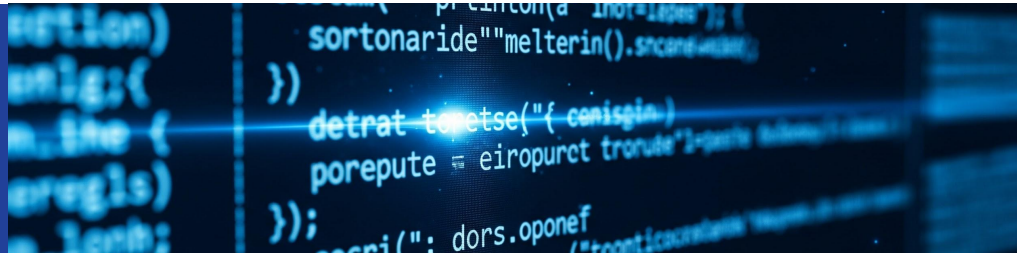
Compiladores - Analisador Léxico em C++

UNISINOS

Nossa sala de aula é o mundo.

Suelen Vicentini Fraga

Objetivo



- Explicar o papel do analisador léxico na compilação.
- Mostrar a implementação em C++.
- Reconhecer os principais tokens de uma mini linguagem.

UNISINOS

Tokens da linguagem

Lista com bullets:

- Identificadores
- Palavras-chave
- Números (inteiros e decimais)
- Strings
- Operadores (+ - * / = == != <= >= && ||)
- Pontuação (; , ( ) { } [ ])
- Comentários de linha (// ...)

Expressões Regulares

Token	Regex	Exemplos válidos	Exemplos inválidos
Identificador	`[A-Z, a-z][A-Za-z0-9]*`	`x, var1, cont`	`1abc, @nome, #teste`
Palavra-chave	`(mesma regra de identificador, mas comparado a tabela keywords)`	`if, else, while, int, float, string, return`	`iff, inta, whiles`
Número int	`[0-9]`	`0, 42, 123`	`12a, .5, 10.`
Número float	`[0-9].[0-9](apenas se houver dígito após o ponto)`	`3.14, 0.5, 10.0`	`3., .5, 10.`
String	`".*"`	`"Olá", "123", "texto simples"`	`"não fecha, "abc\`
Comentário	`//[^\n]*`	`//comentário, //123abc`	`(não existe inválido, só termina na linha)`
Operadores duplos	`== != <= >= && ||`	`==, !=, >=, ||`	`=<, >>`
Operadores simples	`[+\-*/=<>%]`	`+, -, /, %, <, >`	`?, >>`
Pontuação	`[();,{}\[\]]`	`;, (, ), {, }`	`:, . (a não ser parte de número)`
Desconhecido	`Qualquer caractere fora das regras acima`	`@, ?, ^`	`(qualquer caractere não tratado)`

UNISINOS

Estrutura do Analisador

1. **Classe Token**
   - Guarda os atributos de cada token:
     - Tipo (Keyword, Identifier, Number, etc.)
     - Lexema (o texto do token)
     - Linha e Coluna (posição no código)
   - Função principal: representar o resultado da análise léxica.

2. **Classe Lexer**
   - Faz a varredura do código-fonte
   - Usa funções auxiliares:
     - peek() → olha o próximo caractere sem consumir.
     - get() → lê e avança para o próximo caractere.
   - Responsável por aplicar as regras (regex e condições) e gerar os tokens.

3. **Fluxo geral**
   - Entrada: arquivo-fonte (.src)
   - Lexer analisa o texto → produz tokens (Token)
   - Saída: lista de tokens exibida no terminal

```
enum class TokenType {
    IDENTIFIER, NUMBER, STRING, KEYWORD,
    OPERATOR, PUNCTUATION, END_OF_FILE, UNKNOWN, COMMENT
};
```

```
struct Token {
    TokenType tipo;
    string texto;
    int linha;
    int coluna;
```

```
char peek(size_t k=0) const {
    if (i + k >= src.size()) return '\0';
    return src.at(i + k);
}
```

