	Equivalente	Descrição
`KEYWORD_IF`	`siu`	if	Condicional se
`KEYWORD_ELSE`	`autre`	else	Senão
`KEYWORD_ELIF`	`sinon siu`	elif	Senão se
`KEYWORD_SWITCH`	`changer`	switch	Seleção múltipla

Token	Palavra-chave	Equivalente	Descrição
KEYWORD_CASE	cas	case	Caso
KEYWORD_DEFAULT	defaut	default	Padrão
KEYWORD_BREAK	casser	break	Quebrar
KEYWORD_CONTINUE	continuer	continue	Continuar
KEYWORD_RETURN	retour	return	Retornar
KEYWORD_DO	faire	do	Fazer
KEYWORD_WHILE	dembele	while	Enquanto
KEYWORD_FOR	mbappe	for	Para

Operadores

Operadores Aritméticos

Token	Símbolo	Descrição
OP_SOMA	+	Adição
OP_SUB	-	Subtração
OP_MULT	X	Multiplicação
OP_DIV	/	Divisão

Operadores de Atribuição

Token	Símbolo	Descrição
OP_ATRIB	:	Atribuição

Operadores Relacionais

Token	Símbolo	Descrição
OP_MENOR	Λ	Menor que
OP_MAIOR	V	Maior que
OP_MENOR_IGUAL	V/	Menor ou igual
OP_MAIOR_IGUAL	/Λ	Menor ou igual
OP_IGUAL	::	Igualdade

Token	Símbolo	Descrição
OP_DIFERENTE	ney:	Diferente

Operadores Lógicos

Token	Símbolo	Descrição
NEGACAO	ney	Negação lógica

Delimitadores

Token	Símbolo	Descrição
DELIM_PONTO_VIRGULA	;	Fim de instrução
DELIM_VIRGULA	,	Separador
DELIM_ABRE_PARENTESES	(	Abre parênteses
DELIM_FECHA_PARENTESES	)	Fecha parênteses
DELIM_ABRE_CHAVES	{	Abre bloco
DELIM_FECHA_CHAVES	}	Fecha bloco

Comentários

Comentário de Linha Única

```
COMENTARIO_LINHA = "//".*
```

Exemplo: `// Este é um comentário`

Comentário de Múltiplas Linhas

```
COMENTARIO_BLOCO = "/*"([^*]|\*+[^*/])*\*+"*/"
```

Exemplo:

```
/* Este é um comentário
   de múltiplas linhas */
```

Espaços em Branco

```
ESPACOS = [ \t\r]+
NOVA_LINHA = \n
```

Descrição: Espaços, tabs e carriage returns são ignorados. Novas linhas incrementam o contador de linha.

Precedência de Tokens

A ordem de precedência das regras lexicais (da mais alta para a mais baixa):

1. **Palavras-chave** (reconhecidas como strings literais)
2. **Identificadores** (padrão mais geral)
3. **Números de ponto flutuante** (mais específico)
4. **Números inteiros** (menos específico)
5. **Operadores de múltiplos caracteres** (`::` , `ney:` , `V/` , `sinon siu`)
6. **Operadores de caractere único**
7. **Delimitadores**
8. **Comentários**
9. **Espaços em branco**
10. **Caractere de erro** (qualquer outro caractere)

Regras Lexicais Especiais

Tratamento de Erros

```
ERRO_LEXICO = .
```

Qualquer caractere não reconhecido pelas regras anteriores gera um erro léxico.

Contagem de Linhas

- Cada `\n` incrementa o contador de linha
- Usado para reportar a localização de tokens e erros

Autômato Finito

Estados Principais

- **S0**: Estado inicial
- **S1**: Reconhecendo identificador/palavra-chave
- **S2**: Reconhecendo número inteiro
- **S3**: Reconhecendo ponto em número decimal
- **S4**: Reconhecendo número decimal
- **S5**: Reconhecendo operador
- **S6**: Reconhecendo comentário de linha
- **S7**: Reconhecendo comentário de bloco
- **SE**: Estado de erro