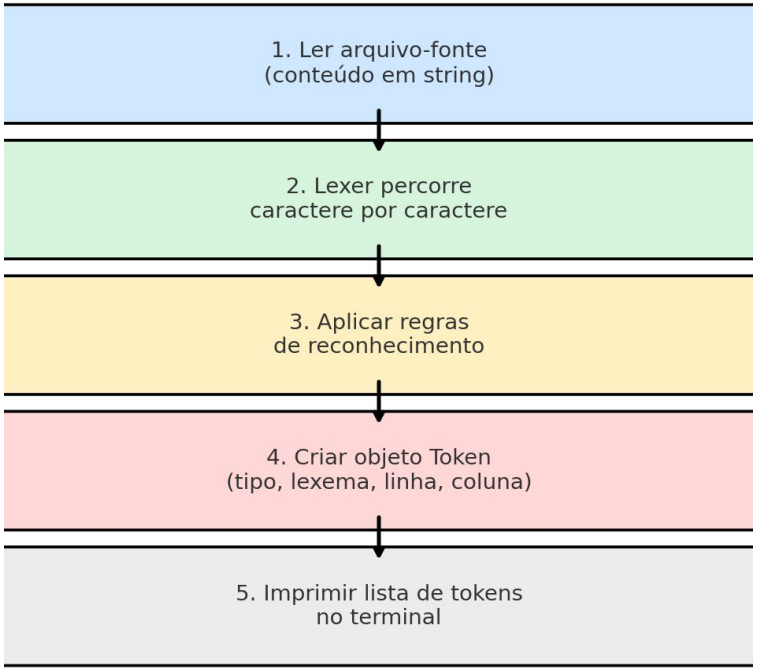
```
char get() {
    if (i >= src.size()) return '\0';
    char c = src[i++];
    if (c == '\n') {
        line++; col = 1;
    } else col++;

    return c;
}
```

Funcionamento do Código

Resumo do fluxo:

1. Lê o arquivo-fonte e converte em string.
2. Lexer percorre caractere por caractere.
3. Aplica regras → identifica tipo de token.
4. Cria objetos Token com tipo, lexema, linha e coluna.
5. Imprime todos os tokens no terminal.



Exemplo de entrada de Código Fonte

```
int x = 42;
string s = "Olá";
if (x > 10 && s != "") {
    x = x + 1;
}
```

- Cada token é mostrado no formato: **TIPO → "lexema" [linha,coluna].**
- O analisador reconhece comentários, operadores compostos (&&, !=), strings, números, pontuação e palavras-chave.

Saída do Analisador

COMMENTARIO -> "// teste simples" [1,1]
KEYWORD -> "int" [2,1]
IDENTIFICADOR -> "x" [2,5]
OPERADOR -> "=" [2,7]
NUMERO -> "42" [2,9]
PONTUACAO -> ";" [2,11]
KEYWORD -> "string" [3,1]
IDENTIFICADOR -> "s" [3,8]
OPERADOR -> "=" [3,10]
STRING -> ""Olá"" [3,12]
PONTUACAO -> ";" [3,18]
KEYWORD -> "if" [4,1]
PONTUACAO -> "(" [4,4]
IDENTIFICADOR -> "x" [4,5]
OPERADOR -> ">" [4,7]
NUMERO -> "10" [4,9]
OPERADOR -> "&&" [4,12]
IDENTIFICADOR -> "s" [4,15]
OPERADOR -> "!=" [4,17]
STRING -> """" [4,20]
PONTUACAO -> ")" [4,22]
PONTUACAO -> "{" [4,24]
IDENTIFICADOR -> "x" [5,5]
OPERADOR -> "=" [5,7]
IDENTIFICADOR -> "x" [5,9]
OPERADOR -> "+" [5,11]
NUMERO -> "1" [5,13]
PONTUACAO -> ";" [5,14]
PONTUACAO -> "}" [6,1]
FIM DE ARQUIVO -> "<EOF>" [6,2]

Tratamento de Erros

- O analisador identifica erros comuns e os classifica como UNKNOWN.
- Principal caso tratado: string não finalizada.
- Exemplo:
  - Entrada: `"texto sem fechar`
  - Saída:

```
UNKNOWN -> "\"texto sem fechar (String nunca foi fechada)" [linha,coluna]
```

  - Entrada: caractere inválido
  - Saída:

```
NUMBER -> "10" [1,9]
UNKNOWN -> "@" [1,12]
NUMBER -> "5" [1,14]
```

UNISINOS

Conclusão

- Implementação do analisador léxico em C++ foi concluída.
- Reconhece todos os tokens definidos e trata erros de strings.
- Base pronta para integração com análise sintática e semântica.