Transições de Estado

```
S0 --[letra]--> S1
S0 --[dígito]--> S2
S0 --[operador]--> S5
S0 --[delimitador]--> ACEITA
S0 --[/]--> S6 ou S5
S0 --[espaço]--> S0 (ignora)
S0 --[qualquer outro]--> SE

S1 --[letra|dígito|_]--> S1
S1 --[outros]--> ACEITA_ID

S2 --[dígito]--> S2
S2 --[.]--> S3
S2 --[outros]--> ACEITA_INT

S3 --[dígito]--> S4
S3 --[outros]--> ERRO

S4 --[dígito]--> S4
S4 --[outros]--> ACEITA_FLOAT
```

Exemplos de Análise

Exemplo 1: Expressão Simples

Entrada: `x : 42 + y`

Tokens gerados:

1. `IDENTIFICADOR -> 'x'`
2. `OP_ATRIB -> ':'`
3. `NUMERO_INT -> '42'`
4. `OP_SOMA -> '+'`
5. `IDENTIFICADOR -> 'y'`

Exemplo 2: Estrutura Condicional

Entrada: `siu (a Λ b) { retour; }`

Tokens gerados:

1. `KEYWORD_IF -> 'siu'`
2. `DELIM_ABRE_PARENTESES -> '('`
3. `IDENTIFICADOR -> 'a'`
4. `OP_MENOR -> 'Λ'`
5. `IDENTIFICADOR -> 'b'`
6. `DELIM_FECHA_PARENTESES -> ')'`
7. `DELIM_ABRE_CHAVES -> '{'`
8. `KEYWORD_RETURN -> 'retour'`
9. `DELIM_PONTO_VIRGULA -> ';'`
10. `DELIM_FECHA_CHAVES -> '}'`

Exemplo 2: Estrutura Condicional

Entrada: `siu (a Λ b) { retour; }`

Tokens gerados:

1. `KEYWORD_IF -> 'siu'`
2. `DELIM_ABRE_PARENTESES -> '('`
3. `IDENTIFICADOR -> 'a'`
4. `OP_MENOR -> 'Λ'`
5. `IDENTIFICADOR -> 'b'`
6. `DELIM_FECHA_PARENTESES -> ')'`
7. `DELIM_ABRE_CHAVES -> '{'`
8. `KEYWORD_RETURN -> 'retour'`
9. `DELIM_PONTO_VIRGULA -> ';'`
10. `DELIM_FECHA_CHAVES -> '}'`

Exemplo 3: Laços de Repetição

Entrada: `mbappe (i : 0; i Λ 10; i : i + 1) { dembele (condicao) { valor : i X 2; } }`

Tokens gerados:

1. `KEYWORD_FOR -> 'mbappe'`
2. `DELIM_ABRE_PARENTESES -> '('`
3. `IDENTIFICADOR -> 'i'`
4. `OP_ATRIB -> ':'`
5. `NUMERO_INT -> '0'`
6. `DELIM_PONTO_VIRGULA -> ';'`
7. `IDENTIFICADOR -> 'i'`
8. `OP_MENOR -> 'Λ'`
9. `NUMERO_INT -> '10'`
10. `DELIM_PONTO_VIRGULA -> ';'`
11. `IDENTIFICADOR -> 'i'`
12. `OP_ATRIB -> ':'`
13. `IDENTIFICADOR -> 'i'`
14. `OP_SOMA -> '+'`
15. `NUMERO_INT -> '1'`
16. `DELIM_FECHA_PARENTESES -> ')'`
17. `DELIM_ABRE_CHAVES -> '{'`
18. `KEYWORD_WHILE -> 'dembele'`
19. `DELIM_ABRE_PARENTESES -> '('`
20. `IDENTIFICADOR -> 'condicao'`
21. `DELIM_FECHA_PARENTESES -> ')'`
22. `DELIM_ABRE_CHAVES -> '{'`
23. `IDENTIFICADOR -> 'valor'`
24. `OP_ATRIB -> ':'`
25. `IDENTIFICADOR -> 'i'`
26. `OP_MULT -> 'X'`
27. `NUMERO_INT -> '2'`
28. `DELIM_PONTO_VIRGULA -> ';'`
29. `DELIM_FECHA_CHAVES -> '}'`
30. `DELIM_FECHA_CHAVES -> '}'`

Exemplo 4: Número Decimal

Entrada: `pi : 3.14159`

Tokens gerados:

1. `IDENTIFICADOR -> 'pi'`
2. `OP_ATRIB -> ':'`
3. `NUMERO_FLOAT -> '3.14159'`